Energy Exhaustion Attack on Barrett's Reduction

Mohannad S. Mostafa
ECE Department
University of Waterloo
mohannad.mostafa@uwaterloo.ca

Tanushree Banerjee
ECE Department
University of Waterloo
tanushree.banerjee@uwaterloo.ca

M. Anwar Hasan
ECE Department
University of Waterloo
ahasan@uwaterloo.ca

***Abstract*—Modular reduction is an integral part of the underlying computation in several cryptographic systems. This paper identifies a potential vulnerability in one of the most well-known reduction algorithms, namely Barrett's reduction algorithm. The vulnerability lies in the algorithm's *while loop* which is usually unbounded in straightforward implementation. An adversary can exploit the vulnerability by tampering with the *sign flag* of the processor's status register. Our work reveals that attacks exploiting such weaknesses can cause a huge amount of completely unnecessary arithmetic operations, rapidly draining energy from a cryptographic device that implements this reduction. Devices running on batteries, for example laptops and smart phones, will lose energy completely when faced with such attacks. This kind of consequences might also result if the sign flag becomes faulty due to natural causes such as voltage glitches or exposure to radiations. Countermeasures for the aforementioned vulnerability are also discussed in this paper.**

***Index Terms*—Barrett's reduction algorithm, cryptographic device, energy exhaustion attack, status register, countermeasure.**

I. Introduction

Power exhaustion attacks, that can be triggered by adversary induced faults or malicious activities, are designed to render the device nonfunctional through denial of service. For example, in [17] the authors investigated how an adversary exploits the cellular data service to exhaust the mobile phone's battery. Similar work was investigated in [14], where attacks were devised to drain the battery power of a mobile computer. Brownfield *et. al.* [8] found that wireless networks are vulnerable to similar attacks which can drain the power of the system reducing network lifetime from years to days. There has been quite an amount of work done so far, with different proposals of attacks trying to exhaust the power of the system in wireless sensor networks [18],[11],[13],[7]. To the best of our knowledge, no proposal has been made with the idea of using the vulnerability of a loop condition in the reduction algorithm.

This paper identifies a potential vulnerability in Barrett's reduction algorithm, that can lead to an energy exhaustion attack[1]. An adversary can exploit the vulnerability by running a malicious code or by injecting a fault in the processor, which is executing Barrett's reduction algorithm. We show that this attack can be successful in draining out the energy of the device by forcing a huge amount of unnecessary computations.

Modular reduction plays an important role in certain crypto algorithms. Public key encryption is primarily done on very large numbers which involve modular reduction. Reduction of an integer z modulo another integer p, is simply the remainder r obtained from dividing z by p ($z \bmod p$), where $0 \leq r < p$. The naive approach to modular reduction is to use a division algorithm, which is iterative in nature and is quite slow. In 1986, P. Barrett [2] proposed a faster algorithm for modular reduction. An important feature of this algorithm is the computational efficiency and the ease of implementation. In this paper, energy exhaustion attack on Barrett's algorithm is investigated. It can increase the computation time of the device which implements the algorithm manifolds leading to a huge loss of energy during its operation. To the best of our knowledge, this kind of energy exhaustion attack has not been mounted on any reduction algorithm before.

The main contribution of our work lies in identifying a potential vulnerability of Barrett's reduction algorithm which can lead to an energy exhaustion of the device implementing the algorithm. The energy exhaustion is due to the execution of extra subtraction operations and we derive lower and upper bounds on the number of such operations. In addition, we present a modified version of the Barrett reduction algorithm that avoids the aforementioned vulnerability.

The paper is organized as follows. In Section II, we provide essential preliminaries about Barrett's reduction algorithm. Next, in Section III we describe the potential vulnerability of Barrett's reduction algorithm and how this vulnerability can be used to mount an energy exhaustion attack. We examine the consequences of the attack in Section IV by performing a simple experiment on a general purpose processor. Countermeasures have been outlined in Section V. Finally, concluding remarks are added in Section VI.

II. Background

Barrett's reduction algorithm was introduced to implement public key encryption on digital signal processors [2]. In the reduction technique, the radix or base b is chosen

[1]In the literature [11], this is also known as power exhaustion attack.

Algorithm 1 Barrett's reduction algorithm [2]

Require: $p, b^{k-1} \leq p < b^k, b \geq 3, k = \lfloor \log_b p \rfloor + 1, 0 \leq z < b^{2k}$, and $\mu = \lfloor b^{2k}/p \rfloor$
Ensure: $z \mod p$
1: $\hat{q} \leftarrow \left\lfloor \lfloor z/b^{k-1} \rfloor . \mu / b^{k+1} \right\rfloor$
2: $r \leftarrow (z \mod b^{k+1}) - (\hat{q} \cdot p \mod b^{k+1})$
3: **if** $r < 0$ **then**
4: $\quad r \leftarrow r + b^{k+1}$
5: **end if**
6: **while** $r \geq p$ **do**
7: $\quad r \leftarrow r - p$
8: **end while**
9: **return** r

suitably ≥ 3 such that $b = 2^W$ where W is the processor word size. For modulus p, one pre-computes $\mu = \lfloor b^{2k}/p \rfloor$ where k is the number of digits in radix b representation of p, i.e $k = \lfloor \log_b(p) \rfloor + 1$. Barrett's algorithm is described in Algorithm 1.

The main reason behind the efficiency of this modular reduction algorithm is about estimating the quotient value in Step 1. The task of estimation involves operations, specifically shifting, truncation and multi-precision multiplication, that are less time consuming than a multi-precision division to determine the exact quotient $\lfloor \frac{z}{p} \rfloor$.

Lemma 1. *[10] Let q denote the actual quotient after dividng z by p, i.e., $q = \lfloor \frac{z}{p} \rfloor$. Then,*

$$q - 2 \leq \hat{q} \leq q$$

where the estimated quotient $\hat{q}$ is as defined in Algorithm 1.

Lemma 1 shows that the estimated value of the quotient $\hat{q}$ will be at the most two less than the actual quotient q. This implies that we need to subtract the estimated remainder $z - \hat{q}p$ from p at most *twice* to get the actual remainder $z - qp$.
In [6], the authors represented the estimated quotient as $\hat{q} = \left\lfloor \lfloor z/b^{2k-t} \rfloor . \mu / b^t \right\rfloor$ where $k < t \leq 2k$. Based on that, 90% of the values of $\hat{q}$ will be equal to $\lfloor \frac{z}{p} \rfloor$ and only in 10% of cases $\hat{q}$ will be two less than actual q. Compared to its main counterpart namely the Montgomery algorithm [15], the Barrett algorithm does not require p to be prime and it produces the actual remainder (without any scaling or multiplying factor).

We explain the main operations of Algorithm 1 using two sets of parameters:

Example 1. *Assume that $z = 204927$ and $\hat{q} \cdot p = 204317$, where $p = 59$, $b = 10$ and $k = 2$.*

$$\begin{aligned} z \mod b^{k+1} &= 9\ 2\ 7 \\ \hat{q} \cdot p \mod b^{k+1} &= 3\ 1\ 7 \\ r = z \mod b^{k+1} - \hat{q} \cdot p \mod b^{k+1} &= 6\ 1\ 0 \end{aligned}$$

Example 2. *Assume that $z = 314927$ and $\hat{q} \cdot p = 313998$, where $p = 59$, $b = 10$ and $k = 2$.*

$$\begin{aligned} z \mod b^{k+1} &= 9\ 2\ 7 \\ \hat{q} \cdot p \mod b^{k+1} &= 9\ 9\ 8 \\ r = z \mod b^{k+1} - \hat{q} \cdot p \mod b^{k+1} &= -7\ 1 \end{aligned}$$

In Example 1, we have a positive remainder which implies Step 3 to 5 in Algorithm 1 is skipped. On the other hand, in Example 2, we get a negative remainder which implies Step 3 to 5 are executed.

The value of z is greater than or equal to $\hat{q} \cdot p$, but under modular operation only the least significant digits are scanned, computed and stored in the remainder r. This step often renders r value as negative forcing it to enter the *if* condition in Step 3 of Algorithm 1 as illustrated in Example 2.

III. Potential Vulnerability

In this section, we identify a vulnerability in the Barrett's reduction algorithm and describe how this could be exploited to mount an energy exhaustion attack.

Lemma 2. *[10] For the symbols as defined in Algorithm 1.*

$$0 \leq z - \hat{q}p \leq b^{k+1}$$

Since $\hat{q}$ is an estimate of the actual quotient, $z - \hat{q}p$ is an estimate of the actual remainder. In Step 2 of Algorithm 1, Lemma 2 is used to reduce the cost of determining the estimated remainder, specifically, the $2k$-digit subtraction $(z - \hat{q}p)$ is reduced to the $(k+1)$-digit subtraction ($z \mod b^{k+1}$) - ($\hat{q}p \mod b^{k+1}$). This is however an "over" optimization leading to the possibility of a negative remainder in Step 2.
If we denote the remainder in Step 2 as r_2, it is easy to see that

$$-b^{k+1} < r_2 < b^{k+1}$$

When an arithmetic operation is performed using a general purpose processor and a negative result is produced, it typically affects the processor's status register. The *sign flag* is one of the bits of the status register which is set to 1 when the immediate past result was negative, otherwise it is cleared to 0. The value of the sign flag determines whether or not the **if** condition in Step 3 is satisfied. If the sign flag is 1, only then b^{k+1} is added, so that the value of the remainder after Step 4 is positive and equal to $z - \hat{q}p$.

In the past, attacks have been reported that exploits the possibility of changing the value of the sign flag [4] to induce an attack. The sign flag can be forced to 1 either be injecting a fault in the relevant circuit or making the processor run a malicious code. When the remainder is

positive, but the sign flag is forced to 1 after Step 2 of Algorithm 1, the **if** condition will satisfy and b^{k+1} will be added to the remainder. Denoting the value of the remainder after Step 4 as r_4, such an attack increases the value of r_4 such that it lies in the range of

$$0 < r_4 < 2b^{k+1}.$$

The consequence of the attack will be maximum if r is close to b^{k+1}. If r is positive, then an adversary can set the sign flag in the status register to 1. This can be done by deliberately making the *if* condition satisfiable. This would leave the reminder value of r to be $\approx 2 \cdot b^{k+1}$. The range of p as described in Algorithm 1 is $b^{k-1} \leq p < b^k$. The following lemma presents the number of times the *while loop* will execute in case of a successful attack.

Lemma 3. *A successful mounting of the attack on Algorithm 1 will result in the number of subtraction operations within the* while *loop ranging between b to* $2b^2$.

Proof. The minimum and the maximum values of the modulus can be b^{k-1} and b^k respectively. The number of iterations of the *while* loop is $\lfloor \frac{r}{p} \rfloor$. Let n be the number of iterations of the *while* loop. Then the range of n can be written as follows:

$$\begin{aligned} \frac{r_{min}}{b^k} < \ n \ < \frac{r_{max}}{b^{k-1}} \\ \frac{b^{k+1}}{b^k} < \ n \ < \frac{2b^{k+1}}{b^{k-1}} \\ b < \ n \ < 2b^2 \end{aligned}$$

□

Therefore, the worst case might arise when the value of p is minimum and the value of r is maximum. The huge amount of excessive computation will affect the power consumption of the system.

If the attack is successful, the minimum number of subtractions required is b. In the worst case, the number of subtraction operations is $2b^2$ as mentioned in Lemma 3. For a practical RSA application, 3072-bit number is used [10]. This number is stored in chunks of 64-bits to be fit in registers. Therefore, the total number of subtractions $= \frac{3072}{64} \times 2 \times b^2 \approx 2^{134}$. For the purpose of this calculation, the processor used is Intel Core i7 6500U. This processor can execute 20735 MIPS at 2.5 GHz. Thus, time taken to complete all subtractions is $\frac{2^{134}}{20735 \times 2^{20}} > \frac{2^{104}}{2^5} = 2^{99}$ seconds $\approx 2^{74}$ years. Below we provide Table I showing the impact of the attack in terms of the number of W-bit word subtraction operations that would be required for three values of W covering low to high end processors. The figures in the table provide a rough idea on the increased number of operations on these integers due to the proposed attack.

Implementation of Barrett's algorithm as described in Algorithm 1 is found in several practical software cryptographic libraries such as Bouncy Castle [9] and LibToMath [1]. When such an implementation running on a handheld device comes under the aforementioned energy exhaustion attack due to malicious codes, the device's battery will drain completely. The consequence can be worse if such an implementation running on-board a satellite, comes under an energy exhaustion attack. The sign flag might be set to 1 through some circuit faults caused by changes in environment.

W	$b = 2^W$	No. of subtractions	
		Min	Max
16	2^{16}	2^{16}	2^{33}
32	2^{32}	2^{32}	2^{65}
64	2^{64}	2^{64}	2^{129}

TABLE I: Estimation of the number of subtractions for different word sizes.

IV. Experimental Results

In order to see the impact of the attack, a simple script was written to run on victim's computer. The script runs the program and changes the sign flag in the status register before executing *if* condition. This results in the execution of the *if* condition from Algorithm 1 even though the number in the register is positive.
In a real world scenario, a malware capable of changing the sign flag while the Barrett's algorithm is executed, could get downloaded into the victim's computer during web surfing or other cyber activities. It is assumed that the program which uses the Barrett reduction in the victim's machine is in public knowledge. An attacker can use this information to develop the malware to accurately identify the execution of various steps of the reduction algorithm.

In our approach, we have implemented Barrett's reduction algorithm (Alg. 1). When the algorithm is executed normally, the average energy consumption has been computed using *powerstat v0.02.09* which is a tool that monitors the system and measures the power consumption. Then, the script that exploits the vulnerability is downloaded and run on the same machine and the average energy consumption is calculated.

To give a simple example of how the attack can change the number of times *while-loop* is run, the parameters of Algorithm 1 is set as follows: $z = 893994278$, $b = 16$, $p = 21987$, $k = 4$, $\mu = 195341$. Then, algorithm 1 computes $\hat{q} = 40660$ and $r = 2858$. In a normal execution of the algorithm, *if* condition will not be executed because $r > 0$. And since $r < p$, *while-loop* will not be executed as well. However, when the script runs the program on the machine, it changes the sign flag before *if* condition is executed, this will result $r = 1051434$. So, *while-loop* will be executed 47 times.

In our experiment, the average energy consumption and the discharge rate of the battery is calculated using

powerstat. On average, if the program is running normally without the attack being injected, the discharge rate of the battery is 8.75 Watt/s. However, when the attack is in place, the discharge rate of the battery is spiked to an average of 22.4 Watt/s in the worst case.

In this experiment, we used 192-bit numbers which is recommended by National Institute of Standards and Technology (NIST). The battery was fully discharged before the attack finishes. The experiment was run on a 64-bit processor Intel Core i7-6500U CPU @ 2.50GHz.

V. Countermeasure

A variety of countermeasures are possible against the above mentioned energy exhaustion attack to secure the cryptographic device, which implements the reduction algorithm. For example, the status registers of the processor will be the main target of the attacker. Securing the targeted storage area will prevent the sign flag to get set to 1 deliberately.

An alternative is to update r in Step 2 of Algorithm 1 as

$$r \leftarrow z - \hat{q}p$$

this increases the number of W-bit subtractions from $k+1$ to $2k$. We can lower the computation cost of updating r using the following lemma:

Lemma 4. *Let u and v be two n digit integers represented in radix $b \geq 3$, that is $u = (u_{n-1}u_{n-2}\cdots u_0)$ and $v = (v_{n-1}v_{n-2}\cdots v_0)$. If $0 \leq u - v < b^l$, where $0 < l < n-1$, then*

$$u - v = (u \mod b^{l+1}) - (v \mod b^{l+1}) \tag{1}$$

Proof. In order to comply with the condition $u - v < b^l$, the most significant digits of u and v should be same up to position $(l+1)$ which implies

$$u_i = v_i \text{ for } (l+1) \leq i \leq (n-1)$$

Thus

$$\begin{aligned} u - v &= (u_{n-1}u_{n-2}\cdots u_{l+1}u_l u_{l-1}\cdots u_0)_b \\ &\quad - (v_{n-1}v_{n-2}\cdots v_{l+1}v_l v_{l-1}\cdots v_0)_b \\ &= (u_l u_{l-1}\cdots u_0)_b - (v_l v_{l-1}\cdots v_0)_b \\ &= (u \mod b^{l+1}) - (v \mod b^{l+1}) \end{aligned}$$

□

Although we have used $b \geq 3$ in Lemma 4 to align with the description of Barrett's algorithm in Algorithm 1, this lemma is also valid for $b = 2$.

Note that

$$\begin{aligned} &0 \leq z - \hat{q}\cdot p < z - (q-2)p < 3p \\ &0 \leq z - \hat{q}\cdot p < b^{k+1} \end{aligned} \tag{2}$$

Applying (1), we can write (2) as follows.

$$0 \leq (z \mod b^{k+2}) - (\hat{q}p \mod b^{k+2}) < b^{k+1}$$

Therefore, in Step 2 of Algorithm 1, the value of r can be updated as follows where the result is always positive:

$$r \leftarrow (z \mod b^{k+2}) - (\hat{q}\cdot p \mod b^{k+2}) \tag{3}$$

This is illustrated in the following example:

Example 3. *We continue with the same parameters used in Example 2. The modification makes the algorithm proceed with 4 least significant digits instead of 3. Therefore under modulus b^{k+2}, an extra digit is checked which eliminates the chance of r to be negative after step 2 of Algorithm 1.*

$$\begin{aligned} z \mod b^{k+2} &= 5\ 9\ 2\ 7 \\ \hat{q}\cdot p \mod b^{k+2} &= 4\ 9\ 9\ 8 \\ r = z \mod b^{k+2} - \hat{q}\cdot p \mod b^{k+2} &= 1\ 9\ 2\ 9 \end{aligned}$$

Below we state a modified version of Algorithm 1, in which we replace the *while loop* in Algorithm 1 by two *if conditions* (Step 3-8 of Algorithm 2).

Algorithm 2 Modified Barrett's reduction algorithm

Require: $p, b^{k-1} \leq p < b^k, b \geq 3, k = \lfloor \log_b p \rfloor + 1, 0 \leq z < b^{2k}$, and $\mu = \lfloor b^{2k}/p \rfloor$
Ensure: $z \mod p$
1: $\hat{q} \leftarrow \left\lfloor \lfloor z/b^{k-1} \rfloor . \mu / b^{k+1} \right\rfloor$
2: $r \leftarrow (z \mod b^{k+2}) - (\hat{q}\cdot p \mod b^{k+2})$
3: **if** $r \geq p$ **then**
4: $\quad r \leftarrow r - p$
5: **end if**
6: **if** $r \geq p$ **then**
7: $\quad r \leftarrow r - p$
8: **end if**
9: **return** r

Compared to Algorithm 1, the modified version requires 1 extra W-bit subtraction. This is due to the use of b^{k+2} as modulus in Step 2 (as opposed to modulus b^{k+1} in Algorithm 1). Also, if only the necessary words are computed for $\hat{q}\cdot p$, the modified algorithm requires the computation of one extra word compared to the original.

VI. Conclusion

In public key cryptosystems, like RSA or elliptic curve signature schemes, the efficiency of implementations mainly lie on the speed of modular multiplications. A modular multiplication can be performed as an integer multiplication followed by modular reduction. In order to achieve a 128-bit security level, the RSA and the Elliptic Curve Cryptography use 3072 and 256 bits long moduli, respectively. Our work shows a potential vulnerability in the well-known Barrett's reduction algorithm. We show

how an injected fault can cause huge number of unnecessary computations, thereby disrupting the normal functioning of a cryptographic device that had implemented Barrett's algorithm. The normal reduction takes 2 subtraction operations but the energy exhaustion attack can shoot the number of subtractions up to $2b^2$, where b is the radix of the number to be reduced. It will adversely affect the device and draw all the energy of the battery.

Finally, it is worth mentioning that the Barrett reduction's vulnerability identified in this article may exist in other algorithms that have unbounded control flows or loop constructs.

Acknowledgment

This research was supported in part through an NSERC grant awarded to Dr. Hasan. A slightly different version of the manuscript will appear in the proceedings of the 2018 IEEE TrustCom Conference.

References

[1] Libtomath user manual. http://www.gtoal.com/src/mobi/clit18/libtommath-0.41/bn.pdf, accessed: 2010-09-30
[2] Barrett, P.: Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In: Conference on the Theory and Application of Cryptographic Techniques. pp. 311–323. Springer (1986)
[3] Biham, E., Shamir, A.: Differential Fault Analysis of secret key cryptosystems. In: Annual International Cryptology Conference. pp. 513–525. Springer (1997)
[4] Blömer, J., Otto, M., Seifert, J.P.: Sign change fault attacks on elliptic curve cryptosystems. In: Fault Diagnosis and Tolerance in Cryptography, pp. 36–52. Springer (2006)
[5] Blömer, J., Seifert, J.P.: Fault based cryptanalysis of the Advanced Encryption Standard (AES). In: International Conference on Financial Cryptography. pp. 162–181. Springer (2003)
[6] Bosselaers, A., Govaerts, R., Vandewalle, J.: Comparison of three modular reduction functions. In: Advances in Cryptology – CRYPTO 1993. pp. 175–186. Springer (1994)
[7] Boubiche, D.E., Bilami, A.: A defense strategy against energy exhausting attacks in wireless sensor networks. Journal Of Emerging (2013)
[8] Brownfield, M., Gupta, Y., Davis, N.: Wireless sensor network denial of sleep attack. In: Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC. pp. 356–364. IEEE (2005)
[9] Castle, B.: The legion of the bouncy castle (2007)
[10] Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to elliptic curve cryptography. Springer Science & Business Media (2006)
[11] Hsueh, C.T., Wen, C.Y., Ouyang, Y.C.: A secure scheme against power exhausting attacks in hierarchical wireless sensor networks. IEEE Sensors journal 15(6), 3590–3602 (2015)
[12] Intel: Ia-32 architectures software developers manual. Volume 3A: System Programming Guide, Part 1(64) (64)
[13] Jo, M., Han, L., Tan, N.D., In, H.P.: A survey: energy exhausting attacks in mac protocols in wbans. Telecommunication Systems 58(2), 153–164 (2015)
[14] Martin, T., Hsiao, M., Ha, D., Krishnaswami, J.: Denial-of-service attacks on battery-powered mobile computers. In: Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on. pp. 309–318. IEEE (2004)
[15] Montgomery, P.L.: Modular multiplication without trial division. Mathematics of computation 44(170), 519–521 (1985)
[16] Piret, G., Quisquater, J.J.: A Differential Fault Attack technique against SPN structures, with application to the AES and KHAZAD. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 77–88. Springer (2003)
[17] Racic, R., Ma, D., Chen, H.: Exploiting MMS vulnerabilities to stealthily exhaust mobile phone's battery. In: Securecomm and Workshops, 2006. pp. 1–10. IEEE (2006)
[18] Shakhov, V.V.: Protecting wireless sensor networks from energy exhausting attacks. In: International Conference on Computational Science and Its Applications. pp. 184–193. Springer (2013)
[19] Takahashi, J., Fukunaga, T.: Differential Fault Analysis on AES with 192 and 256-bit keys. IACR Cryptology ePrint Archive 2010, 23 (2010)