Richer Efficiency/Security Trade-offs in 2PC

Vladimir Kolesnikov⋆, Payman Mohassel, Ben Riva, and Mike Rosulek⋆⋆

[1] Bell Labs
kolesnikov@research.bell-labs.com
[2] University of Calgary
pmohasse@cpsc.ucalgary.ca
[3] Bar-Ilan University
benr.mail@gmail.com
[4] Oregon State University
rosulekm@eecs.oregonstate.edu

Abstract. The dual-execution protocol of Mohassel & Franklin (PKC 2006) is a highly efficient (each party garbling only one circuit) 2PC protocol that achieves malicious security apart from leaking an *arbitrary, adversarially-chosen* predicate about the honest party's input. We present two practical and orthogonal approaches to improve the security of the dual-execution technique.

First, we show how to greatly restrict the predicate that an adversary can learn in the protocol, to a natural notion of "only computation leaks"-style leakage. Along the way, we identify a natural security property of garbled circuits called *property-enforcing* that may be of independent interest.

Second, we address a complementary direction of reducing the probability that the leakage occurs. We propose a new dual-execution protocol — with a very light cheating-detection phase and each party garbling $s+1$ circuits — in which a cheating party learns a bit with probability only 2^{-s}. Our concrete measurements show approximately 35% reduction in communication for the AES circuit, compared to the best combination of state of the art techniques for achieving the same security notion.

Combining the two results, we achieve a rich continuum of practical trade-offs between efficiency & security, connecting the covert, dual-execution and full-malicious guarantees.

1 Introduction

Garbled circuits were initially conceived as a technique for secure computation protocols [23]. Now they are recognized as fundamental and useful to a wide range of cryptographic applications (see the survey in [3]). By themselves, garbled circuits are generally only useful for achieving semi-honest security. For example, in the setting of two-party secure computation, malicious security costs approximately 40 times more than semi-honest security using current techniques.

⋆ Supported by the Office of Naval Research (ONR) contract N00014-14-C-0113.
⋆⋆ Supported by NSF award CCF-1149647.

Y. Dodis and J.B. Nielsen (Eds.): TCC 2015, Part I, LNCS 9014, pp. 229–259, 2015.

© International Association for Cryptologic Research 2015

The majority of this extra overhead is to mitigate the effects of adversarially crafted garbled circuits.

In many circumstances, because of the high cost of achieving full malicious security, a slight security relaxation may be acceptable, in return for performance improvements. This was the motivation for the k-leaked model of Mohassel and Franklin [17], covert security of Aumann and Lindell [2], and others. This work falls into the line of research aiming to get as much security as possible, while bringing the required resources down to, ideally, that of the semi-honest model.

Dual execution. The **dual-execution** 2PC protocol of Mohassel & Franklin [17] is a natural starting point in this line; it works as follows. The parties run two separate instances of Yao's semi-honest protocol, so that each party is the "sender" in one instance and "receiver" in the other. Each party evaluates a garbled circuit to obtain their garbled output. Then the two parties run a (much simpler and smaller) fully-secure "equality test" protocol to check whether their outputs match; each party inputs the garbled output they computed along with output wire labels of the garbled circuit they generated. If the outputs don't match, then the parties abort.

The protocol is not secure against malicious adversaries. An honest party executes an adversarially crafted garbled circuit and then uses the garbled output in the equality-test sub-protocol. However, since the equality test has only one bit of output, it can be shown that the dual-execution protocol leaks at most one (adversarially chosen) bit about the honest party's input.

1.1 Our Results

In this section we summarize our results. In Section 2, we present at the high level the main motivation, intuition and insights of our constructions, as well as put all the pieces in the unifying perspective. In the subsequent corresponding sections, we present formalizations, complete constructions, proofs and performance analysis of each individual contribution.

Our theme is to explore and reduce the leakage allowed by the dual-execution protocol. We develop new techniques for restricting the kinds of predicates that the adversary can learn, as well as for reducing the probability that the adversary succeeds in his attack. Combining the two approaches results in a more efficient continuum of cost-security trade-offs, connecting the covert, dual-execution and full-malicious guarantees.

Limiting Leakage Functions in Dual-Execution. The original security notion introduced in the dual-execution paper [17], and the follow-up [7] allows the adversary to learn an arbitrary predicate of the player's input. We show how to significanly limit this leakage to a conjunction of what we call "gate-local" queries, i.e., boolean queries that only operate on the input wires to a single gate of the original circuit. In our formalization, we follow the framework of [3] and introduce the notion of Property-Enforcing Garbling Schemes (PEGS), which may be of independent interest.

Reducing Leakage Probability in Dual-Execution. In a complementary research direction, using the ϵ-CovIDA security notion of [18], we develop new techniques for reducing the *probability* of leakage in the Dual-Execution framework. We improve on their construction by achieving 2^{-s} security with only s circuits for each party, similar to state-of-the-art results of Lindell [14] and Huang, Katz and Evans [8] from the malicious setting. However, we replace the "cheating-recovery" computation of [14] and repeated dual-execution mechanism of [8] with a much more lightweight procedure based on Private Set Intersection (PSI) that provides significant gains in computation and bandwidth. We note that the protocol of [8] has a similar high-level idea to ours: each party sends (approximately) s circuits, then the parties run a fully-secure processing phase. However, their protocol does not achieve ϵ-CovIDA security. In particular, their protocol performs separate equality checks of wire labels for each output bit of the circuit. Hence, in the event that an adversary successfully passes the circuit-check phase (with probability 2^{-s}), she can learn more than one bit.[1]

Our concrete measurements (see Figure 8) show that our techniques yield 35% reduction in overall communication for the AES circuit (compared to protocol of [18] augmented with Lindell's circuit reduction techniques).

Putting it Together: A Richer and Cheaper Security/Efficiency Trade-offs. Restricting the leakage functions a successful adversary may evaluate, and further limiting the probability of his success, allows for a fine-grained practical trade-offs between security guarantees and efficiency of 2PC. This work can be viewed as interpolating between the guarantees of covert, dual execution (i.e. the k-leaked model [17]) and fully-malicious models. Indeed, setting $s = 2$, our protocols correspond to an improved hybrid of covert and k-leaked models and protocols. We guarantee probability of 1/2 of catching the cheating adversary, and at the same time limit the leakage to an "only computation leaks" one-bit functions. On the opposite end of the spectrum, setting $s = 40$ gives fully-secure 2PC, which, while having better latency (since parties can work in parallel and due to a cheaper cheating recovery) than [14], should be seen as less efficient than [14] as it sends $2s$ total circuits. However, in the extremely important (in practice) set of security parameters/associated costs of $s \in \{1, \ldots, 20\}$, our protocols provide the best "value". Indeed, the guarantee of covert 2PC can be unacceptable in such scenarios as a successful adversary may learn the entire input of the honest party with a non-negligible probability (e.g., the set of long-term keys) making [14] less suitable, while our protocols remain attractive. Further, as noted in Section 6, the parameter s may differ between the two players to reflect different risk/trust assumptions.

1.2 Related Work

To the best of our knowledge, only a handful of prior work consider trading off security for better efficiency in the context of fully-malicious 2PC. Mohassel and

[1] Concretely, suppose Alice passes the circuit-check phase with malicious circuits that compute an arbitrary (multi-bit) function $g(x, y)$. Then Alice will learn the length of the longest common prefix of $g(x, y)$ and the correct output $f(x, y)$.

Franklin [17] introduced the notion of k-leaked model and showed through their dual-execution protocol, that leaking a single bit of information can yield major improvement in efficiency. The follow-up work of [7] implemented/enhanced their protocol, confirming the efficiency gains. The work of [9] also considers leakage of information with the goal of designing more efficient non-interactive secure computation protocols. In fact, they also propose a construction where the leakage function is restricted to disjunction of intermediate wire values in the computation. However, what mainly separates our construction from theirs is that we focus on concrete efficiency and small constant factors for fast implementation, while their work is focused on optimal asymptotic complexity, and results in a construction with noticeably larger constant factors (to the best of our knowledge the exact constants have not been worked out).

In a complementary direction, Aumann and Lindell [2] introduced the notion of covert security where one trades the probability of deterring malicious behavior (i.e., making the probability non-negligible) for more efficient 2PC. The recent work of [18] introduces the notion of ϵ-CovIDA security which can be seen as a strengthening of both the covert and the k-leaked models for 2PC. We adopt their security definition for reducing probability of leakage in dual execution.

Application specifics and hard performance requirements sometimes drive the trade-offs. In recent works on practical private DB [10,19] some of the execution patterns are revealed to the adversary as a trade-off for efficient sublinear execution time. Similarly, in the setting of searchable encryption (e.g., [5]), access patterns can often be leaked, yielding significant improvements in performance.

Besides allowing a bit of leakage, other methods have been suggested for relaxing standard security guarantees: input-indistinguishable MPC [15] that allows for better composability, one-sided/two-sided non-simulatability (e.g. see [1]), superpolynomial-time simulation [20,21,4], and security against uniform adversaries [12].

2 Overview of Our Approach and Constructions

Following our focus — reducing leakage power and probability in dual-execution, — we now go a little deeper into each of our results and present their intuition and insights.

2.1 Limiting Leakage Functions in Dual-Execution

Garbled circuits & property-enforcement. It is clear that an adversary can learn an *arbitrary* predicate in the dual-execution protocol, as long as the honest party evaluates *any* garbled circuit given by the other party. To limit leakage in any way therefore requires two things:

1. The parties must perform some check of the garbled circuit they receive in the protocol. In the extreme, parties could demand a zero-knowledge proof of correctness of the garbled circuit, but in that case dual-execution is not even

needed — the protocol would already be fully-secure in the malicious setting. In our setting it makes sense to consider only lightweight checks on the garbled circuits. In their treatment of the dual-execution protocol, Huang, Katz and Evans [7] mention briefly that a party could perform a “sanity check” of the garbled circuit it received. At the same time, most simple checks, such as verifying the circuit topology and XOR gates placement, seem helpful, but bring little guarantees and can exclude few leakage functions due to a simple attack we describe below.

2. Since the checks on the garbled circuit cannot guarantee complete correctness, we are still in a setting involving possibly malicious garbled circuits. Hence, we need some way of reasoning about what information is leaked when a honest party evaluates a malicious garbled circuit. This natural problem has not been investigated before, to the best of our knowledge. Previous work considered all-or-nothing security from a garbled circuit (i.e., either it is correct or not). We suspect that our conceptual approach regarding malicious garbled circuits may be useful in many other settings, given how ubiquitous and powerful the garbled circuit technique has become throughout cryptography.

To address these needs, we introduce a new security notion for garbled circuits called **property-enforcing** garbling schemes (PEGS). Roughly speaking, in a property-enfocring garbling scheme an honest user can locally verify that a garbled circuit F indeed computes a function with a certain property.

More formally, let prop be some property of (plain) circuits: size, topology, the circuit itself, etc. A property-enforcing garbling scheme has additional procedures Prop and $\mathsf{Extract}$. We require that, for all possibly malicious garbled circuits F, if $\mathsf{Extract}(F) \to f$ (where f is a plain circuit) then (1) $\mathsf{Prop}(F) = \mathsf{prop}(f)$ and (2) F produces garbled outputs in direct correspondence with the output of f. That is, the logic of F is “explained by” a plain circuit f with $\mathsf{prop}(f) = \mathsf{Prop}(F)$. See Section 3.2 for the formal definitions. In our actual definition, $\mathsf{Extract}$ requires extra information typically only available to the simulator, whereas Prop can be computed publicly.

Suppose we use a property-enforcing garbling scheme in the dual-execution protocol. Both parties would ensure that $\mathsf{Prop}(F) = \mathsf{prop}(f)$ for the garbled circuit F they receive and the objective function f that is being computed. We show that with this modification, the adversary cannot learn *arbitrary* predicates of the honest party’s input, but rather only predicates (roughly) of the form $\tilde{f}(x) \stackrel{?}{=} c$ where $\mathsf{prop}(\tilde{f}) = \mathsf{prop}(f)$.

Achieving topology-enforcement. Intuitively, it seems that classical/standard garbling schemes already give the receiver some guarantees along the lines of property-enforcement. An honest party seems to *enforce* the circuit’s topology in how it evaluates the garbled circuit; hence, standard garbling schemes should be topology-enforcing at the least. Interestingly, this is not quite the case.

Imagine a classical garbled circuit for a single gate. This garbled circuit consists of four ciphertexts:

$$\mathsf{Enc}_{A_0,B_0}(C_1) \qquad \mathsf{Enc}_{A_0,B_1}(C_2) \qquad \mathsf{Enc}_{A_1,B_0}(C_3) \qquad \mathsf{Enc}_{A_1,B_1}(C_4)$$

Here A_0, A_1, B_0, B_1 are wire labels of input wires. The garbler is supposed to choose $C_1, \ldots, C_4$ from among two possiblities (i.e., the two wire labels of the output wire). Yet there is no way for the evaluator to check that these four ciphertexts encode only two values. It is trivial to let $C_1, \ldots, C_4$ be distinct. In that case, the garbled output of this circuit reveals the entire input. The behavior of this garbled circuit cannot be "explained" by a single-gate circuit, so the scheme is not topology-enforcing.

Our intuition about standard/classical garbling schemes is thus not quite right, but it is not far off either. These schemes do enforce topology in some sense, but they do not enforce the "information bandwidth" on each wire. In an extreme example, one can make a garbled circuit with just one output wire, but whose garbled output reveals the entire input (in the sense that all distinct inputs give different garbled outputs).

To achieve topology-enforcement in a more reasonable sense (i.e., the behavior of a malicious garbled circuit can always be described by a *boolean* circuit of the advertised topology), it suffices to simply limit the wire bandwidth to 2.[2] In our construction, the sender includes a hash of the two wire labels on each wire. When evaluating a garbled circuit, the receiver checks its wire labels at each step against these hashes. We prove that this construction enforces the topology of the circuit, when the hash function is modeled as a random oracle. Furthermore, the construction remains very practical.

Only computation leaks. "Only computation leaks" (OCL) [16] refers to a paradigm for defining information leakage. In short, OCL means that an adversary cannot leak jointly on two internal values x and y unless they are both computed on simultaneously at some point. Armed with topology-enforcing garbling schemes, we are able to restrict leakage in the dual-execution protocol to OCL-style leakage at the level of gates in the circuit.

More precisely, say that a query is *gate-local* if the query can be expressed as a function of the two input wires to some *single gate* in the circuit. We are able to restrict the dual-execution adversary to learn only a *conjunction* of gate-local queries (with respect to the function f being computed).

The main idea in our construction is as follows. Dual-execution allows parties to (roughly) check the equality of *outputs* of their garbled circuits. To keep a malicious circuit from "building up" a complicated leakage expression (i.e., more complicated than a gate-local query), we try to apply dual execution to check equality of *all* intermediate values in the computation. Hence, we modify the original circuit so that every intermediate wire is secret shared, with each party

[2] Actually, we can only limit the wires to contain 0, 1, or an error, where all errors are guaranteed to propogate forward. This turns out to be sufficient for the dual-execution protocol.

receiving one of the shares. The parties then use the dual-execution mechanism to ensure that their shares (hence, the intermediate values of the computation) all agree.

Formalizing and proving this intuition requires some care, and we also need to extend the dual-execution paradigm to cope with outputs known to only one of the parties.

Overall, our modifications to the dual-execution protocol — adding topology-enforcement to the garbling scheme, and adding secret-sharing gadgets to the ciruict — remain quite practical. The resulting protocol has very limited 1-bit leakage but is still much less expensive than fully malicious-secure 2PC. Exploring the continuum of trade-offs between plain dual-execution and full security is an interesting direction, which we address in combination with our next contribution.

2.2 Reducing Leakage Probability in Dual-Execution

As discussed earlier, an alternative to restricting the leakage function is to restrict the probability of occurrence of leakage. This is indeed the notion of ϵ-CovIDA Security recently introduced in [18] which augments the notion of Covert Security of [2]. Essentially, this notion requires that if a player is trying to cheat, the other players can catch him with probability $1-\epsilon$, but even if he is not caught (i.e., with probability ϵ) the cheater can only learn *a single bit* of extra information about the other players' inputs, and the correctness of the output is still guaranteed. In other words, the leakage of the single bit of information only occurs with probability ϵ. The 2^{-s}-CovIDA security is particularly attractive for low and medium values of s (e.g. $1 \leq s \leq 20$) since it provides a *much* stronger guarantee than covert 2PC, and is at the same time more efficient than fully-malicious 2PC where $s \geq 40$.

[18] presents two protocols that are secure in this model, requiring about $3s$ garbled circuits from each player (a total of $6s$) to obtain 2^{-s}-CovIDA security. We observe that it is possible to combine their dual-execution approach with the underlying ideas of Lindell [14], in order to obtain a 2^{-s}-CovIDA 2PC protocol using only $2s$ garbled circuits, as opposed to $6s$ of [18]. The main observation that makes this possible is that the garbler's input consistency check of [22] can be extended to enforce equality of a party's input not only in the circuits he garbles but also in those garbled by his counterparts. However, simply running [14] as a dual execution still results in high overhead as it requires each party to execute a relatively expensive "cheating recovery" phase. We note that we are not aware of the above observation having been published elsewhere, but we consider it a natural combination of ideas in [14] and [18], and the input-consistency check technique of [22].

Protocol Overview. We propose a new approach for designing a 2^{-s}-CovIDA 2PC, wherein we replace the cheating recovery phase with a private set intersection protocol on the outputs. The high level idea of the protocol is as follows. We follow the insight of Lindell [14] for achieving 2^{-s} security with s circuits, namely that

cheater only can cheat if *all* of the evaluated circuits are incorrect, and not just the majority. At the same time we avoid the expensive cheating-punishment setup and execution. So, our Alice and Bob perform a simplified version of the protocol of Lindell [14], where they skip all the steps associated with the cheating recovery. Instead, Alice and Bob then switch roles (i.e. Bob becomes the garbler) and perform the same steps *à la* dual execution. All the circuits generated by each party have the same output labels, and we use the universal hashing circuit of [22] in both sets of circuits in order to enforce equality of a player's inputs not only in the circuits he generated but also those generated by his counterpart. There are two points to consider when using the universal hashing approach in the two executions. First, we need to use the same hash function in both sets of executions, and second, we need to commit both the garbler and the evaluator to their inputs before generating the hash function (in standard 2PC, only the garbler needed to commit to his input). To achieve the latter, the garbler commits to his garbled inputs, and parties execute the oblivious transfer for the evaluator's input before choosing the hash function at random.

Let's denote the output labels for the s circuits created by Alice (resp. by Bob) by $(\mathsf{out}^1_{A,0}, \mathsf{out}^1_{A,1}), \ldots, (\mathsf{out}^m_{A,0}, \mathsf{out}^m_{A,1})$ (resp. $(\mathsf{out}^1_{B,0}, \mathsf{out}^1_{B,1}), \ldots, (\mathsf{out}^m_{B,0}, \mathsf{out}^m_{B,1})$), where m is the number of output wires in the circuit. At the end of the two executions up to the opening phase, Alice and Bob each create initially empty sets T_A and T_B. For every circuit evaluated by Alice, if the output is valid and equal to, say, $z^A = (z^A_1, \ldots, z^A_m)$, Alice computes $q = \mathsf{out}^1_{A,z^A_1} \oplus \mathsf{out}^1_{B,z^A_1} \oplus \cdots \oplus \mathsf{out}^m_{A,z^A_m} \oplus \mathsf{out}^m_{B,z^A_m}$, and lets $T_A = T_A \cup \{q\}$. Bob does a symmetric computation. Each party then adds enough dummy random values to its set until its size is the same as the number of evaluated circuits (note that the expected size of T_A will be $s/2$, the size of the evaluation set).

The idea is to have the parties run a fully secure two-party private set intersection (PSI) protocol computing $T_A \cap T_B$. If the intersection is empty each party aborts, and otherwise, it uses the translation table to compute the final output from the labels in the intersection (note that the intersection can at most be of size one, since the circuits created by the honest party all evaluate to the same correct output). The intuition is that as long as the malicious party did not cheat for just one of the garbled circuits he created, the correct output of that circuit will be the unique value in $T_A \cap T_B$.

There are several issues to resolve for this approach to work: if we perform the PSI before the opening/checking phase, then the output of the PSI can leak extra information to a malicious party. For example, this leakage can happen with probability one in case of a malicious party that garbles the same bad circuit s times, while we want to reduce the probability of leakage to 2^{-s}. If we perform the PSI after the opening phase, on the other hand, we fix the above. But we encounter a different problem, that at the end of the opening, the output labels are revealed, and this allows a malicious party to modify his input to the PSI and hence trick the honest party to learn an incorrect output. We address this dilemma by using a two-stage PSI (see Section 6.2) where in the first stage parties commit their input sets but learn nothing while in the second stage,

one of the parties learn the intersection. We then perform the first stage of the PSI before the opening phase, while postponing the second stage until after the openings.

This almost works except that all existing PSI protocol with security against malicious adversaries only let one of the parties learn the intersection. Simply sending the result to the other party is not secure since a malicious party can lie and provide a wrong answer (note that since this step takes place after the opening lying about the output is not hard). We solve this problem by having each party randomly permute its set and commit to each element in the permuted set, before the two-stage PSI is invoked. Then, in the final stage of exchanging the output, in order to prove to the other party that the output is correct (i.e. the output of one of the evaluated circuits), each party also opens the commitment to the PSI input corresponding to the intersection (note that these commitments where issued before the opening phase when it was not possible to forge any output value not returned by the evaluated circuits).

The intuition behind 2^{-s}-CovIDA security of the protocol is that with probability $1 - 2^{-s}$, at least one of the outputs evaluated by the honest party is "the correct output" and hence included in his set. On the other hand, before the opening phase, the malicious party only learns the output labels for the correct output and hence can only commit to the correct output in the first stage of the PSI. Hence, with probability $1 - 2^{-s}$, either the honest party aborts, or the computed intersection of the two sets will be the correct output and among those inputs that parties committed to, before the opening phase.

With probability 2^{-s}, however, a malicious party can cheat in all the evaluated circuits and not get caught. In this scenario, whether the output of the intersection is empty or not leaks one bit of additional information to the malicious player. In either case, the correctness is guaranteed since the honest party cannot be tricked into accepting an incorrect output.

3 Property-Enforcing Garbling Schemes

3.1 Garbling Schemes

Bellare, Hoang, and Rogaway [3] introduce the notion of a garbling scheme as a cryptographic primitive. We refer the reader to their work for a complete treatment and give a brief summary here.[3] A garbling scheme consists of the following algorithms: Garble takes a circuit f as input and outputs (F, e, d) where F is a garbled circuit, e is encoding information, and d is decoding information. Encode takes an input x and encoding information e and outputs a garbled input X. Eval takes a garbled circuit F and garbled input X and outputs a garbled output Y. Finally, Decode takes a garbled output Y and decoding information d and outputs a plain circuit-output (or an error $\perp$).

[3] Their definitions apply to any kind of garbling, but we specify the notation for *circuit* garbling.

3.2 Property Enforcing

We extend the definition of garbling schemes as follows. Let prop be a property of circuits; e.g., $\mathsf{prop}(f)$ might output the topology of a circuit f.

A garbling scheme is prop-enforcing if it meets the following additional requirements:

- The property prop is extended to *garbled* circuits. That is, when F is a garbled circuit, anyone can publicly compute a value $\mathsf{Prop}(F)$.
- There is a deterministic procedure $\mathsf{Extract}$ that can "explain" any (possibly adversarially generated) garbled circuit F as a plain circuit f' satisfying $\mathsf{prop}(f') = \mathsf{Prop}(F)$.

More formally, $\mathsf{Extract}(F, e)$ either outputs $\perp$ or a pair (f', d') where d' is a simple mapping of values to wire labels. We define the following security game:

<u>Initialize:</u> $b \leftarrow \{0, 1\}$ <u>Finalize(b'):</u> return $b \stackrel{?}{=} b'$	<u>Query(F, e, x):</u> $(f', d') \leftarrow \mathsf{Extract}(F, e)$ if $b = 0$ then $\quad Y := \mathsf{Eval}(F, \mathsf{Encode}(e, x))$ else $\tilde{y} := f'(x)$ $\quad Y := d'_{1,\tilde{y}_1} \| \cdots \| d'_{m,\tilde{y}_m}$ return Y

Hence, the garbled output Y contains no more information about x than $f'(x)$, a circuit satisfying property $\mathsf{Prop}(F)$.

3.3 Construction: Enforcing Topology

Definition 1. *A* **circuit-with-abort** *is a standard circuit with ternary values* $\{0, 1, \perp\}$ *on the wires, where* $\perp$ *values cascade. That is, for every gate* G *in the circuit,* $G(\perp, \cdot) = G(\cdot, \perp) = \perp$.

Throughout this section, we use the term "circuit" to refer to circuits with abort.

Construction. For simplicity we make minimal additions to the "Garble2" construction of [3]. Our modification to achieve property-enforcement is rather simple. In Garble2, each wire i is associated with two wire labels X_i^0 and X_i^1. The garbled circuit then simply contains the values $C[i, lsb(X_i^0)] = H(X_i^0)$ and $C[i, lsb(X_i^1)] = H(X_i^1)$, where H is a random oracle. It is straightforward that these new values do not compromise the standard security properties.

When evaluating a gate g, the evaluator obtains a visible wire label X_g, and now checks whether it is valid. By *valid*, we mean that $H(X_g) = C[g, b]$, where b is the select bit of wire label X_g. If this is not the case, then the evaluator aborts.

The $\mathsf{Extract}$ procedure maintains the invariant that there exist at most two valid wire labels for each wire. This is true for the input wires by definition.

Provided that the invariant is true for the input wires of a gate, there are at most 4 wire label combinations that Extract needs to try to extract the logic of this gate. If there are more than 2 valid output wire labels for this gate, then we have obtained an explicit collision under H: an event that happens only with negligible probability. Otherwise, the invariant holds at this gate as well.

Note that in the unmodified Garble2 scheme, the number of possible wire labels can be made to grow exponentially at each level of the circuit. Hence, the garbled values can encode more than 1 bit of information on a wire. The key idea here is to limit the "bandwidth" of each wire to a single bit.

The full details of our construction are provided in Figure 1.

```
Garble(1^k, f):
   (n, m, q, A', B', G) ← f
   for i ∈ {1, ..., n+q} do
        t ←$ {0,1};   X_i^0 ←$ {0,1}^{k-1} t
        X_i^1 ←$ {0,1}^{k-1} t̄
 ⋆      C[i,t] ← H(X_i^0);   C[i,t̄] ← H(X_i^1)
   for (g,i,j) ∈ {n+1, ..., n+q} × {0,1} × {0,1}
        a ← A'(g);   b ← B'(g)
        A ← X_a^i;   a ← lsb(A);   B ← X_b^j
        b ← lsb(B)
        T ← g||a||b;   P[g,a,b] ← E_{A,B}^T(X_g^{G_g(i,j)})
 ⋆ F ← (n, m, q, A', B', P, C)
   e ← (X_1^0, X_1^1, ..., X_n^0, X_n^1)
   d ← (X_{n+q-m+1}^0, X_{n+q-m+1}^1, ..., X_{n+q}^0, X_{n+q}^1)
   return (F, e, d)

Extract(F, e):
   (n, m, q, A', B', P, C) ← F
   (X_1^0, X_1^1, ..., X_n^0, X_n^1) ← e
   for (g,i,j) ∈ {n+1, ..., n+q} × {0,1} × {0,1}
        a ← A'(g);   b ← B'(g)
        skip loop iteration if X_a^i or X_b^j undefined
        A ← X_a^i;   a ← lsb(A);   B ← X_b^j
        b ← lsb(B)
        X̃ ← D_{A,B}^T(P[g,a,b]);      x ← lsb(X̃)
        if C[g,x] = H(X̃) then:
             if X_g^x already defined then return ⊥
             X_g^x ← X̃;   G_g(a,b) ← x
        else G_g(a,b) ← ⊥
   d' ← (X_{n+q-m+1}^0, X_{n+q-m+1}^1, ..., X_{n+q}^0, X_{n+q}^1)
   f' ← (n, m, q, A', B', G)
   return (f', d')

Encode(e, x):
   (X_1^0, X_1^1, ..., X_n^0, X_n^1) ← e
   x_1 ··· x_n ← x
   X ← (X_1^{x_1}, ..., X_n^{x_n})

Decode(d, Y):
   (Y_1, ..., Y_m) ← Y
   (Y_1^0, Y_1^1, ..., Y_m^0, Y_m^1) ← d
   for i ∈ {1, ..., m} do:
        if Y_i = Y_i^0 then y_i ← 0
        else if Y_i = Y_i^1 then y_i ← 1
        else return ⊥
   return y ← y_1 ··· y_m

Eval(F, X):
 ⋆ (n, m, q, A', B', P, C) ← F
   (X_1, ..., X_n) ← X
   for g ← n+1 to n+q do:
        a ← A'(g), b ← B'(g)
        A ← X_a;   a ← lsb(A)
        B ← X_b;   b ← lsb(B)
        T ← g||a||b
        X_g ← D_{A,B}^T(P[g,a,b])
 ⋆      if H(X_g) ≠ C[g, lsb(X_g)]:
 ⋆           return ⊥
   return (X_{n+q-m+1}, ..., X_{n+q})
```

Fig. 1. Topology-enforcing garbling scheme construction. H denotes a random oracle, and E denotes a dual-key cipher, following [3]. "$\star$" denotes differences from the Garble2 construction of [3] (the entire Extract procedure is new).

Theorem 1. *The construction in Figure 1 is a secure garbling scheme (in random oracle model) satisfying privacy, authenticity, obliviousness, and* prop*-enforcement, when* prop *denotes the topology of the circuit.*

Proof. We focus on the proof of prop-enforcement. First, observe that Extract can output $\perp$ with only negligible probability, since it only outputs $\perp$ when it explicitly finds a collision under the random oracle H. But, as we will argue, the two branches of the security game are identical except for the possibility of Extract outputting $\perp$.

Extract works by identifying at most 2 "valid" wire labels X_g^0, X_g^1 for each wire g. For input wires, these valid wire labels are given as the encoding information e. For a gate g with input wires i & j, Extract produces a gate with logic G_g such that evaluating the corresponding gate in F with wire labels X_g^a, X_g^b yields $X_g^{G_g(a,b)}$ (or $\perp$ if $G_g(a,b) = \perp$). Hence it follows by induction that the output of $\mathsf{Eval}(F, \mathsf{Encode}(e, x))$ is exactly characterized by the choice of wire labels $f'(x)$.

4 Applications to Dual Execution

In this section, we write a functionality as $f(x_A, x_B) = (y_A, y_B, y_{AB})$ where y_A, y_B, y_{AB} denote outputs for Alice only, Bob only, and both parties, respectively. We must augment the existing dual-execution protocol and proofs of [17,7] to account for functionalities that give different inputs to the two parties.

$\mathcal{L}$-leaked model. In the $\mathcal{L}$-leaked model for computing function f, where $\mathcal{L} = (\mathcal{L}^A, \mathcal{L}^B)$, Alice provides input x_A and Bob provides input x_B to the functionality. The functionality then computes $(y_A, y_B, y_{AB}) \leftarrow f(x_A, x_B)$. Let (y_A, y_{AB}) denote Alice's potential output and let (y_B, y_{AB}) denote Bob's potential output..

The functionality delivers the corrupt party's potential output. If Alice is corrupt, then the adversary supplies a leakage function $L \in \mathcal{L}^A$ to the functionality. If Bob is corrupt, the adversary supplies a leakage function $L \in \mathcal{L}^B$. The functionality evaluates $\ell = L(x_A, x_B)$. If $\ell = 0$ then the functionality delivers $\perp$ to the honest party; otherwise waits for instruction from the adversary before delivering the honest party's potential output.

Dual execution protocol. Given a functionality f, we let $f(\cdot, x)$ and $f(x, \cdot)$ denote residual circuits with one input hard-coded. We assume that for all possible inputs x we have $\mathsf{prop}(f(x, \cdot)) = \mathsf{prop}(f(0^n, \cdot))$, and $\mathsf{prop}(f(\cdot, x)) = \mathsf{prop}(f(\cdot, 0^n))$.

Given a functionality f a garbling scheme $\mathcal{G}$ we define the dual execution protocol $\mathsf{DualEx}^f[\mathcal{G}]$ as follows:

1. Alice has input x_A and Bob has input x_B. Alice does $(F_A, e_A, d_A) \leftarrow \mathsf{Garble}(f(x_A, \cdot))$. Bob similarly does $(F_B, e_B, d_B) \leftarrow \mathsf{Garble}(f(\cdot, x_B))$.
2. Alice commits to F_A, Bob commits to F_B (if the garbling scheme is adaptively secure then the garbled circuits can be sent in the clear here).
3. Using instances of OT, Alice acts as sender with inputs e_A and Bob acts as receiver with input x_B. Bob receives garbled input X_B. Likewise, the parties use OT for Alice to obtain X_A.

4. The parties open their commitments to the garbled circuits. Alice aborts if $\mathsf{Prop}(F_B) \neq \mathsf{prop}(f(\cdot, 0^n))$. Similarly Bob aborts if $\mathsf{Prop}(F_A) \neq \mathsf{prop}(f(0^n, \cdot))$.
5. Alice does $(Y_A, Y_B, Y_{AB}) = \mathsf{Eval}(F_B, X_A)$. Bob also computes values (Y_A, Y_B, Y_{AB}). If either party obtains $\perp$ from executing Eval, then it continues below using randomly chosen values for these garbled outputs (Y_A, Y_B, Y_{AB}).
6. Alice can decode Y_A and Y_{AB} to obtain plain outputs y_A and y_{AB}. She can use d_A to compute $\tilde{Y}_A$, $\tilde{Y}_{AB}$, which are garbled output encodings of y_A and y_{AB} according to d_A. She sends $C = \tilde{Y}_A \| Y_B \| Y_{AB} \| \tilde{Y}_{AB}$ to the equality test functionality.
7. Similarly, Bob can decode Y_B and Y_{AB} to obtain plain outputs y_B and y_{AB}. He computes $\tilde{Y}_B$ and $\tilde{Y}_{AB}$, garbled output encodings of these values according to d_B. He sends $C = Y_A \| \tilde{Y}_B \| \tilde{Y}_{AB} \| Y_{AB}$ to the equality test functionality.
8. If the equality test returns false, then the parties abort; otherwise Alice outputs (y_A, y_{AB}) and Bob outputs (y_B, y_{AB}).

Leakage functions. Let $f(x_A, x_B) = (y_A, y_B, y_{AB})$ be a 2-party functionality as above. Define:

$$L^A_{f,f',\tilde{y}}(x_A, x_B) = \begin{cases} 1 & \text{if } f'(x_B) = (\tilde{y}, y_B, y_{AB}), \text{ where } (y_A, y_B, y_{AB}) \leftarrow f(x_A, x_B) \\ 0 & \text{otherwise} \end{cases}$$

$$L^B_{f,f',\tilde{y}}(x_A, x_B) = \begin{cases} 1 & \text{if } f'(x_A) = (y_A, \tilde{y}, y_{AB}), \text{ where } (y_A, y_B, y_{AB}) \leftarrow f(x_A, x_B) \\ 0 & \text{otherwise} \end{cases}$$

Then define $\mathcal{L}^f_{\mathsf{prop}} = ((\mathcal{L}^f_{\mathsf{prop}})^A, (\mathcal{L}^f_{\mathsf{prop}})^B)$, where:

$$(\mathcal{L}^f_{\mathsf{prop}})^A = \{L^A_{f,f',\tilde{y}} \mid \mathsf{prop}(f') = \mathsf{prop}(f(0^n, \cdot)) \text{ and } \tilde{y} \in \{0,1\}^*\}$$
$$(\mathcal{L}^f_{\mathsf{prop}})^B = \{L^B_{f,f',\tilde{y}} \mid \mathsf{prop}(f') = \mathsf{prop}(f(\cdot, 0^n)) \text{ and } \tilde{y} \in \{0,1\}^*\}$$

Intuitively, the $\mathcal{L}^f_{\mathsf{prop}}$-leaked model allows the adversary to choose a circuit f' such that $\mathsf{prop}(f')$ has the "expected value" and learn whether f', when evaluated on the honest party's input, equals $f(x_A, x_B)$. In addition, for the output of f' that is not revealed to the honest party, the adversary can check that this output of f' is any fixed value of the adversary's choice.

Recall that f' may be a circuit with abort. In the case that f' aborts, these leakage functions will always return 0 (since the main equality condition will not hold).

Security. In Appendix A we prove the following:

Theorem 2. *The dual-execution protocol* $\mathsf{DualEx}^f[\mathcal{G}]$ *is secure in the* $\mathcal{L}^f_{\mathsf{prop}}$*-leaked model when* $\mathcal{G}$ *satisfies* prop*-enforcing, authenticity, and privacy/obliviousness.*

We point out that our simulator chooses f' before seeing the output of the functionality, and can choose only $\tilde{y}$ after seeing the output. Hence, one could slightly strengthen the definition of the $\mathcal{L}^f_{\mathsf{prop}}$-leaked model. For simplicity, we choose not to.

5 Achieving "Only Computation Leaks" with Dual Execution

Micali & Reyzin [16] proposed a model of leakage, one of whose axioms was that *"computation, and only computation, leaks information"* ("only computation leaks", or OCL, for short). One can think of decomposing a large computation into smaller "atomic" steps. Each step does not use all of the information in the system. The OCL axiom restricts us to leakage that is a function of the information used in a single atomic step. If two values are never used in the same step, then OCL leakage precludes (directly) leaking a *joint* function of those two values.

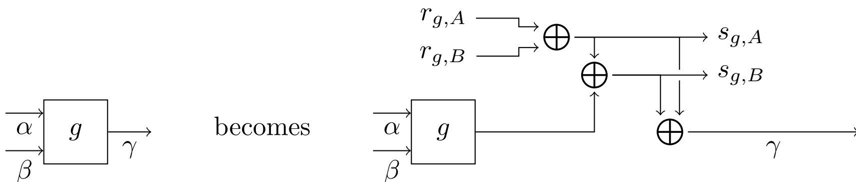
In a circuit model, the smallest "atomic" steps are gates. Hence, we consider leakage on the information available to a single gate (i.e., its two input wire values). By extension, it is natural to consider leaking on the information available to several gates, but only *separately* and not *jointly.* Only when two wires are inputs to a common gate can the leakage be a *joint* function of those two wires' values.

We formalize this kind of leakage as follows:

Gate-local queries. Let f be a circuit. We say that a leakage query $L(x_A, x_B)$ is *gate-local* if there exists a gate g in f such that $L(x_A, x_B)$ can be expressed as a function of the input wires of g in the computation of $f(x_A, x_B)$. We define $\mathcal{L}^f_{\mathsf{ocl}}$ to be the set of conjunctions of gate-local queries; that is:

$$\mathcal{L}^f_{\mathsf{ocl}} = \{L = L_1 \wedge \cdots \wedge L_k \mid \text{each of } L_1, \ldots, L_k \text{ is gate-local for } f\}$$

Circuit transformation. Let f be a circuit and define $\widehat{f}$ as follows: For each gate g in f, we add input bits $r_{g,A}$ for Alice and $r_{g,B}$ for Bob. We add output bits $s_{g,A}$ for Alice only and $s_{g,B}$ for Bob only. We then perform the following transformation for each gate:



Intuitively, we additively secret share the output of g into shares $s_{g,A}$ and $s_{g,B}$, so that each party learns one share. Then the shares are re-assembled to again form the output of g that is used elsewhere in the circuit.

In Appendix B we prove the following:

Theorem 3. *Let* prop *be a property that includes the circuit topology and let* f*,* $\widehat{f}$ *be as above. We define* Π *to be a protocol for* f *in the* $\widehat{f}$*-hybrid,* $\mathcal{L}^{\widehat{f}}_{\mathsf{prop}}$*-leaked model. In* Π*, parties simply send their inputs along with random values*

for $\{r_{g,A}, r_{g,B}\}_g$ *to the ideal* $\widehat{f}$. *Then* Π *is a secure realization of* f *in the* $\mathcal{L}^f_{\mathsf{ocl}}$-*leaked model.*

More concretely, if we would like to compute f restricting adversaries to $\mathcal{L}^f_{\mathsf{ocl}}$ leakage, then we need to simply run the dual execution protocol on $\widehat{f}$ with a topology-enforcing garbling scheme.

Generalizations. One can also define "only computation leaks" at a higher level than individual gates. Indeed, this is more in line with most work on OCL, which does not consider circuit computations. Also, doing so leads to efficiency improvements in our protocols and transformations.

Consider partitioning a circuit into well-defined *components*. Then a *component-level* leakage query is one that can be written as a function of the inputs to a single component. Finally, let $\mathcal{L}$ denote the set of all conjunctions of component-level functions.

Then our results of the previous section can be easily applied to yield a protocol that is secure in the $\mathcal{L}$-leaked model.

We sketch the important differences:

- One would need a garbling scheme which enforces only the topology connecting of *components*, but need not enforce anything about the internals of each component. Our (gate-)topology-enforcing construction of Section 3.3 adds two hashes to each wire. To preserve topology of components, one need only add these hashes to the wires connecting different components. Concretely, this may result in a significantly smaller overhead than gate-topology-enforcement.
- Recall our transformation from a circuit f to a circuit $\widehat{f}$. It replaces each gate g with some gadget of 4 gates. However, the construction and proof go through verbatim with respect to *components*, if one interprets our diagram to let g be a larger component, each line to represent a *bundle* of wires, and the XOR gates to be string-XOR gates.
  Concretely, instead of adding 3 XOR gates for each gate, we add 3 XOR gates for each wire connecting different components. Similarly, the number of additional inputs/outputs is related to the number of "component-connecting wires", not the total size of the circuit.

6 Reducing the Probability of Leakage in Dual Execution

In Section 2.2 we gave a high level overview of our ϵ-CovIDA protocol. Here, we describe the protocol in detail, and evaluate its asymptotic and concrete efficiency. We start with a brief review of the two sub-protocols we use, i.e., committing-OTs and two-stage PSI. (For completeness, we provide a formal definition of CovIDA security in Appendix C.)

6.1 Committing-OTs

Oblivious Transfer (OT) protocol implements securely the following functionality; A sender inputs two tuples $[K_1^0, K_2^0, \ldots, K_s^0], [K_1^1, K_2^1, \ldots, K_s^1]$ and a receiver inputs a bit b. Then, the receiver learns the tuple $[K_1^b, K_2^b, \ldots, K_s^b]$.

A stronger variant of OT, called committing-OT, is one in which the sender is also *committed* to his inputs, meaning, the sender cannot claim in retrospect that his inputs were different than the ones he entered to the OT in the beginning. In other words, if the sender is asked to show what was his inputs $[K_1^0, K_2^0, \ldots, K_s^0], [K_1^1, K_2^1, \ldots, K_s^1]$, he cannot answer with different inputs without being caught.

For simplicity, from now on we abstract out the details of the committing-OT and just work with the following notation: We denote by $\mathrm{COT}_1(b)$ the message sent by the receiver to the sender (where b is the receiver's input bit) and similarly use $\mathrm{COT}_2([K_1^0, K_2^0, \ldots, K_s^0]\ , [K_1^1, K_2^1, \ldots, K_s^1],\ \mathrm{COT}_1(b))$ to denote the message sent by the sender to the receiver. When we say that the sender *decommits* his input, we refer to the operation in which he reveals $[K_1^0, K_2^0, \ldots, K_s^0], [K_1^1, K_2^1, \ldots, K_s^1]$ and proves that these are the correct inputs he had used in the protocol.

Committing-OTs can be realized in several ways, and even be efficiently extended for specific implementations (see [18]). Throughout this work we will assume that the cost of committing-OT is $\mathcal{O}(s)$ exponentiations. (The exact constant can be computed as done in [14], but we prefer stating our efficiency claims for general committing-OT constructions.)

6.2 Two-Stage Private Set Intersection

In standard two-party private set intersection (PSI), player P_i holds his input set S_i and the goal is for one or both parties to learn the intersection $S_1 \cap S_2$.

The two-stage variant of PSI, which we denote by the functionality $\mathcal{F}_{2PSI}$, is split the protocol to two stages in order to emulate a commitment on the inputs before revealing the result (we will use this property in our constructions). I.e., in the first stage, players submit their input sets and learn nothing, and in the second stage, one of the parties ask for the output and obtains the result. The reason we formalize the functionality with only one party receiving the output is that all of the realizations we are aware of are this form. In our constructions we need both parties to learn the intersection, thus we address this issue directly as part of our 2PC constructions. We define the functionality $\mathcal{F}_{2PSI}$ in Figure 6.2.

Realizing two-stage PSI. There are several two-round, fully-simulatable PSI with security against malicious adversaries in the literature (e.g. see [11,6], both in the random oracle model). These protocols do not automatically realize $\mathcal{F}_{2PSI}$ but can be modified at little cost to do so. In particular, two properties that are shared by these constructions are that (1) only one party learns the intersection (denoted by P_1), and (2) P_1 does not learn any information before receiving the second (and last) message of the protocol from P_2.

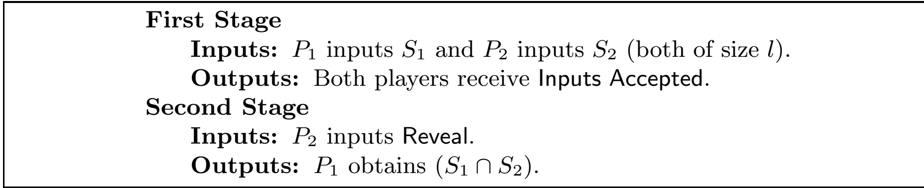


Fig. 2. $\mathcal{F}_{2PSI}$

Given a fully-simulatable PSI that has these two properties, all we need in order to realize the $\mathcal{F}_{2PSI}$ is to execute the protocol until the step in which P_2 is supposed to send his last message. Instead of sending the last message, we modify the protocol so that P_2 only sends a commitment on that message. The commitment in use should be equivocal and extractable. (Such a commitment can be constructed in the random oracle model, using $H(m, r)$, or in the standard model, e.g., [13].) This completes the first stage. When P_2 wants to reveal the intersection (i.e. the second stage), he decommits his last message and P_1 completes the protocol. The intuition is that the simulation for the two-stage PSI is done by calling the original PSI simulators and replacing the last message with a commitment to it. Note that after the commitment to the last message is sent, both players cannot change their inputs (obviously, P_2 cannot decommit to a different message, and P_1 cannot change his inputs since otherwise the original PSI would be insecure). We defer a more formal treatment to the full version of the paper.

Using the above transformation on the protocol of [6], for instance, yields an efficient two-stage PSI that requires only a linear number (in the number of sets) of public key operations by each party.

6.3 The Protocol

A detailed description of the protocol is in Figures 3 and 5. In Appendix D we prove the following theorem:

Theorem 4. *Assume that the committing-OT, the PSI protocol, the commitment, and the garbling scheme are secure. Then, the protocol from Figures 3 and 5 is a 2^{-s+1}-CovIDA secure realization of f.*

Input-consistency check. The consistency of the players' inputs is handled using the technique of [22], where a universal hash function (UH) is evaluated on the input of each player, and the players verify that the outputs of this function are the same in all circuits. A player's input is padded with a short random string r in order to increase its entropy and by that, reduce the amount of information that can be learnt about the input from the output of the UH. Let l be the input length and s be a security parameter. [22] shows a matrix of dimensions $s \times (l + 2s + \log s)$ that can be used as a UH, where the evaluation consists of multiplying this matrix with the input vector (and getting a vector of length s).

In our protocol we require an additional property from this matrix: Given a matrix M and an output vector v, we require that for any input vector v_i, it is easy to find a vector v_r such that $M \times (v_i \| v_r)^T = v^T$. Interestingly, we propose a solution meeting this property that is simpler and more efficient than the solution of [22]. In particular, we generate a random matrix in $\{0,1\}^{s \times l}$ and concatenate it with the identity matrix of size s, resulting in matrix of dimensions $s \times (l + s)$. Evaluation consists of multiplying this matrix with the input vector which is the l bits of real input and s random bits. The construction is a UH for reasons identical to the construction of [22], and knowing the output of UH reveals nothing about the real input since the output vector will be uniformly random (give the s random bits).

Efficiency comparison. We briefly discuss efficiency of our protocol compared to the best alternative using existing techniques. As discussed earlier, the best alternative (referred to as Best Previous in the Figures), is to use the construction of [18] augmented with the technique of [14] in order to reduce the number of garbled circuits. In particular, in this potential solution (not published elsewhere) we run the 2PC once in each direction (with careful incorporation of the input consistency checks), run the cheating-recovery computation at the end of each, and perform a maliciously secure equality-check to compare the two outputs.

We initially focus on the overhead in computation and communication beyond what is required to garble s circuits by each party and the associated input-consistency checks, since those are part of any know solution for ϵ-CovIDA secure 2PC. Ignoring the cost of the equality-check, the overhead here consists of the two cheating-recovery executions. This is included in Figure 6 based on the numbers given in [14] (see Section 3.1).

The overhead of our protocol is simply to run a two-stage PSI with malicious security where both parties learn the output, and where each set is of expected size $s/2$. This requires s commitments and a standard maliciously secure PSI for which we use the concrete numbers given in [6]. As can be seen in the table, the overhead in our construction is significantly smaller, i.e. a factor of 10 or more in communication, and a factor of 200 or more in computation even for input size of 1. This improvement further increases as the input size grows since the overhead in our construction is independent of the input while the cost of cheating recovery linearly grows with it.

While this improvement is only in the "overhead" cost, we stress that the overhead can be a significant portion of the overall cost in small circuits. In Figure 7 we compare the overall costs for the two approaches where to estimate the cost of garbling and the input-consistency checks needed for dual-execution, we use the numbers given in [14] (multiplied by two) both for our solution and the "Best Previous" (with the exception that we assume the use of 2-row reduction techniques when measuring communication for both, but this only effects the bandwidth column). We note that this comparison is on the conservative side, and should be seen as the minimum improvement since more optimized options are available in the RO model (specially for input-consistency checks), and those would highlight our improvements in the overhead even further.

Alice's input: $x_A \in \{0,1\}^\ell$. **Bob's input:** $x_B \in \{0,1\}^\ell$.
Common input: Alice and Bob agree on the description of a circuit C, where $C(x_A, x_B) = f(x_A, x_B)$, and a collision resistance hash function $H : \{0,1\}^* \to \{0,1\}^\ell$. Let $\mathsf{Commit}(\cdot)$ be an extractable and equivocal commitment.
s is a statistical security parameter that represents the bound on the cheating probability. L is a computational security parameter, so, for example, each key label is L-bits long. s' is a statistical security parameter associated with the input-consistency matrix.. Let $\ell' = \ell + m$, and $m' = m + 2s'$.
Output: Both players learn an m-bit string $f(x_A, x_B)$.

Below, we describe the protocol for the case where Alice is the garbler and Bob is the evaluator. But the protocol is symmetric and each step is performed simultaneously in the other direction as well (where Bob is the garbler and Alice is the evaluator) before moving to the next step. In what follows, we slightly abuse the notations we introduced for a garbling scheme. In particular, we feed input/output labels as inputs to the Garble algorithm while in previous section, they were the output of Garble. This is compatible with all existing instantiations.

Garbler's input preparation.

1. Alice chooses s PRF seeds $sd_1^A, \ldots, sd_s^A$, and commits on them using $\mathsf{Commit}(sd_1^A), \ldots, \mathsf{Commit}(sd_s^A)$. All the randomness Alice will use for generating the ith garbled circuit and its input labels will be derived from sd_i^A.
2. Alice chooses $r_A \in_R \{0,1\}^t, r'_A \in_R \{0,1\}^m$ and sets $x'_A = x_A \| r_A \| r'_A$. She will be using x'_A as her input to the circuits instead of x_A. We denote the jth bit of x'_A by $x'_{A,j}$.
3. Alice chooses $\mathsf{in}_b^{A,i,j}, \mathsf{in}_b^{B,i,j} \in_R \{0,1\}^L$ for $b \in \{0,1\}$, $1 \leq i \leq s$ and $1 \leq j \leq \ell'$. $\mathsf{in}_b^{A,i,j}$ would be the b-key for Alice's jth input wire in the ith garbled circuit. ($\mathsf{in}_b^{B,i,j}$ is defined similarly with respect to Bob.) Using the garbling schemes notation we have $e_A^i = (\mathsf{in}_0^{A,i,1}, \mathsf{in}_1^{A,i,1}, \ldots, \mathsf{in}_0^{A,i,\ell'}, \mathsf{in}_1^{A,i,\ell'}, \mathsf{in}_0^{B,i,1}, \mathsf{in}_1^{B,i,1}, \ldots, \mathsf{in}_0^{B,i,\ell'}, \mathsf{in}_1^{B,i,\ell'})$.
4. Alice sends $\mathsf{Commit}(H(\mathsf{in}_{x'_{A,1}}^{A,i,1} \| \cdots \| \mathsf{in}_{x'_{A,l'}}^{A,i,l'}))$ for $1 \leq i \leq s$, i.e. commitments to encoding of her inputs.

Oblivious transfer for evaluator's input.

1. Alice and Bob engage in ℓ' OTs, where in the jth OT, Bob sends $q_j = \mathrm{COT}_1(x'_{B,j})$ and Alice answers with $\mathrm{COT}_2([\mathsf{in}_0^{B,1,j}, \ldots, \mathsf{in}_0^{B,s,j}], [\mathsf{in}_1^{B,1,j}, \ldots, \mathsf{in}_1^{B,s,j}], q_j)$

Fig. 3. 2^{-s}-CovIDA 2PC via PSI

Continued

Circuit Preparation.

1. Alice and Bob jointly choose matrices $M_A, M_B \in_R \{0,1\}^{s' \times \ell'}$, and concatenate each with a $s' \times s'$ identity matrix to obtain M'_A, M'_B respectively. Let $C'(x'_A, x'_B) = (C(x_A, x_B) \oplus r'_A \oplus r'_B, M'_A \cdot x'_A, M'_B \cdot x'_B)$.
2. Alice chooses m' random label pairs and sets $d_A = (\mathsf{out}_0^{A,1}, \mathsf{out}_1^{A,1}) \ldots (\mathsf{out}_0^{A,m'}, \mathsf{out}_1^{A,m'})$.
3. For $1 \leq i \leq s$, Alice computes $\mathsf{GC}_i^A \leftarrow \mathsf{Garble}(C', e_A^i, d_A)$. Note that unlike standard garbling, here the input and output labels are fixed and fed as input to the garbling algorithm, and note that the same output label is used for all s circuits.
4. Alice computes the output decoding table $\mathsf{GDec}^A = \{[i, H(\mathsf{out}_0^{A,i}), H(\mathsf{out}_1^{A,i})]\}_{i=1}^{m'}$. (Note that this is different from d_A and the same table is used for all garbled circuits.)
5. Alice sends garbled circuits $\mathsf{GC}_1^A, \ldots, \mathsf{GC}_s^A$ and the output decoding table GDec^A.

Challenge Generation.

1. Alice chooses $\alpha_1^{(A)}, \alpha_2^{(A)} \in_R \{0,1\}^s$. Similarly, Bob chooses $\alpha_1^{(B)}, \alpha_2^{(B)} \in_R \{0,1\}^s$
2. Alice sends $\mathsf{Commit}(\alpha_1^{(A)})$ and Bob sends $\mathsf{Commit}(\alpha_2^{(B)})$.
3. Alice sends $\alpha_2^{(A)}$ and Bob sends $\alpha_1^{(B)}$.
4. Both players decommit their commitments and set $\alpha_1 = \alpha_1^{(A)} \oplus \alpha_1^{(B)}$ and $\alpha_2 = \alpha_2^{(A)} \oplus \alpha_2^{(B)}$. If one of those values is all zeros, or all one, they go back to step 1.
5. We define the evaluation set $\mathbb{E}_A$ such that $i \in \mathbb{E}_A$ if and only if ith bit of α_1 is one (Similarly, $\mathbb{E}_B$ would be generated using α_2).

Garbled Circuit Evaluation.

1. Alice sends $\mathsf{in}_{x'_{A,j}}^{A,i,j}$ for $i \in \mathbb{E}_A$ and $1 \leq j \leq \ell'$, and decommits $\mathsf{Commit}(H(\mathsf{in}_{x'_{A,1}}^{A,i,1} \| \cdots \| \mathsf{in}_{x'_{A,l'}}^{A,i,l'}))$.
2. For $i \in \mathbb{E}_A$, Bob evaluates GC_i^A and gets the garbled output Z_i^A. Let z_i^A be the actual output resulted from decoding Z_i^A using GDec^A. If any of the decoded bits is $\perp$, Bob sets z_i^A to $\perp^{m'}$.

Committing to PSI input sets.

1. For all $i \in \mathbb{E}_A$, if $z_i^A \neq \perp^{m'}$, Bob parses $z_i^A = z_{i,1}^A \cdots z_{i,m'}^A$. He then computes $q_i = \big((\mathsf{out}_{z_{i,1}^A}^{A,1} \oplus \mathsf{out}_{z_{i,1}^A}^{B,1}) \oplus \cdots \oplus (\mathsf{out}_{z_{i,m'}^A}^{A,m'} \oplus \mathsf{out}_{z_{i,m'}^A}^{B,m'})\big)$. If $z_i^A = \perp^{m'}$, on the other hand, Bob sets q_i to be a random $(L+m')$-bit value.
   If $q_i \in T_B$, he modifies q_i to be a random $(L+m')$-bit value. He adds q_i to T_B .
2. Alice and Bob call the first stage of $\mathcal{F}_{2PSI}$ with their inputs T_A and T_B.
3. For all $i \in \mathbb{E}_A$, Alice commits to q_i using $\mathsf{Commit}(\cdot)$ and sends these commitments in a random order. (Bob does not need to follow this step.)

Fig. 4. 2^{-s}-CovIDA 2PC via PSI

Continued

Opening.

1. Alice decommits sd_i^A for all $i \notin \mathbb{E}_A$. She also reveals her OT inputs corresponding to the opened circuits (the OTs for Bob to learn his input labels).
2. Bob aborts if any of the following occurs:
   - $\exists i \in \mathbb{E}_A$ such that Alice's garbled inputs are invalid.
   - $\exists i \notin \mathbb{E}_A$ in which $\mathsf{GC}_i^A \neq \mathsf{Garble}(C_A, sd_i^A, d_A)$, or, the OTs are not consistent with GC_i^A. (Note that once some sd_i^A is revealed, Bob can compute d_A by himself.)
   - Some of the output labels $(\mathsf{out}_{A,0}^1, \mathsf{out}_{A,1}^1) \ldots (\mathsf{out}_{A,0}^{m'}, \mathsf{out}_{A,1}^{m'})$ are not properly constructed or not consistent with GDec^A.

Output generation.

1. Alice and Bob perform the second stage of $\mathcal{F}_{2PSI}$ in order for Alice to learn $I = T_A \cap T_B$.
2. Alice aborts if $I = \emptyset$. Else, she decommits $\mathsf{Commit}(q_i)$ corresponding to the single element in I (note that the intersection is guaranteed to be of size at most one). Bob aborts if the decommitment is invalid or if the decommitted value is not in set T_B.
3. Recall that the computation output is masked by r'_A and r'_B. Alice sends r'_A and the labels that correspond to r'_A in GC_i^B for $i = \min(\mathbb{E}_A)$. (These labels are used for authenticating r'_A.) If the labels are invalid, Bob aborts.
4. Both players unmask the output from q_i and output the result.

Fig. 5. 2^{-s}-CovIDA 2PC via PSI

Finally, for concreteness, in Figure 8 we look at the overall cost of the two protocols for the AES circuit with 6800 non-XOR gates, input size $\ell = 128$, the symmetric ciphertext size $n = 128$ and group element size 220 bits. We have combined the communication cost into one column named bandwidth which is in bits. We express all costs in terms of parameter s. For instance, our new protocol reduces overall bandwidth by 35% and the number of exponentiations by more than 50%.

Different deterrence value for each player. Besides its better efficiency, an additional advantage of our new protocol is that each party has a separate challenge generation and uses a different challenge set $\mathbb{E}_i$, to determine which circuits to check and which ones to evaluate. This allows us to use a different number of garbled circuits for each party and as a result achieve different deterrence factors for each. This variant can be quite useful in practice where different participants in a protocol may have different reputations (to protect) or different levels of tolerance for risk. One can take these real-world factors into account when adjusting the deterrence factor for each party.

Construction	Fixed-based Exponent.	Regular Exponent.	Symmetric Encryption	Group elements sent	Symmetric communication
Best Previous	$18s\ell + 1080s$	$960s$	$78s\ell$	$42s\ell$	$24sn\ell$
Ours	$4.5s$	s	$2s$	$4s$	sn

Fig. 6. Comparison of overhead cost of our CovIDA 2PC protocol with the best alternative using state of the art techniques. s is the number of circuits, n is length of symmetric-key ciphertext, and ℓ is the input sizes.

Construction	Fixed-based Exponent.	Regular Exponent.	Symmetric Encryption	Group elements sent	Symmetric communication
Best Previous	$42s\ell + 1080s$	$7s\ell + 36\ell + 960s$	$26s\|C\| + 78s\ell$	$52s\ell$	$4ns\|C\| + 28sn\ell$
Ours	$24s\ell + 4.5s$	$7s\ell + 36\ell + s$	$26s\|C\| + 2s$	$10s\ell + 4s$	$4ns\|C\| + sn$

Fig. 7. Comparison of overall cost of our CovIDA 2PC protocol with the best alternative using state of the art techniques. s is the number of circuits, n is length of symmetric-key ciphertext, ℓ is the input sizes, and $|C|$ is the circuit size.

Construction	Fixed-based	Regular	Symmetric	Bandwidth (bits)
Best Previous	$6456s$	$1856s + 4608$	$186784s$	$5404672s$
Ours	$3076s$	897 s + 4608	176802 s	$3483012s$

Fig. 8. Concrete comparison of overall costs for the AES. s is the number of circuits, $n = 128$ is length of symmetric-key ciphertext, $\ell = 128$ is the input sizes, and $|C| = 6800$ is the number of non-XOR gates in the AES circuit.

References

1. Aiello, W., Ishai, Y., Reingold, O.: Priced oblivious transfer: How to sell digital goods. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 119–135. Springer, Heidelberg (2001)
2. Aumann, Y., Lindell, Y.: Security against covert adversaries: Efficient protocols for realistic adversaries. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 137–156. Springer, Heidelberg (2007)
3. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 2012, pp. 784–796. ACM Press (October 2012)
4. Canetti, R., Lin, H., Pass, R.: Adaptive hardness and composable security in the plain model from standard assumptions. In: 51st FOCS, pp. 541–550. IEEE Computer Society Press (October 2010)
5. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. In: Juels, A., Wright, R.N., Vimercati, S. (eds.) ACM CCS 2006, pp. 79–88. ACM Press (October/November 2006)
6. De Cristofaro, E., Kim, J., Tsudik, G.: Linear-complexity private set intersection protocols secure in malicious model. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 213–231. Springer, Heidelberg (2010)
7. Huang, Y., Katz, J., Evans, D.: Quid-Pro-Quo-tocols: Strengthening semi-honest protocols with dual execution. In: 2012 IEEE Symposium on Security and Privacy, pp. 272–284. IEEE Computer Society Press (May 2012)

8. Huang, Y., Katz, J., Evans, D.: Efficient secure two-party computation using symmetric cut-and-choose. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 18–35. Springer, Heidelberg (2013)
9. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Prabhakaran, M., Sahai, A.: Efficient non-interactive secure computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 406–425. Springer, Heidelberg (2011)
10. Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.-C., Steiner, M.: Outsourced symmetric private information retrieval. In: Sadeghi, A.-R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013, pp. 875–888. ACM Press (November 2013)
11. Jarecki, S., Liu, X.: Fast secure computation of set intersection. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 418–435. Springer, Heidelberg (2010)
12. Lin, H., Pass, R., Venkitasubramaniam, M.: A unified framework for concurrent security: Universal composability from stand-alone non-malleability. In: Mitzenmacher, M. (ed.) 41st ACM STOC 2009, pp. 179–188. ACM Press (May/June 2009)
13. Lindell, Y.: Highly-efficient universally-composable commitments based on the DDH assumption. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 446–466. Springer, Heidelberg (2011)
14. Lindell, Y.: Fast cut-and-choose based protocols for malicious and covert adversaries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 1–17. Springer, Heidelberg (2013)
15. Micali, S., Pass, R., Rosen, A.: Input-indistinguishable computation. In: 47th FOCS, pp. 367–378. IEEE Computer Society Press (October 2006)
16. Micali, S., Reyzin, L.: Physically observable cryptography (extended abstract). In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 278–296. Springer, Heidelberg (2004)
17. Mohassel, P., Franklin, M.K.: Efficiency tradeoffs for malicious two-party computation. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 458–473. Springer, Heidelberg (2006)
18. Mohassel, P., Riva, B.: Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 36–53. Springer, Heidelberg (2013)
19. Pappas, V., Krell, F., Vo, B., Kolesnikov, V., Malkin, T., Choi, S.G., George, W., Keromytis, A., Bellovin, S.: Blind seer: A scalable private DBMS. In: Security and Privacy, Oakland (2014)
20. Pass, R.: Simulation in quasi-polynomial time, and its application to protocol composition. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 160–176. Springer, Heidelberg (2003)
21. Prabhakaran, M., Sahai, A.: New notions of security: Achieving universal composability without trusted setup. In: Babai, L. (ed.) 36th ACM STOC, pp. 242–251. ACM Press (June 2004)
22. Shelat, A., Shen, C.-H.: Fast two-party secure computation with minimal assumptions. In: Sadeghi, A.-R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013, pp. 523–534. ACM Press (November 2013)
23. Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS, pp. 162–167. IEEE Computer Society Press (October 1986)

A Proof of Theorem 2

Proof. We sketch only the proof for corrupt Alice: the other case is symmetric.

In the real execution, Alice provides F_A as input to the commitment scheme, e_A as input to the OTs (as sender), and x_A as input to the OTs (as receiver). Finally, Alice sends a string to the equality test functionality. An honest Bob provides honest F_B and causes Alice to receive honest garbled input X_B. An honest Bob further calculates his input to the equality test as a direct result of the garbled output $\mathsf{Eval}(F_B, \mathsf{Encode}(e_A, x_B))$.

We consider a sequence of hybrid interactions, taking hybrid H0 to be the real execution of the protocol:

H1: The simulator extracts $(f', d') \leftarrow \mathsf{Extract}(F_A, e_A)$, and Bob's effective garbled output is instead computed as $\mathsf{Encode}(d', f'(x_B))$.[4] This hybrid is indistinguishable by the prop-enforcing guarantee of $\mathcal{G}$. Provided that Bob does not abort in step 4, we also have $\mathsf{prop}(f') = \mathsf{prop}(f(0^n, \cdot))$.

H2: We focus on the values Y_B and Y_{AB} that Alice provides to the equality test. These values are compared to values that Bob computes by encoding $\tilde{y}_B$ and $\tilde{y}_{AB}$ under encoding d_B, where $(\tilde{y}_A, \tilde{y}_B, \tilde{y}_{AB}) = f'(x_B)$. Importantly, Bob provides *valid* garbled outputs (under encoding d_B). By the authenticity guarantee of $\mathcal{G}$, Alice cannot guess any valid garbled outputs besides the ones she is prescribed via $\mathsf{Eval}(F_B, X_A)$.[5] By the correctness of the garbling scheme, these prescribed garbled outputs encode the "correct" values y_B and y_{AB} computed from $f(x_A, x_B)$. Thus the simulator in this hybrid returns false for the equality test if $f'(x_B)$ and $f(x_A, x_B)$ disagree in their y_B or y_{AB} components, or if Alice provides Y_B or Y_{AB} different from those prescribed via $\mathsf{Eval}(F_B, X_A)$. This change is indistinguishable by the authenticity property of $\mathcal{G}$.

H3: The simulator uses Alice's prescribed output (y_A, y_{AB}) to generate a simulated garbled circuit F_A and garbled input X_A. This hybrid is indistinguishable by the privacy/obliviousness guarantee of $\mathcal{G}$.[6]

H4: We focus on the other values $\tilde{Y}_A$ and $\tilde{Y}_{AB}$ that Alice provides to the equality test. These are compared to Bob's value that is determined from $\mathsf{Encode}(d', f'(x_B))$. The simulator can easily check whether $\tilde{Y}_A, \tilde{Y}_{AB}$ are valid encodings under d' (i.e., possible outputs of $\mathsf{Encode}(d', \cdot)$). If not, then the equality test will always return false and the simulator can also do so. Otherwise, the simulator can easily determine $\tilde{y}_A$ and $\tilde{y}_{AB}$ such that

[4] For simplicity, we are glossing over the case where f' outputs $\perp$ (corresponding to the event that Bob's execution of Eval results in $\perp$). In this event, Bob will run the equality test on random inputs, and the equality test will result in false. Looking ahead, this matches the semantics of $L_{f,f',\tilde{y}_A}$ that will be chosen as the leakage function in the ideal world. For the rest of the proof we can therefore condition on f' not outputing $\perp$.

[5] Since F_B and X_A are designated by Bob, this execution of Eval will not abort.

[6] We use obliviousness since the simulated garbled circuit contains no information about y_B.

$\tilde{Y}_A \| \cdots \| \tilde{Y}_{AB} = \mathsf{Encode}(d', \tilde{y}_A \| \cdot \| \tilde{y}_{AB})$. Then the equality check involving these $\tilde{Y}_A, \tilde{Y}_{AB}$ values is logically equivalent to $(\tilde{y}_A, \cdot, \tilde{y}_{AB}) \stackrel{?}{=} f'(x_B)$.

We see that in the hybrid labeled H4, the equality test outcome is determined by the following logic:

$$f'(x_B) = (\tilde{y}_A, \tilde{y}_B, \tilde{y}_{AB}) \text{ and } f(x_A, x_B) = (y_A, y_B, y_{AB}) \text{ and } \tilde{y}_B = y_B \text{ and } \tilde{y}_{AB} = y_{AB}$$

Indeed, the outcome of the equality test is precisely $L_{f,f',\tilde{y}_A}(x_A, x_B)$ where $\tilde{y}_A$ is the value that the simulator extracts as described above. Overall, we have described a simulator that is indistinguishable from the real execution; it needs to know only Alice's prescribed output (y_A, y_{AB}), and the answer to a $\mathcal{L}^f_{\mathsf{prop}}$-leakage query described above.

B Proof of Theorem 3

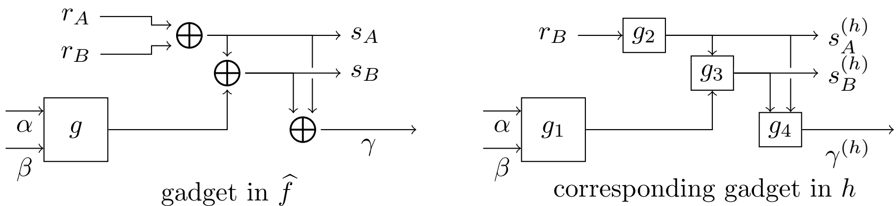
Proof. For simplicity, suppose f gives all of its output to both parties (there is no output given to just one of the parties). Then $\widehat{f}$ has syntax $\widehat{f}((x_a, \{r_{g,A}\}_g), (x_B, \{r_{g,B}\}_g)) = (\{s_{g,A}\}_g, \{s_{g,B}\}_g, y = f(x_A, x_B))$.

First, consider the case where a corrupt Alice attacks the Π protocol. She provides input $(x_A, \{r_{g,A}\}_g)$ to $\widehat{f}$, then receives output $(\{s_{g,A}\}_g, y)$ from $\widehat{f}$. She then chooses a legal leakage function $L^A_{\widehat{f},h,\tilde{y}} \in \mathcal{L}^{\widehat{f}}_{\mathsf{prop}}$ and learns the result. Bob aborts if the leakage function evaluates to zero.

In the simulation, the simulator picks outputs $\{s_{g,A}\}_g$ uniformly at random. It remains to show how the simulator simulates the outcome of the leakage function given only $\mathcal{L}^f_{\mathsf{ocl}}$ leakage to an ideal f.

Alice chooses a leakage function $L^A_{\widehat{f},h,\tilde{y}}$ where $\mathsf{prop}(h) = \mathsf{prop}(\widehat{f}(0^n, \cdot))$. Since prop includes the circuit topology, we can naturally talk about a correspondence between the topology of h and $\widehat{f}$.

The analysis proceeds one gate a time, in a topological order. Suppose we are considering some gate g in f, which corresponds to the larger gadget in $\widehat{f}$, described above. Then h has a similar gadget, in which some of the gate logic may be changed:



By our inductive hypothesis, we'll assume that the values of α and β agree with the corresponding values in $\widehat{f}$ (and hence in f). By construction, the same r_B value is used as input for both circuits. Hence we use the same variable names for these values in the above diagram.

The leakage function $L^A_{\widehat{f},h,\tilde{y}}$ simply performs a string equality related to the outputs of $\widehat{f}$ and h. It is helpful to think of this string comparison as a conjunction of single-bit comparisons (taken in the same order as our gate-by-gate analysis), which will "short circuit" to return 0 as soon as a mismatch is encountered.

The current gadget in $\widehat{f}$ and h includes outputs which the leakage function checks in the following way. The string $\tilde{y}$ (a parameter of the leakage function, chosen by Alice) includes a single position whose value we call $\tilde{s}$. The leakage function checks the two bit-comparisons $s_A^{(h)} = \tilde{s}$ and $s_B^{(h)} = s_B$.

We rewrite these two conditions as follows:

$$\begin{aligned} & (s_A^{(h)} = \tilde{s}) \wedge (s_B^{(h)} = s_B) \\ \iff & (s_A^{(h)} = \tilde{s}) \wedge (g_3(s_A^{(h)}, g_1(\alpha, \beta)) = s_B) \\ \iff & (s_A^{(h)} = \tilde{s}) \wedge (g_3(\tilde{s}, g_1(\alpha, \beta)) = s_B) \\ \iff & (s_A^{(h)} = \tilde{s}) \wedge (g_3(\tilde{s}, g_1(\alpha, \beta)) = g(\alpha, \beta) \oplus s_A) \\ \iff & (g_2(r_B) = \tilde{s}) \wedge (g_3(\tilde{s}, g_1(\alpha, \beta)) = g(\alpha, \beta) \oplus s_A) \\ \iff & (g_2(r_A \oplus s_A) = \tilde{s}) \wedge (g_3(\tilde{s}, g_1(\alpha, \beta)) = g(\alpha, \beta) \oplus s_A) \end{aligned}$$

Observe that the gates $g, g_1, \ldots, g_3$ and values $s_A, r_A, \tilde{s}$ are known to the simulator. Hence, this condition is a function of α, β alone — it is a *gate-local* constraint in f! A simulator only needs to know the result of this gate-local function to simulate the corresponding bit-comparisons in the leakage function $L^A_{\widehat{f},h,\tilde{y}}$.

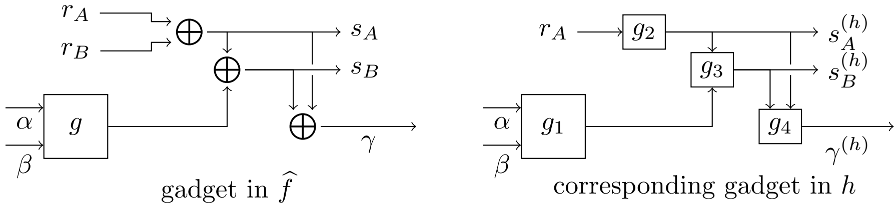
Conditioned on the constraint being true, we examine the output $\gamma^{(h)}$ of the gadget in h. We have:

$$\gamma^{(h)} = g_4(s_A^{(h)}, s_B^{(h)}) = g_4(\tilde{s}, s_A \oplus g(\alpha, \beta)) = g_4(\tilde{s}, s_A \oplus \gamma)$$

Let $\pi(\cdot) = g_4(\tilde{s}, s_A \oplus \cdot)$. Clearly the simulator can extract the unary function π. All gates downstream of $\gamma^{(h)}$ will receive the value of $\pi(\gamma)$, where γ is the "correct" value that leaves this gadget in $\widehat{f}$. But this is equivalent to sending γ along this wire and modifying a downstream gate g' to have logic $g'(\pi(\cdot), \cdot)$ instead. This modified circuit has the same topology as h, so our analysis is not affected. Furthemore, this transformation preserves the invariant that the inputs to all gadgets in h match their counterparts in $\widehat{f}$ (conditioned on the event that the leakage function has not yet short-circuited).

Overall, the simulator will only need to know the conjunction of many gate-local constraints, one for each gate g in f. Hence, the simulator can succeed by asking only a $\mathcal{L}^f_{\mathsf{ocl}}$ query.

The case where Bob is corrupt is similar, but slightly different due to the asymmetry between Alice & Bob in the gadgets. In this case, the gadget is only slightly different (now it is Alice's input r_A which goes to gate g_2):



Also, the leakage function now checks the complementary condition $(s_A^{(h)} = s_A) \wedge (s_B^{(h)} = \tilde{s})$. The key to manipulating this expression is the observation that $s_A = g(\alpha, \beta) \oplus s_B$.

We obtain:

$$\begin{aligned}
&(s_A^{(h)} = s_A) \wedge (s_B^{(h)} = \tilde{s}) \\
\iff &(s_A^{(h)} = s_A) \wedge (g_3(s_A^{(h)}, g_1(\alpha, \beta)) = \tilde{s}) \\
\iff &(s_A^{(h)} = s_A) \wedge (g_3(s_A, g_1(\alpha, \beta)) = \tilde{s}) \\
\iff &(s_A^{(h)} = s_A) \wedge (g_3(g(\alpha, \beta) \oplus s_B, g_1(\alpha, \beta)) = \tilde{s}) \\
\iff &(s_A^{(h)} = g(\alpha, \beta) \oplus s_B) \wedge (g_3(g(\alpha, \beta) \oplus s_B, g_1(\alpha, \beta)) = \tilde{s}) \\
\iff &(g_2(r_A) = g(\alpha, \beta) \oplus s_B) \wedge (g_3(g(\alpha, \beta) \oplus s_B, g_1(\alpha, \beta)) = \tilde{s}) \\
\iff &(g_2(g(\alpha, \beta) \oplus s_B \oplus r_B) = g(\alpha, \beta) \oplus s_B) \wedge (g_3(g(\alpha, \beta) \oplus s_B, g_1(\alpha, \beta)) = \tilde{s})
\end{aligned}$$

As before, all gates $g, g_1, \ldots, g_3$ and values $s_B, r_B, \tilde{s}$ are known to the simulator, making this expression a gate-local function of α, β alone.

Conditioned on the above expression being true, we also have:

$$\gamma^{(h)} = g_4(s_A^{(h)}, s_B^{(h)}) = g_4(s_A, \tilde{s}) = g_4(\gamma \oplus s_B, \tilde{s})$$

As before, $\gamma^{(h)}$ is a fixed unary function of the "correct" value γ, and the rest of the argument goes through analogously.

C CovIDA Security Definition

The following definitions for ϵ-CovIDA security are taken from [18].

Real-model execution. The real-model execution of protocol Π takes place between players (P_1, P_2), at most one of whom is corrupted by a non-uniform probabilistic polynomial-time machine adversary $\mathcal{A}$. At the beginning of the execution, each party P_i receives its input x_i. The adversary $\mathcal{A}$ receives an auxiliary information *aux* and an index that indicates which party it corrupts. For that party, $\mathcal{A}$ receives its input and sends messages on its behalf. Honest parties follow the protocol.

Let $\text{REAL}_{\Pi,\mathcal{A}(aux)}(x_1, x_2)$ be the output vector of the honest party and the adversary $\mathcal{A}$ from the real execution of Π, where *aux* is an auxiliary information and x_i is player P_i's input.

Ideal-model execution. Let $f : (\{0,1\}^*)^2 \to \{0,1\}^*$ be a two-party functionality. In the ideal-model execution, all the parties interact with a trusted party that evaluates f. As in the real-model execution, the ideal execution begins with each party P_i receiving its input x_i, and $\mathcal{A}$ receives the auxiliary information aux. The ideal execution proceeds as follows:

Send inputs to trusted party: Each party P_1, P_2 sends x'_i to the trusted party, where $x'_i = x_i$ if P_i is honest and x'_i is an arbitrary value if P_i is controlled by $\mathcal{A}$.

Abort option: If any $x'_i = \mathsf{abort}$, then the trusted party returns abort to all parties and halts.

Attempted cheat option: If P_i sends $\mathsf{cheat}_i(\epsilon')$, then:
- If $\epsilon' > \epsilon$, the trusted party sends $\mathsf{corrupted}_i$ to all parties and the adversary $\mathcal{A}$, and halts.
- Else, with probability $1 - \epsilon'$ the trusted party sends $\mathsf{corrupted}_i$ to all parties and the adversary $\mathcal{A}$ and halts.
- With probability ϵ',
  - The trusted party sends $\mathsf{undetected}$ and $f(x'_1, x'_2)$ to the adversary $\mathcal{A}$.
  - $\mathcal{A}$ responds with an arbitrary boolean (polynomial) function g.
  - The trusted party computes $g(x'_1, x'_2)$. If the result is 0 then the trusted party sends abort to all parties and the adversary $\mathcal{A}$ and halts. (i.e. $\mathcal{A}$ can learn $g(x'_1, x'_2)$ by observing whether the trusted party aborts or not.)

Otherwise, the trusted party sends $f(x'_1, x'_2)$ to the adversary.

Second abort option: The adversary sends either abort or $\mathsf{continue}$. In the first case, the trusted party sends abort to all parties. Else, it sends $f(x'_1, x'_2)$.

Outputs: The honest parties output whatever they are sent by the trusted party. $\mathcal{A}$ outputs an arbitrary function of its view.

Let $\text{IDEAL}^{\epsilon}_{f,\mathcal{A}(aux)}(x_1, x_2)$ be the output vector of the honest party and the adversary $\mathcal{A}$ from the execution in the ideal model.

Definition 2. *A two-party protocol Π is* secure with input-dependent abort in the presence of covert adversaries with ϵ-deterrent (ϵ-CovIDA) *if for any non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real model, there exists a non-uniform probabilistic polynomial time adversary $\mathcal{S}$ in the ideal model such that*

$$\left\{\text{REAL}_{\Pi,\mathcal{A}(aux)}(x_1, x_2)\right\}_{x_1,x_2,aux\in\{0,1\}^*} \overset{c}{\approx} \left\{\text{IDEAL}^{\epsilon}_{f,\mathcal{S}(aux)}(x_1, x_2)\right\}_{x_1,x_2,aux\in\{0,1\}^*}$$

for all $|x_1| = |x_2|$ and aux.

D Proof Sketch of Theorem 4

We only present the proof for the case when Alice is corrupted. The case of corrupted Bob is symmetric. The probability that at least one of the evaluated

garbled gates that were generated by Alice was constructed properly (and consistently with the COT) is $1 - 2^{-s}$ (because of the cut-and-choose and the fact that Bob checks both the garbled circuits and their corresponding COT inputs). On the other hand, Alice is forced (because of the OTs) to use only one input for the garbled circuits generated by Bob. If that input is different than the one she has used for the garbled circuits that Bob evaluates, then with good probability the outputs of the input-consistency check will be different, causing the PSI to return an empty set (since in order to know one element in Bob's set, Alice has to guess output wire labels which were not revealed to her in her evaluations). Therefore, if the PSI returns at least one element, that element is indeed the result of correct evaluations of valid garbled circuits done by both players, and since Bob is honest, this is the right output. Furthermore, since Bob is honest, Alice will get only a single output from all her evaluations, and will have to use random elements for the rest of her PSI inputs. Since Bob's inputs to the PSI include information that must be learned from the output wire labels chosen by Bob, the only element in the intersection could be the right output.

More formally, let $\mathcal{A}$ be an adversary controlling Alice in the execution of the protocol. We describe a simulator $\mathcal{S}$ that runs $\mathcal{A}$ internally and interacts with the trusted third party (TTP) that computes f. $\mathcal{S}$ does the following:

1. Emulates a honest Bob with input 0 until the end of "Circuit Preparation" Stage.
2. During the emulation, $\mathcal{S}$ extracts the seeds Alice committed on, and learns all her inputs to the OTs (including her input x^A in the OTs for her to learn her input labels for Bob's circuits). We say that a seed, a garbled circuit and its COT inputs constitute a good set if they are consistent and properly generated, and a bad set otherwise. For each of Alice's circuits, $\mathcal{S}$ determines if it is a good or a bad set. Note that unless all Alice's circuits are bad, at this stage $\mathcal{S}$ can compute the output labels of Alice's circuits.
   Let r_B, r'_B be the values chosen by the emulated Bob and let r_A, r'_A be the values chosen by Alice in the OTs. Alice receives only the output $f(x_A, 0) \oplus r'_A \oplus r'_B$ from her evaluated circuits and can compute only a single valid value q_i. (She can also guess other valid values with a negligible in L probability.) $\mathcal{S}$ can also compute this value at this stage. Denote this valid q_i by Q.
   If all sets are good, $\mathcal{S}$ sends x_A to the TTP, receives the output z, and continues the emulation of Bob. If Alice enters Q to the PSI, she receives the same value as the intersection, but otherwise she receives an empty set. Next, if Alice decommits to a value different than Q in the "Output generation" stage, $\mathcal{S}$ aborts. Else, it sends $R'_B = r'_B \oplus z \oplus f(x_A, 0)$ to Alice and outputs whatever she does. (This causes the output of Alice to be z since she received $f(x_A, 0) \oplus r'_A \oplus r'_B$ from the evaluations.) Note that $\mathcal{S}$ knows the corresponding valid labels for R'_B from the OTs.
   If some of the sets are bad, let $c \in \{0,1\}^s$ be a bitstring such that $c_i = 1$ if set i is good, and $c_i = 0$ otherwise. Also, let e_b be the number of bad sets.
   If $e_b = s$, $\mathcal{S}$ simulates Bob aborting and outputs whatever Alice does. This is due to the fact that we generate the challenge set such that at least one circuit is always checked.

If $0 < e_b < s$, then

- Let $p = \frac{2^{s-e_b}-1}{2^s-2} - \frac{1}{2^s-2}$. With probability p, $\mathcal{S}$ extracts $\alpha_1^{(A)}$ and chooses a random $\alpha_1^{(B)}$ such that all bad sets are in $\mathbb{E}_A$, but also at least one good set is also in it. It calls the TTP with x_A and receives the output z and proceeds with the emulation until the output unmasking. As before, if Alice does not send Q to the PSI, it receives an empty set, and if she does not decommit Q afterwards, $\mathcal{S}$ aborts. Last, $\mathcal{S}$ sends r'_A so that the unmasked output would be z, and outputs whatever Alice outputs.
- With probability $1-p$, $\mathcal{S}$ calls the TTP with x_A and then sends the message $\mathsf{cheat}(1/(2^s-2)/(1-p))$. (Note that $(1/2^s-2)/(1-p) \leq 2^{-s+1}$ when $e_b \geq 1$.)

  If the TTP returns a $\mathsf{corrupted}$ message, $\mathcal{S}$ chooses a random $\alpha_1^{(B)}$ such that some bad circuits will be checked, and continues the emulation until Bob aborts once the bad circuits are checked. Otherwise, i.e., in case the TTP returns $\mathsf{undetected}$ and the output z, $\mathcal{S}$ causes $\mathbb{E}_A$ to be the set of all bad garbled circuits, and sends to the TTP the function that has hardcoded the bad garbled circuits, their COT inputs, all output labels (chosen by both players), the values r_B, r'_B and $R'_B = r'_B \oplus z \oplus f(x_A, 0)$, and the value Q. (This part is similar to [7,18].) The function takes Bob's real input x_B, finds R_B such that $M_B \cdot (0^l \| R_B \| R'_B) = M_B \cdot (0^l \| r_B \| r'_B)$, emulates a honest Bob with inputs x_B, R_B, R'_B that receives its input labels from the OTs, evaluates the garbled circuits, and checks if any of the outputs would give $q_i = Q$. In high level, the function emulates what a honest Bob would get from the evaluation using Bob's real and output mask R'_B that makes sure both players receive the same output. If the TTP responds with abort, $\mathcal{S}$ emulates Bob aborting after Alice decommits the output of the intersection (or before in case the emulated Bob aborts). If the TTP does not respond with abort, as before, if Alice inputs Q to the PSI she receives it back, or empty set otherwise, and if she decommits to a different value than Q, then $\mathcal{S}$ emulates Bob Aborting. In case no abort happens until the end of the protocol, $\mathcal{S}$ sends R'_B as Bob's mask. (Note that the COT for the bits of R'_B must be fine since otherwise the function that emulated Bob by the TTP would have failed producing the value Q.)

We now analyse the probabilities of the different cases: (1) If all sets are good, then $\mathcal{S}$ simply retrieves the output and unmask the output accordingly. The simulation looks the same as the real execution except for Bob's inputs which are hidden because of the security of the COT and the garbling scheme; (2) if all sets are bad, then $\mathcal{S}$ will emulate Bob aborting and outputs what Alice does. This is identical to the real world since at least once circuit is always checked and hence Alice caught. (3) If some sets (but not all) are bad then there are three possibilities:

- Alice is caught cheating - Happens with probability $(1-p) \times (1 - \frac{1/(2^s-2}{1-p}) = 1 - p - 1/(2^s-2) = \frac{2^{s-e_b}-1}{2^s-2}$.
- The protocol ends without accusing Alice of cheating - Happens with probability $p = \frac{2^{s-e_b}-1}{2^s-2} - \frac{1}{2^s-2}$.
- Alice successfully cheats - Happens with probability $(1-p) \times \frac{1/(2^s-2)}{1-p} = 1/(2^s-2)$.

Note that the soundness of the protocol is 2^{-s+1} since we call the TTP with the message $\mathsf{cheat}_{\frac{1}{2^s-1}}/(1-p)) \leq 2^{-s+1}$ for the p we have. We stress that the actual cheating probability is only $1/(2^s-1)$. (This "gap" is a because the adversary is not always accused of cheating, even if it gets caught.)

We remark that the adversary can guess one of Bob's output labels with a negligible in L probability. Since it can guess several values and enter them as inputs to the PSI, the probability that at least one of them would be valid is $|\mathbb{E}_A| \cdot neg(L)$. (This affects only the parameter L, and not s.)