A multi-start heuristic for multiplicative depth minimization of boolean circuits

Sergiu Carpov, Pascal Aubry, Renaud Sirdey

May 29, 2017

Abstract

In this work we propose a multi-start heuristic which aims at minimizing the multiplicative depth of boolean circuits. The multiplicative depth objective is encountered in the field of homomorphic encryption where ciphertext size depends on the number of consecutive multiplications. The heuristic is based on rewrite operators for multiplicative depth-2 paths. Even if the proposed rewrite operators are simple and easy to understand the experimental results show that they are rather powerful. The multiplicative depth of the benchmarked circuits was hugely improved. In average the obtained multiplicative depths were lower by more than 3 times than the initial ones. The proposed rewrite operators are not limited to boolean circuits and can also be used for arithmetic circuits.

1 Introduction and related works

An *encryption scheme* describes the way of encrypting and decrypting plaintext messages such that finding which is the plaintext message from encrypted data (denoted *ciphertext* in what follows) is either very hard or even impossible without a secret. An encryption scheme is said to be *homomorphic* when some operations on plaintext messages can be done homomorphically, that is directly in the space of ciphertexts (and without decrypting them). When addition and multiplication operations are supported, the homomorphic encryption (HE) scheme is functionally complete. Since the seminal work of Gentry [8], introducing the first practical (to some extent) homomorphic encryption many other simpler and more efficient schemes have been proposed [5, 6]. A HE scheme with a binary plaintext space allows to execute any boolean circuit directly over encrypted data.

A noise component is added to the ciphertext during the encryption for security reasons. The noise component is a common characteristic for HE schemes. Each new homomorphic operation applied on the ciphertexts increases the noise component in the resulting ciphertext. After a (predefined) number of homomorphic operations the noise is so large that the correctness of the decryption

cannot be not ensured anymore. Usually the noise growth induced by the addition operation is smaller than the noise growth induced by the multiplication operation. That is why many authors consider only the *multiplicative depth*[1] of evaluated circuits when HE schemes are parametrized. In order to support the evaluation of larger multiplicative depth circuits, for an equivalent security level, the ciphertext sizes must be increased and respectively the cost of homomorphic operations increases also. Another solution to this problem is to use ciphertext bootstrapping [9]. The bootstrapping procedure takes a noisy ciphertext as input and executes homomorphically the HE scheme decryption. The noise of the resulting "bootstrapped" ciphertext is lower than the noise of the input ciphertext.

Obtaining low multiplicative depth circuits is a major issue in the practical use of homomorphic encryption. With every new multiplicative level the HE scheme parameters increase in size. Therefore the execution time of the whole boolean circuit increases accordingly. Many works found in the literature treat the problem of boolean circuit optimization for hardware targets or more generally the problem of hardware synthesis. We refer to the open-source software system used for hardware synthesis ABC [3]. It is an open-source environment providing implementations of the state-of-the-art circuit optimization algorithms. The most common objectives used in hardware synthesis are circuit area and circuit depth (latency). To the best of our knowledge none of these algorithms were designed for multiplicative depth minimization.

Cryptographic literature mostly focused on the minimization of multiplicative complexity of circuits, i.e. the number of AND gates in circuits where the XOR gates are for free [4, 10, 13]. The authors of Armadillo compilation chain [7] studied the use of ABC tools for minimizing the multiplicative depth of boolean circuits in the context of a compilation chain targeting homomorphic execution.

Several works [12, 11, 2] study the minimization of bootstrappings in boolean circuits problem. Bootstrapping is a computational heavy procedure. It is straightforward to see that minimizing the number of bootstraps in a homomorphic evaluation of a circuit increases its execution performance. Although the problem we study in this paper shares the same goal (i.e. increase the homomorphic execution performance of boolean circuits) the employed methods to achieve it are orthogonal.

In this work we introduce and study multiplicative depth-2 path rewrite operators which decrease the multiplicative depth of a boolean circuit. We furthermore propose a heuristic method which makes use of these operators. The goal of the heuristic is to minimize the boolean circuit multiplicative depth. The paper is structured as follows, in section §2 are described the proposed circuit rewrite operators and the heuristic itself, section §3 presents experimental results we have performed and finally section §4 concludes the paper and discuss some perspectives.

[1] Multiplicative depth is the number of sequential homomorphic multiplications which can be done on freshly encrypted ciphertexts in order to be able to decrypt and retrieve the result of multiplications.

2 Multiplicative depth minimization multi-start heuristic

2.1 Preliminary definitions

A boolean circuit is a directed acyclic graph $C = (V, E)$ with a set of nodes V and a set of edges E. Circuit nodes represent boolean functions (gates) and circuit edges are connections between nodes. The set of nodes can be split into 2 independent sub-sets:

- Nodes without a predecessor define circuit inputs. An input node can be either a boolean input variable or a boolean constant (e.g. logic "0" or logic "1" inputs).
- Nodes each representing a gate applying a basic boolean function to the values of its predecessors. The input degree of gates is 2. A sub-set of gate nodes represent circuit outputs. Without loss of generality we suppose that the set of output nodes is the same as the set of nodes with zero output degree. In this work we suppose that the boolean circuits use AND and XOR operators only. The set $\{\text{AND}, \text{XOR}\}$ together with the constant "1" is functionally complete [14]. This means that any boolean function can be expressed using these operators.

We denote by $\text{pred}\colon V \to 2^V$ and $\text{succ}\colon V \to 2^V$ the functions giving the set of predecessors, respectively successors, of a node $v \in V$ in a boolean circuit C.

The number of successively executed AND operators, also called *multiplicative depth*, influences the parameters of a HE scheme. The minimization of the multiplicative depth allows not only to obtain smaller ciphertext sizes but also to minimize[2] the overall execution time of the boolean circuit. Let us define a function $d\colon V \to \{0, 1\}$ which returns one for AND nodes and zero otherwise. Only the nodes for which $d(v) = 1$ influence circuit multiplicative depth.

Let $l\colon V \to \mathbb{N}$ be a function which gives the *multiplicative depth* of circuit nodes. The multiplicative depth of node v is equal to the maximal number of AND gates on any path beginning in an input node and ending in node v. The multiplicative depths for circuit nodes are computed recursively using relation:

$$l(v) = \begin{cases} 0 & \text{if } |\text{pred}(v)| = 0, \\ \max_{u \in \text{pred}(v)} l(u) + d(v) & \text{otherwise.} \end{cases}$$

Let $r\colon V \to \mathbb{N}$ be a function which gives the *reverse multiplicative depth* of circuit nodes. The reverse multiplicative depth of node v is the maximal number of AND gates on any path beginning in a successor of node v and ending in an output node. It is somewhat equivalent to the multiplicative depth function except that it does not include the depth due to the node itself. The reverse multiplicative depths for circuit nodes are computed recursively using:

[2] As we shall further see, more precisely it depends on the relative computational cost of circuit AND gates with respect to scheme multiplicative depth.

$$r(v) = \begin{cases} 0 & \text{if } |\text{succ}(v)| = 0, \\ \max_{u \in \text{succ}(v)} (r(u) + d(u)) & \text{otherwise.} \end{cases}$$

The overall multiplicative depth of a circuit C is the maximal multiplicative depth of its nodes:

$$l^{\max} = \max_{v \in V} l(v) = \max_{v \in V} r(v).$$

The *critical nodes* of a circuit C are the nodes for which relation (1) is verified. We denote *critical circuit* the sub-circuit containing all the critical nodes of a circuit C. A *critical path* is a path in this circuit.

$$l(v) + r(v) = l^{\max},\ v \in V \tag{1}$$

2.2 Multiplicative depth-2 path rewrite operators

The multiplicative depth of a boolean circuit equals to the multiplicative depth of its critical part. Decreasing the multiplicative depth of the critical part will necessarily decrease the overall circuit multiplicative depth. In this section we introduce two rewrite operators which when applied to the critical part of a boolean circuit potentially minimize the multiplicative depth of a circuit. The idea behind these operators is to rewrite critical paths of multiplicative depth 2 in such a way that the overall multiplicative depth decreases. We firstly describe an operator which rewrites simple paths composed of two AND gates only. Afterwards, a second rewrite operator is described which allows to obtain such a simple path (from two AND gates) from any path of multiplicative depth 2. Additionally, we introduce the conditions these paths should verify so that the multiplicative depth is lowered after the rewrite operators are applied.

Let P denote the set of all critical paths beginning and ending in an AND gate and containing exactly 2 AND nodes, i.e. the set of paths of multiplicative depth 2. A path $p \in P$ contains at least 2 nodes: 2 AND gates separated by zero or more XOR gates. Figure 1 illustrates such a critical path.

Let us examine a critical path p of length 2, i.e. $p = (v_1, v_{|p|})$ where v_1 and $v_{|p|}$ are AND gates. Such a path is shown on the left-hand side of figure 2. Path p can be rewritten using AND associativity rule: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$. The right-hand side of figure 2 illustrates the circuit part obtained after this rewrite operator is applied to path p. Rewritten path multiplicative depth decreases only if the multiplicative depth of nodes y and z are less than the multiplicative depth of node x, i.e. $l(y) < l(x)$ and $l(z) < l(x)$. In this case the multiplicative depth of gate $v_{|p|}$ decreases by one, from $l(x) + 2$ to $l(x) + 1$. The number of AND gates in the resulting circuit either increases by one or rests the same if node v_1 does not have other successors than node $v_{|p|}$.

In case of critical paths of length larger than 2, the inner XOR gates prevents the direct use of the rewrite operator defined above. A second rewrite operator allows to move an AND gate up the critical path by one place. We call it *AND gate move up* operator. This operator uses XOR distributivity rule: $(x \oplus y) \cdot z =$

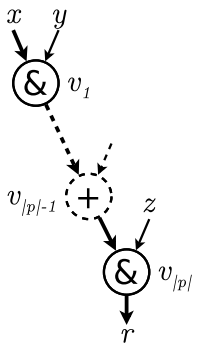

Figure 1: Critical path of multiplicative depth 2. Thick edges represent the critical path.

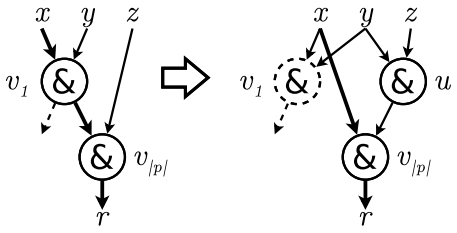

Figure 2: Length 2 critical path $\left(v_1, v_{|p|}\right)$ rewrite operator. Dotted line AND gate v_1 is kept only if needed.

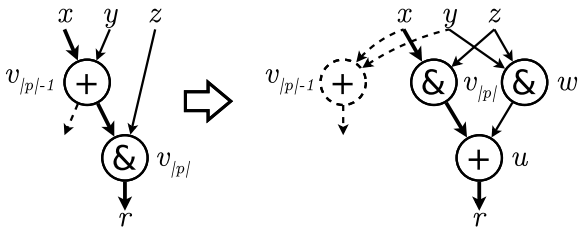

Figure 3: AND gate move up operator. Dotted line XOR gate $v_{|p|-1}$ is kept only if needed.

$(x \cdot z) \oplus (y \cdot z)$. An illustration of initial and resulting paths after the application of this operator is shown in figure 3. In the resulting circuit the number of AND gates increases by one and potentially the number of XOR gates increases by one also.

Suppose that we need to move up an AND gate over a path containing k XOR gates. Let $(((x \oplus y_1) \oplus \ldots) \oplus y_k) \cdot z$ be the formula of this circuit. The direct application of the AND gate move up operator adds an AND gate for each XOR gate on the path. Observing that the initial formula can be rewritten as $(x \oplus (y_1 \oplus \ldots \oplus y_k)) \cdot z$ (XOR associativity) we can transform it into $(x \cdot z) \oplus (y_1 \oplus \ldots \oplus y_k) \cdot z$. This new formulation is functionally equivalent to the one obtained using direct application of AND gate move up operator except that the number of additional AND gates is only one.

Let $p = \left(v_1, v_2, \ldots, v_{|p|}\right)$ be the critical path illustrated in figure 1, we recall that v_1 and $v_{|p|}$ are AND gates. The AND gate move up operator is used to move node $v_{|p|}$ next to node v_1. Afterwards, a critical path of length 2 is obtained, which is rewritten using the first operator. Condition (2) insures that the multiplicative depth of the rewritten node $v_{|p|}$ decreases. It is equivalent to the condition defined earlier for length 2 paths.

$$\min_{u \in \mathrm{pred}(v)} l(u) < l(v_1) - 1,\ v \in \left\{v_1, v_{|p|}\right\} \tag{2}$$

We shall note that the overall boolean circuit multiplicative depth does not necessarily decrease after the above defined rewrite operators are applied, as the critical circuit can contain several parallel critical paths. All these critical paths have to be rewritten in order to decrease the overall circuit multiplicative depth by one.

2.3 Multi-start heuristic

In this section we introduce a multi-start heuristic which uses rewrite operators defined above in order to minimize the multiplicative depth of a boolean circuit. Algorithm 1 is a priority based heuristic which rewrites critical paths of multiplicative depth 2. The path to rewrite is chosen using a priority function

Algorithm 1 Multiplicative depth minimization heuristic.

Input: C – input boolean circuit
Input: $prior\_func$ – priority function
Output: C_{out} – multiplicative depth optimized boolean circtuit

1: $C_{out} \leftarrow C$
2: **while** termination conditions are not verified **do**
3: $P \leftarrow$ critical paths of multiplicative depth 2 from circuit C
4: $P \leftarrow$ filter paths $p \in P$ respecting condition (2)
5: **if** $|P| = 0$ **then**
6: **break**
7: **end if**
8: $p \leftarrow prior\_func(P)$ ▷ get highest priority path
9: $p \leftarrow$ rewrite multiplicative depth-2 path p
10: **if** $l_{\max}(C_{out}) > l_{\max}(C)$ **then**
11: $C_{out} \leftarrow C$
12: **end if**
13: **end while**

(introduced later). The algorithm stops either when a termination condition (e.g. time, number of iterations) is verified or when there are no more reducible critical paths, i.e. paths which respect condition (2). If the set of critical paths P is not empty, the algorithm chooses a path from it according to a priority function $prior\_func$ and rewrites this path using operators presented in previous section. If the multiplicative depth of the obtained circuit lowers then this new circuit is memorized as output circuit (variable C_{out}).

In order to decrease the overall multiplicative depth of a boolean circuit by one, all the parallel critical paths of this circuit must be rewritten. As we have observed empirically, the decrease of circuit multiplicative depth makes the new critical circuit wider and wider, that is to say the number of parallel critical paths increases. Respectively, the number of newly added gates (due to rewrite operators) increase in a non-linear way in the worst-case scenarios.

From the perspective of boolean circuit homomorphic execution, the minimization of multiplicative depth is beneficial (in terms of execution time) if the number of additional AND gates does not exceed a threshold. This threshold is defined by the ratio between the AND gate execution time at the previous multiplicative level and the AND gate execution time at the current multiplicative level. In order to obtain the best boolean circuit for homomorphic execution one can either stop when the number of newly added AND gates exceeds this threshold or store all the obtained circuits C_{out} (algorithm line 11) and choose afterwards the circuit for which the homomorphic execution time is minimal.

We introduce several functions which prioritize the path selection (ties are broken randomly):

- multiplicative depth of first path node: increasing order (d) and decreasing order (D),

- total number of critical predecessors of all path nodes: increasing order (i) and decreasing order (I),
- total number of critical successors of all path nodes: increasing order (o) and decreasing order (O),
- total number of critical predecessors and successors of all path nodes: increasing order (p) and decreasing order (P),
- critical path length: increasing order (l) and decreasing order (L).

Additionally to *non-random priority* functions[3] we have implemented a *random priority* function. Using different random seeds we obtain various search space explorations.

The *multi-start heuristic* consists in executing algorithm 1 several times with different priority functions. In our experimentations we test two versions of the multi-start heuristic. In the first version an input circuit is optimized one time for each non-random priority function (10 executions) and in the second one the input circuit is optimized 10 times using random priority with different seeds. In both cases the best obtained solution (minimal multiplicative depth and minimal number of AND gates in case of equal multiplicative depths) is kept as multi-start heuristic result. In the next section we present the results of the experimentations we have performed for both multi-start algorithm versions.

3 Experimental results

Boolean circuits from the EPFL Combinational Benchmark Suite were used for experimentations. This set of benchmarks contains exclusively combinational circuits. Three types of circuits are provided: arithmetic, random/control and very large (multi-million gate designs). Please refer to [1] for more details about these benchmarks. In our experiments we have used only the first two types of benchmarks[4]: 10 arithmetic and 10 random/control circuits. Before using the benchmarks we have optimized and mapped them with ABC commands `resyn2` and `map`. The last command was used to obtain boolean circuits with AND and XOR gates only. Table 1 shows the characteristics of the obtained benchmarks after these commands were performed.

The heuristic described in the previous section was implemented in C language. The binary uses ABC as helper library. The two versions of the multi-start heuristic were executed on each benchmark circuit. Algorithm 1 execution terminates early if either the number of iterations is greater than 2 times the number of AND gates in the input circuit or the execution time exceeds 1 hour. A middle-end server with AMD Opteron 6172 processors (2.1GHz) was used as execution platform.

[3] By abuse of language we denote so the above defined priority functions

[4] We assume that multi-million gate designs are out of reach for homomorphic execution, at least for the current state of HE schemes.

Circuit name	#input	#output	×depth	#AND
adder	256	129	255	509
bar	135	128	12	3141
div	128	128	4253	25219
hyp	256	128	24770	120203
log2	32	32	341	20299
max	512	130	204	2832
multiplier	128	128	254	14389
sin	24	25	161	3699
sqrt	128	64	4968	15571
square	64	128	247	9147
arbiter	256	129	87	11839
ctrl	7	26	8	108
cavlc	10	11	16	658
dec	8	256	3	304
i2c	147	142	15	1161
int2float	11	7	15	213
mem_ctrl	1204	1231	110	44795
priority	128	8	203	676
router	60	30	21	167
voter	1001	1	36	4229

Table 1: EPFL Combinational Benchmark Suite characteristics after initial optimization with ABC.

Obtained results are shown in table 2. The solutions for first version (column "non-random") and second version (column "random") of multi-start heuristic are illustrated in this table. The initial characteristics of circuits are also recalled (column "initial"). The notations we use are the multiplicative depth ("×depth"), the number of AND gates ("#AND"), the ratio between the multiplicative depth of the input circuit and the optimized one ("ratio") and the non-random priority for which the best solution was obtained ("priority").

The best solution (in terms of multiplicative depth and number of AND gates) was obtained using a non-random priority in 9 cases and using a random priority in 11 cases. For the ctrl and i2c benchmarks both heuristic versions obtained the same result. Multiplicative depth of the obtained circuits is significantly smaller when compared to the multiplicative depth of input circuits. In average the multiplicative depth decreases by more than 3 times. As expected, the price to pay for a smaller multiplicative depth is an increase in the number of AND gates (approximatively 1.2 times more in average).

The most substantial decrease is obtained for the adder benchmark, which is the usual 128-bit ripple carry adder. The proposed heuristic achieves an impressive result being able to transform a ripple carry adder with multiplicative depth 255 into "some sort of" carry-lookahead adder with a multiplicative depth

Circuit	initial		non-random				random		
	×depth	#AND	×depth	#AND	ratio	priority	×depth	#AND	ratio
adder	255	509	12	911	21.2	P	**11**	1125	23.2
bar	12	3141	12	3141	1.0	-	12	3141	1.0
div	4253	25219	1852	29329	2.3	l	**1463**	31645	2.9
hyp	24770	120203	24563	120293	1.0	P	**24562**	120307	1.0
log2	341	20299	**141**	27362	2.4	p	150	22266	2.3
max	204	2832	27	4751	7.6	P	**27**	**4660**	7.6
multiplier	254	14389	60	21884	4.2	p	**59**	17942	4.3
sin	161	3699	**76**	5922	2.1	p	81	4473	2.0
sqrt	4968	15571	**4225**	18435	1.2	i	4391	16785	1.1
square	247	9147	**28**	10478	8.8	d,i	29	9731	8.5
arbiter	87	11839	42	8652	2.1	P	**42**	**8582**	2.1
ctrl	8	108	**5**	**109**	1.6	L	**5**	**109**	1.6
cavlc	16	658	**9**	669	1.8	D,I,o	10	658	1.6
dec	3	304	3	304	1.0	-	3	304	1.0
i2c	15	1161	**8**	**1185**	1.9	D,o	**8**	**1185**	1.9
int2float	15	213	**8**	216	1.9	D,o	9	214	1.7
mem_ctrl	110	44795	45	54889	2.4	p	**45**	**49175**	2.4
priority	203	676	102	1121	2.0	l	**102**	**1106**	2.0
router	21	167	11	261	1.9	o	**11**	**204**	1.9
voter	36	4229	**30**	**4288**	1.2	P	30	4340	1.2

Table 2: Best obtained solutions for heuristic aggregated by priority function (non-random and random). Bold font is used to emphasize the best solution. The best solution considers the multiplicative depth as well as the number of AND gates.

11 only.

The multiplicative depth of 2 benchmark circuits was not improved by the heuristic. In both cases the heuristic was not able to find any reducible multiplicative depth-2 paths. The dec benchmark (a 8 to 256 decoder) was already at its lowest possible multiplicative depth. As for the bar circuit (barrel shifter) we suppose that the proposed rewrite operators are too weak in terms of expressive power and more complex rewrite operators (e.g. circuit cone rewrite operators) are needed for dealing with such type of circuits.

There is not a single priority function which performs well (i.e. for which the best solution is found) on all benchmarks. The best solutions for 13 benchmarks are found using two priority functions: the total number of critical predecessors and successors of all path nodes (p,P), total number of critical successors of all path nodes in increasing order (o). We assume that each priority function performs well for a specific topology of boolean circuits.

The heuristic finished early because of time limit in the case of 5 benchmarks: div, hyp, sqrt, arbiter and mem_ctrl. The obtained multiplicative depths for these

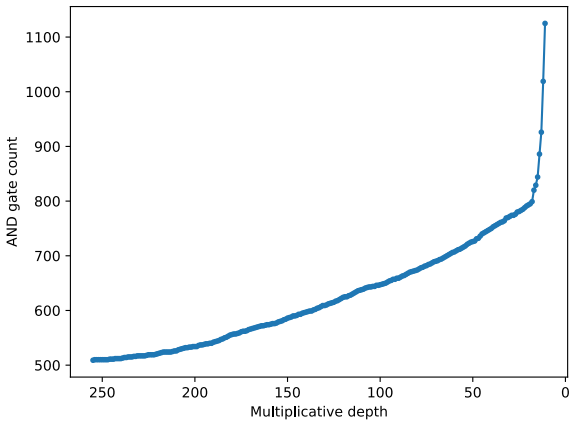

Figure 4: Number of AND gates as a function of the multiplicative depth for the benchmark adder. Final multiplicative depth is 11.

benchmarks are not the lowest possible ones. Allocating more execution time to heuristic will potentially increase the quality of presented results. We have rerun the tests for the 5 benchmarks with time limit increased to 2 hours. The multiplicative depth further lowered for div (from 1463 to 675), hyp (from 24562 to 24417), sqrt (from 4225 to 3709), arbiter (from 42 to 11) and mem_ctrl (from 45 to 43) benchmarks. The exploration did not finish for 3 benchmarks: div, hyp and sqrt.

In order to see how the multiplicative depth influences the number of AND gates we have saved all the intermediary circuits obtained during heuristic execution. The heuristic was executed on the adder circuit. The random priority function (for which the smallest depth circuit was obtained in previous experiments) was used. The dependence between the number of AND gates and the multiplicative depth of intermediary circuits is illustrated in figure 4. We can see that the number of AND gates increases faster when the multiplicative depth is smaller. Moreover this increase is exponential for the last circuits (smallest multiplicative depth ones).

4 Conclusions and perspectives

In this work we have proposed and studied a multi-start heuristic for minimizing the multiplicative depth of boolean circuits. The heuristic uses rewrite operators for boolean circuit critical paths. As a function of the used priority functions several versions of the multi-start heuristic have been studied. We have tested heuristic's performance on a set of circuits found in the literature. In average the multiplicative depth of benchmarked circuits was lowered by more than 3 times

by the proposed heuristic. In perspective we envisage to study more elaborate heuristics together with new priority functions.

The optimization method described in this paper can also be applied to arithmetic circuit. An arithmetic circuit is a generalization of boolean circuits where instead of binary field operations higher degree field/ring operations are used. An arithmetic circuit is functionally complete when defined over addition and multiplication operations. It is easy to see that the optimization algorithm proposed in this paper together with rewrite operators can also be directly applied to arithmetic circuits and how to do so.

References

[1] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. The EPFL Combinational Benchmark Suite. In *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, 2015.

[2] Fabrice Benhamouda, Tancrède Lepoint, Claire Mathieu, and Hang Zhou. Optimization of Bootstrapping in Circuits. In *SODA*, pages 2423–2433. SIAM, 2017.

[3] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification, Release 30308. http://www.eecs.berkeley.edu/~alanmi/abc/.

[4] Joan Boyar and René Peralta. Concrete Multiplicative Complexity of Symmetric Functions. In *MFCS*, volume 4162 of *Lecture Notes in Computer Science*, pages 179–189, 2006.

[5] Zvika Brakerski. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *Advances in Cryptology - Crypto 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, 2012.

[6] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, 2012.

[7] Sergiu Carpov, Paul Dubrulle, and Renaud Sirdey. Armadillo: A Compilation Chain for Privacy Preserving Applications. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing*, SCC '15, pages 13–19, 2015.

[8] Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178, 2009.

[9] Craig Gentry, Shai Halevi, and Nigel P. Smart. Better Bootstrapping in Fully Homomorphic Encryption. In Marc Fischlin, Johannes A. Buchmann, and Mark Manulis, editors, *Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 1–16, 2012.

[10] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. volume 5888 of *Lecture Notes in Computer Science*, pages 1–20, 2009.

[11] Tancrède Lepoint and Pascal Paillier. On the Minimal Number of Bootstrappings in Homomorphic Circuits. In *Financial Cryptography Workshops*, volume 7862 of *Lecture Notes in Computer Science*, pages 189–200, 2013.

[12] Marie Paindavoine and Bastien Vialla. Minimizing the Number of Bootstrappings in Fully Homomorphic Encryption. In *SAC*, volume 9566 of *Lecture Notes in Computer Science*, pages 25–43, 2015.

[13] Thomas Schneider and Michael Zohner. GMW vs. Yao? Efficient Secure Two-Party Computation with Low Depth Circuits. In *Financial Cryptography*, volume 7859 of *Lecture Notes in Computer Science*, pages 275–292, 2013.

[14] William Wernick. Complete sets of logical functions. *Transactions of the American Mathematical Society*, 51(1):117–132, 1942.