Introduction

Exponentiation of large integers (modulo a large integer) is the basis of several well known cryptographic algorithms such as RSA, [1]. The calculations involved are complex, and can be time-consuming especially when performed in software. As a result algorithms which speed up software implementations of modular exponentiation are of considerable practical significance; see, for example, Selby and Mitchell, [2].

The generally accepted method for performing modular exponentiation is the 'square and multiply' technique; see, for example, Beker and Piper, [3], or Knuth, [4]. In brief, if one is required to compute

$$m^e \pmod N$$

and e has binary representation

$$e_{s-1}e_{s-2}\ldots e_0$$

where e_{s-1} is the most significant bit, then the 'left to right' version of the algorithm works as follows:

```
x := 1;
for i := s-1 to 0 do {
  x := x^2 (mod N);
  if ( e_i = 1 ) x = x * m (mod N) }
```

The result will be contained in x.

It is worth observing at this point that the number of modular multiplications by m involved in performing the algorithm is

determined by the number of 1s in the binary representation of *e*. The purpose of this paper is to show how the use of an alternative representation of *e*, in combination with a slight variant of the above algorithm, can considerably reduce the number of multiplications involved.

MSD representations

Suppose that, in a binary representation of a number, entries of -1 are allowed (in addition to 0 and +1). Then any number has many such (*modified signed-digit, or MSD*) representations; for other applications of such representations, including the construction of ripple-free adders, see, for example, Drake et al.,[5]. For example, the number 23 has unique binary representation

10111

but has various MSD representations such as:

++00-, +0-00- and +-+-++

where + represents +1 and - represents -1.

Now suppose that *e* has an MSD representation

$f(s)\ f(s\text{-}1)\ \ldots.\ f(0)$

where $f(s)$ is the most significant digit. Then a modified 'left to right' version of the square and multiply algorithm works as follows:

```
x := 1;
for i := s to 0 do {
  x := x^2 (mod N);
  x = x * m^f(i) (mod N)  }
```

Checking the validity of the modified algorithm is straightforward. It should now be clear that the number of modular multiplications in the revised algorithm depends on the number of non-zero entries in the MSD representation of e which we are using (which we call the *weight* of this representation of e). This can be significantly smaller then the binary weight of e offering considerable performance advantages, given that the pre-computation of m^{-1} can be performed efficiently. Computing m^{-1} uses the Euclidean algorithm which, fortunately, can be made to run very quickly.

The remainder of this paper is concerned with describing a method for finding a minimum weight MSD representation of a number. This method we call the *Weight Minimisation Algorithm (WMA)* and works as follows. Suppose e has an s-bit MSD representation stored in variables

$e[s-1], e[s-2], ..., e[0]$.

Note that the algorithm possibly requires the use of an additional variable $e[s]$, initially set to 0:

1. If consecutive pairs of non-zero elements remain, let i be the least integer for which $e[i]$ and $e[i+1]$ are non-zero. Otherwise terminate.

2. If e[i] $\neq$ $e[i+1]$ then set $e[i] := e[i+1]$ and set $e[i+1] := 0$ and go to step 1.

3. Otherwise let j be the least integer for which $e[j] \neq e[i]$ and $e[j-1] = e[j-2] = \ldots = e[i]$. If $e[j] = 0$ then set $e[j] := e[i]$, else set $e[j] := 0$. Set $e[i] := -e[i]$ and $e[k] := 0$ ($i < k < j$). Go to step 1.

Clearly each step of the algorithm produces an equivalent MSD representation of the number e; moreover the algorithm terminates because the value of i determined in step 1 strictly increases after obeying either step 2 or step 3. It is also important to observe that each step of the WMA either leaves the weight unchanged or reduces it, i.e. the representation output from the WMA has weight less than or equal to the weight of the input representation.

Sparse MSD representations

An MSD representation is said to be *sparse* if no two adjacent entries are non-zero.

Lemma 1: The MSD representation produced by using the WMA will always be sparse.

Proof: This is immediate by examination of the WMA.

Lemma 2: If the positive integer e has a sparse (s+1)-bit MSD representation (with leading entry non-zero), then

$$2^s - d_s \leq e \leq 2^s + d_s$$

where

$$d_s = 2^{s-2} + 2^{s-4} + \ldots$$

Proof: Since e is positive, then the leading term in any MSD representation must be +1. The largest sparse representation for a given number of bits will always have the form

+0+0+0...

and the smallest such representation will have the form

+0-0-0...

and the Lemma follows.

Lemma 3: Every integer has a unique sparse MSD representation.

Proof: Suppose e has two sparse representations ($f(i)$) and ($g(i)$) of s+1 and t+1 bits respectively (in each case with leading entry non-zero). Suppose, without loss of generality that $f(s) = 1$ (and hence e is positive). We now show that $t = s$ and $g(s) = 1$.

Now since ($f(i)$) is sparse, by Lemma 2 e satisfies

$$2^s - d_s \leq e \leq 2^s + d_s$$

Clearly $g(t) = 1$ since e is positive. By the same argument as above:

$$2^t - d_t \leq e \leq 2^t + d_t.$$

Now observe that

$$d_s + d_{s+1} = 2^s - 1$$

and hence

$$2^s + d_s < 2^{s+1} - d_{s+1}$$

and so $s = t$.

The Lemma now follows by induction (modify (f(i)) and (g(i)) by changing the leading term in both to zero).

Lemmas 1 and 3 now give the desired result:

Theorem: The representation generated by the WMA has the minimum possible weight.

Proof: Suppose this is not true, i.e. suppose e has an MSD representation ($g(i)$) which has weight less than the unique (by Lemma 3) sparse representation. By Lemma 1, applying the WMA will generate a sparse representation with weight less than or equal to the weight of ($g(i)$). This immediately gives a contradiction by Lemma 3.

Summary and acknowledgement

From the above discussion it should be clear that the Weight Minimisation Algorithm generates an MSD representation of

minimum weight. This in turn enables the number of modular multiplications to be minimised when performing a modular exponentiation. This offers the possibility of significantly improving the performance of software implementations of RSA.

We would like to acknowledge the contribution of Alex Selby, who originally suggested the idea of using an MSD representation in modular exponentiation.

References

[1] RIVEST, R.L., SHAMIR, A., and ADLEMAN, L.: 'A method for obtaining digital signatures and public-key cryptosystems', *Commun. ACM*, **21** (1978) 120-126.

[2] SELBY, A., and MITCHELL, C.J.: 'Algorithms for software implementations of RSA', *IEE Proceedings Part E*, **136** (1989) 166-170.

[3] BEKER, H.J., and PIPER, F.C.: 'Cipher systems' (van Nostrand, London, 1982).

[4] KNUTH, D.E.: 'The art of computer programming, Volume 2: Seminumerical algorithms' (Addison-Wesley, USA, 1981, 2nd edition).

[5] DRAKE, B.L., BOCKER, R.P., LASHER, M.E., PATTERSON, R.H. and MICELI, W.J.: 'Photonic computing using the modified signed-digit number representation', *Optical Engineering*, **25** (1986) 38-43.