Mind Your Nonces Moving: Template-Based Partially-Sharing Nonces Attack on SM2 Digital Signature Algorithm

Jiazhe Chen
China Information Technology Security Evaluation Center
Beijing 100085, China
jiazhechen@gmail.com

Mingjie Liu
Beijing Research Institute of Telemetry
Beijing 100194, China
liumj9705@gmail.com

Hexin Li
China Information Technology Security Evaluation Center
Beijing 100085, China
lihx@secemail.cn

Hongsong Shi
China Information Technology Security Evaluation Center
Beijing 100085, China

ABSTRACT

This paper gives a partially-sharing nonces attack on SM2 Digital Signature Algorithm (SM2DSA). Templates, which are built in the scenario of no secrets known, are used to detect the collisions on the Most Significant Byte of the Nonces (MSBN). Targeting a real world smartcard with 8-bit precharged bus, the power consumption of data moving procedure after the random number generation is focused, on which the template building and matching phases are based. With the templates, we obtain a number of pairs of nonces whose first bytes are collided, then a lattice attack on SM2DSA is proposed to recover the private key. Experiments show that our attack works smoothly; our attack is the first implemented lattice attack on SM2DSA in a smartcard, which can also be extended to the other ECC algorithms like ECDSA.

Keywords

SM2, Template attack, PCA, Lattice attack

1. INTRODUCTION

SM2 [20], an elliptic curve based cryptosystem, is a Chinese standard for commercial use. A digital signature algorithm, a key exchange protocol and a public key encryption algorithm are given in [20], which form the SM2 cryptosystem. The SM2 digital signature algorithm[1] (SM2DSA for short), is similar to ECDSA [18].

The nonce k in ECDSA/SM2DSA (DSA) is an ephemeral key whose recovery is equivalent to the discovery of the private key. If a small fraction of the information about the nonces is known, then it is also enough for the lattice attacks to break the signature algorithm (like [10, 15, 19, 14]). We call these attacks partially-known nonces attacks. The other lattice attacks do not directly use the partial values of the nonces, but break (EC)DSA with some signatures that share a part of the nonces (like [8]). The latter attacks are called partially-sharing nonces attacks in this paper.

Side channel attacks [12] are of great value to evaluate the security of a cipher running in a cryptographic device; among them, template attack [5] is one of the most effective attacks. The lattice attacks mentioned are sound and practical if the information of the nonces can be obtained from side channels. This paper gives a partially-sharing nonces attack on SM2DSA resulted from the information leakage in the generation procedure of the nonces, utilizing templates.

Unlike most of the previous attacks, this paper will not focus on reducing the number of known (sharing) bits, but try to study the attack from a practical issue (like [17]) and aim to solve the problems that will be practically encountered.

Our contributions. This paper utilizes side channel information of the non-cryptographic operation whose security might not attract much attention of the designers.

Since the lattice-based attacks require that the incorrect guessed signatures should be very limited, the success rate is crucial. Our first contribution is proposing a method to reliably detect the information of MSBN from the moving of the nonces, which is given in Sect. 3.

The second contribution of this paper is giving the first partially-sharing nonces attack on SM2DSA (see Sect. 4), which is also the first lattice attack on SM2DSA in a real smartcard.

2. PRELIMINARIES

This section introduces the SM2DSA, template attack, PCA and some basics on lattice.

2.1 SM2 Digital Signature Algorithm

SM2 is an elliptic curve based cryptosystem that defined over finite fields. Similar to ECDSA, SM2DSA can be defined over prime fields $\mathbb{F}_p$ ($p > 3$) or binary fields $\mathbb{F}_{2^m}$. An

[1]See [22] for an English translation.

ASIA CCS'15, April 14–17, 2015, Singapore, Singapore.

http://dx.doi.org/10.1145/2714576.2714587.

elliptic curve over prime fields[2] is the set of points (x, y) satisfying an equation of the following form:

$$y^2 = x^3 + ax + b \mod p,$$

where $a, b \in \mathbb{F}_p$ satisfy $4a^3 + 27b^2 \neq 0 \mod p$. To form a group, an extra infinity point $\mathcal{O}$ is included in this set. Consequently, the elliptic curve $\mathbb{E}(\mathbb{F}_p)$ is defined as

$$\mathbb{E}(\mathbb{F}_p) = \{P = (x, y) | y^2 = x^3 + ax + b \mod p.\ x, y \in \mathbb{F}_p\} \cup \{\mathcal{O}\}.$$

For $m \in \mathbb{F}_p$, we refer to [2] the scalar multiplication (point multiplication) mP of a point P.

Using the notations in [14], SM2DSA (prime fields) is given as follows:

Key Generation: Choose an elliptic curve $\mathbb{E} : y^2 = x^3 + ax + b$ over $\mathbb{F}_p$ where p is a prime. Select a $G \in \mathbb{E}(\mathbb{F}_p) = (x_G, y_G)$ to be a fixed point of order n, where n is a prime. That is $nG = \mathcal{O}$. For a user A, the private key is $d_A \in \mathbb{F}_p$.

Signature Generation:

1. Compute $w = h(M)$, here $M = Z_A \parallel m$, m is the message, Z_A is the hash value about the user, h is the hash algorithm SM3.
2. Randomly choose an integer $k \in [1, n-1]$. k is called a nonce.
3. Calculate $(x_1, y_1) = kG$.
4. Compute $r = w + x_1 \mod n$. If $r = 0$ or $r + k = n$, go to step 2.
5. Compute $s = ((1 + d_A)^{-1}(k - rd_A)) \mod n$. If $s = 0$, go to step 2.
6. Return (r, s) as the signature.

See [14] for the signature verification procedure.

2.2 Template Attack and PCA

Template attack [5] is one of the most powerful side channel attacks; the attacker uses the information of an interval (consists of λ points) of the power traces that are value-correlated. With the assumption that the noise distribution to be Gaussian, the attacker builds a multivariate normal distribution $N(\vec{\mu}, \mathbf{\Sigma})$ from the ε profiled traces for each value, the estimators $(\hat{\vec{\mu}}, \hat{\mathbf{\Sigma}})$ is a template. Denote each trace by $\vec{\mathfrak{t}}_i = (x_i^1, x_i^2, \cdots, x_i^\lambda)$ $(i = 1, 2, \cdots, \varepsilon)$, and X_j to be the random variable of $(x_1^j, x_2^j, \cdots, x_\varepsilon^j)^T$ $(j = 1, 2, \cdots, \lambda)$. Then $\vec{\mu} = (E(X^1), E(X^2), \cdots, E(X^\lambda))$ is the mean vector, and $\mathbf{\Sigma} = cov(X^i, X^j)$ $(1 \leq i, j \leq \lambda)$ is the covariance matrix. In the first phase of template attack, which is named template building phase, the attacker profiles the traces and classifies them according to different processing values, then builds one template for each value. In the second phase, the template matching phase, the attacker uses the maximum likelihood principle to determine the value of the target secret. I.e., the value of the template that maximizes the probability

$$P(\vec{\mathfrak{t}}, N(\hat{\vec{\mu}}, \hat{\mathbf{\Sigma}})) = ((2\pi)^\lambda \det(\hat{\mathbf{\Sigma}}))^{-\frac{1}{2}} \exp(-\frac{1}{2}(\vec{\mathfrak{t}} - \hat{\vec{\mu}})^T \hat{\mathbf{\Sigma}}^{-1} (\vec{\mathfrak{t}} - \hat{\vec{\mu}}))$$

will be chosen to be the one matched with the secret implied in $\vec{\mathfrak{t}}$. Normally, one will calculate the logarithm of this probability to avoid numerical problems[3].

PCA [9] is a linear transformation that transforms the original points to the points that are orthogonal to each other and the variances are sorted. During the PCA transformation, ε traces are first normalized, i.e., $\vec{\mathfrak{t}}_i \leftarrow \vec{\mathfrak{t}}_i - \hat{\vec{\mu}}$ $(i = 1, 2, \cdots, \varepsilon)$. Then the covariance matrix of the normalized traces $\hat{\mathbf{\Sigma}}'$ is calculated, and an eigendecomposition is applied:

$$\hat{\mathbf{\Sigma}}' = \mathbf{U} \times \mathbf{\Lambda} \times \mathbf{U}^T.$$

Finally, the normalized traces are multiplied to the columns of the orthogonal matrix $\mathbf{U}$ that correspond to the several largest eigenvalues in $\mathbf{\Lambda}$ to get the PCA-transformed traces.

2.3 Basics about Lattice

A lattice is a discrete subgroup of $\mathbb{R}^m$ whose elements are integer linear combinations of n $(n \leq m)$ linearly independent vectors. The fundamental parallelepiped $\mathcal{P}_{1/2}(\mathbf{B})$ of a matrix $\mathbf{B} = (\vec{b}_1, \ldots, \vec{b}_m)$ is $\{\sum_{i=1}^m x_i \vec{b}_i : -\frac{1}{2} \leq x_i < \frac{1}{2}\}$. The volume $\mathrm{Vol}(L)$ of a lattice L is the m-dimensional volume of $\mathcal{P}_{1/2}(\mathbf{B})$ for any basis $\mathbf{B}$ of L. Shortest Vector Problem (SVP) and Closest Vector Problem (CVP) are two classical hard problems in computer science. In practice, lattice reduction algorithms like LLL [13] and BKZ [21] are used to solve SVP with dimension not too large.

Hidden Number Problem (HNP). For integer $\mathfrak{s}$ and $v \geq 1$, $\lfloor \mathfrak{s} \rfloor_v$ denotes the remainder of $\mathfrak{s}$ on division by v. For any real number z, let the symbol $|\cdot|_n$ be $|z|_n = \min_{b \in \mathbb{Z}} |z - bn|$. $\mathrm{APP}_{\ell,n}(v)$ denotes any rational number r satisfying $|v - r|_n \leq \frac{n}{2^\ell}$. The HNP [3, 4] asks to recover $\alpha \in \mathbb{Z}_q$, given many approximations $u_i = \mathrm{APP}_{\ell,n}(\alpha t_i)$ where each t_i is known and chosen uniformly at random in $[1, n-1]$, for $1 \leq i \leq d$.

This HNP problem can be reduced to a BDD problem which is a special case of CVP. When we obtain d such t_i, u_i, the reduction to BDD can be done as follows. One constructs the $(d+1)$-dimensional lattice spanned by the following row matrix:

$$\begin{pmatrix} n & 0 & \cdots & 0 & 0 \\ 0 & n & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & n & 0 \\ t_1 & \cdots & \cdots & t_d & \frac{1}{2^\ell} \end{pmatrix} \tag{1}$$

The target vector is $\vec{u} = (u_1, u_2, \ldots, u_d, 0)$. There exists a lattice vector $\vec{h} = (\alpha t_1 + nh_1, \ldots, \alpha t_d + nh_d, \frac{\alpha}{2^\ell})$, such that $\| \vec{h} - \vec{u} \| \leq \sqrt{d+1}\frac{n}{2^\ell}$. Hence, finding $\vec{h}$ discloses α.

In practice, embedding technique [11] is widely used to reduce CVP to SVP. Given a lattice L with basis $\mathbf{B} = [\vec{b}_1, \vec{b}_2, \cdots, \vec{b}_m]$, and a target vector $\vec{u} \in span(\mathbf{B})$, the embedding method is to construct a new lattice L' with basis $\mathbf{B}' = [\vec{b}'_1, \vec{b}'_2, \cdots, \vec{b}'_{m+1}] = [(\vec{b}_1, 0), (\vec{b}_2, 0), \cdots, (\vec{b}_m, 0), (\vec{u}, \beta)]$, where β is a parameter to be determined. If the distance between the target vector and the lattice is small enough, finding the shortest vector in this embedding lattice implies solving the CVP instance. In fact, we expect that the vector $(\vec{h} - \vec{u}, -\beta)$ to be the shortest vector in the embedding lattice, where $\vec{h}$ is the lattice vector closest to $\vec{u}$.

Gaussian Heuristic. Let L be a random d-dimensional lattice of $\mathbb{Z}^n$. Then with overwhelming probability, all the minima of L are asymptotically close to $\sqrt{\frac{d}{2\pi e}}\mathrm{Vol}(L)^{\frac{1}{2}}$. Thus,

[2]This paper only considers the recommended parameters in [20] which is over $\mathbb{F}_p$ $(\log_2 p = 256)$.

[3]To avoid the numerical problem when calculating $\log(\det(\Sigma))$, one can refer to [6].

```
void GetTRN(BYTE *rand, WORD len)
{
  WORD i;
  open RNG clock;
  for (i = 0; i < len; i ++)
  {
    wait until the random number is generated;
    rand[i] = RNGDATA;
  }
  close RNG clock;
}
```

Table 1: Pseudo-code of the random number generation

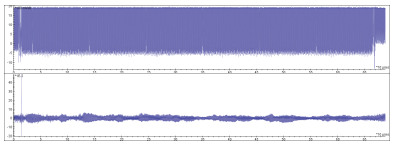

Figure 1: Random number generation by sending APDU (top) and the correlation coefficient between data and traces (bottom)

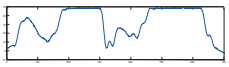

Figure 2: Zoom in the part with the highest correlation in top figure of Fig. 1

it is common in practice to assume that the length of the shortest vector $\vec{v} \in L$ is $\lambda_1(L) \approx \sqrt{\frac{d}{2\pi e}}\mathrm{Vol}(L)^{\frac{1}{2}}$.

3. TEMPLATE BUILDING AND COLLISION DETECTION

The purpose of this section is to illustrate the way to obtain the pairs of signatures that can be used in the partially-sharing nonces attack. We first introduce our target smartcard, and then propose the template attack utilized to obtain the desired signatures.

3.1 The Target Smartcard

The target smartcard has a 8-bit precharged bus whose precharged value is `0x00` or `0xFF` that will not be changed unless the smartcard reboots. This means that before transferring a value, the bus lines will be set to logic 0 or 1 [16]. The two precharged values will be randomly chosen each time the smartcard powers on, which means for a 8-bit value `v`, the power consumption of data moving next time the smartcard powers on can either be the same as itself or the same as that of $\mathtt{v} \oplus \mathtt{0xFF}$. Especially, for the value `0x00`, the power consumption while it is sending over the bus can be the highest or lowest among all the values, which depends on the precharged value; the same situation happens to the value `0xFF`.

To generate the nonce k required by SM2DSA, a random number generator (RNG) is called. Since no particulars about how to generate the random nonces k are stated in [20], our target smartcard generates a 8-bit random number each time with a true random number generator (TRNG). The function `GetTRN()` that generates `len` bytes of random numbers is given in Tab. 1. After 32 bytes of random numbers are generated, they are cascaded to be a big number k with 256 bits. Then k is compared with the modulus n, if $k > n$ then this k is discarded[4] and the random number generation procedure is carried out again, otherwise this k is used as the nonce. Note that for the recommended parameters in [20], n is large enough[5] which means that with very low probability ($\leq 2^{-32}$) k will be discarded.

Note that the line:

`rand[i] = RNGDATA;`

moves the random number from the register `RNGDATA` to a variable `rand[i]` in RAM, and this process may leak information about the moving data due to the `MOV` instruction.

3.2 Building the Templates and Finding the Partial Collisions of Nonces

Note that our target smartcard supports the APDU (Application Protocol Data Unit) for random number generation which also calls the function `GetTRN` and outputs the random numbers. This is the case for most of the real world smartcards as well, for example in the PIN verification application of EMV specifications [7], a get-challenge command should be sent from the terminal to the smartcard by an APDU. The smartcard then generates the random number and returns it to the terminal as the challenge. To make our template attack more powerful, i.e., to avoid the strong assumption that the attacker knows the nonces during the template building phase, the templates will be built by sending the random number generation APDU instead of the signing APDU, to the smartcard. The reason for doing this is that the random numbers, which are needed to build templates, will be returned from the smartcard, thus no known secrets are required.

In order to specify the time interval corresponding to the `MOV` instructions, we send the APDU that generates 8 bytes of random numbers for several thousand times and record the power traces corresponding to the power consumption of the random numbers generation; one of these power traces is illustrated in the top figure of Fig. 1. Then we align all the traces according to the part of the first random byte and calculate the correlation coefficient between the power consumption and the data, see the bottom of Fig. 1. We can see that the points of power traces between 13.6μs to 14.1μs have a relatively strong correlation with the data. Zooming in this part, we get Fig. 2, and we believe that the `MOV` instruction falls in this part.

We also try to align the power traces with the other bytes, however these bytes show lower correlation than the first and thus the first byte of the random number generation is used to mount our attack.[6]

Now the problems we need to handle are how to decide the template building and matching strategies.

First, although the `MOV` instruction leaks the Hamming weight due to the precharged bus, we cannot distinguish the values with the same Hamming weight. Fortunately, there are only one value corresponding to the Hamming weights 0

[4]Note that if this case happens then in our attack the corresponding power trace should be discarded as well.

[5]n=`0xFFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF 7203DF6B 21C6052B 53BBF409 39D54123`

[6]Our earlier experiments tried to detect the collisions of the MSBN by measuring the distances or correlation coefficients directly from the power consumption patterns (i.e., without templates), but ended up with failures due to the low success rate caused by noise.

and 8, respectively. Thus, we only choose the two templates that corresponds to `0x00` and `0xFF` to build.

Second, the partially-sharing(known) nonces attack is a lattice attack that does not tolerate mistakes [17], usually the attack fails due to a single long vector. However, due to the noise, the values are difficult to be recovered without false alarm. A simple example is that two power traces correspond to the same value (e.g. two power traces with the common value `0x00` are on the top in Fig. 3) might have a more significant difference than those with different values (e.g. two power traces with values `0x00` and `0x01` are on the bottom in Fig. 3). These will result in incorrectly estimating the values that are transferred, and the corresponding vector in the lattice might be too long. To overcome this obstacle, our solution is two-folds:

– A signal processing technique, i.e. PCA, is applied to the original traces to amplify the difference of the power consumption between values with different Hamming weights. In [1], Archambeau et al. also proposed to apply PCA to the traces before the template attack, with the purpose of choosing the most interesting points which benefit the template attack. Different from their scenario, we have more traces than the number of points to be considered. As a consequence, we use the original routine as described in Sect. 2.2 to get better results. Moreover, PCA is used in a different way from that of [1]. In [1], PCA is applied inter-class (as stated in [24]), i.e., the mean traces obtained according to different values are used to calculate the covariance matrix $\hat{\mathbf{\Sigma}}'$ and the traces to be matched are also transformed according to the eigendecompose of this matrix. However, in our case, we calculate the covariance also intra-class, that is, we apply PCA to the traces before averaging. In this way, the noise distribution could be better used. For the traces to be matched, an independent PCA is applied instead of using the eigenvectors of the template building phase, in order to better character the distribution of these traces.

– The templates are used in a way that is different from the classical manner as that in [5]. In [5], the templates corresponding to all the values (or all Hamming weights) are built, and the target traces are matched with the templates by the maximum-likelihood principle to decide the value (or Hamming weight) of the secret. Note that in the case of stream ciphers or block ciphers, normally the secret key will keep the same each time we do the encryption. Thus the attacker can use multiple matching traces with the same value to enhance the success rate. However, in our case, the nonce varies each time so that we have only a single trace to match for each value. Remember that we only use two templates; thus we look through the traces to find the matched value for each of the two templates, instead of searching over the templates for each trace. Moreover, we decide not to use the maximum-likelihood principle that are used commonly in the template attack. Instead, we use an opposite approach; the "minimum-likelihood principle" is our decision principle that detects the values with the lowest Bayesian probability. The reason is that in the experiment we found that the success rate is higher in this case. Specifically, for the power traces after signal processing, we match them with each of the two templates corresponding to `0x00` and `0xFF`. Then we sort the traces with the log-likelihood and choose the minimum ones. Usually, we can set a threshold and keep only the ones below this threshold; these traces are expected to have the values that are the same as (when the precharged values of the template building and matching phases are different) or the complement of (when the precharged values of the template building and matching phases are the same) that of the templates. The signatures chosen according to the same template can then be paired to be used in the lattice attack.

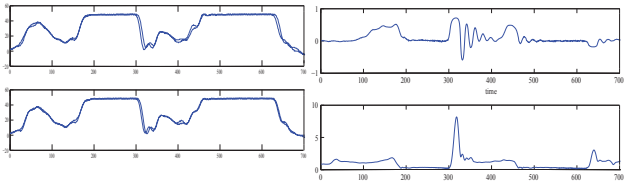

Figure 3: Overlap of two traces (top: traces with the same value; bottom: traces with different values)

Figure 4: Correlation coefficient (top) and standard deviation (bottom)

Following the above technique, we now describe the detailed steps of our template building and matching phases.

Template Building Phase.

1. Keep on sending the APDU for random number generation to the smartcard and sort the recorded traces according to the first byte of the random numbers. For 1,800,000 random numbers, we obtain about 7,000 traces for each of the 256 values.
2. Align all the kept traces and then apply PCA to them.

In order to enhance the efficiency, PCA should be only applied to the "sensitive points" in the traces that are correlated to the values. This can be easily done by calculating the correlation coefficients between the values and the trace points, since we know the random numbers. However, this might cause a problem that in the matching phase, when one has to do the same transformation, i.e. PCA, to the corresponding part of the analyzed traces, he has no idea about how to choose the proper points due to the unknown nonces. As an alteration, we will calculate the standard deviations of each point of the template traces, and try to find the relation between the standard deviations and the correlation coefficients (see Fig. 4). We know from Fig. 4 that the peak of the standard deviations appears at the same point as that of the correlation coefficients.

Then we select a number of points before and after the peak of the standard deviation, to which PCA is applied. See Fig. 5 for the power traces of values with different Hamming weights after PCA, which shows more significant difference than those before PCA.

3. For the traces corresponding to the values `0x00` and `0xFF` after PCA transformation, two templates are built by selecting a certain number of points with high variances. Since the eigendecomposition used in our PCA has sorted the eigenvalues, we select the points on the right hand side whose variances are large.

Template Matching Phase. In the matching phase, the APDU that corresponds to the signing command is sent to the smartcard, a power trace of the whole signing procedure is shown in Fig. 6. The part on the power trace that corresponds to the generation and transfer of k is marked; in the very beginning of this part, we found a very similar interval as that in Fig. 2. We believe that this is the part that we

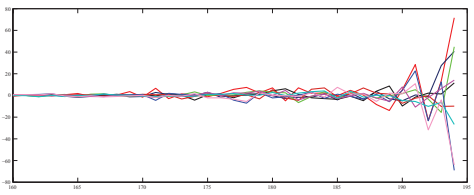

Figure 5: Traces of values with different Hamming weights after PCA

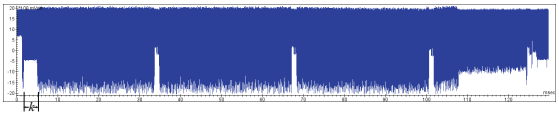

Figure 6: Power consumption of SM2 signing procedure

should use to match the template, thus we keep only this part of the power trace for each signature.

In order to collect enough signatures, we have to repeatedly send the signing APDU to the smartcard. To make the attack more reasonable, we will not assume that we can collect enough signatures within a single power-on period of the smartcard. So we only send the APDU 10,000 times to the smartcard before it powers off. For these traces, we select the interval that is the same as the template building phase by calculating the standard deviation and then apply PCA to them. For the traces after PCA, we match them with the template. A minimum likelihood principle is applied as mentioned, and a threshold is chosen to make the false positives as few as possible. After we find the matching traces, we pair these signatures and then collect the next 10,000 signatures to repeat the procedure. Since the values we targeted occur with probability 2/256, and we have a strict threshold so as to reduce the errors, in our experiment there were only $3 \sim 4$ power traces kept on average for each of the two templates. As a consequence, we can get about 4 pairs of signatures with the same MSBN from 10,000 signatures.

4. PARTIALLY-SHARING NONCES ATTACK ON SM2DSA

In this section, we describe the lattice attack that uses the partially-sharing nonces acquired in the previous section to recover the private key.

For a pair of signatures (r_1, s_1) and (r_2, s_2) that shares MSBN, we have:

$$s_1(1+d_A) = k_1 - r_1 d_A \mod n\ , \quad (2)$$
$$s_2(1+d_A) = k_2 - r_2 d_A \mod n\ ; \quad (3)$$

subtracting Eq. (3) from Eq. (2), then:

$$(s_1 - s_2 + r_1 - r_2)d_A - (s_2 - s_1) = k_1 - k_2 \mod n. \quad (4)$$

We write $k_1 = 2^l k_1^H + k_1^L$, $k_2 = 2^l k_2^H + k_2^L$, where $l = 248$ and k^H is the most significant byte of k. Since in our case $k_1^H = k_2^H$, Eq. (4) can be rewritten as:

$$(s_1 - s_2 + r_1 - r_2)d_A - (s_2 - s_1) = k_1^L - k_2^L \mod n.$$

Since $k_1^L, k_2^L \in [0, 2^l]$, we have

$$-2^l \le \lfloor (s_1 - s_2 + r_1 - r_2)d_A - (s_2 - s_1) \rfloor_n \le 2^l \quad (5)$$

If we can obtain d pairs of signatures $(((r_1^1, s_1^1), (r_2^1, s_2^1)), \cdots, ((r_1^d, s_1^d), (r_2^d, s_2^d)))$ with collided MSBN, then they all fulfill Eq. (5); the recovery of private key d_A can be seen as an HNP problem. Denote $t_i = s_1^i - s_2^i + r_1^i - r_2^i$, $u_i = s_2^i - s_1^i$ $(i = 1, \cdots, d)$.

In our experiments, we use embedding method as shown in Sect. 2.3 to solve the CVP. To avoid the fraction in computation and balance the values in coordinates, one needs to modify the coefficients of the row matrix (1) by some scaling factor. As a result, we construct a CVP in the lattice below:

$$\begin{pmatrix} 1732 \cdot n & 0 & \cdots & 0 & 0 \\ 0 & 1732 \cdot n & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & 1732 \cdot n & 0 \\ 1732 \cdot t_1 & \cdots & \cdots & 1732 \cdot t_d & 4 \end{pmatrix}$$

The target vector is $\vec{u} = (1732 \cdot u_1, 1732 \cdot u_2, \ldots, 1732 \cdot u_d, 0)$, and the parameter β in lattice $\mathbf{B}'$ (Sect. 2.3) is chosen to be $4n$. We expect that the vector $\vec{h} = (1732(d_A t_1 + n h_1 - u_1), \ldots, 1732(d_A t_d + n h_d - u_d), 4d_A, 4n)$ which discloses d_A is the short vector in the embedding lattice. Now, we give a heuristic theoretical result about the number of signature pairs needed to recover the private key: The upper bound of the length of the vector $\vec{h}$ is about $\sqrt{d(1732 \times \frac{n}{2^8})^2 + 32n^2}$; by Gaussian heuristic the expected length of shortest vector is about $\sqrt{\frac{d+2}{2\pi e}}(16 \times 1732^d \times n^{d+1})^{\frac{1}{d+2}}$. When the length of vector $\vec{h}$ is shorter than that of Gaussian heuristic, d_A can be obtained by finding the short vector of the embedding lattice. Here $n \approx 2^{256}$, then we can achieve the bound as long as $d \ge 45$. Since d is relatively small, the lattice basis reduction algorithm BKZ with blocksize 20 can be seen as an SVP oracle. That is, the BKZ 20 algorithm will output the shortest vector for lattice with dimension not much larger than 50.

Experimental Results. Our smartcard runs SM2DSA at 32MHz; a sampling rate of 1G/s and a 1G band width are used when we are collecting the power traces.

To enhance the success rate of the attack, similar to [17], we collect more pairs of signatures than the number calculated from Gaussian heuristic, say 48. The reason is to avoid the failure of the attack due to the false detection of the collision, although the probability of falsely detecting is very low due to our strategy in Sect. 3.

To get 48 pairs of collided signatures, we need about 120,000 signatures in the template matching phase. For our target smartcard, about one hour is required to run 10,000 signings; thus less than 13 hours are needed to process the 120,000 ones. In the template building phase, only the random number generation command should be run, which is much faster than the signing process. In our experiment, we need about 12 hours to get the 1,800,000 traces for template building.

From these 48 pairs of signatures, we randomly choose 45 pairs to form a CVP and then solve the embedding lattice by BKZ 20 algorithm implemented in NTL library [23]. In the worst case, there are about $\binom{48}{45} = 17296$ BKZ reductions with dimension 45 have to be carried out, but this is still feasible since one reduction takes about 2.4 seconds with one core of Intel Xeon CPU W3530 (2.8GHz) and 4G memory. Actually, we recovered the correct private key immediately

after we tried the first group of 45 pairs of signatures, and it turned out that the collided pairs we collected were all correct when we figured out all the nonces for the 48 pairs of signatures.

Furthermore, the previous bound obtained from Gaussian heuristic is an upper bound and we expect that fewer signatures are actually needed to recover the private key. In our experiment, we found that BKZ actually performed better than the bound that we estimated: Less than 40 pairs of signatures were enough to recover the private key. This means that the attacker can collect fewer signature pairs so as to reduce the processing time.

Note that in the experiment, we found that the private key came from the second shortest vector of the embedding lattice, since the shortest vector would always be all zero but the $(d+1)$-th dimension. Let the $(d+1)$-th dimension of the second shortest vector be d'_A, then the private key is $d'_A/4 \mod n$ or $n - (d'_A/4 \mod n)$.

5. COUNTERMEASURES

Common countermeasures of hiding [16] will make our attack more difficult to succeed. These countermeasures include increasing the background noise, balancing the power consumption, randomizing the internal clock, random delay, etc. All the countermeasures could be applied to the random number generation and moving process.

Another countermeasure is to generate the nonce k with the method as specified in Appendix B.5.1 of [18]. In this method, a random number with extra 64 random bits will be produced and then it will be reduced before being used as the nonce. In this scenario, the information obtained in our attack is no longer the same as that of the nonces; thus our attack is invalid in this case.

6. CONCLUSION

This paper implements an attack on SM2DSA by using the information leaked from the generation of the nonces. By using the collisions of the first bytes of the nonces detected with the assistance of templates, we give a partially-sharing nonces attack on SM2DSA with the recommended parameters. Possible countermeasures are also suggested.

Our attack can also be applied to SM2DSA with the other parameters and the other elliptic curve based signature algorithms like ECDSA, as long as the transfer of the random numbers leaks the information of the nonce k.

Acknowledgments. The authors would like to thank the anonymous reviewers for their valuable comments. This work is supported by National Natural Science Foundation of China (No. 61402536, No. 61202493 and No. 61402252) and "12th Five-Year Plan" The National Development Foundation for Cryptological Research (No. MMJJ201401009).

7. REFERENCES

[1] C. Archambeau, E. Peeters, F.-X. Standaert, and J.-J. Quisquater. Template Attacks in Principal Subspaces. In L. Goubin and M. Matsui, editors, *CHES 2006*, volume 4249 of *LNCS*, pages 1–14. Springer, 2006.

[2] R. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. CRC Press, Boca Raton, 2005.

[3] D. Boneh and R. Venkatesan. Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes. In N. Koblitz, editor, *CRYPTO 1996*, volume 1109 of *LNCS*, pages 129–142. Springer, 1996.

[4] D. Boneh and R. Venkatesan. Rounding in Lattices and Its Cryptographic Applications. In M. E. Saks, editor, *SODA 1997*, pages 675–681. ACM/SIAM, 1997.

[5] S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In B. S. K. Jr., Çetin Kaya Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, 2002.

[6] O. Choudary and M. G. Kuhn. Efficient Template Attacks. In A. Francillon and P. Rohatgi, editors, *CARDIS 2013*, volume 8419 of *LNCS*, pages 253–270. Springer, 2014.

[7] EMV. Integrated Circuit Card Specifications for Payment Systems. Version 4.3 (November 2011). `http://www.emvco.com`.

[8] J.-C. Faugère, C. Goyet, and G. Renault. Attacking (EC)DSA Given Only an Implicit Hint. In L. R. Knudsen and H. Wu, editors, *Selected Areas in Cryptography 2012*, volume 7707 of *LNCS*, pages 252–274. Springer, 2013.

[9] H. Hotelling. Analysis of a Complex of Statistical Variables Into Principal Components. *J. Educ. Psych.*, 24, 1933.

[10] N. Howgrave-Graham and N. P. Smart. Lattice Attacks on Digital Signature Schemes. *Des. Codes Cryptography*, 23(3):283–290, 2001.

[11] R. Kannan. Minkowski's Convex Body Theorem and Integer Programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.

[12] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *CRYPTO 1996*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.

[13] A. Lenstra, J. Lenstra, H.W., and L. Lovĺćsz. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[14] M. Liu, J. Chen, and H. Li. Partially Known Nonces and Fault Injection Attacks on SM2 Signature Algorithm. In D. Lin, S. Xu, and M. Yung, editors, *Inscrypt 2013*, volume 8567 of *LNCS*, pages 343–358. Springer, 2014.

[15] M. Liu and P. Q. Nguyen. Solving BDD by Enumeration: An Update. In E. Dawson, editor, *CT-RSA 2013*, volume 7779 of *LNCS*, pages 293–309. Springer, 2013.

[16] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.

[17] D. Naccache, P. Q. Nguyen, M. Tunstall, and C. Whelan. Experimenting with Faults, Lattices and the DSA. In S. Vaudenay, editor, *Public Key Cryptography 2005*, volume 3386 of *LNCS*, pages 16–28. Springer, 2005.

[18] National Institute of Standards and Technology (NIST). FIPS Publication 186-4: Digital Signature Standard, 2013.

[19] P. Q. Nguyen and I. Shparlinski. The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces. *Des. Codes Cryptography*, 30(2):201–217, 2003.

[20] Office of State Commercial Cryptography Administration. Public Key Cryptographic Algorithm SM2 Based on Elliptic Curves (in Chinese). `http://www.oscca.gov.cn/News/201012/News_1198.htm`.

[21] C. P. Schnorr and M. Euchner. Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems. *Mathematics of Programming*, 66:181–191, 1994.

[22] S. Shen and X. Lee. SM2 Digital Signature Algorithm draft-shen-sm2-ecdsa-01. `http://tools.ietf.org/pdf/draft-shen-sm2-ecdsa-01.pdf`.

[23] V. Shoup. Number Theory C++ Library (NTL) version 5.5.2. `http://www.shoup.net/ntl/`.

[24] F.-X. Standaert and C. Archambeau. Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages. In E. Oswald and P. Rohatgi, editors, *CHES 2008*, volume 5154 of *LNCS*, pages 411–425. Springer, 2008.