RSA meets DPA: Recovering RSA Secret Keys from Noisy Analog Data

Noboru Kunihiro[1] and Junya Honda[2]

[1] The University of Tokyo, Japan, `kunihiro@k.u-tokyo.ac.jp`
[2] The University of Tokyo, Japan

Abstract. We discuss how to recover RSA secret keys from noisy analog data obtained through physical attacks such as cold boot and side channel attacks. Many studies have focused on recovering correct secret keys from noisy binary data. Obtaining noisy binary keys typically involves first observing the analog data and then obtaining the binary data through quantization process that discards much information pertaining to the correct keys. In this paper, we propose two algorithms for recovering correct secret keys from noisy analog data, which are generalized variants of Paterson et al.'s algorithm. Our algorithms fully exploit the analog information. More precisely, consider observed data which follows the Gaussian distribution with mean $(-1)^b$ and variance σ^2 for a secret key bit b. We propose a polynomial time algorithm based on the maximum likelihood approach and show that it can recover secret keys if $\sigma < 1.767$. The first algorithm works only if the noise distribution is explicitly known. The second algorithm does not need to know the explicit form of the noise distribution. We implement the first algorithm and verify its effectiveness.

Keywords: RSA, Key-Recovery, Cold Boot Attack, Side Channel Attack, Maximum Likelihood

1 Introduction

1.1 Background and Motivation

Side channel attacks are important concerns for security analysis in the both of public key cryptography and symmetric cryptography. In the typical scenario of the side channel attacks, an attacker tries to recover the full secret key when he can measure some kind of leaked information from cryptographic devices. From the proposal of *Differential Power Analysis (DPA)* by Kocher et al. [6], many studies have been intensively made on the side channel attacks.

We focus on the side channel attacks on RSA cryptosystem. In the RSA cryptosystem [11], a public modulus N is chosen as the product of two distinct primes p and q. The key-pair $(e, d) \in \mathbb{Z}^2$ satisfies $ed \equiv 1 \pmod{(p-1)(q-1)}$. The encryption keys are (N, e) and the decryption keys are (N, d). The PKCS#1 standard [10] specifies that the RSA secret key includes $(p, q, d, d_p, d_q, q^{-1} \bmod p)$

in addition to d, which allows for fast decryption using the Chinese Remainder Theorem. It is important to analyze its security as well as the original RSA.

Recently, the cold boot attack was proposed by Halderman et al. [3] at USENIX Security 2008. They demonstrated that DRAM remanence effects make possible practical, nondestructive attacks that recover a *noisy version of secret keys* stored in a computer's memory. They showed how to reconstruct the full of secret key from the noisy variants for some encryption schemes: DES, AES, tweakable encryption modes, and RSA. How can we recover the correct secret key from a noisy version of the secret key? This is an important question concerning the cold boot attack situation.

Inspired by cold boot attacks [3], much research has been carried on recovering an RSA secret key from a noisy version of the secret key. At Crypto 2009, Heninger and Shacham [5] proposed an algorithm that efficiently recovers secret keys (p, q, d, d_p, d_q) given a random fraction of their bits. Concretely, they showed that if more than 27% of the secret key bits is leaked at random, the full secret key can be recovered. Conversely, this means that even if 73% of the correct secret bits is erased, the key can be recover. As opposed to the Heninger-Shacham algorithm for correcting erasures, Henecka et al. [4] proposed an algorithm for correcting error bits of secret keys at Crypto 2010. They showed that the secret key (p, q, d, d_p, d_q) can be fully recovered if the error probability is less than 0.237. They also showed that the bound for the error probability is given by 0.084 if the associated secret key is (p, q). Paterson et al. proposed an algorithm correcting error bits that occurs asymmetrically at Asiacrypt 2012 [9]. They adopted a coding theoretic approach for designing a new algorithm and analyzing its performance. Sarkar and Maitra [12] revisited the result of [4] and applied the Henecka et al.'s algorithm to break a Chinese Remainder Theorem type implementation of RSA with low weight decryption exponents. Kunihiro et al. [7] proposed an algorithm that generalized the work of [4, 5], and which considered a combined erasure and error setting.

Motivation: Key-Recovery from Noisy Analog Data The previous works [4, 5, 7, 9] considered an erasure and/or error setting, where each bit of the secret key is either erased or flipped. Thus, the noisy version of the secret key is composed of discrete symbols, that is, $\{0, 1\}$ and the erasure symbol "?". However, such discrete data is not always obtained directly and analog data is more natural as observed data obtained through the actual physical attacks such as the cold boot and side channel attacks. We further assume that the observed data follows some fixed probability distributions. It is frequently considered and verified in the practice of side channel attacks (for details, see [8]). Thus, our leakage model is more realistic. Our goal is to propose efficient algorithms that can recover an RSA secret key from noisy analog data.

Paterson et al. [9] concluded that it is an open problem to generalize their approach to the case where soft information (that is, analog data) about the secret bits is available. This is the problem we address in this paper.

1.2 Our Contributions

This paper discusses secret key recovery from a noisy analog data sequence. In our leakage model, the observed value is output according to some fixed probability distribution depending on the corresponding correct secret key bit. Although we cannot directly obtain the true secret key bit, we can observe the noisy and analog variants of the secret key through a side channel or cold boot attack. If the noise is sufficiently small, key recovery from the noisy data is fairly easy in general. However, if the noise is large, the task of recovering the secret key becomes more difficult. Our challenge is to propose an efficient algorithm that recovers the RSA secret key even in the presence of large noise. For this purpose, we adopt a maximum likelihood-based approach.

First, we modify the algorithm of Paterson et al. [9] to adapt an analog data; while their algorithm takes a (noisy) binary bit sequence as input. For the modification, we introduce the concept of *score*; a node with a low score will be discarded, whereas a node with the highest score will be kept and generate a subtree of depth t with 2^t leaf nodes. The score function is calculated from a candidate of the secret key in $\{0,1\}^{5t}$ and the corresponding observed data in $\mathbb{R}^{5t}$. The choice of score function is crucial for our modification.

We propose an algorithm whose score function is constructed from the likelihood ratio and in which the node with the maximal value is regarded as the correct node. We then prove that our algorithm recovers the correct secret key in polynomial time if the noise distribution satisfies a certain condition (Theorem 1). Note that the condition is represented by symmetric capacity. In particular, under the condition that the noise distribution is Gaussian with mean $(-1)^b$ and variance σ^2 for a secret key bit b, we show that we can recover the secret key if $\sigma < 1.767$ (Corollary 2).

The main drawback of the first algorithm is that we need to know the noise distribution exactly; indeed, without this knowledge it does not work. We also propose another algorithm that does not require any knowledge of noise distribution. The score function in the second algorithm is given as a difference between sums of the observed data when the candidate bits are 0 and that when the candidate bits are 1. This score is similar to that of differential power analysis (DPA) [6]. We also prove that the algorithm recovers the correct secret key with high probability if the noise distribution satisfies the certain condition (Theorems 2). Owing to the lack of knowledge of the noise distribution, the condition is slightly worse than that in the first algorithm. However, if the noise follows the Gaussian distribution, the algorithm achieves the same bound as the first one.

We then verify the effectiveness of our algorithm by numerical experiments in the case of Gaussian noise. Our experimental results show that the first algorithm recovers the RSA secret key with significant probability if $\sigma \leq 1.7$, which matches with theoretically predicted bound.

2 Preliminaries

This section presents an overview of the methods [4, 5, 9] using binary trees to recover the secret key of the RSA cryptosystem. We use similar notations to those in [4]. For an n-bit sequence $\mathbf{x} = (x_{n-1}, \dots, x_0) \in \{0,1\}^n$, we denote the i-th bit of $\mathbf{x}$ by $x[i] = x_i$, where $x[0]$ is the least significant bit of $\mathbf{x}$. Let $\tau(M)$ denote the largest exponent such that $2^{\tau(M)} | M$. We denote by $\ln n$ the natural logarithm of n to the base e and by $\log n$ the logarithm of n to the base 2. We denote the expectation of random variable X by $\mathrm{E}[X]$.

2.1 Our Problem: RSA Key-Recovery from Analog Observed Data

Our problem is formulated as follows. We denote the correct secret key by $\mathbf{sk}$. For each bit in $\mathbf{sk}$, a value is observed from the probability distribution depending on the bit value, which means that the analog data are observed according to the leakage model. We denote the observed analog data sequence by $\bar{\mathbf{sk}}$. Our goal is to recover $\mathbf{sk}$ from $\bar{\mathbf{sk}}$.

We will give a more detailed explanation. Suppose that the probability distribution F_x of the observed data is Gaussian with mean $(-1)^x$ and variance σ^2 for $x \in \{0,1\}$. The SNR is commonly used to evaluate the strength of noise is defined by (variance of signal)/(variance of noise). In our leakage model, the variance of the noise is given by σ^2 and that of signal is given by 1. Then, the SNR is given by $1/\sigma^2$. A greater SNR means that the signal is stronger and we can extract information with fewer errors. In this paper, we consider the standard deviation σ for the strength of the noise.

Consider the case that noise level σ is larger. In this case, key-recovery is difficult. In fact, if the noise is extremely large, we cannot recover the secret key, as is discussed in Section 6. Conversely, consider a smaller noise level. In this case, key recovery is relatively easy. Thus, it is important to make a detailed analysis for the value σ.

2.2 Recovering the RSA Secret Key Using a Binary Tree

The first half of explanation of this section is almost the same as previous works [4, 5, 7, 9]. Making this paper self-contained, we give details. We review the key setting of the RSA cryptosystem [11], particular for the PKCS #1 standard [10]. Let (N, e) be the RSA public key, where N is an n-bit RSA modulus and $\mathbf{sk} = (p, q, d, d_p, d_q, q^{-1} \bmod p)$ be the RSA secret key. As in the previous works, we ignore the last component $q^{-1} \bmod p$ in the secret key. The public and secret keys have the following four equations:

$$N = pq,\ ed \equiv 1 \pmod{(p-1)(q-1)},\ ed_p \equiv 1 \pmod{p-1},\ ed_q \equiv 1 \pmod{q-1}.$$

There exist integers k, k_p and k_q such that

$$N = pq,\ ed = 1 + k(p-1)(q-1),\ ed_p = 1 + k_p(p-1),\ ed_q = 1 + k_q(q-1). \quad (1)$$

Suppose that we know the exact values of k, k_p and k_q, then there are five unknowns (p, q, d, d_p, d_q) in the four equations in Eq. (1).

A small public exponent e is usually used in practical applications [15], so we suppose that e is small enough such that $e = 2^{16} + 1$ as is the case in [4, 5, 7, 9]. See [4] for how to compute k, k_p and k_q.

In the previous methods and our new methods, a secret key **sk** is recovered by using a binary-tree-based technique. Here we explain how to recover secret keys, considering $\mathbf{sk} = (p, q, d, d_p, d_q)$ as an example.

First we discuss the generation of the tree. Since p and q are $n/2$-bit prime numbers, there exist at most $2^{n/2}$ candidates for each secret key in (p, q, d, d_p, d_q).

Heninger and Shacham [5] introduced the concept of *slice*. We define the i-th bit slice for each bit index i as

$$\mathbf{slice}(i) := (p[i], q[i], d[i + \tau(k)], d_p[i + \tau(k_p)], d_q[i + \tau(k_q)]).$$

Assume that we have computed a partial solution $\mathbf{sk}' = (p', q', d', d'_p, d'_q)$ up to $\mathbf{slice}(i-1)$. Heninger and Shacham [5] applied Hensel's lemma to Eq. (1) and obtained the following equations

$$\begin{aligned} p[i] + q[i] &= (N - p'q')[i] \bmod 2, \\ d[i+\tau(k)] + p[i] + q[i] &= (k(N+1) + 1 - k(p' + q') - ed')[i + \tau(k)] \bmod 2, \\ d_p[i + \tau(k_p)] + p[i] &= (k_p(p' - 1) + 1 - ed'_p)[i + \tau(k_p)] \bmod 2, \\ d_q[i + \tau(k_q)] + q[i] &= (k_q(q' - 1) + 1 - ed'_q)[i + \tau(k_q)] \bmod 2. \end{aligned}$$

We can easily see that $p[i], q[i], d[i+\tau(k)], d_p[i+\tau(k_p)]$, and $d_q[i+\tau(k_q)]$ are not independent. Each Hensel lift, therefore, yields exactly two candidate solutions. Thus, the total number of candidates is given by $2^{n/2}$.

Henecka et al.'s algorithm (in short, the HMM algorithm) [4] and Paterson et al.'s algorithm (in short, the PPS algorithm) [9] perform t Hensel lifts for some fixed parameter t. For each surviving candidate solution on $\mathbf{slice}(0)$ to $\mathbf{slice}(it - 1)$, a tree with depth t and whose 2^t leaf nodes represent candidate solutions on $\mathbf{slice}(it)$ to $\mathbf{slice}((i + 1)t - 1)$, is generated. This involves $5t$ new bits.

For each new node generated, a pruning phase is carried out. A solution is kept for the next iteration if the Hamming distance between the $5t$ new bits and the corresponding noisy variants of the secret key is less than some threshold as for the HMM algorithm or if the likelihood of the corresponding noisy variants of the secret key for the $5t$ new bits is the highest (or in the highest L nodes) of the 2^t (or $L2^t$) nodes as for the PPS algorithm [9].

The main difference between the HMM and PPS algorithms is how to set the criterion determining whether a certain node is kept or discarded. We adopt a similar approach to the PPS algorithm [9] rather than the HMM algorithm [4]. In other words, we keep the top L nodes with the highest likelihood rather than of the nodes with a lower Hamming distance than the fixed threshold.

3 Maximum Likelihood-based Approach

3.1 Notation and Settings

We denote by m the number of associated secret keys. For example, $m = 5$ if $\mathbf{sk} = (p, q, d, d_p, d_q)$, $m = 3$ if $\mathbf{sk} = (p, q, d)$, and $m = 2$ if $\mathbf{sk} = (p, q)$.

Let $\boldsymbol{x}_{1,a} \in \{0,1\}^m$, $a \in \{1, 2\}$, be the a-th candidate of the first slice $\mathbf{slice}(0)$. We write the two candidates of the first $(i + 1)$ slices when the first i slices are $\boldsymbol{x}_{i,a} = (\mathbf{slice}(0), \cdots, \mathbf{slice}(i-1))$ by $\boldsymbol{x}_{i+1,2a-1}$, $\boldsymbol{x}_{i+1,2a} \in \{0,1\}^{m(i+1)}$.

For notational simplicity we write the j-th slice by $\boldsymbol{x}_{i,a}[j] \in \{0,1\}^m$ and its m elements are denoted by $\boldsymbol{x}_{i,a}[j][1], \boldsymbol{x}_{i,a}[j][2], \cdots, \boldsymbol{x}_{i,a}[j][m] \in \{0,1\}$. Similarly, for a secret key sequence $\boldsymbol{x}_{i,a}$, the observed sequence is denoted by $\boldsymbol{y}_i \in \mathbb{R}^{mi}$ and its element corresponding to $\boldsymbol{x}_{i,a}[j][k]$ is denoted by $\boldsymbol{y}[j][k] \in \mathbb{R}$. We write the sequence of $j, j+1, \cdots, j'$-th elements of a vector $\boldsymbol{x}$ by $\boldsymbol{x}[j : j'] \in \{0,1\}^{m(j'-j+1)}$ for $j' \geq j$. Therefore we have

$$\boldsymbol{x}_{i-1,a} = \boldsymbol{x}_{i,2a-1}[1 : i-1] = \boldsymbol{x}_{i,2a}[1 : i-1]\,. \tag{2}$$

Define $B_l(a) = \lceil a/2^l \rceil$. When we regard $\boldsymbol{x}_{i,a}$ as a node at depth i of the binary tree, the node $\boldsymbol{x}_{i-l,B_l(a)}$ corresponds to the ancestor of $\boldsymbol{x}_{i,a}$ at depth $i-l$. Thus, the relation (2) is generalized to

$$\boldsymbol{x}_{i-l,B_l(a)} = \boldsymbol{x}_{i,a}[1 : i-l] = \boldsymbol{x}_{i,a'}[1 : i-l] \qquad \text{if } B_l(a) = B_l(a'). \tag{3}$$

We also write $\boldsymbol{x}^l$ for the last l elements of a sequence $\boldsymbol{x}$, that is, we write $\boldsymbol{x}^l = \boldsymbol{x}[i-l+1 : i]$ for $\boldsymbol{x} \in \{0,1\}^{mi}$.

Now we introduce the assumption on the secret key.

Assumption 1

(i) Each $\boldsymbol{x}_{i,a}$ is a realization of a random variable $\boldsymbol{X}_{i,a}$ which is (marginally) distributed uniformly over $\{0,1\}^{im}$.

(ii) There exists $c \geq 1$ satisfying the following: for any $i, l, a, a' \in \mathbb{Z}$ such that $c \leq l \leq i$ and $B_l(a) \neq B_l(a')$, a pair of two random variables $(\boldsymbol{X}_{i,a}^{l-c}, \boldsymbol{X}_{i,a'}^{l-c})$ is uniformly distributed over $\{0,1\}^{2m(l-c)}$.

(iii) $\boldsymbol{X}_{i,2a-1}^1 \neq \boldsymbol{X}_{i,2a}^1$ holds almost surely for any a.

Assumptions (i) and (ii) correspond to *weak randomness assumption* considered in [9]. Assumption (iii) asserts that any pair of candidates of the key is not identical.

3.2 Generalized PPS Algorithm

Let F_0 and F_1 be probability distributions of an observed symbol when the correct secret key bits are 0 and 1, respectively. In the following algorithms we compare likelihood of each candidate of the secret key. We call a criterion for choice of candidates *score*. As the score we use the log-likelihood ratio given by

$$R_i(\boldsymbol{x};\boldsymbol{y}) = \sum_{j=1}^{i}\sum_{k=1}^{m} R(\boldsymbol{x}[j][k];\boldsymbol{y}[j][k])\,, \qquad \boldsymbol{x}\in\{0,1\}^{mi},\, \boldsymbol{y}\in\mathbb{R}^{mi}$$

for a single-letter log-likelihood ratio

$$R(x;y) = \log\frac{\mathrm{d}F_x}{\mathrm{d}G}(y)\,, \qquad x\in\{0,1\},\, y\in\mathbb{R}\,, \tag{4}$$

where G is the mixture distribution $(F_0+F_1)/2$ and $\mathrm{d}F_x/\mathrm{d}G$ is the Radon-Nikodym's derivative. When F_0 and F_1 have probability densities f_0 and f_1, respectively, (4) is simply rewritten as

$$R(x;y) = \log\frac{f_x(y)}{g(y)}\,, \qquad x\in\{0,1\},\, y\in\mathbb{R}\,, \tag{5}$$

where $g(y) = (f_0(y)+f_1(y))/2$. We use (4) for a definition of a score since (4) always exists even in the case of discrete noises, which are considered in preceding researches [4, 5, 7, 9].

Let $X\in\{0,1\}$ be a random variable uniformly distributed over $\{0,1\}$. We define $Y\in\mathbb{R}$ as a random variable which follows distribution F_X given X. The mutual information between X and Y is denoted by

$$I(X;Y) = \mathrm{E}[R(X;Y)]\,.$$

Remark 1. $I(X;Y)$ is called a *symmetric capacity* for a channel F_x and is generally smaller than the channel capacity for asymmetric cases. We show in Theorem 1 that $I(X;Y)$, rather than the channel capacity, appears in the asymptotic bound. This corresponds to the fact that in our problem the distribution of the input symbol (i.e., the secret key) is fixed to be uniform and cannot be designed freely.

Now we discuss the following algorithm, which is generalized variant of PPS algorithm proposed in [9]. Note that the original PPS algorithm deals with only discrete noises; while the generalized variant can deal with continuous noises. This algorithm maintains a list $\mathcal{L}_r$, $|\mathcal{L}_r|\le L$, of candidates of the first tr bits of the secret key. We say that the recovery error occurred if the output $\mathcal{L}_{n/2t}$ does not contain the correct secret key. By abuse of notation each element of $\mathcal{L}_r$ denotes both a subsequence $\boldsymbol{x}_{tr,a}$ and its index a.

Now we bound the error probability of generalized PPS algorithm from above by the following theorem.

Theorem 1. Assume that

$$1/m < I(X;Y)\,. \tag{6}$$

Algorithm 1 Generalized PPS Algorithm

```
Input: Public keys $(N, e)$, observed data sequences $\bar{\mathbf{sk}}$
Output: Secret keys $\mathbf{sk}$.
Parameter: $t, L \in \mathbb{N}$.
Initialization: Set $\mathcal{L}_0 := \{1\}$.
Loop: For $r = 1, 2, \cdots, n/2t$ do the following.

  1. Expansion Phase  Generate list $\mathcal{L}'_r$ of all ancestors with depth $t$ from nodes in
     $\mathcal{L}_{r-1}$, that is,
         $\mathcal{L}'_r := \bigcup_{a \in \mathcal{L}_{r-1}} \{(a-1)2^t + 1,\, (a-1)2^t + 2, \cdots, a2^t\}$.

  2. Pruning Phase  If $2^{tr} \le L$ then set $\mathcal{L}_r := \mathcal{L}'_r$. Otherwise, set $\mathcal{L}_r$ to a subset of
     $\mathcal{L}'_r$ with size $L$ such that $R_{rt}(\boldsymbol{x}_{rt,a}; \boldsymbol{y}_{rt})$ are the largest so that for any $a \in \mathcal{L}_r$
     and $a' \in \mathcal{L}'_r \setminus \mathcal{L}_r$
         $R_{rt}(\boldsymbol{x}_{rt,a}; \boldsymbol{y}_{rt}) \ge R_{rt}(\boldsymbol{x}_{rt,a'}; \boldsymbol{y}_{rt})$.
     Here the tie-breaking rule is arbitrary.

Output of Loop: List of candidates $\mathcal{L}_{n/2t}$.
Finalization:  For each candidate in $\mathcal{L}_{n/2t}$, check whether the candidate is indeed
  a valid secret key with the help of public information.
```

Then, under generalized PPS algorithm it holds for any index a and parameters (t, L) that

$$\Pr[\boldsymbol{X}_{n/2,a} \notin \mathcal{L}_{n/2t} | \boldsymbol{X}_{n/2,a} \text{ is the correct secret key}] \le \frac{n}{2t}\rho_1 L^{-\rho_2}\,. \tag{7}$$

for some $\rho_1, \rho_2 > 0$ which only depend on c, m and F_x. Consequently, the error probability converges to zero as $L \to \infty$ for any $t > 0$.

The proof of Theorem 1 are given in the full version.

We evaluate the computational cost of generalized PPS algorithm. The costs of Expansion and Pruning phases in each loop are evaluated by $2L(2^t - 1)$ and $L2^t$. Since each phase is repeated $n/2t$ times, the whole cost of the Expansion phase and Pruning phase are given by $nL(2^t - 1)/t$ and $nL2^t/(2t)$, respectively.

This theorem shows that the error probability is bounded polynomially by L and t. Here note that the RHS of (7) cannot go to to zero for fixed L since $t \le n/2$ is required, whereas it goes to zero[3] for any fixed t as $L \to \infty$. Furthermore, the complexity grows exponentially in t whereas it is linear in L. From these observations we can expect that the generalized PPS algorithm performs well for small t and large L.

[3] In the theoretical analysis in [9], it seems to be implicitly assumed that the score of the first $mt(r-1)$ bits is discarded at each r-th loop, that is $R_{mt}(\boldsymbol{X}^t_{tr,i}; \boldsymbol{y}^t_{tr})$ is considered instead of $R_{mtr}(\boldsymbol{X}_{tr,i}; \boldsymbol{y}_{tr})$. In this case we require $t \to \infty$ to assure that the error probability approaches zero.

Remark 2. It is claimed in [9] for the case of binary observations that the error probability of PPS algorithm goes to zero as $t \to \infty$ for any fixed L oppositely to the above argument. This gap does not mean that the bound (7) is loose but comes from an inappropriate argument in [9]. In fact, we can prove that the error probability never vanishes for any fixed L as shown in Appendix A.

3.3 Implications: Continuous Distributions

Informally, Theorem 1 states that we can recover the secret key with high probability if $I(X;Y) > 1/m$. The actual values of $I(X;Y)$ depends on the distribution F_x. We evaluate the value of $I(X;Y)$ for some continuous distribution F_x which has density $f_x(y)$.

First, we introduce a differential entropy.

Definition 1. The differential entropy $h(f)$ of a probability density function f is defined as

$$h(f) = -\int_{-\infty}^{\infty} f(y)\log f(y)\mathrm{d}y.$$

We give some properties of the differential entropy [1]. Let f be an arbitrary probability density with mean μ and variance σ^2. Then it is shown in [1, Theorem 8.6.5] that

$$h(f) \le h(\mathcal{N}(\mu,\sigma^2)) = \log\sqrt{2\pi \mathrm{e}\sigma^2}\,, \tag{8}$$

where $\mathcal{N}(\mu,\sigma^2)$ is the density of Gaussian distribution with mean μ and variance σ^2.

The symmetric capacity $I(X;Y)$ can be expressed as $h(g)-(h(f_0)+h(f_1))/2$ for $g(y) = (f_0(y)+f_1(y))/2$ since

$$\begin{aligned} I(X;Y) &= \sum_{x\in\{0,1\}}\int \frac{f_x(y)}{2}\log\frac{f_x(y)}{\sum_{x'\in\{0,1\}}\frac{f_{x'}(y)}{2}}\mathrm{d}y = \sum_{x\in\{0,1\}}\int\frac{f_x(y)}{2}\log\frac{f_x(y)}{g(y)}\mathrm{d}y \\ &= \sum_{x\in\{0,1\}}\int\frac{f_x(y)}{2}\log f_x(y)\mathrm{d}y - \int g(y)\log g(y)\mathrm{d}y = h(g) - \frac{h(f_0)+h(f_1)}{2}. \end{aligned}$$

Next, we further assume that the distributions are symmetric: $f_1(y) = f_0(\alpha - y)$ for some α. Since the differential entropy is invariant under translation, we have $h(f_1) = h(f_0)$ and thus $I(X;Y) = h(g) - h(f_0)$ if the distributions are symmetric. A typical example of the symmetric distribution is *symmetric additive noise*: the sample can be written as the sum of a deterministic part and a symmetric random noise part.

Summing up the above discussion, we have the following corollary.

Corollary 1. Assume that F_x has a probability density f_x. Then the error probability of generalized PPS algorithm converges to zero as $L \to \infty$ if

$$h(g) - \frac{h(f_0)+h(f_1)}{2} > \frac{1}{m}.$$

Further assume that the $f_0(y)$ and $f_1(y)$ are symmetric. In this case, the condition is expressed as $h(g) - h(f_0) > 1/m$.

Gaussian Distribution We remind readers of the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$. The density function of this distribution is $f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$, where μ and σ^2 are the mean and variance of the distribution, respectively.

The most standard setting of a continuous noise is an additive white Gaussian noise (AWGN): the density f_x of distribution F_x is represented by

$$f_x(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-(-1)^x)^2}{2\sigma^2}\right) . \tag{9}$$

Note that the expectation of $f_0(x)$ and $f_1(x)$ are $+1$ and -1, respectively. In this case the score is represented by

$$\begin{aligned} R_i(\boldsymbol{x};\boldsymbol{y}) &= \sum_{j=1}^{i}\sum_{k=1}^{m} \frac{(-1)^{\boldsymbol{x}[j][k]}\boldsymbol{y}[j][k]}{2(\ln 2)\sigma^2} \\ &- \sum_{j=1}^{i}\sum_{k=1}^{m} \left(\frac{\boldsymbol{y}[j][k]^2+1}{2(\ln 2)\sigma^2} + \log \frac{\exp(-\frac{(\boldsymbol{y}[j][k]-1)^2}{2\sigma^2}) + \exp(-\frac{(\boldsymbol{y}[j][k]+1)^2}{2\sigma^2})}{2} \right) . \end{aligned} \tag{10}$$

The symmetric capacity for F_x is given by

$$I(X;Y) = h(g) - h(f_0) = h(g) - \log\sqrt{2\pi e \sigma^2} . \tag{11}$$

It is equivalent to use a score

$$\sum_{j=1}^{i}\sum_{k=1}^{m} (-1)^{\boldsymbol{x}[j][k]}\boldsymbol{y}[j][k] \tag{12}$$

instead of (10) in generalized PPS algorithm since the factor $(2(\ln 2)\sigma^2)^{-1}$ and the second term of (10) are common to all candidates $\boldsymbol{x}_{i,a}$.

The following corollary is straightforward from the computation of $I(X;Y)$ for the Gaussian case.

Corollary 2. Assume that f_x is the Gaussian distribution given in Eq. (9). Then the error probability of generalized PPS algorithm converges to zero as $L \to \infty$ if $\sigma < 1.767$ when $m = 5$ and if $\sigma < 1.297$ when $m = 3$ and if $\sigma < 0.979$ when $m = 2$.

Proof. Regarding $h(f)$, we have $h(f) = \log\sqrt{2\pi e\sigma^2}$ from (8). The differential entropy $h(g)$ of the mixture distribution g is not given in explicit form but numerical calculation shows that $h(g) - h(f) = 1/5, 1/3, 1/2$ for $\sigma = 1.767, 1.297, 0.979$, respectively. Thus the corollary follows immediately from (11) and Theorem 1. □

3.4 Implications: Discrete Distribution

Next, we discuss the discrete distribution cases. As an example, we consider the binary symmetric error. In the case that F_x is a discrete distribution on $\{0, 1\}$ such that

$$\begin{cases} F_0(\{0\}) = F_1(\{1\}) = 1 - p\,, \\ F_0(\{1\}) = F_1(\{0\}) = p\,, \end{cases}$$

for some $0 < p < 1/2$, we have

$$R_i(\boldsymbol{x}; \boldsymbol{y}) = mi \log(2(1-p)) + d_{\mathrm{H}}(\boldsymbol{x}, \boldsymbol{y}) \log \frac{p}{1-p}\,, \boldsymbol{x} \in \{0,1\}^{mi},\ \boldsymbol{y} \in \mathbb{R}^{mi}\,, \quad (13)$$

where $d_{\mathrm{H}}(\boldsymbol{x}, \boldsymbol{y})$ is the Hamming distance between $\boldsymbol{x}$ and $\boldsymbol{y}$. Note that it is equivalent to use a score $d_{\mathrm{H}}(\boldsymbol{x}; \boldsymbol{y})$ instead of (13) in generalized PPS algorithm since the factor $mi \log(p/(1-p))$ and the constant $\log(2(1-p))$ are common to all candidates $\boldsymbol{x}_{i,a}$ and do not change the order of scores.

In this case the capacity is given by $I(X; Y) = 1 - h_2(p)$ for the binary entropy function $h_2(p) = -p \log p - (1-p) \log(1-p)$.

3.5 Discussion: Comparison with Quantization-based Approach

The simplest algorithm for our key-recovery problem would be a quantization-based algorithm. Here, we focus on the AWGN setting introduced in Section 3.3. First, we consider the following simple quantization. If the observed value is positive, its corresponding bit is converted to 0 and if it is negative, it is converted to 1. Then, the binary sequence of the secret key with error is obtained by quantization. The HMM algorithm [4] is applied to the obtained noisy secret key sequences. A simple calculation shows that we can recover the secret key if the noise follows the Gaussian distribution and $\sigma < 1.397$ when $m = 5$.

Next, we consider a more clever quantization rule, which uses the “erasure symbol”. Let D be a positive threshold; then the quantization rule is given as follows. The corresponding is converted to 0 if $x \geq D$; 1 if $x < -D$; “?” if $-D \leq x < D$. The binary sequence of the secret key with error and erasure is obtained by quantization. Kunihiro et al.’s algorithm [7] is applied to the obtained noisy secret key sequences. A simple calculation shows that we can recover the secret key if the noise follows the Gaussian distribution and $\sigma < 1.533$ when $m = 5$ under optimally chosen D.

As shown in Corollary 2, our proposed algorithm works well if $\sigma < 1.767$ when $m = 5$ and is significantly superior to the quantization-based algorithms. The reason for this is that generalized PPS algorithm uses all the information of the observed data whereas the quantization-based algorithms ignore the value of observed data after quantization and thus suffer from quantization errors. A consequence is that we can improve the bound of σ.

4 DPA-like Algorithm: Unknown Distribution Case

The generalized PPS algorithm works only if the explicit form of F_x is known since it is needed in the calculation of $R_{rt}(\boldsymbol{x}_{rt,a}; \boldsymbol{y}_{rt})$. However, in many actual attack situations, the explicit form of noise distribution is not known. In this section, we propose an effective algorithm that works well even if these explicit forms are unknown.

4.1 DPA-like Algorithm

Consider the case that we only know the expectation of F_0 and F_1, which we assume without loss of generality to be $+1$ and -1, respectively. In this case it is natural to use (12) as a score from the viewpoint of DPA analysis [6] instead of $R_{mi}(\boldsymbol{x}, \boldsymbol{y})$ itself. We define DPA function as follows.

$$\mathbf{DPA}_{mi}(\boldsymbol{x}; \boldsymbol{y}) := \sum_{j=1}^{i} \sum_{k=1}^{m} (-1)^{\boldsymbol{x}[j][k]} \boldsymbol{y}[j][k]. \tag{14}$$

Note that this score can be calculated without knowledge of the specific form of the noise distribution. In the case that $\mathbf{DPA}_{mi}$ is used as a score, the bound in Theorem 1 is no more achievable for distributions other than Gaussians but a similar bound can still be established.

Theorem 2. *Assume that F_x has a probability density f_x. Under generalized PPS algorithm with score function (14) the error probability converges to zero as $L \to \infty$ if*

$$\frac{1}{m} < h(g) - \log \sqrt{\pi \mathrm{e}(\sigma_0^2 + \sigma_1^2)},$$

where $g(y) = (f_0(y) + f_1(y))/2$ and $\sigma_x^2 = \mathrm{Var}(f_x)$ is the variance of distribution f_x.

The proof of Theorem 2 is almost the same as that of Theorem 1 and given in the full version.

Note that $I(X;Y)$ can be expressed as $h(g) - (h(f_0) + h(f_1))/2$. Thus, in view of Theorems 1 and 2, information loss of the score (14) can be expressed as $\log \sqrt{\pi \mathrm{e}(\sigma_0^2 + \sigma_1^2)} - (h(f_0) + h(f_1))/2$, which is always nonnegative since

$$\begin{aligned}\log \sqrt{\pi \mathrm{e}(\sigma_0^2 + \sigma_1^2)} - \frac{h(f_0) + h(f_1)}{2} &\geq \log \sqrt{\pi \mathrm{e}(\sigma_0^2 + \sigma_1^2)} - \frac{\log \sqrt{2\pi \mathrm{e} \sigma_0^2} + \log \sqrt{2\pi \mathrm{e} \sigma_1^2}}{2} \\ &\geq 0\,,\end{aligned}$$

where the first and second inequalities follow from (8) and from the concavity of the log function, respectively. This loss becomes zero if and only if f_0 and f_1 are Gaussians with the same variances.

Further assume that f_0 and f_1 are symmetric. In this case, it holds that $\sigma_1^2 = \sigma_0^2$ and $h(f_1) = h(f_0)$. Thus information loss of the score (14) can be expressed as $\log \sqrt{2\pi \mathrm{e} \sigma_0^2} - h(f_0) = h(\mathcal{N}(0, \sigma_0^2)) - h(f_0)$, which increases as the true noise distribution deviates from Gaussian.

Remark 3. We need not to know the expectation of F_0 and F_1 in practice. To proceed the argument in this section it is sufficient to know the intermediate value of these expectations, that is, the expectation of $G = (F_0 + F_1)/2$. Since it is the expectation of the observed values, we can estimated it accurately by averaging all the elements of the observation $\boldsymbol{y}$.

4.2 Connection to DPA

We briefly review DPA. It was proposed as a side channel attack against DES by Kocher et al. [6] and was then generalized to other common secret key encryption. It derives the secret key from many physically measured waveforms of power consumption. They introduced the DPA selection function whose input includes a guessed secret key. In attacking phase, an attacker guesses the secret key and calculate the difference between the average of the waveforms of power consumption for which the value of the DPA selection function is one and it is zero. If the guess is incorrect, the waveforms are cancelled since they are uncorrelated. Then, the resulting waveform becomes flat. However, if the guess is correct, they are correlated and the resulting waveform has spikes. By observing the difference, we can find the correct key.

We introduce two sets: $\mathcal{S}_1 = \{(j,k) | \boldsymbol{x}[j][k] = 1\}$ and $\mathcal{S}_0 = \{(j,k) | \boldsymbol{x}[j][k] = 0\}$. The function **DPA** can be transformed as follows:

$$\mathbf{DPA}_{mi}(\boldsymbol{x}; \boldsymbol{y}) = \sum_{(j,k)\in\mathcal{S}_0} \boldsymbol{y}[j][k] - \sum_{(j,k)\in\mathcal{S}_1} \boldsymbol{y}[j][k],$$

which is similar to the DPA selection function used in DPA [6].

We give an intuitive explanation of how the algorithm works. Without loss of generality, we assume that $\mathrm{E}(f_0) = +1$ and $\mathrm{E}(f_1) = -1$. We consider two cases that the candidate solution is correct and incorrect in the following. First, assume that the candidate solution is correct. If the bit in the candidate solution is 0, the observed value follows $f_0(y)$; if it is 1, the observed value follows $f_1(y)$ and then the negative of the observed value can be seen to be output according to $f_1(-y)$. The means of both $f_0(y)$ and $f_1(-y)$ are $+1$. Hence, the expectation of Eq. (14) is mi if the candidate solution is correct. Hence, the score calculated by Eq. (14) is close to mi with high probability. Second, Next, assume that the candidate solution is incorrect. In this case, the observed value is output according to the mixture distribution $g(y) = (f_1(y) + f_0(y))/2$ which has zero mean. Hence, the expectation of Eq. (14) is 0 if the the candidate solution is incorrect and the score calculated by Eq. (14) is close to 0 with high probability. Thus, if the score is high enough (that is, the score is around mi), the estimation is correct, whereas if the score is low enough (that is, the score is around 0), the estimation is incorrect and such a node will be discarded. We give a toy example on the function **DPA** in Appendix B for a better understanding.

4.3 Connection to Communication Theory

The problem of RSA secret key recovery is strongly related to the communication theory. Each candidate of the secret key corresponds to a codeword of a message and the attacker corresponds to the receiver trying to estimate the message from the sent codeword distorted by noise. The key estimation after quantization process in [4, 5, 7, 9] is called a hard-decision decoding and the proposed algorithm exploiting full information of the observed data is called a soft-decision decoding.

The structure of the secret key characterized by Hensel lift can be regarded as a convolutional code with infinite constraint length. It is known that Viterbi algorithm works successfully for convolutional codes with small constraint length and many algorithms such as Fano algorithm and stack algorithm have been proposed for codes with large constraint length [13, 14]. Thus one can expect that these algorithms for large constraint length perform well also for the problem of secret key recovery.

However there also exists a gap between the settings of the secret key recovery and the communication theory. In the case of message transmission, the noise ratio is set to be relatively small because the error probability has to be negligibly small value, say, e.g., 10^{-8}. On the other hand when we consider security of the secret key, 10% success rate of the recovery is critical and we have to consider the success probability under large noise ratio. For this gap it is not obvious whether the above algorithms work successfully for the secret key recovery.

5 Implementation and Experiments

We have implemented generalized PPS algorithm. In our experiments on 1024, 2048-bit RSA, we prepared 100 (or 200 if the success rate is less than 0.1) different tuples of secret keys $\mathbf{sk}$, e.g., $\mathbf{sk} = (p, q, d, d_p, d_q)$. We generated the Gaussian noisy output $\overline{\mathbf{sk}}$ for each $\mathbf{sk}$. In our experiments, the incorrect candidate solution is randomly generated based on Assumption 1 with $c = 1$.

The experimental results for $n = 1024$ and $\mathbf{sk} = (p, q, d, d_p, d_q)$ are shown in Figure 1. We set $(t, L) = (1, 2^{11}), (2, 2^{11}), (4, 2^{10}), (8, 2^7), (16, 2^0)$, which makes the computational cost for Pruning phase to be equal: 2^{21}.

As can be seen in Figure 1, if $\sigma \leq 1.3$, the success rate for small t (say, $t = 1, 2, 4$) is almost 1. Generalized PPS algorithm can succeed to recover the correct secret key with probability larger than 0.1 for $\sigma \leq 1.7$; while it almost fails to recover the key for $\sigma \geq 1.8$. This results match with theoretically predicted bound $\sigma = 1.767$. Figure 1 also shows generalized PPS algorithm fails with high probability if we use a small L, (say $L = 1$); while the computational cost is almost the same as the setting $(t, L) = (1, 2^{11})$. This fact is reinforced by Theorem 3 shown in Appendix A.

Figure 1 suggests that the setting $t = 1$ is enough for gaining high success rates. We then present experimental results for $t = 1$ and $n = 1024, 2048$ in Figures 2 and 3. Figures 2 and 3 show that the success rates for any n and σ will significantly increase if we use a larger list size L. For each bit size n, generalized PPS algorithm almost always succeed to find the secret key if $\sigma \leq 1.3$. The generalized PPS algorithm still has a non-zero success rate for σ as large as 1.7.

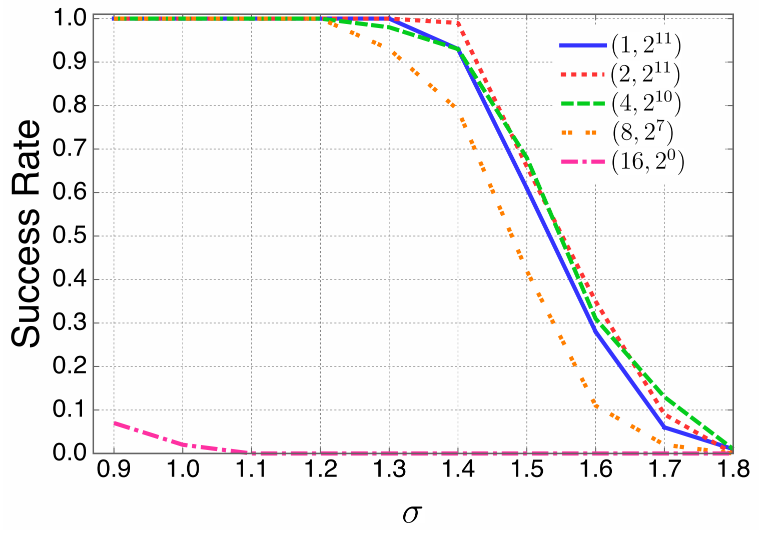

Fig. 1. Experiments for $\mathbf{sk} = (p, q, d, d_p, d_q), n = 1024$

In communication theory, there are many techniques such as stack algorithm [13]. We do not implement such techniques in our experiments. It is possible to increase the success rate if we implement them together in our algorithm.

6 Concluding Remarks

In this paper, we showed that we can recover the secret key if Eq. (6) holds; the symmetric capacity $I(X;Y)$ is larger than $1/m$. As mentioned in [9], the success condition for key-recovery from noisy secret keys has a strong connection to channel capacity. As explained in Remark 1, symmetric capacity is less than channel capacity for asymmetric distribution cases. Hence, one might wonder if the condition can be improved. Unfortunately, it is hopeless to improve the bound since the distribution of input symbol (i.e., the correct secret key) is fixed to uniform in our problem whereas the input distribution is optimized to achieve the channel capacity in coding theory. Then, from the coding theoretic viewpoint, the condition $I(X;Y) > 1/m$ is optimal for our problem.

Acknowledgement

The first author was supported by JSPS Grant Number KAKENHI 25280001.

References

1. C. M. Cover and J. A. Thomas, "*Elements of Information Theory*, 2nd Edition," Wiley-Interscience, 2006.

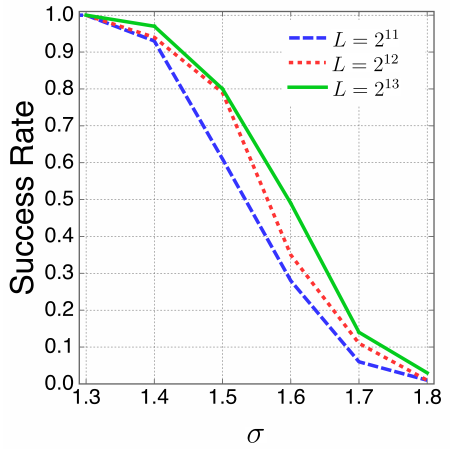

Fig. 2. $m = 5$, $n = 1024$ and $t = 1$

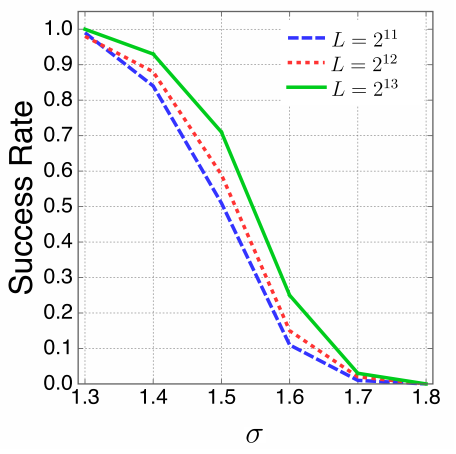

Fig. 3. $m = 5, n = 2048$ and $t = 1$

2. A. Dembo and O. Zeitouni, *Large deviations techniques and applications*, 2nd ed., ser. Applications of Mathematics. New York: Springer-Verlag, 1998, vol. 38.
3. 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," in Proc. of USENIX Security Symposium 2008, pp. 45–60, 2008.
4. W. Henecka, A. May, and A. Meurer, "Correcting Errors in RSA Private Keys," in Proc. of Crypto 2010, LNCS 6223, pp. 351–369, 2010.
5. N. Heninger and H. Shacham, "Reconstructing RSA Private Keys from Random Key Bits," in Proc. of Crypto 2009, LNCS 5677, pp. 1–17, 2009.
6. P. Kocher, J. Jaffe and B. Jun, "Differential Power Analysis," in Proc. of CRYPTO'99, LNCS 1666, pp.388-397, 1999.
7. N. Kunihiro, N. Shinohara and T. Izu, "Recovering RSA Secret Keys from Noisy Key Bits with Erasures and Errors," in Proc. of PKC 2013, LNCS 7778, pp. 180-197, 2013.
8. S. Mangard, E. Oswald and F. -X. Standaert, "One for all - all for one: unifying standard differential power analysis attacks," IET Information Security, vol. 5, no. 2, pp. 100–110, 2011.
9. K. G. Paterson, A. Polychroniadou and D. L. Sibborn, "A Coding-Theoretic Approach to Recovering Noisy RSA Keys," in Proc. of Asiacrypt 2012, LNCS 7658, pp. 386–403, 2012.
10. PKCS #1 Standard for RSA. Available at `http://www.rsa.com/rsalabs/node.asp?id=2125`.
11. R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, vol. 21(2), pp. 120–126, 1978.
12. S. Sarkar and S. Maitra, "Side Channel Attack to Actual Cryptanalysis: Breaking CRT-RSA with low weight decryption exponents," in Proc. of CHES2012, LNCS7428, pp. 476–493, 2012.
13. C. Schlegel and L. Perez, "Trellis and Turbo Codes," Wiley-IEEE Press, 2004.

14. B. Sklar, “Digital Communications: Fundamentals and Applications, 2nd Edition,” Prentice Hall, 2001.
15. S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage, “When Private Keys are Public: Results from the 2008 Debian OpenSSL Vulnerability,” IMC 2009, ACM Press, pp. 15–27, 2009.

A Lower bound on Error Probability of Generalized PPS Algorithm

As mentioned in Remark 2, the upper bound (7) on the error probability of PPS algorithm cannot go to zero for fixed L even though it is claimed in [9] that the error probability vanishes as t increases for any fixed L. We show in this appendix that $L \to \infty$ is actually necessary to achieve an arbitrary small error probability.

Let $p_{\mathrm{e}} = \Pr[R(1-X;Y) > R(X;Y)]$ be the single-letter error probability under decoding $\hat{X} := \mathrm{argmax}_{x\in\{0,1\}}\{R(x;Y)\}$. Note that $p_{\mathrm{e}} = 0$ is a degraded case in which each bit X can be recovered from Y without error by the above decoding rule. We can bound the error probability of PPS algorithm from below in a simple form by using p_{e}.

Theorem 3. Under generalized PPS algorithm it holds for any index a and parameters (t, L) that

$$\Pr[\boldsymbol{X}_{n/2,a} \notin \mathcal{L}_{n/2t} | \boldsymbol{X}_{n/2,a} \text{ is the correct secret key}] \ge p_{\mathrm{e}}^{m(1+\log L)} . \tag{15}$$

Consequently, the error probability does not go to zero as $t \to \infty$ with a fixed L if $p_{\mathrm{e}} > 0$.

We can easily see that the error probability does not vanish for a fixed L by the following argument. For simplicity let us consider the case that the correct secret key is $\boldsymbol{X}_{n/2,1}$. Then the candidate $\boldsymbol{X}_{n/2,2}$ is identical to the correct key except for the last m bits. Similarly, $\boldsymbol{X}_{n/2,3}$ and $\boldsymbol{X}_{n/2,4}$ are identical to the correct key except for the last $2m$ bits. Thus, once the last $2m$ observed symbols $\boldsymbol{y}_{n/2}^2$ become very noisy (the probability of this event does not depend on t) then the likelihood of $\boldsymbol{X}_{n/2,b}$ for $b = 2, 3, 4$ exceeds that of the correct key $\boldsymbol{X}_{n/2,1}$, and the recovery error occurs when the list size is $L \le 3$. This argument always holds when the list size L is fixed and we see that the error probability heavily depends on L.

Proof of Theorem 3. Recall that $l_t = \lfloor (\log L)/t \rfloor + 1$. Since $\boldsymbol{X}_{n/2,1} \in \mathcal{L}_{n/2t}$ implies $\boldsymbol{X}_{n/2,1}[1 : tl_t] = \boldsymbol{X}_{tl_t,1} \in \mathcal{L}_{l_t}$, we have

$$\begin{aligned}\Pr[\boldsymbol{X}_{n/2,1} \notin \mathcal{L}_{n/2t}] &\ge \Pr\left[\boldsymbol{X}_{tl_t,1} \notin \mathcal{L}_{l_t}\right] \\ &\ge \Pr\left[\sum_{a=2}^{2^{tl_t}} \mathbb{1}\left[R_{tl_t}(\boldsymbol{X}_{tl_t,a};\boldsymbol{Y}_{tl_t}) > R_{tl_t}(\boldsymbol{X}_{tl_t,1};\boldsymbol{Y}_{tl_t})\right] \ge L\right] .\end{aligned}$$

For $\bar{l} = \lfloor \log L \rfloor + 1$, we have $\bar{l} \le tl_t$ and

$$\begin{aligned}
\Pr[\boldsymbol{X}_{n/2,1} \notin \mathcal{L}_{n/2t}] &\ge \Pr\left[\sum_{a=2}^{2^{\bar{l}}} \mathbb{1}[R_{tl_t}(\boldsymbol{X}_{tl_t,a};\boldsymbol{Y}_{tl_t}) > R_{tl_t}(\boldsymbol{X}_{tl_t,1};\boldsymbol{Y}_{tl_t})] \ge L\right] \\
&= \Pr\left[\sum_{a=2}^{2^{\bar{l}}} \mathbb{1}\left[R_{\bar{l}}(\boldsymbol{X}^{\bar{l}}_{tl_t,a};\boldsymbol{Y}^{\bar{l}}_{tl_t}) > R_{tl_t}(\boldsymbol{X}^{\bar{l}}_{tl_t,1};\boldsymbol{Y}^{\bar{l}}_{tl_t})\right] \ge L\right] \qquad \text{(by (3))} \\
&\ge \Pr\left[\sum_{a=2}^{2^{\bar{l}}} \mathbb{1}\left[R_{\bar{l}}(\boldsymbol{X}^{\bar{l}}_{tl_t,a};\boldsymbol{Y}^{\bar{l}}_{tl_t}) > R_{tl_t}(\boldsymbol{X}^{\bar{l}}_{tl_t,1};\boldsymbol{Y}^{\bar{l}}_{tl_t})\right] = 2^{\bar{l}} - 1\right] . \quad (16)
\end{aligned}$$

Now consider the case that $R(1-\boldsymbol{X}_{tl_t,1}[j][k];\boldsymbol{Y}_{tl_t}[j][k]) > R(\boldsymbol{X}_{tl_t,1}[j][k];\boldsymbol{Y}_{tl_t}[j][k])$ for all $j = tl_t - \bar{l} + 1, \cdots, tl_t$ and $k = 1, \cdots, m$. Then $R_{\bar{l}}(\boldsymbol{x};\boldsymbol{Y}^{\bar{l}}_{tl_t}) > R_{\bar{l}}(\boldsymbol{X}^{\bar{l}}_{tl_t,1};\boldsymbol{Y}^{\bar{l}}_{tl_t})$ for all $\boldsymbol{x} \ne \boldsymbol{X}^{\bar{l}}_{tl_t,1}$. As a result, (16) is bounded as

$$\begin{aligned}
&\Pr[\boldsymbol{X}_{n/2,1} \notin \mathcal{L}_{n/2t}] \\
&\ge \Pr\left[\bigcap_{j=tl_t-\bar{l}+1}^{tl_t} \bigcap_{k=1}^{m} \{R(1-\boldsymbol{X}_{tl_t,1}[j][k];\boldsymbol{Y}_{tl_t}[j][k]) > R(\boldsymbol{X}_{tl_t,1}[j][k];\boldsymbol{Y}_{tl_t}[j][k])\}\right] \\
&= p_{\mathrm{e}}^{m\bar{l}} \ge p_{\mathrm{e}}^{m(1+\log L)} .
\end{aligned}$$

and we complete the proof.

Remark 4. In the theoretical analysis of Peterson et al. [9], they compared scores between the correct secret key and a subset of $\mathcal{L}'_r$ with size L randomly chosen from $\mathcal{L}'$. However, in the actual algorithm all elements of $\mathcal{L}'_r$ are scanned and such an analysis based on the random choice does not have validity, which led to the conclusion contradicting Theorem 3.

B A Toy Example for Generalized PPS Algorithm with (14)

To better understand the algorithm, we present a toy example. Suppose that the correct solution is 1100010011 and that we observed the data sequence as $\boldsymbol{y} = (-3, -2, +2, +3, +3, -3, +1, +4, -3, -2)$. Attackers know the observed data; but, do not know the correct solution. Assume that we know $\mathrm{E}(f_0) = 3$ and $\mathrm{E}(f_1) = -3$. Suppose that we have three candidate sequences: $\boldsymbol{x}_1 = (1100010011)$, $\boldsymbol{x}_2 = (1001110010)$ and $\boldsymbol{x}_3 = (0101011001)$.

The score for $\boldsymbol{x}_1 = (110001001)$ is given by $\mathbf{DPA}(\boldsymbol{x}_1;\boldsymbol{y}) = 3+2+2+3+3+3+1+4+3+2 = 26$. Since the value 26 is close to $3 \times 10 = 30$, the candidate seems to be correct.

The score for $\boldsymbol{x}_2 = (1001110010)$ is given by $\mathbf{DPA}(\boldsymbol{x}_2;\boldsymbol{y}) = 3-2+2-3-3+3+1+4+3-2 = 6$. The value 6 is close to 0. The score for $\boldsymbol{x}_3 = (0101011001)$ is given by $\mathbf{DPA}(\boldsymbol{x}_3;\boldsymbol{y}) = -3+2+2-3+3+3-1+4-3+2 = 2$. The value 2 is close to 0. Thus these candidates seem to be incorrect.