Addition Chain Heuristics

Jurjen Bos
Matthijs Coster

Centrum voor Wiskunde en Informatica
Kruislaan 413
1098 SJ Amsterdam
The Netherlands

Introduction

Much current research focuses on fast evaluation of RSA, which consists of computing powers modulo a large number n. While some try to increase the speed of multiplications, here we consider reducing the number of multiplications. In particular, we present a precomputation method that reduces the number of multiplications for the computation of a given power.

Although we speak about RSA, our method is also applicable to computing elements in other large cyclic groups, such as elliptic curves [Len86], and the computation of elements of the Fibonacci and Lucas chains [Wil82].

Theoretical results and asymptotic bounds in this area are plentiful (see [Dow81], e.g.), but we are not aware of anyone applying heuristics, as we do, to make chains useable in practice.

Definitions and notation

An *addition chain* for a given number is a list of numbers having the following properties:

- the first number is one;
- every number is the sum of two earlier numbers;
- the given number occurs in the chain (at the end, that is).

In the case of an addition *sequence*, the last condition becomes:

- the given numbers occur in the sequence.

We view such a list as a series of exponents used to do an exponentiation. The *length* of an addition chain or sequence is the number of elements in the chain, apart from the initial one.

For example, the standard (binary) addition chain [Knu69] for the number 15 has length 6:

```
1 2 3 6 7 14 15
```

There is, however, a chain of length 5 that produces 15:

```
1 2 3 6 12 15
```

This means that one can compute x^{15} from x in 5 multiplications.

Naturally we are interested in addition chains with as small a length as possible. Our method is capable of producing an addition chain for a 512-bit number of length 605 on average. This is an improvement of 21% over the binary algorithm (which has length 768 on average) and an improvement of 5% over Knuth's 5-window algorithm (see below).

Finally, we define a *vector addition chain* of a given vector as the shortest list of *vectors* with the following properties:

- the first vectors are [1, 0, 0, ..., 0], [0, 1, 0, ..., 0], ..., [0, 0, 0, ..., 1];
- each vector is the sum of two earlier vectors;
- the last vector is equal to the given vector.

We define the length of a vector addition chain as the number of vectors after the initial vectors.
Example:

```
[1 0 0] [0 1 0] [0 0 1] [1 0 1] [0 1 1] [1 1 2] [1 2 3]
  [2 3 5] [4 6 10] [6 9 15] [8 12 20] [14 21 35] [15 21 35]
```

is a vector addition chain of `[15 21 35]` with length 10.

We use the following simple functions:

$L(a_1, a_2, \ldots)$	a shortest addition sequence containing $a_1, a_2, \ldots$
$l(a_1, a_2, \ldots)$	length of a shortest addition sequence containing $a_1, a_2, \ldots$
$\nu(n)$	the number of ones that occur in the binary representation of n
$\log n$	${}^2\log n$

Brief review of the literature

Computing the shortest addition chain is an NP-complete problem [Dow81].
It is known that

$$\log n + \log \nu(n) - 2.13 \le l(n) \le \lfloor \log n \rfloor + \nu(n) - 1.$$

The lower bound is from [Sch75], and the upper bound is just the binary algorithm [Knu69]. [Bra39] gives an upper bound of

$$l(n) \le \log n + \log n / \log \log n + o(\log n / \log \log n).$$

Less is known about addition sequences. [Yao76] gives the bound:

$$l(a_1, a_2, \ldots, a_k) \leq \log a_k + c n \log a_k / \log \log a_k, \text{ where}$$

$$c \approx 2 + 4 / \sqrt{\log a_k}.$$

Here a_k is the largest number in the sequence.

There is a one-to-one correspondence between addition sequences and vector addition chains [Oli81]: An addition sequence of length l with k requested numbers can be converted to a vector addition chain of length $l + k - 1$ and dimension k, and vice versa. For example, the vector addition chain mentioned above can be mapped to the addition sequence 1 2 3 4 7 10 14 15 21 35.

The algorithm

The computation of an addition chain can of course be done every time the chain is needed. The algorithm must then be fast to be useful. In practice, though, it is often true that an RSA exponent is known a long time in advance and used very often.

For a given number n, the algorithm computes an addition chain. It consists of two parts. The first reduces the computation of an addition chain for n to the computation of an addition sequence that contains a given set of numbers which are much smaller than n. This is comparable to the algorithm mentioned in [Thu73b], but we use it for much bigger numbers. The second part produces a sequence for those numbers.

The Window method

The first method we demonstrate is derived from [Knu69]. The idea is to write the number in binary and split it in pieces (*windows*). With a window size of 1 it is the binary method.

As an example, we take 26235947428953663183191, and windows of width 5.

The window division then looks like this:

```
10110001110010000001110100101001110101000000010111 10000011111001100101010111
  11     7  1          29    5     29 1          23 1        31     25    21  3
```

The addition chain now consists of two parts:

- an addition sequence producing the needed window numbers
- a long part, consisting of doublings, with the addition of numbers from the first chain in the proper places.

The first part is:

1 2 3 4 5 7 11 16 21 23 25 29 31

We then take the first window and repeatedly square it. For each window, we do one extra multiplication to put it in place.

The resulting chain is as follows:

Length of sequence that computes intermediates:	12
Number of squarings needed:	71
Number of multiplies for the intermediates:	12 +
Total number of multiplies:	95

Knuth proposes to make a precomputed table containing all odd numbers up to, say, 5 bits. (Of course, it is more efficient to use an addition sequence producing only the needed numbers).

Because we use addition sequences instead of tables, we can also choose a much bigger window size. In this example, we will use 10. Note that the first window is bigger than 10 bits:

```
10110001110010000011101001010011101010000001011110000011111001100101010111
        5689              933      117          47          499       343
```

The addition sequence yielding the first part is:

```
1 2 4 8 10 11 18 36 47 55 91 109 117 226 343 434 489 499 933
  1422 2844 5688 5689
```

We get this chain:

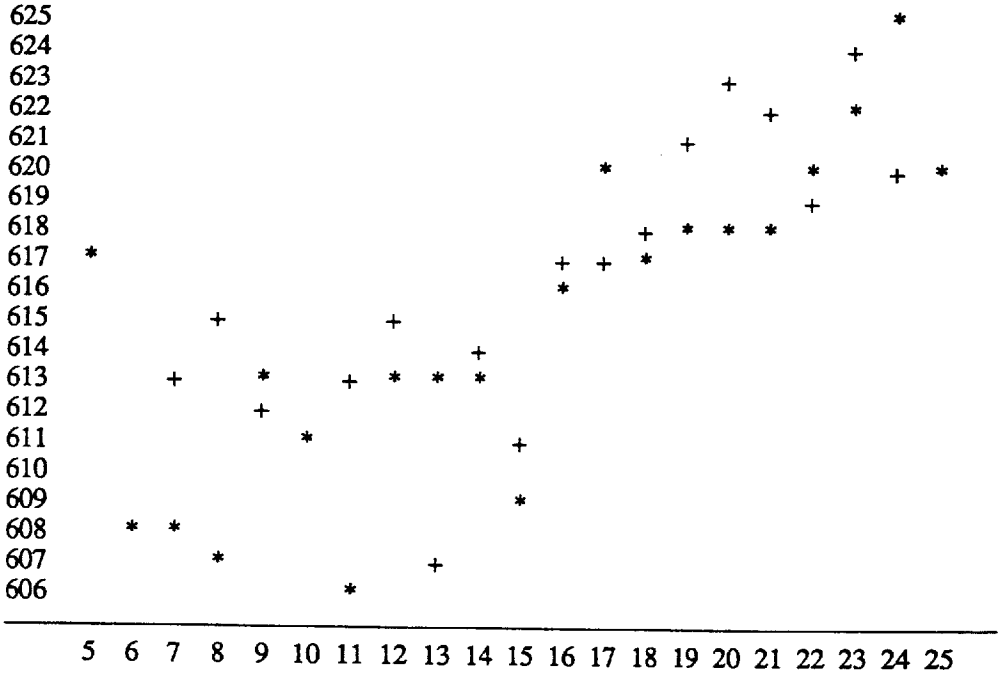
Length of sequence that computes intermediates:	22
Number of squarings needed:	62
Number of multiplies for the intermediates:	5 +
Total number of multiplies:	89

In general, it is the case that using addition sequences instead of tables allows one to use bigger window sizes, giving shorter addition chains for the original number. The graph on the next page shows an example of this effect.

The optimization of the window distribution generates a window distribution with the following properties:

- Each window is maximally the given size.
- There are as few windows as possible.
- The sum of the logarithms of the window values is minimal.

This can be done in linear time. (Actually, a trivial improvement can be done on the window division algorithm: instead of giving the window a maximal size, we give them a maximal *value*. This value can then be chosen as a prefix of the number. For example, the windows of a number that starts with `1101001...` will all be at most equal to `11010` for a window size of 5.)



This is an example of a random 512-bit number whose binary representation has about two thirds ones. The number of bits in one window is shown horizontally, and the total number of multiplications is shown on the vertical axis. The + signs indicates the length of the chain using a crude left-to-right method for assigning the windows. The * signs are obtained using a method that optimizes the window distribution.

The Makesequence algorithm

Here we describe a routine that makes an addition sequence of a set of numbers. It starts with a *protosequence* consisting of 1, 2 and the requested numbers. It then transforms this to another one using a heuristic algorithm. In each step, we reduce the protosequence to a simpler one, having smaller numbers.

We write such a protosequence as 1, 2, ..., f_2, f_1, f. The algorithm inserts some numbers by one of four methods and leave f. The four methods for making an addition sequence are called Approximation, Division, Halving and Lucas. The motivation to choose these four methods is that with these four methods it is possible to find all $L(a,b)$ for pairs (a, b) with $a < b \leq 50$. It is not obvious to find good sequences for the protosequences {4, 23} and {17, 48}. Good sequences can be found by Lucas and Division respectively. The question is which method has to bo chosen. This is a hard and at the moment unsolved question. We use a so called *weight function*, and we choose the method having the lowest related weight function.

The weight function will only be scetched. We found in all cases we tried that for protosequences $\{a_1,a_2,\ldots,a_n\}$ with $a_n \leq 1000$ the length of the sequence could be estimated by $l(a_1, a_2, \ldots,a_n) \leq \frac{3}{2} \log a_n + n + 1$. We use this bound in our weight function for the part of the protosequence which will remain after leaving f. In addition the number of inserted elements has to be added. Finally, in the function Δ we express the "hardness" of elements that have to be inserted.

Approximation

(There are two elements a and b in the sequence with $a+b = f-\varepsilon$, where ε is positive and small; insert $a+\varepsilon$.)

Condition:	$0 \leq f - (f_i + f_j) = \varepsilon$ (is "small"); $f_i \leq f_j$.
Insert:	$f_i + \varepsilon$.
Weight function:	$\frac{3}{2} \log(f_1) + k + 1 + \Delta(\varepsilon)$.
Example:	49 67 85 117
	$\varepsilon = 1$ (namely 117 – (49 + 67))
Insert:	50
Result:	49 <u>50</u> 67 85 (117)

Division

(f is divisible by a small prime p; put f/p, 2f/p, ..., f in the sequence.)

Condition:	p=3, 5, 9 or 17. f is divisible by p.
	$L(p) = \{1, \alpha_1 = 2, \ldots, \alpha_r = p\}$.
Insert:	f/p, $2f/p$, $\alpha_2 f/p$, ..., $\alpha_{r-1} f/p$
Weight function:	$\frac{3}{2} \log(f_i) + k + l(p) + i + \Delta(\max \delta(j))$.
Example:	17 48
Insert:	16 32
Result:	<u>16</u> 17 <u>32</u> (48)

Halving

(Take a (small) number s that occurs earlier in the sequence, and put f–s, (f–s)/2, (f–s)/4, ... to a certain point in the sequence.)

Condition:	$f/f_1 \geq 2^u$; $[f/2^u] = k$.
Insert:	$d = f - k \cdot 2^u, f - d = k2^u, k2^{u-1}, \ldots, k/2, k$.
Weight function:	$\frac{3}{2} \log(f_i) + u + 1 + \Delta(f_i - k)$.
Example:	14 382

	$f/f_1 = 27.2$; $u = 4$; $k = 23$; $d = 14$.
Insert:	23 46 92 184 368
Result:	14 23 46 92 184 368 (382)

Lucas ($u_{n+1} = u_n + u_{n-1}$)

(Put a Lucas sequence in the sequence that has f as last element.)

Condition:	f and f_i are the elements of a Lucas series (i.e. $f_i = u_0$ and $f = u_k$, $k \geq 3$).
Insert:	$u_1, u_2, \ldots, u_{k-2}, u_{k-1}$.
Weight function:	$\frac{3}{2}\log(f_i) + k + \lambda + i + \Delta(r)$.
Example:	4 23
Insert:	5 9 14
Result:	4 5 9 14 (23)

After applying one of those insertions, the process is repeated until the sequence contains only the numbers 1 and 2.

For the faster but less effective real-time version, we use only A and H to remove f. A simple rule decides which of these algorithms to take, and we do not use a weight function in this case.

Further work

We have considered using subtraction in a chain. In practice, this will not make the chain much shorter, while the cost of a subtraction in practice (eg., in RSA) is considerable.

We did not consider the effect of having a squaring operation that takes less time than a multiplication. Of course, if squaring is *very* cheap, we can use that

$$ab = [(a+b)^2 - (a-b)^2]/4$$

to reduce a multiplication to two squarings. However, from the example given earlier in this article one can see that our algorithm uses a lot of squarings. This makes our algorithm relatively more effective over the binary algorithm. It might be interesting to investigate this further.

The weight function we use does not give really satisfactory results. We have several ideas for improving the weight function.

Furthermore, we consider making a makesequence function that also tries steps that have an almost optimal weight, and tries out all possibilities, taking the shortest resulting chain. We tried using using simulated annealing (statistical cooling) [Laa87] methods, but until now we have not found a way to accomplish this. We expect that those improvements yield an algorithm that produces addition chain of length less than 600 for a 512-bit number.

References

[Bel63] R. Bellman: *Advanced problem 5125*, Amer. Math. Monthly **70** (1963), 765.

[BBBD] F. Bergeron, J. Berstel, S. Brlek, C. Duboc: *Addition Chains Using Continued Fractions*, Journal of Algorithms **10**, (1989), 403-412.

[Bra39] A. Brauer: *On addition chains*, Bull. Am. Math. Soc. **45** (1939), 736-739.

[Cot73] A. Cottrell: *A lower bound for the Scholz-Brauer problem*, Notices AMS, Abst. #73T-A200, **20** (1973), A-476.

[Dob80] D. Dobkin and R. J. Lipton: *Addition chain methods for the evaluation of specific polynomials*, Siam J. Comput. **9** (1980), 121-125.

[Dow81] P. Downey, B. Leony and R. Sethi: *Computing sequences with addition chains*, Siam Journ. Comput. **3** (1981), 638-696.

[Erd60] P. Erdős: *Remarks on number theory III, on addition chains*, Acta Arith. **6** (1960), 77-81.

[Fia89] A. Fiat: *Batch RSA*, Abstracts Crypto '89, to be published.

[Gio78] A. A. Gioia and M. V. Subbarao: *The Scholz-Brauer problem in addition chains II*, Proc. eighth Manitoba conference on numerical math. and computing, (1978),251-274.

[Knu69] D. E. Knuth: *The art of computer programming*, Vol. 2, Seminumerical algorithms, Addison-Wesley, Reading, Mass., (1969), pp. 398-419.

[Laa87] P. J. van Laarhoven and E. H. L. Aarts: *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Company, Dordrecht, 1987.

[Lee77] J. van Leeuwen: *An extension of Hansen's theorem for star chains*, J. Reine Angew. Math. **295** (1977), 203-207.

[Len86] H. W. Lenstra, jr.: *Factoring integers with elliptic curves*, Report 86-18, Universiteit van Amsterdam, 1986.

[Oli81] J. Olivos: *On Vectorial Addition Chains*, J. of Algorithms **2** (1981), 13-21.

[Sch75] A. Schönhage: *A lower bound on the length of addition chains*, Theoret. Comput. Sci. **1** (1975), 1-12.

[Thu73a] E. G. Thurber: *The Scholz-Brauer problem on addition chains*, Pacific J. Math. **49** (1973), 229-242.

[Thu73b] E. G. Thurber: *On addition chains $l(mn) \leq l(n)-b$ and lower bounds for $c(r)$*, Duke Math. J. **40** (1973), 907-913.

[Thu76] E. G. Thurber: *Addition chains and solutions of $l(2n) = l(n)$ and $l(2^n-1) = n+l(n)-1$*, Discr. Math. **16** (1976), 279-289.

[Tsa87] Y. H. Tsai and Y. H. Chin: *A study of some addition chain problems*, Intern. J. Comp. Math. **22** (1987), 117-134.

[Veg75] E. Vegh: *A note on addition chains*, J. Comb. Th. (A) **19** (1975), 117-118.

[Vol85] H. Volger: *Some results on addition/subtraction chains*, Inf. Proc. Lett. **20** (1985), 155-160.

[Wil82] H. C. Williams, *A $p+1$ method of factoring*, Math. Comp. **39** (1982), 225-234

[Yao76] Andrew Yao, *On the evaluation of powers*, Siam J. Comput. **5**, (1976).