SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

March 2018, Vol. 61 032111:1–032111:17
https://doi.org/10.1007/s11432-017-9210-3

Similar operation template attack on RSA-CRT as a case study

Sen XU[1], Xiangjun LU[1], Kaiyu ZHANG[1], Yang LI[2*], Lei WANG[1], Weijia WANG[1], Haihua GU[3], Zheng GUO[1], Junrong LIU[1] & Dawu GU[4,1*]

[1]*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai* 200240*, China;*
[2]*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing* 211106*, China;*
[3]*Wanda Internet Technology Group, Shanghai* 200127*, China;*
[4]*Shanghai Institute for Advanced Communication and Data Science, Shanghai* 200241*, China*

Received 13 May 2017/Revised 5 July 2017/Accepted 19 July 2017/Published online 18 January 2018

Abstract A template attack, the most powerful side-channel attack methods, usually first builds the leakage profiles from a controlled profiling device, and then uses these profiles to recover the secret of the target device. It is based on the fact that the profiling device shares similar leakage characteristics with the target device. In this study, we focus on the similar operations in a single device and propose a new variant of the template attack, called the similar operation template attack (SOTA). SOTA builds the models on public variables (e.g., input/output) and recovers the values of the secret variables that leak similar to the public variables. SOTA's advantage is that it can avoid the requirement of an additional profiling device. In this study, the proposed SOTA method is applied to a straightforward RSA-CRT implementation. Because the leakage is (almost) the same in similar operations, we reduce the security of RSA-CRT to a hidden multiplier problem (HMP) over GF(q), which can be solved byte-wise using our proposed heuristic algorithm. The effectiveness of our proposed method is verified as an entire prime recovery procedure in a practical leakage scenario.

Keywords side channel attack, template attack, RSA-CRT, hidden number problem, prime recovery

Citation Xu S, Lu X J, Zhang K Y, et al. Similar operation template attack on RSA-CRT as a case study. Sci China Inf Sci, 2018, 61(3): 032111, https://doi.org/10.1007/s11432-017-9210-3

1 Introduction

In the field of side-channel attacks (SCAs), the seminal differential power analysis (DPA) method was proposed by Kocher et al. [1]. Then, researchers proposed many SCA methods, leading to a central division between non-profiled and profiled attacks. The former attack methods are based on a comparison of actual leakages and a prior leakage model. Examples include correlation power analysis (CPA) [2], mutual information analysis (MIA) [3], and differential clustering analysis (DCA) [4] methods. In the profiled analysis methods, a leakage model is built from the profiling devices that are under adversaries' control, which implicitly relies on the fact that the leakage characteristics of the profiling and target devices are similar. Generally, the origin of the leakage similarity (of profiling and under test devices) is in the similar hardware behaviors or structures. The Gaussian template attack (TA) [5], which is

* Corresponding author (email: liyang_uec@163.com, dwgu@sjtu.edu.cn)

operated by estimating the Gaussian probability density function of the side channel leakages, is the representative attack method. Although it is not yet a perfect model for estimation, the TA can still be viewed as the most powerful type of SCA.

A large variety of public key cryptosystems (PKC) are employed to protect the security of sensitive assets. Typically, a PKC scheme is based on hard mathematical problems, such as a large number factorization problem or discrete logarithm problem, which hereafter are referred to as primitives. In general, a specific PKC scheme implementation includes (informally) two parts: a primitive and a combination phase. The former part usually contains modular exponentiation or scalar multiplication, e.g., RSA or elliptic curve cryptography (ECC). The latter part obtains the final results (encryption, decryption, digital signature, or key exchange) based on the former part's results. The combination of the two parts varies according to different schemes. Both parts can be threatened by SCAs. The attack methods can extract the secret cryptographic key or intermediate values, including power consumption, electromagnetic emissions, timing, or faults collected from a running cryptographic device, by using statistical tools operating on side channel leakages. Many studies have provided practical attack results on embedded devices [6–8]. Recently, papers have been published that describe attack procedures on specific PKC schemes implemented on PCs [9–11] and mobile phones [12, 13]. In the last paper, the SCA and lattice attack were combined to reveal the secret key of elliptic curve digital signature algorithm (ECDSA) implementation in the most recent OpenSSL. The attack technique can also be employed to analyze the security of RSA given a known partial secret prime. A typical technique is Coppersmith's method [14]. The method breaks RSA using half the most significant bytes of a secret prime by applying a lattice attack. Therefore, half the most significant bytes of a prime are fatal in the case of an RSA implementation.

Researchers are dedicated to constructing secure primitive implementations to mitigate SCAs. The first step is to counteract simple power analysis (SPA), which can obtain the secret parameters involved in primitives through different computational patterns, such as modular multiplication and modular squaring in a modular exponentiation or point addition and point doubling in scalar multiplication. Typical SPA-resistant methods are the Montgomery powering ladder [15], atomic implementation [16], and dummy operations [17]. The second step is to thwart DPA. This type of countermeasure includes exponentiation blinding (splitting) and message blinding. In general, SCA-secure primitives, such as those presented in [18], usually combine both types of countermeasures. Attention has seldom been paid to the combination phase of a PKC scheme, even when a CPA attack can be mounted [6, 19]. In [20], the authors provided an SPA attack on modular inversion implemented with an extended Euclidean algorithm, which is further evidence that a secure primitive does not ensure a secure implementation.

The requirement of an identical profiling device is a strong assumption in a practical TA. In this study, we focus on the similarity between two similar operations in the PKC scheme, especially in the combination phase. We propose the similar operation TA (SOTA, subtraction in [21]) in the combination phase of RSA-CRT implementation. In our attack scenario, we can mitigate the requirement of an additional profiling device by construing templates on the public information, based on which we find a new means of analyzing the security of the RSA-CRT.

Contributions. Our work is based on the intuition that similar operations share similar leakage, which is confirmed by our experimental results. First, we present a general attack method named SOTA to exploit the similar leakage of similar operations through side channel leakage. We stress that the effectiveness of SOTA relies on the preprocessing procedure employed and the TA methods.

Second, we observe that there still may exist a difference in the similar operations' side channel leakage characteristics with the same leakage model. This difference negatively affects the SOTA results when the adversary utilizes raw power traces. A preprocessing procedure named zero-mean [22] is employed to unify raw leakage to achieve better performance. SOTA is effective because the templates are built on public information and do not require an additional profiling device. Our results show that the zero-mean method is suitable for situations involving both cross-device and similar operations.

Finally, we find that the SCA against RSA-CRT can be reduced to solving a hidden multiplier problem (HMP) over GF(q). We propose a heuristic algorithm that, after SOTA recovers secret intermediate data, solves the problem by recovering hidden primes byte-by-byte. We applied the proposed attack procedure to an RSA-CRT software implementation in a practical Hamming weight leakage scenario where both the SOTA and the prime recover algorithm are effective. In our experiments, we also considered error matching of secret intermediate data bytes, which means that in the SOTA these bytes' Hamming weight may randomly be categorized into adjacent values with a certain probability. We can recover the hidden prime through 100 inputs with only the Hamming weight even when 50% of the inputs contain noise.

Our work provides a new technique for RSA-CRT security analysis, which is based on the idea of using similar operations instead of constructing a profile using an additional identical device. We can reveal partial information about secret intermediate data by focusing on the data transferal instead of the primitive implementation. Then, a hidden prime can be revealed by solving the HMP. To the best of our knowledge, no previous studies have been published that used a similar attack procedure. Therefore, this study makes a novel contribution to the academic literature. Our experiment shows that the efficiency of our attack is similar to that of existing methods, because SOTA exploits leakage characteristics that are similar to those used in these methods. The final hidden prime recovery requires no additional power traces, and its execution time is practical.

2 Preliminaries

2.1 Template attack

TA is one of the most powerful SCA attack methods. Researchers are dedicated to improving it by introducing new technologies [23–26]. Cross-device TAs can also be practical [22, 27]. This type of TA employs a transformation to maximize the similarity between a profiling device and a target device. Both traditional TA and clustering-based TA [28] remain a matter of concern . They involve two steps: profiling and matching. Let L be the side channel leakage matrix and $l^{t,n}_{m,d}$ be the n-th vector during a time interval t, where m is input plaintext and d is the actual secret key. Typically, l^{t}_{inte} denotes one leakage vector in t under an intermediate value inte. The traditional TA procedure is as follows.

Profiling. In this phase, an adversary needs to control a profiling device that is identical (or very similar) to the target device. Suppose that an adversary obtains $|s|$ leakage vectors for a given class $s \in \mathcal{S}$. The classification is related to an intermediate value $v = f(m, d)$, where the function f is reversible. The intermediate values can be reflected by power traces. A multivariate Gaussian noise model is in general considered to describe the leakage characteristics:

$$\mathcal{N}(l^{t,1}_{m,d}|\mu_s, \Sigma_s) = \frac{1}{(2\pi)^{N/2}}\mathrm{exp}\left\{-\frac{1}{2}(l^{t,1}_{m,d} - \mu_s)^{\mathrm{T}}\Sigma_s^{-1}(l^{t,1}_{m,d} - \mu_s)\right\}, \tag{1}$$

where μ_s is the mean vector and Σ_s is the covariance matrix. In the profiling stage, the parameters of each class are estimated. Both parameters reveal completely the noise distribution associated with each class in $\mathcal{S}$. In a practical situation, an adversary usually utilizes the empirical mean and covariance:

$$\hat{\mu}_s = \frac{1}{|s_i|}\sum_{n=1}^{|s_i|} l^{t,n}_{m,d}, \tag{2}$$

$$\hat{\Sigma}_s = \frac{1}{|s_i|}\sum_{n=1}^{|s_i|}(l^{t,n}_{m,d} - \hat{\mu}_s)(l^{t,n}_{m,d} - \hat{\mu}_s)^{\mathrm{T}}. \tag{3}$$

Matching. Given an unclassified power trace $l^{t,1}_{\mathrm{new}}$, we employ Bayes' rule to determine to which class it belongs. The classification rule is

$$\hat{s} = \arg\max_{s^*}\hat{\Pr}[s^*|l^{t,1}_{\mathrm{new}}] = \arg\max_{s^*}\hat{\Pr}[l^{t,1}_{\mathrm{new}}|s^*]\Pr[s^*], \tag{4}$$

where $\Pr[s^*]$ is the prior probability of the class candidate s^*. If the classification is based on the byte value, the prior probability is $\Pr[s^*] = \frac{1}{256}$. In general, we have $\Pr[s^*] = \frac{1}{|\mathcal{S}|}$ and $\hat{\Pr}[s^*|l^{t,1}_{\mathrm{new}}] =$

$\mathcal{N}(l^t_{x,d}|\mu_{s^*}, \Sigma_{s^*})$. The maximum probability indicates the correct classification, which means the intermediate value v^* is obtained. Then, we obtain $d^* = f^{-1}(x, v^*)$. The leakage matrix $l^{t,m}_{\text{new}}$ can also be utilized to assign power traces to the candidate s^* with a higher probability than one power trace. The classification rule is slightly modified

$$\hat{s} = \arg\max_{s^*} \sum_{i=1}^{N} \hat{\Pr}[l^{t,i}_{\text{new}}|s^*]\Pr[s^*]. \tag{5}$$

Adversaries utilize TA attacks to obtain secret intermediate data in the same time interval t. The traditional TA is effective under a known or an identical leakage model. However, estimation errors exist between the practical leakage and the estimated leakage model. To avoid these errors, cluster-based TA can be employed. In [28], the authors showed how to utilize K-means and agglomerative hierarchical clustering to build a template without prior knowledge of the leakage model. These cluster techniques help obtain a template that is close to the practical leakage situation. In this study, we constructed templates in the practical leakage scenario.

2.2 RSA-CRT implementation

In 1978, the RSA cryptosystem, which has become one of the most widely used public key cryptosystems, was introduced by Rivest, Shamir, and Adleman [29]. RSA is based on a large number factorization problems and can be utilized for encryption and signature schemes. In an RSA scheme, N denotes the public modulus, being the product of two secret large prime integers p and q. d denotes the secret private key and e is the public key satisfying $de = 1 \bmod \phi(N)$, where ϕ denotes Euler's totient function and $\phi(N) = (p-1) \times (q-1)$ is also secret. The RSA signature or decryption of a message $m \in \mathbb{Z}_N$ is achieved by computing the modular exponentiation $C = M^d \bmod N$. To verify or encrypt C, $M = C^e \bmod N$ is computed.

As is well known, modular exponentiation is time consuming. Researchers have utilized the Chinese remainder theorem (CRT) [30] to accelerate the core computation with a factor of four, and this is widely used in devices having limited resources. The RSA-CRT is described in Algorithm 1.

Algorithm 1 RSA-CRT implementation

```
Require: Secret key $d$, secret primes $p$ and $q$ message $M$;
Ensure: $C$=$M^d \bmod N$;
1: $d_p$=$d \bmod p$, $d_q$=$d \bmod q$;
2: $K$=$p^{-1} \bmod q$;
3: $M_p$=$M \bmod p$, $M_q$=$M \bmod q$;
4: $C_p$=$M_p^{d_p} \bmod p$;
5: $C_q$=$M_q^{d_q} \bmod q$;
6: $C$=$(((C_q - C_p) \times K) \bmod q) \times p + C_p$;
7: Return $C$.
```

Step 6 can also be rewritten as

$$C = x \times p + C_p. \tag{6}$$

$x = ((C_q - C_p) \times K) \bmod q$, p, and C_p are secret for adversaries. However, half the most significant bytes of C and the secret $x \times p$ are identical, because C_p receives the same bit length as p. According to SCA theory, side channel leakage reflects all the intermediate data. The remaining question is how to extract this information, which is one of our main concerns in this paper.

3 Methodology

3.1 Similar operation template attack strategy

In this paper, operations that share similar basic hardware behaviors are referred to as similar operations. Their leakage functions may resemble each other. $O_{\rm pi}$ and $O_{\rm si}$ denote two similar operations manipulated on public and secret information, respectively. The SOTA strategy, in its most general form, is as follows.

(1) Acquire power traces on a running cryptographic device, where side channel leakage is $L = l_{\rm si}^{t_1} \| l_{\rm pi}^{t_2}$ and $\|$ is concatenation. $l_{\rm si}^{t_1}$ and $l_{\rm pi}^{t_2}$ are the side channel leakage of $O_{\rm si}$ and $O_{\rm pi}$, respectively.

(2) Select the template building method according to the practical leakage scenario. Construct template T on the side channel leakage of $O_{\rm pi}$. Utilize the corresponding matching method $D(l_{\rm si}^{t_1}, T)$ to reveal si.

(3) Verify attack results.

SOTA attempts to reveal secret parameters through templates built on public information through side channel leakage. In Step 1, we divide the power traces into several parts. $l_{\rm pi}^{t_2}$ denotes the leakage of public information, which can easily be detected by Pearson's correlation. This part includes the RSA final output, C. The public information can also be found in other PKC schemes, such as the signature output in ECDSA. $l_{\rm si}^{t_1}$ contains secret information involving all the intermediate values in the combination phase of RSA-CRT.

Step 2 constitutes the complete template attack procedure. We build templates on leakage $l_{\rm pi}^{t_2}$ and then reveal the secret si by using these templates. In the same power trace collection, the measurements can be utilized for both template profiling and matching. It is worth mentioning that $t_1 == t_2$ is required in a traditional TA, whereas $t_1 \neq t_2$ is evaluated in our attack procedure. In addition to traditional TAs, subspace-based TAs, clustering-based TAs [28], principle component analysis [24, 31], and Fisher's linear discriminant analysis [23] can also be employed in the SOTA procedure.

In Step 3, verification is performed. In a PKC scheme, our attack results may not be the secret key but rather the secret intermediate values, which are dependent on the attack target. Then, the verification step, which attempts to recover the private key or a hidden prime in the RSA scheme, is necessary. Let us take RSA-CRT as an example. If the secret result $x \times p$ is obtained, the factorization of N is successfully achieved by computing $\gcd(x \times p, N)$. If partial information about x is obtained, then we must find a new means of recovering the hidden prime, because no previous method provides a solution. Even half the most significant bytes of one prime $\hat{p}$ are sufficient, since Coppersmith's method can be executed to reveal the secret key.

In general, the SOTA attack procedure can be viewed as a comparison of two independent leakages. The attack method proposed in [21] can be viewed as a variant of similar operations that reveal a secret key by comparing two modular multiplications having one identical input. In our attack procedure, we attempt to exploit the simpler operations in the RSA-CRT combination phase by applying a combination TA.

3.2 SOTA in RSA-CRT combination phase

In this study, we attempted to find two similar operations in the combination phase of RSA-CRT. In Figure 1, the typical RSA-CRT implementation is shown. Message is the input plaintext. In the RSA-CRT naive implementation, the first step is the precomputation, as shown in Algorithm 1 (lines 1–3). All the corresponding results are stored and transferred to two modular exponentiations, which constitute the core computation of an RSA-CRT scheme. After the computation of both modular exponentiations, the final results (C in Algorithm 1) are obtained through the combination.

However, our concern is the combination phase (line 6 in Algorithm 1), which is shown in the lower part of Figure 1, where the four computation blocks are shown: modular subtraction, modular multiplication, multiplication, and addition. The modular subtraction is $A = (C_q - C_p) \bmod q$, the modular multiplication is $B = A \times K \bmod q$, the multiplication is $D = B \times p$, and the addition is $C = D + C_p$. C is the final result of $M^d \bmod N$. In a resource-limited device, the four blocks are executed serially. The results of the previous block must be transferred into the next one, which means that data transferal occurs after

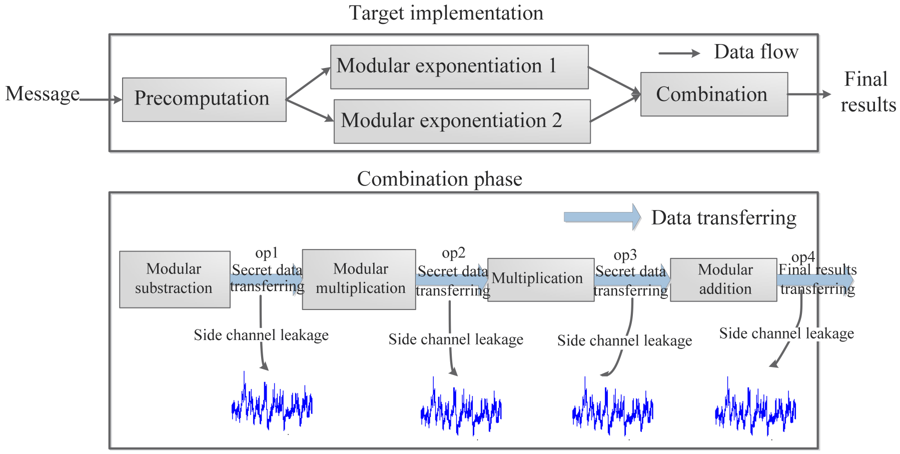

Figure 1 (Color online) Similar operations in the RSA-CRT combination phase.

each computation block. We focus on the data transferal in the combination procedure, shown by arrows in the lower part of Figure 1.

We refer to op1, op2, op3, and op4 as the similar operations in the combination step (lower part of Figure 1). This type of similar operations shares a simpler structure than that used in [21]. The transferal is independent of the specific computation block, but the byte flow is loaded from the previous block to the next one. op1, op2, and op3 denote the secret intermediate data transferal. op4 denotes the final result transferal, which is public information. The basic hardware structure of similar operations is identical, which means that the side channel leakage patterns are similar.

The similar operations provide a link between the secret intermediate data and the public final result. A SOTA can be mounted during these operations. We can build templates on the side channel leakage of op4, which transfers the public final result. Then, we reveal the secret intermediate data manipulated by other operations. Considering practical recovery situations, different adversaries acquire various abilities for recovering target secret values when utilizing SOTA, as described in the previous steps. We give following definition of the attack abilities.

Definition 1 (Adversary's attack ability)**.** The adversary's attack ability is evaluated by the result set of Target when an adversary, $\mathcal{A}$(SOTA, Target, Template), utilizes a SOTA to reveal Target based on Template. The evaluation can be described by

$$\mathcal{A}(\text{SOTA, Target, Template}) = \mathrm{RC}_{\text{Target}} + \zeta, \tag{7}$$

where $\mathrm{RC}_{\text{Target}}$ is the result set of Target and ζ is noise.

The result set $\mathrm{RC}_{\text{Target}}$ indicates the possibilities of actual results, which are typically dependent on leakage models. ζ indicates noise during the SOTA. In the definition, two factors affect the final matching results. Without loss of generality, we take one byte as an example. The upper bound of one byte recovery is that the adversary can uniquely confirm the target byte value without any noise, which means $|\mathrm{RC}_{\text{Target}}| = 1$ and $\zeta = 0$. The lower bound is that the adversary cannot obtain any biased information about Target, which means $|\mathrm{RC}_{\text{Target}}| = 256$ and $\zeta =\propto$. The side channel leakage acquirement quality, template building methods, preprocessing methods, and other elements affect the adversary's attack ability.

3.3 Hidden multiplier problem over GF(q)

The HMP over GF(2^n) was first introduced at Asiacrypt 2014 [32], and then at CHES 2015 [33]. The definition of HMP is as follows [33].

Definition 2 (Hidden multiplier problem)**.** Let $k \leftarrow \mathrm{GF}(2^n)$. Let $\ell \in \mathbb{N}$. Given a sequence $(a_i, \mathcal{L}_i)_{1\leqslant i\leqslant \ell}$, where $a_i \leftarrow \mathrm{GF}(2^n)$ and $\mathcal{L}_i = \mathrm{HW}(a_i \cdot k) + \varepsilon_i$, where $\varepsilon_i \leftarrow \mathcal{N}(0, \sigma)$, recover k.

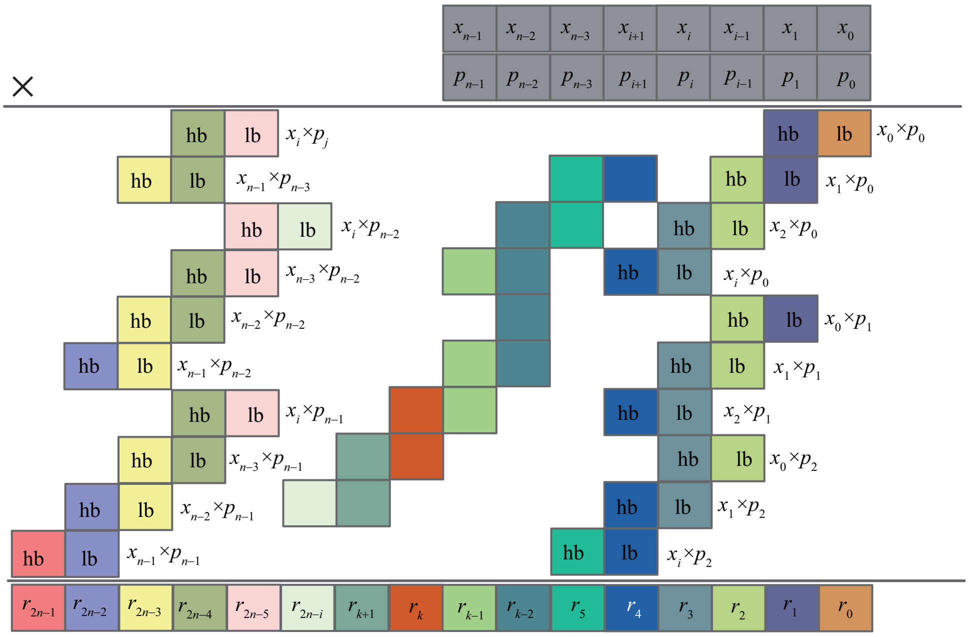
×
x_{n-1} x_{n-2} x_{n-3} x_{i+1} x_i x_{i-1} x_1 x_0
p_{n-1} p_{n-2} p_{n-3} p_{i+1} p_i p_{i-1} p_1 p_0
hb lb $x_i \times p_j$
hb lb $x_{n-1} \times p_{n-3}$
hb lb $x_i \times p_{n-2}$
hb lb $x_{n-3} \times p_{n-2}$
hb lb $x_{n-2} \times p_{n-2}$
hb lb $x_{n-1} \times p_{n-2}$
hb lb $x_i \times p_{n-1}$
hb lb $x_{n-3} \times p_{n-1}$
hb lb $x_{n-2} \times p_{n-1}$
hb lb $x_{n-1} \times p_{n-1}$
hb lb $x_0 \times p_0$
hb lb $x_1 \times p_0$
hb lb $x_2 \times p_0$
hb lb $x_i \times p_0$
hb lb $x_0 \times p_1$
hb lb $x_1 \times p_1$
hb lb $x_2 \times p_1$
hb lb $x_0 \times p_2$
hb lb $x_1 \times p_2$
hb lb $x_i \times p_2$
r_{2n-1} r_{2n-2} r_{2n-3} r_{2n-4} r_{2n-5} r_{2n-i} r_{k+1} r_k r_{k-1} r_{k-2} r_5 r_4 r_3 r_2 r_1 r_0

Figure 2 (Color online) Byte-wise multiplication from the most significant byte.

HW denotes the Hamming weight. ε is the leakage noise where $\mathcal{N}(0,\sigma)$ denotes the Gaussian distribution with null mean and standard deviation σ. Given known a_i and $\sigma = 0$, the adversary can easily recover k, because for known a_i the least significant bit of HW is a linear function of the bit of secret k. In our attack scenario, (full or noisy) intermediate values x^i can be obtained through SOTA, based on which we extend it to a new problem as follows.

Definition 3 (Hidden multiplier problem over GF(q))**.** Let $N = p \times q$, where p and q are two n-byte long big primes. Let $\ell \in \mathbb{N}$. Given a sequence $(\mathcal{L}^i, \mathcal{R}^i)_{1 \leqslant i \leqslant \ell}$, where $\mathcal{L}^i = \mathrm{LM}(x^i) + \varepsilon_i$, $\varepsilon_i \leftarrow \mathcal{N}(0,\sigma)$, $x^i \leftarrow \mathrm{GF}(q)$ and $\mathcal{R}^i = \mathrm{HB}(x^i \times p)$, recover p.

HB$(*)$ denotes half the most significant bytes of $*$. The leakage $\mathcal{L}^i$ is the side channel leakage in a leakage model LM$(*)$. ε is the leakage noise. In an RSA-CRT implementation, x, p, and q are the same length big integer (typically, 64-byte long), where HB$(x^i \times p)$ are identical to the counterpart of the corresponding modular exponentiation result. The security of RSA-CRT is reduced to such a problem. The adversary can easily obtain the hidden prime with known x. However, given biased information about x, no method for solving the problem has thus far been published. We aim to handle it by revealing the hidden prime p under the attack results of $\mathcal{A}(\mathrm{SOTA}, x^i, C)$. For convenience, we denote big integers as n-byte vectors, $x^i = (x^i_{n-1}, x^i_{n-2}, \ldots, x^i_0)$, and $p = (p_{n-1}, p_{n-2}, \ldots, p_0)$, where x^i_j represents the j-th byte of the i-th modular exponentiation, and x^i_0 is the least significant byte and x^i_{n-1} the most significant byte. r^i is the corresponding result of $x^i \times p + C_p$. We represent the results with vector $\{r^i_{2n-1}, \ldots, r^i_0\}$.

We utilize a divide-and-conquer strategy for recovering the hidden prime byte-by-byte. In view of the byte, the detailed multiplication procedure can be represented (from the most significant byte) by the method shown in Figure 2. hb and lb respectively denote the high and low byte of 16-bit intermediate data. Every r^i_j derives from one or several intermediate bytes, which are labeled using the same color, and the carries of the former intermediate bytes. In general, given $x_{n-1}, x_{n-2}, \ldots, x_{n+1-j}$ and $p_{n-1}, p_{n-2}, \ldots, p_{n+1-j}$, the next result byte r_{2n-j} can be represented by the function

$$r_{2n-j} = f(x_{n-j}, x_{n-j-1}, p_{n-j}, c_{2n-j-1}). \tag{8}$$

c_{2n-j-1} represents the carries from the computation procedure of r_{2n-j-1}. Obviously, when j approaches n, the number of intermediate bytes is a linear function of j. For $0 \leqslant j \leqslant n-1$ (or $n \leqslant j \leqslant 2n-1$), r_i derives from $2 \times (j+1) - 1$ (or $2 \times (2n-j) - 1$) intermediate bytes. In this study, we evaluated how to solve the problem defined in definition 3 byte-by-byte with the SOTA results. Two obstacles are easily

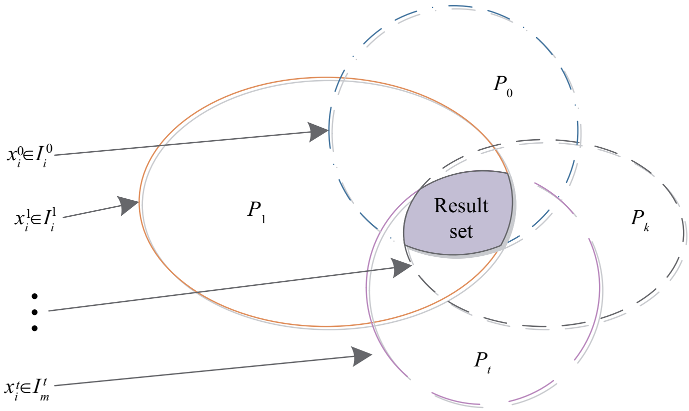

Figure 3 (Color online) Prime byte filtered procedure.

detected. The first is the linear increment of intermediate bytes, which means that the carries may be extremely large, which would prevent us from obtaining a precise guess on p_{n-j}. The second obstacle is the precise calculation of the intermediate bytes, including the selection of a different high or low byte. In Subsection 3.4, we provide the solution for the two obstacles.

3.4 Byte-by-byte recovery of prime

As mentioned in Section 2, a modular exponentiation result C can be represented by a simpler form $C = x \times p + C_p$, where x and C_p are secret. Given biased information about x byte-by-byte, we can recover the secret and fixed prime p byte-by-byte with high probability. Our recovery procedure is shown in Figure 3. According to adversaries' attack ability, we can obtain the result set $\mathcal{I}_m^t$ (corresponding to input x_m^t), where $1 \leqslant m \leqslant w$, as shown in Figure 3. If we denote by $\mathcal{O}$ all possible values of one single byte, we can obtain $\bigcup \mathcal{I}_m = \mathcal{O}$. A typical side channel leakage model is a Hamming weight, where $w = 9$ and $\bigcap \mathcal{I}_m = \emptyset$.

If an adversary attempts to reveal the i-th prime byte, given $x^t = \{x_{n-1}^t, x_{n-2}^t, \ldots, x_{j+1}^t\}$, $p = \{p_{n-1}, p_{n-2}, \ldots, p_{j+1}\}$, and $x_j^t \in \mathcal{I}_m^t$, we traverse all possible values in the set $\mathcal{I}_m^t$ and prime byte values by comparing r_{2n-i} and then we obtain the corresponding result prime set P_0. Increasing t from 0 to a certain number, we can obtain all the corresponding prime byte sets P_t. Several prime bytes exist in each of these result sets, which can be obtained by the interaction between all P_t. These results are filtered in the procedure, as shown in the Figure 3. However, a big problem is that the carries will be linear increment when j approaches 0, which can hinder recovery. Our solution is to utilize two adjacent bytes of input x^t, x_j^t and x_{j-1}^t. During our prime byte recovery procedure, the two adjacent bytes can help eliminate the carry problem and the carries are limited to two choices, 0 and 1.

The carry reduction procedure can be summarized by (8). The details of the procedure are shown in Figure 4. In the figure, cur represents the current prime byte for recovery and pre and nex represent the previous and next prime bytes, respectively. On the left-hand side of the figure, detailed byte multiplications corresponding to the first prime byte are shown. Given x_{n-1}, x_{n-2} and the guess on p_{n-1}, all the green boxes can be viewed as known intermediate values. The unknown values are the gray box and the low byte of the addition of all the next green boxes. The unknown intermediate byte and the known intermediate bytes' addition result provide two possibilities of carries, 0 or 1. A similar situation can be found in p_{n-2} recovery, as shown on the right-hand side of Figure 4. Given x_{n-1}, we try all possible x_{n-2}, x_{n-3}, and p_{n-2}, and then, the number of unknown intermediate bytes is still two and the carries are fixed. Therefore, the recovery procedure of all the prime bytes can control the linear increments of the carries.

As stated in Subsection 3.3, another obstacle is the precise calculation of the intermediate bytes. Algorithm 2 provides a solution of the carry problem. An easily overlooked problem is how to confirm all previous bytes $x_{n-1}, x_{n-2}, \ldots x_{j+1}$ when traversing x_j and x_{j-1}. Given each byte $x_j^t \in \mathrm{RC}_{\mathrm{Target}}$ and $j \to 0$, the traversal time is excessive and computationally infeasible if we cannot confirm all the previous

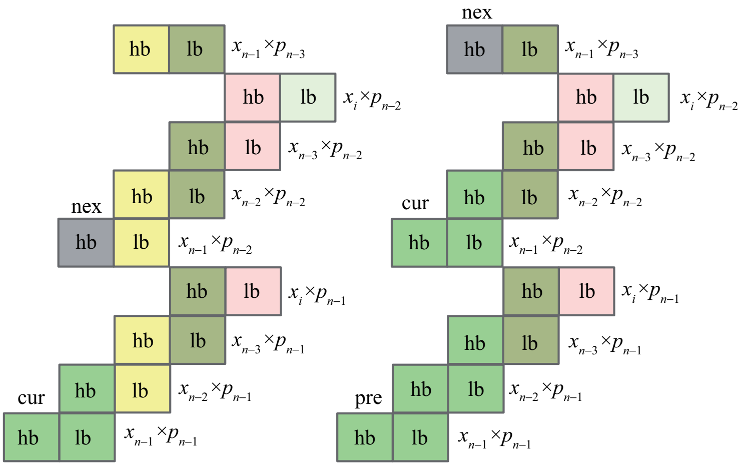

Figure 4 (Color online) Details of carry reduction procedure.

Algorithm 2 Byte choice algorithm of 16-bit intermediate value

```
Require: index $i$ $j$, attack index BytePosition, 16-bit Intermediate value TempResult;
Ensure: PreviousByte, CurrentByte, NextByte=ByteChoice(TempResult);
if $i+j$ == BytePosition-2 then
    PreviousByte ⇐ TempResult & 0xFF;
    CurrentByte ⇐ 0;
    NextByte ⇐ 0;
else if $i+j$ == BytePosition-1 then
    PreviousByte ⇐ TempResult>>8 & 0xFF;
    CurrentByte ⇐ TempResult & 0xFF;
    NextByte ⇐ 0;
else if $i+j$ == BytePosition then
    PreviousByte ⇐ 0;
    CurrentByte ⇐ TempResult>> 8 & 0xFF;
    NextByte ⇐ TempResult & 0xFF;
else if $i+j$ == BytePosition+1 then
    PreviousByte ⇐ 0;
    CurrentByte ⇐ 0;
    NextByte ⇐ TempResult>> 8 & 0xFF;
else
    PreviousByte ⇐ 0;
    CurrentByte ⇐ 0;
    NextByte ⇐ 0;
end if
```

input bytes. Therefore, we must (approximately) confirm these input bytes. Our solution is the simple division of r by the previous prime byte. Half of the most significant bytes of the multiplication leaks, which means that we can easily obtain all the bytes of x given known prime bytes. We claim that the completely hidden p can be recovered through the recovery procedure shown in Algorithm 3.

As shown in the Algorithm 3, typically $|S_{\text{pre}}| > 1$. We have to uniquely confirm the previous prime byte, namely, $|S_{\text{pre}}| = 1$. This recovery algorithm provides the solution for this confirmation after the current prime byte recovery, as shown in line 20 of the algorithm. We emphasize that all the previous prime bytes can be uniquely confirmed during the next byte recovery.

4 Practical experiments

4.1 Measurement setup and experiment environment

In order to evaluate the proposed attack method, we targeted 1024-bit RSA-CRT implemented on 8-bit AVR microcontroller clocked at 10 MHz. Its architecture is serialized without any countermeasure against

Algorithm 3 Single prime byte recovery algorithm

```
Require: $x^t = \{x^t_{n-1}, x^t_{n-2}, \ldots, x^t_i, x^t_{i-1}\}$, where $x^t_i \in \mathcal{I}^t_0$ and $x^t_{i-1} \in \mathcal{I}^t_1$, $p = \{p_{n-1}, p_{n-2}, \ldots, p_{i+1}\}$,
         previous prime byte set $S_{\rm pre}$ where $p_{i+1} \in S_{\rm pre}$, result $r^t = \{r^t_{2n-1}, \ldots, r^t_n\}$.
Ensure: $S_{p_{i+1}, p_i}$.
for $t = 0$ to $n$ do
    for all $p_{i+1} \in S_{\rm pre}$ do
        for prime $= 0$ to 255 do
            Index$\Leftarrow 1$;                                        ▷ flag
            $p = \{p_{n-1}, p_{n-2}, \ldots, p_{i+1}, \text{prime}\}$;
            for all $x^t_i \in \mathcal{I}^t_0$ do
                for all $x^t_{i-1} \in \mathcal{I}^t_1$ do
                    $x^t = \{x^t_{n-1}, x^t_{n-2}, \ldots, x^t_i, x^t_{i-1}, x^t_i, x^t_{i-1}\}$;      ▷ obtain previous input bytes
                    TempResult$=g(x, p, C_{2n-i-1})$;                   ▷ obtain TempResult
                    {PreviousByte,CurrentByte,NextByte}=ByteChoice(TempResult);  ▷ obtain intermediate value
                    if CurrentByte$\leqslant r^t_{2n-i} - 1$ && PreviousByte$\equiv r^t_{2n-i+1}$ && Index then
                        $A[p_{i+1}][\text{prime}] += 1$;           ▷ compare intermediate value and $r^t$, count all possible prime byte
                        Index$\Leftarrow 0$;
                    end if
                end for
            end for
        end for
    end for
end for
$S_{p_{i+1}, p_i} \Leftarrow \max(A^{\rm prime}_{p_{i+1}})$.                    ▷ obtain prime byte results
```

SCAs. Because of the long integer and the serialized architecture, we can detect feasible outputs of certain computations. In other words, we can benefit from this architecture. This implementation utilizes two close primes:

p: 0xcd083568d2d46c44c40c1fa0101af2155e59c70b08423112af0c1202514bba5210765e29ff13036f56c7495894d80cf8c3baee2839bacbb0b86f6a2965f60db1.

q: 0xca0eeea5e710e8e9811a6b846399420e3ae4a4c16647e426ddf8bbbcb11cd3f35ce2e4b6bcad07ae2c0ec2ecbfcc601b207cdd77b5673e16382b1130bf465261.

In addition, a LeCroy 610Zi WaveRunner 8-bit oscilloscope is needed. We evaluated this scheme based on 12000 profiling traces and 150 attack traces and Algorithm 3 on an Intel Xeon personal computer.

4.2 SOTA in RSA-CRT software implementation

The application of SOTA to an unprotected RSA-CRT software implementation follows the steps described in Section 3. We present a detailed description of our attack procedure. We stress that the signal-to-noise ratio (SNR) of our software implementation is high [34], which allows us to obtain accurate templates using 12000 profiling traces and achieve a high success rate within 150 attack traces.

We first scrutinize the power traces to obtain similar operation locations in the same measurement. An exemplary power trace of both parts is shown in Figure 5(a) marked by two black squares. It is easy to detect the two operations that share a similar appearance, as shown in Figure 5(b) and (c), by mere visual inspection. Both parts indicate the power consumption of two similar operations. The first operation is the data transferal of secret intermediate data, and the second is the data transferal of the public final result C. When two operations are executed, the basic hardware behaves similarly and similar leakage patterns can be detected, as shown in Figure 5. Although the envelopes of the two parts are similar, slight differences can also be detected, as shown in Figure 5(c) marked by the black crosses. Noise and different manipulated data cause a slight vibration in the power traces. A practical leakage situation is depicted in Figure 5(d). This figure reflects the leakage situation verified by Pearson's correlation between power traces and final outputs.

We evaluated the leakage model using clustering techniques. The results show that the leakage model is a Hamming weight, which is consistent with our knowledge of the leakage characteristics of an AVR processor. In Figure 6, (a) and (b) are the templates and the secret intermediate data leakage model, respectively. Both obey the Hamming weight leakage model. However, the two leakages are slightly

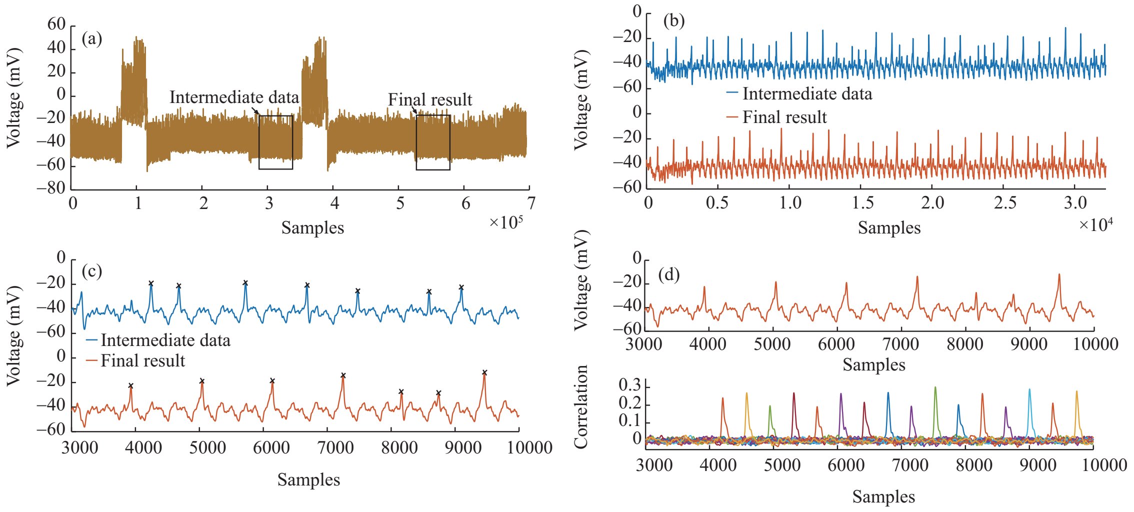

Figure 5 (Color online) Power traces of targeted RSA-CRT implementation. (a) Combination phase overview; (b) public and secret leakage overview; (c) public and secret leakage details; (d) leakage characteristics.

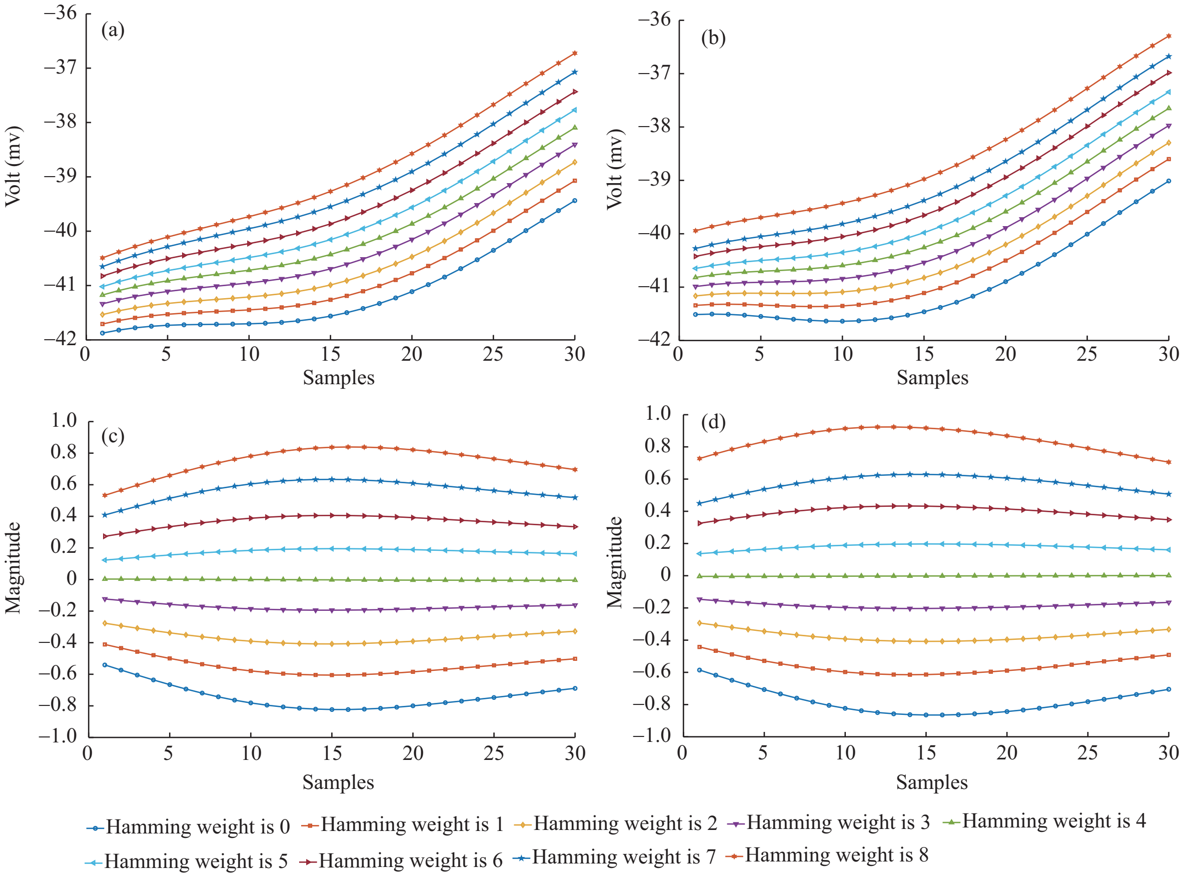

Figure 6 (Color online) Raw power traces before and after preprocessing. (a) Template; (b) leakage; (c) template after zero-mean; (d) leakage after zero-mean.

different, which makes the matching error high. We utilized a zero-mean method [22] to unify the two different leakage characteristics. In both Figure 6(c) and (d), a similar leakage after zero-means is shown. The results show that zero-mean is a feasible preprocessing method for cross-device TA and SOTA.

Despite the visual similarity of the leakage patterns, we must confirm the leakage uniformity of two similar operations in our measurement. Perceived information (PI) [35] is a convenient statistics tool

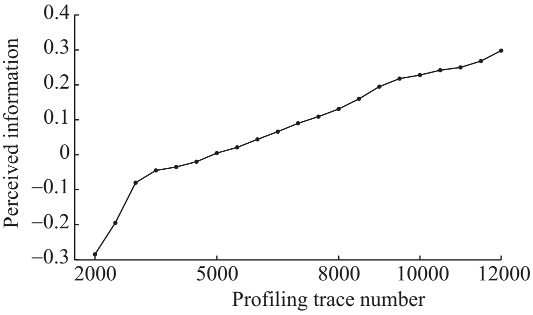

Figure 7 Perceived information vs. profiling traces.

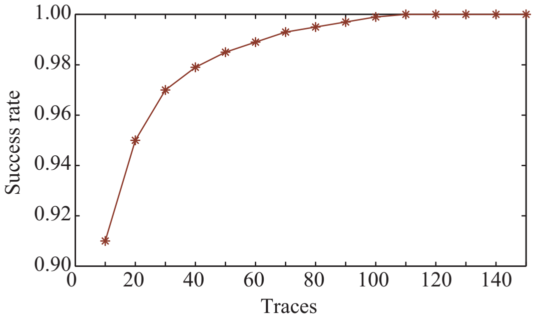

Figure 8 (Color online) Matching success rate vs. various traces.

for this purpose. Researchers utilize PI to evaluate the leakage of a cryptographic implementation. PI reflects the bias between the model and the target leakage, which is defined by

$$\mathrm{PI}(S;L) = H[S] - \sum_{x\in S}\Pr[s]\sum_{l\in L}\hat{\Pr}_{\text{chip}}[l^t|s]\log_2 \hat{\Pr}_{\text{model}}[s|l^t]. \tag{9}$$

$\hat{\Pr}_{\text{model}}[s|l^t] = \mathcal{N}(l^t_{x,d}|\mu_{s^*},\Sigma_{s^*})$ (described in Section 2) corresponds to the 256 maximum likelihood estimates of conditional density functions $\Pr_{\text{chip}}[L|x]$. In our case, PI is computed by

$$\mathrm{PI}(S;L) = H[S] - \sum_{x\in S}\Pr[s]\sum_{l\in L}\hat{\Pr}_{\text{chip}}[l^{t1}_{\text{si}}|s]\log_2 \hat{\Pr}_{\text{model}}[s|l^{t2}_{\text{pi}}]. \tag{10}$$

Our leakage model is built on l^{t2}_{pi}. Then, PI can be used as a tool for testing the similarity between the template and the actual leakage l^{t1}_{si}. The PI results are shown in Figure 7. As can be seen, PI increases with the profiling traces, which means that the leakages of similar operations are similar. Then, we can conclude that SOTA is reasonable. We utilize SOTA to obtain the Hamming weight of x through the templates built on the final results.

We show the success rate results in Figure 8, based on (5) described in Section 2. Given a specific number of power traces acquired under the same intermediate data, the success rate is obtained by dividing the total attempts by the occurrences of correct matching. As shown in Figure 8, the success rate reaches 90% with 10 power traces and 100% with approximately 100 power traces in our experimental environment.

4.3 Secret prime recovery

We show the first two prime byte recovery results based on Algorithm 3 in Figure 9. In Figure 9(a), the results show the occurrences of all the possibilities, where the highest peak is located at 0xCD. However, we did not uniquely confirm the first byte value, but two elements, 0xCD and 0xCE, remain in the result set. We utilize the two values to reveal the second prime byte; the results are shown in Figure 9(b). The peak on the left-hand side of the figure uniquely confirms the first prime byte. As shown in Figure 9, we can obtain our expected results even on 20 inputs x.

We traversed all p_{n-1} and p_{n-2} under 100 inputs x. The results are shown in Figure 10. A similar situation, that the highest peak occurs under the correct p_{n-1}, can be seen in the figure. All the incorrect guess results are apparently lower than the correct one. The curve peak also indicates the result set of p_{n-2}. We can conclude that, given sufficient computational power, adversaries can execute recovery word by word. We present the recovery results from 2nd to 8th prime bytes in Figure 11. Each byte recovery result is distinct. In addition, we present the remaining prime byte recovery results in Figure 12. In both figures, we just select two possibilities of the previous prime byte, where the correct one is arranged on the left-hand side of these figures.

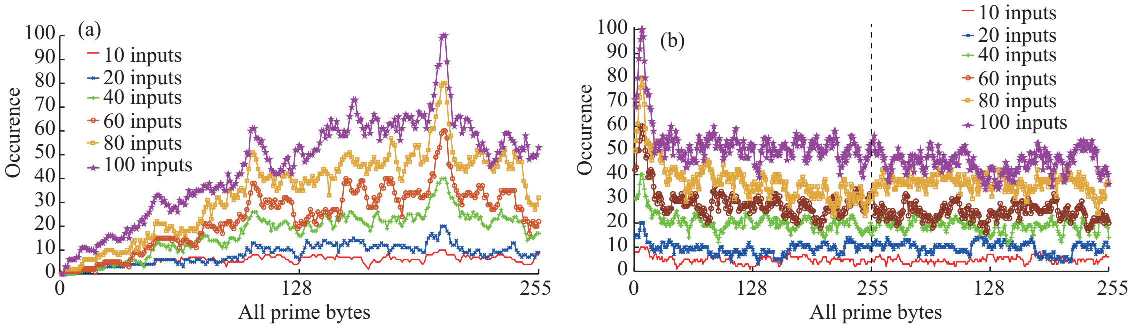

Figure 9 (Color online) Two most significant prime bytes recovery results. (a) First prime byte recovery results; (b) second prime byte recovery results.

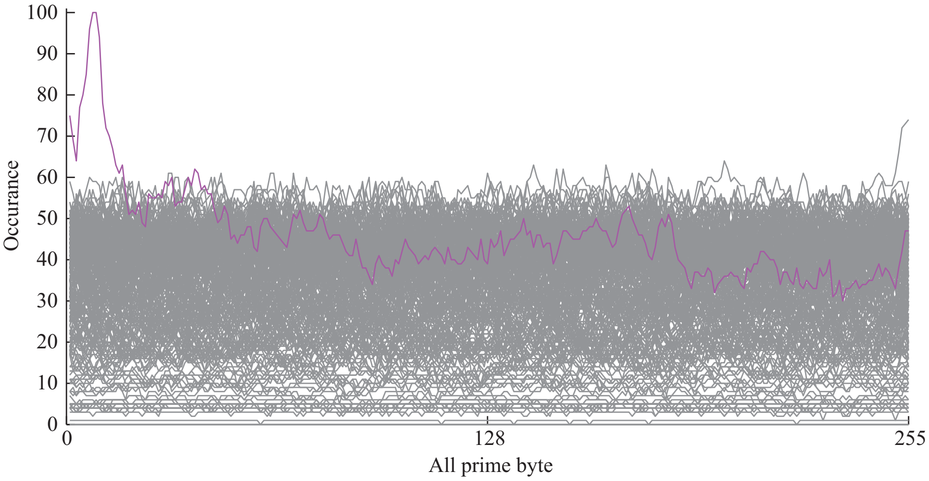

Figure 10 (Color online) First two prime byte recovery results. Pink curve indicates the second byte results based on correct first byte. Gray curves indicate the second byte results based on incorrect first byte.

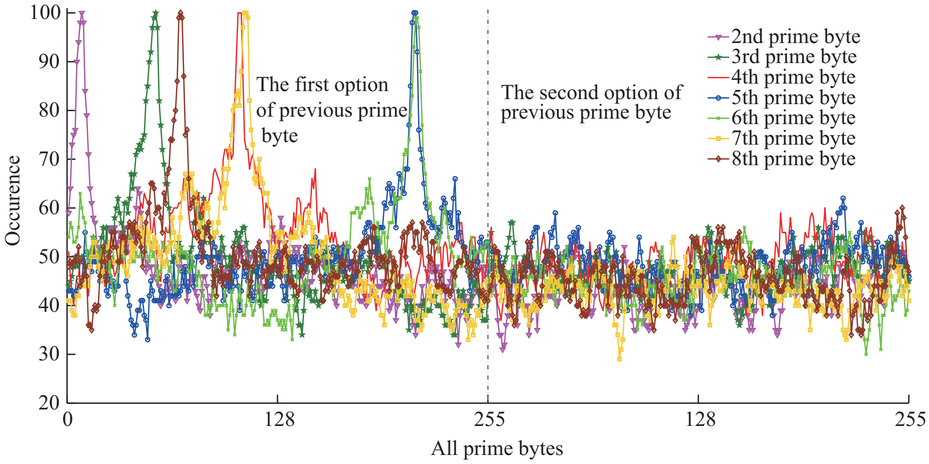

Figure 11 (Color online) Results of 2nd–8th prime byte recovery.

Adversaries' attack abilities vary. In a practical attack scenario, we believe $\zeta \neq 0$, which occurs when poor denoise methods are used, limits the template side channel leakage or limits the matching traces in a TA procedure. Error in matching is a natural result under noise conditions. Here, we consider the situation where the adversary obtains incorrect matching with a certain probability, which means that x_i are randomly categorized into adjacent Hamming weights. The differences between these adjacent and actual values are 1 or 2. We evaluated the situations where 10%, 20%, 30%, and 50% of inputs obtain

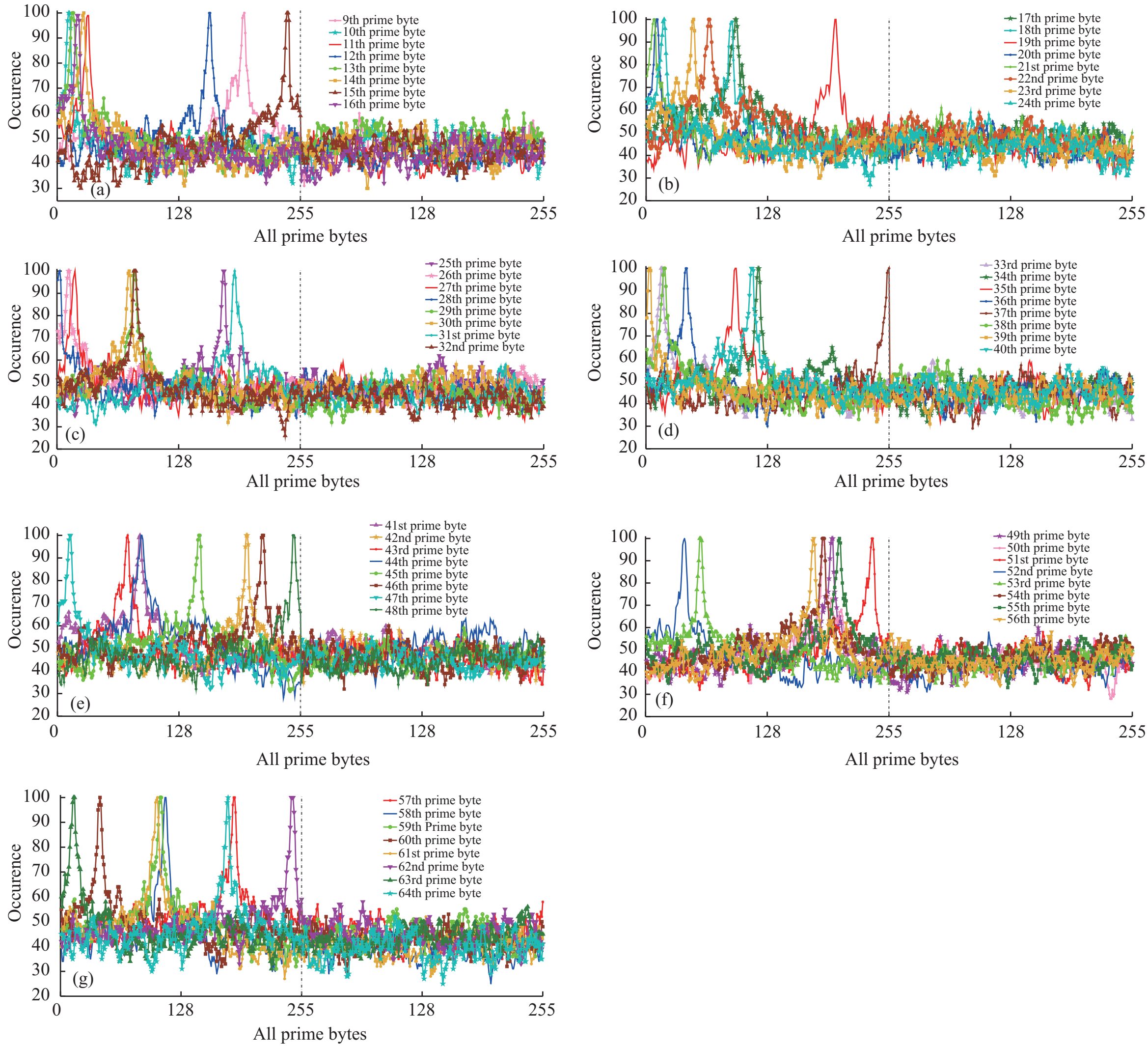

Figure 12 (Color online) During these recovery experiments, the results show the two options of the previous prime bytes. All the left-hand sides of all the figures show the correct previous prime bytes. (a) 9–16 prime byte recovery results; (b) 17–24 prime byte recovery results; (c) 25–32 prime byte recovery results; (d) 33–40 prime byte recovery results; (e) 41–48 prime byte recovery results; (f) 49–56 prime byte recovery results; (g) 57–64 prime byte recovery results.

incorrect matching. The first two prime byte results are shown in Figure 13. The first prime byte recovery is shown in Figure 13(a). The result set is still distinct at 50% noisy traces, when the difference is 1. The adversary can still find the correct prime byte involved in the highest peak with a Hamming weight matching difference of 2 and 30% noisy traces, but the highest one is not the correct result. We need to enlarge the set $S_{\rm pre}$ in the next byte recovery. We cannot obtain sufficient information about the first prime byte when the noisy traces reach 50% and meanwhile the Hamming weight matching difference is 2. However, in similar situations, the second byte recovery result peak can also be detected, as shown in Figure 13(b). The correct prime byte is involved in the highest peak even in the worst case, where 50% of inputs are noisy and the difference is 2.

4.4 Computation complexity estimation

In this subsection, we present a computation complexity analysis of Algorithm 3. Considering the practical side channel leakage scenario, as described in Section 4, we present the analysis based on Hamming weight leakage. The four iterations (lines 1, 3, 6, and 7) of the algorithm are the primary computational load. The expectation of elements in traversing a set is $\sum_{h=0}^{8}{\rm pr}_h \times |{\rm Ham}_h| \approx 50$, where ${\rm pr}_h$ and $|{\rm Ham}_h|$

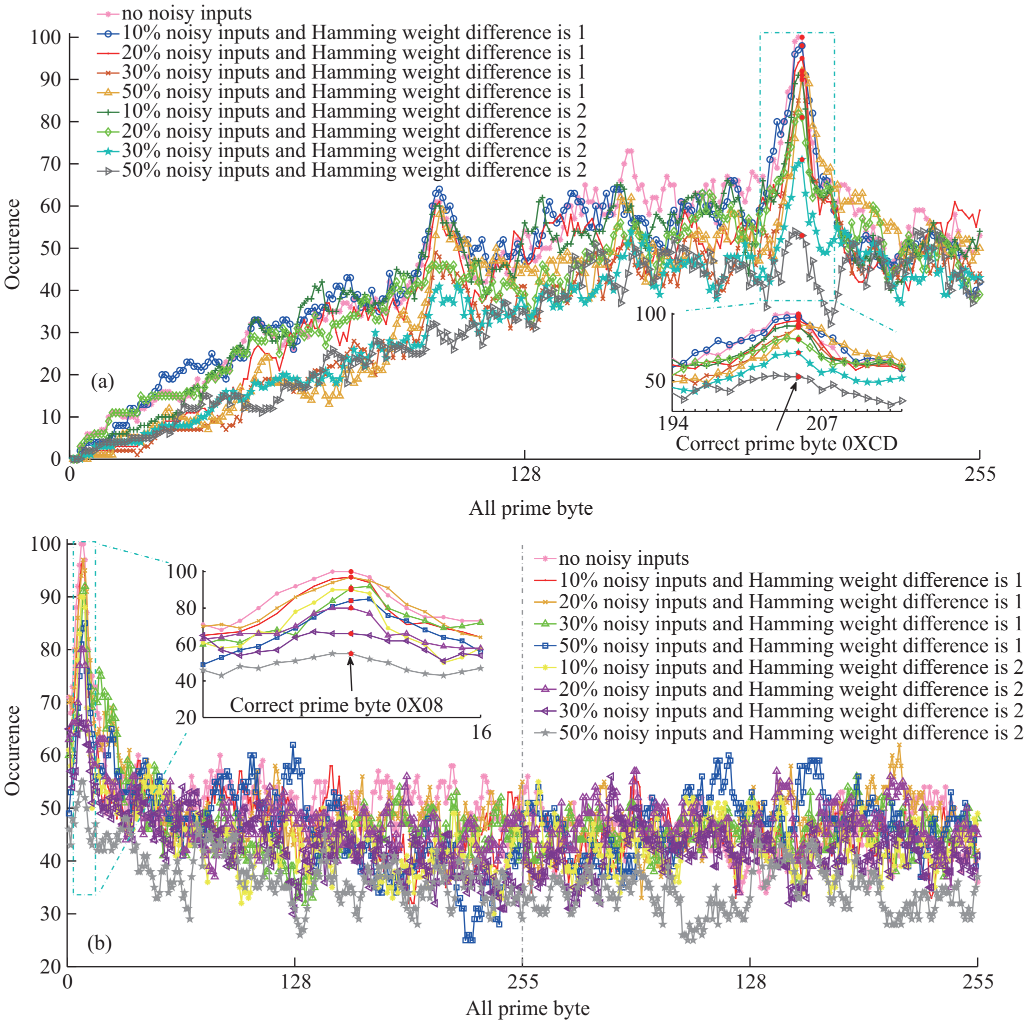

Figure 13 (Color online) First and second prime byte recovery results with different noisy inputs. When recovering the second prime byte, we give two options of the first prime byte. The left part shows the correct one. (a) First prime byte results with noisy inputs; (b) second prime byte results with noisy inputs.

are the probability and the element number of Hamming weight h, respectively. Lines 6 and 7 require approximately 2^{12} loops in total. The prime byte requires approximately 2^8 loops (line 3). The trace iteration (line 1) requires approximately 2^8 loops. Considering line 2, each prime byte recovery needs to traverse 2^{30} elements on average. Therefore, the time complexity is $O(2^n)$, where $n \approx 30$. In our experiments, we utilized 100 inputs for single prime byte recovery, where $n < 30$ is satisfied. The average time of the single byte recovery is about 30 s on the computer used in our experiment.

5 Conclusion

In this study, we introduced the similar operation template attack (SOTA) as a new variant of the TA to evaluate the security of PKC schemes. A heuristic algorithm was proposed to solve the HMP over GF(q) in the secret prime recovery for an RSA-CRT implementation. The proposed SOTA does not require an additional profiling device. The template is constructed based on public values and then used to reveal secret intermediate values. It is noteworthy that our method can be combined with a lattice attack (e.g., Coppersmith's method) to obtain a wider applicability in security analysis for PKC implementations.

Acknowledgements This work was supported by Major State Basic Research Development Program (973 Program) (Grant No. 2013CB338004), National Natural Science Foundation of China (Grant Nos. U1536103, 61402286, 61472249, 61602239, 61572192, 61472250), Minhang District Cooperation Plan (Grant No. 2016MH310), and Natural Science Foundation of Jiangsu Province (Grant No. BK20160808).

References

1 Kocher P C, Jaffe J, Jun B. Differential power analysis. In: Advances in Cryptology — CRYPTO'99. Berlin: Springer, 1999. 15–19

2 Brier E, Clavier C, Olivier F. Correlation power analysis with a leakage model. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2004. 16–29

3 Gierlichs B, Batina L, Tuyls P. Mutual information analysis. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2008. 426–442

4 Batina L, Gierlichs B, Lemke-Rust K. Differential cluster analysis. In: Cryptographic Hardware and Embedded Systems–CHE 2009 Lausanne. Berlin: Springer, 2009. 112–127

5 Chari S, Rao J R, Rohatgi P. Template attacks. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2002. 13–28

6 Amiel F, Feix B, Villegas K. Power analysis for secret recovering and reverse engineering of public key algorithms. In: Proceedings of International Workshop on Selected Areas in Cryptography. Berlin: Springer, 2007. 110–125

7 Balasch J, Gierlichs B, Reparaz O, et al. DPA, bitslicing and masking at 1 GHz. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2015. 599–619

8 Tang M, Qiu Z L, Peng H B, et al. Toward reverse engineering on secret S-boxes in block ciphers. Sci China Inf Sci, 2014, 57: 032208

9 Genkin D, Adi Shamir A, Tromer E. RSA Key Extraction via low-bandwidth acoustic cryptanalysis. In: Proceedings of Advances in Cryptology — CRYPTO 2014. Berlin: Springer, 2014. 444–461

10 Genkin D, Pipman I, Tromer E. Get your hands off my laptop: physical side-channel key-extraction attacks on PCs. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2014. 242–260

11 Genkin D, Pachmanov L, Pipman I, et al. Stealing keys from PCs using a radio: cheap electromagnetic attacks on windowed exponentiation. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2015. 207–228

12 Genkin D, Pachmanov L, Pipman I, et al. ECDSA key extraction from mobile devices via nonintrusive physical side channels. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, 2016. 1626–1638

13 Belgarric P, Fouque P A, Macario-Rat G, et al. Side-channel analysis of Weierstrass and Koblitz curve ECDSA on Android smartphones. In: Proceedings of the Cryptographers' Track at the RSA Conference 2016. Cham: Springer, 2016. 236–252

14 Coppersmith D. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. J Cryptol, 1997, 10: 233–260

15 Joye M, Yen S M. The montgomery powering ladder. In: Proceedings of Cryptographic Hardware and Embedded Systems, Redwood Shores, 2002. 291–302

16 Chevallier-Mames B, Ciet M, Joye M. Low-cost solutions for preventing simple side-channel analysis: side-channel atomicity. IEEE Trans Comp, 2004, 53: 760–768

17 Brier É, Joye M. Weierstraß Elliptic curves and side-channel attacks. In: Proceedings of International Workshop on Public Key Cryptography. Berlin: Springer, 2002. 2274: 335–345

18 Sinha Roy S, Järvinen K, Verbauwhede I. Lightweight coprocessor for Koblitz curves: 283-Bit ECC including scalar conversion with only 4300 gates. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2015. 102–122

19 Witteman M. A DPA attack on RSA in CRT mode. Riscure Technical Report, 2009. https://www.riscure.com/archive/DPA_attack_on_RSA_in_CRT_mode.pdf.

20 Aldaya A C, Sarmiento A J C, Sánchez-Solano S. SPA vulnerabilities of the binary extended Euclidean algorithm. J Cryp Eng, 2016, 7: 273–285

21 Walter C D. Sliding windows succumbs to big Mac attack. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2001. 286–299

22 Montminy D P, Baldwin R O, Temple M A, et al. Improving cross-device attacks using zero-mean unit-variance normalization. J Cryp Eng, 2013, 3: 99–110

23 Standaert F X, Archambeau C. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2008. 411–425

24 Archambeau C, Peeters E, Standaert F X, et al. Template attacks in principal subspaces. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2006. 1–14

25 Hospodar G, Gierlichs B, De Mulder E, et al. Machine learning in side-channel analysis: a first study. J Cryp Eng, 2011, 1: 293–305

26 Lerman L, Bontempi G, Markowitch O, et al. Power analysis attack: an approach based on machine learning. Int J Appl Cryp, 2014, 3: 97–115

27 Choudary O, Kuhn M G. Template attacks on different devices. In: Proceedings of International Workshop on Constructive Side-Channel Analysis and Secure Design. Cham: Springer, 2014. 179–198

28 Whitnall C, Oswald E. Robust profiling for DPA-style attacks. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2015. 3–21

29 Rivest R L, Shamir A, Adleman L M. A method for obtaining digital signatures and public-key cryptosystems. Commun ACM, 1983, 21: 96–99

30 Quisquater J J. Fast decipherment algorithm for RSA public-key cryptosystem. Electron Lett, 2007, 18: 905–907

31 Choudary O, Kuhn M G. Efficient template attacks. In: Proceedings of International Conference on Smart Card Research and Advanced Applications. Cham: Springer, 2013. 253–270

32 Belaïd S, Fouque P A, Gérard B. Side-channel analysis of multiplications in GF(2^{128})-application to AES-GCM. In: Proceedings of International Conference on the Theory and Application of Cryptology and Information Security. Berlin: Springer, 2014. 306–325

33 Belaïd S, Coron J S, Fouque P A, et al. Improved side-channel analysis of finite-field multiplication. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems. Berlin: Springer, 2015. 395–415

34 Merino Del Pozo S, Standaert F X. Blind source separation from single measurements using singular spectrum analysis. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems, Saint-Malo, 2015. 42–59

35 Renauld M, Standaert F X, Veyrat-Charvillon N, et al. A formal study of power variability issues and side-channel attacks for nanoscale devices. In: Advances in Cryptology — EUROCRYPT 2011. Berlin: Springer, 2011. 109–128