Timing Attack against protected RSA-CRT implementation used in PolarSSL

Cyril Arnaud[1] and Pierre-Alain Fouque[2]

[1] École de l'Air,
cy.arnaud@orange.fr
[2] Université Rennes 1
pierre-alain.fouque@ens.fr

Abstract. In this paper, we present a timing attack against the RSA-CRT algorithm used in the current version 1.1.4 of PolarSSL, an open-source cryptographic library for embedded systems. This implementation uses a classical countermeasure to avoid two previous attacks of Schindler and another one due to Boneh and Brumley. However, a careful analysis reveals a bias in the implementation of Montgomery multiplication. We theoretically analyse the distribution of output values for Montgomery multiplication when the output is greater than the Montgomery constant, R. In this case, we show that an extra bit is set in the top most significant word of the output and a time variance can be observed. Then we present some proofs with reasonable assumptions to explain this bias due to an extra bit. Moreover, we show it can be used to mount an attack that reveals the factorisation. We also study another countermeasure and show its resistance against attacked library.

Keywords: Side-channel attacks, timing attacks, embedded device security, cryptography research

1 Introduction

The implementation of light-weight open-source cryptographic libraries is an important security issue. In this paper, we study timing attacks on the current version, 1.1.4, of the open source library PolarSSL [1] which is a well-known library, widely used in embedded systems. We look at the security of the exponentiation used in PolarSSL which is similar to the one used in OpenSSL [2] but presents some differences.

There are mainly three timing attacks [[3,4,5], on the RSA with Chinese Remainder Theorem (CRT) using Montgomery multiplication (MM). In the latter algorithm, a conditional branch leaks some information about the modulus and if RSA-CRT is used, the modulus is a prime factor of the RSA modulus N. These attacks were performed without blinding and can be applied to the version of Montgomery multiplication which contains a final conditional subtraction, called extra-reduction, for reducing the output. A classical and efficient countermeasure consists in using dummy operations to make the time computation constant in the Montgomery implementation.

In this work, we study the dummy operation countermeasure used to avoid these three attacks and we propose an efficient practical chosen-ciphertext attack on this countermeasure used in PolarSSL. First, we show that even though the extra-reduction is not visible, since the dummy operations mask them, when the ouput of Montgomery multiplication before any conditional subtraction is greater than R, an *extra-bit* is set and the computation takes more time when the output is less than R. Then we study the probability of an extra bit in Montgomery multiplication and we show how to exploit it by mounting an attack.

Our paper is organized as follows : §2 describes the different algorithms used to compute efficiently the exponentiation operations. We also recall how the previous attacks operate and the countermeasures that were proposed to defeat these attacks. In §3, we describe the specifications of the PolarSSL exponentiation implementations, we also show that some bias is still present in the time computation and we explain the probability of an extra bit. In §4, our attack details are addressed, and §5 presents our experimental results. We investigate possible countermeasures in §6. We conclude our paper in §7.

2 Background

2.1 Montgomery multiplication

First proposed by Montgomery in [6], the MM algorithm provides an efficient method for computing modular multiplications (given q an odd integer and two integers $a, b \in \mathbb{Z}_q$, we compute $ab \; mod \; q$) and squaring ($a^2 \bmod q$).

In this subsection, we describe a generic multi-precision variant of MM. Long integers are represented as a sequence of words, the size of words is denoted by w. Let $r = 2^w$ and $s = \left\lceil \frac{|Q|}{w} \right\rceil$ represents the number of required words of size w. Large integers are written in basis r, and we note digits in lowercase. Thus, those numbers have the forms $A = \sum_{i=0}^{i=s-1} a_i r^i = (a_{s-1}, ..., a_0)_r$, $B = (b_{s-1}, ..., b_0)_r$, $Q = (q_{s-1}, ..., q_0)_r$, where $0 \leq a_i, b_i, q_i < r$. Let $R = r^s$ and $\mu_0 = -\frac{1}{q_0} \bmod r$. Big number library implement a w-bit multiplication used during the multiplication of a word with a large integer, which is denoted by $\otimes_w$. Multi-precision variant of MM is described in figure 1.

In order to use MM all variables must first be converted in Montgomery representation $\bar{A}(= AR \bmod Q)$ by computing $\textsc{multimontmul}(A, R^2, Q) = \bar{A}$. At the end, the result of the algorithm can be converted to classical representation by performing $\textsc{multimontmul}(\overline{A}, 1, Q) = A \bmod Q$.

In line 8 of $\textsc{multimontmul}$, the conditional subtraction, $Z = Z - Q$, is called an *extra-reduction*. It is occasionally carried out to ensure that the result Z is in the range $[0, Q)$ and causes a timing difference. Timing attacks [3,4,5] detect the timing difference according to the extra-reduction is executed or not. In fact, Schindler [3] showed that the probability of an extra-reduction occurs in

```
1: function MULTIMONTMUL($A, B, Q$)
2:     $Z = (z_s, ..., z_0)_r \leftarrow 0$
3:     for $i = 0$ to $s - 1$ do
4:         $u \leftarrow ((z_0 + a_i \times b_0) \times \mu_0) \bmod r$
5:         $Z \leftarrow (Z + a_i \otimes_w B)$
6:         $Z \leftarrow (Z + u \otimes_w Q)\ div\ r$
7:     if $Z \geq Q$ then $Z \leftarrow Z - Q$
8:     return $Z$ ($= ABR^{-1} \bmod Q$)
```

Fig. 1: MM Multi-precision variant. $A, B < Q$.

```
1: function MONTMUL($A, B, Q$)
2:     $Z = (z_{2s}, ..., z_0)_r \leftarrow 0$
3:     for $i = 0$ to $s - 1$ do
4:         $u \leftarrow (z_i + a_i \times b_0) \times \mu_0$
5:         $(z_{2s}, ..., z_i) \leftarrow ((z_{2s}, ..., z_i) + a_i \otimes_w B)$
6:         $(z_{2s}, ..., z_i) \leftarrow ((z_{2s}, ..., z_i) + u \otimes_w Q)$
7:         $z_i \leftarrow a_i$
8:     $G \leftarrow (z_{2s}, ..., z_s)$
9:     if $G \geq Q$ then
10:        SUB($G, Q, s$)                ▷ Extra-reduction
11:    else
12:        SUB($Z, G, s$)                ▷ defence
13:    return $G$ ($= ABR^{-1} \bmod Q$)
```

Fig. 2: POLARSSL's Montgomery Multiplication multi-precision with countermeasure. $A, B < Q$.

MULTIMONTMUL($\bar{X}, B, W$), where B is uniformly distributed in $\mathbb{Z}_q$, is:

$$P\ (\text{extra-reduction in MONT}(\bar{X}, B, Q)) = \frac{\bar{X} \bmod Q}{2R} \qquad (1)$$

According to (1), when $\bar{X}$ rises and approaches a multiple of Q, the probability of an extra-reduction increases. At exact multiples of Q, the probability of an extra-reduction is null.

2.2 Modular exponentiation algorithm : sliding window

MM is particularly interesting when it is combined with the modular exponentiation algorithm to compute $m = c^d \bmod q$. OpenSSL[2] and PolarSSL[1] use an optimization of the square-&-multiply algorithm. This algorithm, called Sliding Windows Exponentiation (SWE), considers block of bits of the exponent rather than bits. The exponent d is split into windows of size $wsize$ (depending on the size of d), where the windows are not always contiguous. Different ways are available for choosing windows. As shown in figure 3, SWE requires a precomputed table which is computed before the exponentiation process. During the modular exponentiation phase (computing $\bar{M}$), this table is used to process $wsize$ bits of d at each iteration. In both phases of SWE (precomputation phase and exponentiation), the MM is used.

2.3 Decryption of RSA with Chinese Remainder Theorem

Let $N = PQ$ be a n-bit RSA modulus, where P and Q are prime numbers. The public key is denoted by (N, e) and the associated private key by (D, P, Q). RSA decryption consists in computing a modular exponentiation $M = C^D \bmod N$, where C is the ciphertext to decrypt. A well-known optimization of this operation is the RSA-CRT which takes advantage of the decomposition in prime factor of N. Then, RSA-CRT reduces the computation time by a factor of about 75%. The RSA-CRT,with Garner's recombination, is shown in figure 4. The attacked implementations compute lines (2) and (3) using the SWE exponentiation algorithm.

```
1: function EXPONENT($C, D, Q$)
2:     if $C \geq Q$ then
3:         $C \leftarrow C \bmod Q$
4:     $\bar{C}_1 \leftarrow \textsc{mont}(C, R^2, Q)$
5:     Precomputing table phase
6:     Modular exponentiation phase
7:     $M \leftarrow \textsc{mont}(\bar{M}, 1, Q)$
8:     return $M$ $(= C^D \bmod Q)$
```

Fig. 3: Sliding window exponentiation.

```
1: function RSA-CRT($C, D, P, Q$)
2:     $C_p \leftarrow C^{D_p} \bmod P$
3:     $C_q \leftarrow C^{D_q} \bmod Q$
4:     $T \leftarrow (C_q - C_p)\pi \bmod Q$
5:     $M \leftarrow TP + C_p$
6:     return $M$ $(= C^D \bmod N)$
```

Fig. 4: RSA-CRT decryption. $D_p = D \bmod (P-1)$, $D_q = D \bmod (Q-1)$ and $\pi = P^{-1} \bmod Q$ are precomputed.

General idea of timing attacks on RSA-CRT. According to (1), during the modular exponentiation, if two chosen ciphertext X and Y are decrypted, when $\bar{X} < \bar{Y} < Q$ the total number of extra-reductions is greater than the case of $\bar{X} < Q < \bar{Y}$. So, by detecting time difference to perform the decryption of $\bar{X}$ and $\bar{Y}$ with RSA-CRT, we can reduce the search space of Q.

Overview of known timing attacks on RSA-CRT. We focus on two attacks [5,4] against OpenSSL 0.9.7 without using any blinding countermeasure. OpenSSL implements four optimizations for RSA decryption : CRT, SWE, MM and two multiplication procedures, normal and Karatsuba's algorithm. OpenSSL performs Karatsuba's multiplication when multiplying two integers of the same number of words, otherwise uses normal routine. Karatsuba is faster than normal multiplication. The two attacks exploit the factorization of the RSA modulus by exploiting the time variance of RSA-CRT decryption in OpenSSL which depends on the number of extra-reductions in MM and the choice of multiplication procedure. The attacks perform a binary search to find the bits of Q bit-by-bit.

In [5], Boneh and Brumley show that the effect of extra-reductions and Karatsuba depends on the position of the bit being recovered. Thus, there is a prevailing parameter which has an influence on the time variance. Two ways are explored for recovering a bit of Q. In [5], timing attack exploits MM which are carried out during the modular exponentiation phase of SWE while [4] exploits those are carried out during the precomputation phase.

Boneh and Brumley [5] propose two countermeasures to make RSA-CRT decryption time independent on the input ciphertext. The first one is to use only one multiplication procedure. The other one is to carry out a dummy subtraction if an extra-reduction is not needed. The result of dummy operation is not used. This approach is also suggested by Schindler [3].

3 PolarSSL's implementation of RSA-CRT decryption

PolarSSL [1] is a light-weight open source cryptographic and SSL/TLS library written in C. PolarSSL is implemented for embedded systems and has also been ported to Windows and Linux (32 and 64 bit). The PolarSSL implementation of RSA decryption uses, as optimization, the CRT, SWE and MM with defence. We describe the last two algorithms below.

3.1 Montgomery multiplication multi-precision

PolarSSL implements only one procedure for computing the MM and accepts multiplication for $w = 8$, 16, 32, or 64. PolasSSL's MM multi-precision is given in figure 2, where function $\textsc{sub}(X, Y, s)$ returns $X \leftarrow X - Y$ for the s least significant words. In this case, the division of Z by r is performed by bumping a counter on Z. After s iterations, the $s + 1$ most significant words of Z are equals to $ABR^{-1} \bmod Q$ and the s least significant words to A.

In the following, we assume that for fixed parameters Q and r the running times to perform lines 8 to 13 are identical for all inputs. It is worth noticing that if an extra-reduction is not needed, a dummy subtraction is carried out (line 12). Those two operations (lines 10 and 12), subtractions on s words, take the same time to be performed. Moreover, we check that compiler optimizations do not remove the conditional branch. In all cases, the study of the assembly code reveals that the dummy subtraction is still present for all compiler optimizations.

In this paper we perform a cryptanalysis based on the conditions under which G is greater than R before any conditional subtraction is performed.

3.2 Timing variation in PolarSSL's Montgomery multiplication multi-precision

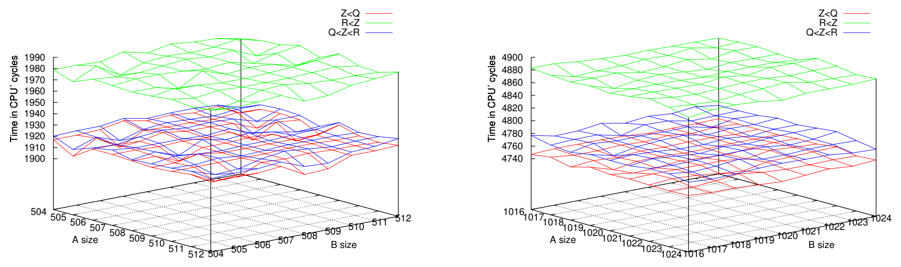
The dummy subtraction is used in $\textsc{montmul}(A, B, Q)$ to make the time required to perform the multiplication independent on the A and B operands. For more clarity, we study the behaviour of the generic Montgomery multiplication multi-precision figure 1.

Theorem 1. *[10] For inputs $0 \leq A, B < Q$,* $\textsc{multimontmul}$*(A,B,Q) returns the ouput $Z \equiv ABR^{-1} \bmod Q$ satisfying $ABR^{-1} \leq Z < Q + ABR^{-1}$ before any conditional subtraction.*

We suppose that $\frac{R}{2} < Q < R$. This assumption is true for standard RSA key lengths, such as 512, 1024 or 2048 bits, but also when these lengths are a multiple of world size. In this case, theorem 1 implied that $Z < 2R$. Moreover, if the value of Z is greater than R then the value of the top most word of Z is 1, i.e. $z_s = 1$, this bit is called extra bit.

In the source code of the multiplier, a while loop propagates a carry until no further carry is generated. Then, if the output Z is greater than R, during the s^{th} computation of line 6 of figure 1, the while loop is used to carry propagation up to the top most word of Z. Thus, a timing difference, whether the extra bit is carried out or not, could allow an attacker to mount a timing attack.

We performed experiments to show if an attacker could observe a timing difference in MONTMUL. We generated two random numbers A, B with known size, converted in Montgomery representation, and a prime number Q where $\frac{R}{2} < Q < R$. We sort the time in CPU's clock ticks to perform MONTMUL(A, B, Q) according to the size numbers and if an extra bit, an extra-reduction without extra bit or neither of them is carried out. The delay observed, Figure 5, to carry out Z before any conditional subtraction confirmed explanation above about timing difference. For the whole of MONTMUL(A, B, Q) we observed the same delay between curves. Thus, extra-reduction, if $Q \leq Z < R$, was actually masked by the dummy subtraction. However, the delay was smaller than the bias observed in previous attacks and it was proportional to the bitsize of Q, noted $|Q|$. In addition, compiler optimisation did not affect delay between curves.



(a) $|Q| = 512$ (b) $|Q| = 1024$

Fig. 5: MONTMUL(A, B, Q) without any conditional subtraction ($w = 64$ bits).

3.3 The Probability of an extra bit

Here we study the distribution of extra bit in MM algorithm inside the modular exponentiation algorithm. We investigate the distribution for some cases and we show that the probability for an extra bit is different for a squaring operation (P_{square}), for a multiplication with two random A and B (P_{mul}) and for a multiplication with a particular value $C \in \mathbb{Z}_Q$ (P_C).

First of all, to establish the probability distribution for the MM output Z after s iterations (figure 1), three reasonable assumptions are made :

1. Q is large and so we can switch from discrete to continuous method,
2. Colin D. Walter [11] showed that during an exponentiation, inputs to MM are uniformly distributed mod Q and independent,
3. for inputs A and B to MM during an exponentiation, the output Z before the conditional subtraction is uniformly distributed over the interval $[ABR^{-1}, Q + ABR^{-1})$ [11].

Lemma 1. *During the exponentiation, for input* $0 \leq A, B < Q$*, the probability of an extra bit is not null if and only if* $Q > \frac{\sqrt{5}-1}{2} \times R$.

Proof (of lemma 1). According to assumptions (2) and theorem 1, we obtain $ABR^{-1} \leq Z < Q + Q^2R^{-1}$. An extra bit is set if and only if $Z \geq R$ after s iterations. Then, if an extra bit is set, $R \leq Z < Q + Q^2R^{-1}$. This inequality is true if and only if $Q^2R^{-1} + Q - R > 0$, solving for Q we obtain $Q > \frac{\sqrt{5}-1}{2} \times R$.

□

Lemma 2. *During the exponentiation, for input* $0 \leq A, B < Q$ *and* $C \in \mathbb{Z}_Q$ *fixed, the probability of an extra bit is :*

1. $P_{mul} = \begin{cases} \frac{Q}{4R} + \frac{(R-Q)^2R}{Q^3} \times (\frac{3}{4} + \frac{1}{2}\log(\frac{Q^2}{(R-Q)R})) - \frac{(R-Q)}{R} & \text{if } Q > \frac{\sqrt{5}-1}{2} \times R \\ 0 & \text{otherwise} \end{cases}$

2. $P_{square} = \begin{cases} \frac{Q}{3R} + \frac{2(R-Q)\sqrt{(R-Q)R}}{3Q^2} - \frac{(R-Q)}{R} & \text{if } Q > \frac{\sqrt{5}-1}{2} \times R \\ 0 & \text{otherwise} \end{cases}$

3. $P_C = \begin{cases} \frac{C}{2R} + \frac{(R-Q)^2R}{2CQ^2} - \frac{(R-Q)}{R} & \text{if } C > \frac{(R-Q)R}{Q} \text{ and } Q > \frac{\sqrt{5}-1}{2} \times R \\ 0 & \text{otherwise} \end{cases}$

The proof is in appendix 2.

If $Q \to R$, then the probability of an extra bit for a squaring, for a multiplication with random numbers and for a multiplication with a particular value in MONTMUL tends to $\frac{1}{3}$, $\frac{1}{4}$ and $\frac{C}{2R}$ respectively. On the other hand, for $Q \to \frac{\sqrt{5}-1}{2} \times R$, these probabilities tend to 0.

Corollary 1. *For* $Q > \frac{\sqrt{5}-1}{2} \times R$ *and* $X, Y \in \left(\frac{(R-Q)R}{Q}, Q\right)$*, if* $X > Y$ *then* $P_X > P_Y$.

Proof (of corollary 1). Let f a function denoted by :

$$\begin{aligned} f : \left(\frac{(R-Q)R}{Q}, Q\right) &\to (0,1) \\ C &\mapsto \frac{C}{2R} + \frac{(R-Q)^2R}{2CQ^2} - \frac{(R-Q)}{R} \end{aligned}$$

$\forall\, C > \frac{(R-Q)R}{Q}$, $f'(C) = \frac{1}{2R} - \frac{1}{C^2} \times \frac{(R-Q)^2R}{2Q^2} > 0$.
Then f is strictly increased in interval $\left(\frac{(R-Q)R}{Q}, Q\right)$. Thus if X, Y within this interval with $X > Y$ then $P_X > P_Y$.

□

3.4 Sliding window exponentiation

In this subsection we treat a variant of SWE, a fixed window exponentiation (FIXEDEXPONENT) which is used in PolarSSL. It differs from the SWE in OpenSSL. In PolarSSL, the precomputation table phase computes $\bar{C}_1$ which is the ciphertext in Montgomery representation, $\bar{C}_1^{2^{wsize-1}}$ with successive $wsize-1$ squares of $\bar{C}_1$, and stores in the table $\bar{C}_i = \text{MONTMUL}(\bar{C}_1, \bar{C}_{i-1}, Q)$, for $i = (2^{wsize-1}+1)$ to $(2^{wsize}-1)$. In the modular exponentiation, a block of bits (only one or $wsize$) of D are processed at each iteration. The secret exponent D is split into windows of fixed size $wsize$ (depending on the size of D) where the most significant bit is 1. We denote by w_d the value of the window. This window is used to carry out $\bar{M} = \text{MONTMUL}(\bar{M}, \bar{C}_{w_d}, Q)$ during an iteration of the modular exponentiation phase.

For fixed Q and D and according to lemma 2, the number of extra bits in modular exponentiation phase of $\text{FIXEDEXPONENT}(C, D, Q)$ is constant and independent of ciphertext input C. However, using lemma 2, numbers of extra bits depend on $\bar{C}_1$ which is equal to $CR \bmod Q$. According to corollary 1, the secret modulus Q can be found by using a chosen ciphertext.

It is worth noticing that Schindler's attack fails while Boneh and Brumley timing attack [5] should work against PorlarSSL's RSA-CRT.

4 A timing attack on PolarSSL

In this section, we will suppose the size of RSA modulus is equal to 512, 1024, or 2048 bits. The precomputing table phase of SWE in modulo Q requires $2^{wsize-1} - 1$ (from 1 to 32) MM with $\bar{C}_1$, i.e. ciphertext in Montgomery representation. Therefore, we exploit these operations in our attack. Because of the bias in PolarSSL's Montgomery multiplication is very small, we also use efficient statistical hypothesis tests such as F-test and T-test in our attack model.

4.1 Two-sample hypothesis testing

Two-sample hypothesis testing is a method estimating two independent samples parameters which are extracted from two populations.

An F-test is used to test equal variance of the two populations. The value generated by F-test, F_{observed}, is used to verify the timing sampling correctness. In some case, we invalidate some erroneous measurements due to the noise of other processes running on the machine. The F-test is :

$$F_{\text{observed}} = \frac{\frac{n_1}{n_1-1}{s_1}^2}{\frac{n_2}{n_2-1}{s_2}^2}, \tag{2}$$

where n_1, n_2 are sample sizes and ${s_1}^2$, ${s_2}^2$ are the sample variances. Let F_α is the critical value of the F-distribution with $(n_1 - 1,\, n_2 - 1)$ degrees of freedom and a significance level α. If $F_{\text{observed}} > F_\alpha$ then variances of the population are different.

In two-sample t-test, we compare two independent sample means from two populations with a same variance. The value generated by T-test, T_{observed}, is used in the decision strategy for recovering bit. The two-sample t-test is :

$$T_{\text{observed}} = \frac{(\bar{x}_1 - \bar{x}_2) - d_0}{S_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}, \tag{3}$$
$$\text{where } {S_p}^2 = \frac{(n_1-1){s_1}^2 + (n_2-1){s_2}^2}{n_1+n_2-2},$$

where $\bar{x}_1$ and $\bar{x}_2$ are the sample means. Let t_α is the critical value of the Student's t distribution with $(n_1 + n_2 - 2)$ degrees of freedom and a significance level α. If the means of the two samples are right, T_{observed} is less than T_α, otherwise greater than the critical value. Furthermore, we define a parameter T_β to increase the correctness of guess.

4.2 Our attack method

Let N be an RSA modulus with $N = pq$, $q < p$, $|N| = n$ and $|q| = |p| = n/2$. To factorize N, it is enough to recover the half most significant bits of p or q, since Coppersmith's [7] algorithm allows one to recover the complete factorisation of N. However, our method allows us to recover around the $n/2 - \log_2(NS) - 1$ most significant bits of modulus, where the parameter NS is called the neighbourhood size and used to increase statically the bias (for more details about NS refer to [5]).Thus, our attack ensures the recovery of bits of p or q one at a time, from most significant to least. Let $q = (1, q_1, ..., q_{\frac{n}{2}-1})$ is the binary coded of q and assume that the attacker knows the k most significant bits of q. We recover q_k as follows :

– **Step** 1 : Generate g and g_h where $g = (1, q_1, , ..q_{k-1}, 0, \ldots, 0)$ and $g_h = (1, q_1, , ..q_{k-1}, 1, ..., 0)$. If $q_k = 1$ then $g < g_h < q$, otherwise $g < q < g_h$.
– **Step** 2 : For $j = 0$ to $NS - 1$ compute $u_j = (g + j)R^{-1} \bmod N$ and $u_{hj} = (g_h + j)R^{-1} \bmod N$. The parameter neighbourhood size, i.e. NS, depend on experiment parameters.
– **Step** 3 : For $j = 0$ to $NS - 1$ measure the time to decrypt both u_j and u_{hj} . Let $T_{u_j} = Time(RSA\text{–}CRT(u_j, d, p, q))$ and $T_{u_{hj}} = Time(RSA\text{–}CRT(u_{hj}, d, p, q))$. We obtain two groups.
– **Step** 4 : Take a time sampling of the two groups, noted ζ_u and ζ_{u_h} .
– **Step** 5 : Compute $F_{observed}$ of ζ_u and ζ_{u_h}. If $F_{\text{observed}} > F_\alpha$, we assume that timing samples are invalidated. So, we replay step 3.
– **Step** 6 : Find the largest interval t in ζ_u and ζ_{u_h} where $F_{observed} \approx 1$.
– **Step** 7 : Compute $T_{observed}$ of ζ_u and ζ_{u_h}, in interval t. If $T_{\text{observed}} < T_\alpha$, the attacker assumes that $q_k = 1$. Otherwise, if $T_{\text{observed}} > T_\beta$, we fixed empirically $T_\beta = 10$, then $q_k = 0$. If $T_\alpha < T_{\text{observed}} < T_\beta$, we assume that timing samples are invalidated and we replay step 3.

Unlike Boneh and Brumley [5], our attack does not need a particular phase to determine the first few bits: the attack begins with $g = (1, 0, ..., 0)$ and recovers p or q.

An optimization of step 3 was to carry out only one of the two samples ζ_u and ζ_{u_h}. Indeed, one of sample of the previous recovering bit could be reused during the recovering process. If we guessed that $q_k = 0$ (resp. $q_k = 1$) then we reused ζ_u (resp. ζ_{u_h}) during the decision process of the next bit. Thus, the number of chosen ciphertext was divided by two. However, we carried out the two samples ζ_u and ζ_{u_h} when these timing samples are invalidated.

5 Experimental results

We performed our attack against the latest version 1.1.4 of PolarSSL with countermeasure (dummy subtraction). All of the experiments presented were run under Ubuntu 12.04 LTS 64 bits on an Intel Core i7. We compiled, using GCC 4.6.3, PolarSSL with its default value : `-D_FILE_OFFSET_BITS=64 -O`. We generated randomly all keys with PolarSSL's key generation routine. To get an accurate time measurement of RSA-CRT decryption, we used Time Stamp Counter (TSC) and Performance Monitor Counters (PMC).

5.1 Time measurement

A well known method for measuring time is the TSC[8], a 64-bit register, which counts front side bus ticks and multiplies it by the CPU's frequency. The TSC is available on Intel CPU since introduction of the Pentium. This counter is read using the `rdtsc` assembly instruction which is not a privilege operation.
Our experiments were performed on a multi-core platform which compromised the use of the TSC registers. Thus, we implemented another way too.

PMC[8] provides the capability to monitor performance events and measure ticks for each core. Every PMC has a number which allows it to be referenced. PMC are supported by Model Specific Register (MSR). MSR are specific to a particular CPU which store data and set information for the CPU. PMC can be read using `rdmsr` while `wrmsr` writes to an MSR. We used these instructions to store ticks values for a chosen core. These instructions must be executed using privilege ring 0. Then, we need to use a kernel driver.

5.2 Experimental results

In this subsection, we present the result of four experiments with the three sizes of modulus : 512, 1024 and 2048 bits. In each case, we showed that recovering half of the most significant bits of a prime factor is possible with a success probability around 100%. The first two experiments were performed in the same computing process. We measured directly the time to perform RSA-CRT for a chosen ciphertext. These experiments ensured we check the effectiveness of our statistical choices and the accuracy of using PMC. The last two experiments were obtained in inter-process in nature, and via TCP socket. We implemented a simple TCP server and client with the two processes ran on the same machine (server and client). The TCP server read a binary string sent by the client which is in PolarSSL's multi-precision representation. When it completed the decryption of the RSA-CRT, it returned 0 to the client. The TCP client measured the time between sending a message and receiving a response.

We attacked several random keys to determinate the efficiency of our method. The size of neighbourhoods was gained empirically. We denote by $\Delta_0 = mean(\zeta_u) - mean(\zeta_{u_h})$ when q_i is 0 and Δ_1 when it is 1. Due to Karatsuba's multiplication, in [5,4] the decryption time during RSA-CRT is variable (depending on the weight of recovering bit) when in PolarSSL is quite stable.

These different experiments demonstrated that our statistical tests are reliable and suitable for timing attack. Moreover, measuring time with PMC made our attack more efficient.

Experiment 1 - Same process with `RTDSC` instruction In this experiment, we measured the time to perform RSA-CRT with `RDTSC` instruction. Table 1 shows when the key size increases, the ratio of replay becomes higher. The decryption time computing rises with the key size because of noise since other processes penalize our attack.

Table 1 also shows that the Δ_0 for $q_i = 0$ depends on the modulus size. The delay between Montgomery multiplication with and without extra bit is increased by a factor of 2 for q's size 512 and 1024 bits. When we perform RSA-CRT with a 1024-bit (resp. 2048-bit) modulus, the size of the window in SWE is 5 (resp. 6). Then, the precomputation table phase of SWE requires 15 (resp 31) MM. In the experiments, a factor of four between Δ_0 for 1024 and 2048 bits modulus is expected.

Table 1: Results of our attack with `RDTSC`. We measure timing computation in the same process.

Modulus size	NS	ratio of replay	Δ_0	Δ_1	Number of query
512 bits	600	2%	2614	-90	78600
1024 bits	800	18%	5553	-134	241600
2048 bits	1000	50%	21855	-1743	768000

Experiment 2 - Same process with `RDMSR` instruction In this experiment we show, in table 2, that for the same size of neighbourhood, the noise due to other processes running in the computer do not interfere too much with our attack. When we use PMC for measuring ticks, we read time for our process core. Thus, processes running on other cores do not penalize the timing attack.

Table 2: Results of our attack with the RDMSR. We measure timing computation in the same process.

Modulus size	NS	ratio of replay	Δ_0	Δ_1	Number of query
512 bits	600	2%	2452	67	78600
1024 bits	800	10%	4554	-216	225600
2048 bits	1000	15%	22434	-992	589000

Experiment 3 - Inter-process with RTDSC instruction Table 3 shows that communication via inter-process does not reduce the effectiveness of our attack. The noise from inter-process is eliminated by increasing the size of neighbourhood, given similar Δ_0 and ratio of replay.

Table 3: Results of our attack with RDTSC. We measure timing computation in inter-process via TCP.

Modulus size	NS	ratio of replay	Δ_0	Δ_1	Number of query
512 bits	1000	2%	2537	-144	131000
1024 bits	1100	21%	5557	-1489	341000
2048 bits	1200	55%	28157	1042	952800

Experiment 4 - Inter-process with RDMSR instruction In this experiment we use processor affinity which allows us to choose the core/CPU where a process is running. Thus, the TCP client and server are carried out on different cores. Table 4 shows that the ratio of replay in inter-process attack is very low when we used RDMSR instruction.

Table 4: Results of our attack with RDMSR. We measure timing computation in inter-process via TCP.

Modulus size	NS	ratio of replay	Δ_0	Δ_1	Number of query
512 bits	1000	0%	2217	-1	128000
1024 bits	1100	5%	5067	-675	295900
2048 bits	1200	10%	17033	872	675600

When we used RDMSR in inter-process we were able to recover a 1024 bits key with an average of 215200 queries. These results are obtained with 800 neighbourhoods and a ratio of replay is around 5%.

5.3 Network attacks

Two ways are performed to measure timing decryption of RSA-CRT during inter-process attack. In [5], Boneh and Brumley show that it is enough to increase neighbourhood to convert an inter-process into a network timing attack. Thus using RTDSC instruction, we should be able to factorise N by measuring the time, from sending the ciphertext on the network and to receiving the response of the server.

Another way is also possible. Assume that an attacker is able to perform a spy process on the server, operating at a high level. The TCP client performs the ciphertext to be decrypted and sends the message. The spy sniffs TCP socket on the server and measure the time, using RDMSR instruction, taken by the server for answering. Once this is completed, the spy process sends the measurement to the client which is responsible for the decision of bit to recover. This scenario should have the same result as RDMSR in inter-process.

In the real world scenario the experimental data is limited. However, in PolarSSL's SSL implementation the key does not have timelife, only the SSL's session (lifetime is one day in PolarSSL's example). Our attack in inter-process with a 1024 key size takes about ten minutes. So, the attack presented in our paper seems to be feasible in the real world.

6 Defences

PolarSSL's library is vulnerable during our timing attack. In order to counteract it, we could make constant the time decryption of the RSA-CRT implementation. The attack results show that it is very complicated to obtain an implementation with those characteristics for any key size. In order to protect PolarSSL against this attack, three countermeasures can be developed in this section. The first one is used by OpenSSL and we suggested two others.

6.1 Blinding

This defence makes the time decryption of RSA-CRT independent on the input ciphertext. RSA-CRT blinding is implemented as follows.

Let a is a random value, e the RSA encryption exponent and c the ciphertext. To decrypt c :

- compute : $x = a^e.c \bmod N$,
- decrypt x : $RSA - CRT(x, d, p, q) = m' = a^{ed}c^d \bmod N$,
- compute : $\frac{m'}{a} \bmod N = a^{ed-1}c^d \bmod N = c^d \bmod N = m$.

Since a is random, then x is a random value. Then, the attacker can not choose the ciphertext being input to MM. This approach is preferred by Boneh and Brumley [5].

6.2 Alternatives to blinding

Our timing attack exploits the behaviour of the MM when an extra bit is carried out, others, when an extra-reduction occurs. Two strategies are possible for cancelling out extra bit : the first one uses particular modulus size whereas the other needs to modify PolarSSL's key generation routine.

In the following, we suppose that attacked library implements a dummy subtraction, such as PolarSSL, used to mask the timing effect of an extra-reduction without extra bit.

Use particular modulus size Colin D. Walter [9] demonstrated that if we choose $s' > s$ such as $2Q < r^{s'-1}$, $s^{'} \geq s+2$, then MM does not need extra-reduction. For cancelling out extra-reduction, large integer needs to be represented with $s+2$ words which is quite inefficient. We could make our timing attack impracticable with particular modulus size.

Suppose that $|Q| = kw+1$. Then, $s = \lceil \frac{<Q>}{w} \rceil = k+1$ and $R = r^s = r^{k+1}$ $= 2^{kw+w}$. We obtain :

$$2^{kw} < Q < 2^{kw+1} \text{ and } Q < \frac{1}{2^{w-1}} \times R \tag{4}$$

where $w \in \{8, 16, 32, 64\}$. According to lemma 1 extra-bit is cancelling out. Thus, our timing is defeated against's attacked library.

Countermeasure effectiveness is equivalent to blinding with a lower penalty. Penalty is 10% (resp. 6%) between 1026 and 1024 (resp. 2050 and 2048) modulus size .

Suppose that $|Q| = kw-1$, so $s = \lceil \frac{|Q|}{w} \rceil = k$. We obtain :

$$2^{kw-2} < Q < 2^{kw-1} \text{ and } \frac{1}{4} < \frac{Q}{R} < \frac{1}{2} \tag{5}$$

where $w \in \{8, 16, 32, 64\}$. According to lemma 1 extra-bit is cancelling out. Thus, our timing is defeated against attacked library.

Modify PolarSSL's key generation routine Another way to counteract timing attacks is to generate keys where primes factors are less than $\frac{\sqrt{5}-1}{2} \times R$. Then, according to lemma 1, extra-bit is cancelling out.

7 Conclusion

In this paper, we present a timing attack against PolarSSL - a protected SSL implementation of RSA-CRT. Our attack exploits an unknown arithmetical bias in Montgomery multiplication. In spite of this countermeasure, our experiments show that a timing attack is still possible using in inter-process for different modulus size. We also present a new way for measuring time decryption via performance monitor counters which improves the efficiency.

References

1. Paul Bakker. *PolarSSL project. Version 1.1.4.* `http://polarssl.org/download_overview?download=1.1.4` 2012-05-31
2. E. A. Young and T. J. Hudson. *OpenSSL project. Version 0.9.7.* `http://openssl.org`
3. Werner Schindler. *A timing attack against RSA with the chinese remainder theorem*, *In CHES 2000, pages 109-124*, 2000.
4. O. Aciiçmez, W. Schindler, and K. Kooç. *Improving Brumley and Boneh timing attack on unprotected SSL implementation.In V. Atluri, C. Meadows, and A. Juels, editors*, *ACM Conference on Computer and Communication Security, pages 139-146. ACM*, 2005.
5. David Brumley and Dan Boneh. *Remote timing attacks are practical*, *In In Proceedings of the* 12*th* *USENIX Security Symposium, pages 1-14*, 2003.
6. P. L. Montgomery. *Modular multiplication without trial division*, *Mathematics of Computations*, 44:519-521, 1995.
7. D. Coppersmith. *Small solutions to polynomial equations, and low exponent RSA vulnerabilities*, *Journal of Cryptology,* $10 : 233 - 260$, 1997.
8. Intel. *Intel 64 and IA-32 : Architectures Software Developer's Manual Combined Volumes 3A and 3B*, *System Programming Guide, Parts 1 and 2.*
9. Colin D. Walter. *Montgomery Exponentiation Needs no Final Subtractions*, *Electronics Letters, 35(21):1831-1832*, 1999.
10. Colin D. Walter. *Precise Bounds for Montgomery Multiplication and Some Potentially Insecure Moduli*, *Proceedings of CT-RSA 2002, LNCS 2271, pp. 30-39,Springer-Verlag*, 2002.
11. W. Schindler, Colin D. Walter. *More Detail for a Combined Timing and Power Attack against Implementations of RSA*, *in: Paterson, K.G. (ed.) Cryptography and Coding* 2003. *LNCS, vol. 2898, pp. 245-263. Springer, Heidelberg* 2003.

A Proof of Lemma 2

Proof. We assume that $Q > \frac{\sqrt{5}-1}{2} \times R$.
According to assumptions (1) and (3), we write P_{mul} as :

$$\begin{aligned} P_{mul} &= P \text{ (extra bit in } \textsc{montmul}(A, B, Q)) \\ &\simeq P(ABR^{-1} + Y > R) \\ &= \int_{\frac{(R-Q)R}{Q}}^{Q} \int_{\frac{(R-Q)R}{A}}^{Q} \int_{R-\frac{AB}{R}}^{Q} p(A, B, Y)\, \mathrm{dY}\, \mathrm{dB}\, \mathrm{dA} \quad , \end{aligned}$$

where Y is unformly distributed on $\mathbb{Z}_Q$ and $p(A, B, Y)$ is the probability density function for $A \times B \times Y$. According to assumptions (1) and (3) A, B and Y are independently distributed mod Q. Thus, $p(A, B, Y) = p(A) \times p(B) \times p(Y)$. As noted above, A, B and Y are uniform on $[0, Q)$.

Thus, $p(A,B,Y) = \frac{1}{Q^3}$. Then

$$\begin{aligned}
P_{mul} &\simeq \frac{1}{Q^3}\int_{\frac{(R-Q)R}{Q}}^{Q}\int_{\frac{(R-Q)R}{A}}^{Q}\int_{R-\frac{AB}{R}}^{Q} \mathrm{dY}\,\mathrm{dB}\,\mathrm{dA}\\
&= \frac{1}{Q^3}\int_{\frac{(R-Q)R}{Q}}^{Q}\int_{\frac{(R-Q)R}{A}}^{Q} Q-R+\frac{AB}{R}\ \mathrm{dB}\,\mathrm{dA}\\
&= \frac{1}{Q^3}\int_{\frac{(R-Q)R}{Q}}^{Q} \frac{AQ^2}{2R}+\frac{1}{A}\times\frac{1}{2}(R-Q)^2R-Q(R-Q)\ \mathrm{dA}\\
&= \frac{Q}{4R}+\frac{(R-Q)^2R}{Q^3}\times\left(\frac{3}{4}+\frac{1}{2}\log\left(\frac{Q^2}{(R-Q)R}\right)\right)-\frac{(R-Q)}{R}.
\end{aligned}$$

In the same way, the probability of extra bit in a squaring operation is :

$$\begin{aligned}
P_{square} = P\ (\text{extra bit in } \textsc{montmul}(A,A,Q)) &\simeq P(A^2R^{-1}+Y>R)\\
&= \int_{\sqrt{(R-Q)R}}^{Q}\int_{R-\frac{A^2}{R}}^{Q} p(A,Y)\,\mathrm{dY}\,\mathrm{dA}\\
&= \frac{1}{Q^2}\int_{\sqrt{(R-Q)R}}^{Q} Q-R+\frac{A^2}{R}\,\mathrm{dA}\\
&= \frac{Q}{3R}+\frac{2(R-Q)\sqrt{(R-Q)R}}{3Q^2}-\frac{(R-Q)}{R}.
\end{aligned}$$

For a fixed C with $\frac{(R-Q)R}{Q} < C < Q$, the probability of extra bit is :

$$\begin{aligned}
P_C = P\ (\text{extra bit in } \textsc{montmul}(A,C,Q)) &\simeq P(CAR^{-1}+Y>R)\\
&= \int_{\frac{(R-Q)R}{C}}^{Q}\int_{R-\frac{CA}{R}}^{Q} p(A,Y)\,\mathrm{dY}\,\mathrm{dA}\\
&= \frac{1}{Q^2}\int_{\frac{(R-Q)R}{C}}^{Q} Q-R+\frac{CA}{R}\,\mathrm{dA}\\
&= \frac{C}{2R}+\frac{(R-Q)^2R}{2CQ^2}-\frac{(R-Q)}{R}
\end{aligned}$$

□