SIPHER: Scalable Implementation of Primitives for Homomorphic EncRyption – FPGA implementation using Simulink

David Bruce Cousins, Kurt Rohloff, Chris Peikert, Rick Schantz
Raytheon BBN Technologies, Georgia Institute of Technology
{dcousins, krohloff, schantz}@bbn.com cpeikert@cc.gatech.edu

Abstract

Practical Fully Homomorphic Encryption (FHE) would be a game-changing technology to enable secure, general computation on encrypted data, e.g., on untrusted off-site hardware. Recent theoretical breakthroughs demonstrated the existence of Fully Homomorphic Encryption schemes [1,4]. However, FHE remains impractical because current implementations are many orders of magnitude too slow for practical use, and do not scale well to the very large keys and ciphertexts needed to assure a sufficient level of security. A new DARPA program (PROCEED) has as its focus the acceleration of various aspects of the FHE concept toward practical implementation and use.

In this paper we present early work on our SIPHER project, an element of the PROCEED program, whose goal is to demonstrate FHE implementations that improve the state of the art by many orders of magnitude. As part of our activity we are developing a set of hardware primitives to accelerate FHE implementations based on lattice problems [3]. As an important aspect of our design methodology we use a state of the art tool-chain offered by the Mathworks to develop FPGA circuits from Simulink Models. We initially develop prototype descriptions in Matlab that we re-implement in a stream oriented, hardware implementable manner in Simulink. The operations of the implementations are compared to verify correctness. A conversion from Simulink to VHDL is done in a completely automated fashion using Mathwork's HDL coder. This tool chain provides us the means to develop our primitives, including cyclic VHDL based FPGA prototyping, much faster than traditional methods.

Fully and Somewhat Homomorphic Encryption

Fully Homomorphic Encryption (FHE) holds the promise to securely run arbitrary computations over encrypted data on untrusted computation hosts [4]. The general FHE concept of operations is that sensitive data is encrypted with a public key, then sent to an untrusted computation host, which can perform arbitrary computations on the encrypted data without first needing to decrypt it. It has been shown to be theoretically possible to evaluate arbitrary programs using just two special purpose FHE operations, EvalAdd and EvalMult, which roughly correspond to bitwise XOR and AND gates operating on encrypted bits. A sequence of these operations is run against the encrypted data, resulting in an encryption of the output of the original program run on the unencrypted data. This encrypted result can then be sent back to the original client, who decrypts the result using its secret key. The encrypted data is protected at all times with reasonable security guarantees based on computational hardness results.

FHE enables more secure and private computation, but to be effective there needs to be multiple orders of magnitude efficiency improvements before it can be practical. Known FHE schemes are highly inefficient partly because they are "noisy" - the encryption schemes' ciphertext is a function not only of the plaintext and encryption keys but also of a noise term. The amount of noise in a ciphertext rapidly increases as the EvalAdd and EvalMult operations are performed, and after too many such operations there is too much noise to correctly decrypt the ciphertext. To run larger numbers of EvalAdd and EvalMult operations, FHE schemes typically address the accumulation of noise with a very computationally expensive "recryption" operation that is periodically run on intermediate ciphertexts to keep the noise at a level that still permits decryption.

A Somewhat Homomorphic Encryption (SHE) scheme supports several (but not unlimited) EvalMult and EvalAdd operations while preserving the correctness of decryption. In other words, SHE can schemes support secure computation for only a small subset of programs. Our development approach is to select an efficient implementation of an SHE scheme which can be converted into a full FHE scheme with the addition of a recryption (noise reduction) operation and/or other supporting modifications. This enables us to incrementally develop SHE results using modest initial resources.

Although there have been some initial FHE implementations [1], there have been no practical implementations that can be used for effective general computation. Current designs of FHE schemes rely on operations (i.e. modular arithmetic with an enormous modulus) that are inefficient on standard CPU architectures and which are too memory intensive. For convenience all of these previous implementations have been limited by their focus on CPUs and do not take advantage of specialized parallel computation hardware like FPGAs.

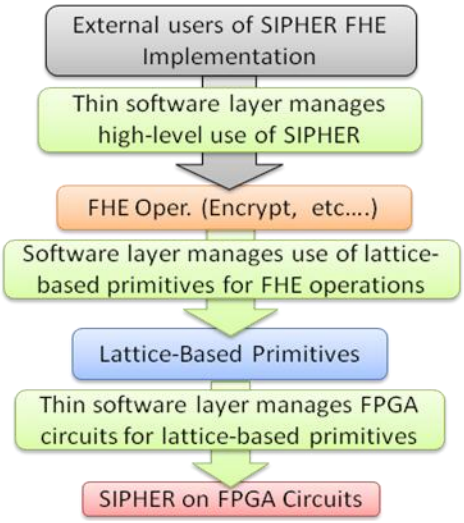

Figure 1: Conceptual diagram of system.

Figure 1 shows our vision for the layered services we provide in our FHE implementation. There are software interfaces for implementations of the basic FHE operations (KeyGen, Encrypt, EvalAdd, EvalMult, Recrypt, Decrypt) as a primitive basis for constructing more general applications on encrypted data. Our approach to the FHE primitives is based on the highly efficient lattice-based techniques developed by one of our investigators [3], which can be implemented with only a handful of core mathematical primitive operations (see Figure 2). Many of these operations are closely related to well-understood operations, such as Fast Fourier Transforms, which we are targeting for efficient implementations on FPGAs. The EvalAdd and EvalMult operations for example are simply element wise vector adds and multiplies taken modulo some particular prime integer q. These are trivial to express using Matlab: c = mod(a+b, q) and c = mod(a.*b, q).

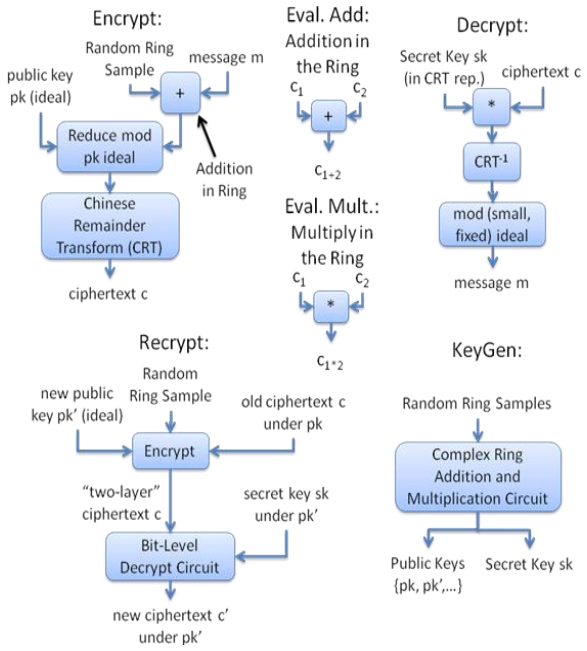

Figure 2: Primitives for a SHE scheme.

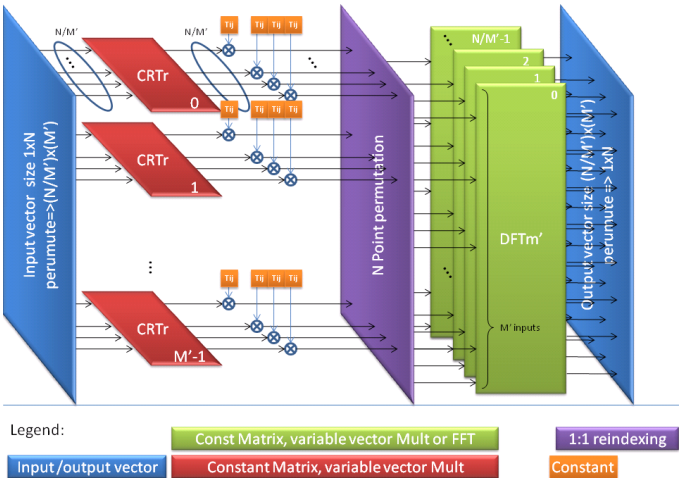

Figure 3: Internal Structure of CRT Primitive showing similarity to signal processing data flow.

We are leveraging previous work on signal processing implementations to implement the primitives (and consequently the FHE scheme) as circuits on FPGAs. The FPGAs provide highly cost-effective parallelism.

One of our primary primitive operations is the Chinese Remainder Transform (CRT). The CRT is mathematically similar to the Discrete Fourier Transform, but implemented using modular integer (instead of complex) arithmetic. Figure 3 shows the CRT implementation we are working with that is structurally very similar to the familiar processing of multi-dimensional signal data. The similarities of our primitives with well understood signal-processing operations that have been efficiently implemented in FPGAs give us confidence toward developing efficient and scalable FPGA implementations of the primitives.

The FFT operation in Figure 4 is similar to the standard FFT [2], except all operations are done in modulo q arithmetic. We were able to take Mathwork's example Simulink streaming FFT model (Figure 4), slightly modify the ordering of the output, and easily change from complex to integer arithmetic simply by altering the input and twiddle factor data types. Converting to modular arithmetic is also straightforward.

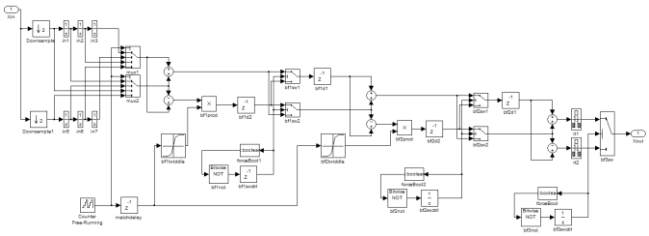

Figure 4: Simulink model for streaming FFT.

To implement the modular arithmetic efficiently in hardware we have taken advantage of the Montgomery Reduction method [5], which allows one to express mod q operations in a larger basis r, which can be a power of two. So while the bits required to represent the integers have grown, all arithmetic operations now are allowed to wrap around on overflow, eliminating the need to do a costly modular reduction operation in the hardware. We implement the Montgomery reduction method in Simulink using the fixed point tool box (Figure 5). The additional complexity this adds to the Simulink model for our ring operations is trivial. Figure 6 shows the Montgomery reduction steps in Red and Blue. Red steps convert to the Montgomery space, and are done once for each input. Any number of additions can be done without the need for a reduction step. Each multiply requires one reduction step. A final reduction step converts back into the original mod q representation. Our modified FFT implementation requires pre-computation of the twiddle factors in Montgomery representation (no real time impact), one reduction step for each input sample, one reduction for each output sample and one reduction at the output of each butterfly multiplication. Since all reduction is done using a pipelined approach, there is no additional computation time added (just latency).

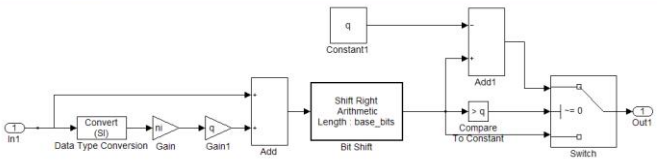

Figure 5: Simulink model for Montgomery Reduction

Interim Results

Our presentation will include examples of our primitives coded in Matlab and Simulink and examples of VHDL code generated by the HDL coder. We will also be able to show timing results from Modelsim based simulations of the resulting code.

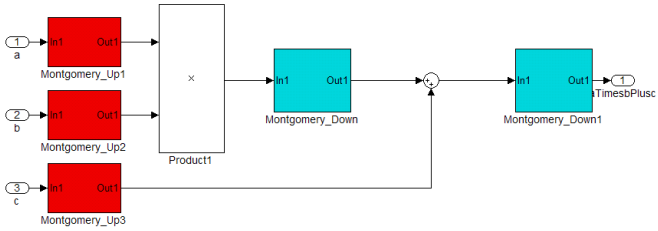

Figure 6: Simulink model for Ring Multiply-Add

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] W. M. Gentleman and G. Sande, "Fast Fourier transforms—for fun and profit," Proc. AFIPS 29, 563–578 (1966).

[3] Vadim Lyubashevsky, Chris Peikert, and Oded 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] D. Micciancio. A first glimpse of cryptography's Holy Grail. Comm. ACM 53, 3 (March 2010), 96-96.

[5] Peter L. Montgomery "Modular Multiplication Without Trial Division", *Mathematics of Computation* Vol. 44, No. 170 (Apr., 1985), pp. 519-521, American Mathematical Society.