Pushing the Communication Barrier in Secure Computation using Lookup Tables (Full Version)*

Ghada Dessouky[1], Farinaz Koushanfar[2], Ahmad-Reza Sadeghi[1], Thomas Schneider[3], Shaza Zeitouni[1], and Michael Zohner[3]

[1]TU Darmstadt, System Security Lab, name.surname@trust.tu-darmstadt.de
[2]University of California, Adaptive Computing and Embedded Systems Lab, fkoushanfar@ucsd.edu
[3]TU Darmstadt, Engineering Cryptographic Protocols Group, name.surname@crisp-da.de

Abstract

Secure two-party computation has witnessed significant efficiency improvements in the recent years. Current implementations of protocols with security against passive adversaries generate and process data much faster than it can be sent over the network, even with a single thread. This paper introduces novel methods to further reduce the communication bottleneck and round complexity of semi-honest secure two-party computation. Our new methodology creates a trade-off between communication and computation, and we show that the added computing cost for each party is still feasible and practicable in light of the new communication savings. We first improve communication for Boolean circuits with 2-input gates by factor 1.9x when evaluated with the protocol of Goldreich-Micali-Wigderson (GMW). As a further step, we change the conventional Boolean circuit representation from 2-input gates to multi-input/multi-output lookup tables (LUTs) which can be programmed to realize arbitrary functions. We construct two protocols for evaluating LUTs offering a trade-off between online communication and total communication. Our most efficient LUT-based protocol reduces the communication and round complexity by a factor 2-4x for several basic and complex operations. Our proposed scheme results in a significant overall runtime decrease of up to a factor of 3x on several benchmark functions.

1 Introduction

Secure computation allows two or more parties to evaluate a public function on their private inputs without revealing any information except what can be inferred from the output. In the context of secure two-party computation with security against passive (semi-honest, honest but curious) adversaries, the most prominent protocols are *Yao's garbled circuits* [Yao86] and the protocol by *Goldreich-Micali-Wigderson (GMW)* [GMW87]. Yao's garbled circuits protocol securely evaluates a function, represented as Boolean circuit, in a constant number of rounds. The Boolean circuit consists of XOR gates, which can be evaluated for free [KS08], and AND gates, for which the parties have to send data. The GMW protocol also works on Boolean circuits where XOR gates can be evaluated locally without any communication and is divided in two phases: a *setup phase* and an *online phase*. The *setup phase* is executed prior to the actual function evaluation and is independent of the pertinent function and the parties' private inputs. It allows to pre-compute all communication-intensive symmetric cryptographic operations and oblivious transfers (OTs, cf. §2.3) to generate helper data. The *online phase* begins when the parties secret-share their private inputs and lasts

*Please cite the conference version of this paper published at NDSS 2017 [DKS+17].

throughout the evaluation of the function circuit using the pre-computed helper data until the final output is computed. The main difference of both protocols is that the round complexity of the online phase is constant for Yao's protocol, but linear in the depth of the circuit in GMW.

In recent years, the practical efficiency of secure two-party computation schemes has been dramatically improved by orders of magnitudes, making solutions ready for deployment in practice [BCD+09, BJSV15, Sec15, SHS+15]. One of the key enablers for these improvements has been the efficient instantiation of underlying cryptographic primitives, which decreased the computational cost per cryptographic operation close to negligible [BHKR13, GLNP15]. While the computation has been dramatically reduced, the communication improvements have been smaller, shifting the bottleneck in current protocol implementations towards communication. In particular, the work of [BHKR13] computes at the speed of nearly 2 Gbit/second per thread. It has been shown in [ZRE15] that today's best instantiation of Yao's protocol of [ZRE15] has hit a lower bound of two κ-bit ciphertexts per AND gate in the Boolean circuit, where κ is the symmetric security parameter.

In contrast, for the GMW protocol, it has been shown that it is still possible to achieve communication less than two κ-bit ciphertexts per AND gate [KK13]. The GMW protocol allows that all symmetric cryptographic operations are pre-computed in the setup phase without knowing the function beforehand, unlike Yao's protocol, and thus offers the possibility of a very efficient online phase. Therefore, GMW is the candidate of choice in our work and the basis of our improved protocols. However, the multi-round online phase of GMW greatly reduces its practicality for many real-world secure computation applications. In order to speed up this online phase, recent work of [IKM+13, DZ16] has introduced protocols that use multi-input tables rather than traditional 2-input Boolean gates to reduce the number of communication rounds. To precompute these tables, the communication complexity in the setup phase was extremely increased, which is a common approach for improving the online phase. However, the large communication overhead introduced by these protocols is particularly intolerable for most practical purposes and real-world applications and would scale very poorly as the function size grows. Furthermore, the protocols are mostly theoretical with no evaluation of applicability besides AES in [DZ16].

In summary, the main bottlenecks in passively secure two-party computation today are the setup communication (which dominates the total communication) and the online round complexity, both of which are often at a trade-off. Existing general purpose schemes achieve either low setup communication or low online round complexity, but not both.

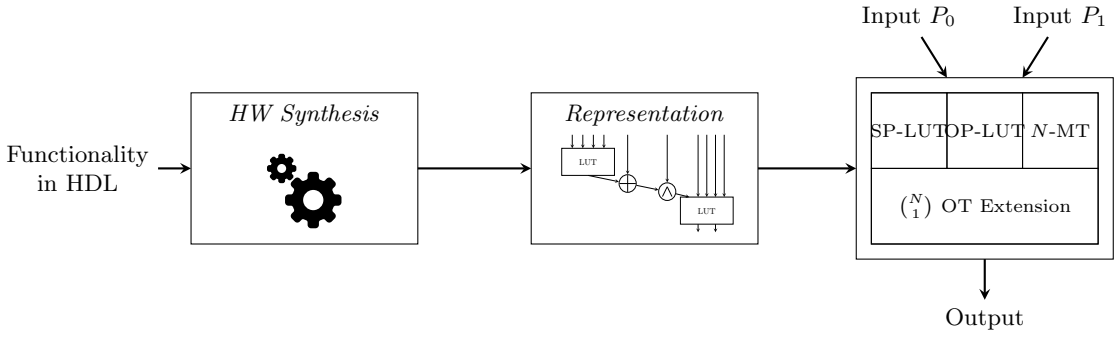


Figure 1: Our toolchain for compiling a high-level hardware description language into a network of 2-input Boolean gates and LUTs and evaluating them using our communication-efficient SP-LUT, OP-LUT, and N-MT protocols, which build on our improved $\binom{N}{1}$ OT extension protocol.

1.1 Our Contributions

In this paper, we present a more holistic end-to-end solution that significantly reduces the communication complexity in semi-honest secure two-party computation, while simultaneously maintaining a low number of communication rounds.

LUT-based Secure Computation. We replace the function representation as 2-input Boolean gate circuit by a more compact multi-input *lookup table (LUT)*-based representation. This enables the evaluation of more complex functions by representing the entire functionality as a compact graph of interconnected LUTs. We describe, optimize and implement protocols to evaluate LUT-based circuits which reduce the communication overhead significantly by a factor 4x compared to state-of-the-art Yao's protocol [ZRE15] and the round complexity by factor 4x compared to the GMW protocol. Using multi-input gates in secure computation has been investigated before, but previous works incurred a drastic communication overhead even for a small number of inputs and only considered this approach for special functions such as the AES S-box, rendering their protocols unpractical, non-generic, and unscalable for real-world applications [HEKM11, KK12, IKM+13, MPS15, DZ16]. Instead, we describe a very natural generalization of the protocol of [Gol04, Sect. 7.3.3] from the case of 2-input gates to d-input gates, which together with the 1-out-of-N OT extension protocol of [KK13] (and the additional optimizations we propose in our work) enables computation of any functionality and makes them more practical and scalable for real-world applications.

LUT Protocols. We construct two protocol variants for evaluating LUT-based circuits, called OP-LUT and SP-LUT, that offer a trade-off between improved online communication (OP-LUT) and improved setup/total communication (SP-LUT). Our LUT protocols can also be used to evaluate 2-input Boolean gates using the GMW protocol at no additional cost, since all use XOR-based secret sharing. This allows that we can effectively and seamlessly combine the benefits of both representations with our protocols. Further details on their construction can be found in §4.

More Efficient $\binom{N}{1}$ OT Extension. A key building block for our LUT protocols is the 1-out-of-N oblivious transfer extension protocol, denoted as $\binom{N}{1}$ OT extension. We use the most communication-efficient OT of [KK13] as a starting point and introduce further optimizations to reduce both its computation and communication overhead. We propose a protocol called N-MT (multiplication-triple generation based on $\binom{N}{1}$ OT), which leverages our optimizations to achieve a communication reduction per AND gate by a factor of 1.9x from 256 bits to 134 bits in the GMW protocol, for security parameter $\kappa = 128$, in comparison to the traditional 2-MT (multiplication-triple generation based on $\binom{2}{1}$ OT) of [ALSZ13] and a reduction by a factor of 1.2x compared to the protocol of [KK13]. We describe our optimization techniques in detail in §3.

Compiler for LUT-based Secure Computation. Since we move away from 2-input Boolean gates, we require new optimized LUT-based circuit representations of functions. However, building such circuits by hand is tedious, challenging, and error-prone. Instead, we construct an automated toolchain that transforms high level function descriptions into a LUT representation. More specifically, we re-purpose hardware synthesis tools for secure computation as first shown in [SHS+15, DDK+15], but for LUT-based synthesis tools, which we customize and manipulate to automatically and efficiently generate multi-input multi-output LUT representations. An in-depth description of the hardware synthesis tool leveraged and how we re-purpose it can be found in §5.

Evaluation on Basic Operations and Applications. We demonstrate the improved efficiency and practicality of our LUT protocols by evaluating a wide range of functionalities. Our protocols are shown to improve on the communication of floating point operations by factor 2-4x and the round complexity by factor 3-4x. We report and discuss our extensive evaluation results for basic operations in §6 and more complex applications in §7. For some operations, our most efficient LUT protocol achieves as little as half a κ-bit ciphertext communication per AND gate. In terms of actual runtime, our protocols achieve up to 3x faster runtime for AES and private set intersection.

1.2 High-Level Idea of Our Scheme

We construct a toolchain, presented in Fig. 1, that compiles functions described in a high-level hardware description language into a mixed representation of LUTs and 2-input Boolean gates using a hardware synthesis tool that we customize and re-purpose for our setting. These circuit representations can then be evaluated in a communication-efficient manner using our OP-LUT or SP-LUT protocols for LUT gates and using GMW with our N-MT pre-computation method for 2-input Boolean gates. Note that the LUT

protocols can be freely combined with GMW at no additional cost, since all schemes are based on XOR secret sharing. Our protocols are based on the $\binom{N}{1}$ OT extension protocol of [KK13] which runs in the setup phase that we further optimize in terms of both communication and computation. We evaluate our protocols on various basic operations and applications in secure computation and show that our LUT protocols often achieve significantly better communication and round complexity than traditional 2-input Boolean gate representations. Our synthesized LUT representations and implementations are available within the ABY framework [DSZ15] at `http://github.com/encryptogroup/ABY`.

1.3 Outline

We provide preliminaries and background in §2. Next, we describe in more detail our improved $\binom{N}{1}$ OT extension protocol in §3, followed by a description of our LUT-based representation and protocols in §4. Our customized hardware synthesis approach is given in §5. Finally, we show an extensive evaluation of our toolchain for basic operations in §6 and applications in §7. We give related works in §8 before we conclude and give future works in §9.

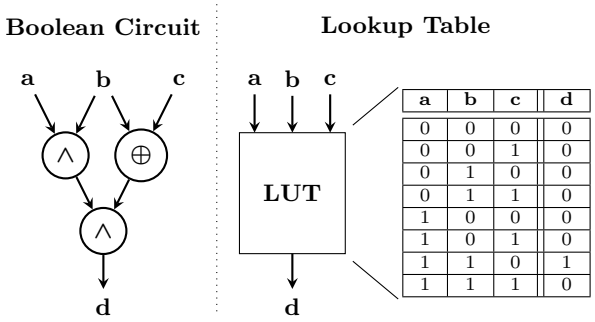
2 Preliminaries

2.1 Notation

We denote the two parties as P_0 and P_1 or sender P_S and receiver P_R and the symmetric security parameter as κ, which we fix to $\kappa = 128$ throughout this paper.

2.2 LUT-based Boolean Circuits

In our context, a Lookup Table (LUT) is the set of all functions that map $\delta \geq 2$ input bits to σ output bits (cf. Fig. 2 for an example). Using this representation, complex functionalities can be built as a compact graph of interconnected LUTs.



a	b	c	d
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Figure 2: A function with $\delta = 3$ input and $\sigma = 1$ output bits represented as Boolean circuit with 2-input gates (left) and lookup table (right).

2.3 Oblivious Transfer

In 1-out-of-N oblivious transfer, denoted $\binom{N}{1}$ OT, a sender P_S inputs N messages $(x_1, ..., x_N)$ from which a receiver P_R with selection input $r \in [1...N]$ obtains message x_r obliviously such that P_S does not learn P_R's choice r and P_R does not learn any information about x_i with $i \neq r$. By $\binom{N}{1}$ OT$_n^m$ we denote m invocations of $\binom{N}{1}$ OT, where each of the N messages has length n bits. OT is a fundamental primitive in cryptography and heavily used in secure computation. In [IR89] it was shown that OTs cannot be based on one-way functions, i.e., OT requires at least some public-key cryptography. Today's fastest public-key based OT protocol of [CO15] is able to compute 10,000 OTs per second.

OT Extension. In [IKNP03] it was shown that it is possible to "extend" a few (around κ) public-key base-OTs to an arbitrary number of OTs using symmetric cryptography only. Due to their nature, these protocols are called *OT extension* protocols. The communication cost of today's most efficient 1-out-of-2 OT extension protocol is $\mathrm{C}(\binom{2}{1}\ \mathrm{OT}_n^m) = m\kappa + 2mn$ bits [ALSZ13, KK13]. The implementation of [ALSZ13] is able to compute one million OTs per second.

Random OT. Random OT is a special-purpose OT functionality, tailored for more efficient secure computation. In contrast to the standard OT functionality, in a 1-out-of-N random OT, denoted $\binom{N}{1}$ R-OT, the sender inputs no messages to the OT protocol, but receives the messages as a random output of the protocol itself, while the receiver still inputs its selection string to choose one. This allows to remove the last message in the OT which decreases the communication to $\mathrm{C}(\binom{2}{1}\ \text{R-OT}_n^m) = m\kappa$ bits [ALSZ13, KK13].

2.4 Yao's Garbled Circuits

Yao's garbled circuits protocol [Yao86] allows two parties to securely evaluate any function, represented as a Boolean circuit. One party, the circuit garbler, assigns symmetric keys corresponding to 0 and 1 to the wires of the Boolean circuit. The garbler then garbles the circuit by encrypting the keys of the output wires of each gate using the keys of the gate's input wires. These encryptions form the garbled tables of the circuit and are transferred to the evaluator, together with the keys that correspond to both parties' input wires to the circuit. The evaluator then iteratively decrypts the correct output keys of the gates using the corresponding input keys and obtains the output of the circuit using a mapping, provided by the garbler.
Several optimizations for Yao's garbled circuits have been proposed, most notably: point-and-permute [MNPS04], free-XOR [KS08], fixed-key AES garbling [BHKR13], and half-gates [ZRE15]. Overall, the garbler has to send 2κ bits to the evaluator per AND gate, which can be done in the setup phase if the function is known. In the online phase, the evaluator locally decrypts the garbled table and computes the output in a constant number of communication rounds.

2.5 Goldreich-Micali-Wigderson

The GMW protocol [GMW87] for secure computation also represents a function as a Boolean circuit and secret shares the values on the wires between the parties using an XOR-based secret-sharing scheme. XOR gates can be evaluated for free locally by XORing the shares while AND gates require one interaction step between the parties using a *multiplication triple*. A multiplication triple (MT) is a set of shares of the form $(c_0 \oplus c_1) = (a_0 \oplus a_1) \wedge (b_0 \oplus b_1)$, where P_i holds the shares labeled with i, for $i \in \{0, 1\}$. MTs can be pre-computed using $\binom{2}{1}$ R-OT$_1^2$ at the cost of 2κ bits of communication [ALSZ13] and are used in the online phase to evaluate AND gates at the cost of 4 bits communication. In §3.5 we show how to pre-compute MTs with less communication overhead using the $\binom{N}{1}$ OT extension of [KK13]. For details on the GMW protocol please refer to [DSZ15].

2.6 Size and Depth of Boolean Circuits

For our later evaluation in §6.1, we bound the multiplicative size (the number of AND gates) and depth (the highest number of ANDs from any input to any output) of a Boolean circuit. For many functionalities, a low multiplicative size and a low multiplicative depth are two mutually exclusive goals. Hence, we first outline the case for Boolean circuits with δ input bits and $\sigma = 1$ output bit, since this allows us to set tighter upper bounds, and then examine the case for $\sigma > 1$.

Boolean Circuits with One Output Bit. It was shown in [TP14] that any functionality with $\delta \leq 5$ input bits can be realized by a Boolean circuit with at most $\delta - 1$ AND gates. For functions with $\delta > 5$ inputs, a bound on the maximum number of AND gates is still unknown but, according to [TP14], "no specific δ-variable function has yet been proven to have multiplicative complexity larger than $\delta - 1$ for any δ". We bound the number of AND gates in a Boolean circuit C with δ inputs by $S(C) \leq \delta - 1$. In [BB94] it was shown that every Boolean circuit of multiplicative size n has an equivalent Boolean circuit of multiplicative

depth $O(\log n)$ and size $O(n^\alpha)$ for arbitrary $\alpha > 1$. We bound the multiplicative depth of a circuit C with δ inputs by $D(C) \leq \log_2(\delta)$.

Boolean Circuits with Multiple Output Bits. Finding a size- or depth-optimal Boolean circuit for functionalities with $\sigma > 1$ outputs is a hard problem for a larger number of inputs δ [BP12] and determining a minimal upper bound is a complex task out of scope of this paper. A more tractable approach to find a possible upper bound is to build optimal Boolean circuits for each output bit separately. In this paper, we take this approach and assume that a Boolean circuit C with δ input and σ output bits has at most size $S(C) \leq \sigma(\delta - 1)$ if optimized for size and $D(C) \leq \log_2 \delta$ if optimized for depth.

3 More Efficient $\binom{N}{1}$ OT Extension

In order to evaluate a function using our LUT protocols, we pre-compute the LUTs using OT. However, using the standard $\binom{2}{1}$ OT extension protocol of [IKNP03, ALSZ13] to pre-compute the LUTs would result in a higher communication overhead than evaluating traditional Boolean circuits. Therefore, for improved communication efficiency, we make use of the $\binom{N}{1}$ OT extension protocol of [KK13]. Although the [KK13] OT extension protocol is very communication-efficient, it incurs a significant computation overhead: N symmetric operations compared to $2\log_2 N$ symmetric operations by the $\binom{2}{1}$ OT extension protocol of [IKNP03, ALSZ13]. In this paper, we take the OT extension protocol of [KK13] as a starting point for improving communication and introduce optimization mechanisms to effectively reduce both, its computation and communication overhead.
In this section, we give an overview of the $\binom{N}{1}$ OT protocol (§3.1), outline how to more efficiently instantiate the underlying error correcting code (§3.2) and sample random choice bits of the receiver to reduce the communication overhead (§3.3). Next, we present our optimizations of the underlying symmetric cryptographic primitives to reduce the computation overhead (§3.4). Finally, we show how to optimize the evaluation of AND gates and reduce the communication overhead in the setup phase of the GMW protocol (§3.5). We call our resulting protocol that combines our proposed optimizations for more efficient evaluation of AND gates N-MT. We give a full description of the $\binom{N}{1}$ OT extension protocol of [KK13] in Prot. 1.

3.1 Protocol Description

In the $\binom{2}{1}$ OT extension protocol of [IKNP03], the parties use multiple base-OTs to obliviously transfer shares of the receiver's selection bits. The main observation of the $\binom{N}{1}$ OT protocol of [KK13] was that this approach can be generalized to have both parties share a ρ-bit codeword from a code Γ^ρ with codewords of Hamming distance κ. These codewords encode the receiver's selection strings and constitute the main component of the communication workload of the OT extension protocol.
For $N = 2$, a repetition code can be used, which has 2 codewords of size $\rho = \kappa$. In this case, the $\binom{N}{1}$ OT protocol of [KK13] is identical to the $\binom{2}{1}$ OT protocol of [IKNP03].
For $2 < N \leq 2\kappa$, the authors of [KK13] propose to use a Walsh-Hadamard code which has codewords of size $\rho = 2\kappa$ to achieve a relative Hamming distance of κ. They show that, for appropriate choice of parameters $(\kappa = 128, \rho = 256, N = 16)$, generating $\binom{2}{1}\text{OT}_1^{\log_2 N}$ from their $\binom{N}{1}$ OT protocol requires only 320 bits of communication while the $\binom{2}{1}$ OT of [IKNP03, ALSZ13] requires 520 bits.
For $N > 2\kappa$, a linear error-correcting code achieves the best performance. In particular, when $N = poly(\kappa)$, the communication cost for the OT extension part of $\binom{N}{1}$ OT invocations decreases asymptotically from $O(\kappa \log N)$ to $O(\kappa)$ compared to a $\binom{2}{1}$ OT instantiation.

3.2 Our Size-Optimized Codes

The efficiency gains from the $\binom{N}{1}$ OT extension protocol of [KK13] for $N > 2$ are due to efficient instantiations of the underlying codes. For $2 < N \leq 2\kappa$, [KK13] uses a Walsh-Hadamard code, which has codewords of size $\rho = 2\kappa = 256$ bits to achieve a Hamming distance of $\kappa = 128$ between codewords. However, for $N = 2^i$,

PROTOCOL 1 ($\binom{N}{1}$ OT extension protocol [KK13])

- **Common Input:** Symmetric security parameter κ; code $\Gamma^\rho = (\gamma_1, ..., \gamma_N)$ with ρ-bit codewords.
- **Input of P_S:** m tuples $(x_j^1, ..., x_j^N)$ of n-bit strings.
- **Input of P_R:** m selection integers $r = (r_1, \ldots, r_m)$ with $r_j \in [N]$.
- **Oracles and cryptographic primitives:** An ideal $\binom{2}{1}$ OT_κ^ρ primitive, a pseudo-random generator $G : \{0,1\}^\kappa \to \{0,1\}^m$ and a correlation-robust function $H : [m] \times \{0,1\}^\rho \to \{0,1\}^n$.

1. P_S initializes a random vector $s = (s_1, \ldots, s_\rho) \in_R \{0,1\}^\kappa$ and P_R chooses ρ pairs of seeds $(k_i^0, k_i^1) \in_R \{0,1\}^\kappa$.
2. The parties invoke $\binom{2}{1}$ OT_κ^ρ, where P_S acts as the *receiver* with input s and P_R acts as the *sender* with inputs (k_i^0, k_i^1) for every $1 \le i \le \rho$.

   P_R forms two $m \times \rho$ bit matrices $T = [t^1 | \ldots | t^\rho]$ with $t^i = G(k_i^0)$ (where its i-th column is t^i and its j-th row is t_j) and $C = [c_1 | \ldots | c_m]$, (where its i-th column is c^i and its j-th row is c_j) with $c_j = \gamma_{r_j}$ for $1 \le i \le \rho$ and $1 \le j \le m$.
3. P_R computes and sends $u^i = t^i \oplus G(k_i^1) \oplus c^i$ to P_S for every $1 \le i \le \rho$.
4. For every $1 \le i \le \rho$, P_S defines $q^i = (s_i \cdot u^i) \oplus G(k_i^{s_i})$. (Note that $q^i = (s_i \cdot c^i) \oplus t^i$.)

   Let $Q = [q^1 | \ldots | q^\ell]$ denote the $m \times \ell$ bit matrix where its i-th column is q^i. Let q_j denote the j-th row of the matrix Q. (Note that $q^i = (s_i \cdot c^i) \oplus t^i$ and $q_j = (c_j \wedge s) \oplus t_j$.)
5. For $p \in [N]$, P_S computes $y_j^p = x_j^p \oplus H(j, q_j \oplus \gamma_p)$ and sends $(y_j^1, \ldots, y_j^N)$ for every $1 \le p \le N$ and $1 \le j \le m$.
6. For $1 \le j \le m$, P_R computes $x_j = y_j^{r_j} \oplus H(j, t_j)$.
7. **Output:** P_R outputs $(x_1, \ldots, x_m)$; P_S has no output.

with $2 \le i \le 8$, the Walsh-Hadamard code is not size-optimal with regard to the codeword size ρ. Hence, we propose to use more size-efficient codes in order to further decrease the communication. We base our code choices on the list of efficient codes in [SS06] and give the codeword sizes for $N = 2^i$ for $1 \le i \le 12$ in Tab. 1. In particular, for $N = 4$ we use a parity check code, for $N \in \{8, 16, 32, 64, 128, 256\}$, we use a Simplex code, for $N = 512$, we use a Reed-Muller code, for $N \in \{1{,}024, 2{,}048\}$, we use a narrow-sense BCH-code, and for $N = 4{,}096$, we use the concatenation of a Denniston code and a Simplex code (see [SS06] for more details). The OT communication improvements achieved by adopting our reduced codeword sizes are the largest for $N = 4$ (reduced by 64 bits) and decrease with N growing towards 256 (reduced by 1 bit). Note that using size-optimized codes does not increase computation or reduce security over using the Walsh-Hadamard codes.

3.3 Random Choice Bits

In all $\binom{N}{1}$ OT protocol invocations throughout this work, the receiver samples and inputs a random selection string $r \in_R [1...N]$. However, we observe that the communication from receiver to sender can be reduced by having the $\binom{N}{1}$ OT protocol sample r randomly during the execution and output it to the receiver. In

N	2	4	8	16	32	64
Our Size-Efficient Codes [bits]	128	192	224	240	248	252
[KK13] Codes [bits]	128	256	256	256	256	256
N	**128**	**256**	**512**	**1024**	**2048**	**4096**
Our Size-Efficient Codes [bits]	254	255	256	264	268	270
[KK13] Codes [bits]	256	256	-	-	-	-

Table 1: Communication for $\binom{N}{1}$ OT with size-optimal codes [SS06] compared to those used in [KK13].

order to randomly sample r, we transform the code Γ^ρ into a *systematic form*, similar to [FJJBT16]. In the systematic form, the input data is embedded into the codeword, i.e., the integer $s \in \{0,1\}^{\log_2 N}$ is a sub-string of codeword c_s. Assume that s is embedded in the first $\log_2 N$ positions of each c_s. We can now let the receiver compute the choice bits r_j in the j-th OT as $r_j = r_{j,1}||...||r_{j,log_2 N}$ with $r_i = G(k_i^0)_j \oplus G(k_i^1)_j$ in Step 2 in Prot. 1. Consequently, we can change Step 3 and Step 4 to avoid sending the u_j values for these positions. In particular, we have P_R perform Step 3 only for $\log_2 N < i \leq \rho$ and P_S compute in Step 4:

$$q^i = \begin{cases} G(k_i^{s_i}), & \text{if } 1 \leq i \leq \log_2 N \\ (s_i \cdot u^i) \oplus G(k_i^{s_i}), & \text{else.} \end{cases}$$

Overall, this enables us to further reduce the communication for $\binom{N}{1}$ OT by $\log_2 N$ bits. Furthermore, this can be combined with $\binom{N}{1}$ R-OT to further reduce the communication overhead.

3.4 Pipelined AES-256

In OT extension [IKNP03, KK13], both parties process several value tuples that are correlated by a constant XOR offset using a correlation-robust function (CRF) (cf. H in Step 5 and Step 6 in Prot. 1). While the CRF has traditionally been instantiated with a hash function, more efficient AES-based constructions have been used to replace it [KSS12, BHKR13, GLNP15]. When using the most efficient, fixed-key AES instantiation [BHKR13], the input is restricted to the block-length of AES, i.e., 128-bit, which suffices for the $\binom{2}{1}$ OT extension protocol of [IKNP03] when $\kappa = 128$-bit. However, in the $\binom{N}{1}$ OT extension of [KK13], we need to process codewords of size $\rho > 128$ bits for $N > 2$, which prevents the use of fixed-key AES. Falling back to a hash function or AES-256 with key schedule [KSS12] greatly decreases performance by about an order of magnitude, as depicted in Tab. 2. Furthermore, the $\binom{N}{1}$ OT protocol requires N invocations of an expensive CRF (instantiated via AES-256 with key schedule or SHA-256) as opposed to $2 \log N$ invocations of a cheaper CRF (instantiated via AES-128) when using $\binom{2}{1}$ OT. In particular, for our protocols in §4 we use $N = 256$, which requires 256 CRF invocations when using $\binom{N}{1}$ OT compared to 16 invocations when using $\binom{2}{1}$ OT. Using AES-256 with key schedule instantiation for $\binom{N}{1}$ OT and the pipelined AES-128 instantiation of [GLNP15] for $\binom{2}{1}$ OT, this results in a computational overhead of 480x.

Primitive	Width	Time [ms]	Pipe-Time [ms]
AES-128 [BHKR13]	128	158	54
AES-128+KS [GLNP15]	128	1,460	358
AES-256+KS [KSS12]	256	1,625	476
SHA-256	arbitrary	2,487	-

Table 2: Instantiations of a correlation-robust function with input width in bits and (pipelined) run-time for 10^7 invocations.

We improve the performance of the CRF instantiation based on AES-256 with key schedule by pipelining the AES-256 key expansion and encryption routines as well as pipelining multiple invocations of AES, similar to the approach of [GLNP15] for AES-128. Thereby, we manage to decrease the computation time for AES-256 by factor 4, which reduces the computational overhead compared to $\binom{2}{1}$ OT to 140x. When evaluating a $\binom{256}{1}$ $\text{OT}_1^{10^7}$ using the [KK13] protocol, this reduces the evaluation time from 79 s to 22 s. For $\rho > 256$, we instantiate the CRF with SHA-256.

A promising line of research is given in [GM16], which outlines how to obtain cryptographic permutations with larger block sizes based on fixed-key AES-128. Due to security concerns, however, we refrain from using their instantiations but point it out as a future alternative to explore.

3.5 Multiplication Triples from $\binom{N}{1}$ OT

To improve the communication in secure computation, the work of [KK13] proposed to use their $\binom{N}{1}$ OT protocol to reduce $\binom{N}{1}$ $\text{OT}^1_{\log_2 N}$ to $\binom{2}{1}$ $\text{OT}^{\log_2 N}_1$. They achieved a communication saving of up to 1.6x per $\binom{2}{1}$ OT^2_1, from 256 bits to 160 bits, when setting $\kappa = 128$ and $N = 16$. In this work, we further improve on their communication savings by using our optimized $\binom{N}{1}$ OT protocol to directly compute a multiplication triple (MT), which corresponds to a $\binom{4}{1}$ OT^1_1. For this reduction, we evaluate $\binom{N}{1}$ $\text{OT}^1_{\log_4(N)}$ which we can directly transform to $\binom{4}{1}$ $\text{OT}^{\log_4(N)}_1$. We vary possible choices for N in Tab. 3 and observe that the highest improvement of 1.9x is obtained for $N = 16$, where one MT can be computed at the cost of 134 bits in the setup phase, and 2 MTs at the cost of 268 bits as shown in Tab. 3. Compared to the protocol in [KK13], our N-MT protocol reduces the communication by factor 1.2x from 160 bits to 134 bits. Adding the 4 bits for the evaluation in the online phase, the total communication is now as low as 138 bits per AND gate.

N	**4**	**8**	**16**	**32**	**64**	**128**	**256**
#Triples	*1*	*1.5*	*2*	*2.5*	*3*	*3.5*	*4*
2-MT	256	384	**512**	640	768	896	1,024
***N*-MT**	194	223	**268**	339	438	759	1,271
Improvem.	1.32	1.72	**1.91**	1.89	1.75	1.18	0.81

Table 3: Communication for generating multiplication triples using $\binom{2}{1}$ R-OT [ALSZ13] (2-MT) and $\binom{N}{1}$ OT [KK13] with our optimizations (N-MT). Best results marked in **bold.**

4 LUT-based Secure Computation

In this section, we discuss how to model the functionality as network of interconnected LUTs with multiple input bits that can be evaluated in a constant number of rounds per layer of LUTs (§4.1). We first summarize the one-time truth table (OTTT) approach of [IKM+13] with pre-computation of [DZ16] (§4.2). We then present our Online-LUT (OP-LUT) scheme, which is optimized for an efficient online phase but has high communication in the setup phase (§4.3). Next, we give the Setup-LUT (SP-LUT) protocol that dramatically reduces the communication in the setup phase but slightly increases the communication in the online phase (§4.4). Finally, we show how to optimize the online phase of the SP-LUT protocol to achieve better round and communication complexity and how to compute LUTs with overlapping inputs more efficiently (§4.5). We give a summary of the communication costs for these protocols in Tab. 4.

# Inputs δ	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Asymptotic**
Setup Communication [bits]								
OTTT [IKM+13]	$\leq 552\sigma$	$\leq 2{,}208\sigma$	$\leq 6{,}624\sigma$	$\leq 17{,}664\sigma$	$\leq 44{,}160\sigma$	$\leq\approx 2^{17}\sigma$	$\leq\approx 2^{18}\sigma$	$\leq 138(\delta-1)2^\delta\sigma$
OP-LUT (§4.3)	$16\sigma + 190$	$64\sigma + 221$	$256\sigma + 236$	$1{,}024\sigma + 243$	$4{,}096\sigma + 246$	$\approx 2^{14}\sigma$	$\approx 2^{16}\sigma$	$\text{C}(\binom{N}{1}\ \text{OT}^1_{\sigma N})-\delta$
SP-LUT (§4.4)	**190**	**221**	**236**	**243**	**246**	**247**	**247**	$\text{C}(\binom{N}{1}\ \text{R-OT})-\delta$
Online Communication [bits]								
OTTT / OP-LUT	**4**	**6**	**8**	**10**	**12**	**14**	**16**	2δ
SP-LUT	$4\sigma + 2$	$8\sigma + 3$	$16\sigma + 4$	$32\sigma + 5$	$64\sigma + 6$	$128\sigma + 7$	$256\sigma + 8$	$2^\delta\sigma + \delta$
Total Communication (Setup + Online) [bits]								
OTTT [IKM+13]	$\leq 554\sigma$	$\leq 2{,}214\sigma$	$\leq 6{,}632\sigma$	$\leq 17{,}674\sigma$	$\leq 44{,}172\sigma$	$\leq\approx 2^{17}\sigma$	$\leq\approx 2^{18}\sigma$	
OP-LUT (§4.3)	$16\sigma + 194$	$64\sigma + 227$	$256\sigma + 244$	$1{,}024\sigma + 253$	$4{,}096\sigma + 258$	$\approx 2^{14}\sigma$	$\approx 2^{16}\sigma$	
SP-LUT (§4.4)	$4\sigma + 192$	$8\sigma + 224$	$16\sigma + 240$	$32\sigma + 248$	$64\sigma + 252$	$128\sigma + 244$	$256\sigma + 255$	

Table 4: Setup, Online and Total communication for a δ-input LUT with σ outputs for the OTTT protocol [IKM+13], Online-LUT (OP-LUT) and Setup-LUT (SP-LUT). Best results marked in **bold.**

4.1 Lookup-Tables

For our protocols in this section, we assume that the parties have XOR secret-shared their private inputs and represent the functionality as network of LUTs and XOR gates. In our context, a δ-input bit LUT with σ

output bits is a table that maps an δ-bit secret-shared input to σ-bit secret-shared output and can thereby be used to represent any function $f : \{0,1\}^\delta \mapsto \{0,1\}^\sigma$. In contrast to Boolean circuits based on 2-input gates, LUT-based circuits do not use internal logic operations to map inputs to outputs and their evaluation costs depend only on the number of inputs and outputs. We show how to pre-compute and evaluate a δ-bit input LUT in the next sections. As in GMW, XOR gates can be evaluated locally by both parties XORing their respective shares. Moreover, we can reduce the number of output bits if one output bit can be computed as a linear combination of other outputs.

4.2 One-Time Truth Tables (OTTT)

In this section we describe the OTTT protocol of [IKM+13] with circuit-based pre-computation from [DZ16], which is given in Prot. 2. The high-level idea behind the OTTT protocol is that two parties hold secret shares T^0 and T^1 of a lookup table T, whose entries were randomly rotated across both dimensions using r, s such that $T^0[i] \oplus T^1[i] = T[r \oplus s \oplus i]$, for all $0 \leq i < 2^\delta$. Each of the parties knows a secret share of the truth-table as well as the rotation value, i.e., P_0 knows (T^0, r) and P_1 knows (T^1, s).

Pre-Computation. During the setup phase, the truth-table T needs to be shared such that P_0 holds (T^0, r) and P_1 holds (T^1, s). A possible method for pre-computing the table was outlined in [DZ16]: Both parties evaluate a Boolean circuit representing the table once for every possible input, resulting in an overhead of factor 2^δ compared to a Boolean circuit evaluation.[1] In more detail, the parties represent the table T as Boolean circuit $C : \{0,1\}^\delta \mapsto \{0,1\}^\sigma$. Then, P_0 and P_1 choose their random rotations values $r, s \in_R \{0,1\}^\delta$, securely evaluate $C(r \oplus s \oplus i) = z_i^0 \oplus z_i^1$ and set $T^0[i] = z_i^0$ and $T^1[i] = z_i^1$ for all $i \in [0...2^\delta - 1]$. Assuming the upper bound of $\delta - 1$ AND gates for a Boolean circuit with δ inputs from §2.6 and the optimized multiplication triple generation at 138 bits per AND gate from §3.5, this results in an overall communication of at most $138(\delta - 1)2^\delta$ bits.

Online Evaluation. In the online phase, the OTTT protocol of [IKM+13] takes as input two δ-bit share values x^0 and x^1 such that $x = x^0 \oplus x^1$ and evaluates a function f, represented as a lookup table T. The parties hold shares (T^0, r) and (T^1, s) of a permuted lookup table T such that $T^0[i] \oplus T^1[i] = T[r \oplus s \oplus i]$, where $r, s \in_R \{0,1\}^\delta$ and for all $0 \leq i < 2^\delta$. To evaluate T, the parties exchange $u = x^0 \oplus r$ and $v = x^1 \oplus s$ and compute the shared result $z^0 = T^0[u \oplus v]$ and $z^1 = T^1[u \oplus v]$. To see that $z = T[x] = z^0 \oplus z^1$, observe that $z^0 \oplus z^1 = T^0[u \oplus v] \oplus T^1[u \oplus v] = T^0[r \oplus s \oplus x] \oplus T^1[r \oplus s \oplus x] = T[x]$.

4.3 Online-LUT (OP-LUT)

We propose another method for pre-computing the shared permuted table, which performs better for small number of inputs δ. Instead of evaluating a circuit on all possible inputs, one can directly transfer all possible choices of the rotated truth-table via our optimized $\binom{N}{1}$ OT protocol described in §3. This protocol is a very natural generalization of the original GMW construction for evaluating 2-input gates using 1-out-of-4 OT as described in [Gol04, Sect. 7.3.3]. We call this protocol OP-LUT and describe it in Prot. 3.

Pre-Computation. P_0 chooses its share $T^0 \in_R (\{0,1\}^\delta \mapsto \{0,1\}^\sigma)$ and its rotation value $r \in_R \{0,1\}^\delta$ of the permuted table and computes the shares of P_1 for all possible rotation values: $(X_0, ..., X_{N-1})$, with $X_{s'} = T[r \oplus s' \oplus i] \oplus T^0$, for all $i \in [0...2^\delta - 1]$. P_0 then engages in a $\binom{N}{1}$ OT_N^1 with P_1 who inputs $s \in_R \{0,1\}^\delta$ as choice bits and obliviously obtains $T^1 = X_s = T[r \oplus s \oplus i] \oplus T^0$.

The communication cost for the pre-computation thereby becomes independent of the circuit representation but it scales with factor $2^{2\delta}$ as opposed to $138(\delta - 1)2^\delta$ for the circuit-based pre-computation. Overall, the $\binom{N}{1}$ OT-based pre-computation performs better for $\delta < 10$, while the circuit-based pre-computation performs better for $\delta \geq 10$ (cf. Tab. 4). The security of this pre-computation method is guaranteed by oblivious transfer: Neither does P_0 learn information about the rotation value or output share of P_1, since

[1] Note that the evaluated circuit can be optimized by removing duplicate gates [KSS12]. Assuming that the last gate in the circuit is an AND gate (otherwise, one could remove that last gate from the LUT), we expect the circuit after the duplicate removal to have at least one AND gate per instance, i.e., 2^δ AND gates for the 2^δ parallel evaluations.

PROTOCOL 2 (OTTT Evaluation [IKM+13, DZ16])

- **Common Input:** Input bit-size δ; Output bit-size σ; $N = 2^\delta$; Truth-table $T : \{0,1\}^\delta \mapsto \{0,1\}^\sigma$.
- **Input of P_0:** $x^0 \in \{0,1\}^\delta$.
- **Input of P_1:** $x^1 \in \{0,1\}^\delta$.

Pre-Computation [DZ16]:

1. The parties represent T as circuit $C : \{0,1\}^\delta \mapsto \{0,1\}^\sigma$.
2. P_0 chooses $r \in_R \{0,1\}^\delta$ and P_1 chooses $s \in_R \{0,1\}^\delta$.
3. P_0 and P_1 compute $z_i^0 \oplus z_i^1 = C(s \oplus r \oplus i)$ and set $T^0[i] = z_i^0$ and $T^1[i] = z_i^1$ for all $0 \leq i < N$.
4. **Output:** P_0 outputs (T^0, r); P_1 outputs (T^1, s). *Note: $\forall i$ with $0 \leq i < N$ it holds that $T^0[i] \oplus T^1[i] = T[r \oplus s \oplus i]$.*

Online Evaluation [IKM+13]:

1. P_0 sends $u = x^0 \oplus r$ to P_1; P_1 sends $v = x^1 \oplus s$ to P_0.
2. P_0 sets $z^0 = T^0[u \oplus v]$; P_1 sets $z^1 = T^1[u \oplus v]$.
3. **Output:** P_0 outputs z^0; P_1 outputs z^1, s.t. $z^0 \oplus z^1 = T[x^0 \oplus x^1]$.

the rotation value is used as selection string, nor does P_1 learn information about the rotation value or share of P_0, since P_1 gains no information on any other than the selected truth-table.

4.4 Setup-LUT (SP-LUT)

While the OTTT and OP-LUT approaches achieve a good online communication, their pre-computation cost scales with at least 2^δ, where δ is the number of input bits of a LUT. This greatly hinders the applicability of these approaches when pre-computation is not negligible, i.e., when the parties do not have a pre-established communication channel or when they wish to perform secure computation ad-hoc. In order to enable LUT-based secure computation even in settings with no pre-computation, we suggest a new protocol for securely pre-computing and evaluating LUTs. This protocol, called Setup-LUT (SP-LUT), achieves much better total communication but increases the online communication compared to the OTTT and OP-LUT protocols. The general idea of SP-LUT is simple: Pre-compute $\binom{N}{1}$ OT in the setup phase and obliviously transfer all possible outcomes of the LUT in the online phase. We give a full description of the protocol in Prot. 4.

Compared to the OP-LUT approach, the SP-LUT protocol only requires correlated randomness in the form of a pre-computed $\binom{N}{1}$ OT, which requires only little communication in the setup phase at the cost of 2^δ bits of communication during the online phase. However, the total communication of SP-LUT is much lower than that of OP-LUT, since only single bits need to be transferred instead of full truth-tables (cf. Tab. 4). The security of the SP-LUT protocol is similar to that of the GMW protocol [GMW87]: Both parties operate on secret-shared data by sacrificing a pre-computed OT on random data.

4.5 Optimizations

In the following, we discuss two optimizations for our LUT protocols: *Switching roles* to reduce the round complexity for SP-LUT and *combining LUTs* with overlapping inputs.

PROTOCOL 3 (Online-LUT (OP-LUT) - our work)

Inputs and Oracles:

- **Common Input:** Symmetric security parameter κ; number of inputs δ; $N = 2^\delta$; Truth-table $T : \{0,1\}^\delta \mapsto \{0,1\}^\sigma$.
- **Input of P_0:** $x^0 \in \{0,1\}^\delta$.
- **Input of P_1:** $x^1 \in \{0,1\}^\delta$.
- **Oracles:** Both parties have access to a $\binom{N}{1}$ $\text{OT}^1_{\sigma N}$ functionality.

Pre-Computation:

1. P_0 chooses $r \in_R \{0,1\}^\delta$ and $T^0 \in_R (\{0,1\}^\delta \mapsto \{0,1\}^\sigma)$. P_1 chooses $s \in_R \{0,1\}^\delta$.
2. P_0 computes $(X_0, ..., X_{N-1})$, with $X_{s'}[i] = T[r \oplus s' \oplus i] \oplus T^0[i]$, for all $0 \leq i, s' < N$.
3. P_0 and P_1 invoke the $\binom{N}{1}$ $\text{OT}^1_{\sigma N}$ functionality where P_0 plays the sender with inputs $(X_0, ..., X_{N-1})$ and P_1 plays the receiver with input s and output $T^1 = X_s$ s.t. $X_s[i] = T[r \oplus s \oplus i] \oplus T^0[i]$, for all $0 \leq i < N$.
4. **Output:** P_0 outputs (T^0, r); P_1 outputs (T^1, s).

Online Evaluation (same as OTTT in Prot. 2):

1. P_0 sends $u = x^0 \oplus r$ to P_1; P_1 sends $v = x^1 \oplus s$ to P_0.
2. P_0 sets $z^0 = T^0[u \oplus v]$; P_1 sets $z^1 = T^1[u \oplus v]$.
3. **Output:** P_0 outputs z^0; P_1 outputs z^1, s.t. $z^0 \oplus z^1 = T[x^0 \oplus x^1]$.

Reducing the Online Round Complexity. The SP-LUT protocol in §4.4 pre-computes $\binom{N}{1}$ OT in a setup phase and then uses these pre-computed values in the online phase to securely evaluate the function. In its vanilla version, the online phase consists of two rounds: 1) the receiver sends its updated choice bits to the sender and 2) the sender rotates its pre-computed masks and sends the updated correlations to the receiver. Thereby, we overall require $2D(C)$ communication rounds in the online phase, where $D(C)$ is the highest number of LUTs from any input to any output of the circuit.
In order to reduce the number of communication rounds, we let both parties switch roles in the online phase after each communication round, similar to [Hua12]. More specifically, assume P_0 plays the sender and P_1 plays the receiver in the first round. P_1 first sends its updated choice bits u_1 to P_0, who plays the receiver in the second round and replies with the updated correlations V_1 and the updated choice bits of the second round u_2. P_1 then updates its local shares using V_1, switches to the role of the sender and replies with its updated correlations V_2, and then again switches to the role of the receiver and sends its updated choice bits u_3 for the third communication round, etc. Overall, this reduces the number of communication rounds from $2D(C)$ to $D(C) + 1$.

Multi-Out LUTs. Note that in our LUT-based approach, we can efficiently combine two or more LUTs that have the same or even only some common inputs. Consider a functionality which has σ LUTs with one output bit each and the same δ input bits. When naively applying our approach, we would generate σ δ-input LUTs, one for each output bit. However, since we build on a $\binom{N}{1}$ OT protocol, we can amortize the cost for computing the OT protocol by sending σ output bits during the OT protocol. More specifically, instead of performing $\binom{N}{1}$ OT^σ_1, we would perform $\binom{N}{1}$ OT^1_σ, thereby saving $\sigma - 1$ executions of the OT protocol. This optimization naturally extends to an arbitrary number of output bits σ. Overall, for a functionality with δ input bits and σ output bits, we can thereby decrease the required communication from $\sigma(256 + 2^\delta)$ to $256 + \sigma 2^\delta$. In §7.1 we use this optimization to decrease the communication for the 8-input and 8-output AES S-box by a factor of 1.8 from 4,096 bit to 2,304 bit. Similarly, we can combine two or more LUTs which

PROTOCOL 4 (Setup-LUT (SP-LUT) - our work)

Inputs and Oracles:

- **Common Input:** Symmetric security parameter κ; number of inputs δ; $N = 2^\delta$; Truth-table $T : \{0,1\}^\delta \mapsto \{0,1\}^\sigma$.
- **Input of P_0:** $x^0 \in \{0,1\}^\delta$.
- **Input of P_1:** $x^1 \in \{0,1\}^\delta$.
- **Oracles:** Both parties have access to a $\binom{N}{1}$ R-OT$^1_\sigma$ functionality.

Pre-Computation:

1. P_0 and P_1 invoke the $\binom{N}{1}$ R-OT$^1_\sigma$ functionality where P_0 plays the sender and P_1 plays the receiver. From the OT, P_0 receives random bits $(m_0, ..., m_{N-1})$ and P_1 receives a random choice $s \in \{0,1\}^\delta$ and message m_s.
2. **Output:** P_0 outputs $(m_0, ..., m_{N-1})$; P_1 outputs (m_s, s).

Online Evaluation:

1. P_1 sends $u = s \oplus x^1$ to P_0.
2. P_0 chooses $z^0 \in_R \{0,1\}^\sigma$ and computes and sends $V = (v_0, ..., v_{N-1})$, where $v_i = T[i \oplus x^0] \oplus m_{i \oplus u} \oplus z^0$.
3. P_1 computes $z^1 = v_{x^1} \oplus m_s$.
4. **Output:** P_0 outputs z^0; P_1 outputs z^1, s.t. $z^0 \oplus z^1 = T[x^0 \oplus x^1]$.

share a sub-set of inputs. For instance, consider the case where one LUT has $\delta_1 = 3$ inputs x_0, x_1, x_2 and a second LUT has $\delta_2 = 4$ inputs x_0, x_1, x_3, x_4. In this case, we can combine both LUTs to one LUT with $\delta = 5$ inputs and thereby replace the $\binom{2^3}{1}$ OT$^1_1$ and $\binom{2^4}{1}$ OT$^1_1$ by a $\binom{2^5}{1}$ OT$^1_2$ which reduces communication from 488 bits to 312 bits.

5 LUT-based Circuit Synthesis

Hand-optimizing circuit representations for secure computation is a laborious and time-consuming task which leaves room for errors in the crafted circuit constructions. This only becomes more challenging for our LUT protocols where LUT-based circuit representations are required, instead of Boolean circuits with 2-input gates. Instead of reinventing the wheel and recreating compilers from scratch, it is much more intuitive to use existing hardware synthesis tools. This approach, which we also follow in our work, allows to automatically generate and optimize circuits even for complex functionalities that cannot be easily hand-optimized. As shown in TinyGarble [SHS+15] and its generalization to GMW [DDK+15], hardware synthesis tools are a key enabler for making secure computation protocols more practical by automating and speeding the process of generating compact and correct Boolean circuits and optimizing them for low size [SHS+15] and/or low depth [DDK+15] depending on the protocol used. In this work, we extend this approach further by exploiting LUT-based synthesis tools to serve the different requirements of our LUT protocols. However, such tools do not generate the LUT representations we require by default, and require heavy re-purposing to adapt them to our protocols. In the following, we briefly introduce hardware synthesis and afterwards discuss the particular synthesis tool we use and how we customize it for our purposes.

5.1 Hardware Synthesis Tools

Hardware synthesis is the process of transforming an abstract form of function description into a functionally equivalent logic implementation using different optimization and technology mapping algorithms, which

have been the subject of research in electronic design automation for decades. The circuit implementation generated usually depends on the target hardware platform and manufacturing technology. Common target hardware platforms include Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs). While ASIC synthesis tools have been the focus of previous works [SHS+15, DDK+15], since the protocols therein required circuits with 2-input gates, our work focuses on exploiting multi-input LUT-based synthesis tools which form the core of FPGA-based synthesis software. ASIC synthesis tools can also map to multi-input gates, given that the gates are defined in custom libraries. However, this is tedious, impractical, and would require very large libraries to accommodate all possible LUTs for each input size. Hence, we opt to use LUT-based synthesis tools instead.

There exists a spectrum of commercial FPGA synthesis tools such as Synplify by Synopsys [Syn], Quartus by Altera [Alt], XST [Xil09] and Vivado Synthesis [Xil] tools by Xilinx. However, these tools synthesize LUT-based circuits that target their devices' specifics such as the number of physically possible inputs to an LUT (a maximum of 6-input LUTs for most current FPGA devices). For our protocol, we aim to generate up to 8-input LUTs and this, to the best of our knowledge, is not available in mainstream commercial tools. Mapping circuits to variable-input LUTs is, however, being investigated by the Berkeley Logic Synthesis and Verification Group who develop ABC [Ber], a growing open-source software for synthesis and verification of binary logic circuits. ABC provides an experimental implementation of different mapping and optimization algorithms based on optimal-delay Directed Acyclic Graph (DAG)-based technology mapping for both standard gates and LUTs. In this work, we leverage the mapping of ABC, coupled with Yosys [Wol]. We use Yosys as an open-source framework for front-end processing of our Verilog circuit descriptions to map them into a network of low-level logic operations in an intermediate format. Then, ABC structures this network into a DAG and maps it into LUTs in a delay-optimized fashion.

However, for generating circuit netlists of more complex functionalities, such as floating-point operations, we utilize built-in Intellectual Property (IP) libraries in the Synopsys Design Compiler (DC) [Syn10], a commercial ASIC synthesis tool. Synopsys DC generates Boolean netlists of these circuits, which we further process with the Yosys-ABC toolchain to re-map them to LUT-based representations.
In the following, we focus on the Yosys-ABC toolchain and our customizations to tailor its output to the requirements of our LUT protocols.

# Inputs δ	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
N-**MT** [bits]	138	276	414	552	690	828	966	1,104	1,242	1,518
OP-LUT [bits]	210	**291**	500	1,277	4,354	$\approx 2^{14}$	$\approx 2^{16}$	$\approx 2^{18}$	$\approx 2^{20}$	$\approx 2^{22}$
OP-LUT / N-**MT**	0.66	**0.95**	0.85	0.43	0.16	0.05	0.01	< 0.01	< 0.01	< 0.01
SP-LUT [bits]	196	232	256	280	326	**372**	511	768	1,288	2,316
SP-LUT / N-**MT**	0.70	1.19	1.62	1.97	2.12	**2.23**	1.89	1.44	0.96	0.66

Table 5: Communication of OP-LUT (§4.2) with $\binom{N}{1}$ OT and SP-LUT (§4.4) compared to a Boolean circuit evaluated with N-MT for a δ-input to $\sigma = 1$ output bit function with $\delta - 1$ AND gates and 138 bit communication per AND gate (§3). The results for OP-LUT and SP-LUT that achieve the best performance compared to N-MT are marked in **bold**.

5.2 Customizing LUT-based Synthesis

ABC is very fitting for our purposes because it maps circuits to variable δ-input LUTs in a generic manner and allows the user to determine the maximum input size $\delta_{\max}$ allowed, regardless of any target-specific FPGA architecture details. The Yosys-ABC toolchain works by structuring the Boolean circuit network into a specific type of Directed Acyclic Graph (DAG) consisting of 2-input, 1-output nodes, and then maps this graph into δ-input LUTs by computing δ-feasible cuts for each graph node. A cut of a node n is a set of nodes (called leaves of the respective cut), such that each path from the circuit primary inputs to node n passes through at least one of these leaves. A cut is δ-feasible if the number of leaves in it does not exceed δ. FPGA mapping either enumerates all or some selected cuts of each node according to the optimization metric. Then, depth-optimized mapping is performed to select the optimal cuts, followed by area recovery heuristics, after which the cuts are mapped to LUTs according to their sizes. Additional details on the DAG-based delay-optimized technology mapping using δ-feasible cuts can be found in [RME+12, MCCB07, MCB07].

For the generation of our netlists, we limit the maximum number of LUT inputs to $\delta_{\max} = 4$ for OP-LUT and $\delta_{\max} = 8$ for SP-LUT, since it provides a good performance trade-off as we describe later in §6.1. We optimize for depth, followed by area recovery, and ensure that the circuits remain topologically ordered.

5.3 Generating Multi-Output LUTs

Extending hardware synthesis tools beyond their original purposes and tailoring their output to serve the purposes of secure computation requires radical engineering and customizations. As discussed in §4.5, our LUT protocols are significantly optimized by combining LUTs with overlapping inputs and hence multiple output bits. However, ABC does not support mapping to multi-output LUTs by default (and neither do commercial hardware synthesis tools except for 2-output LUTs). This remains largely an open and unsolved research area, without efficient tools. Some research efforts such as the work in [MMRR10] propose $\delta\sigma$-feasible cuts mapping to control both the number of inputs δ and the number of outputs σ in mapping circuit cuts. However, their implementation is not available and their approach focuses on contributing to AIG-based mapping algorithms in general and is not specifically focused on mapping to multi-output LUTs. We handle this by post-processing the ABC-generated single-output LUT circuits to map them to multi-output LUT circuits. As already mentioned, we map circuit descriptions to variable-input LUT-based netlists with an allowed maximum of 4 or 8 inputs per LUT using ABC. The generated circuits are then post-processed and layered from input to output according to the input-output dependencies. Each LUT is allocated to its layer according to its topological depth in the circuit. LUTs within the same circuit layer which share one or more common inputs are grouped together into a single multi-output LUT incrementally. Each final multi-output LUT is defined by a set of a maximum of 4 or 8 inputs, and the number of grouped LUTs, their truth-table values, and the subset of inputs on which the output of the included LUT depends. In a second optimization round, LUTs which have no shared inputs but can be combined together while still having a union of a maximum of 4 or 8 inputs are grouped together. It is important to only group LUTs within the same layer to avoid grouping across layers that may increase the circuit depth.
Furthermore, ABC maps circuit descriptions into LUTs only, whereas our protocols allow function representations with both LUTs and XORs. In the post-processing, we map 2-input LUTs that represent the XOR function into explicit XOR gates. Extracting all XORs to reduce the overall number and inputs sizes to LUTs is an interesting direction for future research.

6 Evaluation

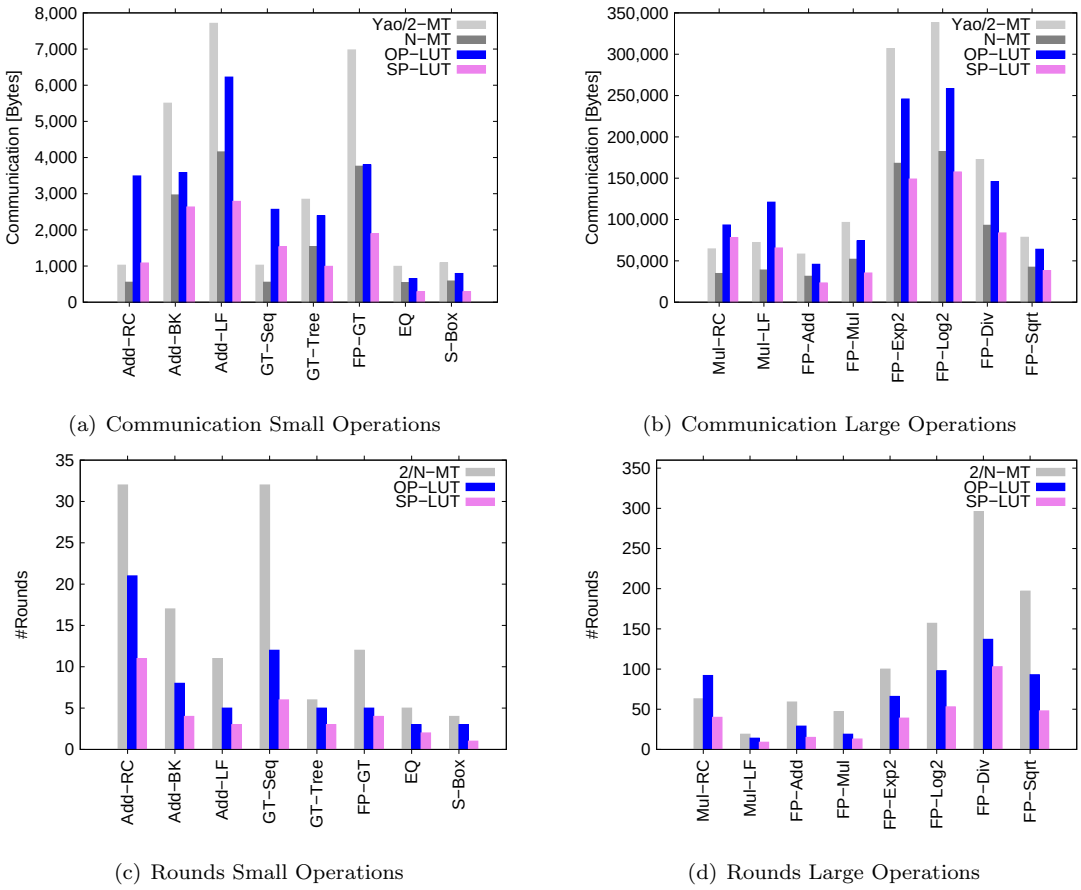
In this section we theoretically compare the performance of our LUT-based approaches to Boolean circuits (§6.1). Since it is not possible to give generic statements about the efficiency comparison between our LUT protocols and Boolean circuits, we then give an empirical performance comparison on typical basic operations (§6.2). All protocols are evaluated for 128-bit symmetric security, i.e., $\kappa = 128$.

6.1 Comparison to Boolean Circuits

In the following, we theoretically compare our LUT representation with a 2-input Boolean circuits representation. We first discuss the advantages of finding an efficient function representation as interconnected LUTs compared to a Boolean circuit. Then, we compare the communication and round complexity of a single δ-input LUT with $\sigma = 1$ output bit to a Boolean circuit equivalent. Finally, we discuss both representations when realizing functionalities with $\sigma > 1$ output bits. We stress that, even though we discuss and compare them separately, our LUT protocols can be easily combined with Boolean circuits using GMW at no cost, achieving the best of both worlds.

Efficient Function Representations. Finding an efficient Boolean circuit representation with low number of AND gates and small multiplicative depth has been subject to extensive research. E.g., [BP05] have shown a lower bound on the number of AND gates for the Hamming weight functionality and [ARS+15] have developed a block-cipher with a small number of AND gates and a small AND depth. Such efficient

Boolean representations, however, are non-trivial to identify for more complex functions. Representing the function as a LUT would reduce the complexity of finding an efficient representation to some extent, since the costs for securely evaluating a LUT only depend on the number of inputs and outputs and not on its internal functionality. Hence, the optimization process can be stopped after the functionality has been separated into connected LUTs and does not need to identify an efficient representation of the functionality as it is the case for Boolean circuits. As an example, consider the AES S-box, which has 8 input bits and 8 output bits. While [BP12] have used a special Greedy-approach to identify a small Boolean circuit, a LUT representation could be obtained by simply evaluating the S-box on all 2^8 possible inputs.



(a) Communication Small Operations

(b) Communication Large Operations

(c) Rounds Small Operations

(d) Rounds Large Operations

Figure 3: Total Communication (a,b) using Yao's garbled circuits (§2.4) and 2-MT (§2.5), our N-MT (§3), our OP-LUT ($\delta \leq 4$ inputs, cf. §4.3) and our SP-LUT ($\delta \leq 8$ inputs, cf. §4.4) and round complexity in the online phase (c,d) for a Boolean circuits evaluation using GMW (MT), OP-LUT, and SP-LUT for 32-bit operations and the 8-bit AES S-box. Yao's round complexity is constant and therefore not included.

Single-Output Functionalities. The communication complexity of a Boolean circuit component with δ inputs depends on the number of AND gates in its function representation, which we bounded by $\delta - 1$ (cf. §2.6). The communication complexity of a δ-input LUT, on the other hand, only depends on δ. Building on these observations, we outline the best achievable communication ratio for a δ-input functionality of our OP-LUT and SP-LUT protocols to a Boolean circuit, evaluated using N-MT, in Tab. 5. We observe that the best communication ratio for OP-LUT is factor 0.95 for $\delta = 3$ and for SP-LUT is factor 2.2 for $\delta = 7$. Hence, we limit the possible LUT sizes for OP-LUT to $\delta \in [2, 4]$ and for SP-LUT to $\delta \in [2, 8]$. Note, however, that using LUTs with more inputs can result in better overall performance due to improved round complexity.

The round complexity when evaluating a Boolean circuit using GMW depends on the AND depth, which we bounded by $\log_2 \delta$ (cf. §2.6). A δ-input LUT, on the other hand, always requires one communication round, independently of δ (plus one global communication round for the whole circuit with SP-LUT). Hence, for basic operations, we expect a significant decrease in rounds by factor $\log_2 \delta$.

Multi-Output Functionalities. For functionalities with multiple outputs, we assume that an optimal circuit is constructed for each output bit separately, resulting in a Boolean circuit with $\sigma(\delta - 1)$ AND gates (cf. §2.6). However, many functions can be optimized and computed more efficiently. In contrast, our LUT protocols can easily be extended to handle functionalities with multiple outputs without requiring an additional logic optimization step (cf. §4.5) but at the cost of at least $\sigma 2^\delta$ bits communication, which cannot be reduced via logic optimization. Hence, a Boolean representation can achieve better communication for multi-output bit functionalities where the number of AND gates can be highly optimized (e.g., ripple-carry addition), while our LUT representation achieves better communication for functionalities with many AND gates per input and output bits (e.g., the AES S-box). Nevertheless, our LUT representation needs fewer communication rounds, independently of the number of outputs.

Network	**LAN**										**WAN**									
AES Encryption																				
# Blocks	**1**					**1,000**					**1**					**1,000**				
Protocol	Yao	2-MT	*N*-MT	OP-LUT	SP-LUT	Yao	2-MT	*N*-MT	OP-LUT	SP-LUT	Yao	2-MT	*N*-MT	OP-LUT	SP-LUT	Yao	2-MT	*N*-MT	OP-LUT	SP-LUT
Setup [s]	0.007	**0.003**	0.004	**0.003**	**0.003**	1.395	0.822	**0.688**	0.781	0.970	0.300	0.423	0.396	0.502	**0.180**	50.758	25.552	13.719	19.315	**2.699**
Online [s]	**0.003**	0.006		0.005	0.006	0.137	0.028		**0.024**	0.453	**0.228**	2.397		1.642	0.790	2.234	2.808		**2.102**	11.080
Total [s]	0.010	0.009	0.010	**0.008**	0.009	1.561	0.850	**0.720**	0.805	1.419	**0.528**	2.823	2.793	2144	0.970	52.992	28.360	16.526	21.417	**13.779**
Sent [MB]	0.194	0.184	0.140	0.127	**0.055**	172	169	96	103	**44**	0.194	0.184	0.140	0.127	**0.055**	172	169	96	103	**44**
Private Set Intersection																				
Set Sizes	**256**					**16,384**					**256**					**16,384**				
Protocol	Yao	2-MT	*N*-MT	OP-LUT	SP-LUT	Yao	2-MT	*N*-MT	OP-LUT	SP-LUT	Yao	2-MT	*N*-MT	OP-LUT	SP-LUT	Yao	2-MT	*N*-MT	OP-LUT	SP-LUT
Setup [s]	0.113	0.069	**0.057**	0.062	0.267	3.180	2.117	**1.784**	1.819	5.878	2.414	1.157	1.069	1.237	**0.901**	61.834	31.347	16.533	18.730	**9.857**
Online [s]	0.026	**0.003**		0.004	0.022	1.227	**0.079**		0.132	0.781	0.802	0.457		**0.348**	0.693	36.750	1.867		**1.742**	4.789
Total [s]	0.139	0.072	**0.060**	0.066	0.310	4.407	2.195	**1.862**	1.951	6.659	3.217	1.705	**1.526**	1.585	1.594	98.584	33.214	18.400	20.472	**14.089**
Sent [MB]	6.923	4.320	2.475	2.971	**1.247**	339.2	209.4	119.6	144.0	**58.6**	6.923	4.320	2.475	2.971	**1.247**	339.2	209.4	119.6	144.0	**58.6**

Table 6: Summary of our application results on AES and PSI for Yao's garbled circuits (§2.4), 2-MT (§2.5) and our N-MT (§3.5) for GMW, and our OP-LUT (§4.3) and SP-LUT (§4.4) protocols. Best results marked in **bold.**

6.2 Basic Operations

A general comparison between our LUT protocols and 2-input gate Boolean circuit-based techniques is difficult to perform, since the performance of both is very function-dependent. To highlight the improvements, we compare the efficiency of several basic operations: addition (ripple-carry Add-RC, Brent-Kung Add-BK, and Ladner-Fischer Add-LF) [SZ13], multiplication (ripple-carry Mul-RC and Ladner-Fischer Mul-LF) [SZ13], equality (EQ), greater-than (sequential GT-Seq and tree-based GT-Tree) [SZ13], floating point operations [DDK+15], and the AES S-Box [BP12]. For each functionality, we give the total communication (setup + online) in bytes and the online round complexity (the setup round complexity is constant). We compare Yao's garbled circuits (256 bits per AND gate, cf. [ZRE15]) and the 2-MT multiplication triple generation (260 bits per AND gate, cf. [ALSZ13], decreased to 256 to match Yao's communication), the N-MT triple generation (138 bits per AND gate, cf. §3.5), our OP-LUT protocol (using $\delta \in [2, 4]$ input LUTs, cf. §4.3) and our SP-LUT approach (using $\delta \in [2, 8]$ input LUTs, cf. §4.4). Note that for SP-LUT we omit the extra round that is added due to the role-switching optimization (cf. §4.5), since it amortizes over the whole protocol execution. Also, we omit Yao's garbled circuits in the round complexity comparison, since it has constant rounds for every functionality. We generate the LUT representations of the basic operations using optimized circuit descriptions fed into our automated toolchain (cf. §5). We present the results for 32-bit operations in Fig. 3.

From the results we can observe that our SP-LUT protocol nearly always has the lowest communication, achieving up to factor 2 less communication than the N-MT generation, which is the next best. Our OP-LUT protocol always performs worse than the N-MT generation but most of the times achieves lower communication than Yao's garbled circuits and the regular 2-MT generation. The only operations where our LUT protocols perform worse than the Boolean circuit-based protocols are the ripple-carry adder (Add-

RC), the multiplication circuits (Mul-RC and Mul-LF), and the sequential greater-than (GT-Seq), where our SP-LUT approach performs similar to Yao and 2-MT. As discussed in §6.1, this is probably due to the low multiplicative complexity of the ripple-carry addition as well as the high number of outputs per LUT. Also notably, our LUT protocols require less communication for the tree-based greater-than (GT-Tree) than for the sequential greater-than (GT-Seq), even though the GT-Tree circuit has around three times more ANDs than the GT-Seq circuit. Hence, building on certain circuit structures results in more efficient LUT circuits and there is still potential for further optimizations.

Regarding the round complexity, we emphasize that our LUT approaches are almost always better than 2-input gate Boolean circuits, except for the ripple-carry adder (Add-RC) evaluated with OP-LUT. On average, OP-LUT reduces the number of communication rounds by factor 2x while SP-LUT even reduces them much further by factor 3-4x.

7 Applications

In this section we evaluate the concrete benefits of our LUT protocols on two practical examples: privacy-preserving AES (§7.1) and private set intersection (§7.2). We compare our OP-LUT and SP-LUT protocols to a Boolean 2-input gate circuit, evaluated using Yao's garbled circuits and GMW using the 2-MT and N-MT pre-computation in a LAN and WAN setting and summarize our results in Tab. 6.

Benchmark environment. We implement our OP-LUT with $\binom{N}{1}$ OT pre-computation and SP-LUT protocols in the ABY framework of [DSZ15], written in C++. We benchmark the protocols in two settings: a *LAN* setting, consisting of two Intel i7 Haswell PCs connected by a Gigabit network, and a *WAN* setting, consisting of a Google n1-standard-4 instance with 4 vCPUs and an Amazon m3Xlarge instance with 4 vCPUs which are connected by a network with 28 MBit bandwidth and 122 ms ping latency on average. We argue that the WAN setting presents a practical MPC setting, since the machines are controlled by two different cloud providers and located at two different continents in the US and Europe. We run the experiments using 4 threads on each machine, average the results over 10 executions, and dismiss outliers with more than twice the runtime. For Yao's garbled circuits, we perform multi-threading by splitting the original circuit into four separate parts that are evaluated in parallel. The variance in the LAN setting was $\approx 1\%$ and in the WAN setting $\approx 5\%$.

Implementation features. Our LUT protocols work in the pre-processing model, where setup and online phase are executed separately. Both phases can be combined in case of an ad-hoc execution, resulting in a lower total time. To process a shared value, our LUT protocols need to read, process, and store a table entry, in contrast to Boolean circuit-based protocols, which can process multiple shares at once. Thereby, the amortization that happens when the same circuit is evaluated a large number of times in parallel is less compared to a Boolean circuit-based evaluation. Finally, our LUT protocols pre-compute and store tables, which results in a larger memory footprint compared to GMW, which only stores single bits. However, the storage requirement is still much lower than for pre-computed Yao's garbled circuits and the table generation and evaluation can be pipelined, similar to garbled circuits [HEKM11].

7.1 AES Encryption

One of the most widely used benchmark examples for secure computation is AES, which has applications in encrypted password authentication [Sec15]. We assume that a client holds either one or 1,000 plaintexts, which should be encrypted with an expanded key, held by a server. We use the Boolean AES S-box circuit from [BP12] which has 34 AND gates and a multiplicative depth of 4. The OP-LUT representation of the S-box consists of a network of $\delta = 2$ to $\sigma = 1$, $\delta = 3$ to $\sigma = 2$, and $\delta = 4$ to $\sigma = 4$ LUTs which requires 795 bytes of communication and has 3 communication rounds. The SP-LUT representation of the AES S-box uses a $\delta = 8$ input to $\sigma = 8$ output LUT to evaluate an S-box.

From the results in Tab. 6, we can observe that no protocol consistently performs best across all experiments. This can be explained by a varying bottleneck, depending on the evaluated function and the setting.

In the AES(1,000) LAN setting, the N-MT protocol performs best since it has a good balance between computation and communication. In the AES(1) WAN setting, Yao's protocol performs best, since it has the lowest number of communication rounds. Finally, in the AES(1,000) WAN setting, the SP-LUT protocol performs best, since it has the lowest communication. The 2-MT and N-MT approaches have the same online phase but the setup phase of the N-MT protocol is more efficient due to the lower communication. For AES(1000), Yao's protocol performs worst since its communication is uni-directional from garbler to the evaluator as opposed to the other protocols, where the communication is evenly divided between both parties.

7.2 Private Set Intersection

In the private-set intersection (PSI) application, two parties want to identify the intersection of their private sets without revealing any element that is not in the intersection. PSI can be used for computing the revenue of online advertisement, for finding common contacts, or for genomic computations [PSSZ15]. For our experiments, we use the circuit-based PSI protocol of [PSSZ15], which computes the intersection between two sets by first mapping the elements into hash tables and then performing a pair-wise comparison between each bin of the hash tables. We compute the intersection between two sets of either 256 or 16,384 elements with length 32-bit. The Boolean circuit for sets of 256 elements has 138,600 AND gates and for sets of 16,384 elements has 6,724,062 AND gates and both have a multiplicative depth of 5. The OP-LUT circuit for 256 elements has 44,352 $\delta = 4$ to $\sigma = 1$ LUTs and 5,544 $\delta = 2$ to $\sigma = 1$ LUTs and the circuit for 16,384 elements has 2,123,388 $\delta = 4$ to $\sigma = 1$ LUTs and 353,898 $\delta = 2$ to $\sigma = 1$ LUTs and both have a depth of 4. The SP-LUT circuit for 256 elements has 16,632 $\delta = 8$ to $\sigma = 1$ and 5,544 $\delta = 5$ to $\sigma = 1$ LUTs and the circuit for 16,384 elements has 707,796 $\delta = 8$ to $\sigma = 1$ and 353,898 $\delta = 6$ to $\sigma = 1$ LUTs and both have a depth of 3.

As shown in Tab. 6, the overall best performing protocol for the PSI experiments in the LAN and PSI(256) WAN settings is our N-MT generation. In the PSI(256) WAN setting, our LUT protocols are only slightly slower, while in the PSI(16,384) setting, our SP-LUT protocol achieves the best performance. The reason for the poor performance of SP-LUT in the LAN setting is its high computation overhead, which is due to the high number of $\delta = 8$ to $\sigma = 1$ LUTs, which is around factor 5 higher than for the N-MT protocol. Nevertheless, in the WAN setting, where communication becomes the bottleneck, the computation overhead of our SP-LUT protocol amortizes and it performs better than the standard 2-MT generation. Yao's garbled circuits protocol performs poorly in the LAN and WAN settings, since it has larger communication per AND gate than the other protocols and the number of input wires to the circuit, which require κ bit communication in the online phase, is as high as the number of AND gates. Finally, the Boolean circuits protocols have a fast online time in the LAN setting, since the number of communication rounds is low, but OP-LUT achieves better online time in the WAN setting, due to the higher latency.

8 Related Work

In this section, we discuss related work on improving secure computation (§8.1), secure computation protocols that represent the functionality as network of multi-input gates (§8.2), and Boolean circuit compilers (§8.3).

8.1 Efficient Secure Computation

One of the main reasons why secure computation was believed to be inefficient was the high number of symmetric cryptographic primitive invocations. In particular, in Yao's garbled circuits the circuit garbler requires 4 invocations to garble an AND gate while for GMW, both parties require 6 invocations to generate a multiplication triple during OT extension. A dramatic improvement on the computation efficiency of secure computation protocols has come with the introduction of the AES-NI processor extensions [KSS12, BHKR13, GLNP15]. Currently, the most efficient instantiation is the fixed-key AES garbling of [BHKR13], which imposes an ideal permutation assumption on AES. Alternative instantiations that require weaker

assumptions and use pipelining techniques to improve efficiency have been given in [GLNP15]. In [ZRE15] it was shown how to reduce the communication in Yao's garbled circuits to 2κ bits per AND gate and it was proven that this matches the lower bound. In [KKS16] the authors utilize the fact that AND gates in Yao's garbled circuits where one party knows the plaintext input can be garbled at lower cost to reduce the communication for specific circuits.
One approach to circumvent the high cost for certain operations are *mixed-protocols*, which mix secure computation protocols that operate on arithmetic and Boolean circuits. Thereby, a function can be divided in sub-blocks that are evaluated in the secure computation protocol for which the representation is more efficient. The TASTY framework [HKS+10] combined additively homomorphic encryption and Yao's garbled circuits protocol. The ABY framework [DSZ15] used OT instead of homomorphic encryption to compute the multiplication. Our work can be combined with these approaches to achieve another degree of freedom when constructing mixed-protocols.

8.2 Multi-Input Gates in Secure Computation

The gate-evaluation secret sharing approach (GESS) [Kol05] is an information theoretic variant of garbled circuits that can be based on OT and performs secure computation in a constant number of rounds. The idea of GESS is to process the circuit from the outputs to the inputs such that shares on the output wires determine the shares on the input wires, which leads to a quadratic size increase in circuit depth for shares on the input wires. Sliced-GESS [KK12] efficiently extends GESS to circuits with higher depth at the cost of an increased number of communication rounds by slicing the circuit into sub-circuits of constant depth, which are connected via a string selection OT (security against a covert client can be achieved using longer strings as selection bits in the OTs). The efficiency of sliced-GESS is not experimentally evaluated but according to an unpublished full version[2], the performance for a rectangular circuit is $(112 + \kappa)/3 = 80$-bit per gate, where each slice has depth $d' = 3$, $\kappa = 128$ and with the OT extension optimization of [KK13, ALSZ13], which reduces the number of sent ciphertexts from two to one. In contrast to sliced-GESS, our SP-LUT approach can achieve less communication for some functionalities, e.g., as little as half a ciphertext (62 bit) per AND gate when computing the equality between two 7 bit values (cf. Tab. 5 where the equality circuit has 6 AND gates).
[IKM+13] outlines a scheme called *one-time truth tables (OTTTs)*, which relies on representing the whole function as a single truth-table and allows the evaluation of an arbitrary-size truth-table in a constant number of rounds and with linear communication complexity in the input length during the online phase using correlated randomness that is pre-computed in the setup phase. However, the scheme scales poorly for functions with large input size as the setup phase requires super-polynomial communication and storage in the length of the function's input. [DZ16] tailors the pre-computed randomness to AES S-boxes to allow an efficient online evaluation of AES with security against malicious adversaries. Their setup phase, however, becomes very communication intensive, since all possible outcomes for every AES S-box have to be pre-computed once. We present and analyze the efficiency of [IKM+13] with pre-computation using [DZ16] in §4.2 and give a protocol that improves on the communication complexity in the setup phase for practical input sizes in §4.3.
FastGC [HEKM11] and its memory-efficient optimization of [HS13] used Yao's protocol to evaluate multi-input gates. Using the garbled row reduction technique [NPS99], this approach requires $\kappa\sigma \cdot (2^\delta - 1)$ bits communication in the setup phase for a LUT with δ input and σ output bits and, in the online phase, requires constant rounds and no communication. However, a traditional 2-input gate Boolean circuit evaluation using Yao's protocol is more efficient than a multi-input gate evaluation, since the communication for multi-input garbled tables scales exponentially in δ.
A concurrent and independent work introduces TinyTable [DNNR16], a malicious secure computation protocol that uses pre-computed tables for secure evaluation of functions. TinyTable was shown to achieve better online communication for 2-input AND gates in the semi-honest model. For tables with more inputs, its online phase was evaluated only on the AES SBox. In order to pre-compute the tables, [DNNR16] uses the

[2] Available at http://www.cs.technion.ac.il/~ranjit/papers/slicegess.pdf.

same idea as [DZ16], outlined in §4.2, namely to pre-compute the circuit once for every input combination, which results in a large communication overhead in the setup phase.
A recent work [GLMY16] proposes to garble a circuit as independent smaller sub-components, which reduces the communication cost in the online phase but results in a multi-round protocol and more overall communication.
An ongoing and independent work outlines a polynomial-based garbling scheme in Yao's protocol [MPS15]. The scheme requires the function to be represented as building blocks with multiple inputs instead of 2-input gates. We view their work as orthogonal to ours, since they focus on the constant-round Yao's protocol while our approach focuses on multi-round secret-sharing based protocols.
New garbling techniques that allow Yao's garbled circuits protocol to evaluate several functions more efficiently than a regular linear garbling scheme were given in [BMR16]. In particular, the authors showed that their garbling techniques allows a more efficient evaluation of arithmetic circuits and multi-input threshold gates in Boolean circuits. For general Boolean circuit constructions, the authors give a construction that is of theoretical interest since it circumvents the 2κ lower bound of [ZRE15] when evaluating a single AND gate but does not generalize to arbitrary circuits.
Our techniques for evaluating d-input gates via 1-out-of-2^d OT on bitstrings is a very natural generalization of the original GMW construction for evaluating 2-input gates using 1-out-of-4 OT as described in [Gol04, Sect. 7.3.3]. The main new observation is that using the 1-out-of-N OT extension protocol of [KK13] (and the additional optimizations we propose in our work), this can be done in a highly efficient manner. This observation was also made and used in the concurrent and independent work of [KKW17], where the authors showed how to efficiently overlay a large number of switch branches in secure computation and propose to evaluate switch statements in GMW using the 1-out-of-N OT extension protocol of [KK13]. They also stress that their scheme can be adapted to efficiently evaluate multi-input gates in the GMW protocol.

8.3 Boolean Circuit Compilers

Circuit compilers abstract the complexity of designing Boolean circuits by compiling a high level language (such as Java or C or Verilog) into a Boolean circuit. CBMC-GC [HFKV12] uses a model checker to generate a Boolean circuit from a description in C. The portable circuit format (PCF) [KSMB13] compiles high level code into an assembler-like representation. The programming framework ObliVM [LWN+15] introduces a special purpose language which is compiled into a memory-trace oblivious program based on Boolean circuits and ORAM. TinyGarble [SHS+15, DDK+15] presents a radically different approach of leveraging already established hardware synthesis tools within a fully automated toolchain to compile a circuit description in a hardware description language such as VHDL or Verilog into a Boolean circuit. In our work, we go beyond the TinyGarble approach and utilize and re-purpose LUT-based synthesis tools which are used to map circuit implementations on Field Programmable Gate Arrays (FPGAs) to generate LUT representations for a wide range of functions.

9 Conclusion and Future Work

The current bottleneck in most implementations of semi-honest secure two-party computation protocols are the network bandwidth and latency. In this work, we show how to significantly reduce the communication as well as the round complexity at the cost of increased computation. For secure computation on Boolean circuits, we reduce the communication from 2κ-bit to nearly a single κ-bit ciphertext per AND gate. Furthermore, we outline how to significantly improve round complexity and the communication complexity by representing the functionality as a network of lookup tables (LUTs). We introduce two protocols, OP-LUT and SP-LUT, for evaluating LUTs and a compiler that leverages a hardware synthesis tool that we customize to automatically translate functions from a high-level description to a LUT representation. Finally, we show that our SP-LUT protocol achieves a remarkable 3-4x better round complexity and reduces the communication beyond even the one κ-bit per AND gate boundary that we achieve by evaluating a Boolean circuit using GMW even with our improved pre-computation for many basic operations. In addition, our

LUT protocols can be freely combined with a Boolean circuit evaluation using GMW, incurring no additional costs and achieving the benefits of both representations.

We see multiple interesting research questions for future work: A) Is it possible to improve the LUT compiler? Even though the hardware synthesis tools already generated optimized circuits with good performance, we see potential in extracting XOR gates from LUTs, since XORs can be evaluated for free. This would result in LUTs with less inputs and outputs. B) Is it possible to combine the efficient setup phase of our SP-LUT approach with the efficient online phase of our OP-LUT approach and thereby obtain a protocol that achieves both, an efficient setup as well as an efficient online phase? C) Do our protocols extend to stronger adversaries?

Acknowledgments

This work has been partially funded by the European Union's 7th Framework Program (FP7/2007-2013) under grant agreement n.609611 (PRACTICE), by the German Federal Ministry of Education and Research (BMBF) within CRISP, by the DFG as part of projects S5 and E4 within the CRC 1119 CROSSING. This work is in parts supported by NSF awards 1619261 and 1649423 and AFOSR/MURI FA9550-14-1-0351.

References

[ALSZ13] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *CCS'13*, pages 535–548. ACM, 2013.

[Alt] Altera Inc. Quartus prime design software. https://www.altera.com/products/design-software/fpga-design/quartus-prime/overview.html.

[ARS+15] M. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In *EUROCRYPT'15*, volume 9056 of *LNCS*, pages 430–454. Springer, 2015.

[BB94] M. L. Bonet and S. R. Buss. Size-depth tradeoffs for Boolean fomulae. *Information Processing Letters*, 49(3):151–155, 1994.

[BCD+09] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. P. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. I. Schwartzbach, and T. Toft. Secure multiparty computation goes live. In *FC'09*, volume 5628 of *LNCS*, pages 325–343. Springer, 2009.

[Ber] Berkeley Logic Synthesis. ABC: a system for sequential synthesis and verification, release 70930. http://www.eecs.berkeley.edu/~alanmi/abc/.

[BHKR13] M. Bellare, V. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *S&P'13*, pages 478–492. IEEE, 2013.

[BJSV15] D. Bogdanov, M. Jõemets, S. Siim, and M. Vaht. How the Estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party computation. In *FC'15*, volume 8975 of *LNCS*, pages 227–234. Springer, 2015.

[BMR16] M. Ball, T. Malkin, and M. Rosulek. Garbling gadgets for boolean and arithmetic circuits. In *CCS'16*, pages 565–577. ACM, 2016.

[BP05] J. Boyar and R. Peralta. The exact multiplicative complexity of the Hamming weight function. *Electronic Colloquium on Computational Complexity (ECCC'05)*, TR05(049), 2005.

[BP12] J. Boyar and R. Peralta. A small depth-16 circuit for the AES S-box. In *Information Security and Privacy Research (SEC'12)*, volume 376 of *IFIP Advances in Information and Communication Technology*, pages 287–298. Springer, 2012.

[CO15] T. Chou and C. Orlandi. The simplest protocol for oblivious transfer. In *Progress in Cryptology – LATINCRYPT'15*, volume 9230 of *LNCS*, pages 40–58. Springer, 2015.

[DDK+15] D. Demmler, G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, and S. Zeitouni. Automated synthesis of optimized circuits for secure computation. In *CCS'15*, pages 1504–1517. ACM, 2015.

[DKS+17] Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. Pushing the communication barrier in secure computation using lookup tables. In *24. Annual Network and Distributed System Security Symposium (NDSS'17)*. The Internet Society, 2017.

[DNNR16] I. Damgård, J. B. Nielsen, M. Nielsen, and S. Ranellucci. Gate-scrambling revisited - or: The TinyTable protocol for 2-party secure computation. Cryptology ePrint Archive, Report 2016/695, 2016.

[DSZ15] D. Demmler, T. Schneider, and M. Zohner. ABY - a framework for efficient mixed-protocol secure two-party computation. In *NDSS'15*. The Internet Society, 2015.

[DZ16] I. Damgård and R. W. Zakarias. Fast oblivious AES: A dedicated application of the MiniMac protocol. In *AFRICACRYPT'16*, volume 9646 of *LNCS*, pages 245–264. Springer, 2016.

[FJJBT16] T. Kasper Frederiksen, T. P. Jakobsen, Nielsen J. B, and R. Trifiletti. On the complexity of additively homomorphic UC commitments. In *TCC'16*, volume 9562 of *LNCS*, pages 542–565. Springer, 2016.

[GLMY16] A. Groce, A. Ledger, A. J. Malozemoff, and A. Yerukhimovich. CompGC: Efficient offline/online semi-honest two-party computation. Cryptology ePrint Archive, Report 2016/458, 2016.

[GLNP15] S. Gueron, Y. Lindell, A. Nof, and B. Pinkas. Fast garbling of circuits under standard assumptions. In *CCS'15*, pages 567–578. ACM, 2015.

[GM16] S. Gueron and N. Mouha. Simpira v2: A family of efficient permutations using the AES round function. Cryptology ePrint Archive, Report 2016/122, 2016.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC'87*, pages 218–229. ACM, 1987.

[Gol04] O. Goldreich. *Foundations of Cryptography*, volume 2: Basic Applications. Cambridge University Press, 2004.

[HEKM11] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security'11*, pages 539–554. USENIX, 2011.

[HFKV12] A. Holzer, M. Franz, S. Katzenbeisser, and H. Veith. Secure two-party computations in ANSI C. In *CCS'12*, pages 772–783. ACM, 2012.

[HKS+10] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for Automating Secure Two-partY computations. In *CCS'10*, pages 451–462. ACM, 2010.

[HS13] W. Henecka and T. Schneider. Faster secure two-party computation with less memory. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS'13)*, pages 437–446. ACM, 2013.

[Hua12] Y. Huang. Practical secure two-party computation. Ph.D. Thesis, 2012. Online: https://yhuangpress.files.wordpress.com/2014/02/dissertation.pdf.

[IKM+13] Y. Ishai, E. Kushilevitz, S. Meldgaard, C. Orlandi, and A. Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In *TCC'13*, volume 7785 of *LNCS*, pages 600–620. Springer, 2013.

[IKNP03] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO'03*, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.

[IR89] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *STOC'89*, pages 44–61. ACM, 1989.

[KK12] V. Kolesnikov and R. Kumaresan. Improved secure two-party computation via information-theoretic garbled circuits. In *SCN'12*, volume 7485 of *LNCS*, pages 205–221. Springer, 2012.

[KK13] V. Kolesnikov and R. Kumaresan. Improved OT extension for transferring short secrets. In *CRYPTO'13*, volume 8043 of *LNCS*, pages 54–70. Springer, 2013.

[KKS16] C. Kempka, R. Kikuchi, and K. Suzuki. How to circumvent the two-ciphertext lower bound for linear garbling schemes. In *ASIACRYPT'16*, volume 10032 of *LNCS*, pages 967–997. Springer, 2016.

[KKW17] W. S. Kennedy, V. Kolesnikov, and G. T. Wilfong. Overlaying conditional circuit clauses for secure computation. In *ASIACRYPT'17*, volume 10625 of *LNCS*, pages 499–528. Springer, 2017.

[Kol05] V. Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In *ASIACRYPT'05*, volume 3788 of *LNCS*, pages 136–155. Springer, 2005.

[KS08] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP'08*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

[KSMB13] B. Kreuter, A. Shelat, B. Mood, and K. Butler. PCF: A portable circuit format for scalable two-party secure computation. In *USENIX Security'13*, pages 321–336. USENIX, 2013.

[KSS12] B. Kreuter, A. Shelat, and C. Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security'12*, pages 285–300. USENIX, 2012.

[LWN+15] C. Liu, X. Wang, K. Nayak, Y. Huang, and E. Shi. ObliVM: A programming framework for secure computation. In *S&P'15*, pages 359–376. IEEE, 2015.

[MCB07] A. Mishchenko, S. Chatterjee, and R. K. Brayton. Improvements to technology mapping for LUT-based FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAS'07)*, 26(2):240–253, 2007.

[MCCB07] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton. Combinational and sequential mapping with priority cuts. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'07)*, pages 354–361. IEEE, 2007.

[MMRR10] O. Martinello, F. S. Marques, R. P. Ribas, and A. I. Reis. KL-cuts: A new approach for logic synthesis targeting multiple output blocks. In *Design, Automation Test in Europe Conference Exhibition (DATE'10)*, pages 777–782. IEEE, 2010.

[MNPS04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *USENIX Security'04*, pages 287–302. USENIX, 2004.

[MPS15] T. Malkin, V. Pastro, and A. Shelat. The whole is greater than the sum of its parts: Linear garbling and applications. Workshop talk at Securing Computation Workshop in Berkley, 2015. Online: https://simons.berkeley.edu/talks/tal-malkin-2015-06-10.

[NPS99] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Electronic Commerce (EC'99)*, pages 129–139. ACM, 1999.

[PSSZ15] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security'15*, pages 515–530. USENIX, 2015.

[RME+12] S. Ray, A. Mishchenko, N. Een, R. Brayton, S. Jang, and C. Chen. Mapping into LUT structures. In *Design, Automation Test in Europe Conference Exhibition (DATE'12)*, pages 1579–1584. IEEE, 2012.

[Sec15] Dyadic Security. Dyadic's DSM web suite use-cases, 2015. Online: https://www.dyadicsec.com/wp-content/uploads/2015/06/dyadicwhitepaper.pdf.

[SHS+15] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar. TinyGarble: Highly compressed and scalable sequential garbled circuits. In *S&P'15*, pages 411–428. IEEE, 2015.

[SS06] R. Schürer and W. Schmid. *Monte Carlo and Quasi-Monte Carlo Methods 2004*, chapter MinT: A Database for Optimal Net Parameters, pages 457–469. Springer, 2006. Online: http://mint.sbg.ac.at.

[Syn] Synopsys Inc. FPGA-based design. http://www.synopsys.com/tools/implementation/fpgaimplementation/pages/default.aspx.

[Syn10] Synopsys Inc. Design compiler, 2010. http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler.

[SZ13] T. Schneider and M. Zohner. GMW vs. Yao? Efficient secure two-party computation with low depth circuits. In *FC'13*, volume 7859 of *LNCS*, pages 275–292. Springer, 2013.

[TP14] M. S. Turan and R. Peralta. The multiplicative complexity of Boolean functions on four and five variables. In *Lightweight Cryptography for Security and Privacy (LightSec'14)*, volume 8898 of *LNCS*, pages 21–33. Springer, 2014.

[Wol] C. Wolf. Yosys open synthesis suite. http://www.clifford.at/yosys/.

[Xil] Xilinx Inc. Vivado design suite - hlx editions. http://www.xilinx.com/products/design-tools/vivado.html.

[Xil09] Xilinx Inc. XST synthesis overview, 2009. http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ise_c_using_xst_for_synthesis.htm.

[Yao86] A. C. Yao. How to generate and exchange secrets. In *FOCS'86*, pages 162–167. IEEE, 1986.

[ZRE15] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *EUROCRYPT'15*, volume 9057 of *LNCS*, pages 220–250. Springer, 2015.