On (Destructive) Impacts of Mathematical Realizations over the Security of Leakage Resilient ElGamal Encryption

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

[1] Institute of Software,Chinese Academy of Sciences,Beijing,China
`guangjunfan@163.com` , `feng@is.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. Leakage resilient cryptography aims to address the issue of inadvertent and unexpected information leakages from physical cryptographic implementations. At Asiacrypt 2010, E.Kiltz et al. [1] presented a *multiplicatively blinded* version of ElGamal public-key encryption scheme, which is proved to be leakage resilient in the generic group model against roughly ***0.50**log(p)** bits of arbitrary, adversarially chosen information leakage about the computation, when the scheme is instantiated over bilinear groups of prime order p (denoted BEG^*). Nonetheless, for the same scheme instantiated over arbitrary groups of prime order p (denoted EG^*), no leakage resilience bound is given, and was only conjectured to be leakage resilient. In this paper, we show that, when some of the leakage happens within the computation of pseudo random number generator (PRNG) used by EG^*, the leakage tolerance of EG^* is far worse than expected. We used three instances of internationally standardized PRNGs to analyze the leakage resilience of different mathematical realizations of EG^*, namely ANSI X9.17 PRNG, ANSI X9.31 PRNG using AES-128, and FIPS 186 PRNG for DSA premessage secrets, respectively. For ANSI X9.17 PRNG and ANSI X9.31 PRNG using AES-128 (resp. DSA PRNG) considered, when the size of p is 1024 bits (resp. 1120 bits), one can successfully recover the long-term secret key x if he learns only ***0.2988**log(p)** and ***0.2832**log(p)** (resp. ***0.2929**log(p)**) bits of leakages of the computation respectively. This shows that mathematical realizations of EG^* can have significant impacts on its leakage resilience. In addition, by presenting *non-generic* attacks, this paper also gives some upper bounds of the amount of leakages that these mathematical realizations of EG^* can tolerate, and these upper bounds are the best known so far.

Keywords: Leakage Resilient Cryptography, ElGamal Encryption, Mathematical Realization, PRNG, Lattice.

1 Introduction

Side-channel attacks belong to an important kind of cryptanalysis techniques on cryptographic implementations. As a matter of fact, many implementations of traditional cryptosystems even provably secure in black-box model were broken by side-channel attacks using electromagnetic radiation [3,7], running-time [4], fault detection [5], power consumption [6] and many more [23,24].

Broadly speaking, countermeasures for protecting against side-channel attacks are taken on two complementing levels: the hardware level and the software level. For example, *hiding* and *masking* are two typical ones used to defend power analysis attacks on both two levels. However, the main problem of the countermeasures based on hardware is that the protection against all possible types of leakages is very hard to achieve [25], if not impossible. On the other hand, most (even not all) software-based approaches proposed so far are only heuristic, and lack of any formal security proofs. Furthermore, they are ad-hoc, which means that they protect only against some specific attacks known at the moment, instead of providing security against a large well-defined class of attacks.

In order to solve these pressing issues, S.Dziembowski et al. [8] proposed one general and theoretical methodology called Leakage Resilient Cryptography (LRC). LRC has a similar goal of reasoning about side-channel attacks, but in an abstract and theoretical manner. The goal of LRC is to research a systematic method of designing cryptographic schemes so that already their mathematical description guarantees that they are provably secure, even if they are implemented on hardware that may be subject to any specific side-channel attack which belongs to a large well-defined class of such attacks. Actually, in their pioneering work, S.Micali et al. [9] in 2004 already put forward a powerful and comprehensive framework for modeling security against side-channel attacks. Their framework captures any such attack in which leakage of information occurs as a result of computation. The framework relies on the basic assumption that "*Only Computation Leaks information*" (OCL). This model assumes that there is no leakage of information in the absence of computation.

Mathematical Realization at algorithmic level refers to a process in which any generic cryptographic construction is being transformed into one specific cryptographic scheme. For example, given any One-Way Function (OWF), it is possible to construct private-key encryption schemes secure against adaptive chosen-ciphertext attack. Furthermore, there are several known candidates of OWF, such as multiplication and factoring, Rabin function (modular squaring), discrete exponential and logarithm, to name just a few. Consequently, in order to establish a specific private-key encryption, one should choose any one of these OWF candidates to complete the corresponding mathematical realization. In this way, once one OWF candidate is chosen, such a specific cryptographic scheme is then constructed. On the other hand, *Physical Realization* at device level refers to a process in which any specific cryptographic scheme is being transformed into a physical cryptographical module that runs as a piece of software, or hardware, or combination of both. Consequently, the output of mathematical realization of any generic cryptographic construction at algorithmic level is a certain mathematical object, while that of physical realization at a device level is an physical object that runs over some specific hardware (e.g. an Intel 32-bit CPU). Broadly, and also more importantly, it has been turned out that physical realizations have significant impact on the physical security of traditional cryptographic schemes, so it is with those of any leakage resilient ones [39]. In this paper, we consider mathematical realization, not physical realization. That is to say, our attacks are regardless of any specific side-channel attacks.

At Asiacrypt 2010, E.Kiltz et al. presented a multiplicative blinded version of ElGamal public-key encryption scheme [1] in the OCL model. Actually, in the context of leakage resilience, the core method of [1] (or variants thereof) were already proposed in [26,27,28]. The central idea is to use *multiplicative secret sharing* to share the secret key x. The secret key $x \in \mathbb{Z}_p^*$ is shared as $\sigma_i = xR_i \bmod (p)$ and $\sigma_i' = R_i^{-1} \bmod (p)$, for some random $R_i \in \mathbb{Z}_p^*$ where p is a strong prime. In the definition of leakage resilience in [1], an invocation of this scheme (which corresponds to a decryption query) is split into two phases, and those two consecutive phases leak individually. Denote these two phases by $Dec1^*$ and $Dec2^*$, which are executed in a sequential order to decrypt the message. Accordingly, two arbitrary (efficiently computable) leakage functions specified by the adversary, f and g, whose range are bounded by λ bits, are associated with $Dec1^*$ and $Dec2^*$ respectively. λ is the leakage parameter.

The scheme, instantiated over bilinear groups of prime order p (where $p-1$ is not smooth) (denoted BEG^*) is proved to be leakage resilient in the generic-group model. Specifically, BEG^* remains chosen-ciphertext secure even if with every decryption query the adversary can learn a bounded amount (roughly **0.50***log(p) bits) of arbitrary, adversary chosen information about the computation. Therefore, for BEG^* to be leakage resilient, the leakage parameter $\lambda =$ **0.25***log(p). On the other hand, the same scheme, when instantiated over arbitrary groups of prime order p (where $p-1$ is not smooth) (denoted EG^*) is just conjectured to be leakage resilient. Interestingly, the amount of leakage that EG^* can tolerate was not specified. Actually, E.Kiltz et al. pointed it out that there exists some attack on EG^* if $\lambda =$ **0.40***log(p), using the method presented in [31]. Apparently, this loose bound is overly conservative. Till so far, no more tighter bound other than $\lambda =$ **0.40***log(p) is evidently known.

Motivations Under traditional black-box model (i.e. leakage-free setting), the security proof of any cryptographic scheme generally work independently of mathematical realization. This means that any mathematical realization of the whole cryptographic scheme would remain secure when it is instantiated with any specific cryptographic components (e.g., PRNG in this paper), provided that the component chosen meets the required cryptographic properties. On the other hand, according to the goal of leakage resilient cryptography, specifying all the details of an implementation is tedious and it is not ***clear*** if it is feasible at all to achieve the original goal of LRC with considering any detail at higher abstraction level (e.g. the mathematical realization at algorithmic level). So even from more practical point of view, working at higher abstraction level seems appealing. The scheme BEG^* in [1] is leakage resilient in generic group model. The proof of the scheme BEG^* has its obvious weaknesses because the generic group model cannot be implemented. In particular in connection with side channel attacks the generic group model may "abstract away" too much important information that an adversary may obtain in a real implementation of the scheme. Hence, the scheme BEG^* goes against our recommendation to at least provide mathematical realization. Consequently, the scheme EG^* in [1] serves as a case of the study about this problem. Another problem is that any non-generic attacks against EG^* is not known. The existing upper bound of leakage rate of EG^* is much too rough (See [36] for more details.), even not unspecified. To answer these important problems, we investigate in this work the amount of leakage that EG^* can tolerate when it is mathematically implemented, by presenting *non-generic* attacks on several mathematical realizations of EG^*.

1.1 Our Contributions

Main contributions of this paper are four-fold as follows.

[1] Under traditional black-box model (i.e. leakage-free setting), any realization of the whole cryptographic scheme would remain secure when it is instantiated with any specific cryptographic components (e.g., PRNG in this paper), provided that the component chosen meets the required cryptographic properties. Our research shows that this statement ***does not*** hold, on the contrary, under the leaky setting. Specifically, taking several mathematical realizations of one specific leakage resilient ElGamal encryption scheme EG^* as cases of study, this paper studies the destructive impacts of mathematical realization of leakage resilient scheme on its claimed theoretical security (i.e. leakage resilience).

[2] For any given leakage resilient cryptographic scheme, leakage rate reflects its expected theoretical security. Therefore, (accurate and/or rough) estimation of information leakage rate of any leakage resilient scheme ***does*** make very good sense. This paper specifies one upper bound of leakage that EG^* can tolerate when it is mathematically implemented or realized. This upper bound is the best known so far, even thought it might not be the tightest.

[3] In order to show the impacts of different mathematical realizations of underlying cryptographic component (i.e. PRNG in this paper) over the theoretical leakage rate of EG^* and also to show the validity of our methods as well, we take three internationally standardized PRNGs to mathematically instantiate EG^*. We implement the proposed non-generic attacks against these three instantiations, and comparatively analyze the results in depth. The analysis results firmly verify both the soundness and the validity of our proposed method.

[4] In order to enhance the resistance of one cryptographic scheme (e.g., EG^*) to any attacks against its underlying mathematically hard problem, it is always a rule of thumb to increase the size of security-critical parameters (e.g, the size of strong prime p) in traditional black-box cryptography. However, this could lead to the decline of leakages that EG^* can tolerate, which is certainly undesirable in practice. Our results show that this commonly-used methodology

might cause some leakage resilient schemes to tolerate less leakage when they are implemented using some specific cryptography components.

1.2 Related Work

In recent years, in the field of LRC, several different kinds of leakage models have been proposed as of today. For example, Only Computation Leaks Model [8,9,37,38], Memory Attacks [10,11,12,13], Bounded Retrieval Model [14,15,16,17,18,19] and Continuous Memory Attacks [20,21,22]. As far as actual side-channel attacks are concerned, the most representative leakage model is the Only Computation Leaks Model (OCL) due to it considers continuous leakage. OCL is the start point of the line of the research. When the adversary is more powerful (as that in continuous memory attack model), whether or not our conclusions of this paper are still hold is a valuable research point.

1.3 Organization of This Paper

The rest of paper is organized as follows. In Section 2, we present the scheme EG^* in [1] and some basic concepts. Section 3 describes our two attack methods against EG^*. The process of building the system of linear congruence equations about the multiplicative secret sharing of the secret key x is presented in this section. We specify one the upper bounds of the scheme EG^* according to our two attack methods in this section. Our analyses are supported by experiments in Section 4. Section 5 concludes the whole paper.

2 Preliminaries

In this section, we will first briefly recall the scheme EG^*. Next, we will present some basic knowledge about lattice theory on which our attacks are based. We then will present some symbols and notations used throughout the paper in the end of this section.

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

2.1 Brief Description of EG^*

We describe the scheme EG^* in the same way as that in [1]. The scheme EG^* is described as a Key Encapsulation Mechanism (KEM) and is based on the assumption that "*Only Computation leaks information*".

The decapsulation algorithm of $EG^* = (KG^*_{EG}, Enc^*_{EG}, Dec1^*_{EG}, Dec2^*_{EG})$ is stateful and formally split into two sequential stages $Dec^*_{EG} = (Dec1^*_{EG}, Dec2^*_{EG})$. Gen is a probabilistic algorithm that outputs a cyclic group $\mathbb{G}$ of order p, where p is a strong prime.

The scheme EG^* is as follows:

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

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

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

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

A KEM achieves CCLA1 (Chosen Ciphertext with Leakage Attack) if for any probabilistic polynomial time adversary $\mathcal{F}$, $Adv^{ccla}_{KEM}(\mathcal{F}, n, \lambda) = 2|1/2 - \mu|$ is negligible in n, where μ is the probability that the output b' of the following experiment is equal to b, n is the security parameter and $\lambda \in \mathbb{N}$ is the leakage parameter.

Experiment $\mathbf{Exp}^{\mathtt{ccla}}_{\mathtt{KEM}}(\mathcal{F}, \kappa, \lambda)$	**Oracle** $\mathcal{O}^{\mathtt{ccla1}}(C, f_i, g_i)$
$(pk, \sigma_0, \sigma'_0) \xleftarrow{*} KG(\kappa)$	If $\lvert f_i \rvert > \lambda$ or $\lvert g_i \rvert > \lambda$ return $\perp$
$\omega \xleftarrow{*} \mathcal{F}^{\mathcal{O}^{\mathtt{ccla1}}}(pk)$	$i \leftarrow i + 1$
$b \xleftarrow{*} \{0, 1\}$	$(\sigma_i, \omega_i) \xleftarrow{r_i} Dec1^*(\sigma_{i-1}, C)$
$(C^*, K_0) \xleftarrow{*} Enc(pk)$	$(\sigma'_i, K_i) \xleftarrow{r'_i} Dec2^*(\sigma'_{i-1}, \omega_i)$
$K_1 \xleftarrow{*} \mathcal{K}$	$\Lambda_i \leftarrow f_i(\sigma_{i-1}, r_i)$
$i \leftarrow 0$	$\Lambda'_i \leftarrow g_i(\sigma'_{i-1}, \omega_i, r'_i)$
$b' \xleftarrow{*} \mathcal{F}(\omega, C^*, K_b)$	return$(K_i, \Lambda_i, \Lambda'_i)$

On the i^{th} invocation of decapsulation, the decapsulated key K_i is computed as follows

$$(\sigma_i, \omega_i) \xleftarrow{r_i} Dec1^*(\sigma_{i-1}, C) \quad (\sigma'_i, K_i) \xleftarrow{r'_i} Dec2^*(\sigma'_{i-1}, \omega_i),$$

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

In the security definition of CCLA1, after the i^{th} querying the oracle $\mathcal{O}^{\mathtt{ccla1}}$, the adversary gets not only K_i, but also the leaked information $\Lambda_i = f_i(\sigma_{i-1}, r_i)$ and $\Lambda'_i = g_i(\sigma'_{i-1}, \omega_i, r'_i)$. The leakage function f_i and g_i are efficient computable functions chosen by adversary and get as input only the secret state that is actually accessed during the invocation. The range of f_i and g_i are bounded by the leakage parameter λ. For the scheme EG^*, the leakage functions f_i and g_i are as follows:

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

The authors of [1] didn't prove the security of EG^*, instead they presented the following conjecture.

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

In [36], the authors of [1] conjectured that roughly $\lambda = 0.25 \cdot logp$ bits. Thus the total leakage bits of one decryption query are $2\lambda = 0.5 \cdot logp$ bits.

2.2 Basics of Lattice Theory

We now give a brief introduction into basic terms of lattice theory on which our attacks are based.

$\mathbb{R}^m$ denotes the m-dimensional real Euclidean vector space and e_i the i^{th} unit vector in $\mathbb{R}^m$. For any vector $v \in \mathbb{R}^m$, $\parallel v \parallel = (\Sigma_{i=1}^m v_i^2)^{1/2}$ is the Euclidean norm. A lattice L is a discrete additive subgroup of the $\mathbb{R}^m$ with

$$L = \{y \in \mathbb{R}^m \mid \ y = a_1 b_1 + \cdots + a_k b_k,\ a_i \in \mathbb{Z}\}$$

where $b_1, \ldots, b_k \in \mathbb{R}^m$ linear independently over $\mathbb{R}^m$ and $k \leq m$. $\{b_1, \ldots, b_k\}$ is called a basis of the lattice L. The i^{th} successive minimum $\lambda_i(L)$ of a lattice L is the smallest positive real number z, such that there exists i linear independent vectors $l_1, \ldots, l_i \in L$ of maximum length z, i.e.

$$\lambda_i(L) = min_{l.u.l_1,\ldots,l_i \in L} max_{j \in \{1,\ldots,i\}} \parallel l_j \parallel .$$

2.3 Symbols and Notations

We define some main symbols and notations in this subsection, while some other will be defined in more appropriate position in the following sections. We use the following notational conventions.

If S is a binary bit string, denote $S^{[a]}$ the most significant a bits of S, and denote $S_{[b]}$ the least significant b bits of S. $|S|$ denotes the length of S. $abs(a)$ denotes the absolute value of a when a is a numerical value. If M represents a matrix, then $det(M)$ is the determinant of M and $M^\top$ is the transpose of M. We assume that the representation for all elements belonging to $\mathbb{Z}_p$ has the same length of binary bit string.

In order to simplify the notation, we ignore the subscript of leakage function f and g. t denotes the number of leakage bits about σ_i and σ'_i from leakage function f and g in one invocation of the decryption query. In Section 3, we can see that the leakage bits are the most significant t bits of σ_i and σ'_i.

The leakage information from leakage function f and g can be divided into two parts, one part is about the multiplicative shares $\sigma_{i-1}, \sigma'_{i-1}$ and the other part is about the randomness r_i. For leakage function f, we use $\mu_{\sigma f}$ to denote the leakage bit number about σ_{i-1} and use μ_{rf} to denote the leakage bit number about r_i leaked from f. $\mu_{\sigma' g}$ and μ_{rg} have the similar meaning for leakage function g. Therefore, we have $|f| = \mu_{\sigma f} + \mu_{rf}$ and $|g| = \mu_{\sigma' g} + \mu_{rg}$.

Due to we present two attack methods in this paper, we use $\rho_{ATTACKI} = \frac{|f|+|g|}{|p|}$ to denote the leakage rate of the whole invocation for the first attack method and use $\rho_{ATTACKII} = \frac{|f|+|g|}{|p|}$ to denote the leakage rate of the whole invocation for the second attack method.

We define $\lambda_{ATTACKI} = max\{|f|, |g|\}$ and $\lambda_{ATTACKII} = max\{|f|, |g|\}$ for our two attack methods respectively.

3 Our Non-Generic Attacks on Mathematical Realizations of EG^*

In this section, we first introduce the overview of our attacks, and then present the details of them.

3.1 Overview of Our Attacks

The goal of our non-generic attacks is to recover the secret key x. To achieve this goal, we try to build two systems of linear congruence equations about the multiplicative secret shares σ_i and σ'_i respectively from leakage information. For this purpose, we need to continually invoke the decryption query dozens of times and get all the bits of the randomness r_i and few bits about σ_i and σ'_i for each invocation.

If the adversary has enough leakage bits about the multiplicative shares σ_i and σ'_i for each invocation, by lattice theory and related analysis techniques, the systems of linear congruence equations have unique solution and the unique solution can be returned by an algorithm in polynomial time with very high probability. When the adversary gets all the bits of σ_i and σ'_i, he can recover a candidate value x' ($x' = \sigma_i\sigma'_i \bmod (p)$) of the secret key x.

In our basic attack method, we treat the underlying PRNG generating the randomness r_i of every invocation as a black box. In this way, the leakage functions f_i and g_i leak half bits about the randomness r_i respectively. Therefore, the minimum value of λ which the adversary needs to recover the secret key successfully is apparently larger than $0.5 * log(p)$ (because some other bits of information about the multiplicative secret shares also need to be leaked thorough leakage functions). In this case, the number of leaked bits per invocation of the decryption query of scheme EG^* is larger than $log(p)$. However, our second attack method shows that the minimum value of λ and the number of leakage bits per decryption query will decrease dramatically when we consider the mathematical structure of some specific PRNGs which are used to generate the randomness in the implementation of EG^*. The first attack is the basis of the second attack. Both attacks are based on the lattice theory. In Section 3.2, we describe the first attack method and in Section 3.3, the second attack method will be presented.

3.2 ATTACK I: Basic Attack Knowing Nothing about the Mathematical Structure of Underlying PRNG

Our basic attack method (ATTACK I) is as follows:

In every invocation of the decryption query of EG^*, the adversary, in ***one*** decryption query, gets some most significant few bits of σ_i and σ'_i and all bits of r_{i+1} through leakage functions f_{i+1} and g_{i+1}. Furthermore, he can get r_{i+1}^{-1} from r_{i+1} easily (Because p is a prime and also is public.). By continual invocations (e.g. n times), the adversary can build two systems of linear congruence equations about the rest of unknown bits of $\{\sigma_i, \sigma_{i+1}, \ldots, \sigma_{i+n-1}\}$ and $\{\sigma'_i, \sigma'_{i+1}, \ldots, \sigma'_{i+n-1}\}$ respectively. By solving the two systems of congruence equations using lattice theory, the adversary can recover a candidate value of the secret key.

In the $(i+1)^{th}$ decryption query of EG^*, the adversary obtains $\sigma_i^{[t]}$ (We will show the specific values of t for different size of p below.) and $r_{i+1}^{[|p|/2]}$ simultaneously from $f_{i+1}(\sigma_i, r_{i+1})$ of the decryption query. He also gets $\sigma_i'^{[t]}$ and $r_{i+1[|p|/2]}$ simultaneously from $g_{i+1}(\sigma'_i, (r_{i+1}, K'), r_{i+1}^{-1})$ of the decryption query. In this case, the leakage functions are defined to be:

$$f_{i+1}(\sigma_i, r_{i+1}) = \langle \sigma_i^{[t]}, r_{i+1}^{[|p|/2]} \rangle$$

$$g_{i+1}(\sigma'_i, (r_{i+1}, K'), r_{i+1}^{-1}) = \langle \sigma_i'^{[t]}, r_{i+1[|p|/2]} \rangle$$

Figure 1 shows the attack process.

In the $(i+2)^{th}$ decryption query, the adversary is able to get $\sigma_{i+1}^{[t]}$ and $\sigma_{i+1}'^{[t]}$ and the whole value of r_{i+2} similarly. At this point, the adversary knows r_{i+1} and r_{i+2}, $\sigma_i^{[t]}$,$\sigma_{i+1}^{[t]}$,$\sigma_i'^{[t]}$ and $\sigma_{i+1}'^{[t]}$. σ_i can be rewritten as

$$\sigma_i = \sigma_i^H + \sigma_i^L,$$

where the σ_i^H is equal to $\sigma_i^{[t]} 2^{|p|-t}$, $\sigma_i^L \leq p2^{-t}$. Similarly, σ_{i+1} can be rewritten as

$$\sigma_{i+1} = \sigma_{i+1}^H + \sigma_{i+1}^L.$$

Thus, the adversary gets the following congruence equation:

$$\sigma_i^L r_{i+1} - \sigma_{i+1}^L \equiv \sigma_{i+1}^H - \sigma_i^H r_{i+1} \bmod (p).$$

In a similar way, $n-1$ congruence equations can be obtained from n continual invocations of the decryption query as follows:

$$\sigma_i^L r_{i+1} - \sigma_{i+1}^L \equiv \sigma_{i+1}^H - \sigma_i^H r_{i+1} \bmod (p)$$

$$\sigma_{i+1}^L r_{i+2} - \sigma_{i+2}^L \equiv \sigma_{i+2}^H - \sigma_{i+1}^H r_{i+2} \bmod (p)$$

$$\ldots\ldots$$

$$\sigma_{i+n-2}^L r_{i+n-1} - \sigma_{i+n-1}^L \equiv \sigma_{i+n-1}^H - \sigma_{i+n-2}^H r_{i+n-1} \bmod (p).$$

The leakage functions are defined to be:

$$f_{i+u}(\sigma_{i+u-1}, r_{i+u}) = \langle \sigma_{i+u-1}^{[t]}, r_{i+u}^{[|p|/2]} \rangle$$

$$g_{i+u}(\sigma'_{i+u-1}, (r_{i+u}, K'), r_{i+u}^{-1}) = \langle \sigma_{i+u-1}'^{[t]}, r_{i+u[|p|/2]} \rangle,$$

for $u = 1, \ldots, n-1$, and

$$f_{i+n}(\sigma_{i+n-1}, r_{i+n}) = \langle \sigma_{i+n-1}^{[t]} \rangle$$

$$g_{i+n}(\sigma'_{i+n-1}, (r_{i+n}, K'), r_{i+n}^{-1}) = \langle \sigma'^{[t]}_{i+n-1} \rangle.$$

We denote $d_1 = \sigma_i^L, \ldots, d_n = \sigma_{i+n-1}^L$, $\beta_2 = r_{i+1}, \ldots, \beta_n = r_{i+n-1}$ and $c_1 = \sigma_{i+1}^H - \sigma_i^H r_{i+1}, \ldots, c_{n-1} = \sigma_{i+n-1}^H - \sigma_{i+n-2}^H r_{i+n-1}$, where $\{d_1, \ldots, d_n\}$ are all unknown, $\{\beta_2, \ldots, \beta_n\}$ and $\{c_1, \ldots, c_{n-1}\}$ are all known. The adversary can obtain the following $n-1$ congruence equations with n unknown quantity.

$$\begin{cases} d_1\beta_2 - d_2 \equiv c_1 \bmod (p) \\ d_2\beta_3 - d_3 \equiv c_2 \bmod (p) \\ \qquad \ldots\ldots \\ d_{n-1}\beta_n - d_n \equiv c_{n-1} \bmod (p) \end{cases} \tag{1}$$

In order to solve the above system of linear congruence equations, the adversary can use the following **Theorem 1** in [2].

Theorem 1. *Let*

$$\sum_{j=1}^{n} b_{ij} d_j \equiv c_i \bmod (p)$$

a system with $b_{ij}, c_i \in \mathbb{Z}$, $i = 1, \ldots, s$, p is a prime and $s \le n$,

$$L = \left\{ y \in \mathbb{R}^n \,\middle|\, y = \sum_{i=1}^{s} a_i (b_{i1}, \ldots, b_{in})^\top + a_{s+1} p e_1 + \cdots + a_{s+n} p e_n,\ a_i \in \mathbb{Z} \right\}$$

a lattice in $\mathbb{R}^n$ satisfying $\|d\| \le p\lambda_n(L)^{-1}2^{-1}$, then there exists at most one solution $d = (d_1, d_2, ..., d_n)$ for this system. If the b_{ij}, c_i and p are all known for all i, j, then there exists an algorithm which computes for fixed n in polynomial time the solution d or proves that there is no solution.

The algorithm computes n linearly independent vectors $w_1, w_2, \ldots, w_n$ (successive minima) for the given lattice L, where $\|w_i\| = \lambda_i(L), (i = 1, 2, \ldots, n)$. Let matrix $W = (w_1, w_2, \ldots, w_n)$ and $det(W^\top) \ne 0$. The coefficient matrix of the system of linear congruence equations (1) can be transformed to W. Therefore, if the adversary can get matrix W successfully, he will solve (1). The secret key x can be recovered from the solution of (1) with high probability. The details of the algorithm are given in Appendix A.

The applicability of Theorem 1 requires that $abs(d_i) \le p\lambda_n(L)^{-1}2^{-1}n^{-1/2}$ and $d_i \le p2^{-t}$. For unknown d_i, this means that one needs to know the most significant $t = log\lambda_n(L) + \frac{1}{2}logn + 1$ bits of every σ_i. Therefore, the number t of known bits in advance only depends on $\lambda_n(L)$.

Similarly to [2], by Theorem 2, we could estimate the value of $\lambda_n(L)$.

Theorem 2. *Let p be a prime, $\epsilon > 0$ and*

$$L = \{y \in \mathbb{R}^n |\ y = \mathbb{Z}b_1 + \cdots + \mathbb{Z}b_{n-1} + \mathbb{Z}pe_1 + \cdots + \mathbb{Z}pe_n\}.$$

A lattice in $\mathbb{Z}^n$, where $b_1 = (r_2, -1, 0, \ldots, 0), b_2 = (0, r_3, -1, 0, \ldots, 0), \ldots, b_{n-1} = (0, \ldots, 0, r_n, -1)$ are randomly chosen in $\mathbb{Z}^n$. Then, with probability$\geq 1 - \epsilon - O(1/p^{(n-1)/n})$ it holds that

$$\lambda_n(L) \leq \left(\frac{\pi^{n/2}}{\Gamma(\frac{n}{2}+1)} \right)^{1/n} n\epsilon^{-1/n} p^{1-(n-1)/n}.$$

Furthermore, we can get the lower bound of t

$$t \geq \frac{1}{n} log_2 p + log_2 n + \frac{1}{n} log_2 \epsilon + 3.06 = t'. \quad (3)$$

Denote t_{min} the minimum value of t ($t_{min} = \lceil t' \rceil$). The adversary could get r_u^{-1} from r_u ($u = i+1, i+2, \ldots, i+n-1$) easily ($p$ is a public prime.). Therefore, knowing the value of $\sigma_u'^H$, ($u = i, i+1, \ldots, i+n-1$), the adversary could get the whole value of σ_i' in a similar way. Thus, a candidate value of secret key can be recovered by computing $x' = \sigma_i \sigma_i' \bmod (p)$. It is clearly that $x' = x$ if and only if $C^{x'} = K$ for a correct plaintext-ciphertext pair (C, K). Figure 2 shows the decapsulation algorithm of EG^* and where leakages take place.

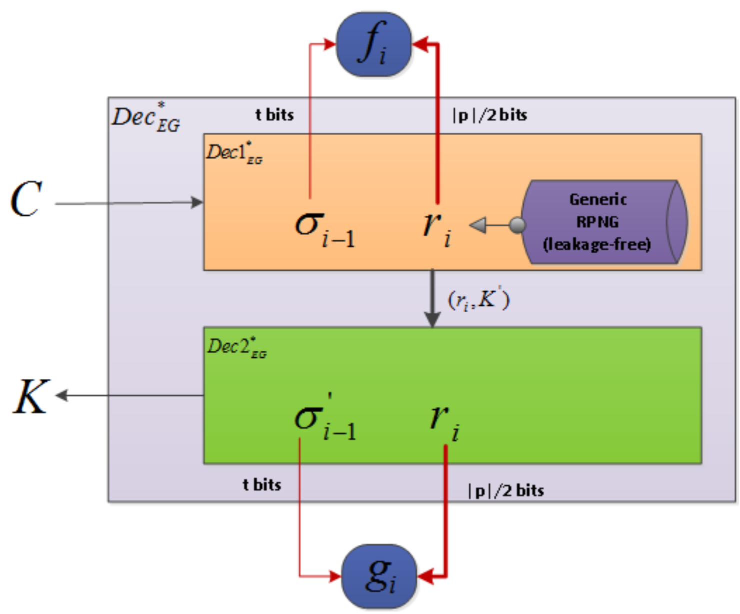

Fig. 1. Our attack on decapsulation of EG^* with a generic and leakage-free PRNG

For different size of prime p and different number of congruence equations (Denoted by $\#_{equ}$, which means the adversary will consecutively invoke the decryption query $\#_{equ}+1$ times), we show the percentage of $t_{min}/|p|$ in Table 1 and the value of t_{min} in Figure 2.

Table 1. Percentage of $t_{min}/|p|$ for different size of strong prime p

$\#_{equ}$ \ $\|p\|$	160	256	512	1024
5	20%	18.75%	17.58%	17.18%
10	13.13%	11.72%	10.35%	9.67%
20	9.38%	7.81%	6.25%	5.47%
30	8.13%	6.64%	4.88%	4%
40	8.13%	5.86%	4.10%	3.32%
50	7.5%	5.47%	3.71%	2.83%

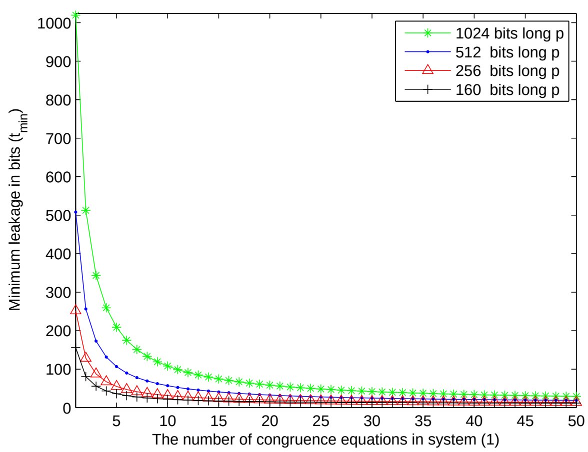

Fig. 2. Relationship between the number of equations and t_{min} for strong prime p of different sizes

Therefore, if $\lambda = t_{min} + |p|/2$, the adversary will recover the secret key x. By Table 1, we can see that if the adversary has 30 equations, the percentage of $\lambda/|p|$ equals to 58.13% ($\lambda = 13 + 80 = 93$ bits) for 160 bits strong primes. For 1024 bits strong primes, the percentage of $\lambda/|p|$ equals to 54% ($\lambda = 41 + 512 = 553$ bits). Thus the number of leakage bits required to execute a successful attack per invocation of the decryption query is larger than $log(p)$ bits. Although this attack poses virtually no threat to its theoretical security, Table 1 and Figure 2 (The different strong primes we use in our analysis are listed in Appendix B) show that for the same number of equations, an increase of the size of the strong prime result in an decrease of the percentage of $t_{min}/|p|$.

Another interesing point of Table 1 is that if the adversary get the whole bits of the randomness r_{u+1}, then he only need very few bits about σ_u and σ'_u, $(u = i, i+1, \ldots, i+n-1)$ to execute a successful attack. If leakage bit number about $r_{u+1},\ (u = i, i+1, \ldots, i+n-1)$ can decrease, the tolerance leakage rate of the scheme EG^* will decrease. In most cases of practices, $r_{u+1},\ (u = i, i+1, \ldots, i+n-1)$ are always generated by PRNGs. Therefore, if the adversary knows the concrete mathematical structure of the underlying PRNGs generating $r_{u+1},\ (u = i, i+1, \ldots, i+n-1)$, he may carry out a successful attack with less leaked bits. This forms the basis of our second attack method.

3.3 ATTACK II: Attack Knowing Only the Mathematical Structure of the Underlying PRNG (Without Any Implementation Aspects)

Our second attack method (ATTACK II) also try to build the same system of linear congruence equations as our first attack method does. However, the difference is that the second attack method considers the concrete mathematical structure of some widely used PRNGs. When the algorithm $Dec1^*_{EG}$ invokes one PRNG to generate the randomness r_i in the decryption query, the internal secret states of the underlying PRNG could be leaked due to the assumption that *"Only Computation Leaks information"*.

Our second attack shows that if EG^* is implemented by some PRNG to generate r_i, to execute a successful attack, the necessary percentage of $\lambda/|p| = max\{|f|, |g|\}/|p|$ will drop to below 25% for these PRNGs. We first assume the adversary will build a system of linear congruence equations which has 49 equations in both methods ([2] built such a system of congruence equations and solve it successfully in practice.). However, due to our experiment environment (See section 4 for more details.), we assume that the adversary will build the system of linear congruence equations which has 30 equations (This case can be solved successfully by our experiment environment.). This means that the decryption query will be continually invoked for 31 times and there exists 31 unknown quantity in the system of the linear congruence equations.

If the decapsulation algorithm of EG^* continually invokes PRNG v times to generate r_i, then we will denote $r_i = (output[1]\|\ldots\|output[v])$. The output of each invocation of PRNG is denoted by $output[u]$, $(u = 1, 2, ..., v)$.

3.3.1 CASE 1: ANSI X9.17 PRNG

The ANSI X9.17 PRNG [29] has been used as a general purpose PRNG in many applications. Let E_k (resp. D_k) denotes DES E-D-E two-key triple-encryption (resp. decryption) under a key k. The k is generated somehow at initialization time. It must be reserved exclusively used only for this generator. It is part of the secret state of PRNG which is never changed by any PRNG input.

The random bits generation algorithm of ANSI X9.17 PRNG is as shown in Algorithm 2.

Algorithm 2 ANSI X9.17 PRNG

INPUT: a random (and secret) 64-bit seed $seed[1]$, integer v, and DES E-D-E triple-encryption with key k.

OUTPUT: v pseudorandom 64-bit strings $output[1], \ldots, output[v]$.

Step 1 For i from 1 to v do the following:

1.1 Compute the intermediate value $I_i = E_k(input[i])$, where $input[i]$ is a 64-bit representation of the date/time.

1.2 $output[i] = E_k(I_i \bigoplus seed[i])$

1.3 $seed[i+1] = E_k(I_i \bigoplus output[i])$

Step 2 Return $(output[1], output[2], \ldots, output[v])$

$input[i], (i = 1, 2, \ldots, v)$ is a 64-bit representation of the date/time. Suppose that each $input[i]$ value has 10 bits that aren't already known to the adversary (For simplicity, let these 10 bits be the least significant 10 bits of $input[i]$.). This is a reasonable assumption for many systems (as described in [33]). For example, consider a millisecond timer, and an adversary who knows the nearest second when an output was generated. Before doing the attack, due to the fact that k is never changed, the adversary can obtain k from leakage function f by invoking the decryption query repeatedly. On knowing k, the adversary could continually queries the oracle $\mathcal{O}^{ccla1}$ for n times and the leakage functions are defined to be as follows:

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

$$g_{i+u}(\sigma'^{[t]}_{i+u-1}, (r_{i+u}, K'), r^{-1}_{i+u}) = \langle \sigma'^{[t]}_{i+u-1}, output[1]_{i+u} \rangle$$

$u = 1, \ldots, n-1$ and

$$f_{i+n}(\sigma_{i+n-1}, r_{i+n}) = \langle \sigma^{[t]}_{i+n-1} \rangle$$

$$g_{i+n}(\sigma'_{i+n-1}, (r_{i+n}, K'), r^{-1}_{i+n}) = \langle \sigma'^{[t]}_{i+n-1} \rangle.$$

For the first $n-1$ invocations, the adversary knows $\{input[1], \ldots, input[v]\}$, and can compute $seed[1] = D_k(output[1]) \oplus E_k(input[1])$. Then the adversary can easily get $seed[u] = E_k(E_k(input[u-1]) \oplus output[u-1])$ as well as $output[u] = E_k(E_k(input[u]) \oplus seed[u])$, $(u = 2, 3, \ldots, v)$. Note that, to simplify description, we omit the subscript in notations here. At this point, the adversary get $\sigma^{[t]}_{i+u-1}, \sigma'^{[t]}_{i+u-1}, r_{i+u},\ (u = 1, 2, \ldots, n-1)$ and $\sigma^{[t]}_{i+n-1}, \sigma'^{[t]}_{i+n-1}$. In this way, the adversary can build the systems of linear congruence equations as that in ATTACK I, and then recover the secret key x. Figure 3 shows the attack process.

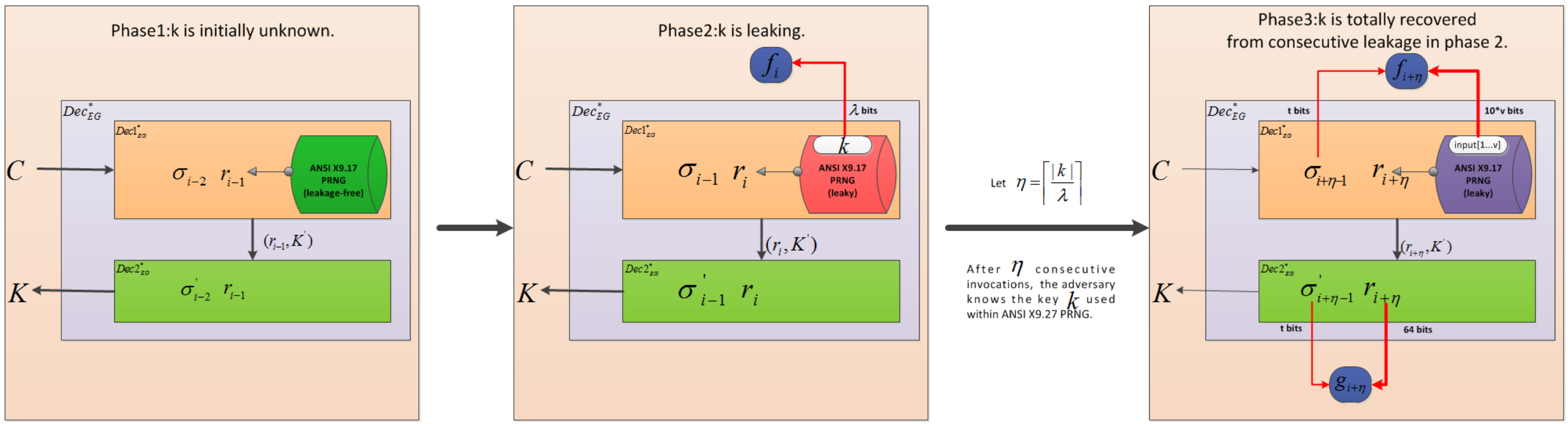

Fig. 3. Our attack on decapsulation of EG^* with a leaky ANSI X9.17 PRNG

Table 2.1.1 and Table 2.1.2 show the leakage bit number about leakage function f for two different parts for our two attack methods and different size of strong primes (Table 2.1.1 shows the leakage bit number in 49 equations case. Table 2.1.2 shows the leakage bit number in 30 equations case.). Table 2.2.1 and Table 2.2.2 show the same content about leakage function g.

Table 3.1 (resp. Table 3.2) shows the percentage of $\rho_{\{ATTACKI,ATTACKII\}}$ and the specific value of $\lambda_{\{ATTACKI,ATTACKII\}}$ for ANSI X9.17 PRNG of our two attacks when the number of equations is 49 (resp. 30).

Table 2.1.1. The specific leakage bit number for ANSI X9.17 PRNG of leakage function f in 49 equations case

$\lvert p\rvert$	Attacks	$\mu_{\sigma f}$	μ_{rf}	$\lvert f\rvert$
448 bits	ATTACK I	18	224	242
	ATTACK II	18	70	88
512 bits	ATTACK I	19	256	275
	ATTACK II	19	80	99
640 bits	ATTACK I	22	320	342
	ATTACK II	22	100	122
768 bits	ATTACK I	24	384	408
	ATTACK II	24	120	144
896 bits	ATTACK I	27	448	475
	ATTACK II	27	140	167
1024 bits	ATTACK I	30	512	542
	ATTACK II	30	160	190

Table 2.1.2. The specific leakage bit number for ANSI X9.17 PRNG of leakage function f in 30 equations case

$\lvert p\rvert$	Attacks	$\mu_{\sigma f}$	μ_{rf}	$\lvert f\rvert$
448 bits	ATTACK I	23	224	247
	ATTACK II	23	70	93
512 bits	ATTACK I	25	256	281
	ATTACK II	25	80	105
640 bits	ATTACK I	29	320	349
	ATTACK II	29	100	129
768 bits	ATTACK I	33	384	417
	ATTACK II	33	120	153
896 bits	ATTACK I	37	448	485
	ATTACK II	37	140	177
1024 bits	ATTACK I	41	512	553
	ATTACK II	41	160	201

Table 2.2.1. The specific leakage bit number for ANSI X9.17 PRNG of leakage function g in 49 equations case

$\|p\|$	Attacks	$\mu_{\sigma' g}$	μ_{rg}	$\|g\|$
448 bits	ATTACK I	18	224	242
	ATTACK II	18	64	82
512 bits	ATTACK I	19	256	275
	ATTACK II	19	64	83
640 bits	ATTACK I	22	320	342
	ATTACK II	22	64	86
768 bits	ATTACK I	24	384	408
	ATTACK II	24	64	88
896 bits	ATTACK I	27	448	475
	ATTACK II	27	64	91
1024 bits	ATTACK I	30	512	542
	ATTACK II	30	64	94

Table 2.2.2. The specific leakage bit number for ANSI X9.17 PRNG of leakage function g in 30 equations case

$\|p\|$	Attacks	$\mu_{\sigma' g}$	μ_{rg}	$\|g\|$
448 bits	ATTACK I	23	224	247
	ATTACK II	23	64	87
512 bits	ATTACK I	25	256	281
	ATTACK II	25	64	89
640 bits	ATTACK I	29	320	349
	ATTACK II	29	64	93
768 bits	ATTACK I	33	384	417
	ATTACK II	33	64	97
896 bits	ATTACK I	37	448	485
	ATTACK II	37	64	101
1024 bits	ATTACK I	41	512	553
	ATTACK II	41	64	105

Table 3.1. The percentage of $\rho_{\{ATTACKI, ATTACKII\}}$ and the specific value of $\lambda_{\{ATTACKI, ATTACKII\}}$ about ANSI X9.17 PRNG in 49 equations case

$\|p\|$ (in bits)	$\rho_{ATTACKI}$	$\rho_{ATTACKII}$	$\lambda_{ATTACKI}$	$\lambda_{ATTACKII}$	v (times)
448	108.04%	37.95%	242	88	7
512	107.42%	35.55%	275	99	8
640	106.88%	32.50%	342	122	10
768	106.25%	30.21%	408	144	12
896	106.03%	28.79%	475	167	14
1024	105.86%	27.73%	542	190	16

Table 3.2. The percentage of $\rho_{\{ATTACKI,ATTACKII\}}$ and the specific value of $\lambda_{\{ATTACKI,ATTACKII\}}$ about ANSI X9.17 PRNG in 30 equations case

$\|p\|$ (in bits)	$\rho_{ATTACKI}$	$\rho_{ATTACKII}$	$\lambda_{ATTACKI}$	$\lambda_{ATTACKII}$	v (times)
448	110.27%	40.18%	247	93	7
512	109.77%	37.89%	281	105	8
640	109.06%	34.69%	349	129	10
768	108.59%	32.55%	417	153	12
896	108.26%	31.03%	485	177	14
1024	108%	29.88%	553	201	16

From Table 3.1, Table 3.2 and Figure 6-9, it is clear that the scheme EG^* is not secure any more, if it uses ANSI X9.17 PRNG for strong primes $|p| \geq 448$ bits. For strong primes longer than 1024 bits, the tolerance leakage rate ρ is much lower, and thus we don't present all the data in the paper.

Note that ANSI X9.31-1998 Appendix A.2.4 in [32] also introduces PRNG using 3-key triple DES and AES Algorithms. In 3-key triple DES case, due to the fact that *input*[i], *seed*[i] and *output*[i] have the identical length as that of ANSI X9.17 PRNG, we could obtain the same attack results as those of the attack on ANSI X9.17 PRNG.

3.3.2 CASE 2: ANSI X9.31 PRNG Using AES-128

For the case of ANSI X9.31 PRNG, we only consider PRNG using AES-128 algorithm in this paper. Let E_k (resp. D_k) denotes the AES-128 encryption (resp. decryption) under a 128-bit key k. The random bits generation algorithm of ANSI X9.31 PRNG using AES-128 is the same as Algorithm 2, except that $input[i], seed[i]$ and $output[i]$ $(i = 1, 2, \ldots, v)$ are 128 bits each and E_k is the AES-128 encryption.

As in CASE 1, we assume that each *input*[i] has 10 bits entropy which the adversary doesn't know, and we also assume that these bits are the least significant 10 bits of $input[i], (i = 1, 2, \ldots, v)$. This assumption is also reasonable, as AES-128 is faster than 3DES.

After the adversary gets the 128-bits key k, he continually queries the oracle $\mathcal{O}^{ccla1}$ for n times and the leakage functions are also similar as those in CASE 1. They are defined to be as follows.

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

$$g_{i+u}(\sigma'^{[t]}_{i+u-1}, (r_{i+u}, K'), r^{-1}_{i+u}) = \langle \sigma'^{[t]}_{i+u-1}, output[1]_{i+u} \rangle,$$

$u = 1, \ldots, n-1$.

$$f_{i+n}(\sigma_{i+n-1}, r_{i+n}) = \langle \sigma^{[t]}_{i+n-1} \rangle$$

$$g_{i+n}(\sigma'_{i+n-1}, (r_{i+n}, K'), r^{-1}_{i+n}) = \langle \sigma'^{[t]}_{i+n-1} \rangle.$$

Through these leakage functions, the adversary can get $\sigma_{i+u-1}^{[t]}, \sigma_{i+u-1}^{\prime[t]}, r_{i+u}$, $(u = 1, 2, \ldots, n-1)$ and $\sigma_{i+n-1}^{[t]}$, $\sigma_{i+n-1}^{\prime[t]}$. Therefore, he can mount the attack. Similarly, we present results of the cases when the number of equations is 49 and 30 respectively. Figure 4 shows the attack process.

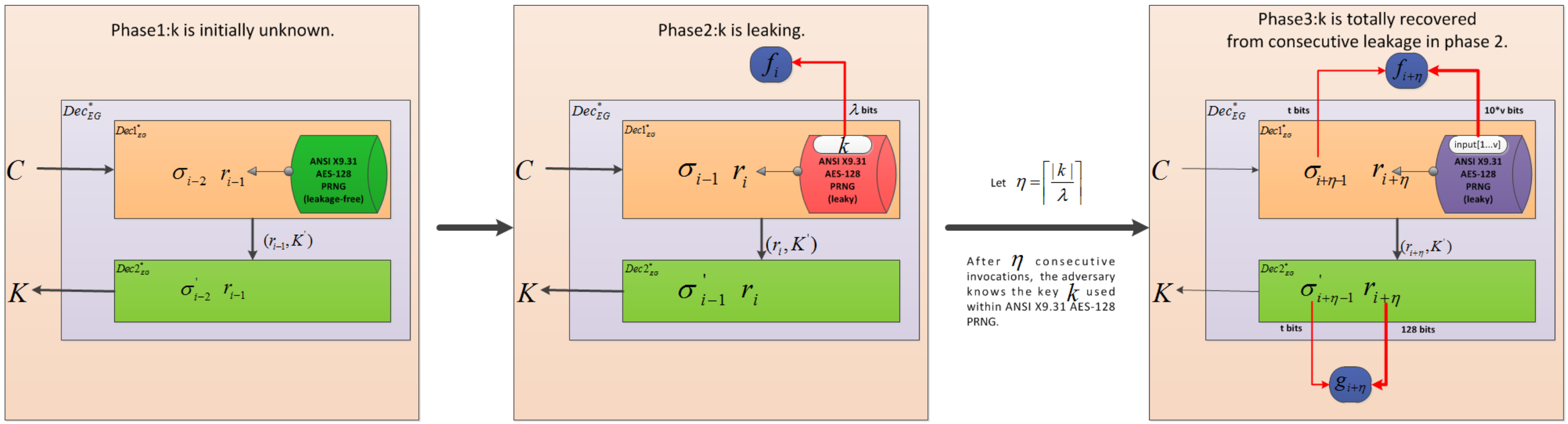

Fig. 4. Our attack on decapsulation of EG^* with a leaky ANSI X9.31 PRNG Using AES-128

Table 4.1.1 (resp. Table 4.1.2) shows the number of leaked bits about leakage function f for its two different parts, and the results correspond to two attacks and different size of strong primes when the number of equations is 49 (resp. 30). Table 4.2.1 and Table 4.2.2 show the similar information about leakage function g.

Table 5.1 (resp. Table 5.2) shows the percentage of $\rho_{\{ATTACKI,ATTACKII\}}$ and the specific value of $\lambda_{\{ATTACKI,ATTACKII\}}$ for ANSI X9.31 PRNG Using AES-128 and the number of iteration times v of the PRNG for different size primes when the number of equations is 49 (resp. 30).

From Table 5.1, Table 5.2 and Figure 6-9, it is clear that the scheme EG^* is not secure any more, if it uses this ANSI X9.31 PRNG Using AES-128 when strong primes $|p| \geq 640$ bits and when the number of equations is 49 (resp. 30). For strong primes longer than 1024 bits, the tolerance leakage rate is much lower, and thus we don't present all the data in the paper.

Table 4.1.1. The specific leakage bit number for ANSI X9.31 PRNG Using AES-128 of leakage function f in 49 equations case

$\|p\|$	Attacks	$\mu_{\sigma f}$	μ_{rf}	$\|f\|$
640 bits	ATTACK I	22	320	342
	ATTACK II	22	50	72
768 bits	ATTACK I	24	384	408
	ATTACK II	24	60	84
896 bits	ATTACK I	27	448	475
	ATTACK II	27	70	97
1024 bits	ATTACK I	30	512	542
	ATTACK II	30	80	110

Table 4.1.2. The specific leakage bit number for ANSI X9.31 PRNG Using AES-128 of leakage function f in 30 equations case

$\|p\|$	Attacks	$\mu_{\sigma f}$	μ_{rf}	$\|f\|$
640 bits	ATTACK I	29	320	349
	ATTACK II	29	50	79
768 bits	ATTACK I	33	384	417
	ATTACK II	33	60	93
896 bits	ATTACK I	37	448	485
	ATTACK II	37	70	107
1024 bits	ATTACK I	41	512	553
	ATTACK II	41	80	121

Table 4.2.1. The specific leakage bit number for ANSI X9.31 PRNG Using AES-128 of leakage function g in 49 equations case

$\|p\|$	Attacks	$\mu_{\sigma' g}$	μ_{rg}	$\|g\|$
640 bits	ATTACK I	22	320	342
	ATTACK II	22	128	150
768 bits	ATTACK I	24	384	408
	ATTACK II	24	128	152
896 bits	ATTACK I	27	448	475
	ATTACK II	27	128	155
1024 bits	ATTACK I	30	512	542
	ATTACK II	30	128	158

Table 4.2.2. The specific leakage bit number for ANSI X9.31 PRNG Using AES-128 of leakage function g in 30 equations case

$\|p\|$	Attacks	$\mu_{\sigma' g}$	μ_{rg}	$\|g\|$
640 bits	ATTACK I	29	320	349
	ATTACK II	29	128	157
768 bits	ATTACK I	33	384	417
	ATTACK II	33	128	161
896 bits	ATTACK I	37	448	485
	ATTACK II	37	128	165
1024 bits	ATTACK I	41	512	553
	ATTACK II	41	128	169

Table 5.1. The percentage of $\rho_{\{ATTACKI,ATTACKII\}}$ and the specific value of $\lambda_{\{ATTACKI,ATTACKII\}}$ about ANSI X9.31 PRNG Using AES-128 in 49 equations case

$\|p\|$ (in bits)	$\rho_{ATTACKI}$	$\rho_{ATTACKII}$	$\lambda_{ATTACKI}$	$\lambda_{ATTACKII}$	v (times)
640	106.88%	34.68%	342	150	5
768	106.25%	30.73%	408	152	6
896	106.03%	28.13%	475	155	7
1024	105.86%	26.17%	542	158	8

Table 5.2. The percentage of $\rho_{\{ATTACKI,ATTACKII\}}$ and the specific value of $\lambda_{\{ATTACKI,ATTACKII\}}$ about ANSI X9.31 PRNG Using AES-128 in 30 equations case

$\|p\|$ (in bits)	$\rho_{ATTACKI}$	$\rho_{ATTACKII}$	$\lambda_{ATTACKI}$	$\lambda_{ATTACKII}$	v (times)
640	109.06%	36.88%	349	157	5
768	108.59%	33.07%	417	161	6
896	108.26%	30.36%	485	165	7
1024	108.01%	28.32%	553	169	8

3.3.3 CASE 3: FIPS 186 PRNG for DSA Pre-message Secrets

The Digital Signature Standard specification (FIPS 186) [30] also describes a fairly simple PRNG based on SHA or DES, which is used for generating DSA per-message secrets. The random secrets generation algorithm of FIPS 186 PRNG is as 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 $160 \le b \le 512$. 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 i from 1 to v do the following:

4.1 $output[i] \leftarrow G(str, seed[i]) \mod (q)$.

4.2 $seed[i+1] \leftarrow (1 + seed[i] + output[i]) \mod (2^b)$.

Step 5 Return $(output[1], output[2], \ldots, output[v])$.

For general purpose PRNG, *mod q* operation could be omitted. It is necessary only for DSS where all arithmetic is done *mod q*. In this paper, we only consider the case of that G function is based on DES. Therefore, the output of this PRNG is 160 bits long. When $|p| = 640$ bits, one can generate r_i by invoking this PRNG only 4 times iteratively. The leakage functions are defined as follows.

$$f_{i+u}(\sigma_{i+u-1}, r_{i+u}) = \langle \sigma_{i+u-1}^{[t]}, seed[1]_{i+u}^{[120]} \rangle$$

$$\begin{aligned} & g_{i+u}(\sigma'^{[t]}_{i+u-1}, (r_{i+u}, K'), r_{i+u}^{-1}) \\ &= \langle \sigma'^{[t]}_{i+u-1}, output[1]_{i+u}^{[40]}, output[2]_{i+u}^{[30]}, output[3]_{i+u}^{[30]}, output[4]_{i+u}^{[20]} \rangle, \end{aligned}$$

$u = 1, \ldots, n-1$, and

$$f_{i+n}(\sigma_{i+n-1}, r_{i+n}) = \langle \sigma_{i+n-1}^{[t]} \rangle$$

$$g_{i+n}(\sigma'_{i+n-1}, (r_{i+n}, K'), r_{i+n}^{-1}) = \langle \sigma'^{[t]}_{i+n-1} \rangle.$$

For each $u = 1, 2, \ldots, n-1$, after the adversary gets $seed[1]_{i+u}^{[120]}$, he could try all possible values of the least significant 40 bits of $seed[1]_{i+u}$ (in a brute-force way), and he will get 2^{40} candidate values of $seed[1]_{i+u}$. Denote one candidate value by $seed[1]'_{i+u}$.

The adversary could use the following procedure to verify the correctness of each guess. For every $seed[1]'_{i+u}$, the adversary computes $output[1]'_{i+u} =$

$G(str, seed[1]'_{i+u})$. If $output[1]'^{[40]}_{i+u} = output[1]^{[40]}_{i+u}$, then the adversary will compute $seed[2]'_{i+u}$ and $output[2]'_{i+u} = G(str, seed[2]'_{i+u})$; otherwise, the adversary will try the next candidate value of $seed[1]_{i+u}$. If $output[2]'^{[30]}_{i+u} = output[2]^{[30]}_{i+u}$, then the adversary will compute $seed[3]'_{i+u}$ and $output[3]'_{i+u} = G(str, seed[3]'_{i+u})$; otherwise, the adversary will try the next candidate value of $seed[1]_{i+u}$. If $output[3]'^{[30]}_{i+u} = output[3]^{[30]}_{i+u}$ then the adversary will compute $seed[4]'_{i+u}$ and $output[4]'_{i+u} = G(str, seed[4]'_{i+u})$; otherwise, the adversary will try the next candidate value of $seed[1]_{i+u}$. If $output[4]'^{[20]}_{i+u} = output[4]^{[20]}_{i+u}$, the adversary will believe that the $seed[1]'_{i+u}$ passes the test and it equals to $seed[1]_{i+u}$ with high probability; otherwise, the adversary will try the next candidate value of $seed[1]_{i+u}$.

Assuming that for every input a, the output of G function $G(str, a)$ is uniformly distributed over $\{0,1\}^{160}$. We will analyze the probability that a candidate $seed[1]'$ (Note that, to simplify description, we omit the subscript in notations here.) passes the above test in every invocation of the decryption query.

For every $seed[1]'$ and $output[1]'$, $\mathbf{Pr}[output[1]'^{[40]} = output[1]^{[40]}] = 2^{120}/2^{160} = 1/2^{40}$. For $output[2]' = G(str, output[1]')$, $\mathbf{Pr}[output[2]'^{[30]} = output[2]^{[30]}] = 2^{130}/2^{160} = 1/2^{30}$. Similarly, we have

$$\mathbf{Pr}[output[3]'^{[30]} = output[3]^{[30]}] = 2^{130}/2^{160} = 1/2^{30},$$

$$\mathbf{Pr}[output[4]'^{[20]} = output[4]^{[20]}] = 2^{140}/2^{160} = 1/2^{20}.$$

Therefore, the probability of $seed[1]'$ passing the test is $1/2^{120}$. Due to the fact that the number of $seed[1]'$ is 2^{40} and there exists only one $seed[1]'$ equals to $seed[1]$, the probability of generating more than one $seed[1]'$ that pass the test is

$$1 - \big(1 - \frac{1}{2^{120}}\big)^{2^{40}-1}.$$

So, using this simple verification method, the adversary can recover $seed[1]$ with high probability and then could recover the randomness from

$$r_i = (output[1] \parallel \ldots \parallel output[4])$$

easily. Figure 5 shows the attack process.

When the size of p are 800 bits, 960 bits and 1120 bits, the probability of successful recovery remains unchanged, even though the recovery processes have a little difference from each other. The difference of recovery process for different size of p is that the leaked bit number for each output is different, while the number of total leaked bits are all 120 bits. Table 6 shows the specific leakage bit number for each output, for different sizes of p.

Table 7.1.1 (resp. Table 7.1.2) shows the leakage bit number about leakage function f for two different parts for our two attacks under different sizes of strong prime p when the number of equations is 49 (resp. 30). Table 7.2.1 and 7.2.2 show the similar information for leakage function g.

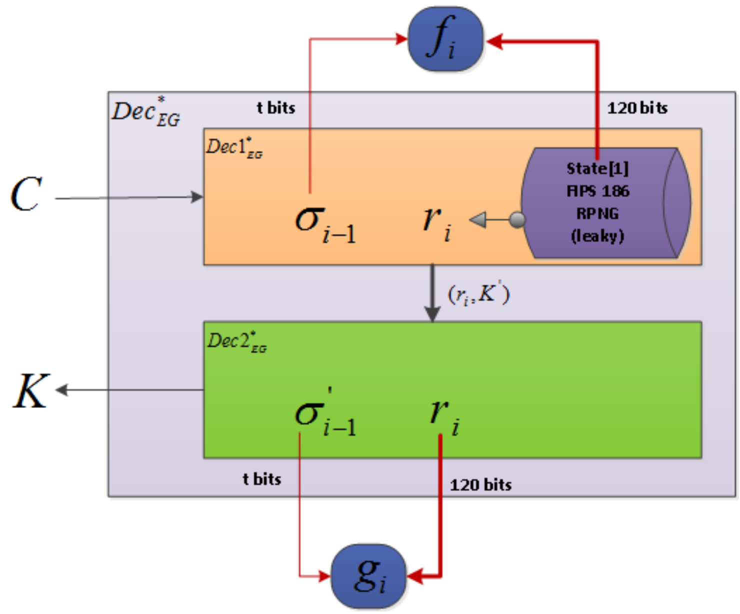

Fig. 5. Our attack on decapsulation of EG^* with a leaky FIPS 186 PRNG

Table 6. The specific leakage bit number of each output for different size p

$\|p\|$ (in bits)	$output[1]$	$output[2]$	$output[3]$	$output[4]$	$output[5]$	$output[6]$	$output[7]$
800	40	30	30	10	10	-	-
960	40	30	30	10	5	5	-
1120	40	30	20	10	10	5	5

Table 7.1.1. The specific leakage bit number for DSA PRNG of leakage function f in 49 equations case

$\|p\|$	Attacks	$\mu_{\sigma f}$	$\mu_{r f}$	$\|f\|$
640 bits	ATTACK I	22	320	342
	ATTACK II	22	120	142
800 bits	ATTACK I	25	400	425
	ATTACK II	25	120	145
960 bits	ATTACK I	28	480	508
	ATTACK II	28	120	148
1120 bits	ATTACK I	31	560	591
	ATTACK II	31	120	151

Table 7.1.2. The specific leakage bit number for DSA PRNG of leakage function f in 30 equations case

$\|p\|$	Attacks	$\mu_{\sigma f}$	μ_{rf}	$\|f\|$
640 bits	ATTACK I	29	320	349
	ATTACK II	29	120	149
800 bits	ATTACK I	34	400	434
	ATTACK II	34	120	154
960 bits	ATTACK I	39	480	519
	ATTACK II	39	120	159
1120 bits	ATTACK I	44	560	604
	ATTACK II	44	120	164

Table 7.2.1. The specific leakage bit number for DSA PRNG of leakage function g in 49 equations case

$\|p\|$	Attacks	$\mu_{\sigma' g}$	μ_{rg}	$\|g\|$
640 bits	ATTACK I	22	320	342
	ATTACK II	22	120	142
800 bits	ATTACK I	25	400	425
	ATTACK II	25	120	145
960 bits	ATTACK I	28	480	508
	ATTACK II	28	120	148
1120 bits	ATTACK I	31	560	591
	ATTACK II	31	120	151

Table 7.2.2. The specific leakage bit number for DSA PRNG of leakage function g in 30 equations case

$\|p\|$	Attacks	$\mu_{\sigma' g}$	μ_{rg}	$\|g\|$
640 bits	ATTACK I	29	320	349
	ATTACK II	29	120	149
800 bits	ATTACK I	34	400	434
	ATTACK II	34	120	154
960 bits	ATTACK I	39	480	519
	ATTACK II	39	120	159
1120 bits	ATTACK I	44	560	604
	ATTACK II	44	120	164

Table 8.1. The percentage of $\rho_{\{ATTACKI, ATTACKII\}}$ and the specific value of $\lambda_{\{ATTACKI, ATTACKII\}}$ about the DSA PRNG in 49 equations case

$\|p\|$ (in bits)	$\rho_{ATTACKI}$	$\rho_{ATTACKII}$	$\lambda_{ATTACKI}$	$\lambda_{ATTACKII}$	v (times)
640	106.88%	44.38%	342	142	4
800	106.25%	36.25%	425	145	5
960	105.83%	30.83%	508	148	6
1120	105.54%	26.96%	591	151	7

Table 8.2. The percentage of $\rho_{\{ATTACKI,ATTACKII\}}$ and the specific value of $\lambda_{\{ATTACKI,ATTACKII\}}$ about the DSA PRNG in 30 equations case

$\|p\|$ (in bits)	$\rho_{ATTACKI}$	$\rho_{ATTACKII}$	$\lambda_{ATTACKI}$	$\lambda_{ATTACKII}$	v (times)
640	109.06%	46.56%	349	149	4
800	108.5%	38.5%	434	154	5
960	108.13%	33.13%	519	159	6
1120	107.86%	29.29%	604	164	7

From Table 8.1, Table 8.2 and Figure 6-9, it is clear that the scheme EG^* is not secure any more, if it uses this DSA PRNG when strong primes $|p| \geq 640$ bits and when the number of equations is 49 (resp. 30). For strong primes longer than 1120 bits, the tolerance leakage rate is much lower, and thus we don't present all the data in the paper.

3.4 Theoretical Analysis of Our Non-Generic Attacks

The basic reason of the success of both our attack is that the multiplicative secret shares σ_i and σ'_i are not updated independently. If they are updated independently, we think that our attacks may be unfeasible.

By our ATTACK II, we can see that both the sizes of the secret states and the mathematical complexity of the PRNGs have important impact on the attack result. Unsuitable size of the secret state of a PRNG may make ATTACK II become successful. On the other hand, when the PRNG has simpler mathematical complexity, we believe that ATTACK II will become more powerful. For example, considering LFSR (e.g., Geffe Generator), the secret state only have 96 bits . As a result, when $|p| = 1024$ bits, the adversary can recover the secret key of EG^* when $\lambda = 0.1338 * log(p)$ bits and $\rho_{ATTACKII} = 17.38\%$ in 30 equations case.

By our attacks, we find that, with the increase of the size of the underlying hard problem, the tolerance leakage rate of EG^* will decrease if the implementation of the scheme EG^* invokes the PRNGs iteratively to generate r_i.

According to [34], 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, from section 3.3, we can see that if the scheme EG^* uses the above-mentioned three PRNGs to generate r_i, the scheme will not be secure any more. On the other hand, nowadays, larger size of p must be used in the ElGamal encryption scheme. Interestingly enough, our attacks need less tolerance leakage rate when the size of p is larger. Therefore, we guess that our attacks could be more effective for the state-of-the-art ElGamal encryption scheme. Because of the reasons above, we need only considering the smaller size of p which could be used now. The smaller size of p reveals one lower bound for successful attacks.

Figure 6 shows the tolerance leakage rate of EG^* according to our first attack method and the three PRNGs in our second attack method in 49 equations case. Figure 7 shows the same content in 30 equations case.

Figure 8 shows the minimum percentage of $\lambda/|p|$ required to successfully recover x for different PRNGs in 49 equations case. Figure 9 shows the same content in 30 equations case.

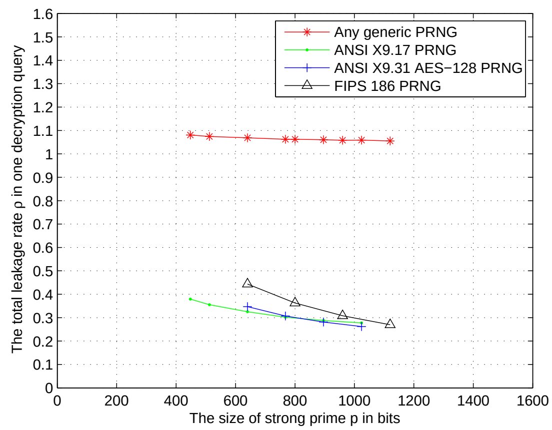

Fig. 6. Minimum leakage rate required to successfully recover x for different PRNGs (When the number of equations is 49)

4 Experimental Implementations of Our Non-Generic Attacks

We implemented all our attacks. For this purpose, we designed two groups of experiments. In our first group of experiments (refers to Phase I), we tried to recover r_i from the leakage information about each kind of PRNG in section 3.3. In our second group of experiment (refers to Phase II), we tested the success rate of recovering the secret key x by solving the system of linear congruence equations (1) when the adversary can build the system (1). We ran our experiments in 64-bit mode over an Intel Core 2 Quad Q9550 processor at 2.83GHz with 4GB of DDR3 SDRAM. ***Note that*** Phase I and Phase II are exactly based on the same sets of leakages from two leakage functions, ***not*** from different sets of leakages, even we divided our attacks into two phases.

4.1 PHASE I: Recovery of r_i

The first group of experiment showed how to recover r_i from the leakage information about three PRNGs concerned in Section 3.3. We used VC++6.0,

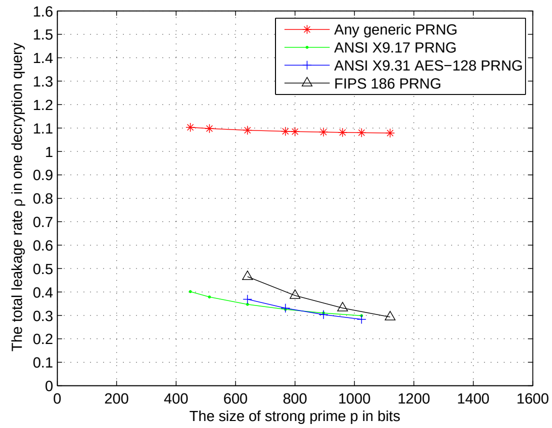

Fig. 7. Minimum leakage rate required to successfully recover x for different PRNGs (When the number of equations is 30)

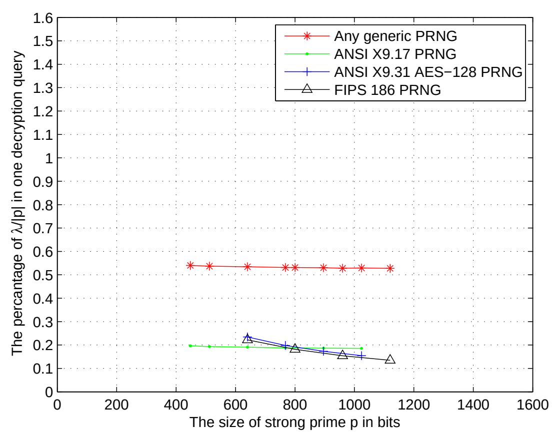

Fig. 8. Minimum percentage of $\lambda/|p|$ required to successfully recover x for different PRNGs (When the number of equations is 49)

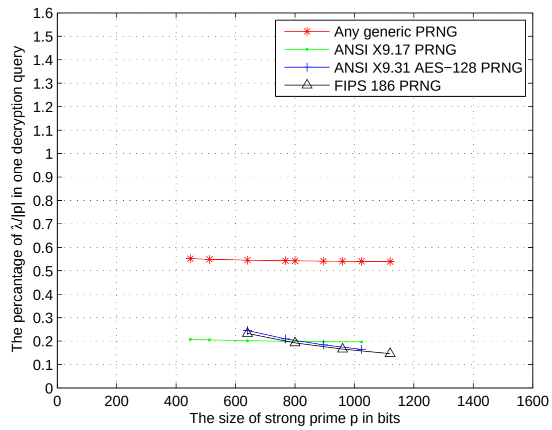

Fig. 9. Minimum percentage of $\lambda/|p|$ required to successfully recover x for different PRNGs (When the number of equations is 30)

GMP4.1.2 and Crypto++5.6.1 to implement our programs.

ANSI X9.17 PRNG

In this experiment, we only considered the case of $|p| = 768$ bits. For other size of strong primes, the processes remains the same. When $|p| = 768$ bits, the PRNG, which uses DES E-D-E two-key triple-encryption algorithm, will be continually invoked 12 times to generate a $r_i \in \mathbb{Z}_p^*$. Denote the key of DES E-D-E two-key triple-encryption by $k = \{key1, key2\}$. The specific value of k used in our experiments was as follows.

$$key1 = \text{0x21 0x97 0x88 0x5A 0x6D 0x8C 0xFD 0x37},$$

$$key2 = \text{0x81 0x89 0xFC 0xCA 0x46 0x87 0x42 0xFB}.$$

We used the function GetLocalTime in VC++6.0 to get the 64 bits system time (wHour,wMinute,wSecond and wMilliseconds). According to the result of the experiment, we found that the system time of one iteration of PRNG is different from each other only in their least significant 8 bits. In the experiment, we chose $input[i]^{[56]}$ =0x00 0x0A 0x00 0x26 0x00 0x10 0x00, $(i = 1, 2, \ldots, 12)$. Table 9 shows the least significant 8 bits of the system time of the 12 times continual invocation of the ANSI X9.17 PRNG.

In our attack, the leakage function will leak $input[i]_{[10]}$, $(i = 1, 2, \ldots, 12)$. Moreover, the adversary already knows $input[i]^{[54]}$, $(i = 1, 2, \ldots, 12)$. Therefore, the adversary completely knows $input[i]$, $(i = 1, 2, \ldots, 12)$.

Furthermore, the adversary knows $output[1]$ from leakage function g. In our experiment, we let $output[1]$ =0x3D 0xB4 0xD2 0x54 0x63 0xAB 0x97 0xF3. The

Table 9. The least significant 8 bits of $input[i]$ for each iteration

i	**1**	**2**	**3**	**4**	**5**	**6**
$input[i]_{[8]}$	0x91	0x92	0x94	0x95	0x96	0x97
i	**7**	**8**	**9**	**10**	**11**	**12**
$input[i]_{[8]}$	0x98	0x99	0x9A	0x9B	0x9B	0x9C

adversary can recover $seed[1]$ =0x67 0xAD 0x74 0xFF 0xEC 0x21 0x59 0x64 by computing $D_k(output[1]) \oplus E_k(input[1])$. And then the adversary can compute $output[i]$, $(i = 2, 3, \ldots, 12)$. Table 10 shows the outputs of 12 invocations of the underlying PRNGs.

Table 10. The output of ANSI X9.17 PRNG

$output[1]$	0x3D	0xB4	0xD2	0x54	0x63	0xAB	0x97	0xF3
$output[2]$	0x20	0xA6	0x5F	0xA5	0x2E	0x7F	0xCF	0xAC
$output[3]$	0xCC	0xD4	0xD6	0xDF	0xFE	0xF4	0x65	0xA8
$output[4]$	0x53	0xEF	0xD8	0xF1	0x8C	0x96	0x89	0x83
$output[5]$	0x0E	0xAA	0x1C	0xE7	0x63	0x86	0xAB	0xD4
$output[6]$	0x11	0xB1	0x2E	0xE0	0x54	0x87	0x32	0x75
$output[7]$	0x95	0xB2	0xA1	0x1E	0xF3	0xB0	0xEF	0xEB
$output[8]$	0xF9	0x3C	0xA9	0x45	0xA8	0x5F	0x06	0x70
$output[9]$	0x03	0x8D	0x8C	0x76	0xE5	0x9B	0x18	0x44
$output[10]$	0x72	0xD9	0x69	0xD6	0xEC	0x91	0x85	0xCF
$output[11]$	0x52	0x27	0xF1	0xDE	0x10	0x0C	0x1B	0x00
$output[12]$	0x43	0xD7	0x9F	0x03	0x0B	0x9A	0xEF	0x76

ANSI X9.31 PRNG Using AES-128

In this experiment, we only considered the case of $|p| = 1024$ bits. For other size of strong primes, the verification procedure remains almost the same. When $|p| = 1024$ bits, the PRNG, which uses 128 bits AES encryption algorithm, will be invoked 8 times to generate a $r_i \in \mathbb{Z}_p^*$. Denote this 128 bits key for AES encryption algorithm by k. The specific value of k used in our experiments was as follows.

0x5F 0x5E 0x8D 0xE6 0x75 0xE1 0x3A 0xE4

0x75 0x20 0x2F 0xAD 0x78 0xC2 0x62 0xD1.

We also used the function GetLocalTime in VC++6.0 to get the 128 bits system date and time (wYear,wMonth,wDay,wDayOfWeek,wHour,wMinute,wSecond and wMilliseconds). According to the result of the experiment, we found that the system time of one iteration of PRNG is different from each other only in their least significant 8 bits. In the experiment, we chose $input[i]^{[120]}$ =0x07

0xDC 0x00 0x08 0x00 0x1C 0x00 0x02 0x00 0x0E 0x00 0x21 0x00 0x1B 0x03, $(i = 1, 2, \ldots, 8)$. Table 11 shows the least significant 8 bits of the system date and time of 8 continual invocations of the ANSI X9.31 PRNG using AES-128.

Table 11. The least significant 8 bits of *input*[*i*] for each iteration

i	**1**	**2**	**3**	**4**
$input[i]_{[8]}$	0x0D	0x0D	0x0E	0x0E
i	**5**	**6**	**7**	**8**
$input[i]_{[8]}$	0x0F	0x0F	0x10	0x10

In our attack, the leakage function will leak $input[i]_{[10]}, (i = 1, 2, \ldots, 8)$. Moreover, the adversary already knows $input[i]^{[118]}, (i = 1, 2, \ldots, 8)$. Therefore, the adversary completely knows $input[i], (i = 1, 2, \ldots, 8)$.

Furthermore, the adversary knows $output[1]$ from leakage function g. In our experiment, we assumed $output[1]$ =0x2E 0x94 0xB2 0x67 0x26 0x43 0xB4 0x4C 0xB4 0xDA 0x4F 0x61 0x09 0x07 0xD4 0xB3. The adversary can recover $seed[1]$ =0x5D 0xEF 0x56 0x4B 0xBE 0x4C 0x3F 0x36 0x21 0x18 0x43 0x26 0x60 0x1A 0xE7 0xF3 by computing $D_k(output[1]) \oplus E_k(input[1])$. And then the adversary can compute $output[i], (i = 2, 3, \ldots, 8)$. Table 12 shows the outputs of 8 invocations of the underlying PRNG.

Table 12. All the output of ANSI X9.31 PRNG Using AES-128

output[1]	0x2E	0x94	0xB2	0x67	0x26	0x43	0xB4	0x4C
	0xB4	0xDA	0x4F	0x61	0x09	0x07	0xD4	0xB3
output[2]	0x53	0x51	0x85	0x3A	0x5C	0x23	0x7E	0xE6
	0x13	0xCA	0x21	0xF0	0xF2	0xFB	0xE6	0xEB
output[3]	0xA5	0x1C	0xE3	0xAB	0x55	0x62	0x4C	0x31
	0x5B	0x37	0xC1	0x0B	0x2E	0xBF	0x97	0x06
output[4]	0xD5	0xE5	0x90	0x9B	0x40	0x10	0x59	0x51
	0xFC	0xC7	0x4E	0xE3	0xA4	0x4F	0xC0	0xE0
output[5]	0xB9	0x38	0x47	0x70	0x35	0x95	0x9F	0x67
	0x01	0x81	0x31	0x14	0x00	0x30	0xDE	0x4B
output[6]	0x51	0x40	0x44	0x7C	0x60	0xD5	0x57	0x60
	0xC0	0x3F	0x90	0x19	0x29	0xDE	0x02	0xDF
output[7]	0x09	0xE3	0x67	0xD0	0x40	0x68	0xBC	0xE5
	0xB9	0x7B	0xA6	0xFA	0xAF	0x84	0x4F	0x59
output[8]	0x93	0xFE	0x1B	0x7C	0x04	0x82	0x51	0x2E
	0x80	0x80	0xCA	0xFE	0x7D	0xFB	0x63	0x40

FIPS 186 PRNG for DSA Pre-message Secrets

In this experiment, we only considered the case of $|p| = 960$ bits. For other size of strong primes, the verification procedure remains the same. When $|p| = 960$ bits, the scheme EG^* will invoke DSA PRNG 6 times to generate a $r_i \in \mathbb{Z}_p^*$.

In our attack, the adversary needs exhaustively try 2^{40} $seed[1]'$ (For simpler description, we ignore the subscript here.). However, due to the fact that our computing resource was limited, we assumed that the adversary will get $seed[1]^{[125]}$, which is 5 bits more than our original assumption. Therefore, the adversary only needs exhaustively try 2^{35} $seed[1]'$. Obviously, the difference of the corresponding experiment results between these two cases is extremely small, which could be neglected. Table 13 shows the specific leakage bit number corresponding to the output of each case for different size of p.

Table 13. The specific leakage bit number of each output for different size of p when the leakage function leaks $seed[1]^{[125]}$

$\|p\|$	$output[1]$	$output[2]$	$output[3]$	$output[4]$	$output[5]$	$output[6]$	$output[7]$
640 bits	40	30	30	25	-	-	-
800 bits	40	30	30	15	10	-	-
960 bits	40	30	30	10	10	5	-
1120 bits	40	30	20	10	10	10	5

We chose $seed[1]$=0110001100101000001110010101011110111010110111 1
001010010011010010100001001101010011111000010000000000110010101 1
1010100010010100010001100111001110001111001000111.

Specifically, the adversary will exhaustively try 2^{35} $seed[1]'$. He has $output[1]^{[40]}$, $output[2]^{[30]}$,$output[3]^{[30]}$,$output[4]^{[10]}$,$output[5]^{[10]}$,$output[6]^{[5]}$. The experiment showed that the adversary can recover the unique $seed[1]$, and then recover all the $output[i], (i = 1, 2, \ldots, 6)$. Table 14 shows all the outputs of DSA PRNG.

As expected, for cases $|p| = 640$ bits, $|p| = 800$ bits and $|p| = 1120$ bits, only one $seed[1]'$ passed the test and it really was $seed[1]$.

Table 14. All the 6 output of DSA PRNG

$output[1]$	0x45	0x2A	0xBF	0x99	0xD4	0xCB	0x9C	0x4E	0xA4	0x54
	0x56	0xE7	0x7E	0x6B	0x8E	0x6C	0x93	0x2F	0xCE	0x14
$output[2]$	0x2D	0x4D	0xBC	0xAC	0xD3	0x76	0x8F	0x50	0x6F	0x42
	0x6B	0x17	0xBA	0xA7	0xB1	0x50	0x96	0xD7	0x46	0x73
$output[3]$	0x9E	0x34	0x52	0x94	0x39	0xC9	0xF2	0x0D	0xC7	0xD5
	0x6C	0x6A	0xEB	0x55	0xA3	0x4E	0x21	0xF4	0x01	0xB6
$output[4]$	0x72	0x3D	0xB7	0xB1	0x11	0x36	0xDD	0xE8	0x20	0xC9
	0xE9	0xC7	0x9C	0x81	0xA9	0xE7	0x30	0x86	0x98	0x9A
$output[5]$	0x80	0xC2	0xA3	0x75	0x45	0x19	0x93	0xEA	0xA7	0xA3
	0xE0	0x9E	0xC0	0xF5	0x77	0x3E	0x09	0x13	0x8C	0x80
$output[6]$	0x5A	0x99	0x98	0xBA	0xA5	0x12	0x3D	0x56	0xC6	0x28
	0x4B	0xF3	0x09	0xC7	0x49	0x77	0xBD	0x5D	0xEE	0x94

4.2 PHASE II: Recovery of Secret Key x by Solving systems of Linear Congruence Equations about σ_i and σ_i'

The goal of our second group of experiment was to check the success rate of recovering the secret key x by solving the systems of linear congruence equations (1) about multiplicative secret shares σ_i and σ_i'. In order to achieve this goal, we used VC++6.0, GMP4.1.2 and MAGMA V2.12-16 to implement the program solving the system of linear congruence equations (1). We chose four strong primes with their length being 160 bits, 256 bits, 512 bits and 1024 bits, respectively (These strong primes are listed in Appendix B.). For each size of strong prime p, we generated 100 sets of random data. For every set of these data, we tried to recover a candidate value of secret key x (denoted by x'). Then we can verify the correctness of x' by a correct plaintext-ciphertext pair (C, K). If $K = C^{x'}$, then we can believe that $x' = x$, which means that the secret key x is recovered successfully. We counted the number of how many test data from which the secret key can be recovered successfully.

The most critical part of the program for solving the system of linear congruence equations (1) is Step 1 of Algorithm 1 (We exploited the function SuccessiveMinima in MAGMA V2.12-16 to implement Step 1). We assumed that if Step 1 does not finish in 30 minutes, or the memory overflows, then the process of solving (1) fails.

Due to our computing ability, we assumed that the adversary builds the system of linear congruence equations (1) with 30 equations. However, [2] claims that the system of linear congruence equations with 49 equations, which is similar to (1), can be solved in practice. Figure 10 shows the success rates of recovering the secret key x for different size of strong primes. During our experiment process, only one set of test data of 160 bits long strong prime returned a wrong answer ($x' \neq x$). The other unsuccessful set of test data failed because of timeout or memory overflow.

5 Conclusions and Future Work

In our attacks, the adversary could exploit the leakage information to build the systems of linear congruence equations about the multiplicative secret shares of the secret key x. By solving the systems of linear congruence equations with lattice theory, the adversary could recover the secret key x with high probability. Our research reveals the following observations:

First of all, some theoretical attacks against one leakage resilient cryptography scheme which does not pose a threat might have a serious threat when this leakage resilient scheme is mathematically implemented using specific cryptography component. For example, in the case of scheme EG^*, when $\lambda = 0.25 * log(p)$, our first attack method will not threaten the theoretical security of mathematical realizations of EG^*. However, our second attack method shows that the tolerance leakage rate of mathematical realizations of EG^* will drop below 25% for some widely used PRNGs. For other leakage resilient cryptography schemes, we conjecture that this problem might still exists.

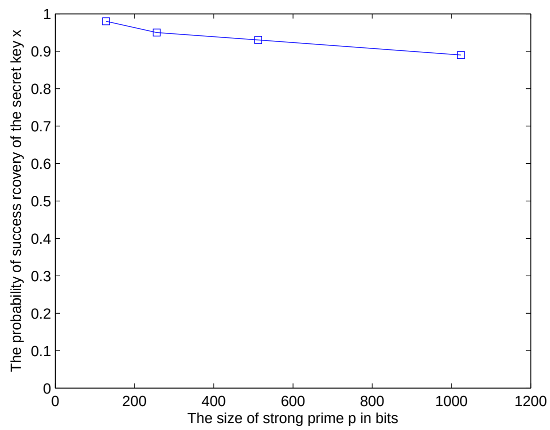

Fig. 10. Relationship between probability of successful recovery of x and size of p in bits

Second, when one leakage resilient scheme is mathematically implemented using some specific cryptography component, the method of increasing the size of its underlying hard problem to resist classical attack against the hard problem scheme may make the scheme tolerate less information leakage. Our attacks clearly reveal this point.

Third, our result illustrates that at least in OCL model, the same number of leakage bits of a leakage resilient scheme has different value for adversaries according to different mathematical implementation technique. Therefore, the problem of designing leakage resilient scheme of which the tolerance leakage bits number is independent with the specific implementation technique is an open and interesting problem.

Fourth, our research shows that the tolerance leakage rate of EG^* will drop below 25% for some widely used PRNGs. This fact illustrates that the implementor of EG^* should carefully chooses specific PRNG and its parameter; otherwise, the scheme might not keep its claimed and expected security. For other leakage resilient cryptography schemes which use PRNGs to generate random numbers, this problem may still exist if the PRNGs are leaky. Additionally, our attacks may be more powerful, if the adversary have other advanced cryptanalysis techniques in hand at the same time. One may conjecture that ATTACK II may be unsuccessful when the scheme EG^* is implemented by leakage resilient PRNG. In fact, with unsuitable size of the secret state of a leakage resilient PRNG, ATTACK II may still feasible. For instance, [40] introduces two leakage resilient PRNGs with essentially the same structure in standard model. The first one uses weak pseudorandom function to construct the leakage resilient PRNG and the second one uses PRNG and two-source extractor. For the first leakage resilient PRNG, when the scheme EG^* uses AES-128 encryption to instantiate the secure block cipher **BC** in its leakage resilient PRNG, the adversary can recover the secret key x successfully with $\lambda = 0.1650 * log(p)$ bits and $\rho_{ATTACKII} = 20.51\%$ when $|p| = 1024$ bits and in 30 equations. The leakage scenario of the leakage resilient PRNGs in [40] is unsuitable for that in EG^*.

Fifth, our attacks does not exploit the mathematical structure of scheme EG^* itself. If the mathematical structure of the scheme EG^* is exploited, we guess there could be some more powerful attacks. For example, [35] exploits the mathematical structure of RSA and uses continued fraction based techniques to successfully attack the RSA. If similar methods for the scheme EG^* were available, it might lead to better results.

Finally, this work make us to consider such a question: where is a suitable frontier for leakage resilient cryptography, i.e. what should be asked to hardware engineers and what should be asked to cryptographers?

Acknowledgments This work was supported by the National Basic Research Program of China (No.2013CB338002), National Natural Science Foundation of China (No. 61272478, 61073178, 60970135 and 61170282), Beijing Natural Science Foundation (No. 4112064), Strategic Priority Research Program of the Chinese Academy of Sciences (No.XDA06010701), and IIE Cryptography Research Project (No. Y2Z0011102).

References

1. E.Kiltz, K.Pietrzak. Leakage Resilient ElGamal Encryption. ASIACRYPT'10, LNCS 6477, pp.595-612, 2010.
2. T.Römer, JP.Seifertl. Information Leakage Attacks against Smart Card Implementations of the Elliptic Curve Digital Signature Algorithm. E-smart'01, LNCS 2140, pp.211-219, 2001.
3. K.Gandol, C.Mourtel and F.Olivier. Electromagnetic Analysis: Concrete Results. CHES'01, LNCS 2162, pp.251-261, 2001.
4. Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. CRYPTO'96, LNCS 1109, pp.104-113, 1996.
5. D. Boneh, R.A. DeMillo and R.J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. EUROCRYPT'97, LNCS 1233, pp.37-51, 1997.
6. P.Kocher, J.Jaffe and B.Jun. Differential Power Analysis. CRYPTO'99, LNCS 1666, PP.388-397, 1999.
7. D. Boneh, G. Durfee and Y. Frankel. An Attack on RSA Given a Fraction of the Private Key Bits. ASIACRYPT'98, LNCS 1514, pp.25-34, 1998.
8. S.Dziembowski, K.Pietrzak. Leakage-Resilient Cryptography. FOCS'08, pp.293-302, 2008.
9. S.Micali, L.Reyzin. Physically Observable Cryptography (Extended abstract). TC-C'04, LNCS2951, pp.278-296, 2004.
10. 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.
11. 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 Cryptography'10, Part 2, pp.99-134, 2010.
12. A.Akavia, S.Goldwasser and V.Vaikuntanathan. Simultaneous Hardcore Bits and Cryptography against Memory Attacks. TCC'09, LNCS 5444, pp.474-495, 2009.
13. M.Naor, G.Segev. Public-key Cryptosystems Resilient to Key Leakage. CRYPTO'09, LNCS 5677, pp.18-35, 2009.
14. S.Dziembowski. Intrusion-Resilience Via the Bounded-Storage Model. TCC'06, LNCS 3876, pp.207-224, 2006.
15. S.Dziembowski. On Forward-Secure Storage (Extended abstract). CRYPTO'06, LNCS 4117, pp.251-270, 2006.
16. D.Cash, Yan Zong Ding, Y.Dodis, W.Lee, R.J. Lipton, S.Walfish. Intrusion-Resilient Key Exchange in the Bounded Retrieval Model . TCC'07, LNCS 4392, pp.479-498, 2007.
17. S.Dziembowski, K.Pietrzak. Intrusion-Resilient Secret Sharing. FOCS'07, pp.227-237, 2007.
18. J.Alwen, Y.Dodis and D.Wichs. Leakage-Resilient Public-Key Cryptography in the Bounded-Retrieval Model. CRYPTO'09, LNCS 5677, pp.36-54,2009.
19. J.Alwen, Y.Dodis, M.Naor, G.Segev, S.Walfish and D.Wichs. Public-Key Encryption in the Bounded-Retrieval Model. EUROCRYPT'10, LNCS 6110, pp.113-134, 2010.
20. Y.Dodis, K.Haralambiev, A.Lόpez-Alt and D.Wichs. Cryptography against Continuous Memory Attacks. FOCS'10, pp.511-520, 2010.
21. Z.Brakerski, Y.T.Kalai, J.Katz and V.Vaikuntanathan. Overcoming the Hole in the Bucket: Public-Key Cryptography Resilient to Continual Memory Leakage. FOCS'10. pp.501-510 , 2010.

22. A.Lewko M.Lewko and B.Waters. How to Leak on Key Updates. STOC'11, pp.725-734, 2011.
23. European Network of Excellence (ECRYPT). The side channel cryptanalysis lounge, http://www.crypto.ruhr-uni-bochum.de/en_sclounge.html (retrieved on 29.03.2008)
24. J.-J. Quisquater, F.Koene. Side channel attacks:State of the art, October 2002.[23].
25. R. Anderson, M. Kuhn. Tamper resistance: a cautionary note. WOEC'96.
26. Christophe Clavier and Marc Joye. Universal exponentiation algorithm. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, CHES2001, volume 2162 of LNCS, page2 300-308. Springer-Verlag, Berlin, Germany, May 2001.
27. Paul C. Kocher and Joshua Jaffe. Leak-Resistant Cryptographic Method and Apparatus. United States Patent 6304658 B1. Oct. 16, 2001.
28. Elena Trichina and Antonio Bellezza. Implementation of elliptic curve cryptography with built-in counter measures against side channel attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, CHES 2002, volume 2523 of LNCS, pages 98-113. Springer-Verlag, Berlin, Germany, August 2002.
29. ANSI X 9.17 (Revised), American National Standard for Financial Institution Key Management (Wholesale)," American Bankers Association, 1985.
30. National Institute for Standards and Technology, Digital Signature Standard," NIST FIPS PUB 186, U.S. Department of Commerce, 1994.
31. Howgrave-Graham, Nguyen, Shparlinski. Hidden number problem with hidden multipliers, timed-release crypto, and noisy exponentiation. Math. Comput. 72(243): 1473-1485 (2003)
32. Sharon 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
33. 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.
34. A. Menezes, P. van Oorschot, and S. Vanstone. Handbook of Applied Cryptography, Chapter 8, pp296, CRC Press,1996.
35. MICHAEL J. WIENER. Cryptanalysis of Short RSA Secret Exponents. IEEE TRANSACTION ON INFORMATION THEORY.VOL.36.NO.3.MAY 1990.
36. http://www.spms.ntu.edu.sg/Asiacrypt2010/AsiaCrypt_slides/pietrzakAC11.pdf.
37. C.Petit, F.-X.Standaert, O.Pereira, Tal G. Malkin, M.Yung. A block cipher based pseudo random number generator secure against side-channel key recovery. ASIACCS'08, pp.56-65, 2008.
38. K.Pietrzak. A leakage-resilient model of operation. Eurocrypt'09, pp.462-482, 2009.
39. F.X.Standaert. How Leaky is an Extractor?. LATINCRYPT'10, LNCS6212, pp.294-304, 2010.
40. Yu Yu, F.-X. Standaert, O.Pereira, M.Yung. Practical Leakage-Resilient Pseudorandom Generators. CCS '10.

Appendix A: The Core Part of Our Attacks

The description of the core part of our attacks is shown in Algorithm 1.

Assuming that the adversary obtains $n-1$ linear congruence equations. Let $b_1 = (r_2, -1, 0, \ldots, 0)^\top, b_2 = (0, r_3, -1, 0, \ldots, 0)^\top, \ldots, b_{n-1} = (0, \ldots, 0, r_n, -1)^\top$ and we define the lattice

$$L = \left\{ y \in \mathbb{R}^n \,\middle|\, y = \sum_{i=1}^{n-1} a_i b_i + a_n p e_1,\ a_i \in \mathbb{Z},\ i = 1, \ldots, n \right\}. \tag{2}$$

Algorithm 1 The algorithm for attacking EG^*

Input: A lattice L like (2)

Output: A solution of the system of equations (1) or a symbol $\perp$

Step 1 Compute n linearly independent vectors for the given lattice L as follows:

$$w_1, \ldots, w_n \in L$$

with $\|w_i\| = \lambda_i(L)$ for $i = 1, \ldots, n$. If there is no such vectors, return $\perp$.

Step 2 Compute integral $n \times (2n-1)$ matrix M satisfying

$$W = \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nn} \end{pmatrix} = M \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n-11} & \cdots & a_{n-1n} \\ p & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & p \end{pmatrix}$$

If there is no such matrix M, then return $\perp$.

Step 3 Multiplying both sides of (1) from left with the matrix M, we get a new system

$$\sum_{j=1}^{n} w_{ij} d'_j = c'_i,\ i = 1, 2, \ldots, n. \tag{3}$$

Choosing c'_i such that $|c'_i| < p/2$. Computing the solution $D = (d'_1, \ldots, d'_n)$ of (3) over $\mathbb{Z}$.

Step 4 Return D.

Appendix B: Strong Primes We Used

Our experiment uses the following strong primes.

$P_{160} = 1220974516789489321772891455348877993516496429083$

$P_{256} = 67578447551123415536940573891022808990516609252954316788185087580595436353407$

$P_{512} = 90761051942387851421384003554833608416684459502573941033177275912365905717613$
$70275365849017977662680994144476299653358016284384318406888090578599829330183$

$P_{1024} = 10496205057038942155086951945845660797046597372002721462696296789520528481 79$
$98196646714946120812366520228682778002458632164630235327791411011238245739381355946$
$76181844028665763235360719453870547110004293785687694493936057876092321346792477869$
$9648484309351641852090775059197103091910029848321468611008957050643$