Privacy-Preserving Functional IP Verification utilizing Fully Homomorphic Encryption

Charalambos Konstantinou*, **Anastasis Keliris***, **Michail Maniatakos**†
*Electrical and Computer Engineering, New York University Polytechnic School of Engineering
†Electrical and Computer Engineering, New York University Abu Dhabi
E-mail: {ckonstantinou, anastasis.keliris, michail.maniatakos}@nyu.edu

***Abstract*—Intellectual Property (IP) verification is a crucial component of System-on-Chip (SoC) design in the modern IC design business model. Given a globalized supply chain and an increasing demand for IP reuse, IP theft has become a major concern for the IC industry. In this paper, we address the trust issues that arise between IP owners and IP users during the functional verification of an IP core. Our proposed scheme ensures the privacy of IP owners and users, by a) generating a privacy-preserving version of the IP, which is functionally equivalent to the original design, and b) employing homomorphically encrypted input vectors. This allows the functional verification to be securely outsourced to a third-party, or to be executed by either parties, while revealing the least possible information regarding the test vectors and the IP core. Experiments on both combinational and sequential benchmark circuits demonstrate up to three orders of magnitude IP verification slowdown, due to the computationally intensive fully homomorphic operations, for different security parameter sizes.**

I. Introduction

System on Chip (SoC) solutions have progressively become a very attractive option for meeting the design requirements imposed by the customers, while maintaining a low time-to-market timeline. As the Integrated Circuit (IC) market is expanding, the number of Intellectual Property (IP) cores' vendors and users increases. Consequently, a plethora of IP cores is being developed by numerous vendors around the world, while users are attracted to the re-usability of optimized, verified, ready-to-deploy IP cores.

The globalized nature of the IP market enables an ecosystem consisting of both trusted and untrusted IP owners, as well as legitimate and malicious users. IP theft accounts for the 62% of the cybercrime damages, while many analysts concur that the extent of IP stolen in reality cannot be currently measured with confidence [1]. IP reverse engineering [2] has emerged as a major concern for IP owners, as adversaries possess increasingly sophisticated tools for revealing the IP design secrets.

A prominent phase in the design cycle, which requires close collaboration between the IP vendor and the IP user, is the verification of the IP core. Verification tools vary from simple implementations that target specific types of processors [3], to methodologies that can be applied to mixed software and hardware descriptions, e.g. SystemC [4]. While recent work has focused on advanced verification methods, such as model checking [5] and equivalence checking [6], verification using simulation methods still remains the preferred verification method [7].

In order to alleviate the trust issues between the involved parties, in this paper, we propose a novel privacy-preserving verification methodology. The proposed procedure follows the principles of Multi-Party Computation (MPC), and enables meaningful manipulation of operands through the use of Fully Homomorphic Encryption (FHE). By transforming the IP logic and encrypting the test vectors, the involved parties can functionally verify the IP core without revealing unwarranted information to each other.

The rest of the paper is organized as follows: Section II gives a brief background to FHE and MPC schemes, while Section III introduces the proposed privacy-preserving verification protocol. Finally, Section IV discusses the experimental results of the implementation, followed by concluding remarks in Section V.

II. FHE Preliminaries

The introduction of the Fully Homomorphic Encryption (FHE) scheme by Gentry [8] allowed the encapsulation of homomorphic encryption in many applications [9]. The advances in computing capabilities, in addition with newer FHE schemes [10], [11], have decreased the –originally– prohibitive computational cost of FHE.

FHE enables operations on the ciphertext without the knowledge of the encryption key, as shown in Eq. 1

and Eq. 2, for homomorphic addition and homomorphic multiplication respectively:

$$E(y_1) + E(y_2) = E(y_1 + y_2) \tag{1}$$

$$E(y_1) \star E(y_2) = E(y_1 \otimes y_2) \tag{2}$$

The deployed FHE includes the basic four FHE algorithms: `Keygen`, `Encrypt`, `Decrypt` and `Eval`. The `Eval` algorithm is build based on other three algorithms: `Add`, `Mult` and `Recrypt`. `Eval` takes as input a circuit C, a public key pk, encryptions of $\epsilon_1, ..., \epsilon_n$ and evaluates $C(\epsilon_1, ..., \epsilon_n)$ using only ϵ_i's ciphertexts. This concept of Multikey-FHE [12] can evaluate any encrypted circuit C using different keys, and consequently can be used to construct MPC protocols. The `Recrypt` operation "cleans" the ciphertext from the noise due to the homomorphic addition and multiplication operations. Without this function the scheme would be Somewhat Homomorphic and therefore it would only evaluate circuits of a fixed depth.

The theoretical background of the FHE scheme and specifically the key generator is based on the Smart-Vercauteren method [11]. The parameter λ defines the cipher size and is the upper bound μ for picking a random polynomial in the key generator. The public key contains a decryption hint that is split into S_2 addends and randomly distributed over an array of size S_1. N is the degree of the monic irreducible polynomial $F(x)$ used for the key generation shown in Eq. 3 [11], [13].

$$F(x) = x^N + 1 \tag{3}$$

For the aforementioned parameters, experimental results in Section IV presents the trade-off between the security level of the implementation and the overall verification time.

Using FHE, we aim to solve the trust issues imposed by the collaboration of the IP designer with the IP user. The implemented MPC protocol essentially allows all parties to securely and jointly perform any computation over their inputs with zero knowledge [14]. The idea of secure MPC protocols of computing a joint function on parties private inputs originate from the works of Yao [15] and Goldreich [16].

III. Privacy-Preserving IP Verification

In this section, we describe the proposed privacy-preserving IP verification methodology. Our objective is to construct a tool that can handle any IP core and various security objectives.

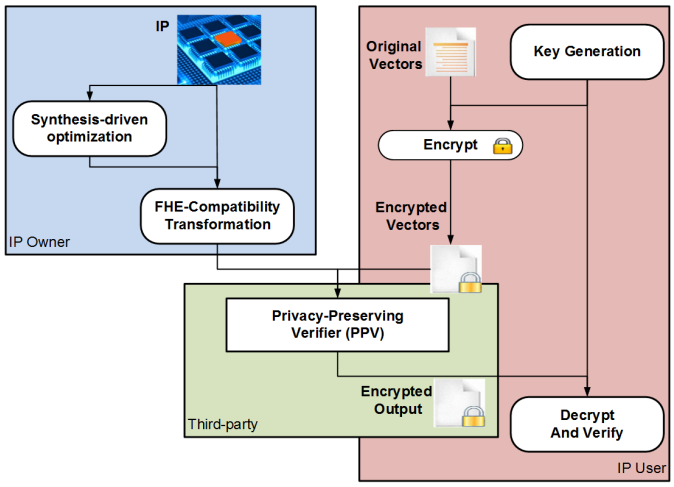

Fig. 1. Flowchart of the privacy-preserving verification protocol

In the target usage model, privacy-preserving IP verification can include either two parties, the IP owner and the IP user, or three parties, in case a third-party service provider is required for verification outsourcing. The proposed methodology is flexible enough to support all different usage models in the context of MPC. In all scenarios, the IP owner seeks to obscure implementation details of the IP core, while the IP user aims to protect –potentially proprietary– test vectors.

Fig. 1 presents the steps required to perform design verification using FHE. Specifically:

1) The IP owner transforms the IP to a 'FHE-compatible' netlist, as discussed in Section III-A. The transformation can potentially include a re-synthesis step to increase performance, also discussed in Section III-A.
2) The IP user generates the encryption key, and encrypts the input vectors using FHE. This process is discussed in Section III-B.
3) The outputs of the previous steps (IP Core + Encrypted Test Vectors) are securely transmitted to a third-party. The third-party uses the developed Privacy-Preserving Verification (PPV) tool, presented in Section III-C, and generates the encrypted test outputs.
4) The encrypted test outputs are transmitted back to the IP user, and they are decrypted using the IP user's key. The IP user compares the decrypted test output with the expected test output and verifies the functionality of the IP core.

It should be emphasized that the PPV tool can be used by either the IP owner or the IP user, instead of a third-party.

TABLE I
CIRCUIT TRANSFORMATION OPERATIONS

Operation	Transformation
$A \oplus B$	$A \oplus B$
$A \wedge B$	$A \wedge B$
$\neg A$	$A \oplus 1$
$A \vee B$	$((A \oplus 1) \wedge (B \oplus 1)) \oplus 1$
$A \barwedge B$	$(A \wedge B) \oplus 1$
$A \barvee B$	$((A \oplus 1) \wedge (B \oplus 1) \oplus 1) \oplus 1$
$\neg(A \oplus B)$	$(A \oplus B) \oplus 1$
$\neg(A \wedge (B \vee C))$	$(((A \wedge B) \oplus 1) \wedge (C \oplus 1)) \oplus 1$
$\neg((A \vee B) \wedge C)$	$((((A \oplus 1) \wedge (B \oplus 1)) \oplus 1) \wedge C) \oplus 1$
MUX2	$(((A \wedge S) \oplus 1) \wedge (((S \oplus 1) \wedge B) \oplus 1)) \oplus 1$
FA	$(A \oplus B) \oplus C$
FA(C_{out})	$((((A \oplus B) \wedge C) \oplus 1) \wedge ((A \wedge B) \oplus 1)) \oplus 1$
HA	$A \oplus B$
HA(C_{out})	$A \wedge B$

A. IP core transformation

As we extensively discussed in Section II, the FHE domain supports homomorphic addition and homomorphic multiplication, which correspond to XOR and AND functions respectively. Therefore, in order to use the IP core in the proposed PPV, we need to transform all combinational logic to corresponding XOR and AND gates, creating an 'FHE-compatible' netlist. Sequential logic (Flip-Flops, Latches etc.) can intuitively remain as-is.

Table I shows the transformation of cells of a typical library to a combination of XOR and AND gates. We assume that the library contains only 2-input gates; gates with more inputs can be transformed to a series of 2-input gates. It should be noted that several transformations include fixed values as one of the inputs; therefore, the developed PPV tool maintains a pool of encrypted fixed values to be used during verification.

With regards to special inputs, such as the clock and reset signals, the IP user can choose to use their unencrypted versions, in order to avoid the extra overhead of encrypting a large set of '0' and '1'. This cannot be applied, however, to designs that incorporate clock gating mechanisms, as unencrypted signals cannot be mixed with encrypted signals.

1) Synthesis optimization: While in a typical IP verification environment the AND and XOR operations have comparable performance overheads, the FHE domain introduces substantial time differences. Specifically, the AND operation adds more performance overhead compared to the XOR operation, for the following two reasons:

- Homomorphic multiplication is by default slower than homomorphic addition, similar to the performance difference between integer multiplication and integer addition.
- Homomorphic multiplication introduces 'more noise per operation', compared to homomorphic addition. In layman's terms, a series of multiplications can render a ciphertext indecipherable much faster than a series of homomorphic additions. Noise is reduced using the `recrypt` function, as explained in Section II. Thus, homomorphic multiplications require more computationally intensive `recrypt` operations.

The `recrypt` algorithm performs a homomorphic decryption on an encryption of the ciphertext bits. As a result, the overhead of producing a 'cleaner' ciphertext is considerably high compared with the `encrypt` and `decrypt` functions. An AND operation, i.e. a multiplication, inherits the sum of the two operands' noises and therefore must be used along the recrypt function when the 'dirtiness' of the circuit becomes greater than the depth of the circuit. In order to identify the cumulative noise of the ciphertexts, we measure the noise of every encrypted bit. XORs, theoretically, do not require the `recrypt` operation [8], [11], [13], as they add considerably less noise to the output operand compared to the AND function. In more detail, the ratio in terms of noise added to output operand after an AND and a XOR operation was experimentally found to be 20:1.

In order to minimize the performance overhead of an IP core, we drive the synthesis tool to remove as many multiplication operations as possible. To this end, a three-step process is required:

1) Remove all but XOR and AND gates from the target library,
2) Modify the AND gate area size to be substantially higher then the XOR,
3) Synthesize the circuit and optimize for area.

Experimental results, presented in Section IV-A, corroborate that synthesis-driven optimization can improve the performance of the PPV by up to 41.2%.

B. Test vector encryption

On the IP user's side, the first step is to generate the FHE key to be used for encrypting/decrypting the test

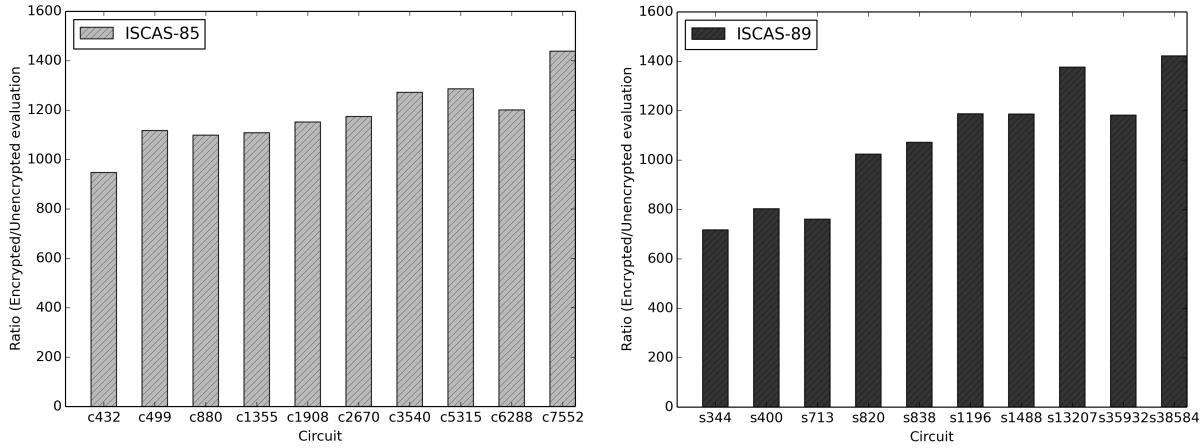

Fig. 2. Verification slowdown for ISCAS-85 and ISCAS-89 benchmarks

inputs patterns. The size of the key and the selected security parameters offer a trade-off between security and performance, as analyzed in the experimental results. The default security parameters our tool uses are $\lambda = 384$, $S_1 = 8$, $S_2 = 5$, $N = 8$ and $\mu = 4$. Every bit of every test vector is encrypted using the generated key. Every encryption includes a random factor as well, in order to generate different encrypted representations for all '0' and '1'. As previously discussed, the IP user can opt-out of clock and reset signals encryption. The encrypted input patterns are then securely transmitted to the third-party, which will run the PPV tool.

C. Privacy-preserving verifier (PPV)

The heart of the PPV tool is a functional simulator, modified to perform homomorphic operations instead of typical boolean logic manipulation. It receives two inputs: 1) The 'FHE-compatible' IP netlist, and 2) the encrypted input vectors. It then proceeds to functionally simulate the design. In case the clock and reset signals are unencrypted, the tool should initiate reset and clocking mechanisms. A major difference between 'FHE-compatible' and typical functional simulation is that all gates are evaluated, even if the output value does not change, due to probabilistic encryption: For example, in typical simulation, when an AND gate has an output of 0, and a logic 0 arrives at the input, the output value does not need to be evaluated and no value change propagation is necessary. In the encrypted domain, however, the tool does not differentiate between 0 and 1, so the output needs to be evaluated and propagated every time.

IV. Results

A. Experimental Setup

In this section, we evaluate our proposed methodology using the ISCAS85 and ISCAS89 benchmarks. The benchmarks have been synthesized using Synopsys Design Compiler with a 45-nm standard cell library. Experimental results are obtained on a 64-bit machine with 3GHz Intel Core i7-3540M single-core processor, with 4 GB RAM; the machine runs the Linux 3.13.0-32 kernel with gcc 4.8.2 compiler.

The developed tool relies on the open-source Verilog software [17] for the functional simulation of the circuit benchmarks. The homomorphic operations are implemented using the `libScarab` cryptographic library [13]. As previously discussed in Section II, the main functions we instantiate from `libScarab` are: `fhe_keygen`, `fhe_encrypt`, `fhe_decrypt`, `fhe_recrypt`, `fhe_add`, `fhe_mul`.

B. IP Verification Performance

This section discusses the performance implications of the proposed privacy-preserving IP verification.

Performance overhead of encrypted test vectors: The first set of results demonstrate the performance overhead of the developed methodology. Fig. 2 shows the IP verification slowdown for a series of ISCAS benchmarks, using Table I to convert the various gates to FHE operations and random input patterns. The overhead ranges from 700x, for the smaller circuits, up to 1400x for the larger designs, demonstrating a correlation between the size of the circuit and the overhead of the privacy-preserving verification methodology. It should be noted that the

results are obtained using a single core. Current research on parallelizing cryptographic operations [18] has the potential to greatly decrease the incurred overhead.

Effect of security parameter trade-offs: The second set of results reveal the effect of the desired security constraints to the proposed methodology. Table II summarizes the key generation, execution time and recrypt figures for different security parameters to the `libScarab` library.

To demonstrate the implementation details of our code integration into the libScarab system, we performed experiments based on security parameters as in [19]. Our goal is to present the correlation between different security and timing factors. Table II summarizes, for a single circuit, the effect on the overall execution time, as well as the required depth d for performing the recrypt function, of various security parameters. The results are normalized to the overhead of the default configuration our tool uses.

As both S_1 and S_2 increase, the time needed for the recrypt and therefore the execution algorithm increases. This arises from the fact that the re-encryption algorithm takes as public key the result of the extended keygen algorithm $PK = (p, \alpha, S_1, S_2, \{c_i, B_i\}_{i=1}^{S_i})$. In addition, Table II shows that the most time-consuming function is the key-generation algorithm, which is directly dependent on the security parameter λ of the implementation. Large values of λ increase the key generation time as well as the security level of the design. As key generation happens only once, however, this overhead becomes less important as the execution time dominates.

Furthermore, the degree of $F(x)$ used for the key generation, N, also determines the required depth for the recrypt algorithm. The latter has a greater effect on N's value compared to the parameter μ. It should be also noted that the case of $\mu = 2$ is chosen in order to obtain as large a depth for the Somewhat HE as possible [11]. Summarizing, as the values of λ, S_1 and S_2 increase, the time needed for calculating the key and using re-encryption increases. With regards to the parameters N and μ, although they affect the overall timings, their major contribution is related to the cumulative noise of ciphertexts and therefore how often the recrypt operation is triggered.

TABLE II
SECURITY PARAMETERS OVERHEADS NORMALIZED TO 384/8/5 KEY GEOMETRY

Key geometry (λ/S1/S2)	**N**	μ	**d**	**Keygen**	**Exectime**	**Recrypt**
384/8/5	8	4	7	1	1	1
384/16/5	8	4	7	0.1	2.0	2.0
384/16/5	8	6	7	1.8	1.7	1.7
384/64/16	8	4	7	0.3	10.0	10.0
512/64/16	8	4	7	5.7	11.8	11.8
1024/64/16	8	4	8	70.7	15.3	15.4
2048/16/5	4	2	10	124.2	1.9	1.9
4096/16/5	4	2	11	830.6	2.8	2.8

C. Synthesis-driven optimization

In this section, we transform the circuits' original netlist to a 'FHE-friendly' netlist, as discussed in Section III. We introduce three different cases for synthesis-driven optimization, and the results, compared to the baseline of simple netlist transformation, are presented in Table III.

XA$^+$IO: In this case, we remove all but the 2-input XOR, AND, INV and OR gates from the library. These four gates are the minimum set of gates required by Synopsys Design Compiler (otherwise the synthesis tool refuses to run). As discussed in Section III-A1, the area of the AND gate is set to be considerably larger than the rest (indicated by the $^+$ symbol). Consequently, we requested Design Compiler to optimize for area (using the `-area_effort high` flag). After gate removal and circuit synthesis, Table III demonstrates that there is no observable benefit in using this approach, as results are worse. Further investigation indicates that the overall execution time in the XA$^+$IO case is dominated by the OR gates of the synthesized circuit. As presented in Section III-A, the OR gate is being substituted by several operations in the FHE domain.

XA$^+$IO$^+$: In order to limit the effect of the OR gate to the performance of the privacy-preserving IP verification, we modified the size of the OR gate to be substantially bigger compared to the default one (arbitrarily chosen to be 10,000 times bigger). Again, we optimized for area, in an effort to exclude as many OR gates as possible. The results are now mixed, as some circuits exhibit improved timing while others are still worse.

XA$^+$IO$^+$N: Finally, we also included NANDs in the library. The reason behind this decision is that NAND is a universal gate, and can potentially substitute a wide combination of AND and OR gates more efficiently. The results demonstrate a substantial improvement of the results: 41.2% average performance overhead reduction. Performance improvement appears on all benchmarks.

TABLE III
SYNTHESIS OPTIMIZATION RESULTS

Circuit	XA^+IO (%)	XA^+IO^+ (%)	XA^+IO^+N (%)
c432	77.66	7.85	-53.99
c499	63.89	-37.18	-34.92
c880	82.52	-4.20	-46.27
c1355	48.22	70.88	-43.77
c1908	36.41	-21.71	-50.79
c2670	73.41	3.07	-39.30
c3540	73.84	-5.04	-37.22
c5315	71.17	-13.98	-31.08
c6288	70.37	0.73	-30.41
c7552	37.63	-25.03	-41.12
s344	113.7	-4.00	-42.47
s400	76.58	-42.61	-51.65
s713	149.23	-21.48	-31.95
s820	78.16	-10.72	-49.13
s838	74.12	-24.45	-50.74
s1196	71.81	-0.21	-41.44
s1488	80.11	-3.19	-38.01
s13207	62.32	-19.20	-37.72
s35932	74.28	-23.91	-37.60
s38584	64.13	-14.93	-34.32
Average	**73.98**	**-9.47**	**-41.2**

V. CONCLUSIONS - FUTURE DIRECTIONS

In this paper, we presented a novel methodology for trusted IP verification. The proposed protocol enables the IP owner to transform the design to a functionally-equivalent format that can be deployed in the homomorphic encryption domain, while the IP user can securely apply proprietary input patterns. Privacy-preserving verification can also be outsourced to a third party for increased security. The expected slowdown of the FHE application is around three orders of magnitude. To reduce the overhead, we also introduced synthesis-driven optimization, which transforms the circuit to a 'FHE-friendly' format.

With regards to future directions, we plan to add several enhancements to the developed tool, such as control of unknowns (Xs) and other logic values. Furthermore, tristate buffers are currently unsupported, and future research will target their inclusion to the tool. Finally, we plan to employ the proposed methodology to large, open-source IP cores.

REFERENCES

[1] R. Anderson *et al.*, "Measuring the Cost of Cybercrime," in *The Economics of Information Security and Privacy*, 2013, pp. 265–300.

[2] E. Castillo *et al.*, "IPP@HDL: Efficient Intellectual Property Protection Scheme for IP Cores," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 15, no. 5, pp. 578–591, 2007.

[3] M. Stadler *et al.*, "Functional verification of intellectual properties (IP): a simulation-based solution for an application-specific instruction-set processor," in *International Test Conference*, 1999, pp. 414–420.

[4] A. Fin *et al.*, "The use of SystemC for design verification and integration test of IP-cores," in *14th Annual IEEE International ASIC/SOC Conference*, 2001, pp. 76–80.

[5] M. Pradella *et al.*, "Bounded Satisfiability Checking of Metric Temporal Logic Specifications," *ACM Transactions on Software Engineering and Methodology*, vol. 22, no. 3, pp. 20:1–20:54, 2013.

[6] B. Keng and A. Veneris, "Path-Directed Abstraction and Refinement for SAT-Based Design Debugging," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 10, pp. 1609–1622, 2013.

[7] G. Moretti *et al.*, "Your Core– My Problem?: Integration and Verification of IP," in *38th annual Design Automation Conference*, 2001, pp. 170–171.

[8] C. Gentry, "A Fully Homomorphic Encryption Scheme," Ph.D. dissertation, 2009.

[9] N. Tsoutsos and M. Maniatakos, "HEROIC: Homomorphically EncRypted One Instruction Computer," in *Design, Automation and Test in Europe Conference and Exhibition*, 2014, pp. 1–6.

[10] Z. Brakerski *et al.*, "(Leveled) Fully Homomorphic Encryption Without Bootstrapping," in *3rd Innovations in Theoretical Computer Science Conference*, 2012, pp. 309–325.

[11] N. P. Smart and F. Vercauteren, "Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes," in *13th International Conference on Practice and Theory in Public Key Cryptography*, 2010, pp. 420–443.

[12] A. López-Alt *et al.*, "On-the-fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption," in *44th Annual ACM Symposium on Theory of Computing*, 2012, pp. 1219–1234.

[13] H. Perl *et al.*, "POSTER: An Implementation of the Fully Homomorphic. Smart-Vercauteren Crypto-System," in *ACM Conference on Computer and Communications Security*, 2011, pp. 837–840.

[14] Y. Ishai *et al.*, "Zero-knowledge from Secure Multiparty Computation," in *39th Annual ACM Symposium on Theory of Computing*, 2007, pp. 21–30.

[15] A. C.-C. Yao, "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science*, 1986, pp. 162–167.

[16] O. Goldreich *et al.*, "How to Play ANY Mental Game," in *19th Annual ACM Symposium on Theory of Computing*, 1987, pp. 218–229.

[17] E. Mednick and M. Hummel. Veriwell Verilog Simulator. [Online]. Available: http://veriwell.sourceforge.net/,(Accessed: 9/18/14).

[18] K. Jarvinen and J. Skytta, "On Parallelization of High-Speed Processors for Elliptic Curve Cryptography," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 16, no. 9, pp. 1162–1175, 2008.

[19] M. Brenner *et al.*, "How Practical is Homomorphically Encrypted Program Execution? An Implementation and Performance Evaluation," in *IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, 2012, pp. 375–382.