Multi-exponentiation

S.-M. Yen
C.-S. Laih
A.K. Lenstra

Indexing terms: Cryptographic protocols, Exponentiation

Abstract: In several cryptographic protocols the product of a small number of exponentiations is required, but the separate exponentiation results are not needed. A simultaneous exponentiation algorithm that takes advantage of this situation and that is substantially faster than the ordinary approach using separate exponentiations is presented.

1 Introduction

Many cryptographic protocols depend on modular exponentiation: given positive integers n and e and an element x in $G = \mathbb{Z}/n\mathbb{Z}$, compute $x^e \in G$. Efficient methods for modular exponentiation can be found in References 1–5 and the references in these papers.

In this note we consider multi-exponentiation: given some small integer $p > 1$, positive integers $e_1, e_2, \ldots, e_p$, and $x_1, x_2, \ldots, x_p \in G$, compute $y = \prod_{i=1}^{p} x_i^{e_i} \in G$. Cryptographic protocols using multi-exponentiation with $p = 2$ can be found in References 6–8 and with $p = 3$ in Reference 9. Evidently y can be determined by computing $x_i^{e_i} \in G$ for $i = 1, 2, \ldots, p$ using p ordinary modular exponentiations, followed by $p - 1$ multiplications in G. We describe a simple algorithm that is on average, and for e_i, of about the same order of magnitude, substantially faster than p separate exponentiations, at the expense of some memory. Our algorithm combines an idea from Shamir [Reference 9, Section V.B] with the 'sliding window' variant [1] of the 'm-ary method' [3, 10]. At the time of writing this note we found that a slightly less efficient variant of our algorithm had been found independently by Strauss [11].

Like other methods [1–5] our method attempts to minimise the number of multiplications in G. It works for any abelian group G, independent of its structure. We do not attempt to minimise the multiplication time in G.

2 Algorithm

Let $r \in \mathbb{Z}$ be maximal such that $2^r \leqslant \max\{e_i\}_{i=1}^{p}$ and let $(e_i(j))_{j=0}^{r}$ be the binary representation of e_i, for $i = 1, 2, \ldots, p$, i.e. $e_i = \sum_{j=0}^{r} e_i(j)2^j$ with $e_i(j) \in \{0, 1\}$. Shamir's idea is to prepare a table with all 2^p values $\prod_{i=1}^{p} x_i^{[0,1]}$, to put $y = \prod_{i=1}^{p} x_i^{e_i(r)}$ using one table look-up, and to replace, for $j = r - 1, r - 2, \ldots, 0$ in succession, y by $y^2 \cdot \prod_{i=1}^{p} x_i^{e_i(j)}$, using one squaring and at most one table look-up and multiplication per j.

This process takes $2^p - p - 1$ multiplications to prepare the table, r squarings, and on average $r(1 - 1/2^p)$ multiplications, all in G. The term $1/2^p$ is the probability that $e_i(j) = 0$ for $i = 1, 2, \ldots, p$, for some j. In the worst case this never happens and the number of multiplications becomes r.

Shamir's method is very similar to the ordinary 'square-multiply' method (for $p = 1$ it is actually the same) except that it uses the combined bit pattern of all p exponents to decide which element from the table should be multiplied in. It is well known that for large exponents the square-multiply method for ordinary exponentiation can be improved considerably on, at the expense of some memory, by using sliding windows [1]: for some window size $w \geqslant 1$ prepare a table of x^d for all odd positive ds of at most w bits, and use this table to compute x^e by scanning the bits of e at least w bits at a time, starting from the w most significant bits. For $w = 1$ this is the same as the square-multiply method. A more precise description of this algorithm, and the analysis of its average run time, can be obtained by taking $p = 1$ in the following combination of Shamir's idea and sliding windows.

2.1 Multi-exponentiation

To compute y, perform steps (*a*)–(*e*) in succession, unless indicated otherwise.

(*a*) Select an appropriate window size $w \leqslant r + 1$, and precompute all elements of the form $\prod_{i=1}^{p} x_i^{d_i}$ such that $0 \leqslant d_i < 2^w$ and at least one of the d_i is odd.

(*b*) Let $t = r - w + 1$ and let $\tilde{e}_i = \sum_{j=0}^{w-1} e_i(t + j)2^j$ be a span of w bits from the ith exponent, for $i = 1, 2, \ldots, p$. Determine the largest integer $s \geqslant 0$ such that 2^s divides $\tilde{e}_i$ for $i = 1, 2, \ldots, p$, and compute d_i with $\tilde{e}_i = 2^s \cdot d_i$. Note that at least one of the d_i is odd. Set $y = (\prod_{i=1}^{p} x_i^{d_i})^{2^s}$ using one table look-up followed by s squarings in G.

(*c*) Stop if $t = 0$. Otherwise, if $e_i(t - 1) = 0$ for $i = 1, 2, \ldots, p$, then replace t by $t - 1$ and y by y^2 and repeat step (*c*).

(*d*) If $t \geqslant w$ then replace t by $t - w$; otherwise, replace w by t and t by 0.

(*e*) Compute $\tilde{e}_i$, s and d_i as in step (*b*), and replace y by $(y^{2^{w-s}} \cdot \prod_{i=1}^{p} x_i^{d_i})^{2^s}$ using a total of w squarings, one table look-up and one multiplication. Go back to step (*c*).

Paper 1271E (C1, C14), first received 28th June 1993 and in revised form 24th January 1994

S.-M. Yen is with the Department of Computer Science, Chung-Hua Polytechnic Institute, 30, Tung Shiang, Hsin Chu, Taiwan, 30067, Republic of China

C.-S. Laih is with the Communication Laboratory, Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan, Republic of China

A.K. Lenstra is at Room MRE-2O334, Bellcore, 445 South Street, Mor-

The research of the first two authors is supported by the National Science Council, Republic of China under Grant NSC81-0408-E-006-02.

2.2 Run time analysis

In step (*a*) a total of $2^{pw} - (2^p)^{w-1} - p$ elements have to actually be computed. For $w > 1$ this can trivially be achieved in $2^{pw} - (2^p)^{w-1} - p$ multiplications in G after computation of the p squares x_i^2; for $w = 1$ no squarings are needed and $2^p - p - 1$ multiplications suffice.

The total number of squarings in G in steps (*b*) and (*e*) is at most r, where we assume the worst case of $s = w - 1$ squarings in step (*b*). Averaging over the e_is, we find for $w > 1$ that s in step (*b*) has the value 0 with the probability $(1 - 1/2^p)$, value 1 with the probability $(1 - 1/2^p)/2^p$, and, more generally, value $v < w - 1$ with probability $(1 - 1/2^p)/2^{pv}$, and value $w - 1$ with probability $1/2^{p(w-1)}$. This implies that the expected value for s in step (*b*) is bounded by

$$\sum_{v=0}^{\infty} \frac{v(1 - 1/2^p)}{2^{pv}} = \frac{1}{2^p - 1}$$

For $w > 1$ we find that the average number of squarings is bounded by $r - w + 1 + 1/(2^p - 1)$.

The total number of multiplications in G is at most $[r/w]$, where we assume that the $e_i(t-1)$ in step (*c*) are never simultaneously zero. If they are simultaneously zero, the window size w until the next execution of step (*e*) effectively increases by at least one, and maybe by more than one if step (*c*) is repeated again. (This possibility of temporarily having a larger window size characterises the sliding window method.) The average increment of the window size can be estimated as above by $1/(2^p - 1)$, and we find that the average number of multiplications is approximately $r/(w + 1/(2^p - 1))$.

From these estimates the best choice for w given r and p can easily by derived. For instance, for $r = 159$ and $p = 2$ as in the verification step of the NIST digital signature algorithm [7] we find that $w = 2$ gives a 37.5% speed-up compared to two separate ordinary sliding window exponentiations [12]. This is noticeably better than the Shamir method (i.e. $w = 1$) which leads for this same r to a speed-up of 25% over the ordinary exponentiations. For $r = 511$ and $p = 2$ a good choice is $w = 3$. The main difference between the proposed algorithm and the one in Reference 11 is the adoption of 'sliding windows', which require less computation and memory space in step (*a*) (i.e. the table preparation) and take fewer multiplications in step (*e*). Although the effect of sliding windows is asymptotically (for $r \to \infty$) negligible, in realistic examples the number of multiplications can be expected to be considerably less than for the method from [Reference 11]: for $w = 2$ and $p = 2$, for instance, sliding windows need on average 14% fewer multiplications.

3 References

1 BOS, J., and COSTER, M.: 'Addition chain heiristics', *in* 'Advances in cryptology, Crypto '89'. Lecture Notes in Computer Science **435**, (Springer-Verlag, 1990), pp. 400–407

2 DOWNEY, P., LEONY, B., and SETHI, R.: 'Computing sequences with addition chains', *SIAM J. Comput.*, 1981, **3**, pp. 638–696

3 KNUTH, D.E.: 'The art of computer programming, Vol. 2: semi-numerical algorithms' (Addison-Wesley, Reading, MA, 1981, 2nd edn)

4 YACOBI, Y.: 'Exponentiating faster with addition chains', *in* 'Advances in cryptology, Eurocrypt '90'. Lecture Notes in Computer Science, **473** (Springer-Verlag, 1991), pp. 222–229

5 YAO, A.: 'On the evaluation of powers', *SIAM J. Comput.*, 1976, **5**, pp. 100–103

6 BRICKELL, E.F., and McCURLEY, K.S.: 'Interactive identification and digital signatures', *AT&T Tech.J.*, 1991, pp. 73–86

7 'NIST: A proposed federal information processing standard for digital signature standard (DSS)'. Federal Register 1991, **56**, pp. 42980–42982

8 SCHNORR, C.P.: 'Efficient identification and signatures for smart cards', *in* 'Advances in cryptology, Crypto '89'. Lecture Notes in Computer Science, **435** (Springer-Verlag, 1990), pp. 239–252

9 GAMAL, T.E.: 'A public key cryptosystem and a signature scheme based on discrete logarithms', *IEEE Trans.*, 1985, **IT-331**, pp. 469–472

10 BRAUER, A.: 'On addition chains', *Bull. Am. Math. Soc.*, 1939, **45**, pp. 736–739

11 STRAUS, E.G.: 'Addition chains of vectors', *AMM*, 1964, **71**, pp. 807–808

12 LENSTRA, A.K.: 'Lip, a long integer package'. Available by anonymous ftp from flash. bellcore. com.