Modular Arithmetic Implementation with the Residue Number System (RNS)

Samuel Antão and Leonel Sousa
INESC-ID, Department of Electrical and Computer Engineering
Email: {samuel.antao,leonel.sousa}@inesc-id.pt

September 20, 2012

The main operations to be supported in a Modular Arithmetic (MA) algorithm are the addition/subtraction, multiplication and reduction as well as the conversion of Residue Number System (RNS) from/to binary. Nevertheless, these two last operations are far more complex and are thoroughly discussed in the following.

1 RNS Forward/Reverse Conversions

Forward conversion corresponds to the conversion of the input data to the RNS representation at the program's startup. The conversion of an input X is usually accomplished by computing for every channel e of the basis B_i: $x_{e,i} = X \bmod m_{e,i}$. Because, in general, $X \geq 2^k$, for the forward conversion the following should be computed for the channel e:

$$x_{e,i} = X \bmod m_{e,i} = \left(\sum_{j=1}^{\lceil S_X/k \rceil} {}^{k}X_j 2^{k(j-1)} \right) \bmod m_{e,i} = \tag{1}$$
$$= \sum_{j=1}^{\lceil S_X/k \rceil} {}^{k}X_j \left(2^{k(j-1)} \bmod m_{e,i} \right) \bmod m_{e,i}$$

where S_X is the size of X in bits and ${}^{k}X_j$ is the j-th word of the binary representation of X split in k-bit words. Considering (1), in order to obtain the RNS representation of X, the constants $2^{k(j-1)} \bmod m_{e,i}$ can be precomputed and stored in the accelerator and the conversion is accomplished with a few RNS channel modular multiply-and-accumulate operations.

For the reverse conversion, which means converting the computation results back to the binary representation, several approaches exist [3, 5, 7]. In the proposed framework, the conversion presented in [5] is adopted since it is suggested to be less complex and require lower dynamic ranges (smaller moduli sets) [1,4]. This conversion is based on the Chinese Remainder Theorem (CRT) that states

for the RNS representation of X in h_i channels e of a given basis B_i:

$$X = \sum_{e=1}^{h_i} \xi_{e,i} M_{e,i} - \alpha M_i, \tag{2}$$
$$\alpha < n,\ \xi_{e,i} = \left| \frac{x_{e,i}}{M_{e,i}} \right|_{m_{e,i}},\ M_{e,i} = \frac{M_i}{m_{e,i}}.$$

In order to the conversion be successful, the value of α in (2) should be computed such that $0 \leq X < M$. The method proposed in [5] involves a fixed point successive approximation approach. This method observes that for $X < M$, therefore $\frac{X}{M} < 1$, and (2) can be rewritten as:

$$\sum_{e=1}^{h_i} \frac{\xi_{e,i}}{m_{e,i}} = \alpha + \frac{X}{M_i} \Rightarrow \alpha = \left\lfloor \sum_{e=1}^{h_i} \frac{\xi_{e,i}}{m_{e,i}} \right\rfloor. \tag{3}$$

Since (3) requires costly divisions by $m_{e,i}$, an approximation ($\hat{\alpha}$) is suggested to this expression:

$$\hat{\alpha} = \left\lfloor \sum_{e=1}^{h_i} \frac{\text{trunc}_t(\xi_{e,i})}{2^k} + \beta \right\rfloor = \left\lfloor \sum_{e=1}^{h_i} \frac{\text{trunc}_t(\xi_{e,i})}{2^{k-t}} + \Phi \right\rfloor / 2^t, \tag{4}$$

where $\text{trunc}_t(\xi_{e,i})$ sets the $k-t$ least significant bits of $\xi_{e,i}$ to zero, with $t < k$, and $\Phi = \lfloor 2^t \beta \rfloor$. The parameter β (and consequently Φ) is a corrective term that should be carefully chosen such that $\alpha = \hat{\alpha}$. A set of inequalities that allow choosing good values for β were stated, supported on the maximum initial approximation errors:

$$\epsilon = \max_e \left(\frac{2^k - m_{e,i}}{2^k} \right),\ \delta = \max_e \left(\frac{\xi_{e,i} - \text{trunc}(\xi_{e,i})}{m_{e,i}} \right). \tag{5}$$

In [5], Theorems 1 and 2 are stated concerning the computation of the value $\hat{\alpha}$, its relationship with β, the number of channels h_i, and the errors stated in (5):

Theorem 1. *If* $0 \leq h_i(\epsilon + \delta) \leq \beta < 1$ *and* $0 \leq X < (1-\beta)M_i$, *then* $\alpha = \hat{\alpha}$;

Theorem 2. *If* $\beta = 0$, $0 \leq h_i(\epsilon + \delta) \leq 1$ *and* $0 \leq X \leq M_i$, *then* $\hat{\alpha} = \alpha$ *or* $\hat{\alpha} = \alpha - 1$.

Theorem 2 refers to the computation of a non-exact, although bounded, value of $\hat{\alpha}$ that allows a non-exact RNS to binary conversion, while Theorem 1 refers to an exact conversion. Considering that the basis B_i one wants to convert from has m_{min} as the minimum value of $m_{e,i} = 2^k - c_{e,i}$, the value $\Delta = h_i(\epsilon + \delta)$ used in the definition of Theorems 1 and 2 can be obtained as:

$$\Delta = h_i(\epsilon + \delta) = h_i \left(\frac{2^k - m_{min}}{2^k} + \frac{2^{k-t} - 1}{m_{min}} \right) = \tag{6}$$
$$= \frac{h_i}{m_{min}} \left(2^k + 2^{k-t} - m_{min} - 1 \right).$$

Summarizing, Theorem 1 allows obtaining suitable values of Φ in (4) such that $\alpha = \hat{\alpha}$, and Theorem 2 bounds the maximum error of $\hat{\alpha}$ (+1) if Φ is set to zero, thus an extra term M_i may exist in (2).

Algorithm 1 RNS channel addition/subtraction.

Require: $a_{e,i}$, $b_{e,i}$ and $m_{e,i}$.
Ensure: $r_{e,i} = a_{e,i} \pm b_{e,i} \bmod m_{e,i}$.
1: $r_{e,i} = a_{e,i} \pm b_{e,i}$;
2: **if** $r_{e,i} \geq m_{e,i}$ or $r_{e,i} < 0$ **then** $r_{e,i} = r_{e,i} \mp m_{e,i}$;
3: **return** $r_{e,i}$.

Another challenge concerning the reverse conversion is the mapping of this conversion to the resources assigned to the channel arithmetic in order to avoid the utilization of a dedicated unit for this purpose. In other words, the challenge is to map the reverse conversion to the k-bit arithmetic used in each one of the RNS channels. Each channel e is able to compute the values $\xi_{e,i} = \left| x_{e,i} \left| M_{e,i}^{-1} \right|_{m_{e,i}} \right|_{m_{e,i}}$ and consequently a value of α can be computed by accumulating the contributions of each channel by using the method in (4). Each channel can also contribute to the final value in (2) if the constants ${}^{k}\left(M_{e,i}\right)_{j}$ such that $M_{e,i} = \sum_{j=1}^{S_{M_{e,i}}/k} {}^{k}\left(M_{e,i}\right)_{j} 2^{kj}$ are available to each channel e, where $S_{M_{e,i}}$ is the size in bits of $M_{e,i}$. With these constants, the values $\xi_{e,i}\ {}^{k}\left(M_{e,i}\right)_{j}$ can be accumulated obtaining k bits of the result at once. The contribution that depends on α have to be computed by one of the channels that should have available the constants ${}^{k}\left(-M_{i}\right)_{j}$ such that $-M_i = \sum_{j=1}^{S_{M_i}/k} {}^{k}\left(-M_{i}\right)_{j} 2^{kj}$. This channel can compute $\alpha\ {}^{k}\left(-M_{2}\right)_{j}$ as a contribution to the final accumulation.

2 RNS Addition/Subtraction and Multiplication

Addition, subtraction, and multiplication in an RNS channel e is straightforward and relies in the corresponding binary operations followed by the final reduction in the channel if the result is not in the range $[0, m_{e,i} - 1]$. For addition and subtraction, a further addition/subtraction with $m_{e,i}$ is accomplished for this reduction, as presented in Algorithm 1.

In the multiplication the reduction stage is more elaborated, since for a k-bit channel the value to be reduced is bounded by $r' < 2^{2k} - 1$. Hence, r' can be rewritten as $r' = {}^{k}r'_{2}2^{k} + {}^{k}r'_{1}$ and since $2^k = m_{e,i} + c_{e,i}$ and $m_{e,i} \equiv 0 \bmod m_{e,i}$, therefore $r' \equiv {}^{k}r'_{2}c_{e,i} + {}^{k}r'_{1} \bmod m_{e,i}$. Algorithm 2 presents a RNS modular multiplication using the aforementioned properties. Note that the number of iterations of the loop in Algorithm 2 can be precomputed given the maximum value of $c_{e,i}$ and the maximum value to be reduced $(2^{2k} - 1)$, since these are the conditions that result in the largest number of iterations.

3 RNS Reduction

The reduction operation as herein discussed is usually introduced as part of a modular multiplication algorithm known as Montgomery Modular Multiplication (MMM) [6]. For convenience purposes, in the proposed framework the reduction part is considered separately from the multiplication itself. The rational of this algorithm is to replace the costly reduction mod N, where N is an arbi-

Algorithm 2 RNS channel multiplication.

Require: $a_{e,i}$, $b_{e,i}$ and $m_{e,i}$.
Ensure: $r_{e,i} = a_{e,i}b_{e,i} \bmod m_i$.
1: $r' = a_{e,i}b_{e,i}$;
2: **while** $r' \geq 2m_{e,i}$ **do** $r' = {}^{k}r'_{2}c_{e,i} + {}^{k}r'_{1}$;
3: $r_{e,i} = r'$;
4: **if** $r_{e,i} > m_{e,i}$ **then** $r_{e,i} = r_{e,i} - m_{e,i}$;
5: **return** $r_{e,i}$.

trary number, by a reduction mod L that is easier to compute. In the original proposal [6], the target was a binary system, hence the good choice for L were powers of 2 because reducing modulo 2 is the same than applying a binary mask. However, in RNS it is not possible to apply such mask which turns the reduction modulo 2 inefficient. Addressing this problem, an RNS variation of the MMM that relies in a Basis Extension (BE) method [3] has been proposed.

In RNS, reduction modulo the dynamic range M_1 is an easy task since if relies on the modulo operation in each one of the RNS channels of the basis B_1. Hence, for the reduction method, the value of L should equal M_1, with the requirement that $gcd(M_1, N) = 1$. In order to apply this method, the input data should be represented in the alternative Montgomery Domain (MD) (see Appendix 4). The correspondence between the MD and the data's Original Domain (OD) is bijective, hence any operation performed in the MD has a correspondence in the OD. In order to define the MD, the modulo N arithmetic and the RNS dynamic range M_1 should be defined. For an element A in the OD its correspondence in the MD is given by ${}_{M}A = A(M_1 \bmod N) \bmod N$. For the addition and subtraction of two elements in MD it is easy to see a direct correspondence to the OD since ${}_{M}A + {}_{M}B = (A + B)(M_1 \bmod N)$. For the multiplication of two elements in MD the result is given by ${}_{M}R = {}_{M}A_{M}B = (AB)(M_1^2 mod N)$, which includes an extra factor $M_1 \bmod N$ in the result. This factor is corrected by assuring that the multiplication is followed by a reduction operation that computes ${}_{M}R = {}_{M}RM_1^{-1} \bmod N$. Also, in order to keep a unified reduction method in RNS, each time the reduction after an addition or subtraction is performed the input value to reduce should be multiplied by $M_1 \bmod N$. Hence, for a value R to be reduced in OD, the input of the reduction in MD has the form $R(M_1^2 \bmod N)$. The reduction result ${}_{M}U$ is obtained by computing:

$$ {}_{M}U = ({}_{M}R + QN)/M_1 \tag{7} $$

Note that since $QN \equiv 0 \bmod N$, (7) corresponds to ${}_{M}U \equiv RM_1 \bmod N$ as desired. The purpose of the value QN is to assure that $({}_{M}R + QN)$ is a multiple of M_1, which allows the division by M_1 to introduce no approximation error in the result. Assuring that $({}_{M}R + QN)$ is a multiple of M_1 is equivalent to $({}_{M}R + QN) \equiv 0 \bmod M_1$, which is the same than computing ${}_{M}r_{e,1} + q_{e,1}n_{e,1} \equiv 0 \bmod m_{e,1}$ for all the RNS channels e of B_1. With the aforementioned property, Q is computed in the RNS channels as $q_{e,1} = {}_{M}r_{e,1}n_{e,1}^{-1} \bmod m_{e,1}$.

One of the drawbacks of this reduction algorithm is that M_1 cannot be represented in the moduli set with dynamic range M_1. This means that an extra basis B_2 with dynamic range M_2 higher than M_1, with $\gcd(M_2, M_1) = 1$ and $\gcd(M_2, N) = 1$, is required to compute (7). This is the reason of the BE

Algorithm 3 RNS reduction.

Require: ${}_M r_{e,1}$, ${}_M r_{e,2}$.
Ensure: ${}_M u_{e,1} = (RM_1 \bmod N)_{e,1}$.
Ensure: ${}_M u_{e,2} = (RM_1 \bmod N)_{e,2}$.
1: In ch. $e,1$: $q_{e,1} = \left|{}_M r_{e,1} n_{e,1}^{-1}\right|_{m_{e,1}}$
2: In ch. $e,1$: $\xi_{e,1} = \left|q_{e,1} \left|M_{e,1}^{-1}\right|_{m_{e,1}}\right|_{m_{e,1}}$
3: In ch. $e,2$: $q_{e,2} = \left|\sum_{j=1}^{h_1} \xi_{j,1} \left|M_{j,1}\right|_{m_{j,1}} - \hat{\alpha}_1 M_{1\ e,2}\right|_{m_{e,2}}$
4: In ch. $e,2$: ${}_M u_{e,2} = \left|({}_M r_{e,2} + q_{e,2} n_{e,2}) M_1^{-1}\right|_{m_{e,2}}$
5: In ch. $e,2$: $\xi_{e,2} = \left|{}_M u_{e,2} \left|M_{e,2}^{-1}\right|_{m_{e,2}}\right|_{m_{e,2}}$
6: In ch. $e,1$: ${}_M u_{e,1} = \left|\sum_{j=1}^{h_2} \xi_{j,2} \left|M_{j,2}\right|_{m_{j,2}} - \hat{\alpha}_2 M_2\right|_{m_{e,1}}$
7: **return** ${}_M u_{e,1}$, ${}_M u_{e,2}$.

method's designation, since the original basis is extended to allow computing (7). The use of the extra basis B_2 requires the data to be reduced (${}_M R$) to be also represented in B_2 and, the value of Q to be converted from B_1 to B_2. This conversion is performed with the method presented in the Appendix 1, with the main difference that for each channel e the operations are computed modulo $m_{e,2}$. After computing the result ${}_M U$ in B_2, it has to be converted to B_1 and the reduction algorithm is complete. Algorithm 3 presents in detail the steps of the presented reduction algorithm for each RNS channel.

In Algorithm 3 two conversions are performed using (2): from B_1 to B_2 with $\hat{\alpha}_1$ and from B_2 to B_1 with $\hat{\alpha}_2$. For the former conversion, the value to be converted is Q which is in the range $[0, M_1 - 1]$, whereas for the latter conversion the value converted is ${}_M U$ in (7). Given the range of Q, the conditions of Theorem 1 cannot be verified, thus $\hat{\alpha}_1$ should be obtained from (4) in the conditions of Theorem 2 instead, with the associated error. Therefore, for the computation of $\hat{\alpha}_1$ the value Φ in (4) should be set to zero and the minimum value of t such that $0 \leq \Delta \leq 1$ should be used (Theorem 2). Note that the minimum possible value of t results in the operands' size in the accumulation in (4) to be minimum as well. For ${}_M U$ it is possible to obtain $\hat{\alpha}_2$ under the conditions of Theorem 1, i.e., a value β such that $0 \leq {}_M U < (1-\beta)M_2$ exists. Therefore, the minimum value of t such that $\max({}_M U) < (1-\Delta)M_2$ should be computed and $\hat{\alpha}_2$ obtained from (4) using $\Phi = \lfloor 2^t \Delta \rfloor$.

Algorithm 3 can be further optimized by merging constants so as to reduce the required precomputed constants and temporary results. Algorithm 4 is an optimized version of Algorithm 3 that comprises the merging of the constants as: $\lambda_{1e} = \left|-n_{e,1}^{-1} M_{e,1}^{-1}\right|_{m_{e,1}}$, $\lambda_{2ej} = \left|-M_{j,1} M_1^{-1}\right|_{m_{e,2}}$, $\lambda_{3e} = \left|-M_1 n_{e,2}\right|_{m_{e,2}}$, $\lambda_{4e} = \left|M_1^{-1}\right|_{m_{e,2}}$, $\lambda_{5e} = \left|M_{e,2}^{-1}\right|_{m_{e,2}}$, $\lambda_{6ej} = \left|-M_{j,2} M_2^{-1}\right|_{m_{e,1}}$, and $\lambda_{7e} = \left|-M_2\right|_{m_{e,1}}$.

4 Montgomery Domains

The MDs are defined as domains where the operands involved in MA modN_i can be alternatively represented to enhance the efficiency of these operations in RNS, namely the reduction. The input variables of an algorithm are said to be in

Algorithm 4 Optimized RNS reduction.

Require: ${}_M r_{e,1}$, ${}_M r_{e,2}$.
Ensure: ${}_M u_{e,1} = (RM_1 \bmod N)_{e,1}$ and ${}_M u_{e,2} = (RM_1 \bmod N)_{e,2}$.
1: In ch. $e,1$: $q_{e,1} = |{}_M r_{e,1} \lambda_{e,1}|_{m_{e,1}}$
2: In ch. $e,2$: $q_{e,2} = \left| \left(\sum_{j=1}^{h_1} q_{j,1} \lambda_{2ej} + \hat{\alpha}_1 \right) \lambda_{3e} \right|_{m_{e,2}}$
3: In ch. $e,2$: ${}_M u_{e,2} = |({}_M r_{e,2} + q_{e,2}) \lambda_{4e}|_{m_{e,2}}$
4: In ch. $e,2$: $q_{e,1} = |{}_M u_{e,2} \lambda_{5e}|_{m_{e,2}}$
5: In ch. $e,1$: ${}_M u_{e,1} = \left| \left(\sum_{j=1}^{h_2} q_{j,1} \lambda_{6ej} + \hat{\alpha}_2 \right) \lambda_{7e} \right|_{m_{e,1}}$
6: **return** ${}_M u_{e,1}$, ${}_M u_{e,2}$.

their OD, thus conversions between MDs and the OD are required. Considering a variable X in its OD represented in an RNS basis B_1, the corresponding variable ${}_M X$ in the MD associated to the modulus N_i is ${}_M X = X(M_1 \bmod N_i) \bmod N_i$. The inverse conversion of domain can be performed as $X =_M X(M_1^{-1} \bmod N_i) \bmod N_i$. There is an isomorphism between any MD domain and the OD, therefore operations in the MD have a direct relationship with the operations in the OD:

- For efficiency reasons, the reduction $U = R \bmod N$ is usually computed recurring to a method that reduces the results but at the same time inserts a term $M_1^{-1} \bmod N_i$ in the result (see Appendix 3). Therefore, in order to assure that the output of this operation is in the required MD the input ${}_M R$ of the reduction is replaced by ${}_M R(M_1 \bmod N_i)$ so that the output is ${}_M R(M_1 \bmod N_i)(M_1^{-1} \bmod N_i) \bmod N_i =_M R \bmod N_i$.
- For the addition $R = A + B \bmod N_i$ the correspondence is direct, i.e, ${}_M A +_M B \bmod N_i = (A + B)(M_1 \bmod N_i) \bmod N_i =_M R$. For the subtraction the rational is the same.
- The multiplication $R = AB \bmod N_i$ in the MD results in $AB(M_1^2 \bmod N_i) \bmod N_i =_M R(M_1 \bmod N_i)$. Therefore, every time a multiplication is performed the factor $M_1 \bmod N_i$ should be corrected. This correction can performed using the aforementioned reduction operation in the MD without using the factor $M_1^{-1} \bmod N_i$. The utilization of the reduction to correct this factor is also useful to keep the required dynamic ranges as low as possible. The correction can be avoided if one of the operands ${}_M A$ or ${}_M B$ is used in the OD, which would result in $AB(M_1 \bmod N_i) =_M R$.

Using the aforementioned operations, it is possible to operate the conversions between OD and MDs. The conversion from OD to MD is accomplished with a multiplication of the operand to be converted X by $M_1^2 \bmod N_i$ followed by a reduction:

$$\begin{aligned} {}_M X &= X(M_1^2 \bmod N_i)(M_1^{-1} \bmod N_i) \bmod N_i \\ &= X(M_1 \bmod N_i) \bmod N_i, \end{aligned} \tag{8}$$

while the conversion from a MD to OD is obtained with a single reduction:

$$\begin{aligned} X &=_M X(M_1^{-1} \bmod N_i) \bmod N_i \\ &= X(M_1 \bmod N_i)(M_1^{-1} \bmod N_i) \bmod N_i. \end{aligned} \tag{9}$$

Note that the MDs depend on N_i, hence there is a MD for each N_i used in a given algorithm. Since the operands need to be in the same MD for the operations to be correct, conversions between MDs may also be required, which can be accomplished with a conversion from the current MD to OD followed by a conversion from the OD to the required MD.

5 Sample Algorithms

Public-key cryptography is probably the most prominent application that can take advantage of efficient and parallel implementations of modular arithmetic. These applications often use operands with hundreds or thousands of bits and can therefore greatly exploit the RNS capabilities that split these operands in small chunks that are easier to handle and moire efficient to compute. The following introduces two sample algorithms that can benefit from the RNS operations described in the previous sections.

5.1 Modular Exponentiation - RSA

The modular exponentiation is the main building block of the Rivest-Shamir-Adleman (RSA) algorithm and can be computed recurring to several multiplication and squaring operations controlled with the content of the exponent. Algorithm 5 presents an approach to compute the modular exponentiation relying exclusively on multiplications and modular reductions.

Algorithm 5 n-bit modular exponentiation.

Require: A a n-bit integer (the exponentiation basis);
Require: E a n-bit integer (the exponent);
Require: N a n-bit integer (the modulo);
Ensure: $R = A^E \bmod N$.

```
1: R = A;
2: for i = n − 1; i > 0; i − − do
3:    R = (R ∗ R) mod N;
4:    if E[i] == 1 /* the i^th bit of E is set*/ then
5:       R = (R ∗ A) mod N;
6:    end if
7: end for
8: return R;
```

5.2 Elliptic Curve Point Multiplication - Elliptic Curve Cryptography

In Algorithm 6 is presented a method to compute the Elliptic Curve (EC) point multiplication often referred to as Montgomery Ladder [2]. This algorithm receives the point information as a single coordinate (X_g) and returns the resulting

projective coordinates (X_p and Z_p). The scalar s that multiplies the point is specified as a constant and its bits are evaluated during the algorithm. One of the characteristics of this algorithm is that its duration (number of operations) does not depend on the scalar, being therefore resistant against time attacks, because the operations being computed in the `if` scope are similar to the ones computed in the `else` scope.

References

[1] S. Antão, J.-C. Bajard, and L. Sousa. Elliptic curve point multiplication on GPUs. In *IEEE International Conference on Application-specific Systems Architectures and Processors - ASAP*, pages 192–199, Rennes, Apr. 2010. IEEE.

[2] S. Antão, J.-C. Bajard, and L. Sousa. RNS based elliptic curve point multiplication for massive parallel architectures. *The Computer Journal 2011 - Oxford Journals*, 55(5):629–647, 2011.

[3] J.-C. Bajard, L.-S. Didier, and P. Kornerup. Modular multiplication and base extensions in residue number systems. In *IEEE Symposium on Computer Arithmetic - ARITH*, pages 59–65, Vail, CO, June 2001. IEEE.

[4] N. Guillermin. A high speed coprocessor for elliptic curve scalar multiplications over Fp. In S. Mangard and F.-X. Standaert, editors, *Lecture Notes in Computer Science: Advances in Cryptology - Cryptographic Hardware and Embedded Systems - CHES 2010*, pages 48–64. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010.

[5] S. Kawamura, M. Koike, F. Sano, and A. Shimbo. Cox-Rower architecture for fast parallel montgomery multiplication. In B. Preneel, editor, *Lecture Notes in Computer Science: Advances in Cryptology - EUROCRYPT 2000*, pages 523–538. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2000.

[6] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[7] A. P. Shenoy and R. Kumaresan. Fast base extension using a redundant modulus in RNS. *IEEE Transactions on Computers*, 38(2):292–297, 1989.

Algorithm 6 EC point multiplication with projective coordinates over a n-bit prime field.

Require: X_g a n-bit integer corresponding to the input EC point's x-coordinate;
Require: s a n-bit integer which is the scalar one wants to multiply the EC point with;
Require: N a n-bit integer (the modulo);
Require: b and c two integer EC parameters for a n-bit prime field;
Ensure: X_p and Z_p are the projective standard coordinates of the input EC multiplied by s.

1: $X_p = X_g$;
2: $Z_p = 1$;
3:
4: $A = X_g \times X_g \bmod N$;
5: $B = b \times X_g \bmod N$;
6: $C = A + 3$;
7:
8: $C = C \times C \bmod N$;
9: $A = X_g \times A \bmod N$;
10: $X_q = C - 8 \times B$;
11: $Z_q = 4 * (A - 3 * X_g + b)$;
12:
13: **for** $i = n$; $i > 0$; $i - -$ **do**
14: **if** $s[i] == 1$ /* the i^{th} bit of s is set*/ **then**
15: $A = X_q \times Z_p \bmod N$; $B = X_p \times Z_q \bmod N$; $C = X_q \times X_p \bmod N$;
16: $D = Z_q \times Z_p \bmod N$; $E = X_q \times X_q \bmod N$;
17: $F = Z_q \times Z_q \bmod N$; $H = b \times Z_q \bmod N$;
18: $Z_p = A - B$;
19: $X_p = A + B$;
20: $C = C + 3D$;
21: $A = E + 3F$;
22:
23: $D = D \times X_p \bmod N$; $X_p = C \times C \bmod N$; $Z_p = Z_p \times Z_p \bmod N$;
24: $A = A \times A \bmod N$; $B = F \times H \bmod N$;
25: $E = E \times X_q \bmod N$; $F = X_q \times F \bmod N$;
26: $G = E + B - 3 \times F$;
27:
28: $D = b \times D \bmod N$; $Z_p = X_g \times Z_p \bmod N$;
29: $B = X_q \times B \bmod N$; $F = Z_q \times G \bmod N$;
30: $X_p = X_p - 4D$;
31: $X_q = A - 8B$;
32: $Z_q = 4F$;
33: **else**
34: ...
35: **end if**
36: **end for**
37: $X_p = X_p \bmod N$;
38: $Z_p = Z_p \bmod N$;
39: **return** X_p, Z_p;