HIGH SPEED LOW POWER GF(2^k) ELLIPTIC CURVE CRYPTOGRAPY PROCESSOR ARCHITECTURE

Adnan Abdul-Aziz Gutub

Computer Engineering Department
King Fahd University of Petroleum and Minerals
Dhahran 31261, SAUDI ARABIA
Email: gutub@ccse.kfupm.edu.sa

Abstract. A new elliptic curve cryptographic processor architecture is proposed in this paper. It gives a choice of performance base depending on the importance of speed and/or power consumption. This flexibility is accomplished by utilizing the normal parallelism in the elliptic curve point operations. Scalable multipliers are adopted to compensate for the extra hardware due to parallelism instead of using the conventional parallel multipliers. It is shown in the paper that this parallelism can be exploited either to increase the speed of operation or to reduce power consumption by reducing the frequency of operation.

1. Introduction

CRYPTOGRAPHY is a well-recognized technique for information protection. It is used effectively to protect sensitive data such as passwords that are stored in a computer, as well as information being transmitted through different communication media. Encryption is the transformation of data into a form, which is very hard to retransform back by anyone without a secret decryption key. Even if someone steals the encrypted information, he can not make any use of it.

Depending on the encryption/decryption key, cryptographic systems can be classified into two main categories: secret key cryptosystems and public key cryptosystems. The secret key cryptosystems uses one key for both encryption and decryption. Public key cryptosystems, however, use two different keys, one for encryption and the other for decryption.

In 1985, Niel Koblitz and Victor Miller proposed the Elliptic Curve Cryptosystem (ECC) [1-9], a method based on the Discrete Logarithm problem over the points on an elliptic curve. Since that time, ECC has received considerable attention from mathematicians around the world, and no significant breakthroughs have been made in determining weaknesses in the algorithm. Although critics are still skeptical as to the reliability of this method, several encryption techniques have been developed recently using these properties. The fact that the problem appears so difficult to crack means that key sizes can be reduced in size considerably, even exponentially [2,5,8], especially when compared to the key size used by other cryptosystems. This made ECC become a challenge to the RSA, one of the most popular public key methods known. ECC is showing to offer equal security to RSA but with much smaller key size [2].

Several recent crypto processors have been proposed in the literature [4,7,17]. A common feature of these processors is that they remove the need for an inversion circuit. It is well known that point operations over an elliptic curve would require an inversion calculation, which is the most time-consuming computation over GF(2^k) [18,19]. To eliminate the need for performing inversion in GF(2^k), designs replace the inversion by several multiplication operations by converting the elliptic curve points as projective coordinate points [1,4,7,9,17]. This approach is also adopted in the processor proposed in this paper, which converts the normal elliptic curve points (x,y) to (X,Y,Z), where $x=X/Z^2$ and $y=Y/Z^3$ [9]. It is assumed that the reader is familiar with the elliptic curve arithmetic. Some necessary theoretical background of point operations, encryption and decryption are covered briefly to lead the reader to the proposed hardware design.

The proposed GF(2^k) projective coordinate elliptic curve processor architecture enjoys three main features:

- It takes advantage of the natural parallelism in the computation of the elliptic curve point operations, which gains the maximum possible reduction in the number of multiplication iterations in the algorithm.
- It adopts scalable multipliers, which allows a small hardware to iteratively compute the multiplication process of large numbers using practical hardware area especially when using several multipliers.
- It consumes less power than the traditional single multiplier designs assuming the overall computation time is the same.

After comparing the proposed hardware with different designs, our proposed architecture demonstrated significant improvement in the speed, cost: AT^2, and in the power consumption. It also showed possibility to operate at a higher clock frequency (to gain more speed) with the expense of higher power consumption.

The next section (Section 2) presents the reason behind choosing scalable multipliers in the hardware.

Then, some brief elliptic curve theoretical background is given in Section 3, followed by Section 4 where a cryptographic illustration of encryption and decryption is presented. Section 5 outlines the algorithm used for ECC multiplication which is the basic theory behind using elliptic curve in cryptography. The elliptic curve point addition and doubling are elaborated using projective coordinates in Section 6 followed by the description of the proposed hardware architecture in Section 7. Section 8 compares the design with others and Section 9 concludes the paper.

2. Scalable Multipliers

An arithmetic unit is called scalable if it can be reused or replicated in order to generate long precision results independently of the data path precision for which the unit was originally designed. To speed up the multiplication operation, various dedicated multiplier modules were developed in [20-22]. These designs operate over fixed finite fields. For example, the multiplier designed for 155 bits [21] cannot be used for any other field of higher degree. When a need for multiplication of larger precision appears, a new multiplier must be designed. Another way to avoid redesigning the module is to use software implementations and fixed precision multipliers. However, software implementations are inefficient in utilizing inherent concurrency of the multiplication because of the inconvenient pipeline structure of the microprocessors being used. Furthermore, software implementations on fixed digit multipliers are more complex and require excessive effort in coding. Therefore, a scalable hardware module specifically tailored to take advantage of the concurrency of the multiplication algorithm becomes extremely attractive [15,16]. Also computation of elliptic point doubling, addition and the algorithm of computing multiples of the base point is such that the multiplication of one stage must be completed before starting the multiplication of the subsequent stage. Therefore pipelining the digits to further stages is not applicable, even if fast digit serial multipliers are used, the throughput of such multipliers can not be exploited since the next multiplication operation can not begin until the multiplication operations in the previous stage has fully completed.

3. Elliptic Curves Over $GF(2^k)$

It will be assumed that the reader is familiar with the arithmetic over elliptic curve. For a good review the reader is referred to [9]. The elliptic curve equation over $GF(2^k)$ is: $y^2+xy=x^3+ax^2+b$; where $x, y, a, b \in GF(2^k)$ and $b \neq 0$.

The addition of two different points on the elliptic curve is computed as shown below:

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3) \text{ ; where } x_1 \neq x_2$$
$$\lambda = (y_2 + y_1)/(x_2 + x_1)$$
$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$$
$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1$$

The addition of a point to itself (Doubling a point) on the elliptic curve is computed as shown below:

$$(x_1, y_1) + (x_1, y_1) = (x_3, y_3) \text{ ; where } x_1 \neq 0$$
$$\lambda = x_1 + (y_1)/(x_1)$$
$$x_3 = \lambda^2 + \lambda + a$$
$$y_3 = (x_1)^2 + (\lambda + 1)\, x_3$$

To add two different points in GF(2^k) we need: nine additions, one inversion, one squaring, and two multiplication operations. To double a point we require: five additions, one inversion, two squaring, and two multiplication computations. Addition of field elements is performed by bit-wise XOR-ing the vector representations. The multiplication rule depends on the selected basis. There are many different bases of $GF(2^k)$ over $GF(2)$. Some bases lead to more efficient software or hardware implementations of the arithmetic in $GF(2^k)$ than other bases [9]. The most popular two bases used are the polynomial and the normal bases. The point operations will be discussed for ECC crypto processors in section 5.

4. Encryption and Decryption

There are many ways to apply elliptic curves for encryption/decryption purposes. In its most basic form, users randomly chose a *base point (x,y)*, lying on the elliptic curve *E*. The plaintext (the original message to be encrypted) is coded into an elliptic curve point (x_m,y_m). Each user selects a private key '*n*' and compute his public key $P = n(x, y)$. For example, user *A*'s private key is n_A and his public key is $P_A = n_A(x, y)$.

For any one to encrypt and send the message point (x_m,y_m) to user *A*, he/she needs to choose a random integer *k* and generate the ciphertext:

$$C_m = \{k(x, y)\, , (x_m, y_m) + kP_A\}.$$

The ciphertext pair of points uses *A*'s public key, where only user *A* can decrypt the plaintext using his private key.

To decrypt the ciphertext C_m, the first point in the pair of C_m, $k(x,y)$, is multiplied by *A*'s private key to get the point: $n_A\ (k(x,y))$. Then this point is subtracted from the second point of C_m, the result will be the plaintext point (x_m,y_m). The complete decryption operations are:

$$((x_m,y_m)+kP_A)-n_A[k(x,y)]=(x_m,y_m)+k(n_A(x,y))-n_A(k(x,y))=(x_m,y_m)$$

5. Point Operation Algorithm

The ECC algorithm used for calculating *nP* from *P* is the binary method, since it is known to be efficient and practical to implement in hardware [2,5,7,9,10]. This binary method algorithm is shown below:

```
Define   k: number of bits in n and  n_i: the i^th bit of n
Input:   P (a point on the elliptic curve).
Output:  Q = nP (another point on the elliptic curve).
    1.    if $n_{k-1} = 1$, then $Q:=P$ else $Q:=0$;
    2.    for $i = k-2$ down to $0$;
    3.        { $Q := Q+Q$ ;
    4.          if $n_i = 1$ then $Q:= Q+P$ ; }
    5.    return $Q$;
```

Basically, the binary method algorithm scans the bits of *n* and doubles the point *Q* *k*-times. Whenever, a particular bit of *n* is found to be one, an extra operation is needed. This extra operation is *Q+P*.

As can be seen from the description of the above binary algorithm, adding two elliptic curve points and doubling a point are the most basic operations in each iteration. As mentioned earlier, adding two points over elliptic curve requires inversion which is the most expensive operation in ECC [9]. A common approach (adopted in this paper) is to eliminate the need for an inversion circuit by representing the elliptic curve points as projective coordinate points [1,4,7,9,17].

6. Projective Coordinates

To eliminate the need for performing inversion in $GF(2^k)$, its coordinates (x,y) are projected to (X, Y, Z), where $x=X/Z^2$, and $y=Y/Z^3$. The projected elliptic curve equation becomes: $Y^2+XYZ=X^3+a\ X^2Z^2+bZ^6$. The formulas for projective point addition of two elliptic curve points are as follows:

$P=(X_1,Y_1,Z_1);Q=(X_2,Y_2,Z_2);P+Q=(X_3,Y_3,Z_3)$; where $P \neq \pm Q$

$$(x, y) = (X/Z^2,Y/Z^3) \Rightarrow (X,Y,Z)$$

$\lambda_1 = X_1 Z_2^2$	2M
$\lambda_2 = X_2 Z_1^2$	2M
$\lambda_3 = \lambda_1 + \lambda_2$	
$\lambda_4 = Y_1 Z_2^3$	2M
$\lambda_5 = Y_2 Z_1^3$	2M
$\lambda_6 = \lambda_4 + \lambda_5$	
$\lambda_7 = Z_1 \lambda_3$	1M
$\lambda_8 = \lambda_6 X_2 + \lambda_7 Y_2$	2M
$Z_3 = \lambda_7 Z_2$	1M
$\lambda_9 = \lambda_6 + Z_3$	
$X_3 = a Z_3^2 + \lambda_6 \lambda_9 + \lambda_3^3$	5M
$Y_3 = \lambda_9 X_3 + \lambda_8 \lambda_7^2$	3M

	20M

The formulas for projective point doubling of *P* are given by:

$$P = (X_1,Y_1,Z_1);\ P+P = (X_3,Y_3,Z_3)$$

$Z_3 = X_1 Z_1^2$	2M
$X_3 = (X_1 + bZ_1^2)^4$	3M
$\lambda = Z_3 + X_1^2 + Y_1 Z_1$	2M
$Y_3 = X_1^4 Z_3 + \lambda X_3$	3M

	10M

The complete data flow graph for doubling a point is shown in Figure 1. It requires ten multiplications and four addition (*k*-bit XOR) operations. Figure 2 shows the data flow graph for adding two elliptic curve points. It requires twenty multipliers and seven *k*-bit XOR gates. From the binary method, any elliptic curve crypto processor that uses projective coordinates must implements the dataflow graphs in Figure 1 and 2 iteratively.

7. Proposed Architecture

The architecture of the new processor is shown in Figure 3. It has the following features:

- It has four scalable multipliers,
- It can perform multiply-add operation within the same instruction because $GF(2^k)$ modulo adder is to be implemented in bit parallel fashion since the adders' area is not significant compared to the multipliers' size and minimizing the addition time will reduce the overall multiply-add cycle time.

The basic motivation behind the design of the proposed architecture is to exploit, as much as possible, the full parallelism that exists in the ECC. The trade-off between power and time can be achieved by reducing the clock frequency and hence consequently the source voltage. It is well known that reducing the source power is the most effective means of reducing power consumption. In the proposed design here, this is exploited at the algorithmic level by using more than one multiplier. The benefits of parallel implementation of ECC on power are discussed in more details in section 7.

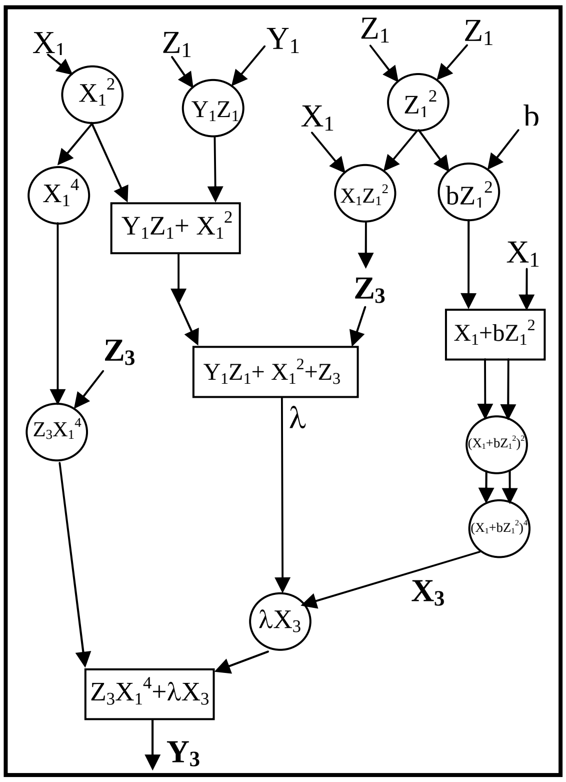

Fig 1. Doubling an elliptic curve point Data flow graph

The reason for using four multipliers only is as follows. From Figures 1 and 2, the corresponding critical path of each dataflow diagram is effectively of 5 $GF(2^k)$ multiplications and 7 $GF(2^k)$ multiplications, respectively. Here the time $GF(2^k)$ addition is ignored since it is negligible compared to multiplication. Therefore, the lower bound of the minimum computation time to perform one elliptic point operation in the calculation of *nP* is 12 $GF(2^k)$ multiplications. It can be easily seen from Figures 1 and 2 that performing four multiplications in parallel will meet this lower bound, and any further concurrent multiplications will not actually achieve any further reduction in the computation time.

The advantage of performing multiply-add operation in one instruction is that the dataflow in Figures 1 and 2 include many computations where the addition of the output of two multipliers must be carried out. Such a feature will avoid the need to store these values back in the registers and fetching them back again for their subsequent addition. This will save both in cycles and power.

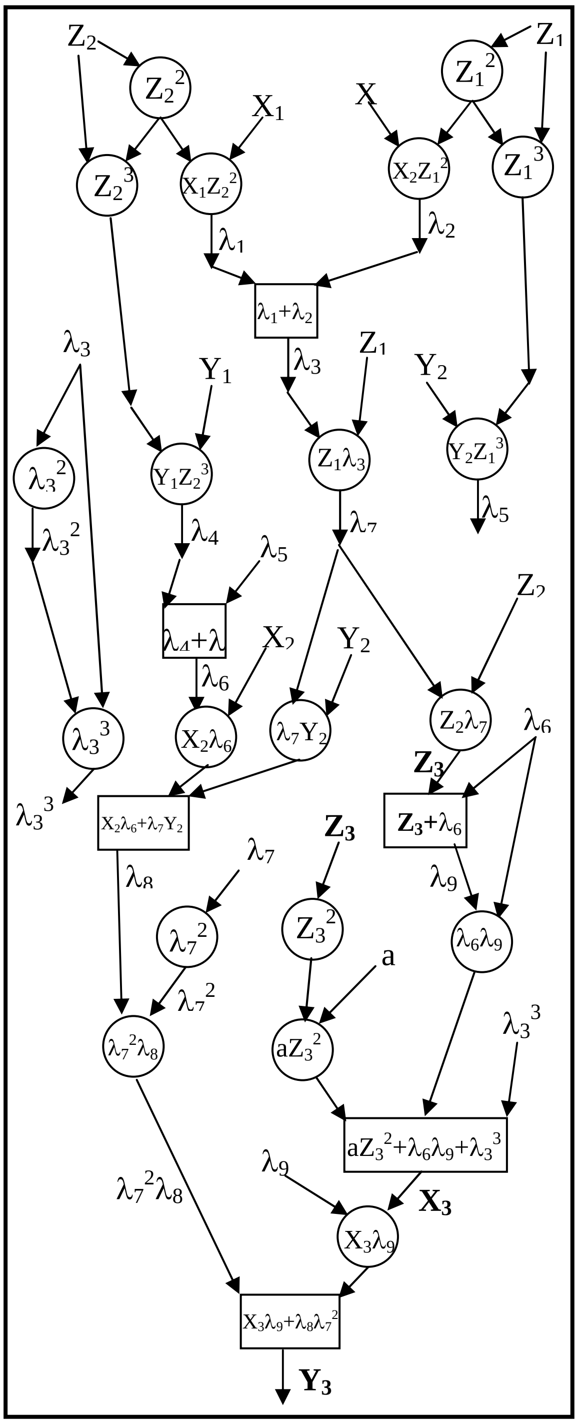

Fig 2. Data flow graph for adding two points

The purpose of the power management unit is to ensure that the power consumption of blocks that are not used is kept to a minimum. This is achieved by clock-gating the registers of these blocks and ensuring that the logic in these blocks is static. There are two possible cases where blocks are not used. The first is when not all four multipliers are used, and the second is when the application word-length is less than the word-length of the processor.

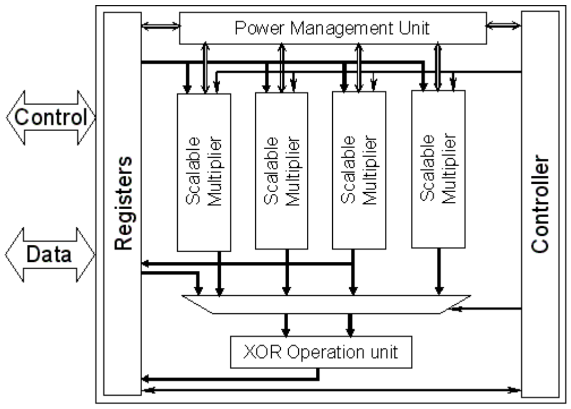

Fig 3. The ECC point operations hardware

8. Performance Comparisons

Since Power, $P=fCV_S^2$ and assuming that $V_S=kf_o$, where f_o is the maximum operating frequency for the given V_s, then $P=kf^3C$. The power consumption of using four multipliers is compared with that of using a single multiplier and two multipliers in Figure 4 for different execution times. Here time is computed as the number of cycles multiplied by the frequency per cycle.

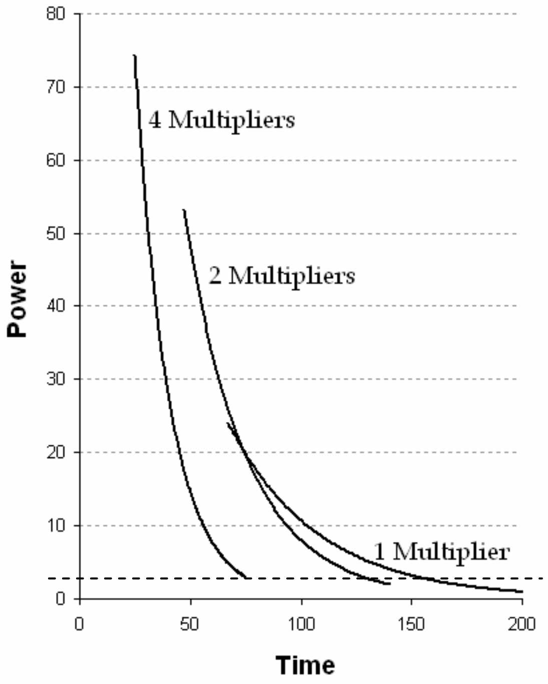

Fig 4. Different number of multipliers comparison.

In existing designs [4,6], a single multiplier is used to perform all the multiplications needed in Figures 1 and 2. The reason is that using more than one multiplier is perceived to be too expensive. However, as can be seen from Figure 4, the proposed architecture would lead to much lower power consumption than using one or two multipliers for the *same* execution time.

It is also clear that using four scalable multipliers gives a wider range of trade-off between power and speed. In fact, the case of using two multipliers does not provide any advantage over the other two options. Finally, the proposed architecture can support a further reduction in power by switching to one multiplier based operation in cases where a further reduction in power is required. In this case the power management unit will simply ensure that the other two multipliers do not consume any dynamic power.

9. Conclusion

A novel GF(2^k) elliptic curve crypto processor is proposed in this paper. The new architecture results in reduction in power consumption as well as offering users a range of trade-off between power and time. The basic feature of the new architecture is that it exploits the inherent parallelism in the computation of doubling and adding points over an elliptic curve as well as in multiplication. The achievement have been achieved through projective coordinates technique of elliptic curve arithmetic. Performance evaluation shows a considerable advantage over sequential implementation in terms of speed and power consumption.

10. Acknowledgments

The Author would like to thank Professor *Mohammad K. Ibrahim* for his valuable suggestions and comments given to improve this work. Also we show appreciation to King Fahd University of Petroleum and Minerals for its support of this research.

11. References

[1] Miyaji A., "Elliptic Curves over F_P Suitable for Cryptosystems", *Advances in cryptology-AUSCRUPT'92*, Australia, December 1992.

[2] Stallings, W. "*Cryptography and Network Security: Principles and Practice*", Second Edition, Prentice Hall Inc., New Jersey, 1999.

[3] Chung, Sim, and Lee, "Fast Implementation of Elliptic Curve Defined over GF(p^m) on CalmRISC with MAC2424 Coprocessor", *Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000*, Massachusetts, August 2000.

[4] Okada, Torii, Itoh, and Takenaka, "Implementation of Elliptic Curve Cryptographic Coprocessor over GF(2^m) on an FPGA", *Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000*, Massachusetts, August 2000.

[5] Crutchley, D. A., "*Cryptography And Elliptic Curves*", Master Thesis under Supervision of Prof. Gareth Jones, submitted to the Faculty of Mathematics at University of Southampton, England, May 1999.

[6] Orlando, and Paar, "A High-Performance Reconfigurable Elliptic Curve Processor for GF(2^m)", *Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000*, Massachusetts, August 2000.

[7] Stinson, "*Cryptography: Theory and Practice*", CRC Press, Boca Raton, Florida, 1995.

[8] Paar, Fleischmann, and Soria-Rodriguez, "Fast Arithmetic for Public-Key Algorithms in Galois Fields with Composite Exponents", *IEEE Transactions on Computers*, 48(10), October 1999.

[9] Blake, Seroussi, and Smart, "*Elliptic Curves in Cryptography*", Cambridge University Press: New York, 1999.

[10] Hankerson, Hernandez, and Menezes, "Software Implementation of Elliptic Curve Cryptography Over Binary Fields", *Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000*, Massachusetts, August 2000.

[11] Orton, Roy, Scott, Peppard, and Tavares, "VLSI implementation of public-key encryption algorithms", *Advances in Cryptology - CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 277-301, 11-15 August 1986. Springer-Verlag, 1987.

[12] Scott, Norman R., "*Computer Number Systems and Arithmetic*", Prentice-Hall Inc., New Jersey, 1985.

[13] Tocci, and Widmer, "*Digital Systems: Principles and Applications*", 8th Edition, Prentice-Hall Inc., New Jersey, 2001.

[14] Ercegovac, Lang, and Moreno, "*Introduction to Digital System*", John Wiley & Sons, Inc., New York, 1999.

[15] Alexandre F. Tenca and Cetin Koc, "A Scalable Architecture for Montgomery Multiplication", *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems - CHES'99*, August, 1999, Worcester, Massachusetts, USA.

[16] Erkay Savas, Alexandre F. Tenca, and Cetin K. Koc, "A Scalable and Unified Multiplier Architecture for Finite Fields GF(p) and GF(2^m)", *Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems – CHES2000*, Worcester, MA, USA, August 2000.

[17] Orlando, and Paar, "A scalable GF(p) elliptic curve processor architecture for programmable hardware", Cryptographic Hardware and Embedded Systems, CHES 2001, May 14-16, 2001, Paris, France.

[18] Gutub, Adnan Abdul-Aziz, Tenca, A., and Koc,C., "Scalable VLSI architecture for GF(p) Montgomery modular inverse computation", *IEEE Computer Society Annual Symposium on VLSI*, pages 53-58, Pittsburgh, Pennsylvania, April 25-26, 2002.

[19] Gutub, Adnan Abdul-Aziz, Tenca,A.F., and Koc,C., "Scalable and Unified Hardware to Compute Montgomery Inverse in GF(p) and GF(2^n)", *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 485-500, August 13-15, 2002.

[20] Royo, Moran, and Lopez, "Design and implementation of a coprocessor for cryptography applications", *European Design and Test Conference Proceedings*, pages 213–217, Paris, France, March 17-20 1997.

[21] Agnew, Mullin, and Vanstone, "An implementation of elliptic curve cryptosystems over $F_{2^{155}}$", *IEEE Journal on Selected Areas in Communications*, 11(5):804–813, June 1993.

[22] Naccache and MRaihi, "Cryptographic smart cards", *IEEE Micro*, 16(3):14–24, June 1996.