# 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 being 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. It is well known that the process of generating random numbers can be mathematically realized with Pseudorandom Number Generator (PRNG) for cryptographic schemes in traditional leakage-free context without affecting their practical security of mathematical realization. However, it is unknown that whether one can use PRNG to mathematically realize this process for leakage resilient cryptographic schemes without affecting practical security of mathematical realization.

Our results show that if one directly uses PRNG to mathematically realize this process, some leakage resilient cryptographic schemes may not be practical secure any more. Furthermore, we give out a suggested way to mathematically realize this process with exponentially hard PRNG and extractor without affecting practical security of mathematical realization of a leakage resilient 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 when the process of generating random numbers is mathematically realized by PRNG.

**Keywords:** Leakage Resilient Cryptography, Mathematical Realization.

## 1 Introduction

In real world, in order to transform any abstract and generic cryptographic scheme into actual physical implementation, one usually undergoes two pro-

cesses: mathematical realization at algorithmic level and physical realization at implementation level. *Mathematical realization* refers to a process in which any abstract and generic cryptographic scheme is being transformed into an exact and specific mathematical scheme (After this process, the cryptographic construction is mathematically realized.). This means that all the cryptographic components used by the cryptographic scheme are instantiated with exact and specific mathematical schemes. 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 user of the public key encryption scheme chooses a specific family of one-way trapdoor permutation (such as RSA trapdoor permutation or Rabin trapdoor permutation) to mathematically realize the public key encryption scheme in this process. *Physical realization* refers to a process in which any exact and specific mathematical scheme of the cryptographic scheme (the output of mathematical realization) is being transformed into a physical cryptographic module that runs as a piece of software, or hardware, or combination of both. Broadly, and also more importantly, it has been turned out that physical realization has significant impact on the practical security of the cryptographic scheme in black-box model (i.e. leakage-free setting), so it is with those of any leakage resilient cryptographic schemes [24].

We say a kind of mathematical realization of a leakage resilient cryptographic scheme is practical secure as long as (1) The adversary can not get more leakage information about all secret states than the amount of leakage information defined in the leakage model even if all the internal states of mathematical cryptographic components can be leaked to the adversary but the amount of information leaked in each invocation remains the same as the amount of leakage information defined in the leakage model. For example, in the leakage model, the amount of leakage information about a secret state $x$ is bounded by leakage parameter $\lambda$ (i.e. $|f(x)| \leq \lambda$, where $f$ is the leakage function.). The adversary can not obtain more leakage information about the secret state $x$ than $\lambda$ bits. (2) The exact and specific mathematical scheme satisfies the requirement of the leakage model and can not be broken by the adversary.

***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 follow the "*Only Computation Leaks information*" axiom, which stats that memory content that is not accessed during an invocation, does not leak [4]. Leakage resilient stream cipher [3] and leakage resilient ElGamal encryption [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 the output length of $f(\cdot)$ is bounded. A lot of leakage resilient cryptographic schemes are build under this assumption [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 schemes in these models ignores any specific mathematical realization of the schemes. In other words, theoretical security only holds for mathematical realization of the scheme which fits the claimed leakage model.

Some cryptographic schemes (e.g. probabilistic encryption schemes) exploit random numbers to guarantee theoretical security. If one wants to transform such a abstract cryptographic scheme into actual physical implementation, the process of generating random numbers must also be mathematically realized. In traditional leakage-free context (i.e. black-box model), this process can be mathematically realized by True Random Number Generator (TRNG) or Pseudorandom Number Generator (PRNG) without affecting theoretical security of the cryptographic scheme (as well as practical security of mathematical realization of the cryptographic scheme).

Now, let us only consider the process of generating random numbers and ignore other possible cryptographic components which also need to be mathematically realized. For a leakage resilient cryptographic scheme, if the process of generating random numbers is mathematically realized by TRNG, theoretical security of the cryptographic scheme still holds. Moreover, the mathematical realization of the leakage resilient cryptographic scheme is practical secure. Although there are some TRNGs, PRNGs are used more widely 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.

However, it is *unknown* that whether mathematical realization of a leakage resilient cryptographic scheme is practical secure when the process of generating random numbers is mathematically realized by PRNG (or leakage resilient PRNG). Furthermore, if mathematical realization of a leakage resilient cryptographic scheme is not practical secure when the process of generating random numbers is mathematically realized by PRNG, whether there exists some sophisticated way to mathematically realize this process with PRNG without affecting practical security of mathematical realization of the cryptographic scheme? No previous work has concerned on these questions.

In this paper, in order to answer these important questions, we will introduce four different kinds of mathematical realization of the leakage resilient ElGamal scheme instantiated over arbitrary groups of prime order $p$ (where $p-1$ is not

smooth) in the paper of E. Kiltz et al. at Asiacrypt2010[1] [1] (i.e. the scheme $\mathsf{EG}^*$). In each mathematical realization, we use a PRNG or a leakage resilient PRNG to mathematically realize the process of generating random numbers. We want to see if the four kinds of mathematical realization are practical secure by attacks against the four kinds of mathematical realization.

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

***Our Contributions*** Main contributions of this paper are two-folds as follows. First, our research shows that if one directly uses PRNG to mathematically realize the process of generating random numbers for some leakage resilient cryptographic schemes, the mathematical realization may not be practical secure. Therefore, one should carefully mathematically realize this process with PRNG. Second, we prove that one can use exponentially hard PRNG and extractor to mathematically realize the process of generating random numbers without affecting practical security of mathematical realization. The generated bit sequence can not be distinguish with a random bit sequence for probabilistic polynomial time adversary even if he obtains leakages from the computation of this process. Moreover, the length of the seed used by the exponentially hard PRNG is shorter than the length of the required bit sequence. Hence, it is not necessary to use TRNG to generate random numbers for some leakage resilient cryptographic schemes.

***Organization of This Paper*** The rest of paper is organized as follows. In Section 2, we first present some basic symbols, notations, and concepts. Then, we briefly review the scheme $\mathsf{EG}^*$. Section 3 introduce several kinds of mathematical realization of $\mathsf{EG}^*$ with TRNG or PRNG and their practical security. In section 4, we suggest a new way to mathematically realize the process of generating random numbers with exponentially hard PRNG and extractor. 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 $\mathsf{EG}^*$.

### 2.1 Symbols, Notations, and Concepts

If $S$ is a binary bit string, we denote the most significant $a$ bits of $S$ by $S^{[a]}$ and denote the least significant $b$ bits of $S$ by $S_{[b]}$. We denote the length of $S$ by $|S|$ and assume that the binary bit string representation of all elements in $\mathbb{Z}_p$ has

[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. the scheme $\mathsf{BEG}^*$). However, it is very hard to implement the generic-group model in practice. Therefore, we do not consider the scheme $\mathsf{BEG}^*$ in this paper. We only consider the scheme $\mathsf{EG}^*$ which can be implemented in practice easily.

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 $|S|^{th}$ bit of $S$. We use the symbol $[S]_{(i)}$ to denote the $i^{th}$ bit of $S$.

We use $U_n$ to denote 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 $\nu$ 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 $\epsilon$ is negligible in the security parameter. With $\delta^{\mathsf{D}}(X;Y)$ the advantage of a circuit $\mathsf{D}$ in distinguishing the random variables $X, Y$, 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 $(\epsilon, k)$ 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 $\delta_s(\mathsf{prng}(U_n); U_m) < \epsilon$.*

We say that a PRNG $\mathsf{prng} : \{0,1\}^{n_{\mathsf{prng}}} \to \{0,1\}^{m_{\mathsf{prng}}}$ 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 $\mathsf{EG}^*$

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 $\mathsf{EG}^*$ is $\kappa$. Let $\mathsf{Gen}$ denotes a probabilistic algorithm that outputs a cyclic group $\mathbb{G}$ of order $p$, where $p$ is a strong prime and $logp = \kappa$. The scheme $\mathsf{EG}^*$ is described as a Key Encapsulation Mechanism (KEM) and is shown as follows:

$\mathsf{KG}^*_{\mathsf{EG}}(\kappa)$: Compute $(\mathbb{G}, p) \stackrel{*}{\leftarrow} \mathsf{Gen}(n)$, $g \stackrel{*}{\leftarrow} \mathbb{G}$, $x \stackrel{*}{\leftarrow} \mathbb{Z}_p$, $h = g^x$. Choose random $\sigma_0 \stackrel{*}{\leftarrow} \mathbb{Z}^*_p$ 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 $\sigma_0$ and $\sigma'_0$.

$\mathsf{Enc}^*_{\mathsf{EG}}(pk)$: 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 state information are $\sigma_i$ and $\sigma'_i$.

The security definition of $\mathsf{EG}^*$ is CCLA1 which was introduced in [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 $\lambda$. For $\mathsf{EG}^*$, the leakage functions $f_i$ and $g_i$ are as follows:

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

In [1], E. Kiltz et al. didn't prove the theoretical security of $\mathsf{EG}^*$ and only presented the following conjecture. They also pointed out that there exists some attack about $\mathsf{EG}^*$ [23,25], which exploits the method of Hiding Number Problem if $\lambda \geq \frac{3}{8} logp + o(logp)$. Therefore, they conjectured that roughly $\lambda$ equals to $0.25 \cdot logp$ bits in [23]. Thus the number of total tolerance leakage bits in one decryption query equals to $2\lambda = 0.5 \cdot logp$ bits.

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

We use $\lambda/|p|$ to denote tolerance leakage rate of the scheme $\mathsf{EG}^*$ and let $\rho = \frac{|f_i|+|g_i|}{|p|}$. Any implementation of the scheme $\mathsf{EG}^*$ 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 on 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 the random number $r_i$ for scheme $\mathsf{EG}^*$. Therefore, the user may use TRNG or PRNG to mathematically realize this process.

## 3 Several Kinds of Mathematical Realization of Scheme $\mathsf{EG}^*$ With TRNG or PRNG and Their Practical Security

In this section, we will introduce four kinds of mathematical realization of scheme $\mathsf{EG}^*$. In each mathematical realization, the process of generating random numbers is mathematically realized by TRNG or PRNG (leakage resilient PRNG). We want to see whether the four kinds of mathematical realization are practical secure, by presenting specific attacks against them. The goal of 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 $r_i$ and 1 bit of each one of the two multiplicative secret shares $\sigma_i$ and $\sigma'_i$ in each continual invocation of the decryption query. The adversary can use these leakage bits to recover all the bits of $\sigma_0$ and $\sigma'_0$ ($\sigma_i$ and

$\sigma_i'$) and then recover a candidate value $x'$ of the 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 process of generating random numbers $r_i$ is mathematically realized by TRNG. The attack against this mathematical realization (denoted by ATTACK I) can also be viewed as an attack method against theoretical security of the scheme $\mathsf{EG}^*$ and satisfies the leakage model of the scheme $\mathsf{EG}^*$ except that it requires a high leakage rate. Therefore, ATTACK I poses no threat on the theoretical security of the scheme $\mathsf{EG}^*$.

In the rest three kinds of mathematical realization, we assume the process of generating random numbers $r_i$ is mathematically realized by PRNG (leakage resilient PRNG). The attacks against the three kinds of mathematical realization are denoted by ATTACK II. ATTACK II has the same basic principle as ATTACK I. However, it is amazing that the results of ATTACK II show that the practical tolerance leakage rate of mathematical realization of the scheme $\mathsf{EG}^*$ will decrease dramatically when PRNG is used to mathematically realize the process of generating random numbers $r_i$. Because, the adversary can obtain leakages from the seed of the PRNG. Therefore, mathematical realization of the scheme $\mathsf{EG}^*$ is not practical secure when the process of generating random numbers $r_i$ is mathematically realized with these PRNGs. For ATTACK I and ATTACK II, we assume the random number $r_i$ is generated by Algorithm 1.

**Algorithm 1** The Algorithm of Generating Random Number $r_i$

**Input:** no input
**Output:** a random number $r_i$
**Step 1** Invoke TRNG 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 $r_i$.

In the following, we will introduce the four kinds of mathematical realization and attacks against them. Finally, we will show results of the attacks.

### 3.1 Mathematical Realization Using TRNG

If the process of generating random numbers $r_i$ is mathematically realized by TRNG, we can attack this kind of mathematical realization with the following attack method (ATTACK I):

In the $1^{st}$ invocation of decryption query of the scheme $\mathsf{EG}^*$, 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 $r_1$, $r_1^{-1}$ (The prime number $p$ is public.), $\sigma_{0[1]}$, and $\sigma'_{0[1]}$. In the $2^{nd}$ invocation of decryption query of the scheme $\mathsf{EG}^*$, 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 decryption query, the adversary knows $r_1$, $r_1^{-1}$, $r_2$, $r_2^{-1}$, $\sigma_{0[2]}$, and $\sigma'_{0[2]}$. Let $R_0 = R_0^{-1} = 1$. For $i > 0$, let $R_i = \prod_{j=1}^{i} r_i \bmod p$ and $R_i^{-1} = \prod_{j=1}^{i} r_i^{-1} \bmod p$. Clearly, after the $(i-1)^{th}$ $(i > 1)$ invocation of decryption query, the adversary knows $R_{i-1}$, $R_{i-1}^{-1}$, $\sigma_{0[i-1]}$, and $\sigma'_{0[i-1]}$. In the $i^{th}$ invocation of decryption query of the scheme $\mathsf{EG}^*$, the adversary chooses the leakage functions as follows:

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

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

In this way, after invoking the decryption query *logp* times, the adversary knows all the bits of $\sigma_0$ and $\sigma'_0$ and can recover a candidate value $x' = \sigma_0 \sigma'_0 \bmod p$ of the 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 $\lambda$ should achieve $0.5 \cdot logp + 1$ bits, which is lager than $0.25 \cdot logp$. Therefore, ATTACK I poses no threat on the theoretical security of the scheme $\mathsf{EG}^*$.

Note that, ATTACK I can also be executed after $i^{th}$ decryption query similarly. The adversary can obtain $\sigma_i$ and $\sigma'_i$ and can recover a candidate value $x' = \sigma_i \sigma'_i \bmod p$ of the 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 PRNG (leakage resilient PRNG). According to Kerckhoffs' principle, the adversary knows concrete mathematical structure of the specific PRNG. When the PRNG is invoked to generate a random number $r_i$ in the decryption query, all internal secret states of the PRNG can be leaked to the adversary due to the *"Only Computation Leaks information"* axiom. Therefore, if the adversary obtains all bits of the seed of the PRNG from leakage function $f_i$, he will recover the output of the PRNG and the random number $r_i$[1]. The leakage functions of ATTACK II are in the following form:

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

[1] For some PRNG, the adversary need to obtain some other internal states from leakage function $f_i$.

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

The symbol $seed_i$ represents the seed of the PRNG when it is invoked to generate the random number $r_i$.

We conjecture the practical tolerance leakage rate of the scheme $\mathsf{EG}^*$ is also 0.25 (i.e. $\frac{\lambda}{|p|} = 0.25$) like the theoretical tolerance leakage rate. However, we surprisingly find that practical tolerance leakage rate of the scheme $\mathsf{EG}^*$ will be reduced when four specific PRNGs are used to mathematically realize the process of generating random numbers $r_i$. Therefore, mathematical realization of the scheme $\mathsf{EG}^*$ is not practical 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 a specific PRNG is refreshed in each invocation of the decryption 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 $key$ is part of the secret state of the PRNG which is never changed by any PRNG input. 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, the adversary can obtain $key$ from leakage function $f_i$ by invoking the decryption query repeatedly. After knowing $key$, the adversary continually invoke the decryption query for $logp$ times and the leakage functions are defined as follows:

$$f_{i+1}(\sigma_i, r_{i+1}) = \langle [\sigma_i]_{(1)}, seed[1]_{i+1}, input[1]_{i+1[10]}, \ldots, input[v]_{i+1[10]} \rangle,$$

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

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

$$f_{i+u}(\sigma_{i+u-1}, r_{i+u}) = \langle [\sigma_{i+u-1}(r_{i+1} \cdot \ldots \cdot r_{i+u-1})^{-1} \bmod p]_{(u)}, \\ seed[1]_{i+u}, input[1]_{i+u[10]}, \ldots, input[v]_{i+u[10]} \rangle,$$

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

$u = 2, \ldots, logp$.

Let $\{seed[1]_{i+u}, input[1]_{i+u}, \ldots, input[v]_{i+u}\}$, $(u = 1, \ldots, logp)$ denote the secret states of this PRNG when it is used to generate the random number $r_{i+u}$. In the $logp$ invocation of the decryption query, the adversary knows all bits about the secret states $\{seed[1]_{i+u}, input[1]_{i+u}, \ldots, input[v]_{i+u}\}$ and can compute the random numbers $r_{i+u}$, $(u = 1, \ldots, logp)$. Therefore, our attack is valid. The attack process is shown in Figure 5 in Appendix B.

Figure 1, Figure 2 and Table A.1 in Appendix A show that the scheme $\mathsf{EG}^*$ is not practical 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 attack against this PRNG here.

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

The Digital Signature Standard 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. If the adversary obtains all the bits of the seed, he can recover the output of the PRNG. Therefore, our attack is valid. Figure 1, Figure 2 and Table A.2 in Appendix A show that the scheme $\mathsf{EG}^*$ is not practical secure any more, if it uses this PRNG for strong prime $p$ with size larger than 644 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[1]. This practical leakage resilient PRNG is based on $(\epsilon, s, n/\epsilon)-$secure weak pseudorandom function(wPRF) $\mathsf{F}(k, pr) : \{0,1\}^\kappa \times \{0,1\}^n \rightarrow \{0,1\}^m$. The symbol $pr$ denotes public randomness. The initial state of this PRNG is $(pr_0, pr_1, k_0)$ for public randomness $(pr_0, pr_1) \xleftarrow{*} (\{0,1\})^2$ and secret key $k_0 \xleftarrow{*} \{0,1\}^\kappa$. 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 PRNG can be instantiated with any length-expanding wPRF ($m > \kappa$), which in turn can be realized from any secure block cipher $\mathsf{BC} : \{0,1\}^\kappa \times \{0,1\}^n \rightarrow \{0,1\}^\kappa$. That is, if $\mathsf{BC}$ 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.

Clearly, if the secret key $k_0$ is leaked to the adversary totally, the adversary can recover all outputs of this PRNG due to the randomness $(pr_0, pr_1)$ are public. We assume the secure block cipher $\mathsf{BC}$ is instantiated with AES-128. Figure 1, Figure 2, and Table A.3 in Appendix A show that the scheme $\mathsf{EG}^*$ is not practical secure any more, if it uses this PRNG for strong prime $p$ with size larger than 516 bits.

When $\lambda = 0.25 \cdot logp$ and $\epsilon = 2^{-a\kappa}$ (The value $a \in (0, 1]$ is a constant.), this PRNG is leakage resilient if and only if the length of the seed is larger than $\frac{1.5 \cdot logp}{a} \geq 1.5 \cdot logp$. In this case, we can directly use TRNG to generate the random number $r_i$. Therefore, this PRNG is not suitable to generate random numbers for scheme $\mathsf{EG}^*$.

### 3.3 The Results of Our Two Attacks

Note that Attack II is still valid for mathematical realization using other PRNGs whose seeds are shorter than $0.25 \cdot logp$ bits. Figure 1. shows the minimum $\rho$ required to successfully recover $x$ for different PRNGs. Figure 2. shows the minimum $\lambda/|p|$ required to successfully recover $x$ for different PRNGs.

[1] We use the same symbol as the paper [26]. See the paper [26] for more details.

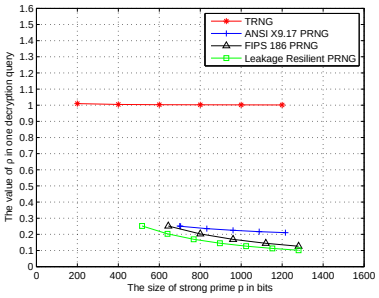


**Fig. 1.** Minimum $\rho$ required to successfully recover $x$ for different random number generators

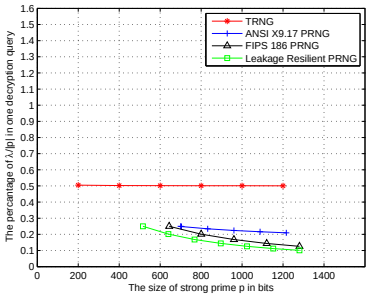


**Fig. 2.** Minimum $\lambda/|p|$ required to successfully recover $x$ for different random number generators

According to [29], a 512-bit modulus $p$ provides only marginal security from concerted attack. As of 1996, a modulus $p$ of at least 768 bits is recommended. For long-term security, 1024-bit or larger modulus should be used. Therefore, we can see that if the scheme $\mathsf{EG}^*$ uses the above-mentioned four PRNGs to generate random numbers $r_i$, the scheme will not be practical secure any more.

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. $\frac{3}{8}logp+o(logp)$) than that of ATTACK I. However, practical tolerance leakage rate of this attack method (also equals to $\frac{3}{8}logp + o(logp)$) is higher than that of ATTACK II when the process of generating random numbers is mathematically realized with PRNGs because this attack method requires leakages from $\sigma_i$ and $\sigma_i'$ but does not require leakages from $r_i$.

## 4 Our Suggested Way to Generate Random Numbers for Scheme $\mathsf{EG}^*$ Without Affecting Practical Security

From Section 3, we can see that mathematical realization of scheme $\mathsf{EG}^*$ is not practical secure when the process of generating random numbers $r_i$ is mathematically realized by some PRNGs. In this section, we try to find a sophisticated way to mathematically realize this process with PRNG. This suggested way should not affect practical security of mathematical realization of scheme $\mathsf{EG}^*$ and depends on random seed with length shorter than $\kappa = logp$ bits and larger than $0.25 \cdot logp$ bits. Therefore, it is not necessary to use TRNG to mathematically realize the process of generating random numbers $r_i$ for scheme $\mathsf{EG}^*$.

Our suggested way is shown in Figure 3 and is based on exponentially hard PRNG and extractor. The output of the exponentially hard PRNG is then input into an extractor. The output of the extractor is the generated random bit sequence (denoted by $t$).

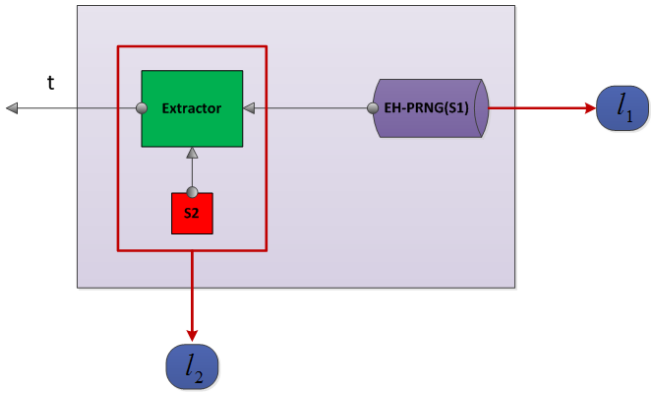


**Fig. 3.** Our Suggested Way to Generate Random Numbers for Scheme $\mathsf{EG}^*$

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.

We use leakage function $l_1(S1) : \{0,1\}^{n_{\mathsf{prng}}} \rightarrow \{0,1\}^{\lambda_1}$ to denote leakages from the seed of the exponentially hard PRNG. Note that, because $l_1$ is an efficiently computable leakage function like $f_i$ and $g_i$, it can output all the intermediate results during the computation of the PRNG[1]. But the length of output of $l_1(S1)$ is bounded by leakage parameter $\lambda_1$. 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 denote leakages from computation of the extractor.

The following theorem guarantees that the output of our suggested way is statistically indistinguishable with $U_\kappa$ even if the adversary obtains leakages from leakage functions $l_1$ and $l_2$ when the exponentially hard PRNG and the extractor satisfy some requirements.

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

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

[1] The adversary knows algorithm structure of the PRNG.

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

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

$$|\mathsf{Pr}[\mathcal{A}(l_1(S1), l_2(S2, \mathsf{prng}(S1))) = 1] - \mathsf{Pr}[\mathcal{A}(l_1(S1), l_2(S2, Y)) = 1]| < 2\mu(\kappa),$$

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

**Proof.** Due to the following Lemma, we can proof Lemma 1.

**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* $2^{(c-1)d\kappa} \leq \frac{\mu(\kappa)^2}{2^{0.25\kappa}} - 2^{-2a\kappa-0.25\kappa-1}$ *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]. Space does not permit to show the proof here.

For any probabilistic polynomial-time adversary $\mathcal{A}$ in $\kappa$, we assume the adversary runs in time $\kappa^b$, where $b$ is a constant. If $2^{cd\kappa} > \kappa^b$, the adversary $\mathcal{A}$ can not distinguish the output of $\mathsf{prng}$ and $U_{m_{\mathsf{prng}}}$ with non-negligible probability. In additional, if $\kappa^b \geq \Omega(\mu^2(\kappa)2^{cd\kappa}/d\kappa)$, the adversary $\mathcal{A}$ can not distinguish the output of $\mathsf{prng}$ and a random variable $Y$ that $\mathsf{H}_\infty(Y) \geq m_{\mathsf{prng}} - 2a\kappa - 0.25\kappa - 1$. Therefore, 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, Lemma 1 can be proven. It is clear that if $\mathsf{prng}$, $l_1$, and $l_2$ satisfy the requirements of Theorem 1, the above three inequations hold simultaneously. □

The adversary can not distinguish the output of $\mathsf{prng}$ and a random variable $Y$ that $\mathsf{H}_\infty(Y) \geq m_{\mathsf{prng}} - 2a\kappa - 0.25\kappa - 1$. Because the output of $\mathsf{prng}$ is input into a $(\epsilon, m_{\mathsf{prng}} - 2a\kappa - 0.5\kappa - 1)$ extractor and the adversary can only obtain at most $0.25 \cdot logp$ bits leakage information from $l_2$. Therefore, the adversary can

not distinguish the bit sequence generated by our suggested way and a random variable $Z \sim U_\kappa$. Therefore, this theorem holds. □

From Theorem 1, we can see that it is possible for our suggested way to output a bit sequence that can not be distinguished by a probabilistic polynomial-time adversary with a random bit sequence when the seed $S1$ is shorter than $logp$. Therefore, it is not necessary to use TRNG to mathematically realize the process of generating random numbers $r_i$ for scheme $\mathsf{EG}^*$. Our suggested way can also be used for other leakage resilient cryptographic schemes similarly.

## 5 Conclusions and Future Work

Our results show that if one directly uses PRNG to mathematically realize the process of generating random numbers, some leakage resilient cryptographic scheme may not be practical secure any more. We give out a suggested way to solve this problem. The suggested way is based on exponentially hard PRNG and extractor. This observation shows the big gap between theoretical security of leakage resilient cryptographic scheme and practical security of mathematical realization of the same cryptographic scheme when one uses PRNG to mathematically realize the process of generating random numbers. In this paper, we only consider mathematical realization of the process of generating random numbers. Whether mathematical realization of other cryptographic components will affect practical security of mathematical realization of a leakage resilient cryptographic scheme is still not known and may be the future work. This is a new perspective about security of leakage resilient cryptographic schemes. We also believe that there are more security drawbacks when we further consider practical security of physical realization of a leakage resilient cryptographic scheme.

## 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, Yan Zong 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, LNCS6212, 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. TCC 2010, LNCS 5978, pp.361-381, 2010.
28. Y. Dodis, Y.T. Kalai, and S. Lovett.: On Cryptography With Auxiliary Input. STOC 2009.
29. A. Menezes, P. van Oorschot, and S. Vanstone.: Handbook of Applied Cryptography, Chapter 8, pp296, CRC Press,1996.

## Appendix A: The Result of Our Attacks

We use $\rho_{ATTACKI}$ (resp. $\rho_{ATTACKII}$) to denote the specific value of $\rho$ for ATTACK I (resp. ATTACK II). We define $\lambda_{ATTACKI}$ (resp. $\lambda_{ATTACKII}$) to denote the specific value of leakage parameter $\lambda$ 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 |
| 1216 | 100.16% | 21.05% | 609 | 255 | 19 |

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

| $\|p\|$ (in bits) | $\rho_{ATTACKI}$ | $\rho_{ATTACKII}$ | $\lambda_{ATTACKI}$ | $\lambda_{ATTACKII}$ | $v$ (times) |
|---|---|---|---|---|---|
| 644 | 100.31% | 25.16% | 323 | 161 | 5 |
| 800 | 100.25% | 20.25% | 401 | 161 | 5 |
| 960 | 100.21% | 16.88% | 481 | 161 | 6 |
| 1120 | 100.18% | 14.46% | 561 | 161 | 7 |
| 1280 | 100.16% | 12.66% | 641 | 161 | 8 |

**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) |
|---|---|---|---|---|---|
| 516 | 100.39% | 25.19% | 259 | 129 | 5 |
| 640 | 100.31% | 20.31% | 321 | 129 | 5 |
| 768 | 100.26% | 16.93% | 385 | 129 | 6 |
| 896 | 100.22% | 14.51% | 449 | 129 | 7 |
| 1024 | 100.20% | 12.70% | 513 | 129 | 8 |
| 1152 | 100.17% | 11.28% | 577 | 129 | 9 |
| 1280 | 100.16% | 10.16% | 641 | 129 | 10 |

## Appendix B: The Attack Processes

We show our attack processes in the following.

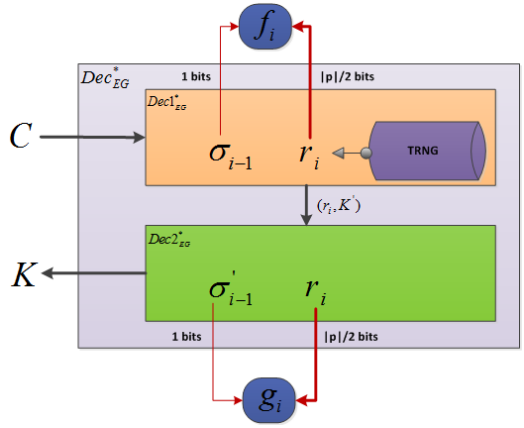


**Fig. 4.** our attack on decapsulation of EG* with TRNG

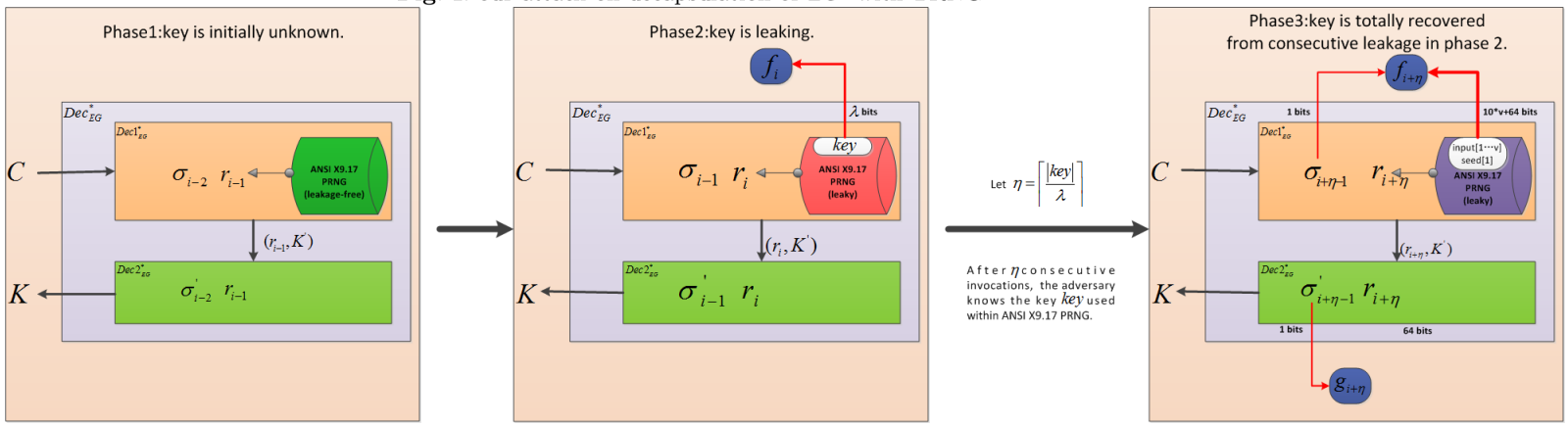


**Fig. 5.** our attack on decapsulation of EG* with a leaky ANSI X9.17 PRNG