On the BRIP Algorithms Security for RSA

Frédéric Amiel[1] and Benoit Feix[2]

[1] TEXAS INSTRUMENTS,
Villeneuve Loubet, France
f-amiel@ti.com
[2] INSIDE CONTACTLESS
41 Parc Club du Golf
13856 Aix-en-Provence, Cedex 3, France
bfeix@insidefr.com

Abstract. Power Analysis has been intensively studied since the first publications in 1996 and many related attacks on naive implementations have been proposed. Nowadays algorithms in tamper resistant devices are protected by different countermeasures most often based on data randomization such as the BRIP algorithm on ECC and its RSA derivative. However not all of them are really secure or in the best case proven to be secure. In 2005, Yen, Lien, Moon and Ha introduced theoretical power attacks on some classical and BRIP exponentiation implementations, characterized by the use of a chosen input message value ± 1. The first part of our article presents an optimized implementation for BRIP that takes advantage of the Montgomery modular arithmetic to speed up the mask inversion operation. An extension of the Yen *et al.* attack, based on collision detection through power analysis, is also presented. Based on this analysis we give security advice on this countermeasure implementation and determine the minimal random length to reach an appropriate level of security.

Keywords: Power analysis, collision attacks, RSA, BRIP, modular multiplication and exponentiation.

1 Introduction

Asymmetric cryptography was introduced by Diffie and Hellman [DH76] in 1976. The most widely used algorithms today are: RSA [RSA78] invented in 1978 by Rivest, Shamir, and Adleman, and elliptic curve cryptosystems (ECC) independently introduced by Koblitz [Kob87] and Miller [Mil86].

Compared with symmetric cryptography, public key algorithms are computationally very intensive. In practice long integer arithmetic is most often handled by specific coprocessors designed for efficient computation in $GF(p)$. This is the case for embedded solutions with strict power consumption and/or timing constraints.

Initially smart cards were considered inherently tamper resistant because any private data was embedded and thus physically inaccessible to an unauthorized user. However in 1996 timing attacks were publicly introduced by Kocher in [Koc96]. Two years later he also introduced power analysis attacks with Jaffe and Jun [KJJ99]. Side Channel Analysis (SCA) is a group of techniques including simple power analysis (SPA) and differential power analysis (DPA). SCA threatens any naive cryptographic algorithm implementation. Since these first articles were published, power analysis has been widely investigated, some publications have focused on countermeasures and their drawbacks [FV03,MPO05,YLMH05] whereas others have focused on improving the efficiency of the attacks [ABDM00,BK03,BCO04].

One such countermeasure is the Binary with Random Initial Point (BRIP) algorithm(s) by Mamiya, Miyaji, Morimoto [MMM04] and Itoh, Izu and Takenada [IIT04], later improved in [IIT06]. BRIP countermeasure was originally designed for ECC and later extended to RSA cryptosystems. Its RSA variant corresponds to the countermeasure also proposed in [KHK+04] and is particularly interesting in terms of implementation as neither the bit size of the prime characteristic of the field is increased nor is the knowledge of the public exponent value needed.

Our study focuses on the exponentiation and for readibility purposes, BRIP acronym will refer here to the BRIP RSA derivative of the countermeasure.

The paper is organized as follows: section 2 gives an overview of embedded asymmetric algorithms and their related side-channel potential vulnerabilities. Section 3 describes the BRIP algorithms with the current identified vulnerabilities and our implementation improvements. New attacks on these algorithms and recommendations will be presented in Section 4. We conclude our research in Section 5.

2 Power Analysis Background

Since the initial publication in [KJJ99] on Simple Power Analysis (SPA), many improvements have been made on this subject. Electronic devices, such as smart cards or other security products, are designed with thousands of logical gates switching differently depending on the executed operations and the data manipulated. The device power consumption of the chip depends on these operations which can be easily monitored and analysed on an oscilloscope. For instance, if the square operation has a different pattern on the power curve than the one for multiplication, it is

obvious that the attacker can easily recover the secret exponent in a naive RSA implementation. Many other differences visible in the power curve can lead to the same kind of leakage on the private key(s). Developers must take into account all the potential vulnerabilities.

One of the first Collision Power Analysis attack is the Doubling Attack by Fouque and Valette [FV03]. It was applied on a scalar multiplication operation in ECC. They also explained how it could be extended to RSA implementations.

Differential Power Analysis (DPA) and its improvements represent the other main class of side channel attacks. The most well known is the Correlation Power Analysis (CPA) by Brier, Clavier and Olivier [BCO04]. It was later applied by Amiel, Feix and Villegas [AFV07] on most asymmetric algorithms. The first DPA attack on RSA was done in 1999 by Messerges, Dabbish and Sloan [MDS99]. Enhanced DPA attacks, such as the Zero Value Point Attack published by Goubin in [Gou03], have also been done on elliptic curve implementations. Goubin's attack threatens Coron's randomization of the projective coordinates [Cor99] in the elliptic curve scalar multiplication. The first combination between fault injection and power analysis has also been applied to XTR in [CG04]. The BRIP algorithms have then been proposed to counteract the Zero Value Point Attack. Moreover, BRIP can also be applied in $GF(p)$ for cryptosystems based on the factorization and discrete logarithm problems, like RSA.

However Yen, Lien, Moon and Ha [YLMH05] presented a power collision attack on the BRIP countermeasure for RSA by using ± 1 values for input messages, and on the Square and Multiply Always algorithm by using $\pm m \mod n$ messages as an input for RSA.

3 Modular Exponentiations for BRIP Algorithms

Firstly we present the BRIP algorithm variant for RSA, we also introduce some improvements and optimizations for this countermeasure when combined with Montgomery modular multiplication.

3.1 Modular Multiplication and Exponentiation

We summarize the principles used later in this paper: modular multiplication and exponentiation, in particular the ones designed by Montgomery, which are particularly suitable for embedded implementations and the RSA public key cryptosystem.

3.2 Modular Multiplication

To compute modular multiplications $x \times y \mod n$ on long integers x, y and n Montgomery proposed the following efficient algorithm in [Mon85].

Montgomery modular multiplication
Given a modulus n and two integers x and y, of size v in base b, with $\gcd(n, b) = 1$ and $r = b^{\lceil log_b(n) \rceil}$, MontMul algorithm computes:

$$\mathsf{MontMul}(x,\ y,\ n) = x \times y \times r^{-1} \mod n$$

We suggest the reader to refer to Appendix A.1 and papers [Mon85] and [KAK96] if more detail on this operation is wished.

We can then use this operation to process efficiently Montgomery modular exponentiation (MontExp) as detailed in [Dhe98]. Compared to a classical Square and Multiply algorithm it consists of multiplying the message operand and the accumulator by $r \mod n$ before the exponentiation loop. In this case any intermediate result during the exponentiation is equal to $m^k.r \mod n$. At the end the r value is removed by doing a modular montgomery multiplication by 1. Refer to Appendix A.2 for the detailed algorithm.

Classical BRIP Implementation for RSA
Alg. 3.1 describes the classical BRIP implementation introduced in [KHK+04] with a random v generated from a h-bit random seed u. It means $v = f(u)$. Value v must be as long as the modulus to prevent the implementation of a chosen message SPA. The security of the random value v is the same as the seed random u, this implies there are only 2^h possible values. For instance $v = (u|u \ldots |u)$.

The major drawback is the time needed to compute the modulo inverse $v^{-1} \mod n$. The next implementation avoids this if Montgomery modular multiplication hardware is available.

Second BRIP Implementation with MontMul
The inversion of random $v \mod n$ is a penalty for the BRIP algorithm performance. A solution consists in using the following property of the Montgomery multiplication: $\mathsf{MontMul}(1,\ 1,\ n) = r^{-1} \mod n$. This gives an efficient way to compute an exponentiation with both a fixed base value (r) and a negative exponent. The idea was introduced by one of the authors in [CF05] and can also be applied to BRIP.

Algorithm 3.1 BRIP Exponentiation from left to right

INPUT: integers m and n such that $m < n$, k-bit exponent $d = (d_{k-1}d_{k-2}\dots d_1 d_0)_2$
OUTPUT: BRIP_Exp(m,d,n)= $m^d \mod n$

Step 1. If $m = 1$ **Return**(1)
Step 2. If $m = n - 1$ **Return**($(-1)^{d_0} \mod n$)
Step 3. Choose a random value v and compute $v^{-1} \mod n$
Step 4. $a = v$, $m_0 = v^{-1} \mod n$, $m_1 = v^{-1}.m \mod n$
Step 5. for i from $k-1$ to 0 **do**
  $a = a \times a \mod n$
  $a = a \times m_{d_i} \mod n$
Step 6. $a = a \times m_0$
Step 7. Return(a)

The $v^{-1} \mod n$ computation can be replaced by $r^{-v} \mod n$ implemented as an exponentiation with a relatively short exponent (typically $|v| << |d|$). This trick saves a lot of time compared to a modular inverse calculation.

Thus we obtained the Algorithm Alg. 3.2.

Step 5. of Alg. 3.2 replaces the costly inversion operation of random v in Alg. 3.1. However both previous algorithms Alg. 3.1 and Alg. 3.2 have a complexity of 2 which is the same as the well known Square and Multiply Always algorithm. Improvements can however be envisaged by using k-ary and sliding window methods [ÇKK]. In [MMM04] the authors also presented optimized versions of BRIP, one version is using the k-ary method.

Algorithm Alg. 3.3 we present here, corresponds to WBRIP for RSA with MontMul. It corresponds to a 2-ary exponentiation with the BRIP countermeasure and the improvement we proposed with the Montgomery multiplication. There is no costly inversion operation and the algorithm complexity is 1.5, but more memory space is required for the precomputations storage compared to both the previous versions.

In this case the mask value for computation is no longer r^{-v} but r^{-3v} as we manipulate scalar bits by 2-bit windows.

Algorithm 3.2 MontExp-BRIP from left to right

INPUT: integers m and n such that $m < n$, k-bit exponent $d = (d_{k-1}d_{k-2}\dots d_1d_0)_2$
OUTPUT: MontExp(m,d,n)$= m^d \mod n$

Step 1. If $m = 1$ **Return**(1)
Step 2. If $m = n - 1$ **Return**$((-1)^{d_0} \mod n)$
Step 3. Choose a h-bit random value v
Step 4. Compute $V = r^v \mod n =$ MontExp(r,v,n)
Step 5. Compute $V_1=$ MontMul(1, 1, n) and $U = r^{-v} \mod n =$ MontExp(V_1,v,n)
Step 6. $a = V.r \mod n$, $m_1 = m.U.r \mod n$, $m_0 = U.r \mod n$
Step 7. for i from $k-1$ to 0 **do**
  $a =$ MontMul(a, a, n)
  $a =$ MontMul(a, m_{d_i}, n)
Step 8. $a =$ MontMul(a, m_0, n)
Step 9. $a =$ MontMul(a, 1, n)
Step 10. Return(a)

Algorithm 3.3 MontExp-WBRIP from left to right

INPUT: integers m and n such that $m < n$, k-bit exponent $d = (d_{k-1}d_{k-2}\dots d_1d_0)_2$
OUTPUT: MontExp(m,d,n)$= m^d \mod n$

Step 1. If $m = 1$ **Return**(1)
Step 2. If $m = n - 1$ **Return**$((-1)^{d_0} \mod n)$
Step 3. Choose a h-bit random value v
Step 4. Compute $V = r^v \mod n =$ MontExp(r,v,n)
Step 5. Compute $U = r^{-3v} \mod n$
Step 6. Compute $a = V.r \mod n$, $m_0 = U.r \mod n$, $m_1 = m.U.r \mod n$
Step 7. Compute $m_2 = m^2.U.r \mod n$, $m_3 = m^3.U.r \mod n$
Step 8. for i from $k-1$ to 0 by 2 **do**
  $a =$ MontMul(a, a, n)
  $a =$ MontMul(a, a, n)
  $a =$ MontMul(a, $m_{(2.d_i+d_{i-1})}$, n)
Step 9. $a =$ MontMul(a, m_0, n)
Step 10. $a =$ MontMul(a, 1, n)
Step 11. Return(a)

Depending on the memory contraints, the size of the window can be modified. For a k-bit window the algorithm complexity becomes equal to $1 + 1/k$.

4 Power Analysis Attacks on BRIP like Algorithms

We present here an improvement to the power collision attack on RSA implementations based on the previous BRIP implementations. Fouque and Valette first [FV03] introduced power collision attacks on some of the classical elliptic curve scalar multiplication algorithms, they also explained how to extend the technique to modular exponentiations. Later Yen *et al.* [YLMH05] introduced collision power attacks based on chosen message values $\pm 1 \mod n$ that allows the secret exponent value d to be recovered from a single curve. Developpers must avoid BRIP computation when the input message equals $n - 1$ and simply return value 1 or $n - 1$ depending on the parity of the secret exponent.
In their article, some other variants of the attacks are presented, especially on the Square and Multiply Always algorithm by using $\pm m \mod n$ messages as input, but none of them compromise a full implementation of BRIP.

4.1 Collision Power Analysis on BRIP and MontExp-BRIP

Modular multiplication on a chip requires relatively long processing time and relatively high power consumption compared with symmetric algorithms, where for example, processing can be carried out in a few clock cycles in hardware implementations of AES.

In figure 1 we analyse power traces of the MonMult operation executed on a tamper resistant device such as a smart card.
We choose two different random messages m_1 and m_2 and for each message we execute three multiplications $\mathsf{MontMul}(m_1, m_1, n)$ and $\mathsf{MontMul}(m_2, m_2, n)$. We then collect the three power curves $C_{1,1}$, $C_{1,2}$ and $C_{1,3}$ of the multiplication with m_1 and three curves $C_{2,1}$, $C_{2,2}$ and $C_{2,3}$ of the multiplication with m_2.

We notice, cf. figure 1, that on the selected chip, the multiplication is a very power consuming operation. This is due to the large number of gates which are switching together in the asymmetric coprocessor logic.
From this curves we observe that power collisions occur for similar data manipulated by the chip. $C_{1,1}$, $C_{1,2}$ and $C_{1,3}$ are similar and have exactly

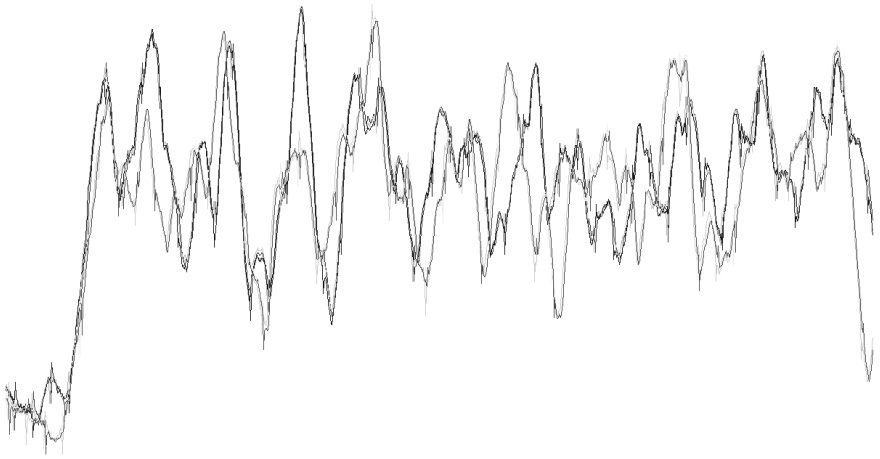

Fig. 1. Power consumption of a single modular multiplication for curves $C_{1,1}$, $C_{1,2}$, $C_{1,3}$ and $C_{2,1}$, $C_{2,2}$, $C_{2,3}$

.

the same power traces, as do $C_{2,1}$, $C_{2,2}$ and $C_{2,3}$. It means that $C_{i,j}$ collides with $C_{k,l}$ when $i = k$ while $C_{i,j}$ is different from $C_{k,l}$ when $i \neq k$.

Due to the important number of clock cycles in a modular multiplication in the power curve, we can assume that different input data will have different power trace patterns. This means we can distinguish collisions with a high probability. The tests we made on the selected chip confirm our assumption.

We analyse if we can exploit eventual collisions on the classical BRIP algorithm 3.1 and the MontExp-BRIP 3.2 for an h-bit random value v. For both algorithms the analysis will be identical.
Depending on the method for generating the random value v, it is obvious that in some cases, collisions on its values could happen when generating it. This will depend on the quality of the random and on its length. However for performance reasons BRIP and especially MontExp-BRIP can not use big values h.

We observe that if a colliding value for the random mask v appears, then by choosing as algorithm input message m for the first execution and $-m \mod n$ for the second one, we can have multiplications with similar operands in both executions. This could lead to distinguishable power collisions between the power curves of both executions. We then try to

exploit these collisions to recover the secret exponent d.

Firstly we execute BRIP a number of times with input message m. For any execution a new random value v_1 is generated by the chip. Then we repeat this operation with the input message $-m \mod n$, for each execution a new random value v_2 is generated.

Let $d = d''.2^{i+1} + d_i.2^i + d'$ where, d_i is the current bit handled by the exponentiation loop, d'' the left part of d previously processed (left-to-right exponentiation) and d' the right remaining part of the exponent. In figure 2 we can observe for a step i of the BRIP execution what operands are manipulated by the chip for modular multiplications. In the first table we see these operand values during a real multiplication ($d_i = 1$), and in the second table when $d_i = 0$.

Message	Square	Message Multiplication ($d_i = 1$)	Square
m	$\left[m^{d''}.v_1\right]^2$	$\left[(m^{2.d''}).v_1^2\right] \times \left[m.v_1^{-1}\right]$	$\left[(m^{2.d''+1}).v_1\right]^2$
$-m$	$\left[(-m)^{d''}.v_2\right]^2$	$\left[(m^{2.d''}).v_2^2)\right] \times \left[-m.v_2^{-1}\right]$	$\left[((-m)^{2.d''+1}).v_2\right]^2$
Collision if $v_1 = v_2$	-	No	No
Message	Square	Fake Multiplication ($d_i = 0$)	Square
m	$\left[m^{d''}.v_1\right]^2$	$\left[(m^{2.d''}).v_1^2\right] \times \left[v_1^{-1}\right]$	$\left[(m^{2.d''}).v_1\right]^2$
$-m$	$\left[(-m)^{d''}.v_2\right]^2$	$\left[(m^{2.d''}).v_2^2)\right] \times \left[v_2^{-1}\right]$	$\left[(m^{2.d''}).v_2\right]^2$
Collision if $v_1 = v_2$	-	Yes	Yes

Fig. 2. BRIP execution for $d_i = 1$ and $d_i = 0$.

We detect collisions on any Fake Multiplication (multiplication by r^{-v}) operation when a collision happens on v and $v_1 = v_2$. Thus collision detection through power analysis is a real threat.
We can then observe on power traces when a collision occurs. We store in memory the power curves C_i of the BRIP execution with message m and C'_i with message $-m \mod n$. Then we search for two curves C_i and C'_j where power collisions appear between the two curves. Then by subtracting C'_j to C_i we can recover the secret exponent d.

Algorithm 4.4 BRIP Collision Attack

INPUT: $s = \mathsf{RSA\text{-}BRIP}(m, d)$, $s' = \mathsf{RSA\text{-}BRIP}(-m, d)$
OUTPUT: Secret exponent d

Step 1. Choose a random value m in $[2, n-2]$.
Step 2. Collect k traces $(C_0, ..., C_{k-1})$ of BRIP execution with m as input message.
Step 3. Collect k traces $(C'_0, ..., C'_{k-1})$ of BRIP execution with $-m$ as input message.
Step 4. Find traces C_i and C'_j such as both traces are colliding on each BRIP Fake Multiply.
Step 5. Compute $S = |C_i - C'_j|$.
Step 6. Each non zero difference on S identify a true multiplication, i.e. $d_i = 1$

The probability of finding at least one colliding couple from both sets of k traces is approximated in [MOV96] (Fact 2.27) by:

$$p_{collision} \simeq 1 - e^{-((k^2)/|h|)}$$

where $|h|$ denotes the number of possible value for v so 2^h.

Figure 3 gives the probability of collision for a 32-bit random v relative to the number of encryptions done.

h	k	collision
32	78000	0.507
32	$2^{17} \approx 131072$	0.864
32	161000	0.951
32	200000	0.990
32	$2^{18} \approx 262144$	0.999

Fig. 3. Probability of collision for $h = 32$

Thus in practice with 2^{32} possible values for v (32-bit random), two sets of $k = 78000$ curves are sufficient to have a probability of $\frac{1}{2}$ for obtaining a collision, where two sets of $k = 200000$ curves will lead to a collision and then a successful attack in 99 percents of cases.($p_{collision} \simeq 0.99$).

This collision attack is a serious threat and also appears on MontExp-BRIP, cf. Alg. 3.2. The random value v is not used in the same way in the algorithm but the analysis and the results of collision are similar.

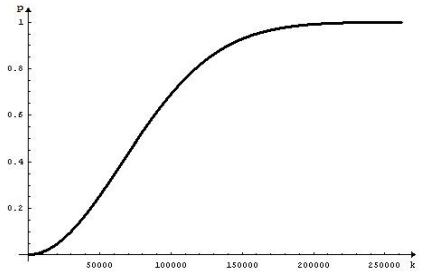

Fig. 4. Graph of probability of collision for a 32-bit value v

h	k	collision probability
16	$2^9 = 512$	0.864
16	$2^{10} = 1024$	0.999
64	5.1×10^9	0.505
64	2^{33}	0.864
64	2^{34}	0.999
96	3.3×10^{14}	0.497
96	2^{48}	0.864
96	2^{49}	0.999

Fig. 5. Probability of collision for other h values

We suggest using at least 96-bit random value v ($h = 96$ cf. figure 5) to prevent MontExp-BRIP and BRIP against such collision attacks. However it is obvious that such random lengths will reduce the performance of these implementations and then the reason for using them.

4.2 Collision Attack on MontExp-WBRIP

We analyse the impact of power collisions in the MontExp-WBRIP implementation. In the figure 6 we replace the notation MontMul by MM.
Let $d = d''.2^{i+1} + d_i.2^i + d_{i-1}.2^{i-1} + d'$ where d_i and d_{i-1} are the two bits of the 2-bit window handled for each exponentiation loop, d'' the left part of d previously processed and d' the right part being process in next steps.

Message	Square	Square	$M_{(2d_i+d_{i-1}=0)}$ MontMul
m	$MM(m^{d''}.r^{v_1}, m^{d''}.r^{v_1})$	$MM(m^{2.d''}.r^{v_1}, m^{2.d''}.r^{v_1})$	$MM(m^{4.d''}.r^{4v_1}, r^{-3v_1})$
$-m$	$MM((-m)^{d''}.r^{v_2}, (-m)^{d''}.r^{v_2})$	$MM(m^{2.d''}.r^{v_2}, m^{2.d''}.r^{v_2})$	$MM(m^{4.d''}.r^{4v_2}, r^{-3v_2})$
If $v_1 = v_2$	-	Yes	Yes
Message	Square	Square	$M_{(2d_i+d_{i-1}=1)}$ MontMul
m	$MM(m^{d''}.r^{v_1}, m^{d''}.r^{v_1})$	$MM(m^{2.d''}.r^{v_1}, m^{2.d''}.r^{v_1})$	$MM(m^{4.d''}.r^{4v_1}, m.r^{-3v_1})$
$-m$	$MM((-m)^{d''}.r^{v_2}, (-m)^{d''}.r^{v_2})$	$MM(m^{2.d''}.r^{v_2}, m^{2.d''}.r^{v_2})$	$MM(m^{4.d''}.r^{4v_2}, (-m).r^{-3v_2})$
If $v_1 = v_2$	-	Yes	No
Message	Square	Square	$M_{(2d_i+d_{i-1}=2)}$ MontMul
m	$MM(m^{d''}.r^{v_1}, m^{d''}.r^{v_1})$	$MM(m^{2.d''}.r^{v_1}, m^{2.d''}.r^{v_1})$	$MM(m^{4.d''}.r^{4v_1}, m^2.r^{-3v_1})$
$-m$	$MM((-m)^{d''}.r^{v_2}, (-m)^{d''}.r^{v_2})$	$MM(m^{2.d''}.r^{v_2}, m^{2.d''}.r^{v_2})$	$MM(m^{4.d''}.r^{4v_2}, m^2.r^{-3v_2})$
If $v_1 = v_2$	-	Yes	Yes
Message	Square	Square	$M_{(2d_i+d_{i-1}=3)}$ MontMul
m	$MM(m^{d''}.r^{v_1}, m^{d''}.r^{v_1})$	$MM(m^{2.d''}.r^{v_1}, m^{2.d''}.r^{v_1})$	$MM(m^{4.d''}.r^{4v_1}, m^3.r^{-3v_1})$
$-m$	$MM((-m)^{d''}.r^{v_2}, (-m)^{d''}.r^{v_2})$	$MM(m^{2.d''}.r^{v_2}, m^{2.d''}.r^{v_2})$	$MM(m^{4.d''}.r^{4v_2}, (-m)^3.r^{-3v_2})$
If $v_1 = v_2$	-	Yes	No

Fig. 6. WBRIP execution for possible $2d_i + d_{i-1}$ values.

We can observe in figure 6 the different possible collisions on curves when random values v and v_1 collide. But in WBRIP it does not give us as much information as in the previous algorithms. The collisions will indicate that the 2-bit window value is either 00 or 10 so $d_{i-1} = 0$ and non collisions will indicate the 2-bit window value is either 01 or 11 so $d_{i-1} = 1$. Then we recover here half of the bits of the secret exponent d.
Indeed, we can extend this result to any k-ary implementation of BRIP

exponentiation as the Collision Attack gives the information on the parity of i in $m^i.r^{-2^{k-1}.v}$ operand used during the Multiplication operation. Therefore the number of bits recovered by a collision is equal to : $|d|/k$, namely $|d|$ for $k = 1$, $|d|/2$ for $k = 2$ and so on.

Thus we also suggest using at least 96-bit random value v to prevent MontExp-WBRIP from such collision attacks.

4.3 Collision Attack of BRIP Implementations for RSA CRT

These collision attacks can be similarly applied to RSA CRT exponentiations protected with MontExp-BRIP, MontExp-WBRIP or BRIP algorithms.
Indeed when $n = p.q$, p and q being prime numbers of equivalent lengths, choosing $\pm m \mod n$ messages leads to the manipulation of $\pm m \mod p$ and $\pm m \mod q$ in the CRT exponentiations once the reductions by p and q have been done.
Then the previous collision analysis applies identically to RSA CRT using any of the previous BRIP algorithms.

4.4 Implementing MontExp-BRIP countermeasure

We notice that both exponentiations: $r^v \mod n = \mathsf{MontExp}(r, v, n)$ and $r^{-v} \mod n = \mathsf{MontExp}(V_1, v, n)$ need to be carefully implemented against the classical power analysis techniques. Indeed it is obvious that if v is recovered, each operand value in algorithm 3.2 becomes deterministic and then statistical attacks can be envisaged to recover the secret exponent.

The most important threat is Timing Attack (TA) for which Double and Add Always or Side Channel Atomicity [CCJ04] are both convenient countermeasures.
Anyway, protection against TA may not be sufficient as the operation $r^{-v} = \mathsf{MontExp}(V_1, v, n)$ can be sensitive to SPA. This is due to the particular Hamming Weight of one of the Multiply operands, explicitely $\overline{m}$ in $a = \mathsf{MonMul}(a, \overline{m}, n)$ with $\overline{m} = f_n(m) = r^{-1} * r = 1$.

Analysing the implementation details of MontMul gives some clues to explaining the leakage. During the computation of $\mathsf{MontMul}(a, 1, n)$, we notice than most of multiplications involved in Step 2 of algorithm A.5 are composed of integer multiplications by 0 or 1 which have a straightforward impact on the power consumption by significantly lowering it

compared to the multiplication of two random operands. It can then be feasible to deduce directly from the power curve the nature of each operation and recover v value for each curve.
A simple tweak to counteract such an SPA attack is to compute $(-r)^{-v}$ rather than r^{-v}, $n-1$ will then replace 1 as input operand of MontMult during Multiply operations. This may still not be sufficient to protect against advanced SPA or Template Analysis attacks as intrinsically r^{-v} or r^{v} exponentiations are not randomized.
Applying additional randomization techniques on r^{-v} and r^{v} exponentiations could be envisaged to protect against such threats but will reduce efficiency and at the same time reason for MontExp-BRIP countermeasures.

5 Conclusion

Several possible implementations of BRIP algorithms have been presented in this paper. We used the efficiency of Montgomery modular arithmetic to provide an efficient message masking technique. We showed with these implementations detecting collisions through power analysis, and especially during modular multiplications, is a realistic threat. We also explained that random length must be chosen very carefully to prevent these implementations from the collision attacks we have described. Thus using 32-bit or even 64-bit random values should be avoided here. In the case where ISO random padding is used, it naturally prevents our implementation from this collision attack and allows a shorter random value (32 bits) to be used, but it is not always the case.

We also stress to the reader that random value manipulation must be strongly protected in the MontExp-BRIP algorithm against the different side channel techniques in order to prevent the random recovery by power leakage. Such random recovery could then lead to other classical power analysis on the secret.

Acknowledgements

The authors would like to thank Mathieu Ciet for its comments, and Sean Commercial for his help for the final version.

References

[ABDM00] M-L. Akkar, R. Bevan, P. Dischamp, and D. Moyart. Power Analysis, What Is Now Possible.... In T. Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 489–502, 2000.

[AFV07] Frederic Amiel, Benoit Feix, and Karine Villegas. Power Analysis for Secret Recovering and Reverse Engineering of Public Key Algorithms. In *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 110–125. Springer, 2007.

[BCO04] E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In M. Joye and J-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[BK03] R. Bevan and E. Knudsen. Ways to Enhance Differential Power Analysis. In P. J. Lee and C. H. Lim, editors, *ICISC*, volume 2587 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2003.

[CCJ04] B. Chevallier-Mames, M. Ciet, and M. Joye. Low-cost solutions for preventing simple side-channel analysis: side-channel atomicity. *IEEE Transactions on Computers*, 53(6):760–768, 2004.

[CF05] M. Ciet and B. Feix. Cryptographic method comprising a modular exponentiation secured against hidden-channel attacks, crypto processor for implementing the method and associated chip card. *Gemplus Patent WO2007074149*, 2005.

[CG04] M. Ciet and C. Giraud. Transient fault induction attacks on XTR. In J. Lopez, S. Qing, and E. Okamoto, editors, *Information and Communications Security - ICICS 2004*, volume 3269 of *Lecture Notes in Computer Science*, pages 440–451. Springer, 2004.

[ÇKK] journal = Computers and Mathematics with Applications year = 1995 volume = 30 pages = 17–24 number = 10 Ç. K. Koç, title = Analysis of Sliding Window Techniques for Exponentiation.

[Cor99] J-S Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 1999*, volume 1717 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 1999.

[DH76] W. Diffie and M. E. Hellman. New Directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[Dhe98] J-F. Dhem. *Design of an efficient public-key cryptographic library for RISC-based smart cards.* PhD thesis, Université catholique de Louvain, Louvain, 1998.

[FV03] P-A. Fouque and F. Valette. The Doubling Attack - *why upwards is better than downwards.* In C. D. Walter, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 269–280. Springer, 2003.

[Gou03] L. Goubin. A refined power-analysis attack on elliptic curve cryptosystems. In Yvo Desmedt, editor, *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 199–210. Springer, 2003.

[IIT04] K. Itoh, T. Izu, and M. Takenaka. Efficient Countermeasures against Power Analysis for Elliptic Curve Cryptosystems. In J-J. Quisquater, P. Paradinas, Y. Deswarte, and A. A. El Kalam, editors, *CARDIS*, pages 99–114. Kluwer, 2004.

[IIT06] K. Itoh, T. Izu, and M. Takenaka. Improving the Randomized Initial Point Countermeasure against DPA. In J. Zhou, M. Yung, and F. Bao, editors, *Applied Cryptography and Network Security, 4th International Conference, ACNS 2006*, volume 3989 of *Lecture Notes in Computer Science*, pages 459–469, 2006.

[KAK96] Ç. K. Koç, T. Acar, and B-S. Kaliski. Analysing and comparing Montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, 1996.

[KHK+04] C. Kim, J. Ha, S-H. Kim, S. Kim, S-M. Yen, and S-J. Moon. A Secure and Practical CRT-Based RSA to Resist Side Channel Attacks. In A. Laganà, M. L. Gavrilova, V. Kumar, Y. Mun, C. J. K. Tan, and O. Gervasi, editors, *Computational Science and Its Applications - ICCSA 2004*, volume 3043 of *Lecture Notes in Computer Science*, pages 150–158. Springer, 2004.

[KJJ99] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In M. J. Wiener, editor, *Advances in Cryptology - CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[Kob87] N. Koblitz. Elliptic curve cryptosystems. *Math. of Comp.*, 48(177):203–209, 1987.

[Koc96] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[MDS99] T. S. Messerges, E. A. Dabbish, and R. H. Sloan. Power analysis attacks of modular exponentiation in smartcards. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 1999*, volume 1717 of *Lecture Notes in Computer Science*, pages 144–157. Springer, 1999.

[Mil86] V. S. Miller. Use of elliptic curves in cryptography. In H.C. Williams, editor, *Advances in Cryptology, CRYPTO '86*, volume 218 of *Lecture Notes in Computer Science*, pages 489–502, 1986.

[MMM04] H. Mamiya, A. Miyaji, and H. Morimoto. Efficient Countermeasures againstRPA, DPA, and SPA. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop, Cambridge*, volume 3156 of *Lecture Notes in Computer Science*, pages 343–356. Springer, 2004.

[Mon85] P.L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation, 44(170)*, pages 519–521, April 1985.

[MOV96] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[MPO05] S. Mangard, N. Pramstaller, and E. Oswald. Successfully attacking masked AES hardware implementations. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

[RSA78] R. L. Rivest, A Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21*, pages 120–126, 1978.

[YLMH05] S-M. Yen, W-C. Lien, S. Moon, and J. Ha. Power Analysis by Exploiting Chosen Message and Internal Collisions - Vulnerability of Checking Mechanism for RSA-decryption. In E. Dawson and S. Vaudenay, editors, *Mycrypt 2005*, volume 3715 of *Lecture Notes in Computer Science*, pages 183–1956. Springer, February 2005.

A Montgomery Arithmetic

A.1 Montgomery Multiplication

Algorithm A.5 MontMul: Montgomery modular multiplication algorithm

INPUT: n, $0 \leq x = (x_{v-1}x_{v-2}\dots x_1x_0)_b, y = (y_{v-1}y_{v-2}\dots y_1y_0)_b \leq n-1$, $n' = -n^{-1} \mod b$
OUTPUT: $x \times y \times r^{-1} \mod n$

Step 1. $a = (a_{v-1}a_{v-2}\dots a_1a_0) \leftarrow 0$
Step 2. **for** i from 0 to $v-1$ **do**
$u_i \leftarrow (a_0 + x_i \times y_0) \times n' \mod b$
$a \leftarrow (a + x_i \times y + u_i \times n)/b$
Step 3. **if** $a \geq n$ **then** $a \leftarrow a - n$
Step 4. **Return**(a)

A.2 Montgomery Exponentiation

Algorithm A.6 MontExp: Montgomery Square and Multiply from left to right

INPUT: integers m and n such that $m < n$, k-bit exponent $d = (d_{k-1}d_{k-2}\dots d_1d_0)_2$
OUTPUT: MontExp(m,d,n)= $m^d \mod n$

Step 1. $a = r$
Step 2. $\overline{m} = m \times r \mod n$
Step 3. **for** i from $k-1$ to 0 **do**
$a =$ MontMul(a,a,n)
if $d_i = 1$ **then** $a =$ MontMul(a,$\overline{m}$,n)
Step 4. $a = a \times r^{-1} \mod n =$ MontMul(a,1,n)
Step 5. **Return**(a)