Fixed-base Exponentiation

André Weimerskirch
escrypt Inc.

Related concepts and keywords

binary exponentiation, k-ary exponentiation, windows exponentiation, fixed-exponent exponentiation, Diffie-Hellman key agreement

Definition

The exponentiation of a fixed base element $g \in G$, with G some group, by an arbitrary positive integer exponent e.

Theory

There are many situations where an exponentiation of a fixed base element $g \in G$, with G some group, by an arbitrary positive integer exponent e is performed. Fixed-base exponentiation aims to decrease the number of multiplications compared to general exponentiation algorithms such as the binary exponentiation algorithm. With a fixed base, precomputation can be done once and then used for many exponentiations. Thus the time for the precomputation phase is virtually irrelevant. Using precomputations with a fixed base was first introduced by Brickell, Gordon, McCurley, and Wilson (and thus it is also referred to as BGMW method) [1]. In the basic version, values $g_0 = g,\ g_1 = g^2, g_2 = g^{2^2}, \ldots, g_t = g^{2^t}$ are precomputed, and then the binary exponentiation algorithm is used without performing any squarings. Having an exponent e of bit-length $n + 1$, such a strategy requires on average $n/2$ multiplications whereas the standard binary exponentiation algorithm requires $3/2\ n$ multiplications. However, there is quite some storage required for all precomputed values, namely storage for $t + 1$ values. The problem of finding an efficient algorithm for fixed-base exponentiation can be rephrased as finding a short vector-addition chain for given base elements $g_0, g_1, \ldots, g_t$ (cf. fixed-exponent exponentiation). Note that there is always a trade-off between the execution time of an exponentiation and the number t of precomputed group elements.

In order to reduce the computational complexity, one might use a precomputed version of the k-ary exponentiation by making the precomputation phase only once. However, there is time saved by multiplying together powers with identical coefficients, and then raising the intermediate products to powers step by step. The main idea of the *fixed-base windowing method* is that $g^e = \prod_{i=0}^{t} g_i^{e_i} = \prod_{j=1}^{h-1}(\prod_{e_i=j} g_i)^j$ where $0 \leq e_i < h$ [1]. Assume that g^e is to be computed where g is fixed. Furthermore, there is a set of integers $\{b_0, b_1, \ldots, b_t\}$ such that any appropriate positive exponent e can be written as $e = \sum_{i=0}^{t} e_i b_i$, where $0 \leq e_i < h$ for some fixed positive integer h. For instance, when choosing $b_i = 2^i$ this is equivalent to the basic BGMW method described above. Algorithm 1 takes as input the set of precomputed values $g_i = g^{b_i}$ for $0 \leq i \leq t$, as well as h and e.

Algorithm 1 Fixed-base windowing method

```
INPUT: a set of precomputed values $\{g^{b_0}, g^{b_1}, \ldots, g^{b_t}\}$, the exponent $e = \sum_{i=0}^{t} e_i b_i$,
and the positive integer $h$
OUTPUT: $g^e$
1. $A \leftarrow 1$, $B \leftarrow 1$
2. For $j$ from $(h-1)$ down to 1 do
  2.1 For each $i$ for which $e_i = j$ do $B \leftarrow B \cdot g^{b_i}$
  2.2 $A \leftarrow A \cdot B$
3. Return $A$
```

This algorithm requires at most $t + h - 2$ multiplications, and there is storage required for $t + 1$ precomputed group elements. The most common version of this method is the case where the exponent e is represented in radix b, where b is a power of 2, i.e., $b = 2^w$. The parameter w is also called the window size. The exponent is written as $e = \sum_{i=0}^{t} e_i (2^w)^i$ or $(e_t \ldots e_1 e_0)_{2^w}$ where $t + 1 = \lceil n/w \rceil$ and $b_i = (2^w)^i$ for $0 \leq i \leq t$, and $h = 2^w$. Then on average there are $(t+1)(2^w - 1)/2^w + 2^w - 3$ multiplications required. Consider the following example [5] for $e = 862$ and $w = 2$, i.e., $b = 4$. Then $b_i = 4^i, 0 \leq i \leq 4$, such that the values g^1, g^4, g^{16}, g^{64}, and g^{256} are precomputed. Furthermore it is $t = 4$ and $h = 4$. The following table displays the values of A and B at the end of each iteration of Step 2:

A method to reduce the required memory storage for precomputation even further was proposed by Lim and Lee [6] which is called *fixed-base comb method*. Here, the binary representation of the exponent e is written in h rows such that there is a matrix EA (exponent array) established. Then v columns of the matrix are processed one at a time. Assume that the exponent is written as $e = (e_n \ldots e_1 e_0)_2$. Then select an integer h (the number of rows of EA) with $1 \leq h \leq n + 1$ and compute $a = \lceil (n+1)/h \rceil$ (the number of columns of EA). Furthermore, select an integer v (the number of columns

Table 1: Example for the windowing method.

j	-	3	2	1
B	1	$g^4g^{256} = g^{260}$	$g^{260}g = g^{261}$	$g^{261}g^{16}g^{64} = g^{341}$
A	1	g^{260}	$g^{260}g^{261} = g^{521}$	$g^{521}g^{341} = g^{862}$

of EA that are processed at once) with $1 \leq v \leq a$, and compute $b = \lceil a/v \rceil$ (the number of processing steps). Let $X = (R_{h-1}||R_{h-2}||\dots||R_0)$ be a bit-string formed from e by padding e on the left with 0's such that X has bit-length ah and such that each R_i is a bit-string of length a. Form the $h \times a$ array EA where row i of EA is the bit-string R_i. The fixed-base comb method algorithm has two phases. First, there is a precomputation phase that is done only once for a fixed base element g, and then there is the exponentiation phase that is done for each exponentiation. Algorithm 2 [5] describes the precomputation phase.

```
Algorithm 2 Fixed-base comb method - precomputation phase
INPUT: a group element $g$ and parameters $h, v, a,$ and $b$
OUTPUT: $\{G[j][i] : 1 \leq i < 2^h, 0 \leq j < v\}$
1. For $i$ from 0 to $(h-1)$ do
  1.1 $g_i \leftarrow g^{2^{ia}}$
2. For $i$ from 1 to $(2^h - 1)$ (where $i = (i_{h-1} \dots i_0)_2$), do
  2.1 $G[0][i] \leftarrow \prod_{j=0}^{h-1} g_j^{i_j}$
  2.2 For $j$ from 1 to $(v-1)$ do
    2.2.1 $G[j][i] \leftarrow (G[0][i])^{2^{jb}}$
3. Return $G[j][i]$ for $1 \leq i < 2^h, 0 \leq j < v$
```

Now let $I_{j,k}, 0 \leq k < b, 0 \leq j < v$ be the integer whose binary representation is column $(jb + k)$ of EA, where column 0 is on the right and the least significant bit of a column is at the top. Algorithm 3 displays the fixed-base comb method exponentiation phase.

```
Algorithm 3 Fixed-base comb method - exponentiation phase
INPUT: a group element $g$ and an exponent $e$ as well as the precomputed values $G[i][j]$
```

```
OUTPUT: $g^e$
1. $A \leftarrow 1$
2. For $k$ from $(b-1)$ down to 0 do
  2.1 $A \leftarrow A \cdot A$
  2.2 For $j$ from $(v-1)$ down to 0 do
   2.2.1 $A \leftarrow G[j][I_{j,k}] \cdot A$
3. Return $A$
```

The number of multiplications required in the computation phase is at most $a+b-2$ of which there are at least $b-1$ squarings. Furthermore, there is space required for $v(2^h-1)$ precomputed group elements. Note that the required computational complexity depends on h and v, i.e., on the available memory capacity for storing precomputed elements. In practice, values $h=4$ or 8 and $v=1$ or 2 offer a good trade-off between running time and memory requirements. Again, assume $e = 862 = (1101011110)_2$, i.e., $t=9$. Choose $h=3$ and thus $a=4$, and choose $v=2$ such that $b=2$. In the first phase Algorithm 2 is applied. Table 2 displays the precomputed values. Here, all possible values that might occur in a column of the EA matrix are precomputed. Note that the values of the second row are the values of the first one to the power of $a=4$ such that later on two columns can be processed at a time. Recall that $g_i = g^{2^{ia}}$.

Table 2: Example for fixed-base comb method precomputation.

i	1	2	3	4	5	6	7
$G[0][i]$	g_0	g_1	g_1g_0	g_2	g_2g_0	g_2g_1	$g_2g_1g_0$
$G[1][i]$	g_0^4	g_1^4	$g_1^4g_0^4$	g_2^4	$g_2^4g_0^4$	$g_2^4g_1^4$	$g_2^4g_1^4g_0^4$

Now form the bit-string $X = (\mathit{00}1101011110)$ with two padded zeros. Table 3 displays the exponent array EA. Note that the least significant bit of e is displayed in the upper right corner of EA and the most significant bit in the lower left corner.

Finally, Algorithm 3 is performed. The following table displays the steps of each iteration. Note that only the powers of the three base values g_i are displayed.

The last row of the table corresponds to $g^e = g_0^{l_0} g_1^{l_1} g_2^{l_2} = g^{14} g^{16\cdot 5} g^{256\cdot 3} = g^{862}$. The fixed-base comb method is often used for implementations as it

Table 3: Example for exponent array EA.

$a = 4$

$I_{1,1}$	$I_{1,0}$	$I_{0,1}$	$I_{0,0}$
$e_3 = 1$	$e_2 = 1$	$e_1 = 1$	$e_0 = 0$
$e_7 = 0$	$e_6 = 1$	$e_5 = 0$	$e_4 = 1$
0	0	$e_9 = 1$	$e_8 = 1$

$h = 3$

$v = 2$

$b = \lceil a/v \rceil = 2$

Table 4: Example for fixed-base comb method exponentiation.

$A = g_0^{l_0} g_1^{l_1} g_2^{l_2}$				
k	j	l_0	l_1	l_2
1	-	0	0	0
1	1	4	0	0
1	0	5	0	1
0	-	10	0	2
0	1	14	4	2
0	0	14	5	3

promises the shortest running times at given memory constraints for the precomputed values. A compact description of the algorithm can be found in [4].

Further examples and explanations can be found in [3, 5]. An improvement of the fixed-base windowing method, which is called *fixed-base Euclidean method*, was proposed by de Rooij [2]. However, in most situations the fixed-base comb method performs more efficient.

Applications

A popular application of fixed-base exponentation is in elliptic curve cryptography, for instance for Diffie-Hellman key agreement and ECDSA signature verification.

References

[1] E.F. Brickell, D.M. Gordon, K.S. McCurley, and D.B. Wilson. Fast exponentiation with precomputations. In *Proceedings of Eurocrypt '92*, LNCS 658, Springer-Verlag, 1992.

[2] P. de Rooij. Efficient exponentiation using precomputation and vector addition chains. In *Proceedings of Eurocrypt '94*, LNCS 950, Springer-Verlag, 1994.

[3] D.M. Gordon. A Survey of Fast Exponentiation Methods. In *Journal of Algorithms*, vol. 27, pp. 129–146, 1998.

[4] D. Hankerson, J. L. Hernandez and A. Menezes. Software Implementation of Elliptic Curve Cryptography Over Binary Fields. In *Proceedings of Cryptographic Hardware and Embedded Systems, CHES 2000*, LNCS 1965, Springer-Verlag, 2000.

[5] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[6] C. Lim and P. Lee. More flexible exponentiation with precomputation. In *Proceedings of Crypto '94*, LNCS 839, Springer-Verlag, 1994.