A Timing Attack on Blakley's Modular Multiplication Algorithm, and Applications to DSA

Bahador Bakhshi and Babak Sadeghiyan

Computer Engineering and Information Technology Department
Amirkabir University of Technology, Tehran, Iran
bbakhshi@aut.ac.ir, basadegh@ce.aut.ac.ir

Abstract. In this paper, we introduce a timing attack scheme against a 160-bit modular multiplication with Blakley's algorithm. It is assumed that a set of public inputs are multiplied by a secret parameter and running time of each multiplication is given, but the multiplication result is not known and a machine similar to victim machine isn't available. The proposed attack extracts all 160 bits of the secret parameter. Running time of Blakley's algorithm is analyzed and it is shown that running time of each step is dependent on the running time of other steps. The dependencies make the parameters of the attack be dependent on the secret key, while it makes the attack rather complicated. A heuristic algorithm is used to find the parameters of the attack. As a real scenario, the attack is applied against on-line implementation of Digital Signature Algorithm, which employs Blakley's modular multiplication. Practical results show that secret key of DSA will be found using 1,000,000 timing samples.

Keywords: timing attack, modular multiplication, Blakley's algorithm, DSA.

1 Introduction

Any cryptographic primitive, such as a digital signature, can be considered in two different aspects. It can be viewed as an abstract mathematical function that takes some inputs and produces outputs. Alternatively, it can be viewed as an implementation of a mathematical function in the real-world software/hardware system. In the latter view, the cryptographic system interacts with environment through side channels, such as power consumption channel and execution time channel. Side channel attacks use the leaked data from the side channels to attack on a certain cryptographic system, while make some assumptions about the implementation. Among different side channel attacks, the timing attack has special feature, i.e. very limited equipment is required to gather timing data.

Idea of timing attack was first introduced publicly by Kocher in [10]. He showed that difference between the required execution times for various inputs, can be exploited in order to find secret parameters of the underlying system.

J. Katz and M. Yung (Eds.): ACNS 2007, LNCS 4521, pp. 129–140, 2007.

Kocher mentioned some systems, which maybe vulnerable to timing attack, including RSA and DSA. He also showed how RSA implementation, which employs square-and-multiply algorithm to implement modular exponentiation is vulnerable to timing attack. Dhem et al. described a practical timing attack on an RSA implementation [4]. These attack schemes are not applicable to CRT based implementations of RSA. In [16], Schindler proposed another timing attack scheme on RSA implementation based on CRT method and Montgomery multiplication. Contrary to Kocher's and Dhem's attacks, Schindler's attack does not directly find the secret key. His attack factorizes RSA-modulus, instead. Schindler also introduced advance statistical and stochastic method to model and optimize timing attack in [17,18]. Brumley and Boneh employed and improved Schindler's idea to attack remotely on the RSA implementation which is used in OpenSSL library [2]. They showed that not only smart cards but also general purpose applications such as cryptographic operations in network communication and operating systems are vulnerable to timing attack. All these attacks are based on timing vulnerability of modular exponentiation, in which the exponent and modulus are constant, but a different base is used in each exponentiation.

Another timing attack, based on the vulnerability of modular exponentiation, was introduced against GPS identification system in [3]. Differing from previous attacks, the exponent isn't constant in this attack. GPS uses two secret keys, short-term key which is generated in each execution and a long-term key that is permanent for each user. The short-term key is used in a modular exponentiation and in a modular addition with the long-term key. In the attack, timing samples are used to find out only the hamming weight of the short-term key. Using inputs and outputs of GPS algorithm and the hamming weight, bits of the long-term key are guessed.

In addition to RSA and GPS, several block ciphers have been also examined under timing attack such as DES [8], RC5 [7] , Rijndael [11] . Timing analysis has been also applied on web privacy [5], in which a malicious web site can determine, using response of browser to the request, whether or not the user has recently visited some other, unrelated web page. The attack on web privacy was formally modeled by Focardi et al. [6]. Timing attack may also be combined with other side channel attacks in order to improve its performance and efficiency, such hybrid attacks were introduced in [14,15].

Modular multiplication is used in some cryptographic algorithms, such as public key encryption and digital signature. In such cases, a secret parameter of system is maybe multiplied by a public value. By default, modular multiplication is not an NP problem, if attacker knows the public input and the multiplication result, he simply finds out the secret parameter. But in some cryptographic algorithms, such as ElGamal Signature and Digital Signature Algorithm, the multiplication result isn't known value; it is kept secret [13].

To our best knowledge, until now almost all known timing attacks, except timing attacks against block ciphers, are based on time measurement of a modular exponentiation. In this paper we present new timing attack on modular multiplication. We use a technique like Dhem's technique, which is used to attack on

RSA, and propose practical timing attack on Blakley's modular multiplication algorithm. In the proposed attack, it is assumed that Blakley's modular multiplication is used to compute $a.b \bmod q$, attacker knows b, q, and running time of algorithm but he is unaware of the multiplication result. The timing attack finds the parameter a. Running time of each step of Blakley's algorithm is not independent of other steps, so correlation between running time of steps will be considered in the attack. The attack is applied on DSA's on-line signature generation phase. Experimental results show that 1,000,000 time measurements are sufficient to find 160 bits of the secret key.

This paper is organized as follows. Blakley's modular multiplication algorithm and its application in DSA's signature generation phase are described in section 2. The proposed timing attack is explained in section 3. Running time of Blakley's algorithm and inter-steps dependencies are also discussed in section 3. Section 4 introduces a heuristic algorithm to solve some involved problems with attack. Practical results of applying the attack on an on-line implementation of DSA are presented in section 5 and section 6 concludes this paper.

2 Blakley's Modular Multiplication Algorithm

Blakley or interleaved algorithm is one of the algorithms are deployed in implementing modular multiplication [1]. Blakley's algorithm interleaves multiplication and modular reduction. At each step of multiplication, intermediate results are reduced to desired modulus q. If all numbers are t bits, the algorithm is as following:

Step 3 is like left shift in multiplication. If current bit of operand a, a_j, is 1, operand b is added to partial product, p. Steps 4 and 5 as well as steps 8 and 9 reduce partial product to modulo q. After step 3 or 7, there is always $p < 2q$, so one subtraction, in steps 4 or 8, is sufficient to reduce partial product to modulo q.

In the remainder of this paper following definitions are used:

Addition-1: The modular addition is done in steps 3, 4, and 5 to compute $p = p + p \bmod q$.

Addition-2: The modular addition is done in steps 7, 8, and 9 to compute $p = p + b \bmod q$.

$Er_{1,i}$: When the condition of step 5 in round $j = i$ is true, an extra assignment is done, this is called as occurrence of $Er_{1,i}$.

$Er_{2,i}$: When the condition of step 9 in round $j = i$ is true, an extra assignment is done, this is called as occurrence of $Er_{2,i}$.

Er : Either $Er_{1,i}$ or $Er_{2,i}$.

$p_{1,i}$: Value of p, which is used in the right hand side of Addition-1 in round $j = i$.

$p_{2,i}$: Value of p, which is used in the right hand side of Addition-2 in round $j = i$.

Blakley's algorithm has a few conditional statements that cause running time of the algorithm be dependent on the input values. Hence, it is vulnerable to

timing attack. In our attack scheme, it is assumed that attacker knows b, q and can measure running time of the algorithm, but he does not know the multiplication result. The input parameter a is the secret parameter. Such situation exist in signature generation phase of DSA. DSA signs a hash of message M, $h(M)$, and produce two values r and s as following:

- $r = (g^k \bmod p) \bmod q$.
- $s = k^{-1}.(h(M) + x.r) \bmod q$.

Where g, p and q are public parameters, k is a random number which is generated for each message, and x is the long-term key, i.e. signer's permanent secret key. In order to improve performance and reduce bit size of intermediate results, it is preferred that compute the output s in the following steps:

1. $z = x.r \bmod q$.
2. $z' = h(M) + z \bmod q$.
3. $s = k^{-1}.z' \bmod q$.

It is assumed that modular multiplications is implemented using Blakley's algorithm instead of Montgomery multiplication. As, Montgomery algorithm is not suitable for a single modular multiplication. Suppose a, b, and q are t-digit integers with $0 < a, b < q$. Montgomery algorithm requires a precomputation on inputs, a, b. Neglecting the cost of the precomputation on the input, Montgomery multiplication algorithm computes $a.b.R^{-1} \bmod q$, where R is a constant parameter of the algorithm, while the result is obtained through $2t(t+1)$ single-precision multiplications. The computation of $a.b \bmod q$ is done in $4t(t+1)$ single-precision operations through the application of Montgomery multiplication to $a.b.R^{-1} \bmod q$ and $R^2 \bmod q$. Using classical modular multiplication (Multiplication $a.b$ then division by q) would require $2t(t+1)$ single-precision operations and no precomputation. Hence, the classical algorithm is superior for doing a single modular multiplication [12]. But Blakley's algorithm is more computational effective that classic modular multiplication, because it requires less memory to store intermediate results and it does not use the complicated division operation. In addition to using Blakley's algorithm, it is assumed that the secret key x is passed as parameter a to Blakley's algorithm. With these assumptions, the conditions of the timing attack are met, i.e. the attacker knows r and q, while z is kept as a private intermediate result. But attacker can not measure the running time of modular multiplication in step 1 directly.

There are two general implementations of DSA: on-line and off-line. In on-line implementation, both r and s are computed when there is signing request. But in off-line implementation, r is computed regardless of a signing request and the computed (r, k^{-1}) is stored. When there is a signing request, only s will be computed using $h(M)$ and a stored (r, k^{-1}). In this paper, the on-line implementation of DSA are attacked. In this case an attacker measure the running time for computing r, z, z', and s altogether. The execution time of r, s, and z' will be considered as noise included in a measured timing data. Attacker can compensate for the noise by increasing the number of measurements.

3 Timing Attack

Our approaches for guessing bits likes the approach which Dhem et al. used to attack on RSA, modular exponentiation [4]. In which, to guess a bit of secret key; it is assumed that the bit is one and algorithm is simulated on attacker's machine using gathered inputs from the under attack machine. According to a feature of implementation, extra reduction in Montgomery algorithm, the input set is divided into two subsets. Statistics of the running time of these two subsets are obtained. According to an oracle function, which is defined on the statistics, the bit of the secret key is guessed. In our attack, occurrence of the $Er_{2,i}$ is used as implementation feature. The statistic is the average running time and the oracle function uses difference between the averages. With the following assumptions, attacker uses "Timing Attack" algorithm to guess bit a_i.

- Attacker collects a set of inputs, which are used for a parameter b of Blakley's algorithm, $B = \{b_{(1)}, b_{(2)}, \ldots, b_{(n)}\}$.
- Attacker measures running timing of the algorithm for each input, $T = \{T_1, T_2, \ldots, T_n\}$, where T_i is the running time of the algorithm for input $b_{(i)}$.
- Attacker knows a few most significant bits of the parameter a, i.e. $a' =< a_{t-1}a_{t-2}a_{t-3}\ldots a_{i+1} >$, where $< a_i a_j >$ denotes concatenation of the a_i and a_j bits.

ALGORITHM Timing Attack
INPUT: B, T, q, $\overline{d}$, and a'
OUTPUT: a_i
1. Generate a temporary key, $a'' =< a'1 >$
2. Run Blakley's algorithm from $j = t-1$ to $j = i$ using a'' for each $b_{(k)} \in B$.
3. According to the occurrence of $Er_{2,i}$, T is divided into two sets, T'_0 and T'_1:
   $T'_0 = \{T_j \in T | Er_{2,i}$ does not occur$\}$
   $T'_1 = \{T_j \in T | Er_{2,i}$ occurs$\}$
4. Compute average of T'_0 and T'_1, which are named $\overline{T_0}$ and $\overline{T_1}$ respectively.
5. Find difference between the average of times: $d = \overline{T_1} - \overline{T_0}$.
6. If $d > \overline{d}$, a_i is guessed as zero,
   else a_i is guessed as one.

Note that step 2 of the "Timing Attack" algorithm, simulating Blakley's Algorithm using gathered data from victim's machine, is done on attacker's machine. The variable d is the statistic of attack, the variable can be either of two random variables d_0 or d_1. When the bit a_i is actually one, d is equal to d_1, and when the bit is zero, d is equal to d_0. The "Timing Attack" algorithm will guess bit a_i correctly, if distributions of the random variables d_0 and d_1 are not overlapped, in this case the input $\overline{d}$ is the border between distribution of d_0 and d_1. In step 6, it is decided whether d equals d_0 through comparing it with $\overline{d}$, and a guess on a_i is given according to the result. Attacker repeats this algorithm to find out the subsequent bits.

Correctness of "Timing Attack" depends upon following claims:

Claim 1: Distributions of the random variables d_0 and d_1 are not overlapped, therefore we can find the border, $\overline{d}$.

Claim 2: $\forall d' \in d_0$ and $\forall d'' \in d_1$ we have $d' > d''$, so the step 6 guesses the a_i correctly.

In following sections, we investigate the claim 1 and 2 and propose an algorithm to find the $\overline{d}$.

3.1 Running Time of Blakley's Algorithm

This subsection elaborates the claim 1 and 2. In order to investigate the claims, distributions of d_0 and d_1 should be obtained. It requires that running time of Blakley's algorithm be inspected in more details. The running time of Blakley's algorithm can be formulated as following:

$$T(r) = \sum_{i=t-1}^{0} (t_1 + \alpha_i t_2 + \beta_i(t_1 + \gamma_i t_2)) \tag{1}$$

where:

- t_1: is the running time of either steps 3 and 4 or steps 7 and 8.
- t_2: is running time of either step 5 or step 9.
- α_i: is 1 if $Er_{1,i}$ occurs, otherwise it is 0.
- β_i: is one if bit a_i is one.
- γ_i: is 1 if $Er_{2,i}$ occurs, otherwise it is 0.

Running time of each modular addition in Blakley's Algorithm is not independent of other modular additions. The dependency between running time of steps of Blakley's algorithm is a major difficulty in obtaining the distributions. To consider effect of modular multiplications on each other, *Extra Reductions Neighborhood Window*, ERNW, is used. $ERNW\{x, y\}$ of an Er is a set, containing x numbers of Ers may be occurred before the Er and y numbers of Ers can be occurred after the Er. It is supposed that occurrence of the Er is independent of other Ers that do not belong to the window. For example, if $a_{i+1} = 1$, $a_i = 1$ and $a_{i-1} = 0$, the $ERNW\{1, 1\}$ of $Er_{2,i}$ is $\{Er_{1,i}, Er_{1,i-1}\}$, and $ERNW\{2, 2\}$ of $Er_{2,i}$ is $\{Er_{2,i+1}, Er_{1,i}, Er_{1,i-1}, Er_{1,i-2}\}$. The $Er_{2,i-1}$ does not belong to this set, because $a_{i-1} = 0$ and $Er_{2,i-1}$ cannot occur.

To compute the distributions, we need expected number of messages in T_0' and T_1' and their running time. Equation 1 states that running time of a message is a summation of running time of modular additions. Two random variables, p and b, are involved in running time of modular additions. Expected number of messages and running time of each message are approximated in $(p, b)-$plan. The $(p, b)-$plan can be divided into some regions according to a Er and its $ERNW\{x, y\}$. For example if $Er_{2,i}$ and its $ERNW\{1, 0\} = \{Er_{1,i}\}$ are considered, the $(p, b)-$plan is first partitioned into two regions, a region where $Er_{1,i}$

is occurred and a region where it does not occurred. Each of these regions are then divided into two sub-regions according to $Er_{2,i}$. This partitioning is shown in Fig. 1. An unique code is assigned to each region that shows a situation of $Er_{1,i}$ and $Er_{2,i}$, for example "region 01" contain all $p_{1,i}$ that cause $Er_{1,i}$ occurs but $Er_{2,i}$ does not occur.

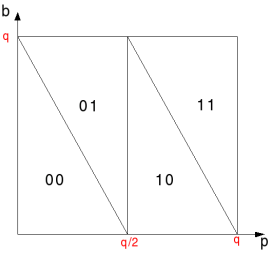

Fig. 1. $(p, b)-$plane for $ERNW = \{1, 0\}$ of $Er_{2,i}$

The "Approximate the d_0 or d_1" algorithm uses such partitioning, obtains expected messages numbers in T_0' and T_1' and their running time, and finds an approximation for d_0 or d_1 for given $ERNW\{x, y\}$.

ALGORITHM Approximate the d_0 or d_1
INPUT: B, T, $ERNW\{x, y\}$ of $Er_{2,i}$, and a'
OUTPUT: Approximation for d_0 or d_1.
1. To obtain an approximation for d_0 , create $ERS = ERNW\{x, y\}$.
To obtain an approximation for d_1, create
$ERS = ERNW\{x, y\} \bigcup Er_{2,i} = \{\underbrace{Er_1, Er_2, Er_3, \ldots, Er_x}_{x}, Er_{2,i}, \underbrace{Er_1, Er_2, \ldots, Er_y}_{y}\}$.
2. According to Ers belong to ERS, the $(p, b)-$plane is divided into $2^{|ERS|}$ regions. Each region specifies a situation of Er occurrence and a code is assigned to this region as $< Er_1 Er_2 \ldots Er_y >$. For example, the region $< 00 \ldots 0 >$ covers all (p, b) that none of modular additions has extra reduction.
3. According to $Er_{2,i}$, B is divided into two subsets, i.e. B_0 and B_1. B_0 contains all inputs that $Er_{2,i}$ does not occur for them, if $a'' =< a'1 >$ is used to simulate the Blakley's algorithm. B_1 contains remaining members of B. B_0 and B_1 also define two regions in (p, b)-plane.
4. For B_0 and B_1, find their regions overlap size with the created regions in step 2.
5. Find approximation for average running time of B_0 and B_1:
$\overline{T_0} = \sum_{\text{for all regions}}(\text{Overlap Sized with } B_0)(\text{Ham(the code of region)})$
$\overline{T_1} = \sum_{\text{for all regions}}(\text{Overlap Sized with } B_1)(\text{Ham(the code of region)})$
Where $\text{Ham}(< \ldots >)$ is a hamming weight of a code.
6. Return $\overline{T_1} - \overline{T_0}$

Table 1. d_0 and d_1 for some $ERNW$

$ERNW$	$\{\}$	$\{Er_{1,i}, Er_{1,i-1}\}$	$\{Er_{1,i+1}, Er_{1,i}, Er_{1,i-1}, Er_{2,i-1}\}$	$\{Er_{1,i+1}, Er_{2,i+1}, Er_{1,i}, Er_{1,i-1}\}$
d_0	0	$\frac{1}{4}t_2$	$\frac{1}{2}t_2$	$\frac{120}{192}t_2$
d_1	t_2	$\frac{7}{16}t_2$	$\frac{1}{3}t_2$	$\frac{1}{3}t_2$

Dividing $(p, b)-$plane in step 2 and finding overlap size in step 4 of the algorithm, will be very complicated, if there is no constraint on inputs. We consider the following assumptions:

1. p and b have uniform distributions in interval $[0, q]$.
2. p and b, which are used in the right hand side of the Addition-2, are independent.

The validity of these assumptions will be discussed in subsection 3.2. Using these assumptions, d_0 and d_1 are approximated for some $ERNW$, which are shown in Table 1. It can be seen from the table that:

- When bigger $ERNW$ are considered, i.e. dependency between more bits are considered, d_0 increases and d_1 decreases.
- For adequate large $ERNW$, we have $d_0 > d_1$.

It is easy to experimentally apply the approximation algorithm to larger $ERNW$ and verify that $d_0 > d_1$. Table 1 also indicates that d_0 and d_1 are dependent on $ERNW$ members. Hence they are dependent on the bits of the input a, which is assumed as secret parameter of a cryptographic function. Attacker can not find distribution of d_0 and d_1 directly, because he does not know the secret parameter. Even if he had a machine similar to the victim's machine, he could not directly find the distribution of the random variables, because the distributions are dependent on the secret parameter value. In the section 4 a heuristic algorithm is described to have a solution for the problem.

3.2 Assumptions

Two assumptions yield $d_0 > d_1$. First, it is supposed that p and b are distributed uniformly on interval $[0, q]$. If an uniform random number generator is used, operand b, which is passed as input to Blakley's algorithm, has uniform distribution. p is obtained from b. It is easy to show that if has b uniform distribution, p will distribute uniformly, so the first assumption is valid.

Second, it is supposed that the used p and b in the right hand side of the Addition-2 are independent. Blakley's algorithm scans bits of the a from the most significant to the least significant bit. Before the first most significant "1" of the a, the Addition-2 hasn't been executed. In a few rounds after the most significant one, if Addition-2 executes, p and b are not independent; however the

Table 2. Correlation coefficient for a few rounds

i	$t-1$	$t-2$	$t-3$	$t-4$	$t-5$	$t-6$
a_i	0	0	1	1	1	1
Correlation Coefficient	-	-	-	0.500	0.167	0.071

correlation coefficient of p and b decreases in each round. Correlation coefficient in each round is dependent on previous bits of a. Table 2 shows the value of bit a_i and the correlation coefficient of $p_{2,i}$ and b in each round, in an exemplary experiment.

Therefore the second assumption is also valid, except a few rounds in start of Blakley's algorithm. Due to these correlations in the rounds, the random variable d (step 5 of "Timing Attack") does not show expected behavior, so $d_0 \leq d_1$ in a few rounds in start of Blakley's algorithm. Thus, the attacker can not guess the most significant bits correctly by simply comparing random variable d, against distributions of d_0 and d_1. The following heuristic algorithm solves this problem, too.

4 Threshold Finding Algorithm

Here, we propose a simple heuristic algorithm to solve the above mentioned problems. First, obtaining the exact distribution of random variables d_0 and d_1 is not necessary to run the attack, but separating the distributions of d_0 and d_1 is sufficient. "Find Threshold" algorithm separates the distributions and finds the border, $\overline{d}$, between the distribution of d_0 and the distribution of d_1. The step 6 in the "Timing Attack" uses the $\overline{d}$ to guess a bit of the secret key. It was already shown that $d_0 > d_1$, hence in this step if $d > \overline{d}$ it means that $d \in d_0$ and the bit a_i is guessed as 0, but if $d < \overline{d}$ then $d \in d_1$. Second, when attacker uses the "Find Threshold" algorithm, he won't be worry about misbehavior of random variable d in start of attack.

"Find Threshold" assumes that attacker knows position of the most significant "1" of the secret parameter a, which is shown by o. He tests all cases of a few subsequent bits and find $\overline{d}$. The method is described formally in "Find Threshold" algorithm.

After $\overline{d}$ is obtained, the "Timing Attack" algorithm is applied on B, T and $a^{(i)}$, and the remaining bits, $a_{t-o-w-1}$ to a_0, are found. Attacker gets 2^w guesses for $< a_{t-o-w-1} \dots a_0 >$ which only one of them is valid. Other constraints on parameters might be used to find the correct guess. For example when we are attacking on DSA. Obtained secret key, x, must satisfy $y = g^x \bmod q$.

In addition to finding an estimation of $\overline{d}$, the attacker ignores misbehavior of d with this algorithm. As if w is large enough, the correlation between p and b is degraded and the assumptions which were discussed in subsection 4.1 are valid, hence it is expected that $d_0 > d_1$.

ALGORITHM Find Threshold
INPUT: Position of the most significant one, o, size of guessing window, w.
A set of inputs of Blakley's Algorithm, $B = \{b_{(1)}, b_{(2)}, \ldots, b_{(n)}\}$.
Running time of algorithm for each input. T_i is running time of $b_{(i)}$, $T = \{T_1, T_2, \ldots, T_n\}$
OUTPUT: $\overline{d}$
1. Construct 2^w guesses for w successive bits after the most significant one:
$a^{(0)} = \underbrace{00\ldots0}_{o-1}1\underbrace{00\ldots0}_{w}$, $a^{(1)} = \underbrace{00\ldots0}_{o-1}1\underbrace{00\ldots1}_{w}$, $a^{(2)} = \underbrace{00\ldots0}_{o-1}1\underbrace{00\ldots10}_{w}, \ldots,$
$a^{(2^w-1)} = \underbrace{00\ldots0}_{o-1}1\underbrace{11\ldots1}_{w}$.
2. For each $a^{(i)}$, apply the "Timing Attack" algorithm using B, T and $a^{(i)}$.
In this case no bit is guessed, only obtained d will be used. The obtained d is added to set D'.
3. Threshold $\overline{d}$ is average of D'.

Table 3. Attack result, when $w = 2$

$\lvert T\rvert \times (10^5)$	5	5	5	10	10	10	10
$\overline{d}$	11370.3	12911.6	11178.7	13096.8	10529.2	11765.0	10239.8
$Err.Num.$	5	0	1	0	1	0	1

5 Practical Results

The proposed timing attack was practically applied on pure modular multiplication and on-line implementation of DSA. Here, only the results of the attack against DSA are presented. Victim machine is an on-line implementation of DSA running in MS-DOS operating system on Athlon-XP 800 MHz. Two internal 32 bit counters of CPU are used as timing measurement facilities [9]. Running time is measured with respect to the number of required CPU cycles to run the DSA algorithm.

There are two parameters in our attacks, i.e. size of T, the number of timing measurements, and size of the guessing window, w, which is used to in "Find Threshold" algorithm. Table 3. and Table 4. show $\overline{d}$ and the number of incorrect guesses in a run of attack for a constant secret key x, for different numbers of timing measurements and window sizes of 2 and 3, respectively.

In these tables, $\overline{d}$ and the number of incorrect guesses are not reported, when the number of measurements is less than 5×10^5. As in such a case, the distributions of d_0 and d_1 are overlapped and the border between them, $\overline{d}$, is meaningless. Hence, applying the attack is impossible. These tables show that as the number of measurements increases, the number of errors degrades. The tables also show that, using bigger window is less erroneous. When very small window are used, d is prone to error. So, obtained $\overline{d}$ is unreliable to properly separate the distribution of d_0 and d_1. Our practical attacks found 160 bit of DSA secret key using window size 3 and 10^6 timing measurements.

Table 4. Attack result, when $w = 3$

$\lvert T\rvert \times (10^5)$	5	5	5	10	10	10	10
$\overline{d}$	11288.3	11731.3	11064.1	13004.4	10410.6	11691.1	10124.6
$Err.Num.$	1	0	5	0	0	0	0

The running time of the attack and the amount of required time to gather the timing samples are directly related to the size of the parameters w and $|T|$. Although increasing the size of parameters enhances the results of attack, it causes more running time. The Running time of the "Timing Attack" algorithm is $O(|T|)$, running time of "Find Threshold" algorithm is exponentially related to w. An increasing in the window size from w to $w + 1$, doubles the attack time to find valid $a^{(i)}$, because the number of $a^{(i)}$ is 2^w.

In our experiments, given the T and the a^i, it takes about 70 min. to run the "Timing Attack". Empirical running time of "Threshold Finding" is dependent on the w. When $w = 3$ it takes about 12 min.

6 Conclusion

In this paper, we proposed a new timing attack on an implementation of modular multiplication, which is called as Blakley's modular multiplication algorithm. To our best knowledge this is first time that timing attack is applied on modular multiplication. To attack, it is assumed that the secret key is passed as parameter a to Blakley's algorithm, attacker knows the parameter b, and can measure the running time of Blakley's algorithm. Blakley's algorithm may be employed for multiplication in public key cryptography or digital signature. If the assumptions are valid, their implementations are vulnerable.

In this paper we focused on DSA and applied our timing attack on on-line implementation of DSA. Our experimental results shows that the proposed timing attack finds out the secret key used in DSA, practically. The attack only gathers timing sample from victim machine and does not require extra information about implementation details or a machine similar to victim's machine. The set of measured timing data are divided into two subsets based on the extra reduction in the Addition-2 of Blakley's algorithm. It is shown that the secret key can be guessed according to the difference between averages of these subsets.

Our timing attack is against a specific implementation of modular multiplication. The modular multiplication $a.b \bmod q$ is a symmetric operation in a and b, this is not anymore the case in the internal operations performed by the Blakley's Algorithm. So, if x (DSA secret key) is passed as input b of Blakley's Algorithm then not much information leaks and our attack is not applicable. Using $Er_{1,i}$ instead of $Er_{2,i}$ in attack, timing attack on more general implementation of Blakley's algorithm, in which secret parameter is passed as parameter b, use mathematical model to prove $d_0 > d_1$, and timing attack on other implementation of modular multiplication are open problems.

References

1. G. R. Blakley. "A computer algorithm for calculating the product AB modulo M". *IEEE Transactions on Computers*, 32(5):497-500, May 1983.
2. D. Brumley, D. Boneh, "Remote Timing Attacks are Practical", *Proceedings of the 12th Usenix Security Symposium*, 2003.
3. J. Cathalo, F. Koeune, and J.-J. Quisquater, "A New Type of Timing Attack: Application to GPS", *Cryptographic Hardware and Embedded Systems - CHES 2003*, LNCS 2779, 2003, pp. 291–303.
4. J. F. Dhem, F. Koeune, P. A. Leroux, P. Mestre, J. J. Quisquater and J. L. Willems, "A Practical Implementation of the Timing Attack", *3rd Working Conference on Smart Card Research and Advanced Applications - CARDIS 1998*, Springer-Verlag, LNCS No. 1820, 1998.
5. E. W. Felten, and M. A. Schneider, " Timing Attacks on Web Privacy", *CCS 2000, Athens, Greece*, 2000.
6. R. Focardi, R. Gorrieri, R. Lanotte, A. Maggiolo-Schettini, F. Martinelli, S. Tini, E. Tronci, "Formal Models of Timing Attacks on Web Privacy", *Electronic Notes in Theoretical Computer Science*, vol. 62, 2001.
7. H. Handschuh, and H. M. Heys, "A Timing Attack on RC5", *SAC'98*, LNCS 1556, 1999, pp. 306–318.
8. A. Hevia, M. Kiwi, "Strength of Two Data Encryption Standard Implementations Under Timing Attacks", *LATIN'98*, LNCS 1380, 1998, pp. 192–205.
9. Intel, "Using the RDTSC instruction for performance monitoring", *Technical report*, 1997.
10. P. C. Kocher, "Timing Attacks on Implementations of Diffie–Hellman, RSA, DSA, and Other Systems", *Advances in Cryptology - CRYPTO'96*, Springer-Verlag, LNCS No. 1109, 1996, pp. 104–113.
11. F. Koeune, J. J. Quisquater, "A Timing Attack against Rijndael", *Technical Report CG-1999/1*, June 1999.
12. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, "Handbook of Applied Cryptography", *CRC Press*, 1996.
13. National Institute of Standard and Technology (NIST), "Digital Signature Standard", FIPS PUB 186-2, http://csrc.nist.gov/publications/fips/fips186-2/fips186-2.pdf
14. W. Schindler, "A Combined Timing and Power Attack", *PKC 2002*, LNCS 2274, 2002, pp. 263–279.
15. W. Schindler, C. D. Walter, "More Detail for a Combined Timing and Power Attack against Implementations of RSA", *9th IMA International Conference on Cryptography and Coding*, LNCS No. 2898, 2003, pp. 245–263.
16. W. Schindler, "A Timing Attack against RSA with the Chinese Remainder Theorem", *Cryptographic Hardware and Embedded Systems - CHES 2000*, Springer-Verlag, LNCS No. 1965, 2000, pp. 109–124.
17. W. Schindler, "Optimized timing attacks against public key cryptosystems", *Statistics and Decisions*, volume 20, 2002, pp. 191–210.
18. W. Schindler, "On the Optimization of Side-Channel Attacks by Advanced Stochastic Methods", *8th International Workshop on Practice and Theory in Public Key Cryptography PKC 2005*, Springer-Verlag, LNCS No. 3386, 2005, pp. 85–103.