Multiplicative Complexity of Autosymmetric Functions: Theory and Applications to Security

Anna Bernasconi
Dipartimento di Informatica
Università di Pisa, Italy
anna.bernasconi@unipi.it

Stelvio Cimato Valentina Ciriani Maria Chiara Molteni
Dipartimento di Informatica "Giovanni Degli Antoni"
Università degli Studi di Milano, Italy
{stelvio.cimato, valentina.ciriani, maria.molteni}@unimi.it

***Abstract*—The multiplicative complexity of a Boolean function is the minimum number of AND gates (i.e., multiplications) that are sufficient to represent the function over the basis {AND, XOR, NOT}. The multiplicative complexity measure plays a crucial role in cryptography-related applications. In fact, the minimization of the number of AND gates is important for high-level cryptography protocols such as secure multiparty computation, where processing AND gates is more expensive than processing XOR gates. Moreover, it is an indicator of the degree of vulnerability of the circuit, as a small number of AND gates corresponds to a high vulnerability to algebraic attacks. In this paper we study a particular structure regularity of Boolean functions, called autosymmetry, and exploit it to decrease the number of ANDs in XOR-AND Graphs (XAGs), i.e., Boolean networks composed by ANDs, XORs, and inverters. The interest in autosymmetric functions is motivated by the fact that a considerable amount of standard Boolean functions of practical interest presents this regularity; indeed, about 24% of the functions in the classical ESPRESSO benchmark suite have at least one autosymmetric output. The experimental results validate the proposed approach.**

I. Introduction

In standard CMOS technology XOR gates are expensive and often considered impractical [28]; they are used only when their presence in a network implementation of a Boolean function guarantees a considerable reduction of some design parameters, usually the area of the network (see [4], [13], [14], [15], [22]). Recently, however, the growing relevance of cryptography-related applications and emerging technologies has revived the interest in XOR gates [12], [18], [19], [25], [26]. For instance, in high-level cryptography protocols such as secure multiparty computation, processing XOR gates is particularly convenient since their evaluation is possible without any interaction between the parties, and then has no communication cost [21] (more details are given in Section II-A). In this context, it is therefore important to consider network representations that assume XOR gates explicitly, and in fact, cryptographic applications often consider Boolean functions represented over the basis {AND, XOR, NOT}. As a result, the widely adopted And-Inverter Graph (AIG) logic networks, implemented in the academic state-of-the-art logic synthesis tool ABC [10], have recently evolved into the new *XOR-AND Graph* (XAG) multi-level logic representations [18], [19], [25]. Recall that an AIG is a directed acyclic graphs of 2-input AND nodes, with possibly inverted edges; analogously, a XAG is an AIG enriched with 2-input XOR nodes.

Logic synthesis on XAGs mainly aims at reducing the number of AND nodes [25]. For example, as already mentioned, for secure multiparty computation AND nodes are the only nodes in a XAG with a communication cost (observe that the NOT operation can be implemented with a XOR). The number of AND nodes in a XAG implementation of a function is called the *multiplicative complexity* of the XAG, while the minimum number of ANDs that are sufficient to represent a function with a XAG defines the *multiplicative complexity of the function*. This complexity measure plays a crucial role in cryptography-related applications: not only it is important for protocols such as secure multiparty computation where processing AND nodes is more expensive than processing XOR nodes, but it also represents an indicator of the degree of vulnerability of the circuit, as a small number of AND nodes corresponds to a high vulnerability to algebraic attacks.

In this paper, we study the multiplicative complexity of a class of Boolean functions exhibiting a special structural regularity, called *autosymmetry*, easily expressed using the XOR operation [7], [8], [22]. Intuitively, a Boolean function f over n variables is *k-autosymmetric* if it can be projected onto a smaller function f_k that depends on $n-k$ variables only, and has a smaller on-set. The XOR operation comes into play as the new $n-k$ variables are XOR combinations of some of the original ones. The new function f_k is called *restriction* of f and can be identified in polynomial time. Observe that, even if autosymmetric functions depend in general on all their n input variables, they can be studied in a $n-k$ dimensional space; i.e., they are in general non-degenerated, whereas all degenerated functions are autosymmetric. The interest in autosymmetric functions in this context is motivated by the fact that a XAG representation of an autosymmetric function f can be easily obtained composing a XAG for the restriction f_k with an additional layer of XOR nodes (as discussed in Sections II-C and III). As a result, the multiplicative complexity of f can be estimated from the multiplicative complexity of the restriction f_k. Actually, we prove a stronger result: f and f_k have exactly the same multiplicative complexity. Since f_k is a smaller function, depending on less variables, computing a XAG representation and minimizing the number of AND nodes become easier problems, whose solutions allow to better assess

the actual multiplicative complexity of the original function (note that determining the exact value of the multiplicative complexity of a function is a computationally intractable problem [11]). We validate the proposed approach through a set of experiments. First of all, we observe that autosymmetry is a property that is frequent enough within Boolean functions to be worth studying, indeed about 24% of the functions in the classical ESPRESSO benchmark suite [29] have at least one non-degenerate autosymmetric output [8]. For this set of functions, we are able to get a better estimate of the multiplicative complexity in about 74% of the cases, with an average reduction of the number of ANDs of about 61%.

The paper is organized as follows. Preliminaries on multiparty computation, multiplicative complexity, XAG networks, and autosymmetric functions are described in Section II. Section III discusses the relationships between the multiplicative complexity of an autosymmetric function and the multiplicative complexity of its restriction, and Section IV reports the experimental results. Finally, Section V concludes the work.

II. PRELIMINARIES

A. Secure Multiparty Computation

The goal of Secure Multiparty Computation (SMC) protocols is to allow a set of mutually distrusting parties to compute a joint function, keeping their own inputs private. Secure computation, also denoted as secure function evaluation, has been studied since the 80's, and a theoretical result has been proved stating that any function can be securely computed [31].

One of the possible approaches to solve the SMC problem has been provided by Yao and is based on the Garbled Circuit (GC) protocol [30], that allows the secure evaluation of the Boolean circuit representing the given function. The idea of the protocol is that the two collaborating parties, say Alice and Bob, will evaluate the circuits by encoding wire values as random strings and by encrypting the truth table of each gate. Using the input wire values corresponding to the inputs held by Alice and Bob, it is possible to decrypt the encoding of the gate output. Alice acts as the garbler and sends Bob an encoding of the circuit including the encrypted truth table for each gate and the encodings of her inputs (that remain hidden to Bob). Bob asks Alice the encodings for his inputs, using an oblivious transfer protocol, a cryptographic primitive that allows Bob to obtain the requested values without revealing anything about his input to Alice. The protocol then goes on with Bob computing the correct output of the circuit by evaluating the encoded inputs gate by gate, and sharing only the final result with Alice.

Recently, several research results have been presented in literature, showing impressive improvement in the performance of GC protocols [20], [23], and proposing many software libraries and prototypes such as JustGarble [3], SCAPI [16], and TinyGarble [24]. Using such libraries, many efficient applications have been presented solving also complex problems [2]. One of the proposed approaches to increase the efficiency of the GC protocol is to transform the given circuit increasing the number of XOR gates for reducing the number of other gates, since a variation of the protocol has shown that the evaluation of XOR gates is possible without any interaction between the parties, and then it has no communication cost [21]. In this direction, logic synthesis algorithms and tools have been proposed aiming at minimizing the multiplicative complexity of the circuit, in logic networks composed of AND, XOR and inverter gates. In [25], a technique based on a cut rewriting algorithm is described, achieving interesting performance in the optimization of benchmarks related to secure multiparty computation applications.

B. Multiplicative Complexity and XOR-AND graphs

The *multiplicative complexity* of a Boolean function is a complexity measure defined as the minimum number of AND gates (i.e., multiplications) that are sufficient to represent the function over the basis $\{$AND, XOR, NOT$\}$, a logic basis widely used to represent Boolean functions for cryptographic applications [9], [11], [25], [27]. More precisely:

Definition 1: The *multiplicative complexity* $M(f)$ *of a Boolean function* f is the number of AND gates that are necessary and sufficient to implement f with a circuit over the basis $\{$AND, XOR, NOT$\}$.

Definition 2: The *multiplicative complexity* $M_C(f)$ *of a circuit* C implementing a Boolean function f over the basis $\{$AND, XOR, NOT$\}$ is the actual number of AND gates in C.

Observe that the multiplicative complexity of a circuit for f only provides an upper bound for the multiplicative complexity of f, i.e., $M(f) \leq M_C(f)$.

As already discussed in Section I, the multiplicative complexity measure plays a crucial role in cryptography-related applications for various reasons. First of all, the minimization of the number of AND gates is important for high-level cryptography protocols such as zero-knowledge protocols and secure multiparty computation, where processing AND gates is more expensive than processing XOR gates [1], as discussed in the previous section. Moreover, the multiplicative complexity is an indicator of the degree of vulnerability of the circuits, as a small number of AND gates in an $\{$AND, XOR, NOT$\}$ circuit corresponds to a high vulnerability to algebraic attacks [11], [17], [27]. Unfortunately, determining the exact value of the multiplicative complexity of a function f is a computationally intractable problem [11]. Thus, the minimization of the number of AND gates in any circuit implementation over the basis $\{$AND, XOR, NOT$\}$ becomes very important to assess the actual multiplicative complexity of the function.

In this work, we consider Boolean functions represented in *XOR-AND graphs* (XAGs) form [18], [25], and use the multiplicative complexity $M_X(f)$ of a XAG implementation of a function f to provide an upper bound for its real multiplicative complexity $M(f)$. XAGs are logic networks which contain only binary XOR nodes, binary AND nodes, and inverters. In particular, we refer to the XAG model described in [25], where regular and complemented edges are used to connect the gates. Complemented edges indicate the inversion of the signals and replace inverters in the network. An example of

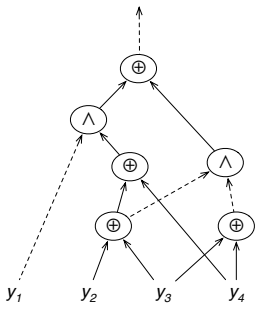


Fig. 1. XAG representation of the 4-input function corresponding to the Karnaugh map in Figure 3.

XAG is shown in Figure 1, where complemented edges are denoted by dashed lines.

C. Autosymmetry

Commonly, the "regularities" of Boolean functions are exploited with the purpose to derive, in shorter synthesis time, more compact circuits. It is not always clear whether a function is "regular", and which type of regularity could be exploited for its synthesis. Some previous papers on logic synthesis have focused on structural regularities of Boolean functions based on the notion of affine spaces and easily expressed using XORs. In this context, we study a particular regularity, i.e., autosymmetry [7], [8], [22], in order to decrease the multiplicative complexity of a XAG.

Intuitively, a Boolean function f over n variables is k-*autosymmetric* if it can be projected onto a smaller function f_k that depends on $n-k$ variables. The regularity of a Boolean function f is then measured computing its *autosymmetry degree* k, with $0 \leq k \leq n$, where $k = 0$ means no regularity. For $k \geq 1$ the Boolean function f is said to be *autosymmetric*, and a new function f_k depending on $n-k$ variables only, called the *restriction* of f, is identified in polynomial time. Moreover, an expression for f can be simply built from f_k: $f(x_1, x_2, \ldots, x_n) = f_k(y_1, y_2, \ldots, y_{n-k})$, where f_k is a Boolean function on $n-k$ variables $y_1 = \oplus(X_1), y_2 = \oplus(X_2), \ldots, y_{n-k} \oplus (X_{n-k})$ and each $\oplus(X_i)$ is a XOR whose input is a set of variables X_i with $X_i \subseteq \{x_1, x_2, \ldots, x_n\}$. Note that $\oplus(X_i)$ can be a single variable, i.e., $X_i = \{x_j\}$ and $\oplus(X_i) = x_j$. The autosymmetry test consists of finding the value of k, the restriction f_k, and each single XOR with its input variables X_i (reduction equations). Note that a degenerate function, i.e., a function that does not depend on all the variables in the Boolean space, is autosymmetric.

The restriction f_k is "equivalent" to, but smaller than f, and has $|S(f)|/2^k$ minterms only, where $S(f)$ denotes the support of f, and thus $|S(f)|$ is the number of minterms of f. The synthesis of f can be reduced to the synthesis of its restriction f_k, which can be identified in polynomial time. As the new $n-k$ variables are XOR combinations of some of the original ones, the reconstruction of f from f_k can be obtained with an additional logic level of XOR gates, whose inputs are the original variables, and the outputs are the new $n-k$ variables and their complementations given as inputs to a circuit for f_k. The restricted function f_k can be synthesized in any framework of logic minimization, in this paper we derive a XAG representation of it. The overall representation of a XAG for the function f using the XAG for f_k and the XOR nodes for the reduction equations is represented in Figure 4.

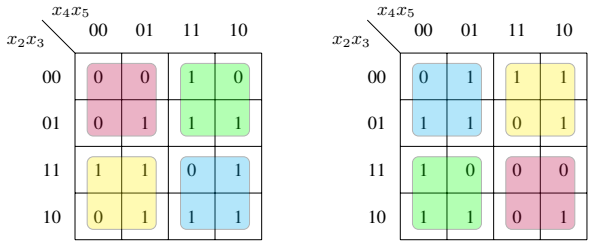
Consider, for example, the second output of the benchmark function *rd53* of the LGSynth'89 benchmark suite [29], i.e., the Boolean function f depicted in Figure 2. The "regularity" of the function is highlighted by the colors in the figure. The autosymmetry degree of f is 1 (i.e., $k = 1$) and the reduction equations are $y_1 = x_1 \oplus x_2$, $y_2 = x_1 \oplus x_3$, $y_3 = x_1 \oplus x_4$, $y_4 = x_1 \oplus x_5$ (for details on the computation see [8]). Thus, the restriction f_1 depends on 4 variables and it is depicted in Figure 3. Note that each point of the restriction corresponds to two points of the original function, as indicated by the colors in the maps. For example, the point 0000 in the Karnaugh map of Figure 3 corresponds to the two points 00000 and 11111 in the Karnaugh map of Figure 2. This is due to the reduction equations. In fact, considering the point $(x_1, x_2, x_3, x_4, x_5) = 00000$, through the reduction equations we get $(y_1, y_2, y_3, y_4) = (x_1 \oplus x_2, x_1 \oplus x_3, x_1 \oplus x_4, x_1 \oplus x_5) = 0000$. Exactly the same holds for the point $(x_1, x_2, x_3, x_4, x_5) = 11111$. It is easy to verify that we can perform a similar computation for any couple of corresponding points depicted in Figure 2, obtaining the Karnaugh map of Figure 3. For this example we have the XAG representation described in Figure 5.

Autosymmetric functions are just a subset of the total number of Boolean functions. Indeed, while the number N_B of the Boolean functions of n variables is $N_B = 2^{2^n}$, the number of autosymmetric ones is $N_A = (2^n - 1)2^{2^{n-1}}$ [8]. Therefore, the set of autosymmetric functions is much smaller than the one containing all the Boolean functions. Nevertheless, a considerable amount of standard Boolean functions of practical interest falls in the class of autosymmetric functions. Indeed, about 24% of the functions in the classical ESPRESSO benchmark suite [29] have at least one truly (i.e., non degenerate) autosymmetric output [7], [8]. Thus, the interest on autosymmetric functions is motivated by 1) their compact (in term of number of AND gates) representation, which consists on a XOR layer that is the input to a XAG for the restriction; 2) the frequency of autosymmetric functions in the set of benchmark functions.

III. Multiplicative Complexity of Autosymmetric Functions

In this section we investigate the relationships between the multiplicative complexity of an autosymmetric function and the multiplicative complexity of its restriction.

First of all, observe that a XAG representation of a k-autosymmetric function f can be easily obtained composing a XAG for the restriction f_k with an additional layer of XOR gates implementing the reduction equations. The inputs to the new layer are the original variables $x_1, x_2, \ldots, x_n$ and the outputs are the new variables $y_1, y_2, \ldots, y_{n-k}$, that become the



(a) $x_1 = 0$ (b) $x_1 = 1$

Fig. 2. Karnaugh map of the second output of *rd53* depending on the 5 Boolean variables x_1, x_2, x_3, x_4, x_5.

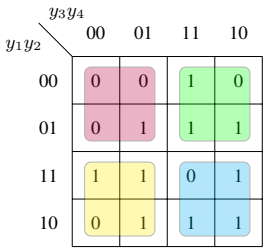


Fig. 3. Karnaugh map of the restriction of the second output of *rd53* depending on the 4 Boolean variables y_1, y_2, y_3, y_4.

inputs to the XAG for f_k, as shown in Figure 4. Since the new layer contains only XOR gates, we immediately conclude that $M(f) \leq M(f_k)$, as formally stated in the following lemma.

Lemma 1: The multiplicative complexity of an autosymmetric function f is less or equal to the multiplicative complexity of its restriction f_k.

Proof. First recall that the multiplicative complexity of a XAG implementation for a function f provides an upper bound for the multiplicative complexity of the function itself. Thus, the thesis follows since we can construct a XAG for f with exactly $M(f_k)$ AND nodes. This can be done adding to a XAG for f_k, containing a minimum number $M(f_k)$ of AND nodes, a layer consisting only of XOR nodes, as shown in Figure 4. ■

Actually, a much stronger result holds: f and f_k have exactly the same multiplicative complexity. To prove this result, we need to recall some properties of autosymmetric functions and of their restrictions. As shown in [7], [8], any k-autosymmetric function f is associated to a k-dimensional vector space L_f, defined as the set of all minterms w s.t. $f(v) = f(v \oplus w)$ for all $v \in \{0,1\}^n$. The k variables that are truly independent onto L_f, i.e., the variables that assume all the possible combinations of $\{0,1\}$ values in the minterms in L_f, are called *canonical variables* and are used to construct the restriction f_k. In fact, f_k corresponds to the projection of f onto the subspace $\{0,1\}^{n-k}$ where all the canonical variables assume value 0 (see [7], [8] for more details).

Consider for instance the 1-autosymmetric benchmark *rd53* (second output) discussed in Section II-C. Its associated vector space is the 1-dimensional space $L_f = \{00000, 11111\}$, whose canonical variable is x_1 (all other variables must be equal to x_1 on L_f), and the restriction corresponds to the projection of the function onto the space where $x_1 = 0$, as it can be noted from Figures 2 and 3.

Exploiting this characterization for the restriction of an autosymmetric function, we can prove the following theorem.

Theorem 1: Let f be a k-autosymmetric function, and let f_k be its restriction. Then,

$$M(f) = M(f_k)\,.$$

Proof. We have proved in Lemma 1 that $M(f) \leq M(f_k)$. Thus, it is enough to show that $M(f) \geq M(f_k)$. By contradiction suppose that the multiplicative complexity of the whole function f is strictly less than the multiplicative complexity of its restriction f_k, i.e., $M(f) < M(f_k)$. This assumption means that any XAG for f_k requires strictly more than $M(f)$ AND nodes, i.e., $M_X(f_k) > M(f)$, where $M_X(f_k)$ denotes the multiplicative complexity of a XAG for f_k. Since the restriction f_k corresponds to the projection of f onto the subspace $\{0,1\}^{n-k}$ where all the canonical variables of f have value 0, we can derive a XAG representation for f_k starting from a XAG for f and substituting all canonical input variables with the constant value 0. Note that the constant value 0 can be obtained computing the XOR of any non-canonical variable with itself. Such a transformation can only decrease the number of ANDs in the original XAG, as all AND nodes that receive in input the constant value 0 can be removed from the circuit, and substituted with the value 0. Therefore, if we start from a XAG implementation of f with the minimum number $M_X(f) = M(f)$ of AND nodes, we can derive a XAG for f_k with $M_X(f_k) \leq M(f)$ ANDs, in contradiction with the initial assumption $M(f) < M(f_k)$. ■

Since the restriction f_k is a smaller function, depending on less variables, computing a XAG representation and minimizing the number of ANDs become easier problems, whose solutions allow to better assess the actual multiplicative complexity of the original function f. For instance, for our running example concerning the benchmark *rd53* (second output), we can derive the XAG representation shown in Figure 5 simply adding four XOR nodes to the XAG for f_k of Figure 1, that contains 4 XORs and only 2 ANDs. Notice that the direct XAG

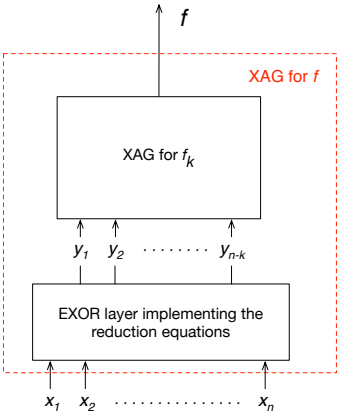


Fig. 4. A XAG for an autosymmetric function f obtained adding a XOR level implementing the reduction equations to a XAG for the restriction f_k.

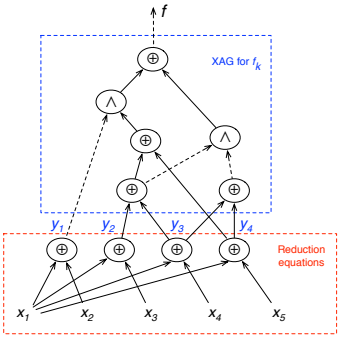


Fig. 5. XAG representation for the benchmark *rd53* (second output), derived exploiting the autosymmetry of the function.

minimization of *rd53* performed using the software from [25] would produce a bigger circuit, with 12 ANDs and 23 XORs.

IV. Experimental results

The approach presented above has been applied to the ESPRESSO and LGSynth'89 benchmark suite [29], running on a Pentium INTEL(R) CORE(TM) i5-5200U 2.20 GHz processor with 4.00 GB RAM, on a virtual machine running OS Ubuntu 64-bit. The experiments consider the subset of single outputs that are autosymmetric. The main aim of the experiments is to compare the synthesized XAG computed starting from an autosymmetric function f and the synthesized XAG computed starting from the corresponding restriction f_k, after the autosymmetry test. Recall that the autosymmetry test computes the autosymmetry degree k of a Boolean function and outputs: 1) the reduction equations, which form the XOR layer, and 2) the corresponding restriction f_k. We performed the autosymmetry test described in [7], [8] considering the onset of the benchmarks. The functions f and f_k are synthesized in XAG form using the heuristic approach proposed in [25] and briefly described at the end of Section II-A. We then compare the number of AND nodes of the XAGs for f and f_k in order to understand how the autosymmetry test can enable the XAG minimization of autosymmetric functions. For the sake of briefness, we report in Table I only a significant subset of the results. The first column reports the name of the function considered (benchmark function and output number). The following one provides its input size. Next column refers to the autosymmetry degree (i.e., k) of the function. The following two pairs of columns report the multiplicative complexity of the XAG (M_X) after applying the heuristic in [25] and the time in seconds required to obtain it, for the entire function f (first couple) and for the corresponding restriction f_k (second couple). Finally, the last column reports the gain in applying the autosymmetry test before XAG synthesis.

TABLE I
EXPERIMENTAL COMPARISON OF AUTOSYMMETRIC BENCHMARKS, CONSIDERING A XAG AFTER THE AUTOSYMMERTY TEST AND THE STANDARD XAG COMPUTED WITHOUT THE AUTOSYMMETRY TEST.

			standard XAG		XAG with autosym. test		
Benchmark	**in**	**k**	$\mathbf{M_X(f)}$	**time (s)**	$\mathbf{M_X(f_k)}$	**time (s)**	**gain**
add6(0)	12	11	3	0.01	0	0.01	100%
bcc(32)	26	11	26	17.15	21	19.45	19%
bcc(33)	26	11	61	56.00	54	85.34	11%
exep(1)	30	18	17	7.33	15	9.36	12%
in5(3)	24	5	31	14.13	27	12.46	13%
in7(1)	26	10	21	15.68	15	11.48	29%
in7(5)	26	5	34	24.94	30	35.27	12%
mainpla(27)	27	3	180	147.01	147	130.00	18%
mish(4)	94	91	2	0.01	2	0.01	0%
opa(24)	17	11	9	2.15	5	1.98	44%
opa(25)	17	2	39	26.51	31	22.84	21%
pdc(13)	16	8	108	8.67	7	2.96	94%
pdc(26)	16	2	25	22.57	12	6.16	52%
t1(19)	21	18	5	0.18	2	0.04	60%
t2(6)	17	4	17	12.49	14	11.6	18%
t2(9)	17	10	21	14.87	18	10.35	14%
vg2(6)	25	11	17	12.15	15	8.37	12%
x2dn(33)	82	79	2	0.01	2	0.01	0%
x6dn(0)	39	11	82	62.37	66	54.57	20%
x6dn(4)	39	10	93	74.83	82	77.12	12%
x7dn(1)	66	51	20	10.64	20	9.26	0%
xparc(0)	41	17	105	84.82	93	75.19	11%

Table II shows a summary of the overall experimental results. We first consider the set of all autosymmetric functions (degenerate[1] and non-degenerate), we then study the truly autosymmetric (i.e., the non-degenerate) ones. In Table II, we denote with $M_X(f_k) < (=, >,$ resp.$)\ M_X(f)$ the number of benchmarks where the number of ANDs of the XAG for f_k is less than (equal to, greater than, resp.) the number of ANDs of the XAG for f. We notice that the XAG minimization algorithm proposed in [25] is sensible to degenerate functions as shown in the first row of Table I, where the number of benchmarks where f_k and f have the same number of ANDs is the majority (i.e., about 68%). Nevertheless, if we concentrate on non-degenerate autosymmetric functions (i.e., second row of the table) we notice that the number of benchmarks where $M_X(f_k) < M_X(f)$ is about 74%. Moreover, in this set the average gain is about 61%, while the overall gain in the entire set of autosymmetric functions (in the case $M_X(f_k) < M_X(f)$) is about 31%. Finally, the results on computational times are not very interesting, since the two compared approaches have similar synthesis times.

From these experiments, we can conclude that, when a function is truly autosymmetric (i.e., non-degenerate), we can

[1]Recall that degenerate functions are, by definition, autosymmetric.

TABLE II
SUMMARY OF THE EXPERIMENTAL EVALUATION, CONSIDERING THE NUMBER OF ANDS IN THE XAGS FOR AUTOSYMMETRIC FUNCTIONS AND NON-DEGENERATE AUTOSYMEMTRIC FUNCTIONS.

	$M_X(f_k) < M_X(f)$	$M_X(f_k) = M_X(f)$	$M_X(f_k) > M_X(f)$
autosymmeric functions	11.34%	68.2%	20.46%
non-degenerate autosymmeric functions	74.2%	19.35%	6.45%

obtain better results computing the XAG on the restriction f_k instead of computing the XAG directly on the function f.

V. CONCLUSION

In this paper we have considered the class of autosymmetric functions and we have shown how autosymmetry can be exploited to better estimate their multiplicative complexity. Moreover, the experimental results show that autosymmetry test can enable the XAG minimization of autosymmetric functions. Section II-A shows that XOR nodes are costless in multiparty computation. In general, the computation between parties can be applied to any Boolean functions; for this reason we have conducted our experimentation on a wide set of standard Boolean benchmarks. As a future work, we plan to investigate benchmarks related to more general security and cryptographic applications.

ACKNOWLEDGMENT

We are in debt to our student Alessandro Poggiali who carried out the benchmark experimentation.

REFERENCES

[1] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner, "Ciphers for MPC and FHE," in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, 2015, pp. 430–454.

[2] D. W. Archer, D. Bogdanov, B. Pinkas, and P. Pullonen, "Maturity and performance of programmable secure computation," *IEEE Security Privacy*, vol. 14, no. 5, pp. 48–56, Sep. 2016.

[3] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 478–492.

[4] A. Bernasconi, V. Ciriani, R. Drechsler, and T. Villa, "Logic Minimization and Testability of 2-SPP Networks," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1190–1202, 2008.

[5] A. Bernasconi and V. Ciriani, "DRedSOP: Synthesis of a New Class of Regular Functions," in *DSD*, 2006, pp. 377–384.

[6] ——, "Dimension-Reducible Boolean Functions Based on Affine Spaces," *ACM Trans. Design Autom. Electr. Syst.*, vol. 16, no. 2, pp. 13:1–13:21, 2011.

[7] A. Bernasconi, V. Ciriani, F. Luccio, and L. Pagli, "Three-Level Logic Minimization Based on Function Regularities," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 22, no. 8, pp. 1005–1016, 2003.

[8] ——, "Synthesis of autosymmetric functions in a new three-level form," *Theory Comput. Syst.*, vol. 42, no. 4, pp. 450–464, 2008.

[9] J. Boyar, R. Peralta, and D. Pochuev, "On the multiplicative complexity of boolean functions over the basis (cap, +, 1)," *Theor. Comput. Sci.*, vol. 235, no. 1, pp. 43–57, 2000.

[10] R. K. Brayton and A. Mishchenko, "ABC: an academic industrial-strength verification tool," in *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, 2010, pp. 24–40.

[11] Ç. Çalik, M. S. Turan, and R. Peralta, "The multiplicative complexity of 6-variable boolean functions," *Cryptography and Communications*, vol. 11, no. 1, pp. 93–107, 2019.

[12] S. Cimato, V. Ciriani, E. Damiani, and M. Ehsanpour, "An obdd-based technique for the efficient synthesis of garbled circuits," in *Security and Trust Management - STM*, 2019, pp. 158–167.

[13] V. Ciriani, "Synthesis of SPP Three-Level Logic Networks using Affine Spaces," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1310–1323, 2003.

[14] D. Debnath and T. Sasao, "A Heuristic Algorithm to Design AND-OR-EXOR Three-Level Networks," in *Asia and South Pacific Design Automation Conference*, 1998, pp. 69–74.

[15] E. Dubrova, D. Miller, and J. Muzio, "AOXMIN-MV: A Heuristic Algorithm for AND-OR-XOR Minimization," in *Int. Workshop on the Applications of the Reed Muller Expansion in circuit Design*, 1999, pp. 37–54.

[16] Y. Ejgenberg, M. Farbstein, M. Levy, and Y. Lindell, "Scapi: The secure computation application programming interface." *IACR Cryptology EPrint Archive*, vol. 2012, p. 629, 2012.

[17] D. Goudarzi and M. Rivain, "On the multiplicative complexity of boolean functions and bitsliced higher-order masking," in *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, Proceedings*, 2016, pp. 457–478.

[18] I. Halecek, P. Fiser, and J. Schmidt, "Are xors in logic synthesis really necessary?" in *20th IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems, DDECS 2017, Dresden, Germany, April 19-21, 2017*, 2017, pp. 134–139.

[19] ——, "Towards AND/XOR balanced synthesis: Logic circuits rewriting with XOR," *Microelectronics Reliability*, vol. 81, pp. 274–286, 2018.

[20] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits." in *USENIX Security Symposium*, vol. 201, no. 1, 2011, pp. 331–335.

[21] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2008, pp. 486–498.

[22] F. Luccio and L. Pagli, "On a New Boolean Function with Applications," *IEEE Transactions on Computers*, vol. 48, no. 3, pp. 296–310, 1999.

[23] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2009, pp. 250–267.

[24] E. M. Songhori, S. U. Hussain, A. Sadeghi, T. Schneider, and F. Koushanfar, "Tinygarble: Highly compressed and scalable sequential garbled circuits," in *2015 IEEE Symposium on Security and Privacy*, May 2015, pp. 411–428.

[25] E. Testa, M. Soeken, L. G. Amarù, and G. D. Micheli, "Reducing the multiplicative complexity in logic networks for cryptography and security applications," in *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA*, 2019, p. 74.

[26] ——, "Logic synthesis for established and emerging computing," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 165–184, 2019.

[27] M. S. Turan and R. Peralta, "The multiplicative complexity of boolean functions on four and five variables," in *Lightweight Cryptography for Security and Privacy - Third International Workshop, LightSec 2014, Istanbul, Turkey*, 2014, pp. 21–33.

[28] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*. Addison-Wesley Publishing Company, 1993.

[29] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," Microelectronic Center, User Guide, 1991.

[30] A. C. Yao, "Protocols for secure computations," in *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*. IEEE, 1982, pp. 160–164.

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