3.3 Modular Multiplication

Modular exponentiation can be performed by implementing a modular multiplication routine which can accomplish a modular squaring as well. In general a modular multiplication such as S = A.B **mod** N consists of two operations: multiplication and modular reduction. So there are two approaches to compute a modular multiplication as follows:

1. First multiplication is completed and then modular reduction are performed.
2. Interleaving of multiplication and modular reduction steps.

There are only a few instances of using the first approach in cryptography applications because of the great increase in the operands' size. The interleaving approach is commonly used and is more suitable as it avoids increasing the size. The operations in this approach may be performed in either binary (low radix level) or in high radix level.

Multiplication: The conventional algorithm for multiplication is "the add-shift algorithm" which is the same method as the hand-paper method. In this algorithm, the multiplier bits are scanned one digit per step, and then this digit is multiplied by the multiplicand followed by adding into the partial product. To form the final result, the partial product is shifted each step. The multiplication procedure usually starts by scanning the MSB (or MSDigit) first for modular reduction of Euclid's division type and the LSB (or LSDigit) first for modular reduction of Hensel's division type as described later in this chapter.

Modular reduction (MR) and division: As has been explained, the modular reduction on large numbers is one of the basic operations in public-key cryptography. Generally this operation can be perfomed using division. Although there are conventional algorithms for the hardware implementation of division (such as the restoring and the non-restoring algorithms [Omo94]) other techniques have been developed to deal with large operands in cryptography applications. These techniques result in a high performance implementation of cryptosystems.

However, a simple description for the modular multiplication A.B **mod** M for A = $\{a_{n-1}.r^{n-1}+a_{n-2}.r^{n-2}+...+a_0\}$ can be given by the following algorithm:

A General Algorithm for Modular Multiplication:

```
      S(0) := 0;
            for i = n-1 downto 0 loop
L1:               T := r.S(i) + a_i.B;
L2:               S(i+1) := T mod M;
      return S(n);
```

where S(n) = A.B **mod** M. In each iteration, a partial multiplication is performed in L1 followed by a modular reduction in L2.

For a better description of the techniques developed for modular reduction, two classifications are introduced: (i) Euclid's division versus Hensel's division; (ii) multiprecision techniques versus binary techniques. These classifications are only to facilitate providing a clear description of the reduction methods and they are just used here in the context of this chapter.

Division types: Euclid's division and Hensel's division

The conventional methods for division are classified by Shand and Vuillemin in [SV93] as Euclid's division versus Hensel's division.

Euclid's division: To divide P = $\{p_{(n+l)-1}p_{(n+l)-2}... p_0\}$ by N, this class accomplishes division based on performing the following:

$$P = Q.N + S(n)$$

where Q is quotient and S(n) < N. In this class the following recursive procedure is executed:

$$S(i+1) = (r.S(i) + p_{l-i}) - q_i.N$$

i is the recursion index, q is quotient digit, r is radix, S(0) = $\{p_{(n+l)-1}p_{(n+l)-2}... p_{l+1}\}$ is the high-order part of the dividend P (initial partial remainder) and S(n) is the final remainder. This class scans the dividend digits left-to-right (MSDigits first). Moreover the partial result is shifted to the left each iteration.

Hensel's division: Hensel introduced a division class in which division is based on performing the following [SV93]:

$$R.P = Q.N + S(n)$$

where Q is quotient, R is r^n and $S(n) < 2N$. The procedure which is recursively executed in this class is as:

$$S(i + 1) = ((S(i) + s_{n+i}) + q_i .N) \textbf{ div } r$$

where $S(0) = \{p_{n-1}p_{n-2}... p_0\}$ is the low-order part of the dividend (initial partial remainder) and S(n) is the final remainder. The Montgomery reduction method is the important member of this class for cryptographic applications. The dividend digits are scanned in this class right-to-left (LSDigits first). Moreover, the partial result is shifted to the right each iteration. A description of the binary algorithms for these two classes is given in [SV93].

Binary techniques versus Multiprecision techniques

Multiprecision techniques: There are techniques developed for modular reduction which are based on multiprecision division techniques. They are originally more suitable for software implementation, but they have been modified to fit hardware implementation as well.

Binary techniques: In this chapter, the binary class is used to refer to those techniques in which bit level operations perform modular reduction. This class is more suitable for hardware implementation than the "opposite" class, multiprecision techniques. A high radix version of the binary techniques in this class is taken into account as a generalisation of the binary technique. So they are still classified as binary techniques in this thesis.

3.5 MR Methods Based on Hensel's Division

Since Montgomery's method is used for implementing the RSA algorithm in this thesis, modifications to this method are examined in more detail in the following sub-sections.

3.5.1 The Montgomery Reduction Algorithm

In 1985 Montgomery presented a method for modular reduction without division by modulus M [Mon85]. In fact the algorithm performs the function $T.R^{-1}$ **mod** M (instead of T **mod** M) where R is a radix (preferably a power of 2) and M and R are relatively prime integers. The following algorithm can perform the modular reduction $T.R^{-1}$ **mod** M where $R.R^{-1} - M.M' = 1$.

The Montgomery Reduction Method

```
M' := -M^-1 mod R
Begin
      Q := (T mod R).M' mod R;
      S := (T + Q.M)/R;
      if  S ≥ M  then  return  S - M  else return  S;
End
```

The basic idea in Montgomery's algorithm is to make T a multiple of R by adding multiples of M. To validate this method, notice that

$$Q.M \equiv (T.M').M \bmod R$$
$$\equiv -T \bmod R$$

So (T + Q.M) is divisible by R because T + Q.M = T + (-T **mod** R) and S is an integer. Then:

$$S.R \equiv (T + Q.M) \bmod M$$
$$\equiv T \bmod M$$

So: $S \equiv T.R^{-1} \bmod M$

S also satisfies in $0 < S < 2M$ because $0 \leq T + Q.M < R.M + R.M$ and this results in $0 < t = (T + Q.M)/R < 2M$).

This description of the Montgomery's method can be used for software implementation and a detailed discussion about its development are given in [KAK96]. However, the Montgomery method has the disadvantages of pre- and post-processing requirements. The term $-M^{-1}$ needs

to be precomputed and the final result is required to be converted from TR^{-1} **mod** M to T **mod** M.

A description about the binary method was given by Montgomery for hardware implementation of the modular multiplication A.B **mod** M in [Mon85] and it can be written as the following algorithm:

The Binary version of Montgomery's Modular Multiplication

```
S(0) := 0;
Begin
      for i = 0 to n-1 loop
            qi := (S(i) + ai.B) mod 2;
            S(i+1) := (S(i) + ai.B + qi.M) / 2;
      if  S(n) ≥ M  then
            return  S(n) - M  else return  S;
End
```

This method has been developed in many research efforts as discussed next.