High Performance Implementation of a Public Key Block Cipher - MQQ, for FPGA Platforms

Mohamed El-Hadedy,[*]Danilo Gligoroski,[†]and Svein J. Knapskog[‡]

August 4, 2008

Abstract – We have implemented in FPGA recently published class of public key algorithms – MQQ, that are based on quasigroup string transformations. Our implementation achieves decryption throughput of 399 Mbps on an Xilinx Virtex-5 FPGA that is running on 249.4 MHz. The encryption throughput of our implementation achieves 44.27 Gbps on four Xilinx Virtex-5 chips that are running on 276.7 MHz. Compared to RSA implementation on the same FPGA platform this implementation of MQQ is 10,000 times faster in decryption, and is more than 17,000 times faster in encryption.

Keywords – Ultra Fast Public Key Cryptosystems, Multivariate Quadratic Quasigroup, MQQ

1 Introduction

The most popular Public Key Cryptosystem (PKC) schemes are the Diffie and Hellman (DH) key exchange scheme based on the hardness of discrete logarithm problem [4], the Rivest, Shamir and Adleman (RSA) scheme based on the difficulty of integer factorization [21], and the Koblitz and Miller (ECC – Elliptic Curve Cryptography) scheme based on the discrete logarithm problem in an additive group of points defined by elliptic curves over finite fields [14, 17]. There are two common characteristics of these well known PKCs (DH, RSA and ECC): 1. their speed – which frequently is a thousand times lower than the symmetric cryptographic schemes, 2. their security – which relies on one of two hard mathematical problems: efficient computation of discrete logarithms and factorization of integers.

Several other ideas have been proposed during the last 30 years, such as:

- Trapdoor functions that are based on multivariate quadratic polynomials such as that of Matsumoto and Imai (MIA) [11], Stepwise Triangular Scheme (STS) (i.e. Birational Permutation Schemes) by Shamir [22], Hidden Field Equations (HFE), by Patarin [18, 19], and Unbalanced Oil and Vinegar (UOV) by Kipnis et al., [12];
- McEliece PKC based on error correcting codes [16];
- Rabin's digital signature method [20];
- PKCs based on lattice reduction problems [1, 8] and on lattice problems over rings such as NTRU [10];

[*]Mohamed El-Hadedy is with Centre for Quantifiable Quality of Service in Communication Systems, Norwegian University of Science and Technology, Trondheim, NORWAY, e-mail: mohamed.el-hadedy@q2s.ntnu.no

[†]Danilo Gligoroski is with Department of Telematics, Faculty of Information Technology, Mathematics and Electrical Engineering, The Norwegian University of Science and Technology (NTNU), Trondheim, Norway, e-mail: danilog@item.ntnu.no

[‡]Svein Johan Knapskog is with Centre for Quantifiable Quality of Service in Communication Systems, Norwegian University of Science and Technology, Trondheim, NORWAY, e-mail: Svein.J.Knapskog@q2s.ntnu.no

- PKCs based on braid groups [13];

Recently a new public key scheme called MQQ which is based on multivariate quadratic polynomials and quasigroup string transformations was proposed by Gligoroski et al., [6, 7]. According to the authors of MQQ, that scheme has potential to be as fast as a typical block cipher. In this paper we are describing an implementation in Xilinx Virtex-5 FPGA that actually confirms the original claims by the authors of MQQ.

Organization of the paper is the following: In Section 2 we give a brief description of the MQQ algorithm. The hardware implementation of MQQ encryption is described in Section 3, and the hardware implementation of MQQ decryption is described in Section 4. A comparative analysis of our implementation with RSA and AES is given in Section 5. Conclusions are given in Section 6.

2 Preliminaries

In this section we will briefly describe the MQQ algorithm. More detailed description the reader can find in [6, 7]. Since it is based on quasigroups we give several definitions about quasigroups. Additional information about quasigroups reader can find in [2, 3, 15, 23].

Definition 1 *A quasigroup $(Q, *)$ is a groupoid satisfying the law*

$$(\forall u, v \in Q)(\exists! x, y \in Q) \quad u * x = v \ \& \ y * u = v. \tag{1}$$

It follows from (1) that for each $a, b \in Q$ there is a unique $x \in Q$ such that $a * x = b$. Then we denote $x = a \setminus_* b$ where $\setminus_*$ is a binary operation in Q (called a left parastrophe of $*$) and the groupoid $(Q, \setminus_*)$ is a quasigroup too. The algebra $(Q, *, \setminus_*)$ satisfies the identities

$$x \setminus_* (x * y) = y, \quad x * (x \setminus_* y) = y. \tag{2}$$

Consider an alphabet (i.e., a finite set) Q, and denote by Q^+ the set of all nonempty words (i.e., finite strings) formed by the elements of Q. In this paper, depending on the context, we will use two notifications for the elements of Q^+: $a_1 a_2 \ldots a_n$ and $(a_1, a_2, \ldots, a_n)$, where $a_i \in Q$. Let $*$ be a quasigroup operation on the set Q. For each $l \in Q$ we define two functions $e_{l,*}, d_{l,*} : Q^+ \to Q^+$ as follows:

Definition 2 *Let $a_i \in Q$, $M = a_1 a_2 \ldots a_n$. Then*

$$\begin{aligned} &e_{l,*}(M) = b_1 b_2 \ldots b_n \iff \\ &\qquad b_1 = l * a_1,\ b_2 = b_1 * a_2, \ldots,\ b_n = b_{n-1} * a_n, \\ &d_{l,*}(M) = c_1 c_2 \ldots c_n \iff \\ &\qquad c_1 = l * a_1,\ c_2 = a_1 * a_2, \ldots,\ c_n = a_{n-1} * a_n, \end{aligned}$$

i.e., $b_{i+1} = b_i * a_{i+1}$ and $c_{i+1} = a_i * a_{i+1}$ for each $i = 0, 1, \ldots, n-1$, where $b_0 = a_0 = l$.

The functions $e_{l,*}$ and $d_{l,*}$ are called the e–transformation and the d–transformation of Q^+ based on the operation $*$ with leader l respectively, and their graphical representations are shown in Fig. 1.

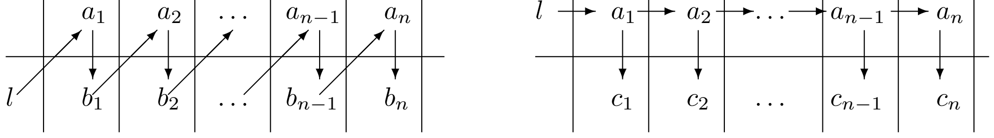

Figure 1: Graphical representations of the $e_{l,*}$ and $d_{l,*}$ transformations

Theorem 1 *If $(Q, *)$ is a finite quasigroup, then $e_{l,*}$ and $d_{l,\backslash_*}$ are mutually inverse permutations of Q^+, i.e.,*

$$d_{l,\backslash_*}(e_{l,*}(M)) = M = e_{l,*}(d_{l,\backslash_*}(M))$$

for each leader $l \in Q$ and for every string $M \in Q^+$. ■

The authors of MQQ in [6, 7] noticed that when a quasigroup is represented as a Boolean function, then there exists a special class of quasigroups, called *multivariate quadratic quasigroups* (MQQs). Those MQQs can be of different types.

Definition 3 *A quasigroup $(Q, *)$ of order 2^d is called Multivariate Quadratic Quasigroup (MQQ) of type $Quad_{d-k}Lin_k$ if exactly $d - k$ of the Boolean polynomials f_i are of degree 2 (i.e., are quadratic) and k of them are of degree 1 (i.e., are linear), where $0 \leq k < d$.*

Theorem 2 *([6, 7]) Let $\mathbf{A_1} = [f_{ij}]_{d\times d}$ and $\mathbf{A_2} = [g_{ij}]_{d\times d}$ be two $d \times d$ matrices of linear Boolean expressions, and let $\mathbf{b_1} = [u_i]_{d\times 1}$ and $\mathbf{b_2} = [v_i]_{d\times 1}$ be two $d \times 1$ vectors of linear or quadratic Boolean expressions. Let the functions f_{ij} and u_i depend only on variables $x_1, \ldots, x_d$, and let the functions g_{ij} and v_i depend only on variables $x_{d+1}, \ldots, x_{2d}$. If*

$$\mathbf{Det}(\mathbf{A_1}) = \mathbf{Det}(\mathbf{A_2}) = 1 \text{ in } GF(2) \tag{3}$$

and if

$$\mathbf{A_1} \cdot (x_{d+1}, \ldots, x_{2d})^T + \mathbf{b_1} \equiv \mathbf{A_2} \cdot (x_1, \ldots, x_d)^T + \mathbf{b_2} \tag{4}$$

then the vector valued operation $*_{vv}(x_1, \ldots, x_{2d}) = \mathbf{A_1} \cdot (x_{d+1}, \ldots, x_{2d})^T + \mathbf{b_1}$ defines a quasigroup $(Q, *)$ of order 2^d that is MQQ. ■

In [6, 7] there are defined several algorithms for generating MQQs, for key generation, for encryption and for decryption. However, in this paper we will present only algorithms (and parts of algorithms) that are important for implementing decryption and encryption. We refer the reader to [6, 7] for other algorithms and for more detailed description of MQQ.

One important part of the MQQ algorithm is the use of the bijection of Dobbertin. Dobbertin has proved [5] that the function $\mathbf{Dob}(X) = X^{2^{m+1}+1} + X^3 + X$ is a bijection in $GF(2^{2m+1})$. Moreover it is multivariate quadratic too. In our implementation of MQQ public key cryptosystem we have used the bijection of Dobbertin for $m = 6$ i.e. a bijection in $GF(2^{13})$. From hardware resources point of view this means that for implementing the bijection of Dobbertin for $m = 6$ (actually its inverse) we need $2^{13} \times 13 = 106496$ bits of ROM.

The algorithm for decryption/signing by the use of the private key $(T, S, *_1, \ldots, *_8)$ is defined in Table 1.

The algorithm for encryption with the public key is straightforward application of the set of n multivariate polynomials $\mathbf{P} = \{P_i(x_1, \ldots, x_n) \mid i = 1, \ldots, n\}$ over a vector $x = (x_1, \ldots, x_n)$, i.e., $y = \mathbf{P}(x)$.

In order to implement MQQ encryption in FPGA we have presented the mapping $y = \mathbf{P}(x)$ as a matrix-vector multiplication, where the operation "+" is the logical XOR operation and where multiplication of two variables is actually the logical AND operation. Namely, every $P_i(x_1, \ldots, x_n)$ can be represented as:

$$P_i(x_1, \ldots, x_n) = a_{i,0,0} + \sum_{j=1}^{n} a_{i,j,0}x_j + \sum_{j=1}^{n-1} \sum_{k=j+1}^{n} a_{i,j,k}x_{k-j}x_k$$

where $a_{i,j,k} \in \{0, 1\}$. That means that we can represent the encryption as:

$$y = \mathbf{P}(x) \;\equiv\; y = \mathbf{A} \cdot X$$

```
Algorithm for decryption with the private key $(T, S, *_1, \ldots, *_8)$
Input: A vector $y = (y_1, \ldots, y_n)$.
Output: A vector $x = (x_1, \ldots, x_n)$ such that $\mathbf{P}(x) = y$.
1. Set $y' = T^{-1}(y)$.
2. Set $W = (y'_1,\ y'_2,\ y'_3,\ y'_4,\ y'_5,\ y'_6,\ y'_{11},\ y'_{16},\ y'_{21},\ y'_{26},\ y'_{31},\ y'_{36},\ y'_{41})$.
3. Compute $Z = (Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_7, Z_8, Z_9, Z_{10}, Z_{11}, Z_{12}, Z_{13}) = \mathbf{Dob}^{-1}(W)$.
4. Set $y'_1 \leftarrow Z_1,\ y'_2 \leftarrow Z_2,\ y'_3 \leftarrow Z_3,\ y'_4 \leftarrow Z_4,\ y'_5 \leftarrow Z_5,$
   $y'_6 \leftarrow Z_6,\ y'_{11} \leftarrow Z_7\ y'_{16} \leftarrow Z_8,\ y'_{21} \leftarrow Z_9,$
   $y'_{26} \leftarrow Z_{10},\ y'_{31} \leftarrow Z_{11}\ y'_{36} \leftarrow Z_{12},\ y'_{41} \leftarrow Z_{13}$.
5. Represent $y'$ as $y' = Y_1 \ldots Y_k$ where $Y_i$ are vectors of dimension 5.
6. By using the left parastrophes $\backslash_i$ of the quasigroups $*_i$,
   $i = 1, \ldots, 8$, obtain $x' = X_1 \ldots X_k$, such that:
   $X_1 = Y_1$, $X_2 = X_1 \backslash_1 Y_2$, $X_3 = X_2 \backslash_2 Y_3$ and
   $X_i = X_{i-1} \backslash_{3+((i+2) \bmod 6)} Y_i$.
7. Compute $x = S^{-1}(x')$.
```

Table 1: Algorithm for decryption

where

$$\mathbf{A} = \begin{bmatrix} a_{1,0,0} & a_{1,1,0} & \cdots & a_{1,n,0} & a_{1,1,2} & a_{1,1,3} & \cdots & a_{1,n-1,n} \\ a_{2,0,0} & a_{2,1,0} & \cdots & a_{2,n,0} & a_{2,1,2} & a_{2,1,3} & \cdots & a_{2,n-1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,0,0} & a_{n,1,0} & \cdots & a_{n,n,0} & a_{n,1,2} & a_{n,1,3} & \cdots & a_{n,n-1,n} \end{bmatrix}$$

and

$$X = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \\ x_1x_2 \\ x_2x_3 \\ \vdots \\ x_{159}x_{160} \\ x_1x_3 \\ x_2x_4 \\ \vdots \\ x_{158}x_{160} \\ \vdots \\ x_1x_{159} \\ x_2x_{160} \\ x_1x_{160} \end{bmatrix}.$$

Note that the Boolean matrix $\mathbf{A}$ is the public key and it is an $n \times (1 + n + \frac{n(n-1)}{2})$ matrix, and the vector X is an $(1 + n + \frac{n(n-1)}{2}) \times 1$ vector obtained from the vector $x = (x_1, \ldots, x_n)$.

3 Hardware design of the encryption procedure

In this and in the following section we will describe a hardware implementation of 160–bit MQQ encryption and decryption. Our goal was to prove or disprove the claim of the authors of MQQ that it can have operational speed that is same as that of the block ciphers. To achieve that goal, we have implemented 160–bit MQQ in VHDL, and have constructed a parallel design, where every component in the design completes its computation in as minimum as possible clock cycles, and with small duration of each cycle. We have used Xilinx Virtex-5 FPGA family and its synthesizing tool "ISE Foundation 10.1".

One of the biggest problems that we faced in this part was the size of the public key (the matrix $\mathbf{A}$) and the goal to finish the matrix-vector multiplication as fast as possible. For 160–bit MQQ, the Boolean matrix $\mathbf{A}$ has 160 rows and 12881 columns. Although theoretically it is possible to perform a

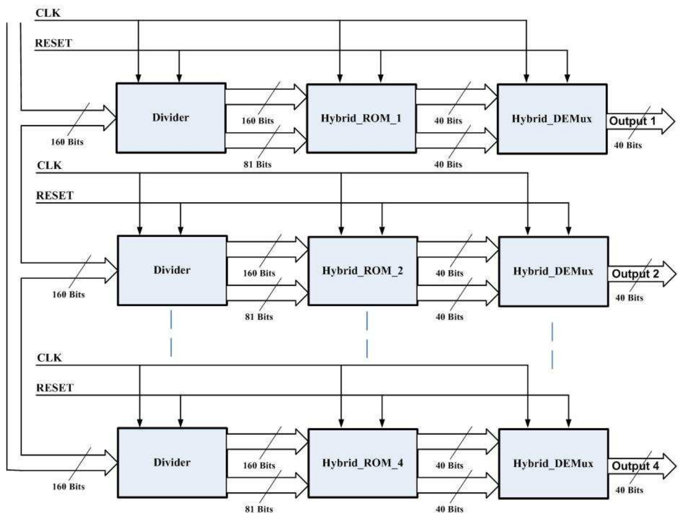

Figure 2: Architecture for encryption.

matrix-vector operation in one cycle, performing operations on the whole matrix in one single FPGA was a source of constant compiler warnings and complete compiler blackouts. And since our work can be seen as a “proof of a concept” to simplify the computations we decided to put the public key into a ROM. For real operational and flexible use of MQQ, Virtex-5 would offer a plenty of RAM.

Figure 2 shows the complete architecture for the entire encryption process, which includes four identical hierarchies; each one contains three main hardware operative parts, named: “Divider”, “Hybrid_ROM” and “Hybrid_DEMux”. Dividing (splitting) the public key $\mathbf{A}$ in four parts, i.e. its splitting in four FPGA chips that work in parallel was done in order to overcome synthesizing difficulties with the big public key $\mathbf{A}$ in ROM in one chip.

So, the idea is to implement a matrix-vector multiplication $\mathbf{A} \cdot X$ in a classical block manner, where we represented the matrix $\mathbf{A}$ as

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \\ \mathbf{A}_3 \\ \mathbf{A}_4 \end{bmatrix},$$

where every submatrix $\mathbf{A}_i, i = 1, 2, 3, 4$ is a 40×12881 Boolean matrix. The operation $\mathbf{A}_i \cdot X$ is realized in the i-th FPGA chip.

The component “Divider” consists of four operative parts, and is shown in Figure 3. The role of this part is to transfer the input data of 160 bits from “REG_1” and by using “Combinational AND Gates” block to expand the input data into 12881 bits. Then those 12881 bits go to “Register_Y” to adjust the synchronization between the data bits (a technique described in Xilinx Virtex-5 user guide [24]). After this, the data are divided in two parts by the multiplexer Hybrid_Mux. This multiplexer has two kinds of inputs, first 80 inputs have 160 bits as a data width and the last input has 81 bits. Similar is with its output, i.e. it has two branches: first has 160 bits and the second has 81 bits.

The second component “Hybrid_ROM” contains 44 operative parts, and its implementation is shown in Figure 4. This part has 40 ROMs (according to 40 rows of the submatrix $\mathbf{A}_i$) and a component “Parallel ROM”. All of them are used in parallel. The output of these ROMs goes directly with the output of the “Divider” in the components “Combinational Logic Gates 1” and “Combinational Logic Gates 2” as shown in Figure 4. The matrix-vector multiplication is realized in these two components by using AND and XOR gates between the outputs of ROMs and “Divider”. The output of the component “Hybrid_ROM” has two sequences, 40 bits each.

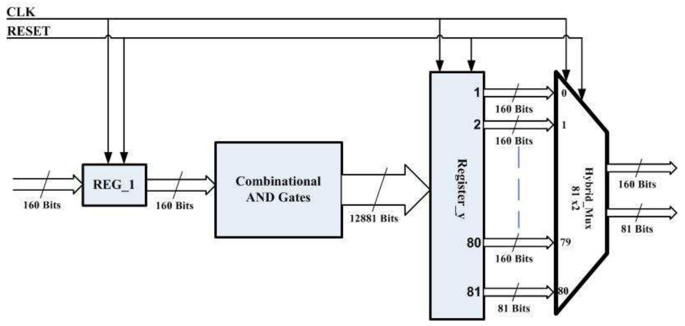

Figure 3: Internal architecture of the “Divider” component.

The role of the third component “Hybrid_DEMux” is to finalize the matrix-vector multiplication that is started in the component “Hybrid_ROM”. It has four operative parts. Its implementation is shown in Figure 5. The output from the “Hybrid_ROM” component goes directly to “Hybrid_DEMux”. The term “Hybrid” comes from the specifics of the de-multiplexer “DEMux”, because this “DEMux” has two inputs. The output of the DEMux will go to the component “Combinational Logic Gates 3” through the “Register_Z” component. This register is used to keep the synchronization between data (see [24]). Then the output of “Combinational Logic Gates 3” goes again to another synchronization register “Register_W” in order to keep the synchronization of the final output.

This implementation of the MQQ encryption is fully pipelined. It takes initially 82 cycles to encrypt 160 input bits, but then, the encryption engine can output 160 bits in every cycle.

4 Hardware design of the decryption procedure

Figure 6 shows the complete architecture for the entire decryption procedure described in Table 1. It has four main hardware components: “Private_Matrix T^{-1}”, “Sequencer”, ”Dobbertin” and “Private_Matrix S^{-1}”. Actually the first and the fourth component are structurally the same (with different Boolean matrices T and S).

The structure of the first and the fourth component, “Private_Matrix T^{-1}” and “Private_Matrix S^{-1}” is shown in Figure 7. It implements Steps 1 and Step 7 from the decryption algorithm described in Table 1. It has 160 parts and each part computes a dot product between two Boolean vectors of length 160. One vector is the input vector of 160 bits and the other vector is one row from the private matrix T^{-1} or S^{-1} placed as a fixed ROM. The role of the operation “+” in this computation of the dot product is a logical XOR (represented by the “Bit by bit XOR” components in the Figure 7), and the role of multiplication is the logical AND operation (represented by the “AND Gate” parts in the Figure 7). The output of these parallel 160 dot products have to go to the next phase through the component “Register_X” to keep the synchronization between the data (see [24]).

The second component “Dobbertin_ROM” is shown in Figure 8. It takes the data which come from “Private_Matrix T^{-1}”. Actually it changes just 13 bits, which positions are determined in Step 2, Step 3 and Step 4 in the decryption algorithm described in Table 1 and computes the inverse Dobbertin function in $GF(2^{13})$. The realization of this component is a simple lookup table reading from a ROM with 2^{13} entries and it outputs 13 bits. The output of this component, goes to the third component “Sequencer”.

The third component “Sequencer” is shown in Figure 9. This is the most complex part and it implements the Step 5 and the Step 6 from the decryption algorithm described in Table 1. This part contains 32 5–bit registers, 2 Multiplexers, a Master ROM, a control component, a “DEMux 1×32” component and two counters.

The internal structure of the “Master ROM” component is shown in Figure 10. It has 8 ROMs

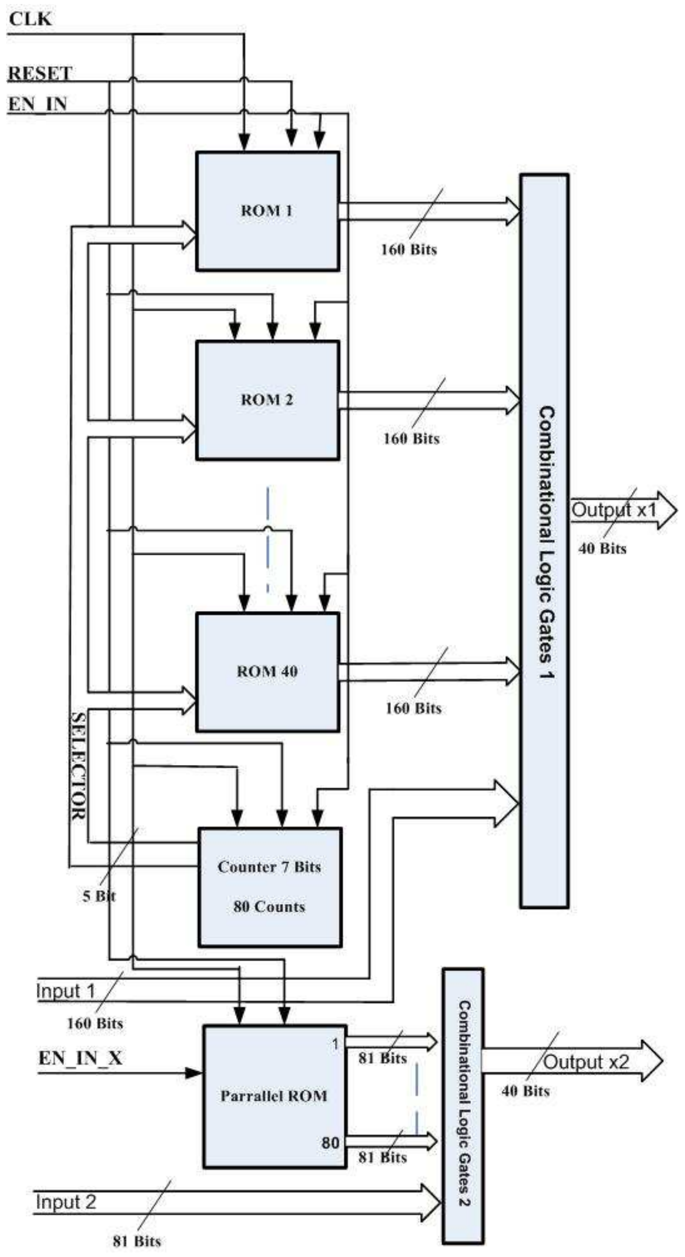

Figure 4: Internal architecture of the "Hybrid_ROM" component.

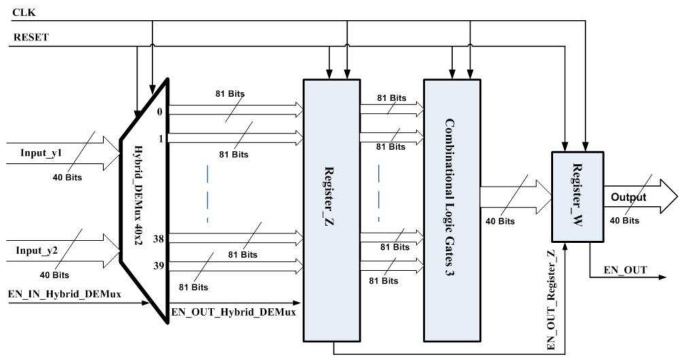

Figure 5: Internal architecture of the "Hybrid_DEMux" component.

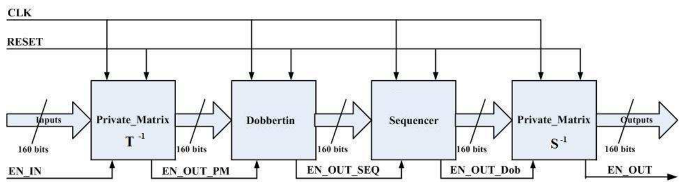

Figure 6: Architecture for decryption.

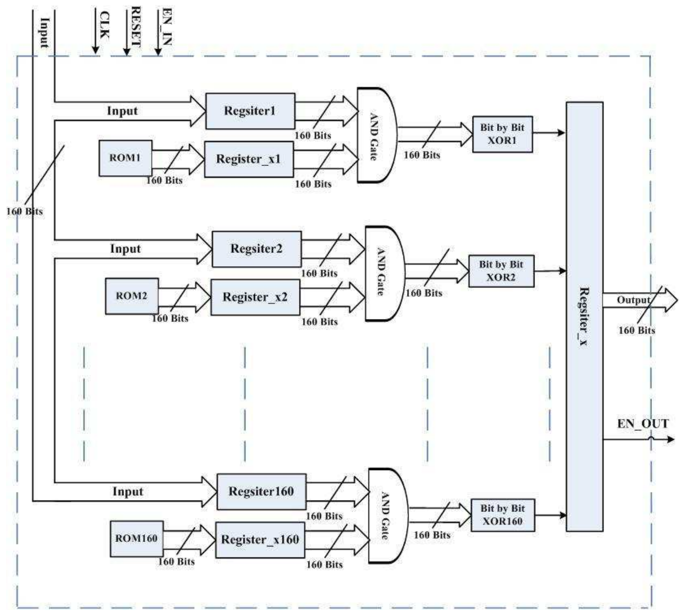

Figure 7: Internal architecture of the "Private_Matrix" component.

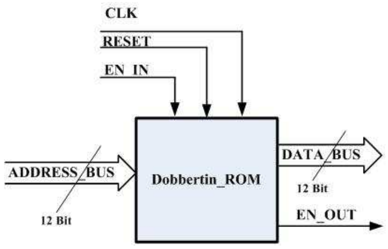

Figure 8: The "Dobbertin_ROM" component.

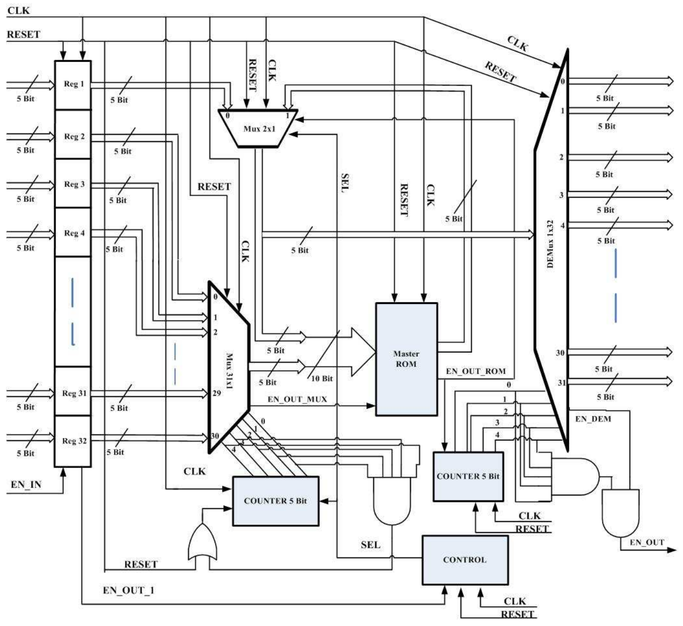

Figure 9: Internal architecture of the "Sequencer" component.

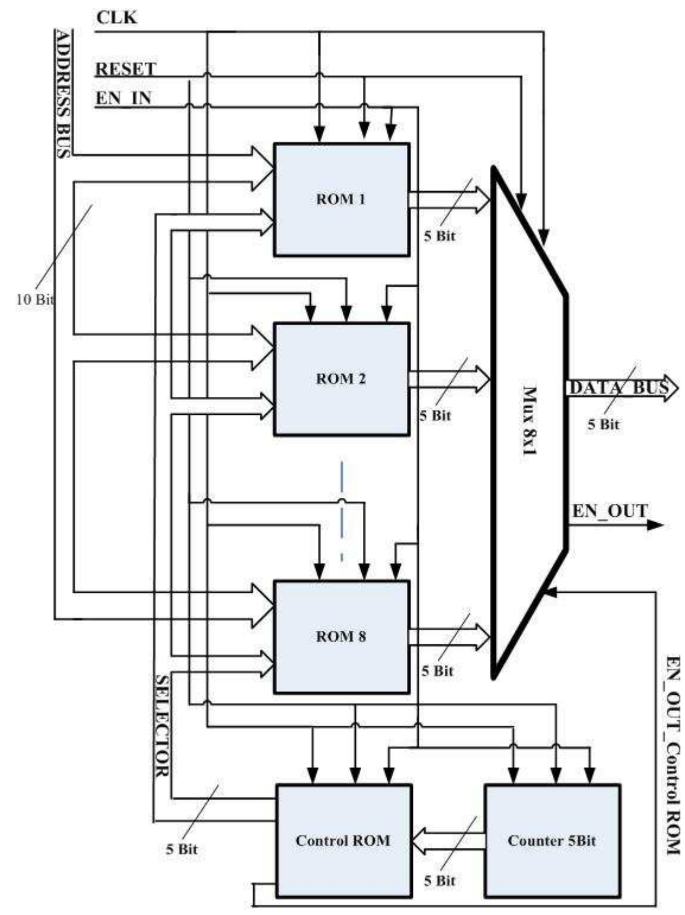

Figure 10: Internal architecture of the "Master ROM" component.

that work in parallel, one control component one 5–bit counter and one 8 × 1 multiplexer.

This implementation of the MQQ decryption takes 100 cycles to decrypt 160 input bits.

5 Performance comparison

We have realized MQQ public key scheme in Virtex-5 chip: xc5vfx70t-2-ff1136. The summary of the synthesizing of our VHDL implementation of the 160–bit MQQ using the Xilinx tool "ISE Foundation 10.1" is given in Table 2.

MQQ public key scheme has a property to be highly parallelized and that property can be evidently demonstrated in its hardware realization. On the other hand, all popular public key algorithms (RSA, DH, ECC, DSA, ECDSA) have essentially a sequential nature. Thus, using highly parallelized hardware such as FPGA, the speed difference between MQQ and those popular public key algorithms is five orders of magnitude. More concretely, implemented in FPGA, MQQ is more than 10,000 times faster than DSA, RSA or ECDSA, and is comparable or even faster than the symmetric block cipher AES.

In Table 3 we compare the speed of 160–bit MQQ with the speed of 1024–bit RSA realized in Xilinx Virtex-5 FPGA chip by the company "Helion Technology Limited" [9]. In the same table we also give the speed of AES (128–bit key) realized in Xilinx FPGA chip in the same Virtex-5 family and by the same company.

	Slice registers	LUTs	Initial delay (ns)	Cycles per operation	Max. frequency (MHz)
Encryption (for each chip)	13,137	25,285	3.614	1	276.7
Decryption	1,148	6,993	4.010	100	249.4

Table 2: Synthesis Results for 160–bit MQQ realized in Virtex-5 chip xc5vfx70t-2-ff1136

Algorithm name	1024-bit RSA, encrypt/decrypt	160–bit MQQ, encrypt/decrypt	128–bit AES, encrypt/decrypt
FPGA type	Virtex-5, XC5VLX30-3	Virtex-5, XC5VFX70T-2	Virtex-5
Frequency	251 MHz	276.7 / 249.4 MHz	325 MHz
Throughput	40 Kbps	44.27 Gbps / 399.04 Mbps	3.78 Gbps

Table 3: Hardware performances of 1024–bit RSA, 160–bit MQQ and 128–bit AES on Xilinx Virtex-5 FPGAs

6 Conclusions

We have implemented in FPGA a 160–bit instance of the newly published public key scheme MQQ. The results of our implementation show that in hardware, MQQ public key algorithm in encryption and decryption (that means also in verification and signing) can be as fast as a typical block cipher and is several orders of magnitude faster than most popular public key algorithms like RSA, DH or ECC.

References

[1] M. Ajtai, "Generating hard instances of lattice problems", in *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, 1996, pp. 99–108.

[2] V. D. Belousov, *Osnovi teorii kvazigrup i lup* (in Russian), Nauka, Moscow, 1967.

[3] J. Dénes, A. D. Keedwell, *Latin Squares and their Applications*, English Univer. Press Ltd., 1974.

[4] W. Diffie and M. Hellman, "New Directions in Cryptography", *IEEE Trans. Information Theory*, Vol. IT-22, No 6, (1976), 644–654.

[5] H. Dobbertin, "One-to-one highly nonlinear power functions on $GF(2^n)$", *Appl. Algebra Eng. Commun. Comput.*, Vol. 9(2), pp. 139-152, 1998.

[6] D. Gligoroski, S. Markovski and S. J. Knapskog, "Multivariate Quadratic Trapdoor Functions Based on Multivariate Quadratic Quasigroups", Proceedings of the AMERICAN CONFERENCE ON APPLIED MATHEMATICS (MATH '08), Cambridge, Massachusetts, USA, March 24-26, 2008.

[7] D. Gligoroski, S. Markovski and S. J. Knapskog, "Public Key Block Cipher Based on Multivariate Quadratic Quasigroups",Cryptology ePrint Archive, Report 2008/320, http://eprint.iacr.org/

[8] O. Goldreich, S. Goldwasser and S. Halevi, "Public-Key Cryptosystems from Lattice Reduction Problems", LNCS, Vol. 1294, pp. 112–131, 1997.

[9] Helion Technology Limited, "Modular Exponentiation Engine for RSA and DH (ModExp)", February 16, 2007, Available: `http://www.xilinx.com/publications/3rd\_party/products/Helion\_ModExp\_AllianceCORE\_data\_sheet.pdf`

[10] J. Hoffstein, J. Pipher and J. H. Silverman, "NTRU: A ring based public key cryptosystem", LNCS, Vol. 1433 pp. 267–288, 1998.

[11] H. Imai and T. Matsumoto, "Algebraic methods for constructing asymmetric cryptosysytems", in *Proceedings of 3rd Intern. Conf. AAECC-3*, LNCS Vol. 29, pp. 108–119, 1985.

[12] A. Kipnis, J. Patarin, and L. Goubin, "Unbalanced Oil and Vinegar signature schemes", in *Advances in Cryptology, EUROCRYPT 1999*, LNCS Vol. 1592, pp. 206–222, 1999.

[13] K.H. Ko, S.J. Lee, J.H. Cheon, J.W. Han, J. Kang C. Park, "New Public-Key Cryptosystem Using Braid Groups", in *Advances in Cryptology, CRYPTO 2001*, LNCS, Vol. 1880, pp. 166–184, 2001.

[14] N. Koblitz, "Elliptic curve cryptosystems", in *Mathematics of Computation 48*, 1987, pp. 203–209.

[15] S. Markovski, D. Gligoroski, V. Bakeva, "Quasigroup String Processing: Part 1", Maced. Acad. of Sci. and Arts, Sc. Math. Tech. Scien. XX 1-2, pp. 13–28, 1999.

[16] R.J. McEliece, "A Public-key cryptosystem based on algebraic coding theory", DSN Progress Report, Jet Propulsion Laboratory, Pasadena, CA, 1978, pp. 114–116.

[17] V. Miller, "Use of elliptic curves in cryptography", in *CRYPTO 85*, 1985.

[18] J. Patarin, "Hidden Field Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of asymmetric algorithms", in *Advances in Cryptology, EUROCRYPT 1996*, LNCS Vol. 1070, pp. 33–48, 1996.

[19] J. Patarin, "Hidden Field Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of asymmetric algorithms - EXTENDED VERSION OF THE EUROCRYPT 1996 paper". Available from the author.

[20] M.O. Rabin, "Digital Signatures and Public-Key Functions as Intractable as Factorization", Technical Report MIT/LCS/TR-212, M.I.T., 1978.

[21] R. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Comm. ACM*, Vol. 21, No 2, 1978, pp. 120–126.

[22] A. Shamir, "Efficient signature schemes based on birational permutations", in *Advances in Cryptology, CRYPTO 1993*, LNCS, Vol. 773, pp. 1–12, 1993.

[23] J. D. H. Smith, *An introduction to quasigroups and their representations*, Chapman & Hall/CRC, ISBN 1-58488-537-8, 2007.

[24] Xilinx, "Virtex-5 Field Programmable Gate Arrays-product Specification", `http://www.xilinx.com/support/documentation/user\_guides/ug190.pdf`, 2008.