Improving Brumley and Boneh Timing Attack on Unprotected SSL Implementations

Onur Acıiçmez
Oregon State University
Corvallis, USA
aciicmez@eecs.oregonstate.edu

Werner Schindler
Bundesamt für Sicherheit in der Informationstechnik
Bonn, Germany
Werner.Schindler@bsi.bund.de

Çetin K. Koç
Oregon State University
Corvallis, USA
koc@eecs.oregonstate.edu

ABSTRACT

Since the remarkable work of Kocher [7], several papers considering different types of timing attacks have been published. In 2003, Brumley and Boneh presented a timing attack on unprotected OpenSSL implementations [2]. In this paper, we improve the efficiency of their attack by a factor of more than 10. We exploit the timing behavior of Montgomery multiplications in the table initialization phase, which allows us to increase the number of multiplications that provide useful information to reveal one of the prime factors of RSA moduli. We also present other improvements, which can be applied to the attack in [2].

Categories and Subject Descriptors

E.3 [**Data Encryption**]: [Public key cryptosystems, Code breaking]

General Terms

Security

Keywords

side-channel cryptanalysis, timing attacks, RSA

1. INTRODUCTION

Several timing attacks have been developed against specific RSA implementations since the introduction of side channel cryptanalysis in [7]. For example, [7] and [5] describe timing attacks on RSA implementations which do not utilize Chinese Remainder Theorem (CRT). These attacks were generalized and optimized by advanced stochastic methods (cf. [9, 11, 12]). In particular, the efficiency of the attack from [5] could be increased by a factor of 50. Since these attacks cannot be applied to RSA implementations that use CRT, it had been thought for years that RSA-CRT was immune to timing attacks.

However, in [10], a new and efficient attack on RSA implementations that use CRT with Montgomery's multiplication algorithm was introduced. Under optimal conditions, it takes about 300 timing measurements to factorize 1024-bit RSA moduli. We note that these attacks can be prevented by blinding techniques (see [7], Sect. 10).

Typical targets of timing attacks are the security features in smart cards. Despite of Bleichenbacher's attack ([1]), which (e.g.) exploited weak implementations of the SSL handshake protocol, the vulnerability of RSA implementations running on servers was not known until Brumley and Boneh performed a timing attack over a local network in 2003 ([2]). They mimicked the attack introduced in [10] to show that RSA implementation of OpenSSL [14], which is the most widely used open source crypto library, was not immune to such attacks. Although blinding techniques for smart cards had already been 'folklore' for years, various crypto libraries that were used by SSL implementations did not apply these countermeasures at that time ([2]).

In this paper, we propose a timing attack, which is an improvement of [2] by a factor of more than 10. All of these timing attacks ([7, 5, 9, 10, 2]) including the one presented in this paper can be prevented by base blinding or exponent blinding. However, it is always desirable to understand the full risk potential of an attack in order to confirm the trustworthiness of existing or, if necessary, to develop more secure and efficient countermeasures and implementations.

Our attack exploits the peculiarity of the sliding windows exponentiation algorithm and, independently, suggests a general improvement of the decision strategy. Although it is difficult to compare the efficiency of attacks performed in different environments (cf. [2]), it is obvious that our new attack improves the efficiency of Brumley and Boneh's attack by a factor of more than 10.

Our paper is organized as follows: §2 gives brief review of Chinese Remainder Theorem and Montgomery multiplication algorithm. Mathematical background of the previous and the new attack is explained in §3, 4, and 5. In §6, implementation details are addressed, and §7 compares the experimental results with those from the attack in [2]. We conclude our paper in §8.

2. BACKGROUND

In this section, we explain some algorithms that are commonly used in RSA-CRT implementations. Our attack, as well as [2], exploits the timing characteristics of the algorithms described here.

2.1 Chinese Remainder Theorem (CRT)

Most of the RSA implementations use Chinese Remainder Theorem (CRT) to compute $y^d (\bmod n)$. CRT reduces the computation time by about 75%, compared to a straightforward exponentiation. We have $n = pq$ and use the notation $d_p = d(\bmod (p-1))$ and $d_q = d(\bmod (q-1))$. The algorithm is described in Figure 1 with suitable constants b_1 and b_2. We point out that Step 3 can be calculated more efficiently by $(x_q - x_p)(p^{-1}(\bmod q))p + x_p(\bmod n)$ (cf. [8]) which is yet not relevant for our purposes.

2.2 Montgomery Multiplication

Montgomery Multiplication (MM) is the most efficient algorithm to compute modular multiplications during a modular exponentiation. It uses additions and divisions by powers of 2, which can be accomplished by shifting the operand to the right, to calculate the result. Since it eliminates time consuming integer divisions, the efficiency of the algorithm is very high.

Montgomery Multiplication is used to calculate

$$Z = abR^{-1}\ (mod\ n),$$

where R is a constant power of 2, $R > n$, and R^{-1} is the inverse of R in modulo n. A conversion to and from n-residue format is required to use this algorithm. Hence, it is more attractive to use it for repeated multiplications on the same residue, just like modular exponentiations. Figure 2 shows the steps of Montgomery Multiplication Algorithm. The conditional subtraction $s - n$ is called 'extra reduction'.

Since the operand size of the arithmetic operations can simply be assumed to be constant during RSA exponentiation, the time required to perform integer operations in MM can also be assumed to depend only on the constants n and R but not on the operands a and b. This assumption is very reasonable for smart cards whereas software implementations may process small operands (i.e., those with leading zero-words) faster due to optimizations of the integer multiplication algorithm. In fact, this is the case for many SSL implementations which complicates the attack described in [2] and ours. (Both attacks are chosen-input attacks where small operands occur.) Under the simplifying assumption from above, we can conclude that $Time(MM(a, b; n)) \in \{c, c + c_{\text{ER}}\}$, where $Time(MM(a, b; n))$ is the execution time of the multiplication MM(a,b;n). The Montgomery operation requires a processing time of $c + c_{\text{ER}}$ iff the extra reduction has to be carried out.

Figure 3 explains how Montgomery's multiplication algorithm can be combined with arbitrary modular exponentiation algorithms to compute $y^d(\bmod n)$. Of course, in Phase 2a and 2b modular squarings and multiplications have to be replaced by the respective Montgomery operations.

3. GENERAL IDEA OF A TIMING ATTACK ON RSA-CRT

The different attacks [10] and [2] exploit the timing behavior of the Montgomery multiplications in Phase 2b of the modular exponentiation (cf. Figure 3). We can interpret the execution time of the i^{th} Montgomery operation in Phase 2b (squaring or a multiplication by a table value) as a realization of the random variable $c + W_i \cdot c_{\text{ER}}$ where $W_1, W_2, \ldots$ denotes a sequence of $\{0, 1\}$-valued random variables. The stochastic process $W_1, W_2, \ldots$ has been studied in detail in [9, 10, 12]. We merely mention that

$$E(W_i) = \begin{cases} \frac{1}{3}\frac{n}{R} & \text{for } MM(temp, temp; n) \\ \frac{1}{2}\frac{\bar{y}_j}{n}\frac{n}{R} & \text{for } MM(temp, \bar{y}_j; n). \end{cases} \quad (1)$$

where $\bar{y}_j$ and *temp* denote a particular table entry and an intermediate result during the exponentiation, respectively. '$E(\cdot)$' denotes the expectation of a random variable. The timing behavior of the Montgomery operations in Phase 2a) can similarly be described by a process $W'_1, W'_2, \ldots$.

When applying the CRT, (1) indicates that the probability of an extra reduction during a Montgomery multiplication of the intermediate result *temp* with $\bar{y}_{1;p} = yR(\bmod p)$ in Step 1 (resp. with $\bar{y}_{1;q} = yR(\bmod q)$ in Step 2) is linear in $\bar{y}_{1;p}/p$ (resp. linear in $\bar{y}_{1;q}/q$). Note that the message $(u * R^{-1})(\bmod n)$ corresponds to the value u during the exponentiation, because the messages are multiplied by R to convert them into Montgomery form. If the base of the exponentiation is $y := uR^{-1}(\bmod n)$, then $\bar{y}_{1;p} = yR \equiv u(\bmod p)$ and $\bar{y}_{1;q} = yR \equiv u(\bmod q)$. The same equation also implies that the same probability does not depend on y during the squarings.

For $0 < u_1 < u_2 < n$ with $u_2 - u_1 \ll p, q$, three cases are possible: The 'interval set' $\{u_1 + 1, \ldots, u_2\}$ contains no multiple of p or q (Case A), contains a multiple of p or q but not both (Case B), or contains multiples of both p and q (Case C). The running time for input $y := uR^{-1}(\bmod n)$, denoted by $T(u)$, is interpreted as a realization of a normally distributed random variable X_u.

If the square and multiply exponentiation algorithm is applied, the computation of x_p requires about $\log_2(n)/4$ multiplications with $\bar{y}_{1;p}$, and hence (1) implies

$$E(X_{u_2} - X_{u_1}) \approx \begin{cases} 0 & \text{for Case A} \\ -\frac{c_{\text{ER}}}{8}\frac{\sqrt{n}}{R}\log_2(n) & \text{for Case B} \\ -\frac{c_{\text{ER}}}{4}\frac{\sqrt{n}}{R}\log_2(n) & \text{for Case C} \end{cases}$$

This property allows us to devise a timing attack that factorizes the modulus n by exposing one of the prime factors,

```
Step 1:  a) $y_p := y(\bmod p)$
         b) $x_p := y_p^{d_p}(\bmod p)$
Step 2:  a) $y_q := y(\bmod q)$
         b) $x_q := y_q^{d_q}(\bmod q)$
Step 3:  Return $(b_1 x_p + b_2 x_q)(\bmod n)$
```

Figure 1: RSA with CRT

```
$s = a * b$
$s = (s - (s * n^{-1} \mathbf{mod}\, R) * n)/R$
if $s > n$ then $s = s - n$
return $s$     ( = MM(a,b;n) )
```

Figure 2: Montgomery Multiplication Algorithm

```
1.) $\bar{y}_1 := MM(y, R^2; n)$      (= yR (mod n))
2.) Modular Exponentiation Algorithm
      a) table initialization (if necessary)
      b) exponentiation phase
3.) Return MM(temp,1;n)   ( $=y^d$(mod n) )
```

Figure 3: Modular Exponentiation with Montgomery's Algorithm

e.g. q, bit by bit. We use the fact that if the interval $(u_1, u_2]$, i.e., the integers in $\{u_1+1,\ u_1+2,\ ...,\ u_2\}$, contains a multiple of q, i.e., in case of Case B or C, then T(u_1) - T(u_2) will be smaller than $c_{\mathrm{ER}} \log_2(n)\sqrt{n}/16R$. Let say the attacker already knows that q is in $(u_1, u_2]$ (after checking several intervals; = Phase 1 of the attack) and trying to reduce the search space. In Phase 2 the decision strategy becomes:

1. Split the interval into two equal parts: $(u_1, u_3]$ and $(u_3, u_2]$, where $u_3 = \lfloor (u_1+u_2)/2 \rfloor$. As usual, $\lfloor z \rfloor$ denotes the largest integer that is $\leq z$.

2. If $T(u_3) - T(u_2) < c_{\mathrm{ER}} \log_2(n)\sqrt{n}/16R$ decide that q is in $(u_3, u_2]$, otherwise in $(u_1, u_3]$.

3. Repeat the first steps until the final interval becomes small enough to factorize n using the Euclidean algorithm

At any time within Phase 2 the attacker can check whether her previous decisions have been correct. To confirm that an interval really contains q the attacker applies the decision rule to similar but different intervals, e.g., $(u_1+1, u_2-1]$, and confirms the interval if they yield the same decision.

In fact, it is sufficient to recover only the upper half of the bit representation of either p or q to factorize n by applying a lattice-based algorithm [4].

Under ideal conditions (no measurement errors) this attack requires about 300 time measurements to factorize a 1024-bit RSA modulus $n \approx 0.7 \cdot 2^{1024}$, if square and multiply algorithm is used. In Phase 2 of the attack, each decision essentially recovers one further bit of the binary representation of one prime factor. The details and analysis of this attack can be found in [10].

4. OVERVIEW OF BRUMLEY AND BONEH ATTACK

We explain the attack of [2], which will be refered as BB-attack from here on, and ours in the following two sections along with a discussion of the advantages of our attack over the other.

RSA implementation of OpenSSL employs Montgomery Multiplication, CRT, and Sliding Window Exponentiation (SWE) with a window size, denoted by $wsize$, of 5. SWE algorithm processes the exponent d by splitting it into odd windows of at most $wsize$ consecutive bits (i.e. in substrings of length $\leq wsize$ having odd binary representation), where the windows are not necessarily consecutive and may be separated by zero bits. It requires a preprocessing phase, i.e., table initialization, to compute odd powers of the base y so that many multiplications can be combined during the exponentiation phase.

The modulus n is 1024-bit number, which is the product of two 512-bit primes p and q. Considering one of these primes, say q, the computation of $y_q^{d_q} (\mathrm{mod}\, q)$ requires 511 Montgomery operations of type $MM(temp, temp; q)$ ('squarings') and approximately $(511 \cdot 31)/(5 \cdot 32) \approx 99$ multiplications with the table entries during the exponentiation phase of SWE (cf. Table 14.16 in [8]). Consequently, in average ≈ 6.2 multiplications are carried out with the table entry $\bar{y}_{1;q}$.

BB-Attack exploits the multiplications $MM(temp, \bar{y}_{1;q}; q)$ that are carried out in the exponentiation phase of SWE. Let assume that the attacker tries to recover $q = (q_0, ..., q_{511})$ and already obtained first, i.e. most significant, k bits. To guess q_k, the attacker generates g and g_{hi}, where $g =$ $(q_0, ..., q_{k-1}, 0, 0, ..., 0)$ and $g_{hi} = (q_0, ..., q_{k-1}, 1, 0, 0, ..., 0)$. Note that there are two possibilities for q: $g < q < g_{hi}$ (when $q_k = 0$) or $g < g_{hi} < q$ (when $q_k = 1$). She determines the decryption time $t_1 = T(g) = Time(u_g^d \bmod n)$ and $t_2 = T(g_{hi}) = Time(u_{g_{hi}}^d \bmod n)$, where $u_g = g * R^{-1} (\mathrm{mod}\, n)$ and $u_{g_{hi}} = g_{hi} * R^{-1} (\mathrm{mod}\, n)$. If q_k is 0, then $|t_1 - t_2|$ must be "large". Otherwise $|t_1 - t_2|$ must be close to zero, which implies that q_k is 1. The message u_g ($u_{g_{hi}}$ resp.) corresponds to the value g (g_{hi} resp.) during the exponentiations, because of the conversion into Montgomery form. BB-attack does not only compare the timings for $gR^{-1} (\mathrm{mod}\, n)$ and $g_{hi}R^{-1} (\mathrm{mod} n)$ but uses the whole neighborhoods of g and g_{hi}, i.e., $\mathcal{N}(g; N) = \{g, g+1, \ldots, g+N-1\}$ and $\mathcal{N}(g_{hi}; N) = \{g_{hi}, g_{hi}+1, \ldots, g_{hi}+N-1\}$, respectively. The parameter N is called the neighborhood size. For details, the reader is referred to [2].

5. DETAILS OF OUR APPROACH

Only about 6 from ca. 1254 many Montgomery operations performed in RSA exponentiation provide useful information for BB-attack. On the other hand, the table initialization phase of the exponentiation in modulo q requires 15 Montgomery multiplications with $\bar{y}_2$. Therefore, we exploit these operations in our attack. In fact, let $R05 = 2^{256} = \sqrt{R}$, the square root of R over the integers. Clearly, for input $y = u(R05)^{-1} (\mathrm{mod}\, n)$ (inverse in the ring Z_n) we have

$$\begin{aligned} \bar{y}_{2;q} &= MM(\bar{y}_1, \bar{y}_1; q) = u(R05)^{-1} u(R05)^{-1} R \\ &\equiv u^2 (\mathrm{mod}\, q). \end{aligned} \qquad (2)$$

Instead of $\mathcal{N}(g, N)$ and $\mathcal{N}(g_{hi}, N)$ we consequently consider the neighborhoods $\mathcal{N}(h; N) = \{h, h+1, \ldots, h+N-1\}$ and $\mathcal{N}(h_{hi}; N) = \{h_{hi}, h_{hi}+1, \ldots, h_{hi}+N-1\}$, resp., where

$$h = \lfloor \sqrt{g} \rfloor \text{ and } h_{hi} = \lfloor \sqrt{g_{hi}} \rfloor. \qquad (3)$$

To be precise, we consider input values $y = u(R05)^{-1}$(mod n) with $u \in \mathcal{N}(h; N)$ or $u \in \mathcal{N}(h_{hi}; N)$. Even if we just copied the other steps of BB-attack this would increase the efficiency by a factor of $\approx (15.0/6.2)^2 \approx 5.8$.

Under the assumption from Section 2.2, specifically

$$Time(MM(a, b; q)) \in \{c, c + c_{\mathrm{ER}}\}$$

for any $a, b \in Z_q$, we can simply replace the threshold value $\log_2(n)\, c_{\mathrm{ER}} \sqrt{n}/16R$ from Sect. 3 (square & multiply exponentiation algorithm) by $60\, c_{\mathrm{ER}} \sqrt{n}/16R$. Clearly, the absolute value of this new threshold is much smaller, which makes the attack less efficient in terms of the number of necessary measurements.

The situation in an actual attack is more complicated as pointed out in [2]. First of all, there are two different integer multiplication algorithms used to compute $MM(a, b; q)$: Karatsuba's algorithm (if a and b consist of the same number of words ($nwords$)) and the 'normal' multiplication algorithm (if a and b consist of different numbers of words ($nwords$, $mwords$)). Karatsuba's algorithm has a complexity of $O(nwords^{1.58})$, whereas the normal multiplication algorithm requires $O(nwords \cdot mwords)$ operations. Normally, the length of each input of Montgomery multiplication is 512 bits, therefore Karatsuba's algorithm is supposed to be applied during RSA exponentiation. However, BB-attack and ours are chosen-input attacks and some operands may be very small, e.g., $\bar{y}_{1;q}$ in BB-attack and $\bar{y}_{2;q}$ in our attack. Beginning with an index (denoting the actual exponent bit under attack) near the word size 32, the value of $\bar{y}_{1;q}$, resp. $\bar{y}_{2;q}$, has leading zero words so that the program applies normal multiplication.

Unfortunately, the effects of having almost no extra reduction for small table values but using less efficient integer multiplications counteract each other. Moreover, the execution time of integer multiplications becomes less and less during the course of the attack (normal multiplication algorithm!). It is worked out in [2] that the time differences of integer multiplications depend on the concrete environment, i.e., compiler options etc. Neither in [2] nor in this paper, we assume that the attacker knows all of these details. Instead, robust attack strategies that work for various settings are used in both cases.

BB-attack evaluates the absolute values

$$\begin{aligned}\Delta_{BB} \;=\; & \left| \sum_{j=0}^{N-1} Time\left((g+j)R^{-1}(\mathrm{mod}\, n)\right) - \right. \\ & \left. \sum_{j=0}^{N-1} Time\left((g_{hi}+j)R^{-1}(\mathrm{mod}\, n)\right) \right| . \end{aligned} \tag{4}$$

Δ_{BB} becomes 'small' when $d_k = 1$, whereas a 'large' value indicates that $d_k = 0$ [2]. Our pendant is

$$\begin{aligned}\Delta \;=\; & \sum_{j=0}^{N-1} Time\left((h+j)(R05)^{-1}(\mathrm{mod}\, n)\right) - \\ & \sum_{j=0}^{N-1} Time\left((h_{hi}+j)(R05)^{-1}(\mathrm{mod}\, n)\right) , \end{aligned} \tag{5}$$

where we omit the absolute value.

Since $(u+x)^2 - u^2 \approx 2x\sqrt{q}$ for $u \in \mathcal{N}(h, N)$ or $u \in \mathcal{N}(h_{hi}, N)$, the value Δ can only be used to retrieve the bits $i \leq 256 - 1 - \log_2(N)$. In fact, it is recommended to stop even at least two or three bits earlier. The remaining bits upto 256^{th} bit of q can be determined by either using the former equation or searching exhaustively.

Network traffic and other delays affect timing measurements, because we can only measure response times rather than mere encryption times. For that reason, identical input values are queried S many times, where S is one of the parameters in BB-attack, to decrease the effect of outliers in [2]. We drop this parameter in our attack, because increasing the number of different queries serves the same purpose as well.

If Δ_{BB} or $|\Delta|$ are 'large' (in relation to their neighborhood size N), that is, if $\Delta_{BB} > N \cdot th_{BB;i}$, resp. $|\Delta| > N \cdot th_i$ for suitable threshold values $th_{BB;i}$ and th_i (both depending on the index i) the attacker guesses $q_i = 0$, otherwise she decides for $q_i = 1$.

On the other hand, sequential sampling exploits the fact that already a fraction of both neighborhood values usually yields the correct decision with high probability. We can apply a particular decision rule not to sums of timings (i.e., to Δ) but successively to individual timing differences

$$\begin{aligned}\Delta_j \;=\; & Time\left((h+j)(R05)^{-1}(\mathrm{mod}\, n)\right) - \\ & Time\left((h_{hi}+j)(R05)^{-1}(\mathrm{mod}\, n)\right) \end{aligned} \tag{6}$$

for $j = 0, 1, \ldots, N_{max}$. The attacker proceeds until the difference

$$\#\{j \mid \tilde{q}_{i;j} = 0\} - \#\{j \mid \tilde{q}_{j;i} = 1\} \in \{m_1, m_2\}, \tag{7}$$

or a given maximum neighborhood size N_{max} is reached. The term $\tilde{q}_{i;j}$ denotes j^{th} individual decision for q_i, and the numbers $m_1 < 0$ and $m_2 > 0$ are chosen with regard to the concrete decision rule. If the process ends at m_1 (resp. at m_2) the attacker assumes that $q_i = 1$ (resp. that $q_i = 0$) is true. If the process terminates because the maximum neighborhood size has been exceeded the attacker's decision depends on the difference at that time and on the concrete individual decision rule (cf. [6], Chap. XIV, and [10], Sect. 7).

The fact that the distribution of the differences varies in the course of the attack causes another difficulty. As pointed out earlier we do not assume that the attacker has full knowledge on the implementation details and hence not full control on the changes of the distribution. A possible individual decision rule could be, for instance, whether the absolute value of an individual time difference exceeds a particular bound th_i ($\rightarrow$ decision $\tilde{q}_{i;j} = 0$). The attacker updates this threshold value whenever he assumes that a current bit q_i equals 1. The new threshold value depends on the old one and the actual normalized value $|\Delta|/N_e$ where N_e denotes the number of exploited individual timing differences.

In Section 4 we use an alternative decision strategy that is closely related to this approach. For $k \in \{0, 1\}$ we define

$$f_{i;\geq;k} = \mathrm{Prob}(\Delta_j \geq 0 \mid q_i = k), \tag{8}$$

and similarly

$$f_{i;<;k} = \mathrm{Prob}(\Delta_j < 0 \mid q_i = k). \tag{9}$$

We want to mention that the following equation surely holds due to the reasons given above.

$$\max\{f_{i;\geq;0}, f_{i;<;0}\} > \max\{f_{i;\geq,1}, f_{i;<,1}\}. \tag{10}$$

The right-hand maximum should be close to 0.5. We subtract the number of negative timing differences from the number of non-negative ones. The process terminates when this difference equals $m_1 = -D$ or $m_2 = +D > 0$, or until a particular maximum neighborhood size N_{max} is reached. For $N_{max} = \infty$, the process will always terminate at either $-D$ or D. However, the average number of steps should be smaller when $q_i = 0$, because of the fact highlighted in equation (10). Consequently, if D and N_{max} are chosen properly, a termination at D or $-D$ is a strong indicator for $q_i = 0$, whereas reaching N_{max} without termination points that $q_i = 1$. We use this strategy in our implementation and

Operating System:	RedHat workstation 3
CPU:	dual 3.06Ghz Xeon
Compiler:	gcc version 3.2.3
Cryptographic Library:	OpenSSL 0.9.7e

Table 1: The configuration used in the experiments

the results are presented in §7. We interpret our decision procedure as a classical gambler's ruin problem. Formula (11) below facilitates the selection of suitable parameters D and N_{max}. If $f_{i;\geq;k} \neq 0.5$ formula (3.4) in [6] (Chap. XIV Sect. 3 with $z = D$, $a = 2D$, $p = f_{i;\geq;k}$ and $q = 1 - p$) yields the average number of steps (i.e., number of time differences to evaluate) until the process terminates at $-D$ or D assuming $N_{max} = \infty$. In fact,

$$E(Steps) = \frac{D}{f_{i;<;k} - f_{i;\geq;k}} - \frac{2D}{f_{i;<;k} - f_{i;\geq;k}} \frac{1 - (\frac{f_{i;<;k}}{f_{i;\geq;k}})^D}{1 - (\frac{f_{i;<;k}}{f_{i;\geq;k}})^{2D}} \quad (11)$$

Similarly, formula (3.5) in [6] yields

$$E(Steps) = D^2 \text{ if } f_{i;\geq;k} = 0.5. \quad (12)$$

These formulae can be used to choose the parameter D and N_{max} (cf. Sect. 7). A deeper analysis of the gambler's ruin problem can be found in [6], Sect. XIV.

The probabilities $f_{i;\geq;k}$ vary with i and this fact makes the situation more complicated. On the other hand, if D and N_{max} are chosen appropriately, the decision procedure should be robust against small changes of these probabilities.

6. IMPLEMENTATION DETAILS

We performed our attack against OpenSSL version 0.9.7e with disabled blinding, which would prevent the attack [14]. We implemented a simple TCP server and a client program, which exchange ASCII strings during the attack. The server reads the strings sent by the client, converts them to OpenSSL's internal representation, and sends a response after decrypting them. The attack is actually performed by the client, which calculates the values to be decrypted, prepares and sends the messages, and makes guesses based on the time spent between sending a message and receiving the response.

We used GNU Multi Precision arithmetic library, shortly GMP, to compute the square roots, i.e., $\lfloor\sqrt{g}\rfloor$ and $\lfloor\sqrt{g_{hi}}\rfloor$ [13]. The source code was compiled using the gcc compiler with default optimizations. All of the experiments were run under the configuration shown in Table 1. We used random keys generated by OpenSSL's key generation routine. We measured the time in terms of clock cycles using the Pentium cycle counter, which gives a resolution of 3.06 billion cycles per second. We used the "rdtsc" instruction available in Pentium processors to read the cycle counter and the "cpuid" instruction to serialize the processor. Serialization of the processor was employed for the prevention of out-of-order execution in order to obtain more reliable timings. Serialization was also used by Brumley and Boneh in their experiments [2].

There are 2 parameters that determine the total number of queries required to expose a single bit of q.

- Neighborhood size N_{max}: We measure the decryption time in the neighborhoods of $\mathcal{N}(h; N_{max}) = \{h, h+1, \ldots, h + N_{max} - 1\}$ and $\mathcal{N}(h_{hi}; N_{max}) = \{h_{hi}, h_{hi}+1, \ldots, h_{hi}+N_{max}-1\}$ for each bit of q we want to expose.
- Target difference D: The difference between the number of time differences that are less than zero and the number of time differences that are larger then zero. If we reach this difference among N_{max} many timings, we guess the value of the bit as 0. Otherwise, our guess becomes as $q_i = 1$.

The total number of queries and the probability of an error for a single guess depend on these parameters. The sample size used by [2] is no longer a parameter in our attack. In the following section, we present the results of the experiments that explore the optimal values for these parameters.

In our attack, we try to expose all of the bits of q between 5^{th} and 245^{th} bits. The first few bits are assumed to be able to determined by the same way as in [2]. The remaining 11 bits after 245^{th} bit can be easily found by using either an exhaustive search or BB-attack itself.

7. EXPERIMENTAL RESULTS

In this section we present the results of our experiments in four subsections. First, we compare our attack to BB-attack. Then, we give the details of our attack including error probability, parameters and the success rate in the following subsections. We also show the distribution of the time differences, which is the base point of our decision strategy.

The characteristics of the decryption time may vary during the course of the attack, especially around the multiples of the machine word size. Therefore, we separated the bits of q into different groups, which we call intervals. The interval [i,j] represents all the bits between i^{th} and j^{th} bit, inclusively. In our experiments, we used intervals of 32 bits: [32,63], [64, 95], ...etc.

All of the results we present in this paper were obtained by running our attack as an inter-process attack. It is stated in [2] that it is sufficient to increase the sample size to convert an inter-process attack into an inter-network attack. In our case, either a sample size can be used as a third parameter or the neighborhood size and the target difference can be adjusted to tolerate the network noise.

7.1 Comparison of our attack and BB-attack

In [2], Brumley and Boneh calculated the time differences, denoted by Δ_{BB}, for each bit to use as an indicator for the value of the bit. The gap between Δ_{BB} when q_i is 0 and when it is 1 is called the zero-one gap in [2]. Therefore, we want to compare both attacks in terms of zero-one gap. We run both attacks on 10 different randomly chosen keys and collected the time differences for each bit in $[5, 245]$ using a neighborhood size of 5000 and a sample size of 1. Table 2 shows the average statistics of the collected values. The zero-one gap is 114% larger in our attack, which means a smaller number of queries are required to deduce a key in ours.

7.2 The details of our attack

Our decision strategy for each single bit consists of:

- Step 1: Sending the query for a particular neighbor and measuring the time difference Δ_j.

	new attack			BB-Attack		
	$\|\Delta\|/N$			Δ_{BB}/N		
interval	bits = 0	bits = 1	0-1 gap	bits = 0	bits = 1	0-1 gap
$[5, 31]$	5871	3744	2127	3423	2593	830
$[32, 63]$	42778	4003	38775	15146	3455	11691
$[64, 95]$	40572	4310	36263	15899	3272	12627
$[96, 127]$	41307	3995	37313	18886	3580	15306
$[128, 159]$	45168	2736	42431	20877	2933	17945
$[160, 191]$	44736	3082	41654	24513	2479	22034
$[192, 223]$	37141	1755	35385	21550	1977	19573
$[224, 245]$	21936	2565	19371	27702	4728	22974

Table 2: Average Δ and Δ_{BB} values and 0-1 gaps. The values are given in terms of clock cycles.

- Step 2: Comparing Δ_j with zero and updating difference between the number of Δ_j values that are less than zero and the number of Δ_j values that are larger than zero.
- Step 3: Repeating first 2 steps until we reach the target difference, D, or a maximum of N_{max} times.
- Step 4: Making the guess $q_i = 0$, if the target difference is reached. Otherwise the guess turns out to be $q_i = 1$.

Note that we normally send only one query in Step 1, although we use a difference of two timings in our decision. This is because one of the timings we use to compute the difference has to be the one used for the decision of the previous bit. Since we halve the interval, which q is in, in each decision step, only one of the bounds, either the upper or lower one, will change. The timings for the bound that does not change can be reused during the decision process of the next bit. Therefore sending one query for a particular neighbor becomes sufficient by storing the data of the previous iteration. Of course, there are some cases that we have to send both queries, specifically when we exceed the total number of neighbors used in the previous decision step. However, just removing the redundant queries, which can also simply be applied to BB-attack, almost doubles the performance.

7.2.1 The distribution of time differences

We use the distribution of the time differences for our decision purposes. Whenever $q_i = 1$, the number of time differences lay above and below zero is very close to each other. However, when $q_i = 0$, the difference between these numbers becomes larger(see Figure 4).

7.2.2 Error probabilities and the parameters

When $q_i = 1$, approximately half of the time differences become positive and the other half become negative. If q_i is 0, the majority of the time differences becomes either positive or negative. We determined the percentage of that majority in order to calculate the error probability for a single time difference. Table 3 shows estimators for $\max\{f_{i;\geq;0}, f_{i;<;0}\}$ and $\max\{f_{i;\geq,1}, f_{i;<,1}\}$. These statistics were obtained using 10 different keys and a neighborhood size of 50000 for $[5, 31]$ and 5000 for other intervals.

The empirical parameters that yield the intended error probabilities are shown in Table 4. We present three different sets of parameters for each accuracy of 95%, 97.5%, and 99%. We used these parameters to perform our attack on several different keys. Note that inserting the values of Table 3 into formula (11) yields the expected values $E(Steps)$ for $q_i = 0$ and $q_i = 1$, resp. The probabilities for correct guesses (95%, 97.5%, 99%) were gained empirically.

interval	$\max\{f_{i;\geq;0}, f_{i;<;0}\}$	$\max\{f_{i;\geq,1}, f_{i;<,1}\}$
$[5, 31]$	0.5315	0.5040
$[32, 63]$	0.6980	0.5097
$[64, 95]$	0.7123	0.5085
$[96, 127]$	0.7079	0.5079
$[128, 159]$	0.7300	0.5080
$[160, 191]$	0.7349	0.5090
$[192, 223]$	0.6961	0.5077
$[224, 245]$	0.6431	0.5194

Table 3: The percentage of the majority of time differences that are either positive or negative (empirical values)

We employed the concept of 'confirmed intervals' (refer to Section 3) to detect the errors occured during the attack. We could recover such errors using the same concept and could expose each bit of q in the interval $[5, 245]$ of any key we attacked. Brumley and Boneh used 1.4 million queries in [2] (interprocess attacks) and they indicated that their attack required nearly 359000 queries in the more favourable case when the optimizations were turned off by the flag (-g). We could perform our attack with as low as 47674 queries for a particular key. The performance of these timing attacks are highly environment dependent, therefore it is not reliable to compare the figures of two different attacks on two different systems. Despite this fact, it is obvious by the arguments explained above (improving the signal-to-noise ratio (cf. also Table 2), reusing previous queries, sequential sampling) that our attack is significantly better than the previous one.

We performed interprocess attacks. Clearly, in network attacks the noise (caused by network delay times) may be much larger, and hence an attack may become impractical even if it is feasible for an interprocess attack under the same environmental conditions. However, this aspect is not specific for our improved variant but a general feature that affects BB-attack as well.

8. CONCLUSION

We have presented a new timing attack against unprotected SSL implementations of RSA-CRT. Our attack ex-

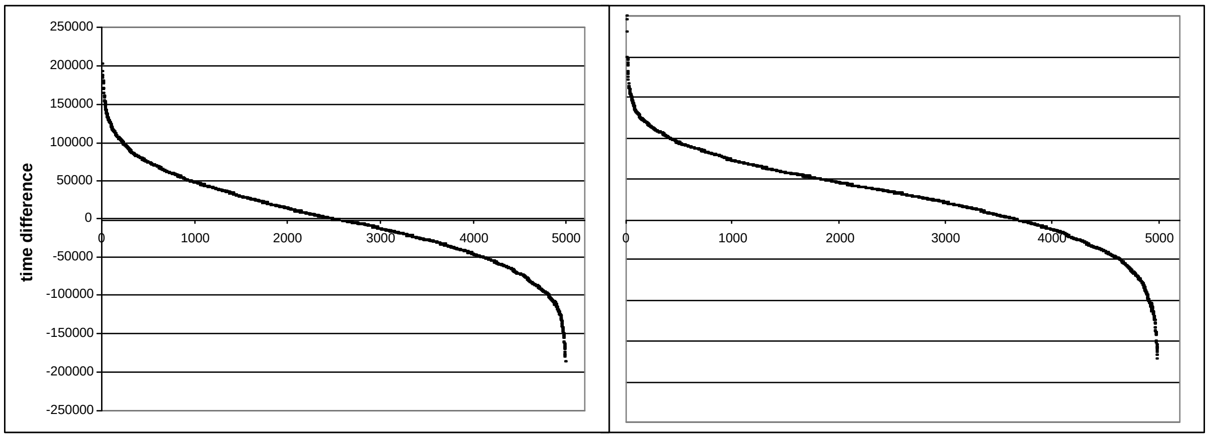

Figure 4: The distribution of Δ_j in terms of clock cycles for $0 \leq j \leq 5000$, sorted in descending order, for the sample bit q_{61}. The graph on the left shows this distribution when $q_{61} = 1$. The distribution on the right is observed when $q_{61} = 0$.

	Accuracy = 95%				Accuracy = 97.5%				Accuracy = 99%			
	parameters		E(Steps) for		parameters		E(Steps) for		parameters		E(Steps) for	
interval	D	N_{max}	$q_i = 0$	$q_i = 1$	D	N_{max}	$q_i = 0$	$q_i = 1$	D	N_{max}	$q_i = 0$	$q_i = 1$
$[5, 31]$	63	1850	998	3667	68	1975	1077	4220	230	6720	3646	27480
$[32, 63]$	25	131	63	579	29	163	73	761	34	240	85	1012
$[64, 95]$	17	67	40	281	36	192	84	1154	46	450	108	1767
$[96, 127]$	18	70	43	315	26	130	62	640	44	250	105	1674
$[128, 159]$	16	50	34	250	31	271	67	889	41	299	89	1477
$[160, 191]$	21	107	44	421	25	127	53	585	29	169	61	771
$[192, 223]$	24	126	61	551	36	264	91	1179	49	333	124	2033
$[224, 245]$	30	230	104	636	31	259	108	667	43	365	150	1032

Table 4: Columns 2 and 3 show the parameters that can be used to yield the intended accuracy. The last columns give the expected number of steps for $N_{max} = \infty$, calculated using Formula (11), to reach the target difference D.

ploits the timing behavior of Montgomery multiplications performed during *table initialization phase* of the sliding window exponentiation algorithm. It is an improvement of Brumley and Boneh attack, which exploits Montgomery multiplication in the *exponentiation phase* of the same algorithm. Changing the target phase of the attack yields an increase on the number of multiplications that provide useful information to expose one of the prime factors of RSA moduli. Only this change alone gives an improvement by a factor of more than 5 over BB-attack.

We have also presented other possible improvements, including employing sequential analysis for the decision purposes and removing the redundant queries that can also be applied to BB-attack. If we use only the idea of removing redundant queries from BB-attack, this will double the performance by itself. Our attack brings an overall improvement by a factor of more than 10.

9. REFERENCES

[1] D. Bleichenbacher: Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1. In: H. Krawczyk (Ed.): Crypto 1998, Springer, Lecture Notes in Computer Science **1462**, 1998, 1–12.

[2] D. Brumley, D. Boneh: Remote Timing Attacks are Practical. In: Proceedings of the 12th Usenix Security Symposium, 2003.

[3] B. Canvel, A. Hiltgen, S. Vaudenay, M. Vuagnoux: Password Interception in a SSL/TSL Channel. In: D. Boneh (ed.): Crypto 2003, Lecture Notes in Computer Science **2729**, Springer, Heidelberg (2003), 583–599.

[4] D. Coppersmith: Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. J. Cryptology **10** (no. 4) (1997) 233–260.

[5] J.-F. Dhem, F. Koeune, P.-A. Leroux, P.-A. Mestré, J.-J. Quisquater, J.-L. Willems: A Practical Implementation of the Timing Attack. In: J.-J. Quisquater and B. Schneier (eds.): Smart Card – Research and Applications, Springer, Lecture Notes in Computer Science **1820**, Berlin (2000), 175–191.

[6] W. Feller: Introduction to Probability Theory and Its Applications (Vol. 1). 3^{rd} edition, revised printing, New York, Wiley (1970).

[7] P. Kocher: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS and Other Systems. In: N. Koblitz (ed.): Crypto 1996, Springer, Lecture Notes in Computer Science **1109**, Heidelberg (1996), 104–113.

[8] A.J. Menezes, P.C. van Oorschot, S.C. Vanstone: Handbook of Applied Cryptography, Boca Raton, CRC Press (1997).

[9] W. Schindler: Optimized Timing Attacks against Public Key Cryptosystems. Statist. Decisions **20** (2002), 191–210.

[10] W. Schindler: A Timing Attack against RSA with the Chinese Remainder Theorem. In: Ç.K. Koç, C. Paar (eds.): Cryptographic Hardware and Embedded Systems — CHES 2000, Springer, Lecture Notes in Computer Science **1965**, Berlin (2000), 110–125.

[11] W. Schindler, F. Koeune, J.-J. Quisquater: Improving Divide and Conquer Attacks Against Cryptosystems by Better Error Detection / Correction Strategies. In: B. Honary (ed.): Cryptography and Coding — IMA 2001, Springer, Lecture Notes in Computer Science **2260**, Berlin (2001), 245–267.

[12] W. Schindler: On the Optimization of Side-Channel Attacks by Advanced Stochastic Methods. In: S. Vaudenay (ed.): Public Key Cryptography — PKC 2005, Springer, Lecture Notes in Computer Science **3386**, Berlin 2005, 85–103.

[13] GNU Project: GMP: `http://www.swox.com/gmp/`.

[14] OpenSSL Project: OpenSSL: `http://www.openssl.org`.