Improving and Extending the Lim/Lee Exponentiation Algorithm

Biljana Cubaleska[1], Andreas Rieke[2], and Thomas Hermann[3]

[1] FernUniversität Hagen, Department of communication systems
Feithstr. 142, 58084 Hagen, Germany
biljana.cubaleska@fernuni-hagen.de
http://ks.fernuni-hagen.de/mitarbeiter/cubalesk/
[2] ISL Internet Sicherheitslösungen GmbH
Feithstr. 142, 58097 Hagen, Germany
andreas.rieke@isl-online.de
http://ks.fernuni-hagen.de/~rieke/
[3] MMK GmbH, Feithstr. 142
58097 Hagen, Germany
thomas.hermann@mmk-hagen.de
http://www.mmk-hagen.de/

Abstract. In [5] Lim and Lee present an algorithm for fast exponentiation in a given group which is optimized for a limited amount of storage. The algorithm uses one precomputation for several computations in order to minimize the average time needed for one exponentiation. This paper generalizes the previous work proposing several improvements and a method for fast precomputation. The basic Lim/Lee algorithm is improved by determining the optimal segmentation of the exponent. Finally, it is shown that the improved Lim/Lee algorithm is faster than the previous one in average case.

1 Introduction

Modular exponentiation is a basic operation widely used in cryptography. In many cryptographic protocols users must perform one or more exponentiations in a given group. Well known examples are encryption, decryption and signatures with RSA [8], signature generation and identification as in Digital Signature Standard (DSS) [7], Brickell/McCurley [2], Schnorr [10], and many other schemes. The exponentiation can be decomposed into a large number of multiplications, so it is an operation which is heavily computational, consumes a lot of time and constitutes a computational bottleneck in many protocols. The efficiency of most public-key crypto systems mainly depends on the speed of the exponentiation algorithm.

Classical algorithms for exponentiation are the binary algorithm (known as the square-and-multiply method, [4]) and the signed binary algorithm [3]. Other algorithms use some amount of storage for intermediate values in order to improve the performance. Examples are the windowing method [4,6] and algorithms based on addition chains [1,9]. In [5] Lim and Lee present a new exponentiation algorithm based on precomputations. The goal of this algorithm is to achieve a minimal number of operations (squarings and multiplications) for an exponentiation under the condition of limited amount of storage. The required number of operations for the precomputation has thereby not been considered.

The Lim/Lee algorithm optimizes the evaluation of the exponentiation g^e in a given group (usually Z_N, N being a large prime or a product of two large primes) in a case when the base g is fixed and the exponent e is randomly chosen. The fixed base g allows the usage of a precomputation table in order to reduce the number of computations required, but the algorithm has an additional cost of storage for the precomputed values. Such an algorithm that is independent of e, but depends on g is suitable for use in most discrete logarithm based protocols for signature generation and identification (e.g. [7,2,10]).

In this paper we present a generalization and several improvements of the exponentiation algorithm of Lim/Lee. We focus our observations on the speed of the algorithm without concerning the storage costs for the precomputed elements, and then compare it's behaviour for limited storage. Furthermore, we present an efficient algorithm for precomputation and therewith optimize the total number of operations for a given exponentiation, i.e. the number of operations for the precomputation and for several computations based on it. Finally, both algorithms are compared for variable length of the exponent.

The Lim/Lee algorithm is described in section 2 and our improvements are presented in section 3. A new precomputation algorithm is proposed in section 4. Some comparisons of the variants of the Lim/Lee algorithm in a case when unlimited storage is available with the windowing method are given in section 5 and the behaviour of the algorithm under the condition of limited storage is presented in section 6. Finally, section 7 concludes the paper.

2 The Lim/Lee Algorithm

In this section the Lim/Lee exponentiation algorithm is briefly presented with a slightly changed terminology. In the next section we present and discuss the improved and extended algorithm.

In order to compute the exponentiation g^e with the Lim/Lee algorithm, the l-bit exponent e is divided into h blocks e_i, each with length $a = \lceil \frac{l}{h} \rceil$. The exponent e can be written as

$$e = e_{h-1}e_{h-2}\dots e_1 e_0 = \sum_{i=0}^{h-1} e_i 2^{ia}. \tag{1}$$

Each of the blocks e_i is further subdivided into v smaller blocks of size $b = \lceil \frac{a}{v} \rceil$ and each block e_i can be represented as

$$e_i = e_{i,v-1}e_{i,v-2}\dots e_{i,1}e_{i,0} = \sum_{j=0}^{v-1} e_{i,j} 2^{jb}. \tag{2}$$

Each block $e_{i,j}$ consists of b bits $e_{i,j,k}$ and can be represented as

$$e_{i,j} = e_{i,j,b-1} e_{i,j,b-2} \ldots e_{i,j,1} e_{i,j,0} = \sum_{k=0}^{b-1} e_{i,j,k} 2^k. \tag{3}$$

It is further assumed that the number of blocks h and v are chosen in a way that e_{h-1} and $e_{i,v-1}, 0 \le i < h-1$ are not equal to zero. The segmentation of the exponent e is shown in figure 1. Based on the length l of the exponent and on the parameters h and v, the precomputation leads to the array $G[j][u]$ with $0 \le j < v$ and $1 \le u < 2^h$. Employing the binary representation $u_{h-1}u_{h-2}\ldots u_1 u_0$ of u and $r_i = g^{2^{ia}}$, the array $G[j][u]$ is defined by the following equations:

$$G[0][u] = r_{h-1}^{u_{h-1}} r_{h-2}^{u_{h-2}} \ldots r_1^{u_1} r_0^{u_0} \tag{4}$$

$$G[j][u] = (G[j-1][u])^{2^b} = G[0][u]^{2^{jb}} \quad \forall \quad 1 \le j < v \tag{5}$$

Using the definition

$$I_{j,k} = \sum_{i=0}^{h-1} e_{i,j,k} 2^i, \tag{6}$$

the exponentiation can be described with the following algorithm:

```
1. SET $R = 1$.
2. FOR $k = b - 1$ DOWNTO 0
   (a) SET $R = R^2$.
   (b) FOR $j = v - 1$ DOWNTO 0
        i. SET $R = R \cdot G[j][I_{j,k}]$.
3. RETURN $R$.
```

We denote the considered algorithm as 1. Lim/Lee algorithm. This algorithm needs $b-1$ squarings[1] and $\frac{2^h-1}{2^h}a-1$ multiplications in average, but $a-1$ multiplications in the worst case. Thus, the average number of operations needed to perform a single exponentiation with this algorithm is

$$C_{Lim/Lee,1} = \alpha\,(b-1) + \frac{2^h-1}{2^h}a - 1. \tag{7}$$

In [5] the exponent is represented either as in figure 1 – in several rows each of v blocks – or as shown in figure 2 – with a shortened last row. The length of the blocks by partitioning as in figure 2 (we denote it as 2. Lim/Lee algorithm) is

$$b_2 = \left\lceil \frac{l}{(h-1)v + v_{last}} \right\rceil \tag{8}$$

$$b_1 = \left\lceil \frac{l - b_2 \cdot h \cdot v_{last}}{(h-1)(v - v_{last})} \right\rceil . \tag{9}$$

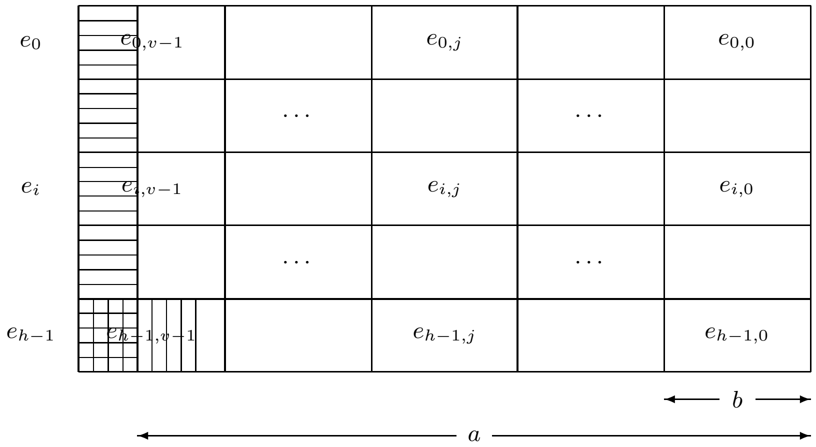

Fig. 1. Partitioning an l-bit exponent e in h rows with the same length according to the 1. Lim/Lee algorithm

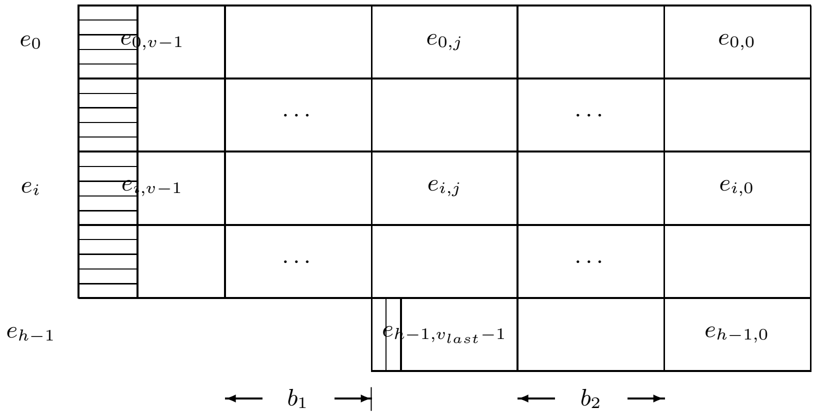

Fig. 2. Partitioning of a l-bit-exponent e with shortened last row according to the 2. Lim/Lee algorithm

The computational cost in this case consists of $b_2 - 1$ squarings, $b_1(v - v_{last}) + b_2 \cdot v_{last} - 1$ multiplications in the maximum and $\frac{2^{h-1}-1}{2^{h-1}} b_1(v - v_{last}) + \frac{2^h-1}{2^h} b_2 \cdot v_{last} - 1$ multiplications in the average. Thus, the average number of operation

[1] Many squaring algorithms are faster than an ordinary multiplication, by making use of the fact that both multiplicands are equal. On the other hand, multiplication is performed with a constant multiplicand and – after performing some precomputations – may also be faster than a squaring. Here α denotes the ratio of the computational complexity of the algorithms for squaring and multiplying.

needed for one exponentiation is

$$C_{Lim/Lee,2} = \alpha\,(b_2 - 1) + \frac{2^{h-1}-1}{2^{h-1}} b_1\,(v - v_{last}) + \frac{2^h - 1}{2^h} b_2 \cdot v_{last} - 1. \quad (10)$$

3 The Improved Algorithm

When using the algorithms described in the last section, the following problems arise:

- Two algorithms are presented by Lim and Lee without a statement which algorithm should be preferred in a given case.
- It is not examined how to find the optimal choice of the parameters h and v resp. h, v and v_{last}; exhaustive search with three parameters is very expensive.

In order to find appropriate solutions for this problem we examine the algorithms described in the previous section in some detail. If we allow the partitioning of the exponent with $v_{last} \geq 0$ in a rectangular form as a special case of the second algorithm, the result in case of same partitioning ($h_1 = h_2 - 1$ and $v_1 = v_2$) is $b_1 = b = b_2$ and thus $C_{Lim/Lee,1} \geq C_{Lim/Lee,2}$. It is obvious that in case of $v_{last} = 0$ the second algorithm can not be better than the first one.

It is also shown in section 5 that the second algorithm is not worse than the first one in any point. This can also be seen from the results of the numerical tests for all lengths of the exponent l in the range up to 512 bit.

Since the second algorithm has been identified to be the better one, we further try to decrease the number of basic parameters in this algorithm. The existence of three basic parameters (h, v and v_{last}) makes the exhaustive search expendable and slow. A fundamental advantage can be achieved in the case if a and b are used as basic parameters instead of h and v. We derive the parameters $h = \lceil \frac{l}{a} \rceil$ and $v = \lceil \frac{a}{b} \rceil$ from the basic parameters a and b, and the partitioning of the exponent that results from this determination of the basic parametar is given in figure 3. We denote the partitioning the exponent in this way as 3. Lim/Lee algorithm. The number of bits in the last row is now only $a_{last} = l - a(h-1)$, and they are divided into $v_{last} = \lceil \frac{a_{last}}{b} \rceil$ blocks. The number of bits in the last block is $b_{last} = a_{last} - b(v_{last} - 1)$. With this new way of partitioning of the exponent we can achieve computational cost of

$$C_{Lim/Lee,3} = \alpha\,(b - 1) + \frac{2^h - 1}{2^h} a - 1 \quad (11)$$

as the average number of operations for one exponentiation. Although this formula is the same as (7), the parameters a, b, and h can have values different from those in (7) due to the different segmentation. It is shown in section 5 that the computational cost for this algorithm is never higher than the computational cost for each of both basic algorithms.

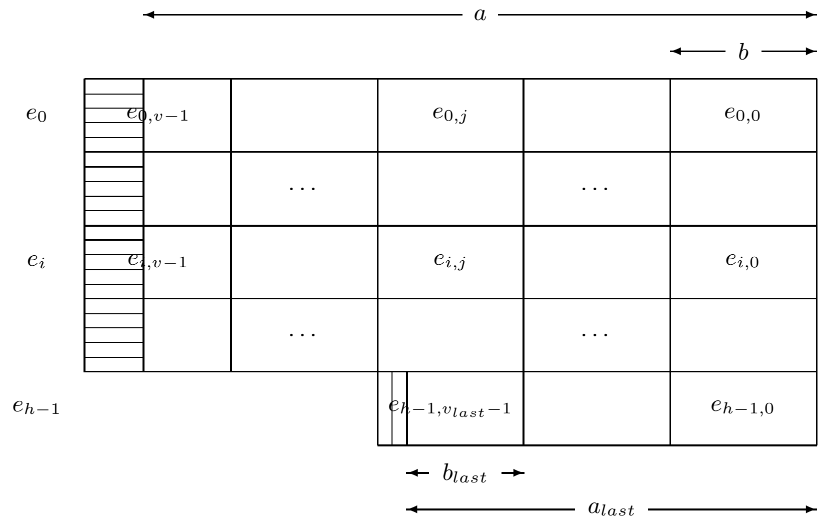

Fig. 3. Partitioning of a l-bit exponent e according to the extended, 3. Lim/Lee algorithm

A further improvement results from the observation of the last row in figure 3: The term $\frac{2^h-1}{2^h}a-1$ in equation 11 results from the fact that the multiplication in line 2.b.i in the algorithm description in section 2 is trivial with probability 2^{-h}. Concerning the fact that the last row must not be filled completely we can use the term $\frac{2^{h-1}-1}{2^{h-1}}(a-a_{last})+\frac{2^h-1}{2^h}a_{last}-1$ instead (we denote it as 4. Lim/Lee algorithm). This improvement does not concern the algorithm itself, but only the specification of its average number of operation, and leads to

$$C_{Lim/Lee,4}=\alpha\,(b-1)+\frac{2^{h-1}-1}{2^{h-1}}\,(a-a_{last})+\frac{2^h-1}{2^h}a_{last}-1 \qquad (12)$$

as the average number of operations for one exponentiaton.

4 Precomputation

In all variations of the Lim/Lee algorithm analyzed above only the number of operations for the computation is concerned, assuming that the precomputation has already been performed. The computational cost for the precomputation of the array $G[j][u]$ has not been considered at all and an algorithm for the precomputation of the array $G[j][u]$ is not given in [5]. Proposals for the precomputation algorithm can be found in other works (e.g. [6, p. 626]), but they are not efficient. Since the precomputation makes a considerable fraction of the total number of operations, we present an efficient algorithm for the precomputation consisting of two steps:

1. Since we have the array $G[j][u]$ for $u = 2^i$, $0 \leq i < h$ and $0 \leq j < v$ in form r^{2^x}, the appropriate values can be computed with repeated squarings beginning with r. Always when the exponent matches to the exponent in (4) and/or (5), the corresponding value is assigned to the array element. The number of squarings needed for this step is
$$a(h-1) + b\left(v_{last} - 1\right). \tag{13}$$
2. The remaining elements can be computed directly from the last step with one multiplication for each case. From $2^{h-1} - 1$ rows in the upper half, $h - 1$ rows have already been computed in the first step, whereas only one from the 2^{h-1} rows from the lower half is already finished. The number of multiplications needed for this step is
$$v\left(2^{h-1} - h\right) + v_{last}\left(2^{h-1} - 1\right). \tag{14}$$

Thus, the total number of operations needed for the precomputation is
$$P = \alpha\left(a(h-1) + b\left(v_{last} - 1\right)\right) + v\left(2^{h-1} - h\right) + v_{last}\left(2^{h-1} - 1\right). \tag{15}$$

An example for precomputation with partitioning of the exponent with $h = 3$, $v = 3$, $v_{last} = 2$ is given in figure 4. The values computed with squarings in the first step are marked with S, and the multiplications in the second step are marked with M. The results of the precomputations and computations with

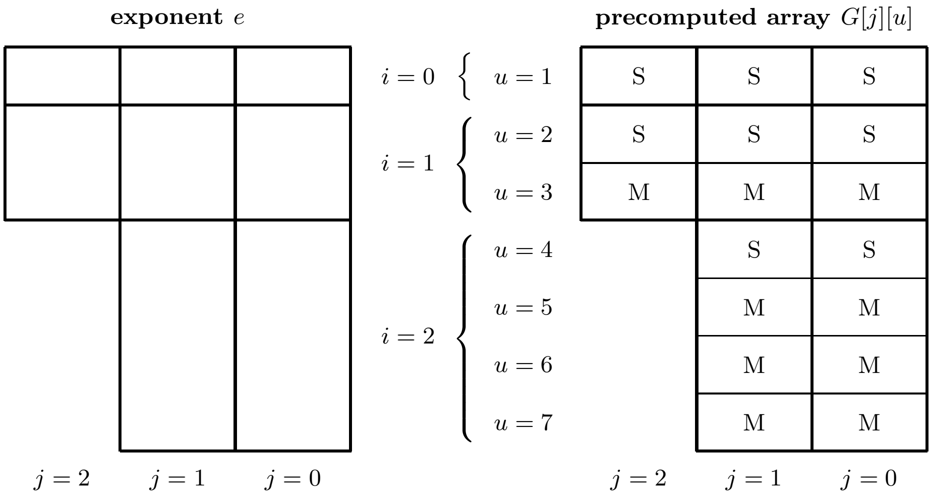

Fig. 4. Precomputation of the array $G[j][u]$ in the Lim/Lee algorithm ($h = 3$, $v = 3$ and $v_{last} = 2$)
M: Multiplication
S: Squaring

various parameters and partitioning for one exponent with length $l = 15$ are shown in table 1. Using exhaustive search, the parameters shown in the table have been found out as optimal for this case. The computational cost is

$$O_{Lim/Lee,x} = P + z \cdot C_{Lim/Lee,x} \tag{16}$$

where z denotes the number of computations based on a single precomputation, and $x = 1, 2, 3, 4$.

Table 1. Comparison of the Lim/Lee algorithms using the example $l = 15, \alpha = 1$ and $z = 1$

C: Number of operations needed for the computation

P: Number of operations needed for the precomputation

O: Total number of operations (precomputation and computation)

	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4
h	1	2	2	2
v	1	2	2	2
v_{last}		1	1	1
a	15		8	8
b_1		5		
b resp. b_2	15	5	7	7
C	20.5	9.25	11	10.75
P	0	11	9	9
O	20.5	20.25	20	19.75

5 Optimization of Precomputation and Computation for (Nearly) Unlimited Memory

The variations of the Lim/Lee algorithm described in the sections 2 and 3 are analyzed and compared in this section for exponents up to 512 bit length. A comparison with the windowing exponentiation algorithm is also given. The efficiency of the algorithms is measured by the average number of operations, where the multiplications and the squarings are treated equally ($\alpha = 1$).

The optimal parameters for each algorithm have been determined with exhaustive search. The results for the range $l \leq 192$ are shown in figure 5. The improvements compared to the first Lim/Lee algorithm are shown in per cents.

An exact comparison of the required number of operations in one exponentiation (precomputation and computation) shows that for all exponent lengths l and a single exponentiation ($z = 1$) the following inequality

$$O_{Lim/Lee,1} \geq O_{Lim/Lee,2} \geq O_{Lim/Lee,3} \geq O_{Lim/Lee,4} \tag{17}$$

is satisfied. This means that the fourth Lim/Lee algorithm has the lowest computational cost, and we set

$$O_{Lim/Lee} = O_{Lim/Lee,4}. \tag{18}$$

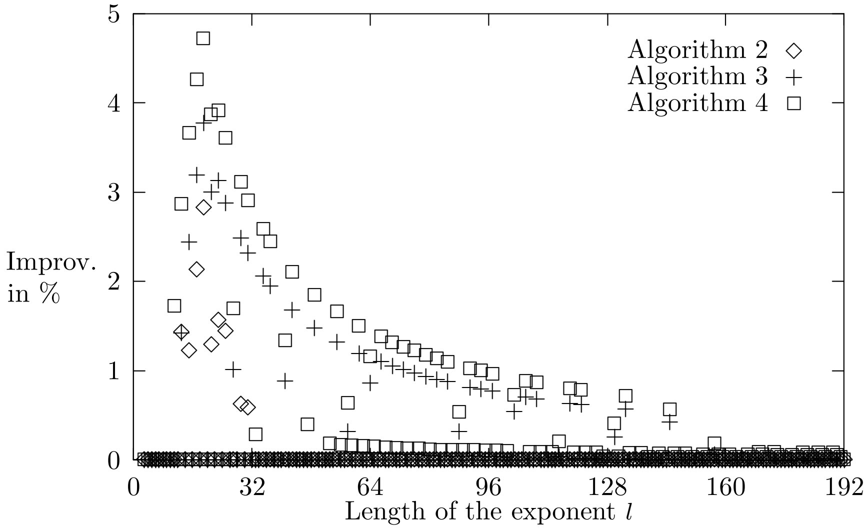

Fig. 5. Comparison of the variants of the Lim/Lee algorithm

In the range of $l > 192$ that is not presented, the algorithms 1-3 behave in the same way and only the fourth algorithm has small improvements.

We now compare the windowing exponentiation algorithm with the Lim/Lee algorithm. In figure 6 we can see the average number of operations, whereby the average number of operations needed for one exponentiation consist of one precomputation and z computations.

First, for a given length of the exponent l and given number of exponentiations z, the optimal values of the parameters (k for the windowing algorithm, a and b for the Lim/Lee algorithm) are determined with exhaustive search. Then the total number of operations is determined as a sum of the number of operations for the precomputation and number of operations for z computations. We get the average number of operations dividing the total number of operations by z, in order to get comparable results.

If only a single exponentiation is performed, the windowing algorithm leads to the lowest number of operations. In a case when several exponentiations which base on the same precomputation need to be performed, and the exponents have the same length l, the Lim/Lee algorithm has the best performance.

6 Optimization of the Computation for Limited Memory

The results presented in the last section concern the case when the amount of storage needed for the intermediate results of the precomputation is practically unlimited. This condition is satisfied when the exponentiation is performed on a PC or workstation. But the case when only limited storage and processing power are available must also be concerned, since many of the cryptographic protocols

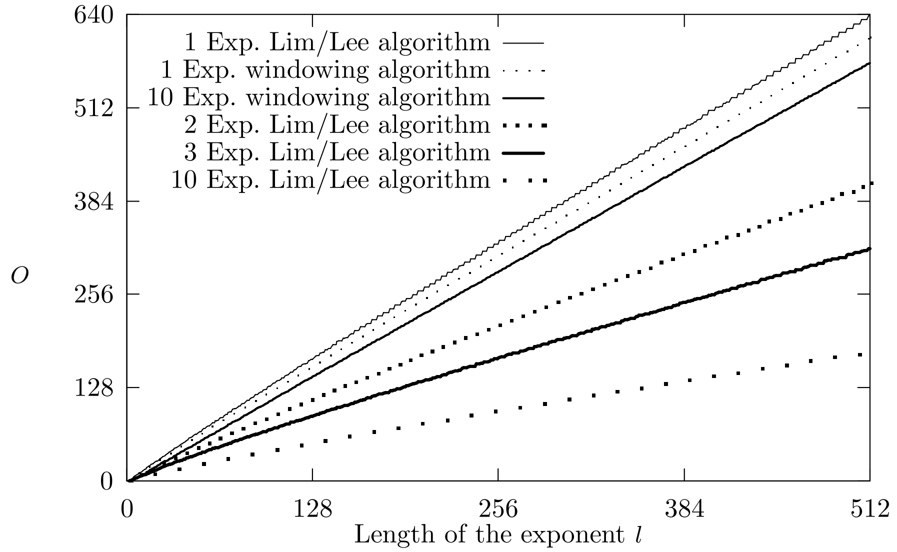

Fig. 6. Comparison of the exponentiation algorithms
O: Average number of operations needed for a single exponentiation

using exponentiation (signature generation and verification schemes) are performed on smart cards. The precomputation table in the Lim/Lee algorithm can reduce the number of multiplications required at the expense of storage for the precomputed values.

The numerical results for the time-memory tradeoffs for exponent lengths of 160 and 512 bit are summarized in the tables 2 and 3 for various variants of the Lim/Lee algorithm and optimal partitioning of the exponent in each case. The same examples as in [5] are considered and it is assumed that the squarings and the multiplications have the same computational cost ($\alpha = 1$). The improved Lim/Lee algorithm is usually faster and never slower than the others in average case. We can see that if more storage is available, the algorithm becomes faster by the decreasing number of multiplications required. The compromise by limited storage is a slower algorithm due to the increasing number of operations required. An exponentiation with 160 bit exponent can be performed with only 19.96 multiplications in average if 2299 intermediate values from the precomputation are stored. If only 10 values can be stored, the same exponentiation requires 82 multiplications. So, by known storage capacity of a smart card, we can estimate how fast the given exponentiation can be and vice versa.

Table 2. Exponentiations with 160 bit (AC: Average case, WC: Worst case)

	Algorithm 1			Algorithm 2			Algorithm 3			Algorithm 4		
Sto-	Conf.	AC	WC	Conf.	AC	WC	Conf.	AC	WC	Conf.	AC	WC
rage	h/v			$h/v/v_{last}$			a/b			a/b		
6	2/2	98	118	3/2/0	98	118	80/40	98	118	80/40	98	118
10	2/3	85	105	3/2/1	82	94	80/27	85	105	64/32	82	94
14	3/2	72.25	79	4/2/0	72.25	79	54/27	72.25	79	54/27	72	79
22	3/3	63.25	70	4/2/1	62.69	67	54/18	63.25	70	46/23	62.63	67
30	4/2	55.5	58	5/2/0	55.5	58	40/20	55.5	58	40/20	55.5	58
45	4/3	49.5	52	5/3/0	51.38	54	40/14	49.5	52	40/14	49.5	52
62	5/2	45	46	6/2/0	45	46	32/16	45	46	32/16	45	46
77	4/5	43.5	46	5/3/2	42.63	44	34/12	42.94	44	34/12	42.63	44
93	5/3	40	41	6/3/0	40.97	42	32/11	40	41	32/11	40	41
124	5/4	37	38	6/4/0	37	38	32/8	37	38	32/8	37	38
157	5/5	36	37	6/3/2	35.44	36	28/10	35.56	36	28/10	35.44	36
189	6/3	33.58	34	7/3/0	33.58	34	27/9	33.58	34	27/9	33.55	34
252	6/4	31.58	32	6/6/2	32.22	33	27/7	31.58	32	27/7	31.55	32
317	6/5	30.58	31	7/3/2	29.75	30	24/8	29.81	30	24/8	29.75	30
381	7/3	28.82	29	6/7/5	29.44	30	23/8	28.82	29	23/8	28.81	29
508	7/4	26.82	27	7/5/2	27.69	28	23/6	26.82	27	23/6	26.81	27
762	7/6	24.82	25	7/6/4	25.75	26	23/4	24.82	25	23/4	24.81	25
1020	8/4	22.92	23	9/4/0	22.92	23	20/5	22.92	23	20/5	22.92	23
1785	8/7	20.92	21	8/11/3	21.85	22	20/3	20.92	21	20/3	20.92	21
2299	8/7	20.92	21	9/5/4	19.96	20	18/4	19.96	20	18/4	19.96	20

Table 3. Exponentiations with 512 bit (AC: Average case, WC: Worst case)

	Algorithm 1			Algorithm 2			Algorithm 3			Algorithm 4		
Sto-	Conf.	AC	WC	Conf.	AC	WC	Conf.	AC	WC	Conf.	AC	WC
rage	h/v			$h/v/v_{last}$			a/b			a/b		
6	2/2	318	382	3/2/0	318	382	256/128	318	382	256/128	318	382
10	2/3	276	340	3/2/1	267.6	306	257/85	275.8	340	205/103	267.5	306
14	3/2	233.6	255	4/2/0	234.5	256	171/86	233.6	255	171/86	233.5	255
22	3/3	204.6	226	4/2/1	204.4	218	171/57	204.6	226	146/74	204.4	218
30	4/2	182	190	5/2/0	182	190	128/64	182	190	128/64	182	190
45	4/3	161	169	5/3/0	161.9	170	128/43	161	169	128/43	161	169
62	5/2	149.8	153	5/3/1	149.9	156	103/52	149.8	153	103/52	149.7	153
93	5/3	132.8	136	6/3/0	134.7	138	103/35	132.8	136	103/35	132.7	136
125	5/4	123.8	127	6/3/1	123.5	126	103/26	123.8	127	96/32	123.5	126
157	5/5	118.8	122	6/3/2	117.2	119	90/31	117.6	119	90/31	117.2	119
189	6/3	111.7	113	7/3/0	112.6	114	86/29	111.7	113	86/29	111.6	113
252	6/4	104.7	106	6/5/3	106.1	108	86/22	104.7	106	86/22	104.6	106
317	6/5	100.7	102	7/3/2	100.2	101	82/21	100.4	101	82/21	99.88	101
381	7/3	96.42	97	7/4/2	97.06	98	79/20	96.38	97	79/20	96.06	97
508	7/4	90.42	91	7/5/3	91.16	92	74/19	90.42	91	74/19	90.38	91
635	7/5	86.42	87	8/5/0	87.41	88	74/15	86.42	87	74/15	86.38	87
892	7/7	82.42	83	8/4/3	80.68	81	66/17	80.74	81	66/17	80.68	81
1020	8/4	77.75	78	9/4/0	77.75	78	64/16	77.75	78	64/16	77.75	78
1275	8/5	74.75	75	9/5/0	75.75	76	64/13	74.75	75	64/13	74.75	75
1530	8/6	72.75	73	8/7/5	73.68	74	64/11	72.75	73	64/11	72.75	73
2040	8/8	69.75	70	9/8/0	69.75	70	64/8	69.75	70	64/8	69.75	70
2555	9/5	66.89	67	9/6/4	67.84	68	59/10	66.88	67	59/10	66.85	67
3066	9/6	64.89	65	9/8/4	65.83	66	57/10	64.89	65	57/10	64.89	65
4089	9/8	62.89	63	10/7/1	61.90	62	56/8	61.95	62	56/8	61.90	62
5626	9/10	60.89	61	10/6/5	58.94	59	52/9	58.95	59	52/9	58.94	59
7672	10/7	57.95	58	10/8/7	56.95	57	53/6	56.95	57	53/6	56.93	57
10234	10/9	55.95	56	11/6/4	53.97	54	48/8	53.98	54	48/8	53.97	54
13305	11/6	52.98	53	11/7/6	51.97	52	47/7	51.98	52	47/7	51.97	52

7 Conclusions

We generalize the exponentiation algorithm [5] by making several improvements in the computations and in the precomputation achieving a decrease of the computational cost for a single exponentiation (precomputation and computation). For the computations, the partitioning of the exponent is modified in a way which reduces the number of multiplications. Furthermore, we propose a new efficient method of precomputation for the Lim/Lee algorithm, minimizing the total time needed for a single exponentiation. We compare the exponentiation algorithms, showing that if several exponentiations based on the same precomputation are performed, the Lim/Lee algorithm has the best performance. The fact that time consuming precomputations have to be done limits the applicability of the algorithm to those cryptosystems where the same base is used often, which holds for most discrete logarithm based systems. Although only slight improvements are made, these are quite remarkable because of the importance of exponentiation. The exponentiation can be additionally speeded up by applying parallel processing and is applicable to various computing environments due to its wide range of time-storage trade-offs.

References

1. J. Bos and M. Coster. *Addition Chain Heuristics.* In G. Brassard, editor, *Advances in Cryptology - CRYPTO '89.* LNCS 435, pp. 400-407. Springer-Verlag, 1990.
2. E. F. Brickell and K. S. McCurley. *An Interactive Identification Scheme Based on Discrete Logarithms and Factoring.* In *Journal of Cryptology 5* (1992), no. 1, pp. 29-39.
3. J. Jedwab and C. J. Mitchell. *Minimum Weight Modified Signed-Digit Representations and Fast Exponentiation.* Elect. Let. 25 (17), pp. 1171-1172 (1989).
4. D. E. Knuth. *The Art of Computer Programming, Vol.2: Seminumerical algorithms.* Third edition, Addison-Wesley, 1997.
5. C. H. Lim and P. J. Lee. *More Flexible Exponentiation with Precomputation.* In Yvo G. Desmedt, editor, *Advances in Cryptology - CRYPTO '94.* LNCS 839, pp. 95–107. Springer-Verlag, 1994.
6. A. J. Menezes, P. C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press series on discrete mathematics and its applications. CRC Press, Boca Raton, 1997.
7. National Institute of Technology and Standards. *Specifications for the Digital Signature Standard (DSS).* Federal Information Processing Standards Publication XX, US Department of Commerce, February 1 1993.
8. R. L. Rivest, A. Shamir, and L. Adleman. *A Method for Obtaining Digital Signatures and Public Key Cryptosystems.* Communications of ACM 21 (1978), pp 120-126.
9. P. de Rooij. *Efficient Exponentiation Using Precomputation and Vector Addition Chains.* In de Santis, editor, *Advances in Cryptology - EUROCRYPT '94.* LNCS 950, pp. 389-399. Springer-Verlag, 1994.
10. C. P. Schnorr. *Efficient Signature Generation by Smart Cards.* In *Journal of Cryptology 4 (1991), no. 3, pp. 161-174.*