3.2 Modular Exponentiation

In order to compute the modular exponentiation M^e **mod** N for cryptography applications, the exponentiation M^e is not computed first, rather a part of the exponentiation operation, usually a partial multiplication, is performed. Then the partial result is reduced modulo N at each step. This is simply because the exponentiation result of large numbers is very large. As an instance, the result of M^e when M and e are 256-bit integers, requires about 2^{256} bits which is larger than the amount of total articles in the universe. Similar to the exponentiation procedure which performs a successive multiplications, modular exponentiation uses the same procedure as exponentiation, but it performs modular multiplication. Therefore the exponentiation methods can be used for modular exponentiation without loss of generality. The next section briefly describes the exponentiation algorithms, further details of which can be found in the appropriate references.

3.2.1 Binary Methods

The two commonly used algorithms which are for hardware implementation of exponentiation are the left-to-right and right-to-left algorithms. These algorithms, which are also called the square and multiply method, perform exponentiation by repeated squaring and multiplication.

Left-to-Right algorithm: For the integers M, N, C and $e = \{e_{n-1}e_{n-2}...e_0\}$, $e_i \in \{0,1\}$, the following algorithm computes $C = M^e$ **mod** N:

The Left-to-Right Exponentiation Algorithm

```
C := 1;
for i = n-1 downto 0 loop
      C := $C^2$ mod N;
      if $e_i$ = 1    then   C := M.C  mod N;
return C;
```

where $C = M^e$ **mod** N. According to Knuth [Knu98], this method was known as early as 200 B.C. by Indian mathematicians. However, a clear discussion of how to compute 2^n efficiently for arbitrary n was given by al-Uqlidisi, an Islamic mathematician, in 952 A.D. Beginning initially with C = 1, this algorithm sets C to C^2 and then, if $e_i = 1$, sets C to C.M. The exponent bits are scanned from left to right and the zeros before the MSB are ignored. The following binary representation of the exponent e can be used for validity of this algorithm as:

$$e = \sum_{i=0}^{n-1} e_i.2^i = ((\ldots((e_{n-1}).2 + e_{n-2}).2 + \ldots).2 + e_1).2 + e_0$$

Therefore the exponentiation can be expressed as:

$$M^e = M^{((\ldots((e_{n-1}).2 + e_{n-2}).2\ldots.).2 + e_1).2 + e_0)} = ((\ldots((M^{e}{}_{n-1})^2)^2.M^{e}{}_{n-2})^2.M^{e}{}_1)^2.M^{e}{}_0$$

For example:

$$M^{23} = M^{(10111)b} = (((((1)^2.M)^2)^2.M\ )^2.M)^2.M$$

The left-to-right algorithm involves $\lfloor \log_2{}^e \rfloor$ squarings and $\nu(n)$ - 1 multiplications (or $\lfloor \log_2{}^e \rfloor$ + $\nu(n)$ -1 multiplications if both operands use the multiplier) when both operations on the initial 1 are ignored. $\nu(n)$ is the number of non-zero bits in the binary representation of e. In a hardware implementation, this algorithm requires one storage register to keep the intermediate result.

The right-to-Left algorithm: for i from 0 up to k-1, this algorithm first sets C to C.M if $e_i = 1$ and then sets M to M^2.

The algorithm can be also written for modular exponentiation as follows :

The Right-to-Left Exponentiation Algorithm

```
C   := 1;
for i = n-1 downto 0 loop
       IF $e_i$ = 1  THEN   C := M.C   mod N;
       M :=  $M^2$  mod N;
return   C;
```

where $C = M^e$ **mod** N. According to Knuth [Knu98], al-Kashi an Iranian mathematician stated this algorithm about 1414 A.D. The method is closely related to the multiplication procedure used by Egyptian mathematicians as early as 1800 B.C and widely used in Russia in the nineteenth century. It is often called "Russian peasant method" of multiplication.

This method is easily justified by a consideration of the sequence of exponents in the calculation. Since:

$$e = \sum_{i=0}^{n-1} e_i.2^i = 2^{n-1}.e_{n-1} + 2^{n-1}.e_{n-1} +...+2^1.e_1 +2^0.e_0$$

The exponentiation can be written as:

$$M^e = M^{((...((e_0 + 2^1 .e_1 +....+ 2^{n-2} .e_{n-2} +2^{n-1} .e_{n-1})} = (M^{e_0}).(M^{2.e_1})....(M^{2^{n-2} . e_{n-2}}).(M^{2^{n-1} .e_{n-1}})$$

$$M^e = (M)^{e_0}.(M^2)^{.e_1}....(M^{2^{n-2}})^{e_{n-2}}.(M^{2^{n-1}})^{e_{n-1}}$$

and it corresponds to the algorithm procedure.

For example $M^{23} = M^{(10111)b} = (((((1 . M) . M^2) . M^4)) .M^{16})$.

Ignoring multiplication by the initial 1 and also ignoring the last squaring which will not be used, the number of multiplications required is $\lfloor \log_2 e \rfloor + \nu(n) - 1$. Squaring and multiplication can be performed in parallel in this method [Riv84] [Kor95] [Poc98]. Two extra registers are needed to keep the partial result and the squaring result.