An Integrated Radix-4 Modular Divider/Multiplier Hardware Architecture for Cryptographic Applications

Lo'ai Tawalbeh[1], Yaser Jararweh[2], and Abidalrahman Mohammad[3]
[1]Computer Engineering Department, Jordan University of Science and Technology, Jordan
[2]Electrical and Computer Engineering Department, University of Arizona, USA
[3]Engineering Mathematics and Internetworking Department, Dalhousie University, Canada

Abstract: *The increasing importance of security in computers and communication systems introduces the need for several public-key cryptosystems. The modular division and multiplication arithmetic operations in GF (p) and GF (2^n) are extensively used in many public key cryptosystems, such as El-Gamal cryptosystem, Elliptic Curve Cryptography (ECC), and the Elliptic Curve Digital Signature Algorithm (ECDSA). Processing these cryptosystems involves complicated computations, therefore, it is recommended to develop specialized hardware to speed up these computations. In this work, we propose efficient hardware design to compute both operations (division and multiplication) in the binary extension finite filed (GF (2^n). The common points in both operations are utilized in our design to reduce the design area and delay. making the proposed architecture faster than other previously proposed designs. The FPGA implementation of the proposed design shows better results compared with other designs in this field.*

Keywords: *Cryptography, number theory, finite field arithmetic, hardware design, and radix-4.*

Received January 5, 2010; accepted August 10, 2010

1. Introduction

Very important data is transferred every second along wide distances. This data might be military information, financial records, and other vital information that is transmitted over public non-secure channels like the internet for example. The need to secure data transmission over channels became a priority.

Cryptography is widely used technique to secure data transmission. The cryptographic algorithms provide main security services such as confidentiality, integrity, and authentication. So, these algorithms must be used in a certain application according to the security service needed. The implementation of cryptographic algorithms in hardware offers better performance when compared with software implementations.

1.1. Cryptography

Cryptography involves the encryption process, which refers to converting formation (plaintext) into an unreadable format (cipher text) using a secret parameter (key). The decryption is the reverse process. Cryptanalysis attacks refers to the attempts of deducing the key used in the encryption. The modern cryptography falls into two main categories: Symmetric-key cryptography and Public-key cryptography (Asymmetric). In symmetric-key cryptography both the sender and the receiver of the message share the same key for encryption and decryption. Data Encryption Standards (DES) and Advance Encryption Standards (AES) are well known examples of symmetric ciphers. Using the same key between each sender and receiver of a message, requires a key management process to assure assigning each communication pair in the network a different private key [13].

In [7], a public-key cryptography methodology was proposed to overcome the key exchange problem in the symmetric key cryptosystems. A pair of two different but mathematically related keys is used: a public key (made public for all senders), and a private key (kept private with the receiver of the message only).

The public key is typically used for encryption, while the private key is used for decryption. Public-key algorithms are often based on the computational complexity of "difficult" problems in number theory. For example, the Hellman [7] and El-Gamal [3] cryptosystems are related to the discrete logarithm problem. Recently, Elliptic Curve Cryptography [13] technique was developed, in which security is based on elliptic curves operation such as point addition and point multiplication. The complexity of most public-key algorithms is due to difficult mathematical operations they involve, such as modular division, multiplication, and exponentiation. As a result, speeding up these computations directly influence the

performance of such cryptosystem and this was among the motivations for this work.

1.2. Modular Algorithms Over GF(2^n)

The extended Binary GCD algorithm [10] is an efficient way to calculate modular division by intertwining the procedure for finding the modular quotient with that for calculating the greatest common divisor of two polynomials in GF(2^n). In this work, we considered polynomial basis to represent the elements in GF(2^n). An efficient algorithm to compute modular multiplication is the Montgomery Multiplication (MM) algorithm. It has many advantages over ordinary modular multiplication algorithms. The main advantage is that the division step in taking the modulus is replaced by shift operations which are easy to implement in hardware [9, 14, 15]. The MM algorithm has been expanded from its original form [11], which is a fixed-precision implementation in radix-2, to a scalable, word-based implementation on multiple radices [15, 17].

1.3. Motivation for Radix-4

Implementing an encryption algorithm in hardware is faster and more secure than implementing it in software. That's because the security of software implementation depends on the security of the operating system which might not be fully achieved. The modular arithmetic operations such as division and multiplication over finite fields GF(p) and GF(2^n) are heavily used in several public-key cryptographic algorithms that are used to provide security services in many applications. So, modular division is a complex and necessary operation at the same time, and also it is considered an essential operation in the Elliptic Curve Cryptography [2, 7, 11, 15].

The importance of the modular arithmetic operations (including division) and the need for efficient implementation motivate the researchers to provide hardware architecture to accelerate the huge amount of computations required by public-key cryptographic algorithms.

Many previous works concentrated only on radix-2 and radix-8 designs for Montgomery Multiplication and division. A radix-4 Montgomery multiplication algorithm was shown in [15], which involves an encoding step for the multiples of the modulus. Other work use multi-bit shifting which provide ability to shift the operands K bits in each iteration [4], but this design increases the area in tremendous percent without providing increasing in the processing speed. Comparing with radix-2, radix-4 design is twice faster than radix-2 (since radix-4 algorithm scans 2 bits of the multiplier at a time which reduces the total number of computation cycles to half of what is needed for radix-2 [15].

2. Related Work

A high-radix Montgomery multiplication algorithm with radix-8 Montgomery modular multiplier as an example was proposed in [17] and an elliptic curve hardware that uses high radix multiplier was proposed in [5]. Compared to radix-2, the radix-8 design has less total computational time, but on the other hand, there was a significant increase in area and complexity. This result reveals an expected trade-off between chip area and computational time, and it should be considered in any hardware implementation of Montgomery multipliers. By increasing the radix, the multiplier operand is scanned faster; however, the determination of the quotient digit (qM) becomes more complex. Simplifying the determination of qM in high-radix modular multipliers was discussed in [8].

The multiplication algorithm can be distributed among a ring of processors, while each processor operates on a certain set of data, and then forwards this data to the next processor. This was a new approach for modular multiplication based on residue arithmetic presented in [1]. Also, the flexibility of the design should be taken in consideration, and the main candidates for flexible hardware are FPGAs. It was shown in [1] that a flexible and scalable design would have flexibility and adaptability comparable to conventional software and good performance because of the hardware speed. In [12] a unified multiplier architecture was suggested for finite fields GF(p) and GF(2^n). The proposed multiplier can operate in both fields without significant increases in the design area compared to a multiplier that works on GF(p) only. A radix-4 Montgomery multiplication algorithm that involves an encoding step for the multiples of the modulus was presented in [15]. The scalable (variable-precision) hardware design for varying operands size that implements the algorithm uses booth encoding of multiples to reduce the number of iterations.

On the other hand, there are many techniques to perform modular division (X(t)/Y(t) over GF(2^n). Among the best techniques to compute modular division is by using the iterative transformations of the greatest common divisor based on euclid algorithm [2]. The only limitation of the hardware implementations of the algorithms that is based on the Euclid's algorithm is the comparison step between the degrees of the polynomials at each iteration.

In [2], a method replacing this comparison by a much simpler operation (counter) was proposed, which significantly reduced the complexity of such division algorithms. By exploiting the counter idea, many efficient division algorithms were proposed [9, 10, 16]. The authors in [18] presented a binary shift-right algorithm and showed that this modification leads to better area-time complexity.

A multi bit-shift techniques are shown by Gutub [4]. The proposed inversion algorithms work in both

finite fields and are based on montgomery inversing algorithms. Most of the proposed designs compute the inverse in the binary extension fields GF(2^n) [16, 17].
On the other hand, a VLSI algorithm for modular division in GF(p) based on the Binary GCD algorithm was proposed in [10]. The algorithm is based on the plus-minus algorithm, which is a modification of the binary method for calculating the Greatest Common Divisor (GCD).

In [14] the authors proposed a novel unified algorithm for modular division and multiplication in both fields (GF(p) and GF (2^n)). The algorithm and its proposed hardware architecture which are based on readix-2 were among the first designs that combine both operations in one unit with minimum area-complexity trade-off.

3. Modular Algorithms Design in GF(2^n)

In this section, we provide the radix-4 division and multiplication algorithms that will be integrated in the next section to accelerate the arithmetic operations execution in cryptographic applications.

3.1. Modular Division Algorithms in GF(2^n)

The modular division algorithm on this work is based on the Extended Binary GCD algorithm [10]. The modular division algorithm computes the modular division in GF(2^n) as: C(x)=A(x)/B(x) mod p(x). Figure 1 shows the radix-4 modular division algorithm in GF(2^n). This algorithm was presented in [9] and it will be used as our basic design component in the integrated design.

```
Function:Modular Division in GF(2^n) field Inputs:0≤X < P(x), 0≤Y <
P(x), 2^(n-1) <p< 2^n outputs:Z(x) = X(x)/Y(x) mod p(x)
Algorithm:
C=Y, U=X, D=2p, W=0, δ = 1, Sgn=0
while C ≥ 0 do
IF c0 = 0 THEN
    IF c1 = 0 THEN C := C/4, C := RED(C,D)
    IF Sgn = 0 THEN δ := δ + 1
    ElseIF δ = 1 THEN δ := δ + 1, Sgn := 0 ELSE δ := δ − 1
  ElseIF Sgn = 1 THEN C := (C ⊕ D)/4, U := RED(U, W)
      IF δ = 1 THEN Sgn := 0, Else δ := δ − 1
      Else Sgn = 1, { C := (C ⊕ D)/4, D := C/4},
      { U := RED(U, W), C := RED(C,D)}
ElseIF Sgn = 1 THEN
    IF c1 = 0 THEN C := (C ⊕ D/2)/4, W := RED(U, W)
    Else C := (C ⊕ D ⊕ D/2)/4, W := RED(U, W)
    IF δ = 1 THEN Sgn := 0, Else δ := δ − 1
Else
    IF c1 = 0 THEN{ C := (C ⊕ D/2)/4, D := C/2},
  { W := RED(U, W), C := RED(C,D)}
Else{ C := (C ⊕ D ⊕ D/2)/4, D := C/2},{ W := RED(U, W), C :=
RED(C,D)}
IF δ = 1 THEN Sgn := 1, δ := δ − 1
Return W
```

Figure 1. Radix-4 modular division algorithm in GF(2^n) [9].

The shown radix-4 division algorithm uses a digit size of 2. It needs a maximum of 1.2 n iterations to compute the result, where n is the operand size. The double shift right (»4) operator stands for the division by the square of the polynomial root (i.e., X2).

The bits c0 and c1 are used to control of the algorithm flow. The body of the algorithm is divided into three main cases, with the set of three operations: shift, shift-add, shift-add and swap. For more details about these cases, the reader is forwarded to [9].

One important part of the algorithm is the ± counter (Denoted delta: δ) which replaced the hard comparison operations in the previously proposed algorithm [4, 6, 8]. The counter (δ) would never be incremented or decremented by 1, but only by 2 since we are using radix-4. Moreover, the counter is initialized with an odd value, so it will never reaches zero. It is important to take care of the counter to ensure an optimal implementation as the counter variable is used to limit the number of iterations and the comparisons, forcing the swap of variables to be performed, which is required for convergence.

3.2. Montgomery Modular Multiplication Algorithms in GF(2^n)

In Montgomery [11], described a modular multiplication algorithm which proved to be very efficient in both hardware and software implementations. The algorithm replaces division operations with simple shift operations which significantly reduces the algorithm complexity. In our work, we modify the algorithm proposed in [17] in order to be compatible with our division algorithm.

In our case, multiplication is performed in radix-4 and over GF(2^n): C(x)=A(x)/B(x) mod p(x). Therefore, the LSDs (least significant digits or last two bits) of B(x), p(x), C(x), and of the current digit of A(x) are used in order to determine the multiple quotient (q) to generate the partial product. The LSB of p(x) is always 1 (p is odd), then only the second least significant bit of the modulus is included in the computations. The complete details about the algorithm are shown in [9].

4. Radix-4 Integrated Modular Divider/Multiplier

The extended binary GCD and Montgomery modular multiplication algorithms can be modified and combined based on the similar operations they have. By exploiting these similarities we can introduce an integrated Division/ Multiplication algorithm with reasonable area increase. In this section, we propose a Radix-4 Division and Multiplication (R4DM) algorithm over GF(2^n) and its hardware architecture.

4.1. Radix-4 Modular Division/Multiplication Algorithm in GF(2^n)

Figure 2 shows the proposed R4DM algorithm over GF(2^n). The algorithm has two modes of operation: (div or mult). Most of the arithmetic computations in the algorithm are common to both modes of operation. The complete details and the proof of the R4DM are proposed in [9].

Function: Modular Division and Multiplication in GF (2^n) field
Inputs: $0 \le X < P(x)$, $0 \le Y < P(x)$, $2^{n-1} < p < 2^n$, *Op, n*
Outputs: $Z(x) = X(x)/Y(x) \bmod p(x)$ *when Op = div,* $Z(x)=X(x)Y(x) \bmod p(x)$ *when Op = mult.*
Algorithm:
IF Op = div THEN C=Y, U=X, D=2p, W=0, δ = 1, Sgn=0
Else C=Y, U=0, D = 2p, W=X, δ = Ceiling (n/2), Sgn=1
while[(C≥ 0 AND Op = div)OR (δ = 0 AND Op = mult)] do
IF c_0 = 0 THEN
IF c1 = 0 THEN C := C/4, C := RED (C,D)
IF (Sgn = 1 AND Op = div) THEN δ := δ + 1
ElseIF δ =1THEN δ := δ + 1, Sgn:=0 ELSE δ:=δ−1
ElseIF Sgn=1THEN C:= (C⊕D)/4, U := RED(U, W)
IF δ = 1 THEN Sgn := 0, Else δ := δ − 1
Else Sgn = 1, { C := (C ⊕ D)/4, D := C/4},
{U := RED(U, W), C := RED(C,D)}
ElseIF ((Sgn=1 AND Op=div)OR(Op=mult)) THEN
IF c1=0 THEN C:=(C⊕D/2)/4, W, U:=RED(U, W)
Else C := (C ⊕ D ⊕ D/2)/4, W,U := RED(U,W)
I F δ=1 TH EN Sgn := 0, Else δ := δ − 1
Else
IF c_1 = 0 THEN{ C := (C ⊕ D/2)/4, D := C/2},
{W := RED(U, W), C := RED(C,D)}
Else{C:=(C⊕D/2)/4, D:=C/2},{W:=RED(U,W), C:= RED(C,D)}
IF δ≠ 1 THEN Sgn := 1, δ := δ − 1
IF Op = div THEN Z := W Else Z := U
Return Z

Figure 2. R4DM algorithm in GF(2^n) [9].

The initialization of variables depends on that division or multiplication being performed by the algorithm. For simplicity, the polynomials X(x), Y(x), and p(x) are denoted as X, Y, and p, respectively, which corresponds to the bit-vector representation of these polynomials.

The R4DM performs n/2 iterations to compute Montgomery Modular multiplication using an n bit modulus p. The counter δ is initialized with value n/2, and in each iteration it is decremented by one. The variables used in the algorithm are initialized as: C=Y, U=0, D=2p, W=X, δ=n/2, Sgn=1. The partial product U is reduced mod p in each iteration. Notice that the addition in GF(2^n) is done without carry propagation (bitwise XORING). The multiplication is completed when δ=0 and the final result is (Z=U).

On the other hand, the R4DM algorithm computes modular division when the variable Op=div. The variables are initialized as: C=Y, U=X, D=2p, W=0, δ=1, Sgn=0. The division is completed when C=0, and the final result is (Z=W). Notice that the modular reduction step is performed every iteration for both operations. The reader is forwarded to [9] for the complete details of the R4DM algorithm operation.

4.2. Hardware Design of The Radix-4 Modular Divider/Multiplier in GF(2^n)

Figure 3 shows the top level design of the R4DM that implements the R4DM algorithm. The hardware design has a Register File, Data path, and Control units. The complete details about each component of this design are presented in [9].

The register file has four registers (R1 to R4). The computations are done in GF(2^n), and the elements are represented in non-redundant format (not Carry-Save format). Each intermediate variable (C, U, D, W) is represented only as one vector (sum), and there is no carry vector (carry free addition in GF(2^n)). So, each register inside the register file stores one n-bit vector. The register file has one input, and two output ports.

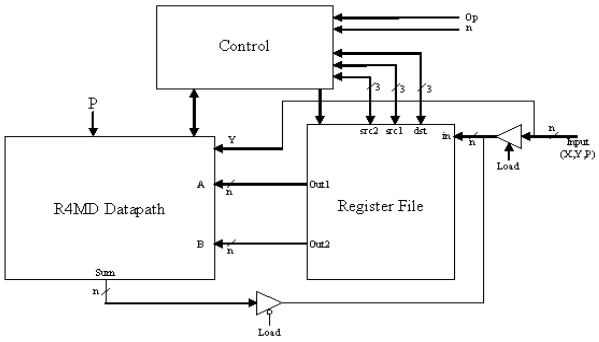


Figure 3. R4DM hardware design.

The Control block provides the register file with the signals necessary to perform reading/writing operations. The 3bit signal dst determines the destination register to be written. The signals src1, src2 (3bits each), specify the registers to be read at output ports out1, out2, respectively.

The proposed-bit data path is simple. The main operation that determines the critical path is the addition, and so, the main components of the data path are two XOR gates array adders to perform carry-free addition in GF(2^n) field. Also, the data path includes shift register that is used to implement the 2bit right shift operation (C>>2). This shift register is loaded with the multiplier (C=Y) and shifted right by 2bit when a control signal is asserted. The least significant two bits of the shifted operands are used by the control section to perform the test on least two signification bits of C (c1 c0). Finally, the outputs of the data path (Sum) is shifted by 2bit to the right.

The delay of the adders used to perform addition in GF(2^n) (XOR Gates Array) equals to the delay of one XOR gate, which is very small delay when compared with carry save adders that used for addition in GF(p). The data path also is responsible for generating the

suitable multiples of the irreducible polynomial (p) according to the two lest significant bits of U(u1, u0). As it is clear from the algorithm, these multiplies of (p) will be added to U, in order to keep the two lest significant bits (u1, u0)=00 before the shifting operation to avoid data loss. More explanations about each component of this design are presented in [9].

5. Experimental Results

We show in this Section an estimation of the number of iterations and the critical path delay results for the hardware description of the algorithm.

5.1. The Number of Iterations

The Unified Modular Division (UMD) algorithm presented in [16] computes modular division in two different fields. The UMD works only in Radix-2 with some exception in GF(p) which is not our concern here. Our proposed division algorithm in section 3.1 operates in Radix-4, and needs about 40% less iteration than UMD when computing the inverse in GF(2^n). And we notice that the number of iterations for both algorithm increase linearly with the operand size.

In [14], it stated that the inversion in GF(2^n) takes on average 3.3 cycles for each bit. UMD needs a maximum of 2 iterations/bit and on average 1.5 iterations/bit to compute the modular inverse in GF(2^n). Our proposed algorithm takes maximum1.2 iterations/bit, and on average 1.14 iterations/bit to compute the modular inverse in GF(2^n). These results are justified by the facts that our algorithm scans two bits in each iteration, but UMD scans one bit in each iteration.

5.2. Synthesis Results

The hardware design of the modular divider/multiplier that implements the R4DM algorithm was described in VHDL and simulated in ModelSim. Then, the design was synthesized using Xilinx ISE 10.1i to obtain area and delay results. The target technology used was FPGA Vertex 5 (xc5vfx30t-2ff665).

5.2.1. Area Results

Table 1 shows the area results obtained by synthesizing the design for the R4DM algorithm in number of slices for operand size form 16-512bits.

Table 1. The Area results of (R4DM) design in number of slices for operand size form 16-512 bits (Vertex 5).

Operand Size (n)	Area (No. of Slices)
16	216
32	328
64	552
128	1000
256	1896
512	3688

For the purpose of comparison we synthesized our design using vertex II chip (xc2v250-6cs144), in order to compare with the results presented in [14]. The results are shown in Figure 4.

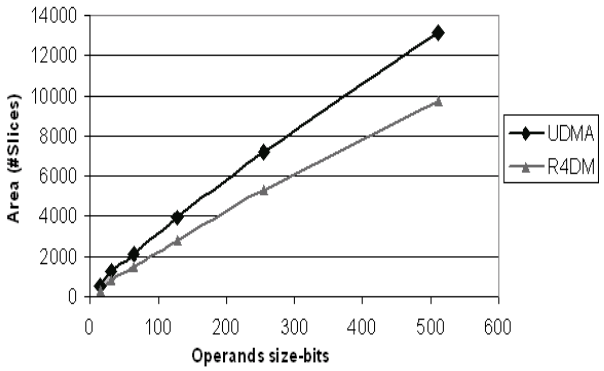


Figure 4. Area comparison (vertexII).

From Figure 4, we notice that the area increases linearly as the operand size increases in both design. The difference between the two results is due to the fact that our R4DM design works on GF(2^n) only where the multiplication and addition operations are easily implemented by simple AND, XOR operations. On the other hand, the UDMA design works on both fields (GF(p) and GF(2^n)) which needs extra hardware to perform the complex operations in GF(P).

5.2.2. Critical Path Delay Results

Table 2 shows the critical path delay (clock period) in nano-seconds for operand size in the range: 16-512bits. The operating frequency of the R4DM design is the reciprocal of the clock period. Form Table 2, the lowest clock period (11.79 ns) happened at 16bits operand size, and so, the maximum operating frequency is around 84.8MHz.

Table 2. The critical path delay in nano-seconds for operand sizes 16-512 bits (vertex 5).

Operand Size (n)	Delay (Nano Sec)
16	11.79
32	12.58
64	14.77
128	18.24
256	19.04
512	19.32

Again, and for the purpose of comparison with [14], we synthesized our design using vertex II (xc2v250-6cs144). Figure 5 shows the critical path delay comparison results. The difference between the two results is explained by the fact of working only in GF(2^n) in our design. The UDMA design uses adders to perform the addition in GF(p) which is more complicated and has more delay than the simple adders used in our design which are arrays of XOR gates.

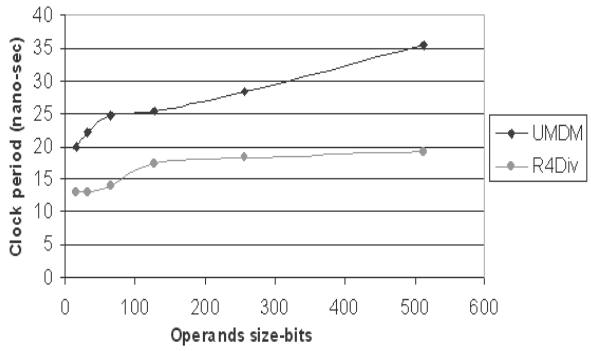


Figure 5. The critical path delay comparison (vertex II).

6. Conclusions

In this work, we proposed a radix-4 modular division algorithm to compute modular division in GF(2^n). The proposed algorithm computes the division in GF(2^n) field in an efficient way when compared with other algorithms. It uses counter to replace the polynomial comparison step. The proposed algorithm was integrated with a modified version of the Montgomery multiplication algorithm to produce a R4DM algorithm. The hardware design that efficiently implements the R4DM algorithm is also proposed.

The proposed hardware design of the R4DM was described in VHDL, and simulated using ModelSim. The area and delay synthesis results using FPGAs vertex 5 and vertex 2 chips are obtained and compared with other designs. The experimental results showed that the computation time and the area of the proposed R4DM design is competitive with other designs.

Acknowledgements

I would like to thank Jordan University of Science and Technology (JUST) for the continuous support for the research.

References

[1] Bajard J., Didier L., and Kornerup P., "An RNS Montgomery Modular Multiplication Algorithm," *IEEE Transactions on Computers*, vol. 47, no. 7, pp.766-776, 1998.

[2] Brent R. and Kung H., "Systolic VLSI Arrays for Polynomial GCD Computation," *IEEE Transactions on Computers*, vol. 33, no. 8, pp. 731-736, 1984.

[3] ElGamal T., "A Public Key Cryptosystem and Signature Scheme Based on Discrete Logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469-472, 1998.

[4] Gutub A., "New Hardware Algorithms and Designs for Montgomery Modular Inverse Computation in Galois Fields GF(p) and GF(2^n)," *PhD Thesis*, Oregon state University, USA, 2002.

[5] Gutub A., "Fast 160-Bits GF(P) Elliptic Curve Crypto Hardware of High-Radix Scalable Multipliers," *The International Arab Journal of Information Technology*," vol. 3, no. 4, pp. 342-349, 2006.

[6] Hasan M. and Bhargava V., "Bit-Serial Systolic Divider and Multiplier for Finite Fields GF(2^m)," *IEEE Transaction on Computers*, vol. 41, no. 8, pp. 972-980, 1992.

[7] Hellman M. and Difie W., "New Directions in Cryptography," *IEEE Transactions Information Theory*, vol. 22, no. 6, pp. 644-654, 1976.

[8] Itoh T. and Tsujii S., "A Fast Algorithm for Computing Multiplicative Inverses in GF(2^m) using Normal Bases," *Information and Computation*, vol. 78, no. 3, pp.171-177, 1988.

[9] Jararwah Y., "A Hardware Architecture of an Efficient Modular Division Algorithm for Cryptographic Applications," *Master's Thesis*, Jordan University of Science and Technology, Jordan, 2007.

[10] Kaihara M. and Takagi N., "A VLSI Algorithm for Modular Multiplication/Division," *in Proceedings the IEEE 16th Symposium on Computer Arithmetic*, Japan, pp. 220-227, 2003.

[11] Montgomery P., "Modular Multiplication without Trial Division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519-521, 1985.

[12] Savas E., Tenca A., and Koc C., "A Scalable and Unified Multiplier Architecture for Finite Fields GF(p) and GF(2^m)," *in Proceedings of Cryptographic Hardware and Embedded Systems*, USA, pp. 281-296, 2000.

[13] Stallings W., *Cryptography and Network Security Principles and Practices*, Prentice Hall, USA, 2005.

[14] Tawalbeh L. and Tenca A., "An Algorithm and Hardware Architecture for Integrated Modular Division and Multiplication in GF(p) and GF(2^n)," *in Proceedings of IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, USA, pp. 247-257, 2004.

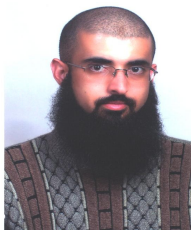
[15] Tawalbeh L., Tenca A., and Koc C., "A Radix-4 Scalable Design," *IEEE Potentials Magazine*, vol. 24, no. 2, pp. 16-19, 2005.

[16] Tenca A. and Tawalbeh L., "An Algorithm for Unified Modular Division in (GF(p) and (GF(2^n)) Suitable for Cryptographic Hardware," *IEE Electronics Letters*, vol. 40, no. 5, pp. 304-306, 2004.

[17] Tenca A., Todorov G., and Koc C., "High-Radix Design of a Scalable Modular Multiplier," *in Proceedings of Cryptographic Hardware and Embedded Systems*, pp.189-206, 2001.

[18] Hsing-Wu C., Ming-Wu C., Shieh M., and Hwang Y., "High-Speed, Low Complexity

Systolic Designs of Novel Iterative Division Algorithms in GF(2^m)," *IEEE Transactions on Computers*, vol. 53, no. 3, pp. 375-380, 2004.



Lo'ai Ali Tawalbeh is an assistant professor of Computer Engineering at Jordan University of Science and Technology (JUST), and a part time professor at the New York institute of Technology (NYIT)-Jordan's campus. He received his BS in Electrical and Computer Engineering from Jordan University of Science and Technology in June of 2000. Then he worked as a research and development engineer in a leading company, and then as a Teaching Assistant in JUST before he joined the graduate program at Oregon State University (OSU) in September 2001. He received his MS degree in electrical and computer engineering from Oregon State University in October 2002, and his PhD MSc. He research interests include network and computer security. Intrusion detection and computer forensics. Hardware implementations for cryptography, and cryptographic co-processor design using scalable modules, Elliptic Curve Cryptography, network security and embedded systems, FPGA design, VLSI design, computer architecture and finite field arithmetic algorithms. Dr. Lo'ai has many journal publications and conference proceedings in the above research topics. He received many grants and research awards.

Yaser Jararweh received his Bsc and Msc from Jordan University of Science and Technology, Jordan in 2005, and 2007 respectively. He is now a PhD student at the Department of Electrical and Computer Engineering, The University of Arizona, USA. His research interest are in Power and Performance Management of Large-scale Data Centers.

Abidalrahman Mohammad is currently a PhD candidate in Engineering Mathematics and Internetworking Department at Dalhousie University MSc. His main research concern is to develop high throughput security protocols and cryptographic algorithms for bandwidth-intensive multimedia applications, as well as, energy efficient and low-power architectures such as wireless sensor networks. He received both his BSc and MSc Degrees in computer engineering under the supervision of Dr. Loai Tawalbeh, from Jordan University of Science and Technology in February, 2006 and July, 2007 respectively.