On the Impacts of Mathematical Realization over Practical Security of Leakage Resilient Cryptographic Scheme

Guangjun Fan[1], Yongbin Zhou[2], François-Xavier Standaert[3], Dengguo Feng[1]

[1] Trusted Computing and Information Assurance Laboratory,Institute of Software,Chinese Academy of Sciences,Beijing,China
guangjunfan@163.com , feng@tca.iscas.ac.cn
[2] State Key Laboratory of Information Security,Institute of Information Engineering,Chinese Academy of Sciences,Beijing,China
zhouyongbin@iie.ac.cn
[3] UCL Crypto Group,Université catholique de Louvain,Belgium
fstandae@uclouvain.be

Abstract. In real world, in order to transform an abstract and generic cryptographic scheme into actual physical implementation, one usually undergoes two processes: mathematical realization at algorithmic level and physical realization at implementation level. In the former process, the abstract and generic cryptographic scheme is transformed into an exact and specific mathematical scheme, while in the latter process the output of mathematical realization is being transformed into a physical cryptographic module runs as a piece of software, or hardware, or combination of both. In black-box model (i.e. leakage-free setting), a cryptographic scheme can be mathematically realized without affecting its both theoretical security and practical security of mathematical realization as long as the mathematical components meet the required cryptographic properties. However, it is unknown that whether one can mathematically realize a leakage resilient cryptographic scheme in accustomed ways without affecting its practical security of mathematical realization.

Our results give a negative answer to this important question by introducing attacks against several kinds of mathematical realization of a practical leakage resilient cryptographic scheme. Our results show the big gap between theoretical security of leakage resilient cryptographic scheme and practical security of mathematical realization of the same scheme. Therefore, on one hand, we suggest that all (practical) leakage resilient cryptographic schemes should at least come with a kind of mathematical realization whose practical security can be guaranteed. On the other hand, our results inspire cryptographers to design advanced leakage resilient cryptographic schemes whose practical security of mathematical realization is independent of details of the mathematical realization.

Keywords: Physical Attacks, Leakage Resilient Cryptography, Mathematical Realization, Physical realization.

1 Introduction

Countermeasures for protecting against physical attacks (such as the most studied side-channel attacks) are taken on three levels: the software level, the hardware level, and combination of the above two levels. However, these countermeasures have many issues [2,3]. In order to solve these pressing issues, S. Dziembowski et al. firstly proposed one general and theoretical methodology called Leakage Resilient Cryptography (LRC) [2,3].

In real world, in order to transform an abstract and generic cryptographic scheme into actual physical implementation, one usually undergoes two processes: mathematical realization at algorithmic level and physical realization at implementation level. *Mathematical realization* refers to a process in which an abstract and generic cryptographic scheme is transformed into an exact and specific mathematical scheme (After this process, we say the cryptographic scheme is mathematically realized.). This means that all the cryptographic components utilized by the cryptographic scheme are instantiated with exact and specific mathematical components. For example, it is well known that a public key encryption scheme can be constructed from an arbitrary family of one-way trapdoor permutations. The implementor chooses a specific family of one-way trapdoor permutations (such as RSA trapdoor permutations or Rabin trapdoor permutations) to mathematically realize the public key encryption scheme in this process. Another example is that the implementor chooses AES-128 or 3DES to mathematically realize a cryptographic scheme which uses block ciphers as a building block. *Physical realization* refers to a subsequent process in which any exact and specific mathematical scheme (the output of mathematical realization) is transformed into a physical cryptographic module that runs as a piece of software, or hardware, or combination of both.

Both for cryptographic schemes in black-box model and leakage resilient cryptographic schemes, their cryptographic security proofs generally work independent of both mathematical realization and physical realization. However, both for the above two kinds of cryptographic schemes, it has been turned out that physical security of physical realization highly depends on details of the physical realization. For example, the physical cryptanalysis results of the leakage resilient cryptographic scheme in paper [24] do not contradict its security proof and show that the tolerance leakage rate that is assumed in the theoretical security depends on details of the physical realization.

Now, let's concentrate on mathematical realization. We call a kind of mathematical realization of a cryptographic scheme in black-box model is practically secure if the adversary can not break security of the scheme (e.g. IND-CPA of a public key encryption scheme) even if the adversary knows all details of the mathematical realization. Similarly, we call a kind of mathematical realization of a leakage resilient cryptographic scheme is practically secure as long as (1) the adversary knows all details of the mathematical realization and can not get more leakage bits about all the secret states than those assumed in the theoretical security even if all internal states of all the mathematical components can be leaked to him but the number of leakage bits in each invocation is less

than the leakage parameter. For example, the number of leakage bits about a secret state x is bounded by leakage parameter λ (i.e. $|f(x)| \leq \lambda$, where f is the leakage function.). The adversary can not obtain more leakage bits about x than λ bits even if he can obtain at most λ bits about all internal states of all the mathematical components. (2) the adversary can not break security of the scheme.

Motivation In recent years, in the field of LRC, many leakage models have been proposed. These leakage models are mainly based on two different leakage assumptions.

"*Only Computation Leaks Information*" There are some leakage models that follow the "*Only Computation Leaks Information*" axiom, which states that memory contents that are not accessed during computation, do not leak [4]. Leakage resilient stream cipher [3], practical leakage resilient PRNG [26] and leakage resilient ElGamal encryption scheme[1] follow this axiom are given out.

"*Memory Attack*" Akavia et al. [6] introduced the leakage model of "security against memory attacks" where one requires that the scheme remains secure even if a function $f(sk)$ of the secret key sk is leaked *once*, where the only restriction on $f(\cdot)$ one makes is that the output length of $f(\cdot)$ is bounded. A lot of leakage resilient cryptographic schemes are built in this model [5,7,8]. *Continuous Memory Attack* [15,16,17] extends *Memory Attack*.

There are some other leakage models, such as *Bounded Retrieval Model* [9,10,11,12,13,14] and *Auxiliary Input Model* [27,28].

Theoretical security of leakage resilient cryptographic schemes in these models ignores details of mathematical realization. In other words, theoretical security only holds for a kind of mathematical realization which *rigorously* fits the claimed leakage model.

In black-box model (i.e. leakage-free setting), a cryptographic scheme can be mathematically realized without affecting both its theoretical security and practical security of mathematical realization as long as all the mathematical components meet the required cryptographic properties. However, it is ***unknown*** that whether one can mathematically realize a leakage resilient cryptographic scheme in accustomed ways without affecting its practical security of mathematical realization. No previous work has concerned on this question.

In this paper, in order to answer this important question, we will take the leakage resilient ElGamal encryption scheme instantiated over arbitrary groups of prime order p (where $p-1$ is not smooth) in the paper [1][1] (i.e. scheme EG^*) as an example. The scheme EG^* is constructed in a leakage model that follow the "*Only Computation Leaks Information*" axiom which is regarded as the most

[1] The same leakage resilient ElGamal scheme instantiated over bilinear groups of prime order p (where $p-1$ is not smooth) is leakage resilient in the generic-group model (i.e. scheme BEG^*). However, it is very hard to implement the generic-group model in practice. This drawback of the generic-group model goes against our recommendation to at least provide mathematical realization for a cryptographic scheme. Therefore, in this paper, we consider the scheme EG^* which can be implemented in practice easily.

representative axiom according to side-channel attacks. For simplicity, we only concentrate on how to mathematically realize the process of generating random numbers for scheme EG^* and ignore other abstract cryptographic components which also need to be mathematically realized. We will introduce five different kinds of mathematical realization of scheme EG^*. In each mathematical realization, we use generic Random Number Generator (RNG) or Pseudorandom Number Generator (PRNG) to mathematically realize the process of generating random numbers (Note that, PRNG is used widely for generating random numbers in practice.). We want to see if the five kinds of mathematical realization are practically secure by attacks against them.

Note that, in this paper, we only consider mathematical realization, not physical realization. That is to say, our work is regardless of any specific physical attack against physical realization.

Our Contributions Main contributions of this paper are three-folds as follows. First, our results give a negative answer to the important question that whether one can mathematically realize a leakage resilient cryptographic scheme in accustomed ways without affecting its practical security of mathematical realization by some counterexamples. For example, our research shows that if one directly uses some PRNGs (even if international standard PRNGs or leakage resilient PRNG) to mathematically realize the process of generating random numbers for some leakage resilient cryptographic schemes, the mathematical realization will not be practically secure. Furthermore, we have analyzed drawbacks of mathematical structures of these PRNGs which cause the mathematical realization with these PRNGs becomes practically insecure. Our results show that there exists a big gap between theoretical security of leakage resilient cryptographic scheme and practical security of mathematical realization of the same scheme.

Second, we give out a suggested way to generate random numbers for scheme EG^* using exponentially hard PRNG and Extractor, which is a better choice toward practical security of mathematical realization of scheme EG^*. According to different leakage scenarios, the generated bit sequence of the suggested way can not be statistically distinguished from a true random bit sequence or has high min-entropy (HILL pseudoentropy) for any probabilistic polynomial-time adversary. Moreover, the number of random bits needed by the suggested way is reduced. We believe that the suggested way can also be exploited for other leakage resilient cryptographic schemes which have similar drawbacks of practical security of mathematical realization.

Third, for any leakage resilient cryptographic scheme, tolerance leakage rate reflects its expected security. Therefore, (accurate or rough) estimation of tolerance leakage rate of any leakage resilient cryptographic scheme does make very good sense. For each kind of mathematical realization of scheme EG^*, this paper specifies an upper bound of *practical* tolerance leakage rate that scheme EG^* can tolerate *by-product*. These upper bounds are the best known so far, even thought it might not be the tightest one.

Organization of This Paper The rest of this paper is organized as follows. In Section 2, we first present some basic symbols, notations, and concepts. Then, we briefly review the scheme EG^*. Section 3 introduces five kinds of mathematical realization of scheme EG^* and their practical security. In section 4, we will suggest a new way to generate random numbers for scheme EG^*. Section 5 concludes the whole paper.

2 Preliminaries

In this section, we first present some symbols, notations and concepts used throughout this paper. Then, we briefly review the scheme EG^*.

2.1 Symbols, Notations, and Concepts

If S is a binary bit string, we denote the most significant a bits of S by $\mathsf{S}^{[a]}$ and denote the least significant b bits of S by $\mathsf{S}_{[b]}$. We denote the length of S by $|\mathsf{S}|$ and assume that the binary bit string representation of all elements in $\mathbb{Z}_p$ has the same length. We denote the least significant bit of S is the 1^{st} bit of S and the most significant bit of S is the $|\mathsf{S}|^{th}$ bit of S. We use the symbol $[\mathsf{S}]_{(i)}$ to denote the i^{th} bit of S.

We use U_n to denote the random variable with distribution uniform over $\{0,1\}^n$. With $X \sim Y$ we denote that the two random variables X and Y have the same distribution. Define the statistical distance between two random variables X and Y over a common domain ν as $\delta(X;Y) = \frac{1}{2}\sum_{s\in\nu} |\mathsf{Pr}[X=s] - \mathsf{Pr}[Y=s]|$. We say two random variables X and Y are statistically indistinguishable if $\delta(X;Y) < \epsilon$, where ϵ is negligible in the security parameter. With $\delta^{\mathsf{D}}(X;Y)$ denote the advantage of a circuit D in distinguishing the random variables X and Y, i.e. $\delta^{\mathsf{D}}(X;Y) \stackrel{\mathsf{def}}{=} |\mathsf{E}[\mathsf{D}(X)] - \mathsf{E}[\mathsf{D}(Y)]|$. Let $\mathcal{D}_s$ denote the class of all probabilistic circuits of size s with binary output $\{0,1\}$. With $\delta_s(X;Y)$ we denote $max_{\mathsf{D}}\{\delta^{\mathsf{D}}(X;Y)\}$ where the maximum is over $\mathsf{D} \in \mathcal{D}_s$. We say that a random variable Y has min-entropy k, denoted by $\mathsf{H}_\infty(Y) = k$, if $max_y\{\mathsf{Pr}[Y=y]\} = 2^{-k}$. We define HILL pseudoentropy as follows.

Definition 1. *We say that X has HILL pseudoentropy k* (denoted by $\mathsf{H}^{\mathsf{HILL}}_{\epsilon,s} \geq k$), *if there exists a distribution Y where $\mathsf{H}_\infty(Y) \geq k$ and $\delta_s(X,Y) \leq \epsilon$.*

In this paper, we also exploit the following cryptographic tools.

Definition 2. *A function $\mathsf{ext} : \{0,1\}^{n_{\mathsf{ext}}} \times \{0,1\}^{r_{\mathsf{ext}}} \to \{0,1\}^{m_{\mathsf{ext}}}$ is an (ϵ,k) extractor if for any X with $\mathsf{H}_\infty(X) \geq k$ and random input $S \sim U_{n_{\mathsf{ext}}}$, it holds that $\delta((\mathsf{ext}(S,X),S);(U_{m_{\mathsf{ext}}},S)) \leq \epsilon$.*

Definition 3. *A function $\mathsf{prng} : \{0,1\}^{n_{\mathsf{prng}}} \to \{0,1\}^{m_{\mathsf{prng}}}$ is a $(\epsilon,s)-$secure pseudorandom number generator if $\delta_s(\mathsf{prng}(U_{n_{\mathsf{prng}}}); U_{m_{\mathsf{prng}}}) < \epsilon$.*

We say that a PRNG $\mathsf{prng} : \{0,1\}^{n_{\mathsf{prng}}} \rightarrow \{0,1\}^{m_{\mathsf{prng}}}$ is exponentially hard if it is a $(2^{(c-1)n_{\mathsf{prng}}}, 2^{cn_{\mathsf{prng}}})-$secure pseudorandom number generator, where $c \in (0,1)$. The paper [3] exploited exponentially hard PRNG to construct leakage resilient stream cipher.

2.2 Brief Description of scheme EG^*

We describe the scheme $\mathsf{EG}^* = (\mathsf{KG}^*_{\mathsf{EG}}, \mathsf{Enc}^*_{\mathsf{EG}}, \mathsf{Dec1}^*_{\mathsf{EG}}, \mathsf{Dec2}^*_{\mathsf{EG}})$ and the corresponding security definition in the same way as that in the paper [1]. Let the security parameter of scheme EG^* is κ. Let Gen denote a probabilistic algorithm that outputs a cyclic group $\mathbb{G}$ of order p, where p is a strong prime and $|p| = \kappa$. The scheme EG^* is described as a Key Encapsulation Mechanism (KEM) and is shown as follows:

$\mathsf{KG}^*_{\mathsf{EG}}(\kappa)$: Compute $(\mathbb{G}, p) \xleftarrow{*} \mathsf{Gen}(n)$, $g \xleftarrow{*} \mathbb{G}$, $x \xleftarrow{*} \mathbb{Z}_p$, $h = g^x$. Choose random $\sigma_0 \xleftarrow{*} \mathbb{Z}^*_p$ and set $\sigma'_0 = x\sigma_0^{-1} \bmod (p)$. The public key is $pk = (\mathbb{G}, p, h)$ and the secret key is $sk = x$. Two secret states are σ_0 and σ'_0.

$\mathsf{Enc}^*_{\mathsf{EG}}(pk)$: Choose random $r \xleftarrow{*} \mathbb{Z}_p$. Let $C \leftarrow g^r \in \mathbb{G}$ and $K \leftarrow h^r \in \mathbb{G}$. The ciphertext is C and the symmetric key is K.

$\mathsf{Dec1}^*_{\mathsf{EG}}(\sigma_{i-1}, C)$: Choose random $r_i \xleftarrow{*} \mathbb{Z}^*_p$, $\sigma_i = \sigma_{i-1} r_i \bmod (p)$, $K' = C^{\sigma_i}$, return(r_i, K').

$\mathsf{Dec2}^*_{\mathsf{EG}}(\sigma'_{i-1}, (r_i, K'))$: Set $\sigma'_i = \sigma'_{i-1} r_i^{-1} \bmod (p)$, and $K = K'^{\sigma'_i}$. The symmetric key is K and the updated states are σ_i and σ'_i.

The security definition of scheme EG^* is CCLA1 which was introduced in the paper [1]. In CCLA1, the two leakage functions f_i and g_i are efficient computable functions chosen by the adversary and get as inputs only the secret states that are actually accessed during computation. The ranges of f_i and g_i are bounded by leakage parameter λ. For scheme EG^*, the leakage functions f_i and g_i are as follows:

$$\Lambda_i \leftarrow f_i(\sigma_{i-1}, r_i),\ \Lambda'_i \leftarrow g_i(\sigma'_{i-1}, (r_i, K'), r_i^{-1}), \text{ and } |\Lambda_i| \leq \lambda, |\Lambda'_i| \leq \lambda\ .$$

Although the authors of the paper [1] didn't prove theoretical security of scheme EG^* and only presented the following conjecture, the crucial technique of scheme EG^* (i.e. multiplicative secret sharing) is used widely in the context of LRC [30,31,32] and scheme EG^* is more practical than other leakage resilient cryptographic schemes. Therefore, we take scheme EG^* as an example.

Conjecture 1 *The scheme EG^* is CCLA1 secure if $p-1$ has a large prime factor (say, $p-1 = 2q$ for a prime q).*

Therefore, authors of the paper [1] conjectured that roughly λ equals to $0.25|p|$ bits in [23]. Thus the number of total tolerance leakage bits in one decapsulation query equals to $2\lambda = 0.5|p|$ bits.

We use $\lambda/|p|$ to denote tolerance leakage rate of scheme EG^* and let $\rho = \frac{|f_i|+|g_i|}{|p|}$. Any implementation of scheme EG^* will be secure against every side-channel attack that fits the leakage model, i.e. as long as the amount of information that is leaked during each invocation is sufficiently bounded, and moreover

the cryptographic device adheres the "*Only Computation Leaks Information*" axiom. However, the authors said nothing about how to generate random numbers r_i for scheme EG^*. Therefore, actual implementors may use True Random Number Generator (TRNG) or PRNG to mathematically realize this process.

3 Five Kinds of Mathematical Realization of Scheme EG^* With Generic RNG or Specific PRNG and Their Practical Security

It is well known that one can use TRNG or PRNG to generate random numbers. Although there exist some TRNGs, PRNG is used more widely than TRNG in practice. The reasons of this fact are in the following. First, TRNG requires a naturally occurring source of randomness. Designing a hardware device or software program to exploit this randomness and produce a bit sequence that is free of biases and correlations is a difficult task. Second, for most cryptographic applications, the random number generator must not be subject to observation or manipulation by an adversary. However, TRNG is subject to influence by external factors, and also to malfunction. Third, the generation of true random number is an inefficient procedure in most practical environments. Finally, it may be impractical to securely store and transmit a large number of true random bits if these are required in applications. Therefore, we mainly consider the case of utilizing PRNG to mathematically realize the process of generating random numbers in this paper.

In this section, we will introduce five kinds of mathematical realization of scheme EG^*. In each mathematical realization, the process of generating random numbers is mathematically realized by generic RNG or PRNG (leakage resilient PRNG). We want to see whether the five kinds of mathematical realization are practically secure, by presenting specific attacks against them. The goal of all our attacks is to recover the secret key x. To achieve this goal, our attacks need to obtain all the bits of the random number r_i for each invocation of the decapsulation query of scheme EG^*. The adversary can recover all the bits of σ_i and σ'_i $(i = 0, 1, \ldots)$ and obtain a candidate value x' of the real secret key x. The adversary can verify the correctness of x' by a correct pair (C, K).

In the first kind of mathematical realization, we assume the the process of generating random numbers r_i is mathematically realized by generic RNG and the adversary does not know the internal mathematical structure of the generic RNG. The attack against this kind of mathematical realization (denoted by ATTACK I) can also be viewed as an attack against theoretical security of scheme EG^*. ATTACK I satisfies the leakage model of scheme EG^* defined in the paper [1] except that it requires a high leakage rate. Therefore, ATTACK I poses no threat on the theoretical security of scheme EG^*.

In the rest four kinds of mathematical realization, we assume the process of generating random numbers r_i is mathematically realized by a specific PRNG (leakage resilient PRNG). For convenience, the attacks against the four kinds of mathematical realization are denoted by ATTACK II. ATTACK II have the

same basic principle as ATTACK I. However, it is amazing that the results of ATTACK II show that practical tolerance leakage rate of mathematical realization of scheme EG^* will decrease dramatically when some specific PRNGs are used to mathematically realize the process of generating random numbers r_i.

In the following, we will introduce the five kinds of mathematical realization and attacks against them. Finally, we will show some discussions and results of the attacks. For both ATTACK I and ATTACK II, we assume the random number r_i is generated by Algorithm 1.

Algorithm 1 The Algorithm of Generating Random Numbers r_i

Input: no input
Output: a random number r_i
Step 1 Invoke generic RNG or PRNG to generate a new random number t and $|t| = |r_i|$.
Step 2 If $t = 0$ then return to Step 1 else go to Step 3.
Step 3 If $t < p$ then go to Step 4 else go to Step 5.
Step 4 Let $r_i := t$ and return r_i.
Step 5 Let $r_i := t \bmod p$ and return r_i.

3.1 Mathematical Realization Using Generic RNG

If the process of generating random numbers r_i is mathematically realized by generic RNG, we can attack this kind of mathematical realization as follows (ATTACK I):

In the 1^{st} invocation of decapsulation query of scheme EG^*, the adversary chooses the leakage functions as follows:

$$f_1(\sigma_0, r_1) = \langle [\sigma_0]_{(1)}, r_1^{[|p|/2]} \rangle,\ g_1(\sigma_0', (r_1, K'), r_1^{-1}) = \langle [\sigma_0']_{(1)}, r_{1[|p|/2]} \rangle.$$

Now, the adversary knows r_1 ($r_1 := r_1^{[|p|/2]} \parallel r_{1[|p|/2]}$), r_1^{-1} (Note that, the prime number p is public.), $\sigma_{0[1]}$, and $\sigma'_{0[1]}$. In the 2^{nd} invocation of decapsulation query, the adversary chooses the leakage functions as follows:

$$f_2(\sigma_1, r_2) = \langle [\sigma_1 r_1^{-1} \bmod p]_{(2)}, r_2^{[|p|/2]} \rangle = \langle [\sigma_0]_{(2)}, r_2^{[|p|/2]} \rangle,$$

$$g_2(\sigma_1', (r_2, K'), r_2^{-1}) = \langle [\sigma_1' r_1 \bmod p]_{(2)}, r_{2[|p|/2]} \rangle = \langle [\sigma_0']_{(2)}, r_{2[|p|/2]} \rangle.$$

After the 2^{nd} invocation of decapsulation query, the adversary knows r_1, r_1^{-1}, r_2, r_2^{-1}, $\sigma_{0[2]}$, and $\sigma'_{0[2]}$. Let $R_{\{a,b\}} := \prod_{s=a}^{b} r_s \bmod p$ and $R_{\{a,b\}}^{-1} := \prod_{s=a}^{b} r_s^{-1} \bmod p$. In the i^{th} $(i = 2, \ldots, |p| - 1)$ invocation of decapsulation query, the adversary chooses the leakage functions as follows:

$$f_i(\sigma_{i-1}, r_i) = \langle [\sigma_{i-1} R_{\{1,i-1\}}^{-1} \bmod p]_{(i)}, r_i^{[|p|/2]} \rangle = \langle [\sigma_0]_{(i)}, r_i^{[|p|/2]} \rangle,$$

$$g_i(\sigma'_{i-1}, (r_i, K'), r_i^{-1}) = \langle [\sigma'_{i-1} R_{\{1,i-1\}} \bmod p]_{(i)}, r_{i[|p|/2]} \rangle = \langle [\sigma'_0]_{(i)}, r_{i[|p|/2]} \rangle.$$

In the $|p|^{th}$ invocation of decapsulation query, the adversary chooses the leakage functions as follows:

$$f_{|p|}(\sigma_{|p|-1}, r_{|p|}) = \langle [\sigma_{|p|-1} R^{-1}_{\{1,|p|-1\}} \bmod p]_{(|p|)} \rangle = \langle [\sigma_0]_{(|p|)} \rangle,$$

$$g_{|p|}(\sigma'_{|p|-1}, (r_{|p|}, K'), r_{|p|}^{-1}) = \langle [\sigma'_{|p|-1} R_{\{1,|p|-1\}} \bmod p]_{(|p|)} \rangle = \langle [\sigma'_0]_{(|p|)} \rangle.$$

In this way, after invoking the decapsulation query $|p|$ times, the adversary knows all the bits of σ_0 and σ'_0. Then, he can recover a candidate value $x' = \sigma_0 \sigma'_0 \bmod p$ of the real secret key x. Then, the adversary can verify the correctness of x' by a correct pair (C, K). The attack process is shown in Figure 4 in Appendix B.

To successfully execute ATTACK I, the leakage parameter λ should achieve $0.5|p| + 1$ bits, which is larger than $0.25|p|$. Therefore, ATTACK I poses no threat on the theoretical security of scheme EG^*. Note that, ATTACK I can also be executed after the i^{th} decapsulation query similarly. After the adversary obtaining σ_i and σ'_i, he can recover a candidate value $x' = \sigma_i \sigma'_i \bmod p$ of the real secret key x.

3.2 Mathematical Realization Using PRNG

Now, we assume that the process of generating random numbers r_i is mathematically realized by specific PRNG (leakage resilient PRNG). According to Kerckhoffs' principle, the adversary knows concrete mathematical structure of the specific PRNG used by the mathematical realization. When the PRNG is invoked to generate a random number r_i in the decapsulation query, all internal secret states of the PRNG can be leaked to the adversary due to the *"Only Computation Leaks Information"* axiom.

We know that if one obtains all bits of all the secret states (such as the seed) of any PRNG, he can totally recover the output of the PRNG trivially. Therefore, for ATTACK II, we ***don't*** allow the adversary to obtain all bits of all the secret states of the PRNG from leakages ***directly*** in one invocation. Specifically speaking, what the adversary can obtain from leakage functions in one invocation of decapsulation query of scheme EG^* includes only *part* of bits about the secret states of the PRNG and *part* of bits about the output of the PRNG. But the amount of leakages is bounded by λ (the leakage parameter) bits. The central idea of ATTACK II is that the adversary tries to recover all bits of the seed of the PRNG (not from direct leakages) using the specific mathematical structure of the PRNG with at most λ bits from leakages. In this manner, we show the impacts of mathematical realization over practical security of the leakage resilient scheme EG^*.

If the practical tolerance leakage rate of a kind of mathematical realization of scheme EG^* can not achieve 0.25 (i.e. the theoretical tolerance leakage rate $\frac{\lambda}{|p|} = 0.25$), we say the mathematical realization is not practically secure (as

the definition about practical security of mathematical realization of a leakage resilient cryptographic scheme in Section 1).

We surprisingly find that practical tolerance leakage rate of scheme EG^* will reduce to a value less than 0.25 when four specific PRNGs are used to mathematically realize the process of generating random numbers r_i. Therefore, mathematical realization of scheme EG^* is not practically secure when the four specific PRNGs are used. The four specific PRNGs are ANSI X9.17 PRNG, ANSI X9.31 PRNG, FIPS 186 PRNG for DSA per-message secrets, and a leakage resilient PRNG in [26] instantiated with AES-128. We also assume that the seed of the specific PRNG is refreshed in each invocation of the decapsulation query.

3.2.1 Case 1: ANSI X9.17 PRNG and ANSI X9.31 PRNG

The ANSI X9.17 PRNG [18] has been used as a general purpose PRNG in many applications. Let E_{key} (resp. D_{key}) denotes DES E-D-E two-key triple-encryption (resp. decryption) under a key key, which is generated somehow at initialization time and must be reserved exclusively used only for this generator. The key is a internal secret state of the PRNG which is never changed for every invocation of the PRNG. ANSI X9.17 PRNG is shown in Algorithm 2.

Algorithm 2 ANSI X9.17 PRNG

Input: a random (and secret) 64-bit seed $seed[1]$, integer v, and E_{key}.
Output: v pseudorandom 64-bit strings (denoted by $output[1], \ldots, output[v]$).
Step 1 For l from 1 to v do the following:
1.1 Compute $I_l = E_{key}(input[l])$, where $input[l]$ is a 64-bit representation of the system date/time.
1.2 $output[l] = E_{key}(I_l \bigoplus seed[l])$
1.3 $seed[l+1] = E_{key}(I_l \bigoplus output[l])$
Step 2 Return $(output[1], output[2], \ldots, output[v])$

Suppose that each $input[l]$ $(l = 1, 2, \ldots, v)$ has 10 bits that the adversary does not know (We assume these 10 bits are the least significant 10 bits of each $input[l]$.). This is a reasonable assumption for many systems[1] [22]. Before doing our attack, due to the fact that key is never changed for every invocation of the PRNG (*stateless*), the adversary can completely obtain key from leakage function f_i by invoking the decapsulation query repeatedly. In each invocation, the leakage function f_i leaks only part of bits about key (not all the bits of key). After knowing key completely, the adversary continually invoke the decapsulation query for $|p|$ times. Let

$$state_{i+u} := \{output[1]_{i+u}, input[1]_{i+u[10]}, \ldots, input[v]_{i+u[10]}\}$$

[1] For example, consider a millisecond timer, and an adversary who knows the nearest second when an output was generated.

and the leakage functions are defined as follows:

For $u = 1$,

$$f_{i+u}(\sigma_{i+u-1}, r_{i+u}) = \langle [\sigma_i]_{(1)}, state_{i+u} \rangle,$$

$$g_{i+u}(\sigma'_{i+u-1}, (r_{i+u}, K'), r^{-1}_{i+u}) = \langle [\sigma'_i]_{(1)} \rangle.$$

For $u = 2, \ldots, |p| - 1$,

$$f_{i+u}(\sigma_{i+u-1}, r_{i+u}) = \langle [\sigma_{i+u-1} R^{-1}_{\{i+1,i+u-1\}} \bmod p]_{(u)}, state_{i+u} \rangle,$$

$$g_{i+u}(\sigma'_{i+u-1}, (r_{i+u}, K'), r^{-1}_{i+u}) = \langle [\sigma'_{i+u-1} R_{\{i+1,i+u-1\}} \bmod p]_{(u)} \rangle.$$

For $u = |p|$,

$$f_{i+u}(\sigma_{i+u-1}, r_{i+u}) = \langle [\sigma_{i+u-1} R^{-1}_{\{i+1,i+u-1\}} \bmod p]_{(|p|)} \rangle = \langle [\sigma_i]_{(|p|)} \rangle$$

$$g_{i+u}(\sigma'_{i+u-1}, (r_{i+u}, K'), r^{-1}_{i+u}) = \langle [\sigma'_{i+u-1} R_{\{i+1,i+u-1\}} \bmod p]_{(|p|)} \rangle = \langle [\sigma'_i]_{(|p|)} \rangle.$$

The adversary obtains

$$\{output[1]_{i+u}, input[1]_{i+u}, \ldots, input[v]_{i+u}\}, \ (u = 1, \ldots, |p| - 1)$$

and he can further compute

$$seed[1]_{i+u} := D_{key}(output[1]_{i+u}) \oplus E_{key}(input[1]_{i+u}).$$

Then the adversary can easily get

$$seed[s]_{i+u} := E_{key}(E_{key}(input[s-1]_{i+u}) \oplus output[s-1]_{i+u})$$

as well as

$$output[s]_{i+u} := E_{key}(E_{key}(input[s]_{i+u}) \oplus seed[s]_{i+u}), \ (s = 2, 3, \ldots, v).$$

Thus the adversary obtain all the bits of r_i for every decapsulation query. Figure 1, Figure 2 and Table A.1 in Appendix A show that scheme EG^* is not practically secure any more, if it uses ANSI X9.17 PRNG for strong prime p with size larger than 700 bits. Note that ANSI X9.31-1998 Appendix A 2.4 in [21] introduces PRNGs using 3-key triple DES or AES. In 3-key triple DES case, due to the fact that $input[l]$, $seed[l]$ and $output[l]$ have the same length as that of ANSI X9.17 PRNG, we can obtain the same attack results as those of the attack against ANSI X9.17 PRNG. Our attack is still valid for this PRNG using AES-128 similarly. Therefore, we do not introduce the attack against this PRNG for AES-128 case here.

Analysis Although this PRNG is not secure even in leakage-free setting if the adversary knows the key, what we want to emphasize here is drawbacks of the mathematical structure of this PRNG. The drawbacks make this PRNG become

insecure in leakage setting. The designers of the two PRNGs exploit block ciphers (such as 3DES and AES-128) to mathematically realize an abstract One-Way Permutations (OWP). The PRNG (ANSI X9.17 PRNG or ANSI X9.31 PRNG) itself can compute the output of the OWP because it knows the key of the block cipher. In leakage-free setting, if the adversary does not know the key, he can not recover the input of the block cipher (the seed of the PRNG) and the "One-Way" property holds. However, in leakage setting, the adversary can obtain *key* completely from leakages because it is *stateless*. This means that the One-Way Permutation becomes to a One-Way Trapdoor Permutation and the adversary knows the trapdoor (i.e. the *stateless* key) from leakages. Therefore, the "One-Way" property does not hold.

Due to the drawbacks, we think possible solutions which can make this attack become invalid are as follows: *Solution 1* Using advanced mathematical components to mathematically realize the abstract OWP to guarantee the "One-Way" property in leakage setting. *Solution 2* To make the key *key* become *stateful* may be another solution. This means that the implementor needs to refresh the key and to guarantee the adversary can not obtain the key completely in every invocation of the PRNG.

3.2.2 Case 2: FIPS 186 PRNG for DSA Pre-message Secrets

The Digital Signature Standard (DSS) specification (FIPS 186) [19] also describes a fairly simple PRNG based on SHA or DES, which is used for generating DSA per-message secrets. This PRNG is shown in Algorithm 3.

Algorithm 3 FIPS 186 PRNG for DSA pre-message secrets

Input: an integer v and a 160-bit prime number q.
Output: v pseudorandom numbers $output[1], \ldots, output[v]$ in the interval $[0, q-1]$, which may be used as the per-message secret numbers in the DSA.

Step 1 If the SHA based G function is to be used in step 4.1 then select an integer $160 \leq b \leq 512$. If the DES based G function is to be used in step 4.1 then set $b \leftarrow 160$.
Step 2 Generate a random (and secret) b-bit seed $seed[1]$.
Step 3 Define the 160-bit string $str = efcdab89\ \ 98badcfe\ \ 10325476\ \ c3d2e1f0\ \ 67452301$ (in hexadecimal).
Step 4 For l from 1 to v do the following:
4.1 $output[l] \leftarrow G(str, seed[l]) \mod (q)$.
4.2 $seed[l+1] \leftarrow (1 + seed[l] + output[l]) \mod (2^b)$.
Step 5 Return $(output[1], output[2], \ldots, output[v])$.

For general purpose PRNG, *mod q* operation in this PRNG could be omitted. It is necessary only for DSS where all arithmetic is done *mod q*. In this paper, we only consider the DES version of this PRNG, where the G function is based on DES. Therefore, the seed (as well as the output) of this PRNG is 160 bits

long. We show the attack against this PRNG when $|p| = 964$ bits as an example. To generate a 964 bits long random number, one needs to invoke this PRNG 7 times ($v = 7$) iteratively to obtain a 1120 bits long random number and discards $output[v]_{[156]}$. Let

$$state_i = \{output[1]_i^{[40]}, output[2]_i^{[30]}, output[3]_i^{[20]},$$

$$output[4]_i^{[10]}, output[5]_i^{[10]}, output[6]_i^{[6]}, output[7]_i^{[4]}\}.$$

The leakage functions are as follows:

For $i = 1$,

$$f_i(\sigma_{i-1}, r_i) = \langle [\sigma_0]_{(1)}, seed[1]_i^{[120]}, state_i \rangle,$$

$$g_i(\sigma'_{i-1}, (r_i, K'), r_i^{-1}) = \langle [\sigma'_0]_{(1)} \rangle.$$

For $i = 2, \ldots, |p| - 1$,

$$f_i(\sigma_{i-1}, r_i) = \langle [\sigma_{i-1} R^{-1}_{\{1,i-1\}} \bmod p]_{(i)}, seed[1]_i^{[120]}, state_i \rangle,$$

$$g_i(\sigma'_{i-1}, (r_i, K'), r_i^{-1}) = \langle [\sigma'_{i-1} R_{\{1,i-1\}} \bmod p]_{(i)} \rangle.$$

For $i = |p|$,

$$f_i(\sigma_{i-1}, r_i) = \langle [\sigma_{i-1} R^{-1}_{\{1,i-1\}} \bmod p]_{(i)} \rangle = \langle [\sigma_0]_{(|p|)} \rangle,$$

$$g_i(\sigma'_{i-1}, (r_i, K'), r_i^{-1}) = \langle [\sigma'_{i-1} R_{\{1,i-1\}} \bmod p]_{(i)} \rangle = \langle [\sigma'_0]_{(|p|)} \rangle.$$

After the adversary getting the most significant 120 bits of the seed (i.e. $seed[1]_i^{[120]}$ ($i = 1, 2, \ldots, |p|-1$)) from leakages, he could compute all the possible values of the least significant 40 bits of $seed[1]_i$ (i.e. $seed[1]_{i[40]}$) and gets 2^{40} candidate values of $seed[1]_i$. Denote a candidate value by $seed[1]'_i$. For each $seed[1]'_i$, the adversary computes

$$state'_i = \{output[1]_i'^{[40]}, output[2]_i'^{[30]}, output[3]_i'^{[20]},$$

$$output[4]_i'^{[10]}, output[5]_i'^{[10]}, output[6]_i'^{[6]}, output[7]_i'^{[4]}\}.$$

using $seed[1]'_i$ and test the correctness of this candidate value $seed[1]'_i$ using $state_i$ obtained from leakages. For the correct candidate value $seed[1]'_i$ (i.e. $seed[1]_i$), $state'_i$ must equal to $state_i$. This test fails with extremely low probability. For larger size p, the adversary also obtain $seed[1]_i^{[120]}$ from leakages. The number of leakage bits about the output of the PRNG keeps 120 bits unchanged but the distribution of the leakage bits is changed. For every output block $output[l]$ ($l = 1, 2, \ldots, v$), the adversary must obtain some bits about it from leakages (In other words, there does not exist a block of the output of the PRNG ($output[l]$,$l \in \{1, 2, \ldots, v\}$) such that no bit of the block leaks.).

We verified this attack by experiments for different size p. For each size of $|p| \in \{1120, 1280, 1440, 1600\}$ bits, we generated 500 sets of random data and ran the above test with the 500 sets of random data. The success rates of all experiments were 100%. Therefore, we believe our attack is valid. Giving out theoretical success rate of this attack would be interesting but is beyond the scope of this paper. Figure 1, Figure 2 and Table A.2 in Appendix A show that the scheme EG^* is not practically secure any more, if it uses this PRNG for strong prime p with size larger than 964 bits.

3.2.3 Case 3: A Practical Leakage Resilient PRNG

In section 4 of the paper [26], a practical leakage resilient PRNG in the standard model was introduced. This practical leakage resilient PRNG is based on $(\epsilon, s, n/\epsilon)-$secure weak Pseudorandom Function (wPRF) $\mathsf{F}(k, pr) : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^m$. The symbol pr denotes public randomness. The initial state of this PRNG is (pr_0, pr_1, k_0) for public randomness $(pr_0, pr_1) \xleftarrow{*} (\{0,1\}^n)^2$ and the random seed $k_0 \xleftarrow{*} \{0,1\}^\kappa$. The i^{th} round of this PRNG is then computed as $(k_i, o_i) = \mathsf{F}(k_{i-1}, pr_{\rho(i-1)})$, where $\rho(i) = i \bmod 2$, k_i is the secret key for the next round and o_i is the output of this round. In each round, the adversary can obtain not only the output of the PRNG (i.e. o_i), but also leakages from non adaptive leakage function $\mathsf{L}_i(k_{i-1}, pr_{\rho(i-1)})$.

This leakage resilient PRNG can be instantiated with any length-expanding wPRF ($m > \kappa$), which in turn can be mathematically realized from any secure block cipher $\mathsf{BC} : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^\kappa$. That is, if BC is an $(\epsilon, s, 2q)-$secure wPRF, then $\mathsf{F}(k, pr_l \parallel pr_r) = \mathsf{BC}_k(pr_l) \parallel \mathsf{BC}_k(pr_r)$ is an $(\epsilon, s, q)-$secure wPRF.

We assume the secure block cipher BC is mathematically realized using AES-128 (i.e. $\kappa = n = 128$) which is the most studied one by side-channel attacks. Similarly to the attack against FIPS 186 PRNG in Case 2, the adversary can obtain 100 bits of k_0 ($|k_0| = 128$ bits) and 100 bits of the outputs (i.e. o_i $(i = 0, 1, \ldots, v)$) from leakages, then he tries to recover k_0 completely by brute-force search. This attack against the leakage resilient PRNG was verified by experiments and is also valid. Figure 1, Figure 2, and Table A.3 in Appendix A show that scheme EG^* is not practically secure any more, if it uses this leakage resilient PRNG mathematically realized by AES-128 for strong prime p with size larger than 804 bits.

However, what we want to emphasize is not the above attack against the leakage resilient PRNG. What we want to emphasize is that the leakage resilient PRNG is not suitable to mathematically realize the process of generating random numbers for scheme EG^*. Note that, the amount of leakages this PRNG can tolerate (denoted by λ_{prng}) equals to $log(\epsilon^{-1})/6$ and depends on the hardness of the underlying wPRF F [33]. Thus, if F is secure against adversaries of super-polynomial size (i.e. $\epsilon = 2^{\omega(log\kappa)}$), then the amount of leakages λ_{prng} equals to $\omega(log\kappa)$, which is quite small. If $\lambda = 0.25|p| \leq \lambda_{\mathsf{prng}}$, the size of the seed k_0 (i.e. κ) should be much larger than $|p|$. In this case, it is not necessary for the implementor to use this PRNG. The implementor should use $|p|$ bits long random number as r_i directly. What's worse, even if the wPRF F is exponentially hard

(i.e. $\epsilon = 2^{-\Omega(\kappa)}$), this PRNG is also not suitable. In this case , $\epsilon = 2^{-a\kappa}$ ($a \in (0, 1]$) and $\lambda_{\mathsf{prng}} = \frac{log(\epsilon^{-1})}{6} = \frac{a\kappa}{6}$, this PRNG is leakage resilient if and only if $\kappa \geq 1.5 \cdot |p|$.

This leakage resilient PRNG is not suitable to mathematically realize the process of generating random numbers for some other leakage resilient cryptographic schemes due to the similar reasons.

3.3 Discussions and The Results of Our Attacks

Figure 1. shows the minimum ρ required to successfully recover x for different kinds of mathematical realization. Figure 2. shows the minimum $\lambda/|p|$ required to successfully recover x for different kinds of mathematical realization. According to [29], for long-term security, 1024-bit or larger modulus should be used. Therefore, we can see that if scheme EG^* uses the above-mentioned four PRNGs to mathematically realize the process of generating random numbers r_i, the scheme will not be practically secure any more. Although practical tolerance leakage rate can be made arbitrarily small for Case 2 and Case 3 with increase of the size of p, the success of *all* our attacks against these PRNGs is not depend on the size of p but is depend on the mathematical structures of these PRNGs.

The authors of the paper [1] conjectured that theoretical tolerance leakage rate λ of scheme EG^* equals to $0.25|p|$. If the actual value of $\lambda > 0.25|p|$, all our attacks are valid. Otherwise, if the actual value of $\lambda < 0.25 \cdot |p|$, some attacks against these PRNGs *may* become invalid. However, there still exist some kinds of mathematical realization which are not practically secure. For example, for large size p, the attacks against FIPS 168 PRNG and the leakage resilient PRNG are still valid.

The process of generating random numbers r_i for scheme EG^* can also be mathematically realized by TRNGs or other PRNGs. However, it is very difficult to guarantee all the possible mathematical realization using TRNGs or other PRNGs in accustomed ways are practically secure when the actual value of $\lambda < 0.25|p|$. Therefore, an accustomed way to mathematically realize the process of generating random numbers will make the corresponding mathematical realization become practically insecure with high probability.

Note that, the secret key x can also be recovered with an attack method based on Hidden Number Problem [23,25] with lower theoretical tolerance leakage rate (i.e. $\frac{3}{8}|p| + o(|p|)$) than that of ATTACK I. However, practical tolerance leakage rate of this attack method (also equals to $\frac{3}{8}|p| + o(|p|)$) is higher than that of ATTACK II when only the process of generating random numbers is mathematically realized with PRNGs because this attack method requires leakages from σ_i and σ_i' but does not require leakages from r_i.

If we set t to r_i in Algorithm 1 directly, our attacks can work with much less practical tolerance leakage rate. The reason is that the bits about t (the output of the PRNG) needed by the attacks can be leaked from leakage function g_i.

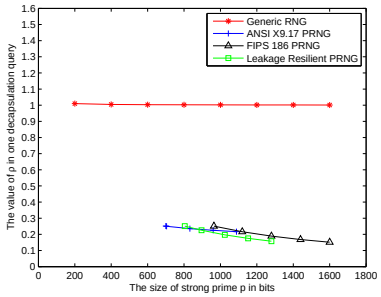

Fig. 1. Minimum ρ required to successfully recover x for different kinds of mathematical realization

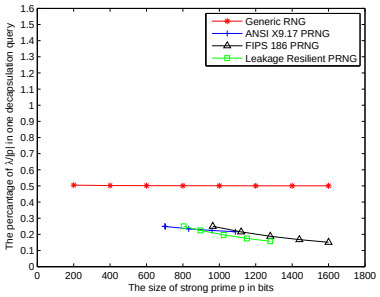

Fig. 2. Minimum $\lambda/|p|$ required to successfully recover x for different kinds of mathematical realization

4 Our Suggested Way to Generate Random Numbers for Scheme EG^*

From Section 3, we can see that the mathematical realization of scheme EG^* is not practically secure when the process of generating random numbers r_i is mathematically realized by some PRNGs. In this section, we suggest to utilize a sophisticated way which is based on *abstract* exponentially hard PRNG and Extractor to to generate random numbers for scheme EG^* under the *"Only Computation Leaks Information"* axiom. For this suggested way, any probabilistic polynomial-time adversary can not distinguish the generated bit sequence from a true random bit sequence when only the internal states of the suggested way are leaked to the adversary but the amount of leakages is less than λ bits. Although we do not give out specific mathematical structures about the abstract exponentially hard PRNG and Extractor in the suggested way, the suggested way is a better choice toward the practical security of mathematical realization of scheme EG^*. The implementor should choose suitable mathematical components with no drawbacks to mathematically realize the exponentially hard PRNG and Extractor respectively.

The security parameter of scheme EG^* is κ which equals to $|p|$. We also assume that λ equals to 0.25κ for introducing the suggested way. If the actual value of λ does not equal to 0.25κ, the suggested way is still valid but with different parameters. The suggested way merely need a random seed with length shorter than κ bits and larger than 0.25κ bits. Therefore, the suggested way reduces the number of random bits needed to generate random numbers for scheme EG^*. If the seed of the suggested way is shorter than 0.25κ bits, the adversary can directly obtain the seed completely from leakages and recover all bits of the generated bit sequence.

Our suggested way is shown in Figure 3 and can be divided into two successive phases. In the first phase, the suggested way chooses a seed uniformly

at random and computes the output of the exponentially hard PRNG with this seed. In the subsequent second phase, the output of the exponentially hard PRNG is input into an Extractor and the output of the Extractor is computed. The output of the Extractor is the generated random bit sequence (denoted by t). In Figure 3, we use "**EH-PRNG(S1)**" to denote the exponentially hard PRNG $\mathsf{prng} : \{0,1\}^{n_{\mathsf{prng}}} \rightarrow \{0,1\}^{m_{\mathsf{prng}}}$ with seed $S1 \in \{0,1\}^{n_{\mathsf{prng}}}$. The symbol "**Extractor(S2)**" denotes the Extractor $\mathsf{ext} : \{0,1\}^{n_{\mathsf{ext}}} \times \{0,1\}^{m_{\mathsf{prng}}} \rightarrow \{0,1\}^{m_{\mathsf{ext}}}$ with random input $S2 \in \{0,1\}^{n_{\mathsf{ext}}}$. The random input $S2$ is chosen uniformly at random from $\{0,1\}^{n_{\mathsf{ext}}}$ at initialization time and it remains unchanged during each invocation.

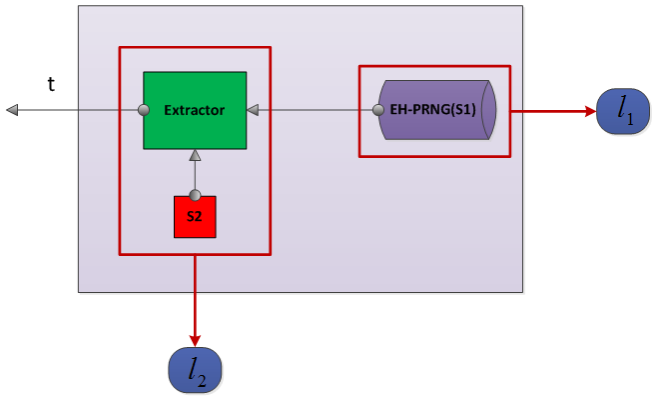

Fig. 3. Our Suggested Way to Generate Random Numbers for Scheme EG^*

According to *"Only Computation Leaks Information"* axiom, the two phases leak information individually and we use two leakage functions (efficient computable functions chosen by the adversary) to describe leakages during computation of the two phases. We use leakage function $l_1(S1) : \{0,1\}^{n_{\mathsf{prng}}} \rightarrow \{0,1\}^{\lambda_1}$ to describe leakages from the first phase. The leakage function $l_1(S1)$ can simulate computation during the first phase (The adversary can encode the exponentially hard PRNG into l_1.) and output information about both the seed and the output of the exponentially hard PRNG $\mathsf{prng}(S1)$. But the length of output of $l_1(S1)$ is bounded by leakage parameter λ_1. We use leakage function $l_2(S2, \mathsf{prng}(S1)) : \{0,1\}^{n_{\mathsf{ext}}} \times \{0,1\}^{m_{\mathsf{prng}}} \rightarrow \{0,1\}^{\lambda_2}$ to describe leakages from computation of the second phase. We do not allow the leakage function $l_2(S2, \mathsf{prng}(S1))$ to output any information about $\mathsf{ext}(S2, \mathsf{prng}(S1))$ (i.e. the generated bit sequence). This means that the adversary only obtains leakages from the internal states of the suggested way. We will discuss leakages from the generated bit sequence later. Note that, no leakages occur at the absence of computation during the two phases according to *"Only Computation Leaks Information"* axiom.

The following theorem guarantees that any probabilistic polynomial-time adversary can not distinguish the bit sequence generated by the suggested way from a true random bit sequence even if he obtains leakages from leakage functions l_1 (with sufficiently short output about both $S1$ and $\mathsf{prng}(S1)$) and l_2 when the exponentially hard PRNG and Extractor satisfy some requirements.

Theorem 1. *The seed $S1$ is chosen uniformly at random from $\{0,1\}^{d\kappa}$. Let $\mathsf{prng}(S1) : \{0,1\}^{d\kappa} \to \{0,1\}^{m_{\mathsf{prng}}}$ is a $(2^{(c-1)d\kappa}, 2^{cd\kappa})-$secure pseudorandom random number generator, where the constant $c \in (0,1)$. Let $\mathsf{ext}(S2, \mathsf{prng}(S1)) : \{0,1\}^{n_{\mathsf{ext}}} \times \{0,1\}^{m_{\mathsf{prng}}} \to \{0,1\}^{\kappa}$ be a $(\epsilon, m_{\mathsf{prng}} - 2a\kappa - 0.5\kappa - 1)$ extractor, where ϵ is negligible in κ and $S2$ is chosen uniformly at random from $\{0,1\}^{n_{\mathsf{ext}}}$. Let $l_1(S1) : \{0,1\}^{d\kappa} \to \{0,1\}^{0.25\kappa}$ $(\lambda_1 = 0.25\kappa)$ and $l_2(S2, \mathsf{prng}(S1)) : \{0,1\}^{n_{\mathsf{ext}}} \times \{0,1\}^{m_{\mathsf{prng}}} \to \{0,1\}^{0.25\kappa}$ $(\lambda_2 = 0.25\kappa)$ be two leakage functions but $l_2(S2, \mathsf{prng}(S1))$ is restricted to output any information about $\mathsf{ext}(S2, \mathsf{prng}(S1))$. For a probabilistic polynomial-time adversary in κ, it holds that*

$$\delta((\mathsf{ext}(S2, \mathsf{prng}(S1)), S2); (U_\kappa, S2)) \leq \epsilon$$

even if he obtains $\{l_1(S1), l_2(S2, \mathsf{prng}(S1))\}$ as long as

$$d \in \Big(max\Big\{\Omega\Big(\frac{1}{\kappa 2^{2a\kappa}}\Big), \frac{(8a+1)\kappa + 4}{4(1-c)\kappa}, 0.25\Big\}, 1\Big),$$

where parameter $a \in (0,1)$ depends on the security level.

Proof. The function $\mu(\kappa) = 2^{-a\kappa}$ (The value $a \in (0,1)$ is a constant.) is negligible in κ. We first prove the following Lemma holds.

Lemma 1 *Let prng, l_1, and l_2 satisfy the requirements of Theorem 1. Then, for any probabilistic polynomial-time adversary $\mathcal{A}$ in κ, it holds that*

$$\Big|\mathsf{Pr}[\mathcal{A}(l_1(S1), S2, \mathsf{prng}(S1)) = 1] - \mathsf{Pr}[\mathcal{A}(l_1(S1), S2, Y) = 1]\Big| < 2\mu(\kappa),$$

where $\mathsf{H}_\infty(Y) \geq m_{\mathsf{prng}} - 2a\kappa - 0.25\kappa - 1$.

Proof. We can prove Lemma 1 based on the following Lemma.

Lemma 2 *Let $\mathsf{prng}(S1) : \{0,1\}^{d\kappa} \to \{0,1\}^{m_{\mathsf{prng}}}$ is a $(2^{(c-1)d\kappa}, 2^{cd\kappa})-$secure pseudorandom random number generator, where the constant $c \in (0,1)$. Then if*

$$2^{(c-1)d\kappa} \leq \tfrac{\mu(\kappa)^2}{2^{0.25\kappa}} - 2^{-2a\kappa - 0.25\kappa - 1}$$

and $S1 \sim U_{d\kappa}$, it holds that

$\mathsf{Pr}_y[\mathsf{H}^{\mathsf{HILL}}_{2\mu(\kappa), \Omega(\mu^2(\kappa)2^{cd\kappa}/d\kappa)}(\mathsf{prng}(S1)|l_1(S1) = y) \geq m_{\mathsf{prng}} - 2a\kappa - 0.25\kappa - 1] \geq 1 - \mu(\kappa)$.

This Lemma can be proved by Lemma 3 in [3] and Lemma 2.1 in [20]. For any probabilistic polynomial-time adversary $\mathcal{A}$ in κ, we assume the adversary runs in time κ^b, where b is a constant. According to Lemma 2, if the following three inequations

$$2^{(c-1)d\kappa} \leq \frac{\mu(\kappa)^2}{2^{0.25\kappa}} - 2^{-2a\kappa-0.25\kappa-1},$$

$$2^{cd\kappa} > \kappa^b,$$

$$\kappa^b \geq \Omega(\mu^2(\kappa)2^{cd\kappa}/d\kappa)$$

hold simultaneously, the adversary $\mathcal{A}$ can not distinguish the output of prng (i.e. $\mathsf{prng}(S1)$) and a random variable Y that $\mathsf{H}_\infty(Y) \geq m_{\mathsf{prng}} - 2a\kappa - 0.25\kappa - 1$ even if he obtains leakages about $S1$ with length at most 0.25κ bits. Then, Lemma 1 can be proven. It is clear that if prng, l_1, and l_2 satisfy the requirements of Theorem 1, the above three inequations hold simultaneously. □

The adversary can not distinguish the output of prng and a random variable Y that $\mathsf{H}_\infty(Y) \geq m_{\mathsf{prng}} - 2a\kappa - 0.25\kappa - 1$ even if he obtains $l_1(S_1)$. Because the adversary can only obtain at most 0.25κ bits leakage information about $\mathsf{prng}(S1)$ from l_2 and $\mathsf{prng}(S1)$ is input into a $(\epsilon, m_{\mathsf{prng}} - 2a\kappa - 0.5\kappa - 1)$ extractor. Therefore, it holds that $\delta((\mathsf{ext}(S2, \mathsf{prng}(S1)), S2); (U_\kappa, S2)) \leq \epsilon$ and this theorem holds. □

Analysis Note that, the adversary can also obtain the output of the suggested way (i.e. t) from leakages. Let leakage function $l_3(\mathsf{ext}(S2, \mathsf{prng}(S1))) : \{0,1\}^\kappa \to \{0,1\}^{\lambda_3}$ (chosen by the adversary) describes leakages from the output of the suggested way. We will analyze the possible leakage scenarios in the following.

Leakage scenario 1: $|l_1| + |l_2| \leq \lambda$ and $|l_3| = 0$

According to Theorem 1, for any probabilistic polynomial-time adversary $\mathcal{A}$ in κ, he can not distinguish the bit sequence generated by the suggested way from a true random bit sequence in this case.

Leakage scenario 2: $|l_1| + |l_2| > 0$, $|l_3| > 0$, and $|l_1| + |l_2| + |l_3| \leq \lambda$

In this case, any probabilistic polynomial-time adversary $\mathcal{A}$ in κ can not distinguish the bit sequence generated by the suggested way from a random variable Y such that $\mathsf{H}_\infty(Y) \geq \kappa - (\lambda - |l_3|)$.

Leakage scenario 3: $|l_1| = |l_2| = 0$ and $|l_3| \leq \lambda$

The adversary only obtains $|l_3|$ bits information about t from leakages and all the internal states of the suggested way are leakage-free.

From Theorem 1, we can see that the length of the seed of our suggested way (i.e. $d\kappa$) depends on the security level (the value of a) and the size of the adversary against the exponentially hard PRNG (the value of c). Specifically speaking, on one hand, for a fixed security level, the length of the seed of our suggested way should be longer for more powerful adversary. On the other hand,

for a fixed size adversary, the length of the seed of our suggested way should be longer for higher security level. Our suggested way can also be used for other leakage resilient cryptographic schemes similarly.

5 Conclusions and Future Work

Our results show that there exists a big gap between theoretical security of leakage resilient cryptographic scheme and practical security of mathematical realization of the same scheme. A leakage resilient cryptographic scheme may not be practically secure when it is mathematically realized in accustomed ways even if its theoretical security still holds.

It is well known that specifying all details of implementation in a leakage model is tedious. Moreover, it is not clear if it is feasible at all to prove anything without assuming some kind of bounded leakages at higher abstraction level (like mathematical realization at algorithmic level). For example, the paper [34] shows that it is very difficult to state assumptions at logic gate level. So, even from the practical point of view, working at higher abstraction level seems appealing. Therefore, we suggest that all (practical) leakage resilient cryptographic schemes should at least come with a kind of mathematical realization whose practical security can be guaranteed. Our results also inspire cryptographers to design advanced leakage resilient cryptographic schemes whose practical security of mathematical realization is independent of specific mathematical realization.

In this paper, we only consider mathematical realization of the process of generating random numbers. Whether mathematical realization of other cryptographic components would affect practical security of mathematical realization of a leakage resilient cryptographic scheme is still not known. For other leakage resilient cryptographic schemes in different kinds of leakage models, we believe similar problems are also existent. These questions themselves are rather interesting and worthy of research. Our results show a new perspective about security of leakage resilient cryptographic schemes.

References

1. E. Kiltz, K. Pietrzak.: Leakage Resilient ElGamal Encryption. ASIACRYPT2010, LNCS 6477, pp.595-612, 2010.
2. http://homepages.cwi.nl/∼pietrzak/publications/DP08.pdf
3. S. Dziembowski, K. Pietrzak.: Leakage-Resilient Cryptography. FOCS2008, pp.293-302, 2008. See [2] for an improved version of this paper.
4. S. Micali, L. Reyzin.: Physically Observable Cryptography (Extended abstract). TCC2004, LNCS2951, pp.278-296, 2004.
5. J.A. Halderman, S.D. Schoen, N. Heninger, W. Clarkson, W. Paul, J.A. Calandrino, A.J. Feldman, J. Appelbaum and E.W. Felten.: Lest we remember: cold-boot attacks on encryption keys. Communications of the ACM - Security in the Browser Volume 52 Issue 5, pp.91-98, 2009.
6. F.-X. Standaert, O. Pereira, Yu Yu, J.J. Quisquater, M. Yung and E. Oswald.: Leakage Resilient Cryptography in Practice. Towards Hardware-Intrinsic Security, Information Security and Cryptography2010, Part 2, pp.99-134, 2010.

7. A. Akavia, S. Goldwasser and V. Vaikuntanathan.: Simultaneous Hardcore Bits and Cryptography against Memory Attacks. TCC2009, LNCS 5444, pp.474-495, 2009.
8. M. Naor, G. Segev.: Public-key Cryptosystems Resilient to Key Leakage. CRYPTO2009, LNCS 5677, pp.18-35, 2009.
9. S. Dziembowski.: Intrusion-Resilience Via the Bounded-Storage Model. TCC2006, LNCS 3876, pp.207-224, 2006.
10. S. Dziembowski.: On Forward-Secure Storage (Extended abstract). CRYPTO2006, LNCS 4117, pp.251-270, 2006.
11. D. Cash, Y.Z. Ding, Y. Dodis, W. Lee, R.J. Lipton, S. Walfish.: Intrusion-Resilient Key Exchange in the Bounded Retrieval Model. TCC2007, LNCS 4392, pp.479-498, 2007.
12. S. Dziembowski, K. Pietrzak.: Intrusion-Resilient Secret Sharing. FOCS2007, pp.227-237, 2007.
13. J. Alwen, Y. Dodis and D. Wichs.: Leakage-Resilient Public-Key Cryptography in the Bounded-Retrieval Model. CRYPTO2009, LNCS 5677, pp.36-54,2009.
14. J. Alwen, Y. Dodis, M. Naor, G. Segev, S. Walfish and D. Wichs.: Public-Key Encryption in the Bounded-Retrieval Model. EUROCRYPT2010, LNCS 6110, pp.113-134, 2010.
15. Y. Dodis, K. Haralambiev, A. López-Alt and D. Wichs.: Cryptography against Continuous Memory Attacks. FOCS2010, pp.511-520, 2010.
16. Z. Brakerski, Y.T. Kalai, J. Katz and V. Vaikuntanathan.: Overcoming the Hole in the Bucket: Public-Key Cryptography Resilient to Continual Memory Leakage. FOCS2010. pp.501-510 , 2010.
17. A. Lewko M. Lewko and B. Waters.: How to Leak on Key Updates. STOC2011, pp.725-734, 2011.
18. ANSI X 9.17 (Revised), American National Standard for Financial Institution Key Management (Wholesale)," American Bankers Association, 1985.
19. National Institute for Standards and Technology, Digital Signature Standard," NIST FIPS PUB 186, U.S. Department of Commerce, 1994.
20. B. Fuller, L. Reyzin.: Computational Entropy and Information Leakage. IACR Cryptology ePrint Archive 2012: 466, 2012.
21. S.S. Keller.: NIST-Recommended Random Number Generator Based on ANSI X9.31 Appendix A.2.4 Using the 3-Key Triple DES and AES Algorithms
22. J. Kelsey, B. Schneier, D. Wagner, and C. Hall.: Cryptanalytic Attacks on Pseudorandom Number Generators. Fifth International Workshop Proceedings(March 1998), Springer-Verlag, 1998, pp. 168-188.
23. http://www.spms.ntu.edu.sg/Asiacrypt2010/AsiaCrypt_slides/pietrzakAC11.pdf.
24. F.-X. Standaert.: How Leaky is an Extractor?. LATINCRYPT2010, LNCS 6212, pp.294-304, 2010.
25. D. Galindo, S. Vivek.: Limits of a conjecture on a leakage-resilient cryptogystem. Information Processing Letters Vol. 114, Issue 4, pp.192-196, 2014.
26. Y. Yu, F.-X. Standaert, O. Pereira, and M. Yung.: Practical Leakage-Resilient Pseudorandom Generators. CCS2010.
27. Y. Dodis, S. Goldwasser, Y.T. Kalai, C. Peikert, and V. Vaikuntanathan.: Public-Key Encryption Schemes With Auxiliary Inputs. TCC2010, LNCS 5978, pp.361-381, 2010.
28. Y. Dodis, Y.T. Kalai, and S. Lovett.: On Cryptography With Auxiliary Input. STOC2009.
29. A. Menezes, P. van Oorschot, and S. Vanstone.: Handbook of Applied Cryptography, Chapter 8, pp.296, CRC Press,1996.

30. C. Clavier, M. Joye.: Universal exponentiation algorithm. CHES2001, LNCS 2162, pp.300-308, 2001.
31. P.C. Kocher.: Timing Attacks On Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. Crypto1996, LNCS 1109, pp.104-113, 1996.
32. E. Trichina, A. Bellezza.: Implementation of Elliptic Curve Cryptography with Built-in Counter Measures against Side Channel Attacks. CHES2002, LNCS 2729, pp.61-77, 2003.
33. K. Pietrzak.: A Leakage Resilient Mode of Operation. EUROCRYPT2009, LNCS 5479, pp.462-482, 2009.
34. S. Mangard, T. Popp, and B.M. Gammel.: Side-Channel Leakage of Masked CMOS Gates. CT-RSA2005, LNCS 3376, pp.351-365, 2005.

Appendix A: The Result of Our Attacks

We use $\rho_{ATTACKI}$ (resp. $\rho_{ATTACKII}$) to denote the specific value of ρ for ATTACK I (resp. ATTACK II). We define $\lambda_{ATTACKI}$ (resp. $\lambda_{ATTACKII}$) to denote the specific value of leakage parameter λ for ATTACK I (resp. ATTACK II). We use v to denote how many times the PRNG is invoked in order to generate the random number r_i.

Table A.1. Attack results about ANSI X9.17 PRNG

$\|p\|$ (in bits)	$\rho_{ATTACKI}$	$\rho_{ATTACKII}$	$\lambda_{ATTACKI}$	$\lambda_{ATTACKII}$	v (times)
700	100.29%	25.14%	351	175	11
704	100.28%	25.00%	353	175	11
832	100.24%	23.56%	417	195	13
960	100.21%	22.50%	481	215	15
1088	100.18%	21.69%	545	235	17

Table A.2. Attack results about about FIPS 186 PRNG

$\|p\|$ (in bits)	$\rho_{ATTACKI}$	$\rho_{ATTACKII}$	$\lambda_{ATTACKI}$	$\lambda_{ATTACKII}$	v (times)
964	100.21%	25.10%	483	241	7
1120	100.18%	21.61%	561	241	7
1280	100.16%	18.91%	641	241	8
1440	100.14%	16.81%	721	241	9
1600	100.13%	15.13%	801	241	10

Table A.3. Attack results about leakage resilient PRNG instantiated with AES-128

$\|p\|$ (in bits)	$\rho_{ATTACKI}$	$\rho_{ATTACKII}$	$\lambda_{ATTACKI}$	$\lambda_{ATTACKII}$	v (times)
804	100.24%	25.12%	403	201	7
896	100.22%	22.54%	449	201	7
1024	100.20%	19.73%	513	201	8
1152	100.17%	17.53%	577	201	9
1280	100.16%	15.78%	641	201	10

Appendix B: The Attack Processes

We show our attack processes in the following.

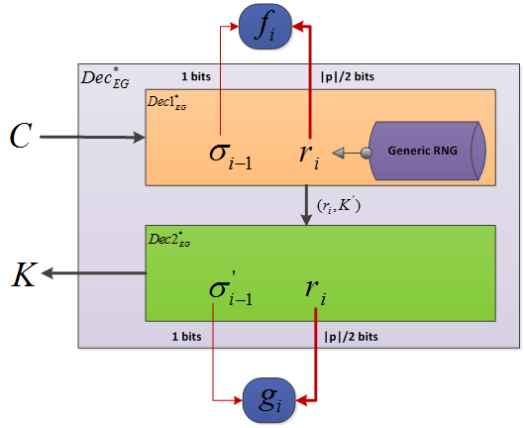

Fig. 4. our attack on decapsulation of EG* with generic RNG

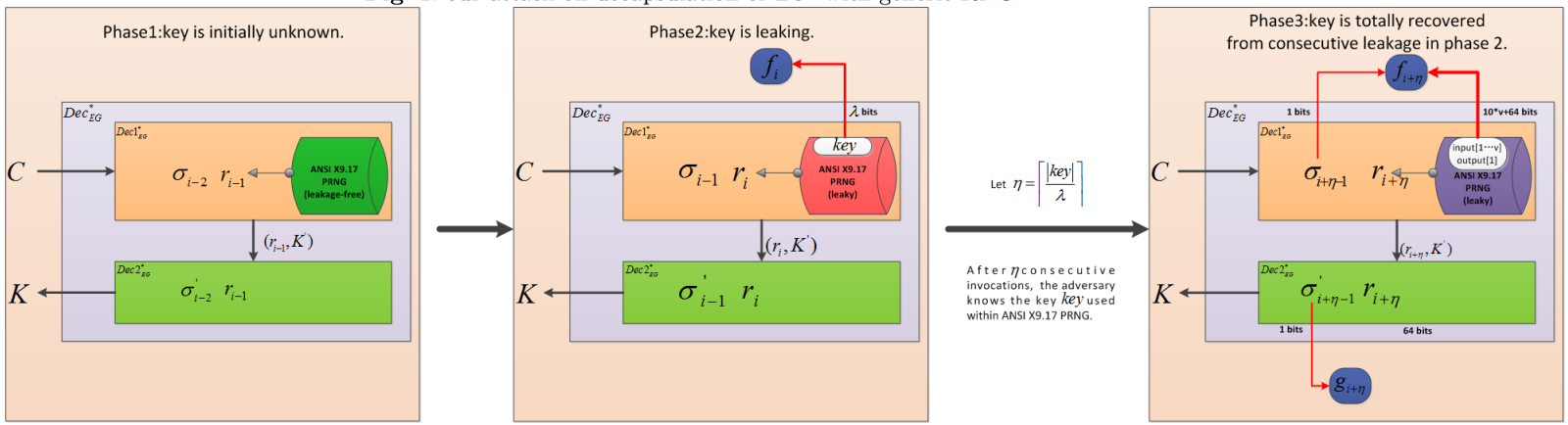

Fig. 5. our attack on decapsulation of EG* with a leaky ANSI X9.17 PRNG