Constant Round Maliciously Secure 2PC with Function-independent Preprocessing using LEGO

Jesper Buus Nielsen[1,⋆], Thomas Schneider[2,⋆⋆], and Roberto Trifiletti[1,⋆] (✉)

[1] Aarhus University
{jbn|roberto}@cs.au.dk
[2] Technische Universität Darmstadt
thomas.schneider@crisp-da.de

Abstract. Secure two-party computation (S2PC) allows two parties to compute a function on their joint inputs while leaking only the output of the function. At TCC 2009 Orlandi and Nielsen proposed the LEGO protocol for maliciously secure 2PC based on cut-and-choose of Yao's garbled circuits at the gate level and showed that this is asymptotically more efficient than on the circuit level. Since then the LEGO approach has been improved upon in several theoretical works, but never implemented. In this paper we describe further concrete improvements and provide the first implementation of a protocol from the LEGO family. Our protocol is optimized for the offline/online setting and supports function-independent preprocessing using only a constant number of rounds. We have benchmarked our prototype and find that our protocol can compete with all existing implementations and that it is often more efficient. As an example, in a LAN setting we can evaluate an AES-128 with online latency down to 1.13 ms, while if evaluating 128 AES-128 in parallel the amortized cost is 0.09 ms per AES-128. This online performance does not come at the price of offline inefficiency as we achieve comparable performance to previous, less general protocols, and significantly better if we ignore the cost of the function-independent preprocessing. Also, as our protocol has an optimal 2-round online phase it is significantly more efficient than previous protocols' when considering a high latency network.

Keywords: Secure Two-party Computation, Implementation, LEGO, XOR-Homomorphic Commitments, Selective OT-Attack

1 Introduction

Secure two-party computation is the area of cryptology dealing with two mutually distrusting parties wishing to compute an arbitrary function f on private inputs. Say A has input x and B has input y. The guarantee offered from securely computing f is that the only thing learned from the computation is the output $z = f(x, y)$, in particular nothing is revealed about the other party's input that cannot be inferred from the output z. This seemingly simple guarantee turns out to be extremely powerful and several real-world applications and companies using secure computation have arisen in recent years [BCD+09, BLW08, dya, par, sep]. The idea of secure computation was initially conceived in 1982 by Andrew Yao [Yao82, Yao86], particularly for the *semi-honest* setting, in which all parties are assumed to follow the protocol specification but can try to extract as much information as possible from the protocol execution. Yao gave an approach for preventing any such extraction using a technique referred to as the *garbled circuit technique*. At a very high level, using the abstraction of [BHR12b], A starts by garbling or "encrypting" the circuit f using the garbling algorithm $(F, e, d) = \mathsf{Gb}(f)$ obtaining a garbled circuit F, an input encoding function e and an output

⋆ The authors acknowledge support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61361136003) for the Sino-Danish Center for the Theory of Interactive Computation and from the Center for Research in Foundations of Electronic Markets (CFEM), supported by the Danish Strategic Research Council. Also partially supported by the European Research Commission Starting Grant 279447.

⋆⋆ This work has been partially funded by the German Federal Ministry of Education and Research (BMBF) within CRISP, by the DFG as part of project E3 within the CRC 1119 CROSSING, and by the European Union'sCC 7th Framework Program (FP7/2007-2013) under grant agreement n.609611 (PRACTICE).

NDSS '17, 26 February - 1 March 2017, San Diego, CA, USA
 ISBN 1-1891562-46-0, http://dx.doi.org/10.14722/ndss.2017.23075

decoding function d. It then encodes its input $X = \mathsf{En}(x, e)$ and sends (F, X, d) to B. Then, using oblivious transfer (OT), A blindly transfers a garbled version Y of B's input which enables B to compute a garbled output $Z = \mathsf{Ev}(F, X \| Y)$ which it can then decode to obtain $z = \mathsf{De}(Z, d)$. Finally B returns z to A.

	Secret Sharing based			Garbled Circuit based			
Protocol	[KSS13]*	[DZ16]*	[DNNR16]*	[WMK16]	[LR15]	[RR16]	**This Work**
Constant round	✗	✗	✗	✓	✓	✓	✓
Independent preprocessing	N/A	✗	✗	✗	✗	✗	**13.84 ms**
Dependent preprocessing	✗	N/A	N/A	62 ms	74 ms	5.1 ms	**0.74 ms**
Best online latency	12 ms	6 ms	**1.05 ms**	21 ms	7 ms	1.3 ms	1.13 ms
Best online throughput	∼1 ms	0.4 ms	**0.45 µs**	N/A	N/A	0.26 ms	0.08 ms

* Uses dedicated techniques for evaluating AES.

Table 1. Overview of state-of-the-art protocols and the best reported timings for AES-128 with malicious security on a LAN in the offline/online setting. All timings are per AES-128. N/A stands for Not Available.

As already mentioned, the above sketched approach can be proven secure in the semi-honest setting [LP09]. In the stronger *malicious* setting however it completely breaks down as in this model the parties are allowed to deviate arbitrarily from the protocol specification. One of the most obvious attacks is for A to garble a different function $f' \neq f$ which could enable A to learn y from the resulting value z' without B even noticing this. To combat this type of attack the cut-and-choose technique can be applied: instead of garbling a single circuit, A garbles several copies and sends these to B. A random subset of them is now selected for checking by B and if everything is correct, with overwhelming probability some fraction of the remaining circuits must be correct as well. Leaving out many details these can now be evaluated to obtain a single final output. While this approach thwarts the above attack it unfortunately opens up for several new ones that also need to be dealt with, for instance ensuring *input consistency* for all the remaining evaluation garbled circuits. Another more subtle issue in the malicious setting is the *Selective-OT Attack* (also called Selective Failure Attack) as pointed out in [MF06, KS06]. A corrupt A can cheat when offering B the garbled inputs in the OT step by using a bogus value for either the 0 or the 1 input. This will either result in B aborting as it cannot evaluate the garbled circuit or it will go through undetected and B will return the output to A. Either way the input bit of B is revealed to A which is a direct breach of security. It is easy to see that using this attack A can learn any l bits of B's input with probability 2^{-l} if not properly dealt with.

Over the last decade several solutions to the above issues have been proposed, along with dramatic efficiency improvements for secure 2PC protocols based on the cut-and-choose approach of garbled circuits [LP07, PSSW09, LP11, sS11, HEKM11, KsS12, Bra13, FN13, HKE13, Lin13, MR13, sS13, HMsG13, FJN14, AMPR14, WMK16]. Finally we note for completeness that secure computation has also been studied in great detail for many other settings, including the more general multi-party case (MPC). Several different adversarial models such as honest majority [GMW87, BOGW88, CCD88], dishonest majority [DPSZ12] and covert security [AL07] have also been proposed in the literature. In this work we focus solely on the special case of two parties with malicious security and in the next section we discuss the reported concrete efficiency of state-of-the-art protocols in this setting.

1.1 Related Work

In the less than 10 years since the first reported implementation of maliciously secure 2PC based on garbled circuits [LPS08], the performance advancements have been enormous [PSSW09, sS11, KsS12, sS13, FN13, FJN14, AMPR14, LR15, RR16, WMK16]. Furthermore different settings and hardware configurations have been explored, notably using commodity grade GPU's in [FN13, FJN14] and large-scale CPU clusters [sS13] to parallelize the bulk of the computation. In the single-execution setting based solely on standard hardware

the best reported performance time is that of [WMK16] which evaluates an AES-128 in total time 65 ms. In addition, the works of [LR15] and [RR16] explore the more restricted setting of amortizing secure 2PC based on the cut-and-choose and the dual execution approach, respectively. By amortizing we mean that the protocols exploit constructing multiple secure evaluations of the same function f yielding impressive performance benefits over the more general single execution setting. Furthermore these protocols are in the offline/online setting where the bulk of the computation and communication can be done before the inputs are determined. We highlight that for both protocols, the offline computation depends on the function to be computed and we will refer to this as *dependent* preprocessing. However both protocols allow for the inputs to be chosen sequentially when securely evaluating f. This allows for a low latency online phase which is desirable for many applications. For securely computing 1024 AES-128 [LR15] reports 74 ms offline and 7 ms online per evaluation, while the more recent [RR16] reports 5.1 ms offline and 1.3 ms online for the same setting. Furthermore [RR16] achieves a 0.26 ms online phase when considering throughput alone, *i.e.* batched evaluation.

Another direction in secure computation is the secret sharing approach where the parties initially secret share their inputs and then interactively compute the function f in a secure manner. A particularly nice property of these protocols is that when considering the offline/online setting the offline phase can usually be done independently of the circuit f which we call *independent* preprocessing. This allows for naively utilizing parallelism in the preprocessing phase and also adds more flexibility as the offline material produced is universal. Another benefit is that in general this secret-sharing technique works for any number of parties and over any field, which depending on the desired functionality f can significantly increase performance. We note however that these protocols usually employ expensive public-key cryptography in the preprocessing phase and are therefore much slower than the offline phases of *e.g.* [LR15, RR16]. Finally the inherent interactiveness of the online phase, which has $\mathcal{O}(\mathsf{depth}(f))$ rounds of interaction, makes these protocols ill-suited for high latency networks such as WANs. There are many variations of the secret sharing approach but they typically enjoy the same overall pros and cons in terms of independent preprocessing and required interactivity. Examples of recent protocols following this paradigm are: TinyOT [NNOB12,LOS14,BLN+15], SPDZ [DPSZ12, DKL+13, KSS13, KOS16], MiniMac [DZ13, DLT14, DZ16], and TinyTables [DNNR16]. The fastest reported online time for computing an AES-128 in this setting is 1.05 ms by [DNNR16] using dependent preprocessing. The work of [KSS13] reports 12 ms online time using independent preprocessing, but the evaluation exploits the algebraic structure of AES-128. Furthermore [DNNR16] has an impressive throughput of 0.45 µs per AES-128 while [KSS13] and [DZ16] have throughput ∼1 ms and 0.4 ms, respectively.

In Table 1 we give an overview of the properties of the mentioned protocols and the reported timings for securely evaluating AES-128 on LAN in the offline/online setting. As the secret sharing based, non-constant round, protocols are ill-suited for high latency networks we omit this from Table 1 since no AES-128 timings are published for these schemes in a WAN setting (however see Section 6.2 for a WAN comparison of the garbled circuit protocols). The timings reported for [DNNR16,LR15,RR16] and **This Work** are all measured on the same hardware (Amazon Web Services, c4.8xlarge instances on 10 Gbit LAN), while the timings for [WMK16] are on a less powerful instance (Amazon Web Services, c4.2xlarge instances on 2.5 Gbit LAN). Finally the results of [KSS13, DZ16] have been obtained on high-end Desktop machines with 1 Gbit LAN. The timings of [LR15, RR16] and **This Work** are all for 1024 AES-128 evaluations. For comparability the timings of [WMK16] in Table 1 are for the offline/online setting and not for a single-execution. We believe the difference in performance ((62 ms + 21 ms) vs. 65 ms) of their protocol can be explained by the inability to interleave the sending/checking and evaluation of garbled circuits in the offline/online setting. In summary, as can be seen in the table our work is the first implementation of a protocol combining the advantages of independent (and dependent) preprocessing using only a constant number of rounds.

1.2 LEGO

The Large Efficient Garbled-circuit Optimization (LEGO) was first introduced by Nielsen and Orlandi in [NO09] which showed a new approach for maliciously secure 2PC based on cut-and-choose of garbled *gates*. This gave an asymptotic complexity improvement to $\mathcal{O}(s/\log(|f|))$ as opposed to $\mathcal{O}(s)$ for the standard *circuit* cut-and-choose approach for statistical security s. However the construction of [NO09] was heavily based on

expensive public-key cryptography and was mainly considered an asymptotic advancement. This was later improved in the two follow-up works of MiniLEGO [FJN+13] and TinyLEGO [FJNT15] yielding incrementing asymptotic and concrete efficiency improvements. In a nutshell, the LEGO technique works by the generator A first garbling multiple individual AND gates (as opposed to garbling entire circuits) and sending these to the evaluator B. Then a cut-and-choose step on a random subset of these gates is carried out and finally the remaining unchecked gates are combined (or soldered) into a garbled fault tolerant circuit computing f. A crucial ingredient for securely soldering the garbled gates into circuits are XOR-homomorphic commitments which in [NO09] were realized using expensive Pedersen commitments [Ped92]. In the follow-up construction of [FJN+13] these were replaced by an asymptotically more efficient construction, however the concrete communication overhead of the proposed commitment scheme was inadequate for the protocol to be competitive for realistic circuit sizes and parameters. In the recent works of [FJNT16, CDD+16] this overhead has been improved to an optimal rate-1 and the resulting UC-secure XOR-homomorphic commitment scheme is both asymptotically and concretely very efficient. Finally the work of [HZ15] introduced a different primitive for LEGO soldering called XOR-Homomorphic Interactive Hash, which has some advantages over the commitment approach. However, the best instantiation of XOR-Homomorphic Interactive Hash still induces higher overall overhead than the commitment approach when using the schemes of [FJNT16, CDD+16].

Although the original LEGO protocol, and the above-mentioned follow-up works, asymptotically are very efficient, the overall consensus in the secure computation community has been that the reliance of XOR-homomorphic commitments for all circuit wires hinders actual practical efficiency. In this work we thoroughly investigate the practical efficiency of the LEGO approach and, in contrast to earlier beliefs, we demonstrate that it is indeed among the most practical protocols to date for general secure 2PC using garbled circuits.

Setting	Ind. Preprocessing*	Dep. Preprocessing	Online Latency	Online Throughput
Single Execution				
1 x AES-128	89.61 ms	13.23 ms	1.46 ms	1.46 ms
1 x SHA-256	478.54 ms	164.40 ms	11.19 ms	11.19 ms
Amortized				
128 x AES-128	14.85 ms	0.68 ms	1.15 ms	0.09 ms
128 x SHA-256	173.05 ms	12.13 ms	9.35 ms	1.09 ms

* Not including the time to compute the initial BaseOTs.

Table 2. Performance summary of our protocol on a high bandwidth (10Gbit) LAN network.

1.3 Our Contributions

We implement the TinyLEGO protocol with added support for both independent and dependent preprocessing. Furthermore our protocol supports fully reactive computation, meaning that when a function result has been obtained, another function depending on this result can be evaluated. Also, the independent preprocessing phase can be rerun at any time if additional garbling material is necessary. As part of our prototype we also implement the XOR-homomorphic commitment scheme of [FJNT16] and report on its efficiency separately as we believe our findings can be of independent interest. This is to our knowledge the first implementation of a protocol based directly on the LEGO paradigm and of the mentioned commitment scheme. The support for independent preprocessing is achieved from the fact that the bulk of the computation using the LEGO approach is based on cut-and-choose of independently garbled gates and hence only depends on the security parameter and the number of AND gates one wishes to preprocess. The subsequent soldering phase can then be seen as a dependent preprocessing phase where knowledge of the circuit f is required. This

multi-level preprocessing is in contrast to previous non-LEGO protocols based on cut-and-choose of garbled circuits in the offline/online setting where the entire offline phase depends on the circuit to be evaluated. In more detail our main contributions are as follows:

1. We propose a new technique for dealing with the selective-OT attack on 2PC protocols based on garbled circuits. Our technique makes use of a globally correlated OT functionality ($\mathcal{F}_{\Delta\text{-ROT}}$) combined with XOR-homomorphic commitments and a Free-XOR garbling scheme. Using the well-known fact of Beaver [Bea95] that OTs can be precomputed, we can mitigate the selective-OT attack by having the circuit constructor decommit to a single value per input bit of the evaluator in the online phase. This ensures that if the constructor tries to cheat, the evaluator aborts regardless of the value of his input. The technique is general and we believe that it can be used in other 2PC protocols based on garbled circuits as well. We also provide a more efficient instantiation of $\mathcal{F}_{\Delta\text{-ROT}}$ than previously appearing in the literature by tightening the analysis of the construction presented in [BLN+15].
2. As part of our 2PC prototype we also implement the XOR-homomorphic commitment scheme of [FJNT16]. It is already known that this scheme is asymptotically very efficient, but this is to our knowledge the first time its practical efficiency has been thoroughly investigated. The result is a very efficient scheme achieving an amortized cost of less than a microsecond for both committing and decommitting to a random value. To maximize performance we utilize cache efficient matrix-transposition and inspired by the construction of [CDD+16] we use the Intel Streaming SIMD Extension (SSE) instruction PCLMULQDQ to efficiently compute the required linear combinations.
3. We build our LEGO prototype on top of the above-mentioned implementation which results in a very efficient and flexible protocol for maliciously secure reactive 2PC. As our online phase consists of an optimal two rounds, we can securely evaluate an AES-128 with latency down to 1.13 ms. When considering throughput we can do each AES-128 block in amortized online time 0.08 ms (considering 1024 blocks). In applications where independent preprocessing can be utilized our offline phase is superior to all previous 2PC protocols, in particular based on our experiments we see a 6-54x gain over [RR16] depending on network and number of circuits considered. If preprocessing is not applicable, for most settings we cannot compete with the offline phase of [RR16], but note that the difference is within a factor 1.2-3x. See Table 2 for an overview of our performance in different settings and Section 6 for a more detailed presentation and comparison of our results.

2 Preliminaries

In this section we introduce some of the technical background for LEGO garbling, adopting the notation and conventions of the original TinyLEGO protocol [FJNT15] for ease of exposition.

2.1 Notation

We will use as shorthand $[n] = \{1, 2, \ldots, n\}$ and $[i; j] = \{i, i+1, i+2, \ldots, j\}$. We write $e \in_R S$ to mean: sample an element e uniformly at random from the finite set S. We sometimes (when the semantic meaning is clear) use subscript to denote an index of a vector, *i.e.*, x_i denotes the i'th bit of a vector x. We use k to denote the computational security parameter and s to represent the statistical security parameter. Technically, this means that for any fixed s and any polynomial time bounded adversary, the advantage of the adversary is $2^{-s} + \text{negl}(k)$ for a negligible function negl. *i.e.*, the advantage of any adversary goes to 2^{-s} faster than any inverse polynomial in the computational security parameter. If $s = \Omega(k)$ then the advantage is negligible.

2.2 Circuit Conventions

We assume A is the party constructing the garbled gates and call it the *constructor*. Likewise, we assume B is the party evaluating the garbled gates and call it the *evaluator*. Furthermore, we say that the functionality

they wish to compute is $z = f(x, y)$, where A gives input x and B gives input y. We assume that f is described using only NOT, XOR and AND gates. The XOR gates are allowed to have unlimited fan-in, while the AND gates are restricted to fan-in 2, and NOT gates have fan-in 1. All gates are allowed to have unlimited fan-out. We denote the bit-length of x by $|x| = n_{\mathsf{A}}$, the bit-length of y by $|y| = n_{\mathsf{B}}$ and let $n = n_{\mathsf{A}} + n_{\mathsf{B}}$. We will denote the bit-length of the output z by $|z| = m$. Furthermore, we assume that the first n_{A} input wires are for A's input and the following n_{B} input wires are for B's input.

We define the *semantic* value of a wire-key of a garbled gate to be the bit it represents. We will use K_j^b to denote the j'th wire key representing bit b. Sometimes, when the context allows it, we will let $L_g^{b_l}$, $R_g^{b_r}$, and $O_g^{b_o}$ denote the left input, right input, and output key respectively for garbled gate g representing the bits b_l, b_r and b_o. When the bit represented by a key is unknown we simply omit the superscript, *e.g.* K_j.

2.3 Free-XOR and Soldering

The LEGO protocols [NO09,FJN+13] and [FJNT15] all assume that the underlying garbling scheme supports the notion of Free-XOR, meaning that the XOR of the 0- and 1-key on any wire of any garbled gate yields the same value, Δ, which we call the *global difference*. In addition to making garbling and evaluating XOR gates virtually free [KS08], this optimization also allows for easily soldering wires together. A soldering of two wires is a way of transforming a key representing bit b on one wire into a key representing bit b on the other wire. As we will see in more detail below, with Free-XOR, a soldering is simply the XOR of the 0-keys on the two wires. Furthermore, in order to avoid any cheating all wires of all garbled gates are committed to using a XOR-homomorphic commitment functionality $\mathcal{F}_{\mathsf{HCOM}}$ and the solderings are then always decomitted when needed.

As an example, assume we wish to solder the output wire of gate g onto the left input wire of gate $g+1$. In doing so we decommit the value $S_{g+1}^L = O_g^0 \oplus L_{g+1}^0$ using $\mathcal{F}_{\mathsf{HCOM}}$. When gate g outputs the key representing the bit b one can now learn the left b-key for gate $g+1$. Specifically it can be computed as $O_g^b \oplus S_{g+1}^L = \left(O_g^0 \oplus (b \cdot \Delta)\right) \oplus O_g^0 \oplus L_{g+1}^0 = L_{g+1}^0 \oplus (b \cdot \Delta) = L_{g+1}^b$. This obviously generalizes when one wishes to solder together several wires, *e.g.* if we wish to solder the output wire of gate g to the left input wire of gate $g+1$, $g+2$ and $g+3$, then it is enough to decommit the values $S_{g+1}^L = O_g^0 \oplus L_{g+1}^0, S_{g+2}^L = O_g^0 \oplus L_{g+2}^0, S_{g+3}^L = O_g^0 \oplus L_{g+3}^0$.

It is also straightforward to evaluate XOR gates as part of the soldering: To compute the XOR of g and $g+1$ and then use this result as the left input to gate $g+2$ we decommit the value $S_{g+2}^L = \left(O_g^0 \oplus O_{g+1}^0\right) \oplus L_{g+2}^0$. We see now that $O_g^a \oplus O_{g+1}^b \oplus S_{g+2}^L = a \cdot \Delta \oplus b \cdot \Delta \oplus L_{g+2}^0 = L_{g+2}^{a \oplus b}$ as desired. In conclusion a soldering is therefore always the XOR of the 0-keys of the wires going into an XOR gate and the 0-key of the wire we wish to solder the result onto.

3 A New Approach to Eliminate the Selective-OT Attack

As already mentioned in Section 1 the selective-OT attack enables a corrupt A to learn any input bit of B with success probability $1/2$ for each bit. Prior work has dealt with this attack in different ways, but the off-the-shelf black-box solution has typically been the s-probe resistant matrix approach of [LP07, sS13, LR15]. These approaches augment the evaluation circuit $f \to f'$ so that learning any $s-1$ input bits of B in f' leaks nothing about the actual input used in the original f. The downside of this approach is that the input length of B needs to be increased, which in turn results in more communication, computation and OTs. For the approach of [LP07] and [LR15] the increase is to $n'_{\mathsf{B}} = n_{\mathsf{B}} + \max(4n_{\mathsf{B}}, {}^{20s}/_{3})$ while for the approach of [sS13] we have $n'_{\mathsf{B}} \leq n_{\mathsf{B}} + \lg(n_{\mathsf{B}}) + n_{\mathsf{B}} + s + s \cdot \max(\lg(4n_{\mathsf{B}}), \lg(4s))$. In addition to extending the input size, experiments of [LR15] show that producing the s-probe resistant version of f can be a computationally expensive task (up to several seconds for 1000-bit input).

3.1 Our New Approach

We propose a new approach that combines the use of 1-out-of-2 Δ-ROTs (also called globally correlated OTs), XOR-homomorphic commitments and the Free-XOR technique that sidesteps the need of expanding

the input size of B as described above. We recall that Δ-ROTs are similar to Random OTs (ROT), except that all OTs produced are correlated with a global difference Δ. In other words, for each Δ-ROT i produced the following relation holds for a fixed Δ: $r_i^1 = r_i^0 \oplus \Delta$ where $r_i^0, \Delta \in \{0,1\}^k$ are uniformly random strings known to the sender and $b_i \in \{0,1\}$ is the uniformly random choice-bit of the receiver who learns $r_i^{b_i}$ as part of the OT protocol. Our approach is described below and is inspired by the protocol of Beaver [Bea95] for precomputing OT.

1. The parties precompute $n_{\mathsf{B}} + s$ Δ-ROTs such that the sender learns (Δ, r_i^0) and the receiver learns $(r_i^{b_i}, b_i)$ for $i \in [n_{\mathsf{B}} + s]$. The sender will now commit, using the XOR-homomorphic commitment scheme, to Δ and each r_i^0. In order to verify that the sender indeed committed to the Δ used in the OTs, the parties run a simple check in the following way: B sends $\{(r_j^{b_j}, b_j)\}_{j \in [n_{\mathsf{B}}; n_{\mathsf{B}}+s]}$ to A which in turn needs to successfully decommit to the received values. The s OTs used for the check are hereafter discarded. The reason why B needs to send the values to A in the first place is that it needs to prove knowledge of the value $r_j^{b_j}$ before it is safe for A to open it. For each of the s tests, if A did not commit to the Δ used in the OTs, then it can only pass the test with probability at most $1/2$ as the choice-bits of B are uniformly random. Because these are also chosen independently we see that the check therefore catches a cheating A with overwhelming probability $1 - 2^{-s}$.
2. If the check succeeds A uses the Δ learned from the OTs above as the global difference in the Free-XOR garbling scheme. Recall that this means that all garbling keys will be correlated in the same way as the Δ-ROTs, *i.e.* $K_l^1 = K_l^0 \oplus \Delta$ for all l. In particular this is the case for the keys associated to the input of B which need to be obliviously transferred in the online phase. In addition, in our 2PC protocol all 0-keys K_l^0 have been committed to using the same XOR-homomorphic commitment scheme as used for the OT strings r_i^0 and Δ.
3. Finally when B learns its real input y, it computes $e = y \oplus b$ and sends this to A where b are the choice-bits used in the precomputed Δ-ROTs. A will respond by decommitting the values $\{D_i = K_i^0 \oplus r_i^{e_i}\}_{i \in [n_{\mathsf{B}}]}$. B can now compute its actual input keys $K_i^{y_i} = D_i \oplus r_i^{b_i} = K_i^0 \oplus r_i^{e_i} \oplus r_i^{b_i} = K_i^0 \oplus y_i \cdot \Delta$.

The above approach eliminates the selective-OT attack as the only way a corrupt A can cheat is by committing to different values $r'^0_i \neq r_i^0$ where r_i^0 is the value sent using the Δ-ROTs. However if this is the case then $D'_i \oplus r_i^{b_i} \notin \{K_i^0, K_i^1\}$ and B will abort regardless of the value of his input y_i. One caveat of the above approach is that it allows a corrupt A to flip an input bit i of B without getting caught by instead of committing to r_i^0, it commits to $r_i^0 \oplus \Delta$. In our 2PC protocol we eliminate this issue by ensuring that $\mathrm{lsb}(\Delta) = 1$ and by securely leaking $\mathrm{lsb}(r_i^0)$ to B. This enables B to check that the resulting key $K_i = D_i \oplus r_i^{b_i}$ indeed carries the correct value y_i, by verifying that

$$\begin{aligned} y_i &= \mathrm{lsb}(K_i) \oplus \mathrm{lsb}(D_i) \oplus \mathrm{lsb}(r_i^0) \oplus e_i \\ &= \mathrm{lsb}(K_i) \oplus \mathrm{lsb}(K_i^0) \oplus \mathrm{lsb}(r_i^{e_i}) \oplus \mathrm{lsb}(r_i^0) \oplus e_i \\ &= \mathrm{lsb}(K_i) \oplus \mathrm{lsb}(K_i^0) \ . \end{aligned}$$

This secure leaking is described in the *VerLeak* step of Fig. 1 and the check is carried out as part of the **Eval** step of Fig. 2, both of which are presented in Section 4.

3.2 On Constructing Δ-ROTs

The above technique requires $n_{\mathsf{B}} + s$ Δ-ROTs to obliviously transfer the input keys of B. However, current state-of-the-art protocols for OT extension [NNOB12,BLN+15,ALSZ15,KOS15] all produce ROTs. It can be seen by inspecting the above OT extension protocols that they all produce a weaker variant of Δ-ROT called *leaky* Δ-ROT as an intermediate step. The leaky Δ-ROT is identical to Δ-ROT in that all OT pairs are correlated with a global Δ, however a corrupt receiver can cheat and learn some bits of Δ with non-negligible probability. It can be seen that for each bit learned of Δ, the receiver gets caught with probability $1/2$, which means it can learn up to $s - 1$ bits of Δ, while the other bits remain uniformly random in its view. The work of [BLN+15] gives a construction for Δ-ROTs of string length v from leaky Δ-ROTs of string length

$22v/3 \sim 7.33v$ using linear randomness extraction. Concretely they propose multiplying all the strings learned from the leaky Δ-ROT protocol with a random matrix $A \in \{0,1\}^{22v/3 \times v}$. In this work we observe that the factor $22/3$ is not tight and by applying Theorem 1 below we can reduce the number of rows in A down to $v + s$, going from a multiplicative to an additive factor.

Theorem 1 ([ZB11], Theorem 7). *Let $X = x_1, x_2, \ldots, x_u$ be a binary sequence generated from a bit fixing source in which l bits are unbiased and independent, the other $u - l$ bits are fixed or copies of the l independent random bits. Let A be a $u \times v$ random matrix such that each entry of A is 0 or 1 with probability $1/2$. Given $Y = XA$, then we have that*

$$\Pr_A[\rho(Y) \neq 0] \leq 2^{v-l}$$

where $\rho(Y)$ is defined as the statistical distance to the uniform distribution over $\{0,1\}^v$, i.e. $\rho(Y) = \frac{1}{2}\sum_{y \in \{0,1\}^v} |\Pr[Y = y] - 2^{-v}|$.

Now let $u = v + s$ and let Δ have length u and let ΔA have length v. Consider an adversary B who is allowed to try to learn some of the bits of Δ to make ΔA non-uniform. In our setting, if an adversary B tries to learn λ bits of Δ it is caught except with probability $2^{-\lambda}$. If B is not caught then it learns λ bit positions and the remaining bits are independent and uniform. Since we have $u = v + s$, the l in the above theorem equals $u - \lambda = v + (s - \lambda)$ when B learns λ bits, *i.e.*, $2^{v-l} = 2^{\lambda - s}$. This implies that for all $0 \leq \lambda < s$ it holds that the probability that B is not caught and at the same time ΔA is not uniform is at most $2^{-\lambda}2^{\lambda-s} = 2^{-s}$. Clearly, for all $\lambda \geq s$, then the probability that B is not caught and at the same time ΔA is not uniform is at most the probability B is not caught, which is at most 2^{-s}. This shows that when $u = v + s$ then for all B, the probability that B is not caught and at the same time ΔA is not uniform is at most 2^{-s}, which is negligible.

The consequence of our new analysis is that we can choose the random matrix as $A \in \{0,1\}^{(v+s)\times v}$ and thus we only have to produce leaky Δ-ROTs of length $v+s$ instead of length $22v/3$, a substantial optimization. As we ultimately require (non-leaky) Δ-ROTs of length k, we can utilize any of the mentioned OT extension protocols to produce leaky Δ-ROTs of length $k + s$ and then apply the linear randomness extraction on the resulting OT-strings. For the parameters $s = 40$ and $k = 128$ considered in this work our refined analysis ultimately yields an improvement of around a factor 5.6x compared to the previous best known result of [BLN+15].

4 The Protocol

As already mentioned in Section 1, our protocol is based on TinyLEGO [FJNT15], but modified to support preprocessing of all garbled components along with our new approach for dealing with the selective-OT attack of Section 3.1. This includes removing the restriction of B choosing input and committing to the cut-and-choose challenges before obtaining the garbling material and solderings. As a consequence, our modifications allow for multi-leveled preprocessing and offers a very efficient online phase. A description of our resulting protocol can be found in Fig. 1 and Fig. 2 while we refer to [FJNT15] for a more detailed specification of the original TinyLEGO protocol. At a high level our protocol can be broken down into four main steps.

1. The **Setup** phase which initializes the commitment scheme. All public-key operations of our protocol can be carried out in this initial step, including the BaseOTs required for bootstrapping OT extension.
2. The **Generate** step takes as input the number of gates, number of inputs and number of outputs the parties wish to preprocess. After sending the garbled gates and wire authenticators and committing to all associated wires a cut-and-choose step is run between the parties. The wire authenticators is a gadget that either accepts or rejects a given key (without revealing the value of the key) and it was shown in [FJNT15] that constructing AND buckets from both garbled gates and wire authenticators can significantly reduce the overall communication compared to using garbled gates alone. After the cut-and-choose step, using the XOR-homomorphic commitments, the parties solder the remaining garbled gates and wire authenticators randomly into independent fault tolerant AND buckets.

The Setup step is only required to be run once, regardless of the number of future calls to Generate.
Setup(pp):

1. On input $(k, s, p_g, p_a, \beta, \alpha, \tilde{\beta}, \tilde{\alpha}) \leftarrow \text{pp}$, A and B initialize the functionality $\mathcal{F}_{\mathsf{HCOM}}$ by sending $(\mathsf{init}, \mathsf{sid}, \mathsf{A}, \mathsf{B}, \kappa)$ to it, where κ is the key-length of the garbling scheme.

The Generate step produces $\bar{q}$ garbled AND gates which can be soldered into circuits that in total can have $\bar{n}$ inputs and $\bar{m}$ outputs. It is possible to do multiple calls to Generate at any point in time in order to produce more garbling material.
Generate$(\bar{q}, \bar{n}, \bar{m})$**:**

1. Let Q and A be chosen such that after running the below cut-and-choose step, with overwhelming probability $(\bar{q}\beta + \bar{n}\tilde{\beta})$ garbled gates and $(\bar{q}\alpha + \bar{n}\tilde{\alpha})$ wire authenticators survive.
2. A and B invoke $\mathcal{F}_{\Delta\text{-ROT}}$ $(\bar{n} + s)$ times from which A learns Δ and random strings r_i^0 and B learns the choice-bits b_i and $r_i^{b_i}$ for $i \in [\bar{n} + s]$. Furthermore A instructs $\mathcal{F}_{\Delta\text{-ROT}}$ to ensure that $\text{lsb}(\Delta) = 1$.
3. Next A garbles Q AND gates and constructs A wire authenticators using Δ and sends these to B.
4. A then commits to each wire of the garbled AND gates, each authenticated wire produced, Δ, the 0-strings received from $\mathcal{F}_{\Delta\text{-ROT}}$ and $\bar{m} + s$ random values $\{v_j\}_{[\bar{m}+s]}$. Thus it sends $3Q + A + 1 + \bar{n} + s + \bar{m} + s$ values to $\mathcal{F}_{\mathsf{HCOM}}$.

VerLeak:

5. For $i \in [\bar{n}]$ and $j \in [\bar{m} + s]$, A sends $\text{lsb}(r_i^0)$ and $\text{lsb}(v_j)$ to B.
6. B challenges A to send, using $\mathcal{F}_{\mathsf{HCOM}}$, s random linear combinations of r_i^0, v_j and Δ for $i \in [\bar{n}]$ and $j \in [\bar{m}]$. Also, the l'th combination is set to include a one-time blinding value $v_{\bar{m}+l}$ for $l \in [s]$.
7. B verifies that lsb of the received s values correspond to the same linear combinations of the initial lsb values sent by A in step 5. In addition, if Δ is included in a linear combination B flips the value. This ensures that indeed $\text{lsb}(\Delta) = 1$.

Cut-and-Choose:

8. After receiving the garbled gates (wire authenticators), B chooses to check any gate (wire authenticator) with probability p_g (p_a). B then challenges A to send, using $\mathcal{F}_{\mathsf{HCOM}}$, two random inputs and the corresponding AND output of the selected garbled gates and a random input of the selected wire authenticators.
9. B evaluates the selected garbled gates and wire authenticators and checks that they output the received output key and that they verify the values received from $\mathcal{F}_{\mathsf{HCOM}}$, respectively.
10. In addition, for $i \in [\bar{n}; \bar{n} + s]$ B sends $r_i^{b_i} = r_i^0 \oplus (b_i \cdot \Delta)$ to A which in turn instructs $\mathcal{F}_{\mathsf{HCOM}}$ to send back the same value. This is to ensure that the committed Δ is the one used in $\mathcal{F}_{\Delta\text{-ROT}}$.

Bucketing:

11. For the remaining garbled gates (wire authenticators), B samples and sends a random permutation that fully describes how these are to be combined into $\bar{q}$ AND buckets of size $\beta + \alpha$, $\bar{n}$ input buckets of size $\tilde{\beta}$ and $\bar{n}$ input authenticators of size $\tilde{\alpha}$.
12. A then sends, using $\mathcal{F}_{\mathsf{HCOM}}$, all the required solderings such that for all the specified bucket gadgets, each component is defined with the same input/output keys.

Fig. 1. The modified TinyLEGO protocol with support for preprocessing in the $\mathcal{F}_{\mathsf{HCOM}}$, $\mathcal{F}_{\Delta\text{-ROT}}$-hybrid model – part 1.

3. The **Build** step takes as input the circuit description f and through the XOR-homomorphic commitments, A sends the required solderings to glue together a subset of previously produced AND buckets so that they compute f.
4. Finally the **Eval** step depends on the parties' inputs to f. It consists of two rounds, first B sends a correction value e which depends on his input y, and as a response A decommits to B's masked input

The Build step uses the garbling material created in Generate to construct a fault tolerant garbled circuit computing f.
Build(f)**:**

1. A instructs $\mathcal{F}_{\mathsf{HCOM}}$ to send the required solderings such that the first $|f|$ unused AND buckets correctly compute f. This includes the solderings to attach n_{A} input buckets and n input authenticators onto the final garbled circuit.

The Eval step involves the parties transferring to B all input keys in an oblivious manner which then allows B to evaluate and decode a garbled circuit previously constructed using the Build step.
Eval(x, y)**:**

1. For input $y \in \{0,1\}^{n_{\mathsf{B}}}$, B sends $e = b \oplus y$ to A, where b is the first n_{B} unused choice-bits of $\mathcal{F}_{\Delta\text{-ROT}}$.
2. A then instructs $\mathcal{F}_{\mathsf{HCOM}}$ to send to the values $\{D_i = r_i^0 \oplus K_i^0 \oplus e_i \cdot \Delta\}_{i\in[n_{\mathsf{B}}]}$ where K_i^0 is the 0-key on the i'th input wire of B and r_i^0 is the first unused Δ-ROT string. Also, for the input $x \in \{0,1\}^{n_{\mathsf{A}}}$, it sends the corresponding input keys $\{K_i^{x_i}\}_{i\in[n_{\mathsf{A}}]}$ directly to B.
3. Finally A instructs $\mathcal{F}_{\mathsf{HCOM}}$ to send the output decoding values $\{D_j = v_j^0 \oplus K_j^0\}_{j\in[m]}$ to B where K_j^0 is the j'th output 0-key of the garbled circuit and $\{v_j\}_{j\in[m]}$ are the first m unused blinding values setup in the *VerLeak* step.
4. Upon receiving the above, for $i \in [n_{\mathsf{B}}]$ B computes $K_{n_{\mathsf{A}}+i} = r_i^{b_i} \oplus D_i$ and $\mathrm{lsb}(K_i^0) = \mathrm{lsb}(D_i) \oplus \mathrm{lsb}(r_i^0) \oplus e_i$ and verifies that $\mathrm{lsb}(K_{n_{\mathsf{A}}+i}) \oplus \mathrm{lsb}(K_i^0) = y_i$. Then using the input authenticators, B also verifies that the keys $\{K_i\}_{i\in[n_{\mathsf{A}}]}$ of A are valid input keys to the garbled circuit.
5. If everything checks out B evaluates the previously constructed garbled circuit on the input keys $(K_1, K_2, \ldots, K_n)$ to obtain the output keys $(Z_1, Z_2, \ldots, Z_m)$. For $j \in [m]$ it then computes $d_j = \mathrm{lsb}(v_j) \oplus \mathrm{lsb}(D_j)$ and decodes $z_j = \mathrm{lsb}(Z_j) \oplus d_j$. Finally it outputs $z = (z_1, z_2, \ldots, z_m)$.

Fig. 2. The modified TinyLEGO protocol with support for preprocessing in the $\mathcal{F}_{\mathsf{HCOM}}$, $\mathcal{F}_{\Delta\text{-ROT}}$-hybrid model – part 2.

keys as well as sending it's own input keys directly. Finally A also decommitments to the lsb of all output 0-keys. This allows B to evaluate the garbled circuit and decode the final output.

We highlight that our modified protocol also naturally supports the notion of *streaming* or *pipelining* of garbled circuit evaluation [HEKM11] which was not the case in [FJNT15]. This can be seen by the fact that one can evaluate the circuit f in a layered approach and using the XOR-homomorphic commitments to glue the output of one layer onto the input of the next layer. Each of these layers can be processed on the fly with our protocol and in this way the entire circuit never needs to be stored at any given time. This approach is similar to that proposed in [MGBF14] for reusing garbled values, however in this setting everything works out-of-the-box due to the XOR-homormophic commitments on all circuit wires.

The LEGO approach also has the advantage compared to traditional cut-and-choose protocols that only a single *fault tolerant* garbled circuit is produced and evaluated. This removes the necessity of ensuring input consistency for all the evaluation circuits. It also sidesteps the overhead of transferring multiple sets of input keys in the online phase, one set for each evaluation circuit.

4.1 Bucketing

With the bucketing approach of [FJNT15] each AND bucket consists of β garbled gates and α wire authenticators. For any garbled gate (wire authenticator) the probability p_g (p_a) is used to determine if it is checked in the cut-and-choose or not. The value of p_g and p_a therefore induces a certain sense of "quality" level of the remaining non-checked garbled components which affects the required bucketing size. In addition there are also the special cases of input buckets and input authenticators, which are buckets that consist of garbled gates only (size $\tilde{\beta}$) and wire authenticators only (size $\tilde{\alpha}$), respectively. These are attached to the input wires of the final garbled circuit and serve to guarantee validity of the input keys, along with guaranteeing that

B can always learn the final output $f(x, y)$, even if A is cheating. This is so since the input buckets can be seen as a trapdoor that together with the global difference Δ allows B to extract the input x of A. It is then clear that it can compute $f(x, y)$ directly. These special buckets are necessary as our regular AND buckets do not rule out outputting both the 0 and 1-key (say if one of the garbled gates in the bucket is in fact a NAND gate). However if a bucket outputs two distinct keys it is guaranteed that they are both valid and hence their XOR is Δ and B can extract x. If no cheating is detected then the input buckets are simply ignored by B.

As already established in the original LEGO paper [NO09], the number of AND gates q directly affects the required size of the buckets, meaning that as q grows the required bucket size can be decreased while still retaining the same level of security. Theorem 2 below gives a direct way of computing the success probability of a corrupt A given the parameters $q, n, \beta, \alpha, \tilde{\beta}, \tilde{\alpha}$.

Theorem 2 ([FJNT15], Lemma 9). *Given the bucketing parameters $q, n, \beta, \alpha, \tilde{\beta}, \tilde{\alpha}$ for the case where $\alpha = \beta - 1$ we can bound the probability of the bad bucketing events occurring as:*

$$\Pr[\textit{Any bad bucket}] \leq$$
$$q \cdot \Bigg(\prod_{i=\beta}^{1} \Big(\frac{(1-p_g)4i}{p_g(q\beta + n\tilde{\beta}) + (1-p_g)4i} \Big) + \sum_{l=2}^{\beta} \prod_{i=\beta}^{l} \Big(\frac{(1-p_g)4i}{p_g(q\beta + n\tilde{\beta}) + (1-p_g)4i} \Big) \cdot \prod_{j=\alpha}^{\alpha+2-l} \Big(\frac{(1-p_a)2j}{p_a(q\alpha + n\tilde{\alpha}) + (1-p_a)2j} \Big) \Bigg)$$

$$\Pr[\textit{Any bad input authenticator}] \leq$$
$$n \cdot \sum_{v=1}^{\lceil \frac{\tilde{\alpha}}{2} \rceil} \prod_{l=\tilde{\alpha}}^{v} \Big(\frac{(1-p_a)2l}{p_a(q\alpha + n\tilde{\alpha}) + (1-p_a)2l} \Big)$$

$$\Pr[\textit{Any bad input bucket}] \leq$$
$$n \cdot \sum_{l=1}^{\lceil \frac{\tilde{\beta}}{2} \rceil} \prod_{i=\tilde{\beta}}^{l} \Big(\frac{(1-p_g)4i}{p_g(q\beta + n\tilde{\beta}) + (1-p_g)4i} \Big)$$

Based on Theorem 2, given q and n we directly compute the optimal choices of $\beta, \alpha, \tilde{\beta}, \tilde{\alpha}$ for minimizing the overall communication of the protocol while still guaranteeing a negligible upper bound on the success probability of a corrupt A. This is a once and for all computation so in our implementation these values are stored in a precomputed table that can be looked up when q and n have been decided. We note that it is also possible to minimize for lowest possible bucket size if desired. This has the effect of reducing the computational overhead in the online phase at the price of increasing both communication and computational overhead in the independent preprocessing phase. In our experiments in Section 6 we solely minimize for overall communication.

4.2 Security

Our protocol is similar to the TinyLEGO protocol in [FJNT15] and the proof follows the same general outline. We will therefore only give a very brief sketch of the overall proof strategy and then describe how to deal with the changes we made relative to TinyLEGO.

Consider first the case where the garbler A is corrupted. As is typically the case it is easy to see that the communication of the protocol does not leak any information on the input of B as long as the protocol does not abort. The garbler A might however give wrong input to some of the OTs used by B to choose its input keys, giving rise to selective errors where the abort probability depends on the input of B. The garbler A might also create some bad garbled gates which could *a priori* result in an abort or a wrong output, which might both leak information on the input of B. The problem with bad gates is handled exactly as in [FJNT15], by setting the cut-and-choose parameters and buckets sizes appropriately. We however handle the case with bad inputs to the OTs differently, as described below. In the universal composable (UC) framework [Can01], when A is corrupt, we also need to be able to extract the input of the corrupted A from its communication and input to ideal functionalities (OT and commitment). We handle this exactly as [FJNT15]: the cut-and-choose ensures that most key authenticators only accept their two corresponding committed keys. For a good key authenticator the accepted key can then be compared to the committed values to compute its semantic value. The bucket size has been set such that there is a majority of good key authenticators on all input wires. This allows to compute the semantic of any accepted key by taking majority.

Consider then the case where the evaluator B is corrupted. As is typically the case, the communication clearly does not leak information to B about the input of A. All that is left is therefore to describe how to handle two technical requirements imposed by the UC framework. First we have to describe how to extract the input y of a corrupted B. Second, after learning y and $z = f(x, y)$ we must enforce that the simulated protocol constructs a circuit that evaluates to z. This must be done without knowing x. Extracting y is handled exactly as in [FJNT15]. We simply inspect which choice bits B uses in the OTs for selecting its input. Hitting z in the simulation is also handled exactly as in [FJNT15]. We simply construct the circuit correctly and run with input 0 for A. This gives a potentially wrong output $z' = f(0, y)$. We patch this by giving appropriately chosen wrong output decoding information by opening a wrong least significant bit of the output key for the output wires i where $z'_i \neq z_i$. This is possible as the simulator controls the ideal functionality for commitment in the simulation.

We now focus on the changes we made to [FJNT15].

Change 1 In both protocols the output decoding information consists of the least significant bit of the output keys, securely leaked via the commitment scheme. However, the implementation differs. We use a non-interactive implementation which is slightly heavier on communication. In [FJNT15] they use an interactive protocol with less communication.

Change 2 In [FJNT15] they let B commit to the cut-and-choose challenges and choose his input via OT before obtaining the garbled gates, wire authenticators and solderings. We have removed this step. Now B picks his input after the circuit is constructed and does not commit to his challenges.

Change 3 In our protocol we take the global Δ-value output by the OT extension and reuse it as the global Δ in the Free-XOR garbling scheme. In [FJNT15] they use two independent values.

Change 4 We protect against selective error on the input of B by using the same Δ in OT extension and garbling and using a Δ-ROT to offer the input keys to B. We also leak the least significant bit of the Δ-OT 0-strings to B for all his input wires. In [FJNT15] a different technique was used.

Change 1 does not affect security. It was introduced to give better execution time for typical circuits.

We now address Change 2. The reason why B commits to the cut-and-choose challenges and chooses his input via OT before obtaining the garbled material in [FJNT15] is that security is proven via a reduction to a standard (non-adaptive) selective garbling scheme (*e.g.* [BHR12b]), where the adversary in the security game must supply its input before it gets the garbled circuit. Therefore they need to be able to extract the input and cut-and-choose challenges of B before assembling the garbled circuit in the simulation. We have skipped this step as it would prevent independent preprocessing. Now that we assemble the circuit before B picks its input, the hope would be that we could do a reduction to an adaptive garbling scheme.

However due to the soldering approach of LEGO where the XOR of 0-keys are sent to the evaluator before the input is determined, it is unclear how to reduce security to the standard notion of adaptive garbling, as some of these 0-keys are not known to the simulator. To overcome this we instead identify the defining property we need from the underlying garbling function which is that *the outputs of the hash*

function appear random as long as the inputs are all unknown and have sufficient entropy, even if the inputs are related. A non-extractable, non-programmable random oracle clearly satisfies this property [BR93]. The type of garbling scheme considered in this work [ZRE15] could in principle be made adaptively secure by using a programmable random oracle using techniques described in [BHR12a, LR14].[1] We avoid using the programability of the random oracle by changing the usual approach a little. Normally security proofs need to program the circuit to hit the right output. We instead garble the circuits correctly and then program or equivocate the output decoding information to decode to the value we need to hit. Specifically we use equivocation of the UC commitment scheme to incorrectly open the least significant bit of the output keys when we need to hit a different value. Returning to the simulation, we therefore garble all gates and answer all cut-and-choose challenges honestly. As we now know all garbling keys we can also open consistently to differences between 0-keys for the remaining evaluation gates. Finally in order to make the complete soldered garbled circuit "hit" the output z we equivocate the openings of the least significant bits of the output keys such that this becomes the decoded value. This is possible as the decoding information is only opened after we extract the input of the corrupt receiver. If the garbling is done using a random oracle this will have the same distribution as in the protocol.

For Change 3 we again use that we are in the random oracle model. The first step in the proof will be to go from the case where Δ is reused in the garbling scheme to the case where an independent Δ' is used for garbling as in [FJNT15]. In this hybrid we also let A commit to Δ'. We then use equivocation of the commitment scheme to make the cut-and-choose proof that $\Delta' = \Delta$ go through. Since B only sees one key for each wire and has high entropy on Δ and Δ', this change will be indistinguishable to B if the output of the hash function appears random as long as the inputs are all unknown and have sufficient entropy, even if inputs are related. As above, a non-extractable, non-programmable random oracle clearly satisfies this property.

We finally address Change 4. Using a Δ-ROT to offer the input keys to B ensures that when A inputs the keys to the OT, either both are correct or both are incorrect. If both are incorrect, it will be detected by a key authenticator independently of the input of B. This means that the only remaining attack vector is for A to swap the two correct keys. This is detected by B as B knows the least significant bit of the Δ-ROT 0-strings and the two correct keys have different least significant bits. Again the detection is independent of the input of B. Notice that the output decoding information is sent to B using $\mathcal{F}_{\mathsf{HCOM}}$ *after* he sends his input correction value e, so we can equivocate it to hit the correct output z. This, together with the fact that the outputs of the hash function appear random, is why it is secure to perform the *VerLeak* step before learning the input of B.

As argued above our modified protocol can be proven secure in the non-extractable, non-programmable random oracle model following the proof of [FJNT15]. As we do not require programmability of the random oracle we conjecture that our protocol can be proven secure in the standard (OT hybrid) model using the recently proposed ICE framework of [FM16], an extension of the UCE framework of [BHK13]. However we note that it does not seem like our scheme can be proven secure using the UCE framework as in our setting all the garbled gates are related (all garbled with the same Δ) and therefore a single leakage phase as prescribed in the UCE framework seems insufficient.

5 Implementation

In this section we highlight some of the more technical details of our implementations of the XOR-homomorphic commitment scheme and our final 2PC protocol supporting independent preprocessing. The source code of the project can be found at `https://github.com/AarhusCrypto/TinyLEGO`.

[1] It is also possible to build an adaptively secure garbling scheme (with short input keys) using a non-programmable random oracle [BHK13], but this particular scheme is not as efficient as the one of [ZRE15].

5.1 UC-Secure XOR-Homomorphic Commitments

As part of our full 2PC prototype we implement the XOR-homomorphic commitment scheme of [FJNT16] as a separate subprotocol. This is to our knowledge the first time a scheme following this OT + PRG blueprint has been implemented and we believe our experimental findings are of independent interest. At a high level, the scheme works by the parties initially doing $\tilde{n}$ BaseOTs of security parameter k-bit strings, where $\tilde{n}$ is the code-length for some linear error correcting code $\mathcal{C}$ with parameters $[\tilde{n}, \kappa, s]_{\mathbb{F}_2}$ with κ being the bit-length of the committed messages. The parties then expand the received k-bit strings into bit-strings of length γ using a PRG which then define the γ random commitments the sender is committing to. Next the sender sends a correctional value for each produced commitment to turn these into codewords of $\mathcal{C}$. Finally, to ensure that the sender sent valid corrections, the receiver challenges the sender to decommit to $2s$ random linear combinations of all produced commitments. This is done in a way such that no information is leaked about the γ committed values. Additively homomorphism then follows from the fact that the code $\mathcal{C}$ is linear and all operations on the expanded PRG strings are linear as well. We highlight the fact that any XOR homomorphic commitment scheme supports the notion of batch opening/decommitment which is similar in nature to the above consistency check. The idea is that the sender initially sends the decommited values directly to the receiver, who hereafter challenges the sender to decommit to s linear combinations of the postulated values where s is the statistical security parameter. Notice that it is only in the initial commit step that $2s$ combinations is necessary. If the decommitted values match the linear combinations of the postulated values the receiver accepts. As now only s values are decommitted this approach has the benefit of making the communication overhead independent of the number of values decommitted to. For the full details we refer to [FJNT16].

We implement the above scheme in C++14 taking advantage of multi-core capabilities and Intel SSE instructions. We can therefore base the PRG on AES-NI in counter mode and for the error correcting code we use a modified version of the linux kernel implementation of the BCH code family [Hoc59, BRC60]. As part of the commitment step the parties are required to transpose a binary matrix $S \in \{0,1\}^{\tilde{n} \times \gamma}$ in order to efficiently address the committed values in column-major order. As γ in our case can be huge (> 220 mio. for 2000 AES-128 computations) we use the efficient implementation of Ekhlund's cache-efficient algorithm for binary matrix transposition [Ekl72] presented in [ALSZ13, ALSZ15].[2] As a side note we also augment the OT extension code to support the randomness extraction technique described in Section 3.1 to implement the $\mathcal{F}_{\Delta\text{-ROT}}$ functionality needed in our 2PC protocol.

During the development of our implementation we identified the main computational bottleneck of the scheme to be the computation of the random linear combinations. Even if these operations are based on mere XORs, when implemented naively, the number of required instructions is still γs in expectation. Therefore, inspired by [CDD+16] we use a different approach for computing the consistency checks using Galois field multiplication. Combined with efficient matrix transposition the effect of using $\mathrm{GF}(2^l)$ multiplication can be seen as computing l linear combinations in parallel. For our particular setting we set $l = 128$ as this is the smallest power of 2 greater than the required $2s$ for $s = 40$. We can then use the Intel SSE instruction PCLMULQDQ to very efficiently compute the $\mathrm{GF}(2^{128})$ multiplications. In detail, our approach to compute the checks is as follows:

1. Given a random challenge element $\alpha \in_R \mathrm{GF}(2^l)$ the matrix S of committed values in column-major order is split into $u = \lceil s/l \rceil$ blocks $B_i \in \{0,1\}^{\tilde{n} \times l}$. Each block is then transposed into row-major order.
2. For $i \in [u]$ and $j \in [\tilde{n}]$, the new matrix $B'_i = B_i^j \cdot \alpha^i$ is computed where $B_i^j \in \{0,1\}^l$ is the j'th row of B_i interpreted as an element of $\mathrm{GF}(2^l)$.
3. Finally the combined matrix $B' = \sum_{i=1}^{u} B'_i$ is produced and transposed back into column-major order.

Each column of B' can now be seen as a random linear combination of all values of S. As a further optimization we see that most GF elements are only multiplied a single time in the above and we can therefore postpone the expensive degree reduction step of the multiplication until B' has been fully computed. This is not so for computing α^i which we therefore reduce at each iteration. In total the required number of degree

[2] Available at `https://github.com/encryptogroup/OTExtension`

reductions therefore becomes $n + u$ as opposed to $(n + 1)u$. Our experiments show that using the above method of computing 128 linear combinations compared to the naive approach is between 10-13x faster starting at even moderately sized $\gamma > 8000$.

γ	#Threads	Commit (µs)	Decommit (µs)
500	1	7.21 (2815.28)	2.20
1,000	2	3.85 (1401.83)	1.40
15,000	4	0.64 (93.99)	0.34
50,000	8	0.57 (28.49)	0.22
500,000	20	0.45 (3.25)	0.17
10,000,000	200	0.21 (0.35)	0.14
200,000,000	400	0.20 (0.21)	0.14

Table 3. Timings for committing to bit strings of length 128 with $s = 40$. All timings are µs per commitment. The commit time in parentheses includes the cost of the initial BaseOTs.

As our implementation of the XOR-homomorphic commitment scheme might be of independent interest we here present our observed timings for committing and decommitting to γ random bit-strings of length 128 with $k = 128$ and $s = 40$. We instantiate the binary BCH code with parameters $[312, 128, 41]$ and for convenience we use the implementation of [ALSZ15] augmented with our randomness extraction technique to compute the required 312 Random OTs. In total this takes about 1400 ms with our implementation, where 1392 ms are due to the BaseOTs (using PVW [PVW08]). From the timings reported in [CO15] we predict that this initial setup step can be done much faster (around 20 ms) using their implementation, but since this requires a programmable random oracle assumption and this cost amortizes away as γ grows we did not pursue this. Also, if the commitment scheme is used in an application that already relies on oblivious transfer, OT extension can be used to produce the starting BaseOTs at very low cost. We report our findings in Table 3. As the scheme requires $\tilde{n}$ BaseOTs to setup we include this cost in the commit timings in parentheses. It can thus be seen by comparing the commitment numbers how the initial OT cost amortizes away as γ increases. As there is no initial cost associated with decommitment these timings are only affected by the number of worker-threads we spawn. Furthermore these experiments were performed on the local LAN setup described in Section 6 and *not* on the Amazon Web Services (AWS) architecture.

5.2 2PC with preprocessing using LEGO

We implement the TinyLEGO protocol with our modifications on top of the previously described commitment scheme. The code is also written in C++14 and makes heavy use of parallelism and Intel SSE instructions for garbling and evaluation of the garbled gates. At a high level, the **Generate** step is implemented by first partitioning the inputs $(\bar{q}, \bar{n}, \bar{m})$ into t equally sized subsets for some free parameter t. The main thread then starts t parallel executions of the generate step with two synchronization points, one where the commitment to Δ is sent (which only one execution is charged with), and one after the cut-and-choose step. The latter is necessary as the random permutation that describes the initial bucketing must only be revealed after all garbled components have been sent to B. We emphasize that it is due to our preprocessing being independent of the structure of f that we can trivially parallelize the above step using any number of threads t. Due to the above design we also run t executions of the commitment scheme, however for the PRG expansion we use the same seed OT values in all executions. As the PRG is based on a block cipher in counter mode this is not an issue as execution $i + 1$ sets it's counter sufficiently high compared to the i'th execution so there is

no overlap. Since they all use the same seed OTs the choice-bits are also the same across all executions and they can therefore be combined in the same way as for a single execution.

The **Build** and **Eval** phases follow roughly the same design pattern as above. We note however that in these phases each thread is responsible for the soldering and evaluation of an entire circuit. The garbling and evaluation of garbled gates and wire authenticators are implemented purely as 128-bit SSE instructions to maximize performance. We base the hash function for garbling gates and producing wire authenticators on Fixed-Key AES-NI as advocated in [BHKR13]. This choice is mainly motivated by producing as comparable results as possible to previous works that are also based on Fixed-Key AES-NI.

As already mentioned the **Eval** phase consists of two rounds, one where B specifies the input mask and a second where A replies with its keys and decommitments. The reply of A has communication complexity kn_{A} for A's input keys and $(\tilde{n}+k)(n_{\mathsf{B}}+m)$ for the decommitments of B's input keys and the lsb masks of the output keys. Here $\tilde{n}$ is the code-length of the BCH code. We remark that it is possible to reduce the communication cost of the decommitments to $(\tilde{n}+k)s+k(n_{\mathsf{B}}+m)$ using the batch decommit approach mentioned in Section 5.1, but at the cost of adding an additional round. For the circuits we considered in our experiments (AES-128, SHA-256) we observed a loss of around a factor 1.25 in the LAN setting and much more in the WAN setting with this approach. Though for other circuits where the ratio $(n_{\mathsf{B}}+m)/|f|$ is substantial and both network latency and bandwidth are low we suspect that adding this extra round can pay off. Finally if one is willing to assume a programmable random oracle the online cost for the output bits can be eliminated entirely as the security proof simulator in that case can program the oracle to output matching output keys for a preprocessed decommitment lsb-bit once it learns the final output.

6 Performance

To give as broad a view as possible of the performance of our prototype we run our experiments in a local LAN setting and on both a LAN and WAN on the Amazon Web Services (AWS). In more detail:

Local LAN with two machines, one acting as A and the other acting as B. We measured a total bandwidth of 942 Mbits/sec with round trip time (rtt) 0.12 ms. Both machines run Ubuntu 16.04 with an Intel Ivy Bridge i7 3.5 GHz quad-core processor and 32 GB DDR3 RAM.

AWS LAN with two c4.8xlarge instances located in the Virginia region connected via a high performance LAN. We measured a total bandwidth of 9.52 Gbits/sec with rtt 0.16 ms. Both machines run Amazon Linux AMI 2016.03.2 with an Intel Xeon E5-2666 v3 (Haswell) processor with 36 vCPUs and 60 GB RAM.

AWS WAN with two c4.8xlarge instances, one in Virginia and the other in Ireland. We measured an available bandwidth of 214 Mbits/sec on average for a single TCP connection and up to 3.17 Gbit/sec when running many parallel connections. The rtt measured was 81.32 ms. Both machines run Amazon Linux AMI 2016.03.2 with an Intel Xeon E5-2666 v3 (Haswell) processor with 36 vCPUs and 60 GB RAM.

For all settings the code was compiled using GCC-5.4 with the -O3 optimization flag set. As mentioned in Section 5.1 the implementation used for the BaseOTs are based on [ALSZ15] using PVW [PVW08]. If one is willing to assume a programmable random oracle, these can be replaced with the fast protocol and implementation of [CO15] and we would expect a total cost around 20 ms as opposed to 850 ms (AWS LAN) with the current implementation.

6.1 Our Performance Results

We summarize our measured results in Table 4 on the facing page for the three above-mentioned settings. All numbers reported are averages of 10 executions. Not surprisingly we see the best performance on the AWS machines in the LAN setting where we can evaluate an AES-128 with latency 1.13 ms or 0.08 ms per AES-128 of throughput in the online phase. We also see that when considering 1024 AES-128 evaluations the dependent preprocessing + the online phase is below 2 ms. When including the cost of the independent

Setting	Circuit	Number	BaseOTs	Ind. Preprocessing	Dep. Preprocessing	Online (latency)	Online (throughput)
Local LAN	AES-128	1	1400.89	220.44	15.12	2.52	2.52
		32	44.90	87.63	3.35	2.25	0.35
		128	11.23	70.89	2.95	1.86	0.26
		1024	1.40	61.33	2.85	1.60	0.25
	SHA-256	1	1400.39	1381.05	208.00	22.65	22.65
		32	44.94	812.12	44.59	11.77	3.12
		128	11.22	771.27	37.72	10.43	3.02
AWS LAN	AES-128	1	850.42	89.61	13.23	1.46	1.46
		32	26.61	27.91	0.85	1.23	0.18
		128	6.65	14.85	0.68	1.15	0.09
		1024	0.84	13.84	0.74	1.13	0.08
	SHA-256	1	852.90	478.54	164.40	11.19	11.19
		32	26.82	165.26	14.87	9.14	1.42
		128	6.67	173.05	12.13	9.35	1.09
		256	3.34	183.51	11.70	9.56	1.05
AWS WAN	AES-128	1	2980.25	1881.63	96.66	83.17	83.17
		32	93.75	142.00	5.19	83.21	2.71
		128	23.44	72.31	3.96	83.65	0.73
		1024	2.96	39.18	2.12	83.15	0.62
	SHA-256	1	3043.64	2738.62	350.01	93.94	93.94
		32	92.98	670.98	42.01	92.42	4.04
		128	23.66	431.71	25.44	92.38	1.70
		256	11.75	356.48	27.97	92.74	1.87

Table 4. Evaluator timings measured in the Local LAN, AWS LAN and AWS WAN setting for AES-128 and SHA-256 with $k = 128$ and $s = 40$. All timings are ms per circuit.

Circuit	Number	β	p_g	α	p_a	BaseOTs	Ind. Preprocessing	Dep. Preprocessing	Online
AES-128	1	7	2^{-4}	6	2^{-3}	19.52 kB	14.94 MB	226.86 kB	16.13 kB
	32	4	2^{-2}	3	2^{-2}	19.52 kB (610 B)	279.78 MB (8.74 MB)	7.26 MB (226.86 kB)	516.10 kB (16.13 kB)
	128	4	2^{-3}	3	2^{-3}	19.52 kB (153 B)	924.68 MB (7.22 MB)	29.04 MB (226.86 kB)	2.06 MB (16.13 kB)
	1024	4	2^{-5}	3	2^{-6}	19.52 kB (19 B)	6.57 GB (6.42 MB)	232.31 MB (226.86 kB)	16.52 MB (16.13 kB)
SHA-256	1	5	2^{-3}	4	2^{-4}	19.52 kB	120.34 MB	2.93 MB	22.23 kB
	32	4	2^{-4}	3	2^{-5}	19.52 kB (610 B)	2.73 GB (85.19 MB)	93.63 MB (2.93 MB)	712.70 kB (22.23 kB)
	128	4	2^{-6}	3	2^{-5}	19.52 kB (153 B)	10.31 GB (80.54 MB)	374.52 MB (2.93 MB)	2.85 MB (22.23 kB)
	256	4	2^{-7}	3	2^{-5}	19.52 kB (76 B)	20.28 GB (79.20 MB)	749.03 MB (2.93 MB)	5.70 MB (22.23 kB)

Table 5. Bucketing parameters and data received by the evaluator in the different phases of our protocol for AES-128 and SHA-256 with $k = 128$ and $s = 40$. Numbers in parentheses are data per circuit produced.

preprocessing each AES-128 can be done in total time less than 16 ms. Similarly when considering 256 SHA-256 evaluations the online phase can be done with latency 9.14 ms or 1.05 ms of throughput per SHA-256. Also the dependent preprocessing + online phase and total cost is below 22 ms and 205 ms, respectively (when preprocessing material enough for 256 SHA-256 evaluations).

For the single execution setting we see a significant increase in execution time for the dependent preprocessing compared to above. This is due to the design of our prototype which only uses multiple execution threads in the dependent preprocessing and online phases if several, possibly different, circuits are processed at the same time. We also note that our prototype requires a large amount of RAM as we store all garbling material and commitments in-memory. This design choice is due to convenience, but for a deployed system based on the LEGO approach this should be addressed using external memory sources with support for pipelined evaluation as described in Section 4.

For the AWS WAN setting we see that a single execution of AES-128 takes around 83 ms of online time where almost all of the execution time is spent waiting due to a latency of ∼81 ms. The latency also severely impacts the two preprocessing phases where the independent preprocessing takes around 1882 ms (20x compared to AWS LAN) and the dependent offline phase takes 96 ms (7x compared to AWS LAN). However this overhead can be somewhat mitigated when considering several circuits, down to a factor 2-4x compared to AWS LAN due to the computation and communication being more interleaved and better utilization of the available bandwidth with several TCP connections.

Finally in Table 5 we report on the amount of data our prototype transfers from the circuit constructor to the circuit evaluator for both AES-128 and SHA-256. For clarity we have not included the communication from evaluator to constructor, but note that for 1024 AES-128 and 256 SHA-256 a total of 8.12 MB and 4.09 MB are transferred, respectively, and for both cases around 99% of the communication stems from the initial BaseOTs. The table also summarizes the bucketing parameters used in our experiments, which have been chosen so that the probability bound given by Theorem 2 in Section 4.1 is negligible. Also we set the two input bucket parameters $\tilde{\beta} = 2\beta + 1$ and $\tilde{\alpha} = 2\alpha + 1$ which ensures a correct majority for all the input buckets and authenticators except with negligible probability. For the data numbers in Table 5 it can be seen in parentheses how the relative preprocessing cost of a circuit decreases as more evaluations are considered. We highlight that in this work (and previous LEGO protocols) this is due to the increasing number of *gates* produced, not by the number of circuits. As an example of this effect, going from a single AES-128 with 6928 gates to 1024 AES-128 with 7,094,272 gates decreases the cost of the independent preprocessing by a factor 2.3x per AES-128, from 14.94 MB to 6.42 MB. It is worth noting that the "LEGO effect" only applies to the independent preprocessing. This is because in the subsequent dependent preprocessing step two solderings (k bits each) are sent per gate of the circuit f and not for each garbled gate produced. In addition a small constant 2.2 kB of decommitment data is transferred in this phase for the s challenge linear combinations. For the online step the communication consists of $n_{\mathsf{A}}k$ bits for the constructors input + $(\tilde{n}+k)(n_{\mathsf{B}}+m)$ bits for the decommitments to the evaluators input and the output decoding bits where $\tilde{n}$ is the code-length of the ECC $\mathcal{C}$ used in the commitment scheme. In Section 5.2 we discussed how this could further be reduced to $(\tilde{n}+k)s + k(n_{\mathsf{B}}+m)$ at the price of adding an extra round to the online phase.

6.2 Comparison with Related Work

We compare our measured timings to those reported in the recent works of [LR15] and [RR16], both of which are solely applicable in the amortized setting. In contrast our protocol can naturally handle the single execution setting, along with a more general amortized setting where several *distinct* functions can be preprocessed in the same batch. However to make a meaningful comparison we focus on the "traditional" amortized setting considering $32, 128$ and 1024 AES-128 computations and we summarize the comparison in Table 6. The independent preprocessing timings for our protocol consists of the BaseOTs + the independent preprocessing. The first thing to notice is that for applications where independent preprocessing is applicable, and can therefore be disregarded, our dependent offline performance is superior to both prior works for any number of AES-128 computations by a large margin. Compared to [LR15] our reported timings are better by 100-358x depending on the setting and number of circuits. For [RR16] the gap is smaller, but still substantial, namely by 6-54x. For applications where independent preprocessing can not be utilized we are still superior

Protocol	Setting	Number	Ind. Preprocessing	Offline	Online
[LR15]	AWS LAN	32	✗	197	12
		128	✗	114	10
		1024	✗	74	7
	AWS WAN	32	✗	1,126	163
		128	✗	919	164
		1024	✗	759	160
[RR16]	AWS LAN	32	✗	45	1.7
		128	✗	16	1.5
		1024	✗	5.1	1.3
	AWS WAN	32	✗	282	190
		128	✗	71	191
		1024	✗	34	189
This Work	AWS LAN	32	**54.52**	**0.85**	**1.23**
		128	**21.5**	**0.68**	**1.15**
		1024	**14.68**	**0.74**	**1.13**
	AWS WAN	32	**235.75**	**5.19**	**83.21**
		128	**95.75**	**3.96**	**83.65**
		1024	**42.14**	**2.12**	**83.15**

Table 6. Comparison of the reported timings for AES-128 in the AWS LAN and AWS WAN setting with $k = 128$ and $s = 40$. The preprocessing column includes the cost of the BaseOTs for This Work. All timings are ms per AES-128.

to the work of [LR15], but for most settings and number of circuits we cannot compete with the offline phase of [RR16]. However the differences are typically within a factor 1.2-3x.

For the online timings in the AWS LAN setting for AES-128 we measure faster overall timings than [RR16] for all number of circuits by a tiny margin. As the differences are less than half a millisecond we believe the only reasonable thing to conclude is that the online times are comparable. Though when looking at raw throughput we outperform [RR16] by more than a factor 3x (0.08 ms vs. 0.26 ms). Going beyond Table 6 and considering the larger SHA-256 circuit we note that our online phase is not faster than [RR16] (9.35 ms vs. 8.8 ms for 128 circuits). This is again due to the design of our prototype that uses a single execution thread in the online phase per circuit, while [RR16] uses several threads. We therefore see it as an interesting problem for future research to tailor the LEGO online phase to better exploit parallelism for a single circuit.

With regards to online latency in the AWS WAN setting there is however no doubt that our two round online phase outperforms both [LR15] and [RR16]. This is directly related to the previous protocols having more rounds which in a high latency network significantly decreases performance. For comparison [LR15] has a 4 round online phase and [RR16] has 5. One thing to note however is that [RR16] delivers output to both parties in 5 rounds, whereas both our work and [LR15] would need an extra round to support this. Also the implementation of [LR15] is written in a mix of Java and C++ using JNI which definitely adds overhead to the running time. It is however unclear how much of a speedup a native implementation would achieve, but we suspect it would be substantial. The implementation of [RR16] is written solely in C++ and according to the paper also takes full advantage of parallelization.

In Table 7 we summarize the required communication for the previously considered protocols and our work for the same setting as Table 6. As was the case for the measured timings, when disregarding the cost of the independent preprocessing, our protocol requires significantly less communication in both the offline and online phase compared to the previous works. For the offline phase the communication is 5-12x less than [RR16] and 16-358x less than [LR15]. If the independent preprocessing is included as part of the

Protocol	Number	Ind. Preprocessing	Offline	Online
[LR15]	32	✗	8.13 MB	312 kB
	128	✗	5.45 MB	238 kB
	1024	✗	3.76 MB	170 kB
[RR16]*	32	✗	3.75 MB	25.76 kB
	128	✗	2.5 MB	21.31 kB
	1024	✗	1.56 MB	16.95 kB
This Work	32	**8.74 MB**	**226.86 kB**	**16.13 kB**
	128	**7.22 MB**	**226.86 kB**	**16.13 kB**
	1024	**6.42 MB**	**226.86 kB**	**16.13 kB**

* Dual-execution, so total offline communication is double the reported numbers.

Table 7. Comparison of the data received by the evaluator for different number of executions of AES-128 with $k = 128$ and $s = 40$. All numbers are data per AES-128.

offline phase our protocol however requires transferring more raw data than the previous two works for any of the considered number of circuits. However this is only so because we are considering multiple copies $(32, 128$ and $1024)$ of the same function AES-128. If we instead consider settings with few copies (the larger the better), a single copy or several different circuits, the amount of data received in the independent + dependent preprocessing phase of our protocol can match or be lower than the dependent offline phase of [RR16], depending on circuit sizes and number of circuits considered. Also, as [RR16] uses the dual-execution paradigm where both parties send and receive the same amount of data, the above comparison is only meaningful assuming a full-duplex channel which might not always be available. Finally, even if the amount of data received by the evaluator in our protocol is up to ∼4x that of [RR16] in Table 7, due to the highly parallelizable nature of our independent preprocessing phase, this does not translate into equivalently lower performance as can be seen in Table 6.

For the online phase, our protocol is more data-efficient than the previous works for any of the considered settings. In particular, we require sending 16.13 kB per AES which is around 1.05-1.6x less data than [RR16] and 10.5-19x less than [LR15], depending on the number of executions considered. Furthermore if one is willing to assume a programmable random oracle, as is already the case of both [RR16] and [LR15], our online phase can easily be modified to only sending 6.30 kB (using 3 rounds) or 9.09 kB (using 2 rounds) as explained in Section 5.2.

Finally as mentioned in Section 1 and summarized in Table 1 the best reported timings for evaluating a single AES-128 is 65 ms in [WMK16]. Based on the reported numbers in their paper we estimate that ∼20 ms of the execution time consists of the initial BaseOTs. We therefore consider the actual cost of their protocol to be around 45 ms and motivate this by observing that a single computation of BaseOTs can be reused for any number of future executions using OT extension. To give as meaningful a comparison as possible we also ran our implementation on the same AWS setup (c4.2x instance) for the single execution AES-128 case measuring 105.7 ms of ind. preprocessing time, 12.07 ms dep. preprocessing time and 1.41 ms online time on a LAN. Therefore our protocol performs around 2.5x times slower than theirs in total time (when also ignoring the cost of our initial BaseOTs). However if ind. preprocessing can be applied, then from the time the circuit is input by the parties, our protocol takes around 13.48 ms to evaluate, which is around 3.5x faster than [WMK16]. We ran the same experiment in the WAN setting where we evaluate an AES-128 in 1837 ms of ind. preprocessing time, 82.51 ms dep. preprocessing time and 72.63 ms online time. As [WMK16] takes 1513 ms in total, when adjusting for initial BaseOT cost the difference is about a factor 1.5x in favor of the latter. However, when ignoring time for independent preprocessing our protocol can perform around an order of magnitude faster. We believe this difference in factors between LAN and WAN is due to our protocol having fewer rounds (when ignore our preprocessing) and our implementation fully saturating the network as it sets up multiple parallel TCP connections for maximal bandwidth utilization.

Acknowledgments

The authors would like to thank Michael Zohner for his assistance in augmenting the code of [ALSZ15] to support the randomness extraction technique used to produce Δ-ROTs. We also thank Tore Kasper Frederiksen and Thomas P. Jakobsen for sharing their initial unpublished work on implementing TinyLEGO which was used as inspiration for part of our codebase.

References

AL07. Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 137–156. Springer, February 2007.

ALSZ13. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 535–548. ACM Press, November 2013.

ALSZ15. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 673–701. Springer, April 2015.

AMPR14. Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 387–404. Springer, May 2014.

BCD+09. Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In Roger Dingledine and Philippe Golle, editors, *FC 2009*, volume 5628 of *LNCS*, pages 325–343. Springer, February 2009.

Bea95. Donald Beaver. Precomputing oblivious transfer. In Don Coppersmith, editor, *CRYPTO 1995*, volume 963 of *LNCS*, pages 97–109. Springer, August 1995.

BHK13. Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Instantiating random oracles via UCEs. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 398–415. Springer, August 2013.

BHKR13. Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society Press, May 2013.

BHR12a. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 134–153. Springer, December 2012.

BHR12b. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012.

BLN+15. Sai Sheshank Burra, Enrique Larraia, Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, Emmanuela Orsini, Peter Scholl, and Nigel P. Smart. High performance multi-party computation for binary circuits based on oblivious transfer. *IACR Cryptology ePrint Archive*, 2015:472, 2015.

BLW08. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In Sushil Jajodia and Javier López, editors, *ESORICS 2008*, volume 5283 of *LNCS*, pages 192–206. Springer, October 2008.

BOGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

Bra13. Luís T. A. N. Brandão. Secure two-party computation with reusable bit-commitments, via a cut-and-choose with forge-and-lose technique - (extended abstract). In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 441–463. Springer, December 2013.

BRC60. Raj Chandra Bose and Dwijendra K Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and control*, 3(1):68–79, 1960.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CCD88. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

CDD+16. Ignacio Cascudo, Ivan Damgård, Bernardo David, Nico Döttling, and Jesper Buus Nielsen. Rate-1, linear time and additively homomorphic UC commitments. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 179–207. Springer, 2016.

CO15. Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In Kristin E. Lauter and Francisco Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 40–58. Springer, August 2015.

DKL+13. Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, September 2013.

DLT14. Ivan Damgård, Rasmus Lauritsen, and Tomas Toft. An empirical study and some improvements of the MiniMac protocol for secure computation. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 398–415. Springer, September 2014.

DNNR16. Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. Gate-scrambling revisited - or: The TinyTable protocol for 2-party secure computation. *IACR Cryptology ePrint Archive*, 2016:695, 2016.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, August 2012.

dya. Dyadic Security. `https://www.dyadicsec.com`.

DZ13. Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 621–641. Springer, March 2013.

DZ16. Ivan Damgård and Rasmus Winther Zakarias. Fast oblivious AES A dedicated application of the MiniMac protocol. In *AFRICACRYPT 2016*, volume 9646 of *LNCS*, pages 245–264. Springer, 2016.

Ekl72. J. O. Eklundh. A fast computer method for matrix transposing. *IEEE Trans. Computers*, 21(7):801–803, 1972.

FJN+13. Tore Kasper Frederiksen, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. MiniLEGO: Efficient secure two-party computation from general assumptions. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 537–556. Springer, May 2013.

FJN14. Tore Kasper Frederiksen, Thomas P. Jakobsen, and Jesper Buus Nielsen. Faster maliciously secure two-party computation using the GPU. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 358–379. Springer, September 2014.

FJNT15. Tore Kasper Frederiksen, Thomas P. Jakobsen, Jesper Buus Nielsen, and Roberto Trifiletti. TinyLEGO: An interactive garbling scheme for maliciously secure two-party computation. *IACR Cryptology ePrint Archive*, 2015:309, 2015.

FJNT16. Tore Kasper Frederiksen, Thomas P. Jakobsen, Jesper Buus Nielsen, and Roberto Trifiletti. On the complexity of additively homomorphic UC commitments. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 542–565. Springer, January 2016.

FM16. Pooya Farshim and Arno Mittelbach. Modeling random oracles under unpredictable queries. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 453–473. Springer, March, 2016.

FN13. Tore Kasper Frederiksen and Jesper Buus Nielsen. Fast and maliciously secure two-party computation using the GPU. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 2013*, volume 7954 of *LNCS*, pages 339–356. Springer, June 2013.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

HEKM11. Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security 2011*. USENIX Association, 2011.

HKE13. Yan Huang, Jonathan Katz, and David Evans. Efficient secure two-party computation using symmetric cut-and-choose. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 18–35. Springer, August 2013.

HMsG13. Nathaniel Husted, Steven Myers, abhi shelat, and Paul Grubbs. GPU and CPU parallelization of honest-but-curious secure two-party computation. In Charles N. Payne Jr., editor, *ACSAC 2013*, pages 169–178. ACM, 2013.

Hoc59. Alexis Hocquenghem. Codes correcteurs d'erreurs. *Chiffres*, 2(147-156):8–5, 1959.

HZ15. Yan Huang and Ruiyu Zhu. Revisiting LEGOs: Optimizations, analysis, and their limit. *IACR Cryptology ePrint Archive*, 2015:1038, 2015.

KOS15. Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 724–741. Springer, August 2015.

KOS16. Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 830–842. ACM Press, Oct. 2016.

KS06. Mehmet S. Kiraz and Berry Schoenmakers. A protocol issue for the malicious case of Yao's garbled circuit construction. In *27th Symposium on Information Theory in the Benelux*, pages 283–290, 2006.

KS08. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, July 2008.

KsS12. Benjamin Kreuter, abhi shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security 2012*, pages 285–300. USENIX Association, 2012.

KSS13. Marcel Keller, Peter Scholl, and Nigel P. Smart. An architecture for practical actively secure MPC with dishonest majority. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 549–560. ACM Press, November 2013.

Lin13. Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 1–17. Springer, August 2013.

LOS14. Enrique Larraia, Emmanuela Orsini, and Nigel P. Smart. Dishonest majority multi-party computation for binary circuits. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 495–512. Springer, August 2014.

LP07. Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 52–78. Springer, May 2007.

LP09. Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

LP11. Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 329–346. Springer, March 2011.

LPS08. Yehuda Lindell, Benny Pinkas, and Nigel P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *SCN 2008*, volume 5229 of *LNCS*, pages 2–20. Springer, September 2008.

LR14. Yehuda Lindell and Ben Riva. Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 476–494. Springer, August 2014.

LR15. Yehuda Lindell and Ben Riva. Blazing fast 2PC in the offline/online setting with security for malicious adversaries. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 2015*, pages 579–590. ACM Press, October 2015.

MF06. Payman Mohassel and Matthew Franklin. Efficiency tradeoffs for malicious two-party computation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 458–473. Springer, April 2006.

MGBF14. Benjamin Mood, Debayan Gupta, Kevin R. B. Butler, and Joan Feigenbaum. Reuse it or lose it: More efficient secure computation through reuse of encrypted values. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 582–596. ACM Press, November 2014.

MR13. Payman Mohassel and Ben Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 36–53. Springer, August 2013.

NNOB12. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, August 2012.

NO09. Jesper Buus Nielsen and Claudio Orlandi. LEGO for two-party secure computation. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 368–386. Springer, March 2009.

par. Partisia. `http://www.partisia.dk`.

Ped92. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO 1991*, volume 576 of *LNCS*, pages 129–140. Springer, August 1992.

PSSW09. Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, December 2009.

PVW08. Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, August 2008.

RR16. Peter Rindal and Mike Rosulek. Faster malicious 2-party secure computation with online/offline dual execution. In *USENIX Security 2016*, pages 297–314. USENIX Association, 2016.

sep. Sepior. `https://www.sepior.com`.

sS11. abhi shelat and Chih-Hao Shen. Two-output secure computation with malicious adversaries. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 386–405. Springer, May 2011.

sS13. abhi shelat and Chih-Hao Shen. Fast two-party secure computation with minimal assumptions. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 523–534. ACM Press, November 2013.

WMK16. Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. Faster two-party computation secure against malicious adversaries in the single-execution setting. Cryptology ePrint Archive, Report 2016/762, 2016. `http://eprint.iacr.org/2016/762`.

Yao82. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS 1982*, pages 160–164. IEEE Computer Society Press, November 1982.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS 1986*, pages 162–167. IEEE Computer Society Press, October 1986.

ZB11. Hongchao Zhou and Jehoshua Bruck. Linear extractors for extracting randomness from noisy sources. In Alexander Kuleshov, Vladimir Blinovsky, and Anthony Ephremides, editors, *ISIT 2011*, pages 1738–1742. IEEE, 2011.

ZRE15. Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, April 2015.