A New Attack with Side Channel Leakage during Exponent Recoding Computations

Yasuyuki Sakai[1] and Kouichi Sakurai[2]

[1] Mitsubishi Electric Corporation,
5-1-1 Ofuna, Kamakura, Kanagawa 247-8501, Japan
ysakai@iss.isl.melco.co.jp
[2] Kyushu University,
6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan
sakurai@csce.kyushu-u.ac.jp

Abstract. In this paper we propose a new side channel attack, where exponent recodings for public key cryptosystems such as RSA and ECDSA are considered. The known side channel attacks and countermeasures for public key cryptosystems were against the main stage (square and multiply stage) of the modular exponentiation (or the point multiplication on an elliptic curve). We have many algorithms which achieve fast computation of exponentiations. When we compute an exponentiation, the exponent recoding has to be carried out before the main stage. There are some exponent recoding algorithms including conditional branches, in which instructions depend on the given exponent value. Consequently exponent recoding can constitute an information channel, providing the attacker with valuable information on the secret exponent. In this paper we show new algorithms of attack on exponent recoding. The proposed algorithms can recover the secret exponent, when the width-w NAF [9] and the unsigned/signed fractional window representation [5] are used.

Keywords: Side channel attack, exponent recoding, RSA cryptosystems, elliptic curve cryptosystems.

1 Introduction

Smart cards are one of the major application fields of cryptographic algorithms, and may contain sensitive data, such as RSA private key. Some implementations of cryptographic algorithms often leak "*side channel information.*" Side channel information includes power consumption, electromagnetic fields and timing to process. Side channel attacks, which use side channel information leaked from real implementation of cryptographic algorithms, were first introduced by Kocher, Jaffe and Jun [2, 3]. Side channel attacks can be often much more powerful than mathematical cryptanalysis. Thus, many papers on side channel cryptanalysis have been published.

RSA based cryptosystems and elliptic curve based cryptosystems require computation of the modular exponentiation (or the point multiplication on an elliptic curve). The computational performance of cryptographic protocols with public key cryptosystems strongly depends on the efficiency of the modular exponentiation procedure. Therefore, it is very attractive to provide algorithms that allow efficient implementation. One approach to reducing the computational complexity of the modular exponentiation is to replace the binary representation of the given exponent with a representation which has fewer non-zero terms. Transforming an exponent from one representation to another is called "*exponent recoding*."

Since the concept of the side channel attacks was firstly proposed by Kocher, Jaffe and Jun [2, 3], various methods of attacks and countermeasures have been proposed. For example related to public key cryptosystems, Coron proposed three concepts of countermeasures for differential power analysis (DPA) on elliptic curve cryptosystems [1]: 1) randomization of the secret exponent, 2) blinding the point, 3) randomized projective coordinates. Side channel information leaked from cryptographic devices may provide much information about the operations that take place and involved parameters. Known power analysis in the literature were related to modular exponentiation (or elliptic point multiplication).

If we carefully observe the exponentiation procedure, we can find one more stage in which instructions depend on the secret exponent. Some efficient modular exponentiations have to perform the exponent recoding in advance, then the main stage (square and multiply stage) of the modular exponentiation is computed. Since the exponent recoding may be executed with conditional branches depending on the secret exponent value, the computation of the exponent recoding can constitute an information channel which provides valuable information for the attacker. In this paper, we discuss side channel information leaked from cryptographic devices during exponent recoding computation. We will show methods of attacks on the exponent recoding for width-w NAF [9] and signed/unsigned fractional window representation [5].

The rest of this paper is organized as follows. Section 2 will give a brief description of exponent recoding algorithms. In Section 3, information leakage during exponent recoding will be discussed. Section 4 gives algorithms of attacks on the exponent recodings. Finally, Section 5 gives our conclusion.

2 Exponent Recoding

In this section we will give a brief description of exponent recodings. In some efficient modular exponentiations, an exponent recoding has to be performed in advance, then the main stage (square and multiply stage) of the modular exponentiation is computed. In the exponent recoding stage, the binary representation of the given exponent is replaced by a representation which has fewer non-zero terms. We will examine three methods of exponent recoding: width-w NAF [9] and signed/unsigned fractional window representation [5].

2.1 NAF

The signed binary representation, which is also referred to as non-adjacent form (NAF), is a "*sparse*" signed binary representation of an integer. An algorithm of recoding to NAF is given in Algorithm 1 [8]. Step 3 and 4 of Algorithm 1 can be carried out with a table which contains all possible inputs to i-th iteration of Step 3 and 4 and the corresponding outputs. Algorithm 2 also generates NAF, and is equivalent to Algorithm 1.

The average density achieved by NAF is 1/3 for exponent $d \to \infty$.

Algorithm 1 Exponent recoding for NAF

`Input` a non-negative t-bit integer $d = (d_{t-1} \cdots d_0)_2$
`Output` $b = (b_t \cdots b_0)$, where $b_i \in \{-1, 0, 1\}$ and $b_i b_{i+1} = 0$ for all i
1. $c_0 \leftarrow 0$, $d_{t+1} \leftarrow 0$, $d_t \leftarrow 0$
2. `for` i `from` 0 `to` t `do`
3. $\quad c_{i+1} \leftarrow \lfloor (c_i + d_i + d_{i+1})/2 \rfloor$
4. $\quad b_i \leftarrow c_i + d_i - 2c_{i+1}$
5. `end for`
6. `Return` $b = (b_t \cdots b_0)$

Algorithm 2 Exponent recoding for NAF: a variant

`Input` a non-negative t-bit integer $d = (d_{t-1} \cdots d_0)_2$
`Output` $b = (b_t \cdots b_0)$, where $b_i \in \{-1, 0, 1\}$ and $b_i b_{i+1} = 0$ for all i
1. $b = (s_t \cdots s_0 d_0)_2 \leftarrow 3d$
2. $b \leftarrow b - 2d$ $\;(b_i = s_i - d_{i+1}$ with $0 - 1 = \tilde{1}$, where $\tilde{1}$ denotes $-1)$
3. `Return` $b = (b_t \cdots b_0)$

2.2 Width-w NAF

Width-w NAF, proposed by independently Solinas [9] and Cohen, can be viewed as a generalization of NAF. The case $w = 2$ is that of the ordinary NAF. Algorithm 3 is a typical implementation of the exponent recoding for width-w NAF representation.

The average density achieved by width-w NAF is $1/(w + 1)$ for exponent $d \to \infty$.

Algorithm 3 Exponent recoding for the width-w NAF

`Input` a non-negative t-bit integer d, an integer $w \geq 2$
`Output` $b = (b_t \cdots b_0)$, where $b_i \in \{0, \pm 1, \pm 3, \cdots, \pm(2^{w-1} - 1)\}$ and among any w consecutive $b_i s$, at most one is nonzero.
1. $c \leftarrow d$, $i \leftarrow 0$

```
while c > 0 do
  if c odd then
     b_i ← c mod 2^w
     if b_i ≥ 2^(w−1) then
        b_i ← b_i − 2^w
     end if
     c ← c − b_i
  else
     b_i ← 0
  end if
  c ← ⌊c/2⌋, i ← i + 1
end while
Return b = (b_t ··· b_0)
```

2.3 Fractional Window

In small devices, the choice of w for exponentiation using the width-w NAF may be dictated by memory limitations. Möller proposed a *fractional window* technique [5], which can be viewed as a generalization of the sliding window and width-w approach. The fractional window exponentiation has better flexibility of the table size. See [5] for details.

The fractional window method of exponentiation has two variants. One is the signed version, where negative digits are allowed. The other is the unsigned variant of the fractional window method for the case that only non-negative digits are permissible. The recoding methods for unsigned fractional window representation and signed fractional window representation are described in Algorithm 4 and 5, respectively.

Let $w \geq 2$ be an integer and m an odd integer such that $1 \leq m \leq 2^w - 3$. The average density of signed fractional window representations with parameters w and m is

$$\frac{1}{w + \frac{m+1}{2^w} + 2}$$

for $d \to \infty$ [5]. The average density of unsigned fractional window representations with parameters w and m is

$$\frac{1}{w + \frac{m+1}{2^w} + 1}$$

for $d \to \infty$ [5].

Algorithm 4 Exponent recoding for the unsigned fractional window representation

Input a non-negative integer d, an integer $w \geq 2$, an odd integer m, $1 \leq m \leq 2^w - 3$
Output $b = (b_{i-1} \cdots b_0)$, $b_i \in \{0, 1, 3, \cdots, 2^w + m\}$

$c \leftarrow d$, $i \leftarrow 0$
while $c > 0$ **do**
  if c odd **then**
    $b_i \leftarrow c \bmod 2^{w+1}$
    if $2^w + m < b_i$ **then**
      $b_i \leftarrow b_i - 2^w$
    end if
    $c \leftarrow c - b_i$
  else
    $b_i \leftarrow 0$
  end if
  $c \leftarrow \lfloor c/2 \rfloor$, $i \leftarrow i + 1$
end while
Return $b = (b_{i-1} \cdots b_0)$

Algorithm 5 Exponent recoding for the signed fractional window representation

Input a non negative integer d, an integer $w \geq 2$, an odd integer m, $1 \leq m \leq 2^w - 3$
Output $b = (b_{i-1} \cdots b_0)$, $b_i \in \{0, \pm 1, \pm 3, \cdots, \pm(2^w + m)\}$

$c \leftarrow d$, $i \leftarrow 0$
while $c > 0$ **do**
  if c odd **then**
    $b_i \leftarrow c \bmod 2^{w+2}$
    if $2^w + m < b_i < 3 \cdot 2^w - m$ **then**
      $b_i \leftarrow b_i - 2^{w+1}$
    else if $3 \cdot 2^w - m \leq b_i < 2^{w+2}$ **then**
      $b_i \leftarrow b_i - 2^{w+2}$
    end if
    $c \leftarrow c - b_i$
  else
    $b_i \leftarrow 0$
  end if
  $c \leftarrow \lfloor c/2 \rfloor$, $i \leftarrow i + 1$
end while
Return $b = (b_{i-1} \cdots b_0)$

3 Information Leakage during Exponent Recoding

As we have seen in Section 2, exponent recodings of Algorithms 3, 4 and 5 have conditional branches during the computation.

While the effects of the conditional branch might be difficult to identify from direct observations of a device's power consumption, the statistical operations used in DPA are able to reliably identify extraordinarily small differences in power consumption [4]. In this section we will discuss the information leaked from cryptographic devices during the exponent recodings.

3.1 The Model of the Attacker

We assume the model of the attacker as follows.

- The attacker has access to a device which calculates exponent recoding.
- The attacker has knowledge about the implementation, i.e., he knows the algorithm, including the parameter w and m, of the exponent recoding implemented on the device.
- The attacker can distinguish the conditional branch in Algorithm 3, 4 and 5 by monitoring power consumption during the computation of the exponent recoding.

The goal of the attacker is:

- Recovering the secret exponent.

3.2 Leakage during Exponent Recoding

Width-w NAF recoding (Algorithm 3) has two conditional branches, Step 3 and 5, in the main loop. If the symbol in the window of width-w is odd, Step 4 through 8 should be performed. Else if the symbol is even, Step 10 should be performed. When the symbol is odd, inner conditional branch will be evaluated, then if the symbol $b_i \geq 2^{w-1}$, b_i has to be subtracted by 2^w. Consequently the execution path of width-w recoding branches into three cases. In the execution path, subtraction instruction have to be performed depending on the exponent value.

We define a "$\mathcal{SN}$-sequence" as follows.

- At the i-th loop of the main loop, if the subtraction instruction is performed two times, we call the observed power consumption $\mathcal{SSN}$.
- If the subtraction instruction is performed one time, we call the observed power consumption $\mathcal{SN}$.
- If no subtraction instruction is performed, we call the observed power consumption $\mathcal{N}$.

$\mathcal{SN}$-sequence is a series of $\mathcal{SSN}$, $\mathcal{SN}$ and $\mathcal{N}$ observed during the computation of the exponent recoding:

$$\mathcal{SN}\text{-sequence} := \{\mathcal{SSN}, \mathcal{SN}, \mathcal{N}\}^*$$

The attacker can obtain $\mathcal{SN}$-sequences by monitoring power consumption. A $\mathcal{SN}$-sequence is the side channel information leaked from cryptographic devices.

As in the case of width-w NAF recoding, both the unsigned/signed fractional recoding (Algorithm 4, 5) have conditional branches depending on the exponent value. Hence the attacker can observe $\mathcal{SN}$-sequences during the computation of the recoding.

It is straight forward to implement the basic NAF recoding without conditional branches as described in Algorithm 1 with a look-up table. Algorithm 2 is also a secure implementation of NAF recoding.

4 The Attacks

In this section we describe new attacks on exponent recodings for width-w NAF and signed/unsigned fractional window representation given in Section 2. The attacker tries to recover the secret exponent from observed $\mathcal{SN}$-sequences. We assume that exponent recodings are implemented by Algorithm 3, 4 and 5.

4.1 Basic Strategy

As we have already mentioned, execution of width-w recoding has three branches. For a given exponent, the $\mathcal{SN}$-sequence is uniquely determined. For example, when width-3 NAF recoding is implemented by Algorithm 3, $\mathcal{SN}$-value ($\in \{\mathcal{SSN}, \mathcal{SN}, \mathcal{N}\}$) in each i-th loop will be as the following Table 1.

Table 1. Relation between data in the window and resulting $\mathcal{SN}$-value: width-3 NAF

data in the window	mapped digit b_i	$\mathcal{SN}$-value
$0 = (000)_2$	0	$\mathcal{N}$
$1 = (001)_2$	1	$\mathcal{SN}$
$2 = (010)_2$	0	$\mathcal{N}$
$3 = (011)_2$	3	$\mathcal{SN}$
$4 = (100)_2$	0	$\mathcal{N}$
$5 = (101)_2$	-3	$\mathcal{SSN}$
$6 = (110)_2$	0	$\mathcal{N}$
$7 = (111)_2$	-1	$\mathcal{SSN}$

Basically an attack can be constructed based on Table 1. However, there exist a difficulty in guessing the secret exponent value from given $\mathcal{SN}$-sequence. While the $\mathcal{SN}$-sequence is uniquely determined from the given exponent, the converse is not true. When $\mathcal{SN}$ is observed, the data in the window should be $(001)_2$ or

$(011)_2$. Therefore the attacker can decide the LSB and MSB in the window, but the middle bit can not be guessed uniquely. The same situation happens when $\mathcal{SSN}$ is observed.

The second difficulty is the "*carry*." When the data in the window is odd and greater than 2^{w-1}, mapped digit b_i has to be subtracted by 2^w, then the resulting b_i (Step 6) will have negative value. The following subtraction instruction (Step 8) should be addition and a carry will always occur. Consequently, at the next $(i+1)$-th loop, the temporary exponent c will has a different bit-pattern from the originally given exponent d. For example, assume $45 = (101101)_2$ is the given exponent d and width-3 NAF recoding is performed. At the first loop, mapped digit b_i should be -3 (Step 6). Then at Step 8, the temporary exponent c should be $48 = (110000)_2$. Since the carry occurs, at the later loop, observed $\mathcal{SN}$-value can not be a direct information of the exponent value.

We have to construct attacks in consideration of above two observations.

4.2 An Attack on Width-w NAF Recoding

We show an attack on width-w NAF recoding. The algorithm of the attack is based on the following observations.

- In the case of $\mathcal{SSN}$, a carry occurs at Step 8.
- In the case of $\mathcal{SSN}$ and all the passed values after previously observed $\mathcal{SSN}$ were $\mathcal{N}$s, i.e., sub-sequence is $(\mathcal{SSN}, \mathcal{N}, \mathcal{N}, \cdots, \mathcal{N}, \mathcal{SSN})$, the carry was transmitted to the current $\mathcal{SSN}$.
- In the case of $\mathcal{SSN}$ and if a carry is transmitted, the attacker should guess $d_i = 0$. Otherwise if a carry is not transmitted, the attacker should guess $d_i = 1$.
- In the case of $\mathcal{SSN}$, the attacker should guess $d_{i+w-1} = 1$.
- In the case of $\mathcal{N}$ and a carry is transmitted, the attacker should guess $d_i = 1$. Otherwise if a carry is not transmitted, the attacker should guess $d_i = 0$.
- In the case of $\mathcal{SN}$, the attacker should guess d_i as the same strategies as in the case of $\mathcal{SSN}$.
- In the case of $\mathcal{SN}$ and the length of the rest bits to be guessed is smaller than the window width w, the attacker should guess $d_{t-1} = 1$ by definition such that MSB of d is always "1".
- In the case of $\mathcal{SN}$, the transmission of a carry stops at this place.

The attack is described in Algorithm 6. The symbol "*state*" is used for the consideration of a carry. If the data of the middle in the window can not be guessed uniquely, the symbol "*unknown*" is used.

Algorithm 6 An attack on width-w NAF recoding

`Input` $\mathcal{SN}$-sequence $(v_0, v_1, \cdots)$, where $v_i \in \{\mathcal{SSN}, \mathcal{SN}, \mathcal{N}\}$, an integer $w \geq 2$, t =bitlength of d

`Output` an exponent $d = (d_{t-1} \cdots d_0)_2$

```
i ← 0
state ← 0
while i < t do
   if v_i = SSN then
      if state = 1 then d_i ← 0
      else if state = 0 then d_i ← 1
      for j from i + 1 to i + w − 2 do d_i ← "unknown"
      d_{i+w−1} ← 1
      i ← i + w
      state ← 1
   else if v_i = SN then
      if i + w − 1 > t
         if state = 1 then d_i ← 0
         else if state = 0 then d_i ← 1
         for j from i + 1 to t − 2 do d_i ← "unknown"
         d_{t−1} ← 1
         i ← t
      else
         if state = 1 then d_i ← 0
         else if state = 0 then d_i ← 1
         for j from i + 1 to i + w − 2 do d_i ← "unknown"
         d_{i+w−1} ← 0
         i ← i + w
      end if
      state ← 0
   else if v_i = N then
      if state = 1 then d_i ← 1
      else if state = 0 then d_i ← 0
      i ← i + 1
   end if
end while
Return d = (d_{t−1} ··· d_0)_2
```

We can evaluate that how many bits can be recovered from observed $\mathcal{SN}$-sequences in the case of the width-w NAF recoding.

Theorem 1. *Assume that width-w NAF recoding is implemented by Algorithm 3 and that $\mathcal{SN}$-sequence can be observed during the recoding. The ratio of successfully recovered bits can be evaluated by $3/(w+1)$ for $t \to \infty$.*

Proof. It is easy to prove from the fact that the density of width-w NAF recoding is $1/(w+1)$ for $t \to \infty$. $\square$

4.3 An Attack on Unsigned Fractional Window Recoding

We show an attack on the unsigned fractional window recoding. Similar strategies as in the width-w recoding case can be applied. Data dependent subtraction instructions have to be carried out at Step 6 and 8 of Algorithm 4. Therefore, the attacker can observe $\mathcal{SN}$-sequence during the recoding. The difference on the strategies from the case of width-w NAF is that even when the subtraction instruction at Step 8 is carried out, no carry occur.

The attack is shown in Algorithm 7.

Remark 1. Even if the case of "*unknown*", the probability of "0" or "1" may not be equal in some cases depending on the parameter m. In such a case we can modify the attack with weighted probability of guessed value. In appendix A relations between successive unknown bits are described.

Algorithm 7 An attack on unsigned fractional window recoding

`Input` $\mathcal{SN}$-sequence $(v_0, v_1, \cdots)$, $v_i \in \{\mathcal{SSN}, \mathcal{SN}, \mathcal{N}\}$, an integer $w \geq 2$, an odd integer m, $1 \leq m \leq 2^w - 3$, t =bitlength of d
`Output` an exponent $d = (d_{t-1} \cdots d_0)_2$

$i \leftarrow 0$
`while` $i < t$ `do`
  `if` $v_i = \mathcal{SSN}$ `then`
    $d_i \leftarrow 1$
    `for` j `from` $i+1$ `to` $i+w-1$ `do` $d_i \leftarrow$ "*unknown*"
    $i \leftarrow i + w$
  `else if` $v_i = \mathcal{SN}$ `then`
    $d_i \leftarrow 1$
    `if` $i + w + 1 > t$
      `for` j `from` $i+1$ `to` $t-2$ `do` $d_i \leftarrow$ "*unknown*"
      $d_{t-1} \leftarrow 1$
      $i \leftarrow t$
    `else`
      `for` j `from` $i+1$ `to` $i+w$ `do` $d_i \leftarrow$ "*unknown*"
      $i \leftarrow i + w + 1$
    `end if`
  `else if` $v_i = \mathcal{N}$ `then`
    $d_i \leftarrow 0$
    $i \leftarrow i + 1$
  `end if`
`end while`
`Return` $d = (d_{t-1} \cdots d_0)_2$

4.4 An Attack on Signed Fractional Window Recoding

An attack on the signed fractional window recoding is shown in Algorithm 8. The similar strategies as width-w recoding can be applied, but handling of the transmission of a carry should be more complicated. Only when $w+1$ continuous $\mathcal{N}$s are observed after $\mathcal{SSN}$, a carry may be transmitted to $\mathcal{SN}$ or $\mathcal{SSN}$. The variable c in Algorithm 8 is used to store the number of continuous $\mathcal{N}$.

Algorithm 8 An attack on signed fractional window recoding

`Input` $\mathcal{SN}$-sequence $(v_0, v_1, \cdots)$, $v_i \in \{\mathcal{SSN}, \mathcal{SN}, \mathcal{N}\}$, an integer $w \geq 2$ an odd integer m, $1 \leq m \leq 2^w - 3$, t =bitlength of d
`Output` an exponent $d = (d_{t-1} \cdots d_0)_2$

$i \leftarrow 0$
$state \leftarrow 0$
$c \leftarrow 0$
`while` $i < t$ `do`
  `if` $v_i = \mathcal{SSN}$ `then`
    `if` $c \geq w + 1$ `and` $state = 1$ `then` $d_i \leftarrow 0$
    `else if` $c = w$ `and` $state = 1$ `then` $d_i \leftarrow$ "*unknown*"
    `else` $d_i \leftarrow 1$
    `for` j `from` $i + 1$ `to` $i + w$ `do` $d_i \leftarrow$ "*unknown*"
    $i \leftarrow i + w + 1$
    $c \leftarrow w$
    $state \leftarrow 1$
  `else if` $v_i = \mathcal{SN}$ `then`
    `if` $c \geq w + 1$ `and` $state = 1$ `then` $d_i \leftarrow 0$
    `else if` $c = w$ `and` $state = 1$ `then` $d_i \leftarrow$ "*unknown*"
    `else` $d_i \leftarrow 1$
    `if` $i + w + 1 > t$
      `for` j `from` $i + 1$ `to` $t - 2$ `do` $d_i \leftarrow$ "*unknown*"
      $d_{t-1} \leftarrow 1$
      $i \leftarrow t$
    `else`
      `for` j `from` $i + 1$ `to` $i + w$ `do` $d_i \leftarrow$ "*unknown*"
      $d_{i+w+1} \leftarrow 0$
      $i \leftarrow i + w + 2$
    `end if`
    $c \leftarrow w$
    $state \leftarrow 0$
  `else if` $v_i = \mathcal{N}$ `then`
    `if` $state = 1$ `then` $d_i \leftarrow 1$
    `else if` $state = 0$ `then` $d_i \leftarrow 0$
    $c \leftarrow c + 1$
    $i \leftarrow i + 1$
  `end if`
`end while`
`Return` $d = (d_{t-1} \cdots d_0)_2$

4.5 Experimental Results

We carried out experiments by a simulation on the attacks described in the previous sections as follows.

1. randomly generate 10,000 exponents d, which have 160-bit, 512-bit or 1024-bit.
2. implement algorithms for exponent recodings described in Algorithms 3, 4 and 5 in S/W written in C-language.
3. generate $\mathcal{SN}$-sequences using this S/W.
4. using the $\mathcal{SN}$-sequences, we guess the secret exponent d by Algorithm 6, 7 and 8.
5. count the successfully recovered bits (= number of recovered bits / bitlength of exponent d)

The results are given in Table 2. The experiments were carried out with 160-bit, 512-bit and 1024-bit exponents. Almost the same percentages were obtained in each case.

The window width w were examined from 2 through 5. In the fractional window expansion the parameter m were examined from 1 through the upper bound, i.e. $2^w - 3$. The intermediate ($2 \leq m \leq 2^w - 4$) are omitted because of the space limitation. The successfully recovered bits decrease in larger w, because as we have already mentioned, the bits of the middle in the window can not be guessed uniquely. In the fractional window expansion the successfully recovered bits increase in larger m. Examples of the three proposed attacks with small exponents are illustrated in Appendix B.

Remark 2. No guessing errors occur in the attacks. Only "*unknown*" bits can be un-recovered bits.

Table 2. Successfully recovered bits (%) (= number of recovered bits / bitlength of exponent d): Experimental results

w	m	width-w NAF	unsigned fract.	signed fract.
2	—	100	—	—
	1	—	50.5	50.3
3	—	75.1	—	—
	1	—	38.9	36.3
	5	—	40.0	46.0
4	—	60.2	—	—
	1	—	31.3	28.3
	13	—	33.7	41.4
5	—	50.3	—	—
	1	—	26.2	23.5
	29	—	29.1	37.1

5 Concluding Remarks

We have shown that unless the exponent recoding is carefully implemented, RSA and elliptic curve based cryptosystems are vulnerable to power analysis. We have introduced new side channel attacks on exponent recoding.

While the effects of a single transistor switching would be normally be impossible to identify from direct observations of a device's power consumption [4], the statistical operations are able to reliably identify extraordinarily small differences in power consumption.

Okeya and Takagi proposed efficient counter measures for side channel attacks [6, 7]. The exponent recodings given in [6, 7] are based on width-w NAF or fractional window. Therefore, it may be possible to construct attacks on the recodings.

References

1. J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," CHES'99, LNCS 1717, pp.292–302, Springer-Verlag, 1999.
2. P.C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," Advances in Cryptology – CRYPTO'96, LNCS 1109, pp.104–113, Springer-Verlag, 1996.
3. P.C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," Advances in Cryptology – CRYPTO'99, LNCS 1666, pp.388–397, Springer-Verlag, 1999.
4. P.C. Kocher, J. Jaffe, and B. Jun, "Introduction to differential power analysis and related attacks," Cryptography Research, http://www.cryptography.com
5. B. Möller, "Improved techniques for fast exponentiation," ICISC 2002, LNCS 2587, pp.298–312, Springer-Verlag, 2002.
6. K. Okeya, T. Takagi, "The width-w NAF method provides small memory and fast elliptic scalar multiplications secure against side channel attacks," CT-RSA 2003, LNCS 2612, pp.328–342, Springer-Verlag, 2003.
7. K. Okeya, T. Takagi, "A more flexible countermeasure against side channel attacks using window method," CHES 2003, LNCS 2779, pp.397–410, Springer-Verlag, 2003.
8. G.W. Reitwiesner, "Binary arithmetic," Advances in Computers, vol.1, pp.231–308, 1960.
9. J.A. Solinas, "Efficient arithmetic on Koblitz curve," Designs, Codes and Cryptography, vol.87, pp.195–249. Kluwer Academic Publishers, 2000.

A Relation between Unknown Bits

In this appendix we show relations between unknown bits. If several unknown bits occur in succession, some unknown bits can be "0" or "1" with high probability. Table 3 shows the probability that unknown bits can be "0" or "1" in the case of the unsigned fractional window recoding with the parameter $w = 3$ and $m = 1$.

Table 3. Observed $\mathcal{SN}$-values and unknown bits: unsigned fractional window recoding with $w = 3$, $m = 1$

observed $\mathcal{SN}$-value	$\mathcal{SN}$	$\mathcal{SSN}$
candidates of the secret exponent in the window	0 0 0 1	1 0 1 1
	0 0 1 1	1 1 0 1
	0 1 0 1	1 1 1 1
	0 1 1 1	
	1 0 0 1	
recovered bits (“x” denotes the unknown)	x x x 1	1 x x 1
probability that x=0	$\frac{4}{5}$ $\frac{3}{5}$ $\frac{3}{5}$	$\frac{1}{3}$ $\frac{1}{3}$

B Examples of the Attacks

In this appendix we illustrate small examples of the three attacks. In examples below, given randomly generated 32-bit exponents, recoded representations and expected $\mathcal{SN}$-sequences are described. The attacker can recover the exponents from the observed $\mathcal{SN}$-sequences as shown below. The symbol “x” in the recovered bits denotes the “*unknown*” bit.

width-3 NAF

exponent:	1 1 1 0 0 0 0 1 1 1 1 0 1 0 1 0 1 1 1 1 0 1 1 0 1 0 1 1 0 0 1 1
recoded:	1 0 0 -1 0 0 0 1 0 0 0 0 0 -3 0 0 3 0 0 0 0 -1 0 0 0 -3 0 0 3 0 0 0 3
$\mathcal{SN}$-sequence:	1 0 0 0 1 0 0 2 0 0 0 2 0 0 0 0 1 0 0 2 0 0 0 0 0 1 0 0 0 2 0 0 1
recovered:	1 x 1 0 0 x 0 1 x 1 1 0 x 0 1 0 1 x 1 1 x 1 1 0 1 0 x 1 0 0 x 1

unsigned fractional window with w = 3, m = 1

exponent:	1 0 1 1 1 0 0 0 1 1 1 1 0 0 0 0 1 0 1 1 1 0 1 0 1 0 0 0 0 1 1 1
recoded:	1 0 0 0 7 0 0 0 1 0 0 7 0 0 0 0 1 0 0 0 7 0 0 0 5 0 0 0 0 0 0 7
$\mathcal{SN}$-sequence:	1 0 0 0 1 0 0 0 1 0 0 2 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1
recovered:	1 x x x 1 x x x 1 x x 1 0 x x x 1 x x x 1 x x x 1 0 0 0 x x x 1

signed fractional window with w = 3, m = 1

exponent:	1 0 0 1 1 0 1 1 0 0 1 1 1 1 1 0 1 0 0 1 1 0 1 0 0 0 1 0 0 1 0 0
recoded:	1 0 0 0 0 7 0 0 0 -3 0 0 0 0 -1 0 0 0 5 0 0 0 -3 0 0 0 0 0 0 9 0 0
$\mathcal{SN}$-sequence:	1 0 0 0 0 1 0 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0 0 0 0 0 0 1 0 0
recovered:	1 0 x x x x x x x 0 1 x x x x x x x x x x x 1 0 0 0 x x x 1 0 0