Information Processing Letters 129 (2018) 5–10

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Fast batch modular exponentiation with common-multiplicand multiplication

Jungjoo Seo, Kunsoo Park *,[1]

Department of Computer Science and Engineering, Seoul National University, Seoul 08826, Republic of Korea

ARTICLE INFO

Article history:
Received 3 June 2017
Received in revised form 2 September 2017
Accepted 4 September 2017
Available online 7 September 2017
Communicated by K. Chao

Keywords:
Algorithm
Cryptography
Modular exponentiation
Common-multiplicand multiplication

ABSTRACT

We present an efficient algorithm for batch modular exponentiation which improves upon the previous generalized intersection method with respect to the cost of multiplications. The improvement is achieved by adopting an extended common-multiplicand multiplication technique that efficiently computes more than two multiplications at once. Our algorithm shows a better time-memory tradeoff compared to the previous generalized intersection method. We analyze the cost of multiplications and storage requirement, and show how to choose optimal algorithm parameters that minimize the cost of multiplications.

1. Introduction

Modular exponentiation is a fundamental operation in the field of public-key cryptography for encryption and signing features. The efficiency of modular exponentiation is increasingly important because the required key size has grown to ensure the cryptographic security levels. It is also important because exponentiation is performed in devices with weak computing power as the internet of things becomes ubiquitous.

In a cryptographic application that requires digital signatures [4,7] of multiple messages simultaneously, an efficient batch exponentiation algorithm is required. As a solution to the batch exponentiation problem, the generalized intersection method was proposed by M'Raïhi and Nacacche [6]. This method takes advantage of the fact that intersecting exponents leads to a less number of multiplications. Chung et al. [1] proposed a decremental combination strategy that removes the overlapping multiplications in the combination stage of the generalized intersection method. They also pointed out that various time-memory tradeoffs can be obtained by grouping exponents.

We propose a k-way batch exponentiation algorithm that improves the evaluation stage of the generalized intersection method with exponent-grouping. Our algorithm divides exponents into several groups and produces many common-multiplicand multiplications in the evaluation stage. To handle multiple multiplications at once, the common-multiplicand multiplication technique is extended to compute more than two multiplications. This approach significantly enhances the performance of the evaluation stage and the whole procedure achieves a better time-memory tradeoff compared to Chung et al.'s.

2. Generalized intersection method

Throughout the paper, the i-th element in an array A is denoted by A_i. The i-th bit of an integer $a = \sum_{i=0} a[i]2^i$ is denoted by $a[i]$. For binary operations, let $\wedge$, $\vee$ and $\oplus$ denote the bitwise AND, OR and XOR, respectively. Let $d = (d[n]d[n-1]...d[1])_2$ be an n-bit integer and $A =$

* Corresponding author.
E-mail addresses: jjseo@theory.snu.ac.kr (J. Seo), kpark@theory.snu.ac.kr (K. Park).

[1] Supported by the Bio & Medical Technology Development Program of the NRF funded by the Korean government, MSIP (NRF-2014M3C9A3063541).

http://dx.doi.org/10.1016/j.ipl.2017.09.003
0020-0190/

Algorithm PART

Input: n l-bit exponents $X = \{x_1, x_2, ..., x_n\}$
Output: Position array P
1: $P \leftarrow 0$ for $0 \le i < l$
2: **for** $d \leftarrow 1$ to $2^n - 1$ **do**
3: $\quad c \leftarrow \bigwedge_{i:d[i]=1} x_i \oplus \left(\bigwedge_{i:d[i]=1} x_i \wedge \bigvee_{i:d[i]=0} x_i\right)$
4: $\quad$ **for** $i \leftarrow 0$ to $l - 1$ **do**
5: $\quad\quad$ **if** $c[i] = 1$ **then** $P_i \leftarrow d$

Fig. 1. Algorithm for exponent partitioning.

Table 1
Example of exponent partitioning.

Index	7	6	5	4	3	2	1	0
x_1	1	0	1	0	0	1	1	1
x_2	1	0	1	0	1	1	1	1
x_3	1	1	1	0	0	0	1	1
C_{001}	0	0	0	0	0	0	0	0
C_{010}	0	0	0	0	1	0	0	0
C_{011}	0	0	0	0	0	1	0	0
C_{100}	0	1	0	0	0	0	0	0
C_{101}	0	0	0	0	0	0	0	0
C_{110}	0	0	0	0	0	0	0	0
C_{111}	1	0	1	0	0	0	1	1
P	7	4	7	0	2	3	7	7

$\{a_1, a_2, ..., a_n\}$ be an array of n integers. Given a bit b, $\bigwedge_{i:d[i]=b} a_i$ denotes the result of bitwise AND of all a_i's such that $d[i] = b$. $\bigvee_{i:d[i]=b} a_i$ is defined in the same way. Also the exponents are assumed to be randomly chosen non-negative integers which means that each bit in an exponent is 1 with probability $\frac{1}{2}$.

In a batch modular exponentiation, given g, p, and n l-bit exponents $X = \{x_1, x_2, ..., x_n\}$, we want to compute $R = \{R_i = g^{x_i} \bmod p \mid 1 \le i \le n\}$ simultaneously. We introduce three stages of the generalized intersection method with the decremental combination strategy. For the pictorial description, we refer readers to [1].

2.1. Exponent partitioning

In the generalized intersection method, the n exponents are partitioned to $2^n - 1$ disjoint cells by the following formula:

$$C_d = \bigwedge_{i:d[i]=1} x_i \oplus \left(\bigwedge_{i:d[i]=1} x_i \wedge \bigvee_{i:d[i]=0} x_i \right)$$

where $1 \le d < 2^n$ and $d = (d[n]d_[n-1]...d[1])_2$ for $d[i] \in \{0, 1\}$. Because of the bitwise mutual exclusiveness of the disjoint cells, the values can be represented as a position array P of length l where $P_i = d$ if the i-th bit of C_d is 1, and $P_i = 0$ otherwise. The position array P can be computed by Algorithm PART in Fig. 1.

Example 1. Let us consider a set X of three exponents $x_1 = 10100111$, $x_2 = 10101111$ and $x_3 = 11100011$ with $n = 3$ and $l = 8$. The disjoint cells and the position array for X are shown in Table 1. The bit at position 6 is set in C_{100} since C_{100} is composed of bits that are set in x_3, but not in x_1 and x_2. Likewise, the bit at position 2 is set in C_{011}, because bits in C_{011} are set in both x_1 and x_2, but not in x_3. As Table 1 shows, there is at most one cell for each bit column that is set to 1. For instance, the bit at position 5 is set to 1 only in C_{111}.

Algorithm EVAL

Input: g, p, position array P
Output: $G_d = g^{C_d} \bmod p$ for $0 < d < 2^n$
1: $G_i \leftarrow 1$ for $0 < i < 2^n$
2: **for** $i \leftarrow 0$ to $l - 1$ **do**
3: $\quad$ **if** $P_i \neq 0$ **then**
4: $\quad\quad G_{P_i} \leftarrow G_{P_i} \times g \bmod p$
5: $\quad g \leftarrow g \times g \bmod p$

Fig. 2. Algorithm for evaluation.

Algorithm COMB

Input: n, p, $G_d = g^{C_d} \bmod p$ for $0 < d < 2^n$
Output: $R_i = g^{x_i} \bmod p$ for $1 \le i \le n$
1: **for** $i \leftarrow n$ downto 1 **do**
2: $\quad R_i \leftarrow G_{2^{i-1}}$
3: $\quad$ **for** $j \leftarrow 1$ to $2^{i-1} - 1$ **do**
4: $\quad\quad R_i \leftarrow R_i \times G_{2^{i-1}+j} \bmod p$
5: $\quad\quad G_j \leftarrow G_j \times G_{2^{i-1}+j} \bmod p$

Fig. 3. Algorithm for decremental combination.

2.2. Evaluation

With the position array P, we can compute $G_d = g^{C_d} = \prod_{i:P_i=d} g^{2^i} \bmod p$ by the repeated squaring method as described in Algorithm EVAL of Fig. 2. All G_d's are initialized to 1. At the i-th iteration, we test if P_i is not 0. If so, g^{2^i} is multiplied to G_{P_i}, since P_i contains the index of the partitioned cell that has 1 for its i-th bit. In Example 1, g^{2^2} is multiplied to G_{011} and squared to g^{2^3}.

2.3. Decremental combination

Each value $R_i = g^{x_i} = \prod_{d:d[i]=1} g^{C_d} \bmod p$ is computed by combining the output values of the evaluation stage. To avoid redundant multiplications, Chung et al. [1] devised a decremental combination method where the final results are calculated from R_n down to R_1 (or any arbitrary order) as in Algorithm COMB of Fig. 3. For computing R_i, we initialize R_i to $G_{2^{i-1}}$, and multiply each evaluated value $G_{2^{i-1}+j}$ for $1 \le j < 2^{i-1}$ to R_i and G_j. After R_i is computed, the set of merged values G_d for $1 \le d < 2^{i-1}$ is equivalent to the output of the evaluation stage for the exponent set $\{x_1, ..., x_{i-1}\}$.

2.4. Performance

The expected number of multiplications of the evaluation stage in Algorithm EVAL is $(2^n - 1)\left(\frac{l}{2^n} - 1\right)$ and the number of squares is $l - 1$. Since the number of multiplications of the combination stage in Algorithm COMB is $2\sum_{i=1}^{n}(2^{i-1} - 1)$, the total cost is given by $\left(2 - \frac{1}{2^n}\right)l + 2^n - 2n - 2$. The amount of required memory in bits to store P and the evaluated values are nl