An FPGA Co-Processor Implementation of Homomorphic Encryption

David Bruce Cousins, John Golusky, Kurt Rohloff, Daniel Sumorok
Raytheon BBN Technologies
Cambridge, Massachusetts USA
{dcousins, jgolusky, dsumorok, krohloff}@bbn.com

Abstract— **One of the goals of the DARPA PROCEED program has been accelerating the development of a practical Fully Homomorphic Encryption (FHE) scheme. For the past three years, this program has succeeded in accelerating various aspects of the FHE concept toward practical implementation and use. FHE is a game-changing technology to enable secure, general computation on encrypted data on untrusted off-site hardware, without the data ever being decrypted for processing. FHE schemes developed under PROCEED have achieved multiple orders of magnitude improvement in computation, but further means of acceleration, such as implementations on specialized hardware, such as an FPGA can improve the speed of computation even further.**

The current interest in FHE computation resulted from breakthroughs demonstrating the existence of FHE schemes [1, 2] that allowed arbitrary computation on encrypted data. Specifically, our contribution to the Proceed program has been the development of FPGA based hardware primitives to accelerate the computation on encrypted data using an FHE cryptosystem based on NTRU-like lattice techniques [3] with additional with additional support for efficient key switching and modulus reduction operations to reduce the frequency of bootstrapping operations [4]. Cipher texts in our scheme are represented as rectangular matrices of 64-bit integers. This bounding of the operand sizes has allowed us to take advantage of modern code generation tools developed by Mathworks to implement VHDL code for FPGA circuits directly from Simulink models. Furthermore the implicit parallelism of the scheme allows for large amounts of pipelining in the implementation in order to achieve efficient throughput. The resulting VHDL is integrated into an AXI4 bus "Soft System on Chip" using Xilinx platform studio and a Microblaze soft core processor running on aVirtex7 VC707 evaluation board. This report presents new Simulink primitives that had to be developed to deal with these new requirements.

Keywords—Fully Homomorphic Encryption; Co-processor; SIMULINK; FPGA

I. Introduction - A Quick Review of Fully- and Somewhat- Homomorphic Encryption

Our team recently published our work to design, implement and evaluate a scalable FHE scheme which addresses the limitations for secure arbitrary computation [4]. Our implementation uses a variation of a not previously implemented bootstrapping scheme [5] simplified for power-of-2 rings. We also use a "double-Chinese Remainder Transform (CRT)" representation of cipher texts which is discussed in [6]. With this double-CRT representation, we can select parameters so that cipher texts are secure when represented as matrices of 64 bit integers, but still support the secure execution of programs on commodity computing devices without expending unnecessary computational overhead manipulating large multi-hundred-bit or even multi-thousand-bit integers. Additionally, the parallelism implicit in this data representation is easily exploited to achieve efficiencies during implementation.

Our implementation encrypts a plaintext bit into a two dimension array of 64 bit unsigned integers[1]. We use a residue number system implementation to represent cipher texts as T sets of length-N integer vectors. A ring in the tower entry t has a unique modulus q_t which bounds all entries in that ring. The n dimension is known as the ring size, and the t dimension as the tower size. This representation allows us to operate in parallel on the smaller bit width modulo q_t values instead of on a single modulus q of much larger bit width, where $q = q_1 * q_2 * \ldots * q_T$ for pairwise co-prime moduli q_i..

As outlined in [4], our implementation requires only a few elementary operations to be implemented on the FPGA hardware in order to achieve large run time speedups over conventional CPU implementations. These operations are:

- RingAdd: $c_{n,t} = (a_{n,t} + b_{n,t}) \% q_t$
- RingSub: $c_{n,t} = (a_{n,t} - b_{n,t}) \% q_t$
- RingMul: $c_{n,t} = (a_{n,t} * b_{n,t}) \% q_t$

All three of the above operations can be parallelized or pipelined over both n and t . Also required are the

- CRT and Inverse CRT, which are implemented as a Number Theoretic Transform [7] coupled with a pre- or post-RingMul with an appropriate Twiddle Vector.
- Round: A function to perform modulo rounding using different tower moduli (detailed below).

In our cryptosystem, two key operations are defined: EvalAdd and EvalMult. When our parameters are chosen such that a single plaintext bit is encrypted, the resulting operations on the encrypted data are XOR and AND respectively. These

Sponsored by Air Force Research Laboratory (AFRL) Contract No. FA8750-11-C-0098. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

[1] While the actual number of bits is determined by the parameter selection of the cryptosystem, we select 64 as our maximum dimension for FPGA implementation.

two operations allow us to implement any Boolean operation of input cipher text [2].

Our crypto-system, like many FHE systems, is random (noisy) in nature. Because of this, only a limited number of operations can be performed on the encrypted data before the noise dominates and decryption is no longer guaranteed. EvalAdd does not add noise to the system, so an unlimited number of such operations are allowed to be chained together. EvalMult however does add noise, and this limits the number of such operations that can be chained together. The double CRT representation allows a very straightforward implementation that controls this noise. This requires the use of both key switching and modulus reduction whenever an EvalMult is performed. The combination of these three steps is known as a Composed EvalMult (CEM). The property of CEM is that for a pair of inputs of a given tower size *t*, the output is a cipher text of tower size *t-1*. Thus for an initial tower size of *T*, at most *(T-1)* CEM operations can be performed before the noise in the cypher-text grows beyond the point where it can be reliably decrypted. An implementation that has a limit to the allowable number of Homomorphic operations is called *Somewhat Homomorphic*.

The dimensions of the cryptosystem are determined algorithmically, and are a function of security required and the number of CEM operations required to implement the desired application. If the number of operations required by the application exceeds O(16), then a Bootstrapping operation will be required to reset the noise generated by the cryptographic operations. Bootstrapping is currently on the order of 10 CEM equivalent operations for reasonable security parameters. Bootstrapping has the property of taking a cipher text of tower size *t*, and generating a new 'refreshed' cipher text of the system's original tower size *T*. Thus an unlimited number of operations can be performed on the data. This kind of implementation is called *Fully Homomorphic*. The remainder of our paper will discuss the current FPGA implementation of the functions required for Somewhat Homomorphic operation. Our planned implementation of the functions needed for Fully Homomorphic operation will be implemented in our final phase of the program this year.

II. VHDL Implementations of Fast Modulus Arithmetic and Chinese Remainder Transforms (CRT) Using Simulink-Based Models

A. Optimisations and Refinements To Previous Implementations

We have previously reported on our Simulink-based implementations of the three modulus arithmetic functions, as well as the forward and inverse CRT functions[8, 9]. Our current work has updated these implementations to allow VHDL code generation with a doubling of circuit clock speeds to 200 MHz. This was done by performing the following optimizations.

Mathworks determined that by selecting synchronous vs. asynchronous reset in the Simulink to HDL generation parameters, the resulting VHDL mapped more efficiently into the registers built into the DSP48E blocks on the Vertex 7 FPGA, increasing the efficiency of the resulting mapped VHDL by eliminating extra routing traces.

The previous circuits were designed to run at a minimum speed of 100 MHz. We determined that adding explicit pipelining stages in the form of delay lines to the model enabled the Xilinx tools to better optimize FPGA mapping during place and route pipelining stages. Specifically pipelines were added between arithmetic operations within the RingAdd (4 stages), RingSub (3 stages), RingMul (188 stages) models. Since our target ring size can be as large as 2^{14}, and all the towers of a variable are processed sequentially, the delay incurred from filling the pipeline is expected to be minimal.

Once the models were maximally pipelined, we identified several large (64 by 64 bit) product blocks within our RingMul Barret multiplication implementation [9, 10] as being the slowest components, and re-implemented them as an expanded multiplication model consisting of four parallel 32 by 32 bit products, and a pipelined accumulation of partial sums. This further increased the achievable clock speeds. We discovered that adding additional pipelines of length four, both before and after each resulting smaller product block further allowed the Xilinx optimizer to break these product blocks into multiple DSP48E multipliers in a distributed fashion. This allowed the RingMul circuit to perform at speeds in excess of 350 MHz, well in excess of our target 200 MHz.

Several of our circuits utilize lookup tables, both for storing the moduli q_i and for storing various twiddle table entries for the CRT and inverse CRT. Our previous direct implementation of the table lookup using the Simulink Lookup function block maps the resulting ROM directly into gate circuitry. This can increase the place and route drastically for very large tables, and also can result in less efficient circuits. Mathworks determined that by placing an additional delay line, with a `"ResetType = none"` HDL block property let the Xilinx tools map the table to block ram in the FPGA, which is a more efficient utilization of resources on the chip.

B. FPGA Hardware Selection

Our FPGA selection was driven by the need for a large number of hardware multipliers on the chip. Due to cost constraints we wanted to use a commercial off-the-shelf FPGA board for our experiments. Our selection of the Virtex 7 VC707 evaluation board was driven by the following sizing requirements. Our target ring size of 2^{14} requires 1110 DSP48 blocks for the CRT and the same number for the inverse CRT. The VC707 has a Virtex 7 485T chip which contains 2800 such blocks, more than sufficient to implement our projected set of FHE primitives. Additionally, we require on-board DDR memory for storage of encrypted variables, and high speed Ethernet and PCI interfaces to exchange data with the host computer. All these are present on the VC707.

[2]Any arbitrary Boolean function can be constructed from NAND operations. Since NOT(a) == XOR(a, 1), and NAND(a, b) == NOT(AND(a, b)), the two Homomorphic operations are a sufficient set.

C. FPGA System Architecture

The design goal of our FPGA system was to be able to operate as an attached processor to accelerate the FHE primitive operations in way that allows one to chain together several operations in order to minimize the overhead due to data transfer. An attached processor design was developed in which a software programmable microcontroller would manage I/O communications with the host via Ethernet or PCI memory map, manage on board data storage in the form of an encrypted register file, and manage data transfer to and from the FHE primitive modules in as efficient manner as possible.

We decided to use the Xilinx Platform Studio Microblaze soft core processor and AXI4 interconnect architecture to implement the attached FHE processor. Fig. 1 shows a system block diagram of the resulting system. The Xilinx platform studio enables us to implement our FHE primitives as streaming co-processors on the AXI bus. An AXI4 lite bus is used to set control parameters of our Ring operation circuits, such as ring size, and tower size.

The main AXI4 interconnect is a 256 bit bus connecting the DDR ram with the various FHE primitives. The I/O rate into and out of DDR memory limits the overall processing speed of the system. Our RingAdd, RingSub and RingMul primitives each require two input streams and one output streams. Fig. 2 shows how we currently integrate our FHE primitives with the AXI4 stream interconnect. Each of these three operations is parallelized across ring elements as well as tower indices. These data streams are implemented using a pair of AXI4 DMA controllers, each handling one input and one output. Data is clocked in and out of the bus at 400 MHz, and streamed via individual AXIS buses between the DMAs to the AXI stream blocks where they are buffered with FIFOs and split into eight parallel 64 bit input data streams, and four 64 bit output data streams. Current implementations of these three functions are clocked at 100 MHz, so four parallel instantiations of each operation are used to keep the I/O pipelines full. Future implementations of these primitives are planned to be clocked at 200 MHz, and as such will only support two instantiations in parallel.

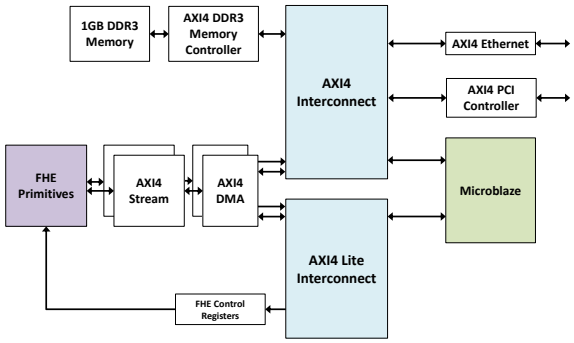

Figure 1: System block diagram showing major components and the AXI4 interconnect.

The forward and inverse CRT modules require slightly different interfaces. CRT operations are parallelizable across tower entry but not across ring index. Thus CRT's cannot be parallelized in the same way as the Ring operations. Currently we have a single CRT or inverse CRT at a time operating at 100 MHz. Future implementations will run these two operations at 200 MHz, but the multiplier resources required for the planned ring size of 2^{14} will prohibit mapping more than one forward and one inverse CRT onto the 485T chip.

D. Microblaze Software Architecture

The Xilinx platform studio is used to implement a Microblaze soft core processor. The system architecture is based on the demo hardware self-test example that is provided with the Xilinx board. The software architecture is based on the web-service example provided with the Xilinx Virtex 6 ML605 evaluation kit, updated with the Xilinx SGMII 144 Ethernet controller. The software controlling the system on the Microblaze is written in C code. The PC "host" end of the software interface is also written in C. The host interface currently is implemented in two versions. The first is a stand-alone test bench that can test and exercise the operation of the attached FHE processor. The second version interfaces with Matlab via a file interchange mechanism to support demonstration Homomorphic Encryption application programs. The interfaces use either Ethernet or PCI bus I/O based on compile flags.

The system software is multithreaded to allow the use of Ethernet TCP/IP socket I/O. A network thread manages socket level I/O between the host and the attached processor. Another thread reads the incoming messages from the socket, parses the commands received and dispatches execution to various subroutines. The PCI interface is written to emulate the buffer I/O of the Ethernet interface, allowing the same software to be used for both Ethernet and PCI operation.

The DDR3 ram is partitioned into a set of register data structures, as well as a set of internal registers to store constants used in our encryption schemes. Each register can hold one encrypted bit in the form of a two dimensional vector of

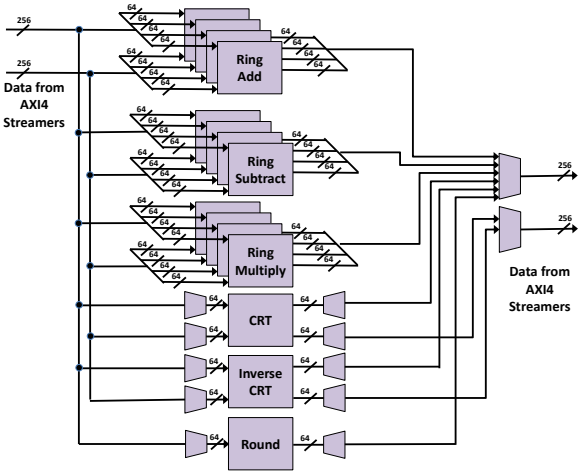

Figure 2: Integration of FHE primitives with the AXI stream data streams.

unsigned long longs that are allocated out of DDR ram. One dimension (the fastest index) is the ring size N and is a compiled constant. The other dimension, the tower size, varies with the state of the register. Typically registers are loaded into the FHE coprocessor with a fixed starting number of the tower elements (up to MAX_TOWER_SIZE = 32 elements). We eliminate the highest tower entries one by one as each CEM operation is performed.

The registers are allocated out of heap in the DDR ram. There are three flavors of registers: Input, Output and Scratch. This design decision was made in order to allow us to later segregate I/O and scratch registers into different memory locations if that were to increase throughput (allowing simultaneous host access to the I/O registers while the FGPA was processing with the Scratch registers. The quantity of each register type is software defined at compile time but there is usually a small numbers of Input and Output registers and as many Scratch registers as will utilize all the available heap space. Control structures mark the current tower size of each register, and if the register is used or not. Registers are allocated so they are aligned to 32 byte address boundaries in order to allow the AXI4 DMA engines to move the register data into and out of the FHE primitives. This format allows the contents of an entire register (all used towers) to be streamed with only one DMA transfer.

The communication protocol between the PC host and the FPGA board is message based. The messages are in ASCII. Messages can span multiple socket buffers; with multiple socket calls made until enough text has been parsed to complete a message (double cr/lf indicates the end of a message). Each message can contain several instructions to the processor, separated by cr/lf). Each processor instruction is then parsed. The parsing test starts with a keyword that defines the rest of the instruction format. The keywords are shown in Table 1. The system's assembly language has the syntax shown in Table 2.

TABLE I. CONTROL PROTOCOL KEYWORDS

Key word	Function
LOAD	Transfer the contents of the message (ASCII) into a particular Input register.
GET	Request the contents of a particular output register to be loaded into an ASCII message buffer and sent back to the host.
STATUS	Generates a short report on the FPGA board console for debugging showing the contents of all used registers, a listing of the current program loaded.
PROG	Loads a sequence of operations to be performed on the register data, in a simple assembly language.
RUN	Starts a software Finite State Machine to run the stored program to completion.
CRT, ICRT, CEM	A single command that will LOAD two registers, perform a forward CRT, inverse CRT or Composed EvalMult on them and GET the resulting output. Used for accelerating applications that only require these three operations.
RESET	Resets the system to its original state.

TABLE II. AVALIABLE OPCODES FOR HOMOMORPHIC ENCRYPTED PROGRAMS

Opcode	Example	Description
LOAD	R1 = LOAD(In0)	Moves data from an input register to scratch register, all active tower elements are moved.
STORE	Out4 = STORE(R3)	Moves data from a scratch register to output register, all active tower elements are moved.
RADD	R2 = RADD(R3, R4)	Sets up DMAs of the two input and one output registers to the RingAdd circuit. All active tower elements are processed I one large data flow.
RSUB	R2 = RSUB(R3, R4)	Same as RingAdd, except the RingSub circuit is the target/source of the I/O DMAs.
RMUL	R2 = RMUL(R3, R4)	Same as RingAdd, except the RingMul circuit is the target/source of the I/O DMAs.
CRT	R3= CRT(R1, R2)	Same as RingAdd, except the input and output registers are used as endpoints for pairs of DMA transfers, each moving one half of the ring data. Note second input register is used as a scratch register so is contents are destroyed.
ICRT	R2 = ICRT(R4, R5)	Same as CRT except an inverse CRT circuit is used.
EMULC	R2 = EMULC(R3, R4)	Executes a ComposedEvalMult, in software which in turns executes several Ring primitives (see below). Note that output register is one tower smaller than the input registers.

An example simple program in now given in Table 3. The program first moves encrypted data from input register 0, to scratch register 0, then repeats the process for a second input variable to register 1. It then computes a RingAdd, RingSub and RingMul using the two inputs, and storing the result in scratch registers 2, 3 and 4 respectively. It then stores those three results in output registers 0, 1 and 2 respectively.

Typical system operation would be for the user to execute two LOAD commands to load the contents of input registers 0 and 1 with encrypted data (the encryption being done on the secure host). The user then executes a RUN command to allow the Homomorphic operations to be run on the unsecure FPGA processor. Then subsequent calls to GET commands will

TABLE III. SAMPLE PROGRAM

```
R0 = LOAD(In0)
R1 = LOAD(In1)
R2 = RADD(R0,R1)
R3 = RSUB(R0,R1)
R4 = RMUL(R0,R1)
Out0 = STORE(R2)
Out1 = STORE(R3)
Out2 = STORE(R4)
```

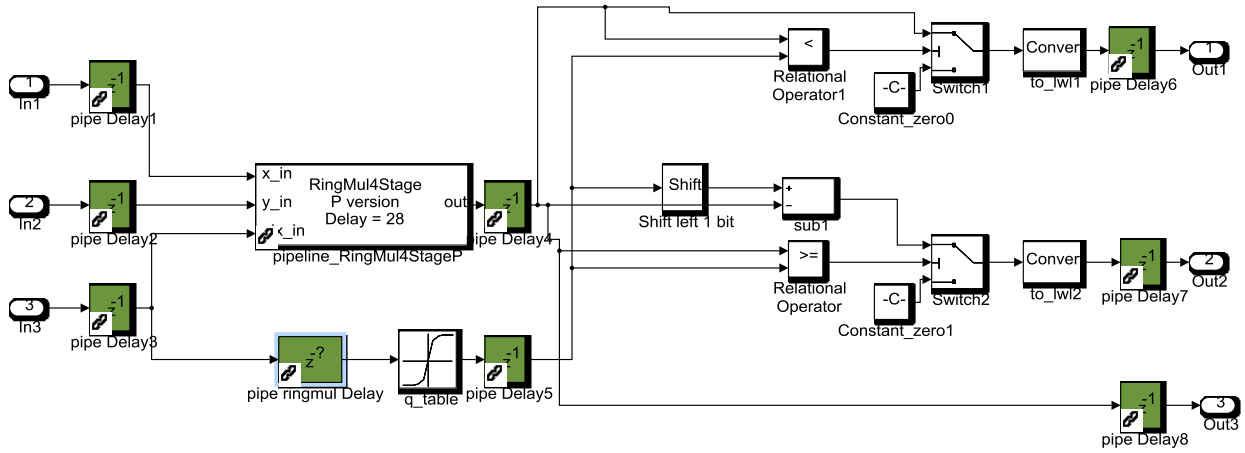

Figure 3: Simulink Model of Round Function.

transfer the resulting encrypted result data back to the host. Finally decryption would be done on the secure host.

E. Microcode Implementation of ComposedEvalMult

As mentioned above, one of the new functions implemented in our system is the ComposedEvalMult (CEM) which is fully detailed in [4]. This function is implemented in our software controller as a series of C function calls, all but one of which are executed with previously existing primitives. First, a RingMul operation performs the multiply. Next a key-switch operation is performed consisting of another RingMul of the product with a hint variable defined by the cryptosystem. Then, a modulus reduction operation is performed on the single highest tower entry of the result which consists of an inverse CRT and a new Rounding operation.

This Rounding operation is implemented as a new hardware function because it contains operations not available in the other ring functions. Fig. 3 shows the Simulink Model consisting of a modified EvalMult operation (using a modified set of moduli qi), and a pair of operations selected by the range of the result which ensure the output is bounded within an appropriate range. The operations are performed in a pipelined manner as well, to allow execution at 200 MHz.

The result of the rounding operation is a pair of new ring vectors that are then in turn applied to each remaining tower entry to reduce the noise accumulated by the initial product. These vectors are first processed with a series of RingAdds, RingSubs and a CRT using each of the corresponding ring moduli. The end result is that the highest tower ring is eliminated from the cipher text, and the overall noise of the system remains at a usable (i.e. de-cryptable) level.

III. Current Results and Next Steps

Our presentation will include I/O timing, run-time and chip utilization details of our attached processor performing the suite of ring primitives on various ring sizes, based on the implementation in our Virtex 7 VC707 evaluation board.

Future plans for our FPGA system include adding all Ring primitives that will be required to accelerate the Bootstrapping operation described in [4]. The CRT and inverse CRT operations will be modified to allow the Number Theoretic Transform (NTT) portion [7] to be combined into one circuit, saving a large amount of FPGA multiplier resources. Additionally, multiple ring sizes will be supported by modifications to the NTT to support multiple power-of-two ring sizes. This will allow us to support the Ring Reduction operation in [4] for increased computational efficiency. Note that all of the other primitives can at arbitrary ring sizes. The final target ring size is 2^{14}, which will support relatively secure computation.

Acknowledgment

We would like to acknowledge Christopher Peikert for all his numerous invaluable contributions to the theoretical and practical implementation aspects of this project.

References

[1] C. Gentry and S. Halevi. Implementing Gentry's Fully-Homomorphic encryption scheme. In Kenneth Paterson, editor, Advances in Cryptology – EUROCRYPT 2011, volume 6632 of Lecture Notes in Computer Science, chapter 9, pages 129–148. Springer, 2011.

[2] D. Micciancio. A first glimpse of cryptography's Holy Grail. Comm. ACM 53, 3 (March 2010), 96-96.

[3] V. Lyubashevsky, C. Peikert, and O. Regev. "On ideal lattices and learning with errors over rings". In Henri Gilbert, editor, Advances in Cryptology – EUROCRYPT 2010, volume 6110 of Lecture Notes in Computer Science, chapter 1, pages 1–23. Springer Berlin / Heidelberg, Berlin.

[4] K. Rohloff, D. B. Cousins, "A Scalable Implementation of Fully Homomorphic Encryption Built on NTRU." 2nd Work-shop on Applied Homomorphic Cryptography and Encrypted Computing (WAHC). Mar. 7, 2014.

[5] J. Alperin-Sheriff and C. Peikert. "Practical bootstrapping in quasilinear time". In Ran Canetti and JuanA. Garay, editors, Advances in Cryptology CRYPTO 2013, volume 8042 of Lecture Notes in Computer Science, pages 1–20. Springer Berlin Heidelberg, 2013.

[6] C. Gentry, S. Halevi, and N. Smart. "Homomorphic evaluation of the AES circuit." In Reihaneh Safavi-Naini and Ran Canetti, editors, Advances in Cryptology CRYPTO 2012, volume 7417 of Lecture Notes

in Computer Science, pages 850–867. Springer Berlin / Heidelberg, 2012.

[7] H. Cohen A Course in Computational Algebraic Number Theory. New York: Springer-Verlag, 1993.

[8] D. Cousins, K. Rohloff, C. Peikert, R. Schantz "Scalable Implementation of Primitives for Homomorphic EncRyption – FPGA implementation using Simulink" 2011 High Perfor-mance Extreme Computing Workshop Sep 21-22 2011, Lex-ington MA

[9] D. Cousins, K. Rohloff, C. Peikert, R. Schantz "An Update on SIPHER (Scalable Implementation of Primitives for Ho-momorphic EncRyption) – FPGA implementation using Simulink" 2012 IEEE Conference on High Performance Ex-treme Computing (HPEC) Sep 10-12 2012, Waltham MA

[10] M. Knezevic, F. Vercauteren, and I. Verbauwhede, "Faster Interleaved Modular Multiplication Based on Barrett and Montgomery Reduction Methods", IEEE Transactions on Computers, Vol. 59, No. 12, Dec 2010