Secure Computation From Millionaire

abhi shelat* and Muthuramakrishnan Venkitasubramaniam**

Abstract. The standard method for designing a secure computation protocol for function f first transforms f into either a circuit or a RAM program and then applies a generic secure computation protocol that either handles boolean gates or translates the RAM program into oblivious RAM instructions.

In this paper, we show a large class of functions for which a different *iterative* approach to secure computation results in more efficient protocols. The first such examples of this technique was presented by Aggarwal, Mishra, and Pinkas (J. of Cryptology, 2010) for computing the median; later, Brickell and Shmatikov (Asiacrypt 2005) showed a similar technique for shortest path problems.

We generalize the technique in both of those works and show that it applies to a large class of problems including certain matroid optimizations, sub-modular optimization, convex hulls, and other scheduling problems. The crux of our technique is to *securely reduce* these problems to secure comparison operations and to employ the idea of *gradually releasing* part of the output. We then identify conditions under which both of these techniques for protocol design are compatible with achieving simulation-based security in the honest-but-curious and covert adversary models. In special cases such as median, we also show how to achieve malicious security.

Keywords: Secure Computation, Semi-honest, Covert Security, Greedy Algorithms

1 Introduction

Secure two-party computation allows Alice with private input x and Bob, with input y, to jointly compute $f(x, y)$ without revealing any information other than the output $f(x, y)$.

Building on Yao's celebrated garbled circuits construction [25], many recent works [17, 18, 4, 11, 15, 14, 10, 16, 21] construct such protocols by first translating f into a boolean circuit and then executing a protocol to securely evaluate *each gate* of that circuit. Alternatively, Ostrovsky and

* University of Virginia, Charlottesville, Virginia, abhi@virginia.edu. Research supported by Google Faculty Research Grant, Microsoft Faculty Fellowship, SAIC Scholars Research Award, and NSF Awards TC-1111781, 0939718, 0845811.

** University of Rochester, Rochester, New York, muthuv@cs.rochester.edu. Research supported by Google Faculty Research Grant and NSF Award CNS-1526377

Shoup [20] demonstrated a way to construct two-party secure computation protocols for RAM programs by first translating the RAM program into a sequence of oblivious RAM (ORAM) instructions and then applying a secure computation protocol to implement each ORAM operation. Further refinements of this idea and state of the art approaches to ORAM design [7, 23, 22] limit the overhead in terms of bandwidth, client storage and total storage to roughly $\tilde{O}(\log^3(n))$ for each operation on a memory of size n resulting in protocols [9, 16, 12, 24] that are efficient enough for some problems in practice.

Reduction-based techniques In both of the above approaches, the secure evaluation of f is reduced to the secure evaluation of either a boolean gate or an ORAM instruction.

Instead of reducing function f into such low-level primitives and securely evaluating *each* primitives, one can also consider reducing f into a program that only makes secure evaluations of a higher-level primitive. A natural candidate for this secure primitive is the comparison function, or the *millionaires* problem.

Aggarwal, Mishra, and Pinkas [1] begin to investigate this approach by studying the problem of securely computing the k^{th}-ranked element of dataset $D_A \cup D_B$ where Alice privately holds dataset $D_A \subset F$ and Bob privately holds dataset $D_B \subset F$. They reduce the computation of the k^{th}-ranked element element to $O(\log k)$ secure comparisons of $(\log M)$-bit inputs where $\log M$ is the number of bits needed to describe the elements in F; this protocol outperforms the naive method for the same problem since a circuit for computing ranked elements has size at least $|D_A \cup D_B|$.

Their algorithm follows the classic communication-optimal protocol for this problem: each party computes the median of its own dataset, the parties then jointly compare their medians, and depending on whose median is larger, each party eliminates half of its input values and then recurses on the smaller datasets. Aggarwal, Mishra and Pinkas observe that by replacing each comparison between the parties' medians with a secure protocol for comparing two elements, they can argue that the overall protocol is secure in the honest-but-curious setting. In particular, for the case of median, they observe that the sequence of answers from each secure comparison operation can be simulated using only the output k^{th}-ranked element.

Brickell and Shmatikov [6] use a similar approach to construct semi-honest secure computation protocols for the all pairs shortest distance (APSD) and single source shortest distance (SSSD) problems. In both

cases, their protocols are more efficient than circuit-based secure computation protocols. While the work of [6] considers only the two-party setting the work of [1] additionally considers the multiparty setting.

1.1 Our Results

We continue the study of reducing the secure evaluation of function f to secure evaluation of comparisons in the *two-party* setting. Our first contribution is to generalize the approach of Aggarwal, Mishra, and Pinkas and that of Brickell and Shmatikov as the parameterized protocol in Fig. 1. The parameters to this protocol are the comparison function $\mathbf{LT}_f$, and the method F. The comparison function takes two input elements with their corresponding key values and returns the element with the smaller key value; F is the local update function that determines how each party determines its input for the next iteration based on the answers from the previous iteration $(c_1 \ldots, c_j)$ and its local input U (or V).

GENERIC ITERATIVE SECURE COMPUTATION

Alice Input: Distinct elements $U = \{u_1, \ldots, u_n\}$

Bob Input: Distinct elements $V = \{v_1, \ldots, v_n\}$

Output: The final output is $c_1, \ldots, c_\ell$

1. Alice initializes $(u_a, k_a) \leftarrow F(\perp, U)$ and Bob initializes $(v_b, k_b) \leftarrow F(\perp, V)$.
2. Repeat for $\ell(|U|, |V|)$ times:
   (a) Alice and Bob execute the secure protocol $c_j \leftarrow \mathbf{LT}_f((u_a, k_a), (v_b, k_b))$.
   (b) Alice updates $(u_a, k_a) \leftarrow F((c_1, \ldots, c_j), U)$ and Bob updates $(v_b, k_b) \leftarrow F((c_1, \ldots, c_j), V)$.

Fig. 1. The generic structure of a secure iterative protocol.

We show that this parameterized protocol can be used to construct more efficient secure computation protocols for a much larger class of optimization problems than the three specific instances they considered. In §4, we construct choices for $\mathbf{LT}, F$ that securely compute several combinatorial optimization problems, matroid optimization problems, sub-modular optimizations, and computation of the convex hull.

A key reason for the improved efficiency of this approach over both circuits and ORAM techniques is the fact that the output is *gradually released* to both parties. The result of one iteration of the loop is used to select inputs to the next iteration of the loop; more generally, the output

of the secure computation can be thought to be released *bit-by-bit*, *node-by-node*, or *edge-by-edge*. Thus it is not immediately clear that such an approach can be secure, even against honest-but-curious adversaries.

Our next contribution is to show that an instantiation of the above generic protocol can be made secure in the honest-but-curious model when the functions f is *greedy-compatible*, i.e. it satisfies a few simple properties. First, the problem must have a unique solution. One can often guarantee this property by specifying simple rules to break ties among comparisons. Second, the order in which the output is revealed must be unique, and finally, we require a local updatability property for the function F which essentially states that F has a very weak homomorphism with the comparison function **LT**. (See definition 1). When these three conditions are met, and when the $\mathbf{LT}_f$ function can be securely evaluated efficiently, then the instantiated protocol can be asymptotically (and practically) superior to other approaches. See Table 2 for several examples.

Algorithm	This Work	Circuit	ORAM
Convex Hull	$O(\|Z\|l_M)$	$\Omega(\|I\|^2 l_M)$	$\Omega(\|I\| \log^3 \|I\| l_M)$
MST	$O(Vl_M)$	$\Omega((V\alpha(V))^2 l_M)$	$\Omega(V\alpha(V) \log^3 Vl_M)$
Unit Job Sched	$O(\|Z\|l_M)$	$\Omega(\|I\|^2 l_M)$	$\Omega(\|I\| \log^3 \|I\| l_M)$
Single-Src ADSP	$O(Vl_M)$	$\Omega(E^2 l_M)$	$\Omega(E \log^3 El_M)$
Submodular Opt	$O(\|Z\|l_M)$	$\Omega(\|I_S\|^2 l_M)$	$\Omega(\|I_S\| \log \|I_S\| l_M)$

Fig. 2. Communication costs for secure protocols in the semi-honest case. $I = U \cup V$ the union of Alice and Bob's inputs and Z is the output. V and E are the number of vertices and edges in graph problems. $\alpha(\cdot)$ is the Inverse Ackermann function. For problems where each element is a set, I_S represents the sum of the set sizes. $l_M = \log M$ where M typically represents the maximum integer values the inputs can take. For each case, the complexity for the generic Circuit-based approach was obtained by relating it to the number of (dependent) memory accesses made by the best algorithm and the ORAM complexity was obtained by relating it to the time complexity of best known algorithm. In many cases, our communication complexity is related to the output-size, which can be much smaller than the input size for many problems.

1.2 Malicious and Covert Security

We also consider stronger notions of security for our protocols, namely security against fully malicious adversaries, and security against covert adversaries (which can be achieved much more efficiently) in the two-party setting.

Recall from the previous section that efficiency gain of our approach owes in part to the gradual release of output during each iteration. Herein lies a difficulty: A malicious adversary can manipulate its input used at each iteration of the protocol based on the results from previous iterations. This cheating ability complicates the construction of a proper Ideal-simulator.

As a first step, we can require the adversary to commit to its input before the protocol starts and force the adversary to only use committed data. To perform a simulation, we will first attempt to extract the adversaries' input, and then use this input to simulate the rest of the computation. We can use standard ideas with extractable commitments to perform this extraction. However, this is not enough. The main technical problem arises in the case that the adversary *selectively aborts* or *selectively uses* his committed inputs in the iterative protocol based on the intermediate results of the computation.

The prior work of Aggarwal, Mishra and Pinkas [1] claim that their simulation works for malicious adversaries; however, their simulation fails to account for the case when the adversary *selectively aborts*. Our second technical contributions is to present a *hardened* version of the protocol by Aggarwal, Mishra and Pinkas [1] for securely computing the median and a simulation argument which proves that it achieves malicious security.

As we discuss in §6.1, the techniques we use to show full malicious security rely on two specific properties that holds for the median problem. At a high level, for any input A of Alice and any element $a \in A$, we need that only one sequence of outputs from the previous iterations of our general protocol framework lead to Alice using a as an input. Furthermore, there are at most polynomially-many execution "traces" for any set of inputs from Alice and Bob (in contrast, graph problems have exponentially many traces). If there were multiple traces that lead to the use of element $a \in A$, then the adversary can selectively decide to abort on one of the traces and such an adversary cannot be simulated since its view depends on the trace and therefore the honest party's input. If the second property fails to hold then it can be argued that it would be hard for the simulator to extract the "right" input of the adversary.

Indeed, the selective abort issue seems to be a fundamental bottleneck to overcome. When these two properties fail to hold, e.g. in the case of the convex hull, or submodular optimization problems, we augment our basic protocol into one that achieves *covert security* as introduced Aumann and Lindell [2]. Covert security guarantees that if an adversary deviates in a way that would enable it to "cheat", then the honest party is

Algorithm	This Work (covert)	Circuit (malicious)
Convex Hull	$O(\|Z\|l_M + \|I\|l_M^2)$	$\Omega(\|I\|^2 l_M)$
MST	$O(V \log V l_M)$	$\Omega((V\alpha(V))^2 l_M)$
Unit Job Scheduling	$O((\|Z\| + \|I\|)l_M)$	$\Omega(\|I\|^2 l_M)$
Single-Source ADSP	$O((V + E)l_M)$	$\Omega(E^2 l_M)$

Fig. 3. Comparison of the communication costs of covert security with the malicious security using circuits ignoring $poly(k)$ factors. $l_M = \log M$. $I = U \cup V$ the union of Alice and Bob's inputs. V and E are the number of vertices and edges in graph problems. We remark that since we ignore $poly(k)$ factors, the complexity of the Circuit-based approach would be the same as above even if we considered only achieving covert security. We were unable to estimate costs for achieving malicious security with ORAM.

guaranteed to detect the cheating with reasonable probability. The covert model handles situations in which a malicious party has a strong incentive "not to be caught cheating," while offering substantial improvements in efficiency versus the fully malicious model.

To achieve covert security, we must handle both selective aborts, and also ensure that an adversary does not cheat by using only a subset of its committed input during the protocol (perhaps based on a predicate of intermediate output). To handle this last issue, we require the adversary to prove at the end of the protocol that all of the committed inputs are either part of the output or used properly during the protocol. The adversary will provide one proof per "element" of the input, and thus, we need to design proofs that are sub-linear in the input size n, preferably logarithmic or even constant-sized.

For each of our selected problems, we provide these novel consistency checks. In cases such as the convex-hull, single-source shortest paths and job-scheduling, these checks are simple and have constant size (modulo the security parameter). For the case of the Minimum Spanning Tree, however, we required an elegant application of the Union-Find data structure to achieve communication efficient consistency checks for this problem. We summarize our performance for many problems in Table 3.

Although one might be able to use either Universal arguments [19, 3] or SNARKS [8, 5] to achieve malicious security with low communication, both of those techniques dramatically increase the computational overhead of the protocol. In particular, when proving an NP-relation of size t on an input statement x, the prover's computational complexity is proportional to $\tilde{O}(t)$ and the verifier's computational complexity is pro-

GENERALIZED COMPARE

Alice Input: Tuple (u, x) with k-bit integer key x

Bob Input: Tuple (v, y) k-bit integer key y

LT$_f$ Output: Return u if $x < y$ and v otherwise

Fig. 4. Generic comparison protocol

portional to $\tilde{O}(|x|)$. In our context, since such a proof will be required in each iteration, the computational complexity for both parties would be $\tilde{O}(|Z| \times |I|) + c(f)$ where Z and I are the inputs and outputs of the computation and $c(f)$ is the complexity for computing f itself. In contrast, our covert security and semi-honest protocol computation complexity is $O(|Z| + |I|) + c(f)$.

2 Preliminaries

We denote $(c_1, \ldots, c_j)$ by $c_{\leq j}$. Two sequences of distributions $\{C_n\}_{n\in\mathbb{N}}$ and $\{D_n\}_{n\in\mathbb{N}}$ are said to be computationally indistinguishable if for any probabilistic polynomial time algorithm A, $|\Pr[A(C_n) = 1] - \Pr[A(D_n) = 1]$ is a negligible function in n. We formally describe a generalized millionaire (comparison) function in Fig. 4.

3 Honest-But-Curious Protocols

For many well-known greedy algorithms we show how to securely compute them using our Generic Protocol specified in Figure 1. On a high-level, in this protocol abstraction, Alice and Bob have a set of inputs U and V. In every iteration, each of them provide an input element e from their local inputs and an associated key k_e to the comparison functionality $\mathbf{LT}_f$ which returns the element with smaller key value. More precisely, in iteration i, Alice supplies input (u_a, k_a) and Bob supplies input (v_b, k_b) where $u_a \in U$ and $v_b \in V$ to the $\mathbf{LT}_f$-functionality. The functionality returns as output $c_i = k_a < k_b \ ?\ u_a \ :\ v_b$. At the end of each iteration there is a local update rule that determines the next input and key for the next iteration. Finally, Alice and Bob output $c_1, \ldots, c_\ell$ as their outputs.

We make the following requirements on the function f that we wish to compute using a greedy-algorithm. For each instantiation, we show that the requirements are satisfied.

Definition 1. *We say that a two-party function f is greedy compatible if there exists functions $\boldsymbol{LT}, F$ such that the following holds:*

1. ***Unique Solution:*** *Given the inputs U and V of Alice and Bob, there is a unique solution.*
2. ***Unique Order:*** *There is a unique order in which the greedy-strategy outputs the solution. More precisely,*

$$f(U, V) = (c_1, \ldots, c_\ell)$$

where $c_1 = F(\perp, U \cup V)$ and $c_{i+1} = F(c_{\leq i}, U \cup V)$ for every $i = 1, \ldots, \ell - 1$.
3. ***Local Updatability:*** *Informally, we require that F on the union of Alice and Bob's inputs can be obtained by applying F locally to U and V and then computing a comparison. More precisely, we require that*

$$F_1(c_{\leq j}, U \cup V) = \boldsymbol{LT}_f(F(c_{\leq j}, U), F(c_{\leq j}, V))$$

where F_1 represents the first member in the tuple output by F.

3.1 Honest-but-Curious Security

Theorem 1. *For any function f that is greedy compatible, the Generic Iterative Secure Computation algorithm from Figure 1 securely computes f on the union of the inputs held by Alice and Bob, for the case of semi-honest adversaries.*

We argue correctness and privacy of our protocols. Our analysis of the protocol will be in the $\mathbf{LT}_f$ hybrid, where the parties are assumed to have access to a trusted party computing the $\mathbf{LT}_f$.

Correctness: First we observe that if U and V are Alice and Bob's inputs, then from the Unique Order property it holds that $f(U, V) = (c_1, \ldots, c_\ell)$ where $c_1 = F(\perp, U \cup V)$ and $c_{i+1} = F(c_{\leq i}, U \cup V)$ for $i = 1, \ldots, \ell - 1$. The output computed by Alice and Bob by executing the Generic Iterative Secure Computation algorithm is $\tilde{c}_1, \ldots, \tilde{c}_\ell$ where

$$\begin{aligned} \tilde{c}_1 &= \mathbf{LT}_f(F(\perp, U), F(\perp, V)) \\ \tilde{c}_{i+1} &= \mathbf{LT}_f(F(\tilde{c}_{\leq i}, U), F(\tilde{c}_{\leq i}, V)) \text{ for } i \text{ in } \{1, \ldots, \ell - 1\} \end{aligned}$$

Correctness now follows from the Local Updatability property of f.

Privacy: Next to prove security in the honest-but-curious case, we construct a simulator that given the parties input and output can simulate the interaction indistinguishably.

Recall that, our analysis of the security of the protocol is in the $\mathbf{LT}_f$ hybrid. Thus the simulator that we describe will play the trusted party implementing $\mathbf{LT}_f$, when simulating the adversary. Below we prove security when one of the parties are corrupted. We argue for the case when Alice is corrupted and the case for Bob follows symmetrically since the protocol is symmetric in both parties.

Alice is corrupted. The simulator needs to produce a transcript indistinguishable to the honest adversary A_h in the $\mathbf{LT}_f$ hybrid.

- The simulator upon corrupting Alice receives her input U. It feeds U to the ideal functionality computing f to receive the output $c_{\leq l}$.
- Next run the honest Alice's code for the Generic Algorithm. Alice in iteration i for $i = 1, \ldots, \ell$, submits an input (u_a, k_a) to the $\mathbf{LT}_f$ functionality. S simulates the output by feeding c_i to Alice.
- Finally, at the end of ℓ-iterations, S outputs the view of Alice.

From the Unique Order property and the fact that Alice is honest, the view generated by S in the $\mathbf{LT}_f$-hybrid is identical to the view of Alice in the real experiment. More precisely,

$$\text{IDEAL}^{\mathbf{LT}_f}_{f,S(z),I}(U, V, k) \equiv \text{REAL}^{\mathbf{LT}_f}_{f,A_h(z),I}(U, V, k)$$

Therefore, from the composition theorem stated in Section **??**, security against semi-honest adversaries holds. This concludes the proof sketch of Theorem 1.

4 Instantiations of Our Protocol

4.1 Convex Hull

In this problem, Alice and Bob have as input sets of points U and V in a plane and the goal is to securely compute the convex hull of the union of points. Each element $u = (x, y)$ consists of two $\log M$-bit integers that represent the X and Y coordinate of the point. We assume that the union of points are such that no two points share the same X-coordinate and no three of them are collinear. The function F for the convex hull is defined as $F(c_{\leq j}, U) = (u_a, k_a)$ where:

– If $j = 0$, then u_a is point with the least value for the x-coordinate (i.e. the leftmost point) and k_a is set to be the x-coordinate of u_a.
– If $j > 0$, u_a is the point in U that attains the minimum value for $\mathsf{angle}(c_j, c)$ where $\mathsf{angle}(\mathsf{pt}_1, \mathsf{pt}_2)$ is the (clockwise) angle made by the line joining pt_1 and pt_2 with the vertical drawn through pt_1 and $k_a = \mathsf{angle}(c_j, a)$.

The correctness of the Convex-hull instantiation follows from the Gift-Wrapping (or Jarvis march) algorithm. Furthermore, it is easy to verify that Convex Hull is greedy compatible with F if no two-points have the same x or y coordinate and no three-points are collinear. Hence, we have the following theorem.

Theorem 2. *The* Generic Iterative Secure Computation *protocol instantiated with the F described above securely computes the convex hull of the union of inputs of Alice and Bob, for the case of semi-honest adversaries, assuming all inputs of Alice and Bob are distinct, no two of which share the same x-coordinate and no three points are collinear.*

Overhead: The total number of rounds of communication is $|Z|$, the size of the convex-hull Z which is at most $|I|$ where $I = U \cup V$. In each round, the protocol performs at most one secure comparison of $\log M$-bit integers. A circuit for performing the comparison has $O(\log M)$ gates and $\log M$ inputs. The overhead of the protocol for computing this circuit, secure against semi-honest adversaries, is $\log M$ oblivious-transfers. This can be thought of as $O(\log M)$ public-key operations, $O(\log M)$ symmetric key operations and communication of $O(\log M)$. The overall communication complexity is $O(|Z| \log M)$.

In comparison, the naive circuit implementation will have at least $|I|$ (dependent) memory accesses which will result in a circuit size of $\Omega(|I|^2 \log M)$. If we considered an ORAM implementation it would result in total communication of $\Omega(|I| \log^3 |I| \log M)$ since the best algorithm would require $O(|I| \log |I|)$ steps and the overhead for each step is $\log^2 |I|$ since we need to maintain a memory of size $O(|I|)$.

In the full version, we provide more examples: Job Interval Scheduling problem; general Matroid optimization problems for which membership in set $\mathcal{I}$ can be tested locally including minimum spanning tree problems and unit cost scheduling problems; the single-source shortest distance problem; and sub modular optimization problems such as set-cover and max cover approximations.

5 Covert Security

We describe the main issues to overcome with the current protocol:

Adaptively chosen inputs As our iterative protocol gradually releases the answer, it is possible for the adversary to modify its input as the protocol proceeds. To defend, we include an input commitment phase. Then in the secure computation phase, the adversary provides decommitments with every input it uses in the computation of the LT_f-functionality.

Missing inputs Consider an adversary that commits to its inputs but fails to follow the greedy strategy, namely, does not perform the local update rule using F honestly. This is an issue even if the adversary is restricted to only use inputs that it committed to because it can adaptively decide to use only a subset of them. Consider the minimum spanning tree problem in which the adversary can decide to drop a certain edge based on the partial output released before an iteration. To prevent this attack, we will rely on digital signatures.

Alice and Bob will first pick signature keys and share their verification keys. Next, in every computation using LT_f, Alice and Bob will obtain signatures of the output along with some specific *auxiliary* information that will later be used by each party to demonstrate honest behavior. More precisely, after the secure computation phase and the output is obtained, for every input $u \in U$ of Alice, it does the following:

- If u is part of the output, then we require Alice to prove to Bob that it has a signature on u under Bob's key and modify $\mathbf{LT}_f$ to reveal the Commitment of u to Bob in that iteration. This will allow Bob to determine which of the Commitments made by Alice in the input commitment phase is not part of the output.
- If u is not part of the output, Alice proves to Bob that u is not part of the solution. We prove in many of our examples how we can achieve this efficiently. In essence, Alice will show that in the iteration after which u was eliminated, a better element was chosen. For instance, in the minimum spanning tree problem, we demonstrate that an edge $e = (a, b)$ was eliminated because a cheaper edge e' got added to the output that connected the components containing vertices a and b.

Input Commitment Phase: To resolve, we add an Input Commitment Phase at the beginning of the protocol and a Consistency-Check Phase at the end of the protocol described in the Section 5.1. In an Input Commitment Phase executed at the beginning of the protocol, both parties commit to their input using an extractable commitment scheme Π_{Ext}.

Modifications to $\boldsymbol{LT}_f$ functionality: Besides the inputs (u_a, k_a) and (v_b, k_b) that Alice and Bob submit, they also submit $(\mathsf{aux}_a, \mathsf{sk}_A)$ and $(\mathsf{aux}_b, \mathsf{sk}_B)$ which are the auxiliary information corresponding to their inputs and their signing keys respectively. The function besides outputting the answer as in Figure 1 additionally signs (u_a, aux_a) if u_a is the output and (u_b, aux_b) if u_b is the output using both keys sk_A and sk_B. We remark here that for modularity we describe that the signatures are computed by **LT** functionality. However, in all our instantiations the message to be signed (u, aux) in the i^{th} iteration can be computed directly from the outputs of the current and previous calls to the $\mathbf{LT}_f$ functionality, namely, $c_1, \ldots, c_i$ and signature of these messages under the keys of Alice and Bob can be computed and sent directly to the other party. In particular, these signatures need not be computed securely.

Consistency-Check Phase: Alice and Bob need to prove they followed the greedy strategy at every iteration. Recall that, c_i for each i belongs to Alice or Bob. Alice proves that corresponding to every commitment C in the Input Commitment Phase, there exists an input u such that either

- u is one of the c_i's and it has a signature on c_i using sk_B, or
- u could not have been selected by the greedy strategy.

We achieve this by securely evaluating this consistency check procedure where in the first case, u is revealed to both parties and in the second case, only the result of the check is revealed.

5.1 Generic Algorithm for Covert Security

GENERALIZED COMPARE WITH COVERT SECURITY

Alice Input: Tuple $(u_a, x, \mathsf{aux}_a, \mathsf{sk}_A)$ with k-bit integer key x

Bob Input: Tuple $(v_b, y, \mathsf{aux}_b, \mathsf{sk}_B)$ k-bit integer key y

$\mathbf{LT}_f$ Output: $(u_a, \mathsf{aux}_a, \sigma_A, \sigma_B)$ if $x < y$ and $(v_b, \mathsf{aux}_b, \sigma_A, \sigma_B)$ otherwise where σ_A and σ_B are signatures on message m under keys sk_A and sk_B respectively and $m = (u_a, \mathsf{aux}_a)$ if $x < y$ and $m = (v_b, \mathsf{aux}_b)$ otherwise.

Fig. 5. Generic Millionaire's protocol with Covert security

We make the following requirements on the function f we compute. For each instantiation, we show that the requirements are satisfied. We say that a function f is covert-greedy compatible if it is greedy compatible and additionally the following holds:

GENERIC CONSISTENCY CHECK

Prover Input: Tuple $(u, \mathsf{key}(u), C, D, \{u_i, \mathsf{aux}_i, \sigma^i\}_{i \in I})$

CC_{vk} **Output:** It outputs 1 to Verifier and additionally outputs u when $u = u_i$ for some i, if all the following hold:

1. *Correct Input:* D is a valid decommitment information for C to $(u, \mathsf{key}(u))$.
2. *Consistency Check:* Either $CC(u, \mathsf{key}(u), \{u_i, \mathsf{aux}_i\}_{i \in I})$ returns true or $u_i = u$ for some $i \in I$.
3. *Signature Check:* $\mathsf{Ver}_{\mathsf{vk}}((u_i, \mathsf{aux}_i), \sigma^i) = 1$ for $i \in I$

Fig. 6. Generic Consistency Check Procedure GCC_{vk}

– **Consistency Check:** There exists a consistency-check procedure CC, functions key, wit and aux which satisfies the following property: Given inputs U and V for Alice and Bob and any output $\tilde{c}_1, \ldots, \tilde{c}_\ell$, it holds that, for every input u of Alice (respectively, v of Bob), such that: $CC(u, \mathsf{key}(u), \{\tilde{c}_i, \mathsf{aux}(i, \tilde{c}_i)\}_{i \in I})$
  - Returns TRUE: if u is not part of the solution and $I = \mathsf{wit}(U)$ or $u = u_i$ for some $i \in I$
  - Returns FALSE: if u is in the solution and $u \neq u_i$ for any $i \in I$.

  Furthermore, we require that $\mathsf{aux}(u_a)$ for an input u_a in iteration i can be determined by $c_{<i}$.

Let $\Pi_{\mathsf{Ext}} = \langle C, R \rangle$ be an extractable commitment scheme. In Figure 7, we give our general protocol to achieve covert security. Then for each of our problems, we specify how we modify the $\mathbf{LT}_f$ functionality and provide the Consistency Check procedure. Let com be a statistically-binding commitment scheme. In Figure fig:conc-check, we give the generic structure of the consistency-check procedure.

Theorem 3. *Let f be a functionality that is covert-greedy compatible. Then the Generic Covert Security protocol described in Figure 7 securely computes f in the presence of covert adversaries with 1-deterrence.*

Proof. Our analysis of the security of the protocol is in the $\mathbf{LT}_f, CC_{\mathsf{vk}}$ hybrid, where the parties are assumed to have access to a trusted party computing the respective functionalities. Thus the simulator that we describe will play the trusted party implementing the two functionalities, when simulating the adversary. We consider the different corruption cases: (1) When no party is corrupted (2) When one of the parties are corrupted. In the first case, the security reduces to the semi-honest case and follows

GENERIC ITERATIVE SECURE COMPUTATION WITH COVERT SECURITY

Alice Input: A set of distinct elements $U = \{u_1, \ldots, u_n\}$

Bob Input: A set of distinct elements $V = \{v_1, \ldots, v_n\}$

Output: $c_1, \ldots, c_\ell$.

Input Commitment Phase:

1. For every $i \in [n]$, Alice acting as the Sender with input $m = (u_i, \mathsf{key}(u_i))$ interacts with Bob as the Receiver using the protocol Π_{Ext}.
2. For every $i \in [n]$, Bob acting as the Sender with input $m = (v_i, \mathsf{key}(v_i))$ interacts with Alice as the Receiver using the protocol Π_{Ext}.
3. Alice and Bob run $\mathsf{Gen}(1^k)$ to obtain the key-pairs $(\mathsf{sk}_A, \mathsf{vk}_A)$ and $(\mathsf{sk}_B, \mathsf{vk}_B)$ respectively. Alice sends vk_A to Bob and Bob sends vk_B to Alice.

Secure Computation Phase:

1. Alice initializes $(u_a, k_a) \leftarrow F(\perp, U)$ and Bob initializes $(v_b, k_b) \leftarrow F(\perp, V)$.
2. Repeat for $\ell(|U|, |V|)$ times:
   (a) Alice and Bob execute the protocol computing $\mathbf{LT}_{f,\mathsf{vk}_A,\mathsf{vk}_B}$ on inputs $(u_a, k_a, \mathsf{aux}(i, u_a), \mathsf{sk}_A)$ and $(v_b, k_b, \mathsf{aux}(i, u_b), \mathsf{sk}_B))$ and receives as output $(c_j, \sigma_A, \sigma_B)$ where i is the iteration number.
   (b) Alice updates $(u_a, k_a) \leftarrow F(c_{\leq j}, U)$ and Bob updates $(v_b, k_b) \leftarrow F(c_{\leq j}, V)$. They store σ_B and σ_A respectively.
3. Alice outputs $c_1, \ldots, c_\ell$. Bob outputs $c_1, \ldots, c_\ell$.

Consistency Check Phase:

1. For every commitment C made by Alice in the Input Commitment Phase to an element u_a, Alice and Bob execute the protocol

$$CC_{\mathsf{vk}_B}((u_a, key(u_a), C, D, \sigma, \{(c_i, \mathsf{aux}_i, \sigma_B^i)\}_{i \in I}), \perp)$$

   where (a) $I = \mathsf{wit}(u_a)$, if u_a is not part of the output, (b) σ is a signature on a message of the form $(u_a, \cdot)$, otherwise. If CC outputs 0, then Bob outputs $corrupt_A$. If CC returns u_a, Bob stores u_a in store $OutCheck_A$.
2. For every commitment C made by Bob in the Input Commitment Phase to an element v_b, Alice and Bob execute the protocol

$$CC_{\mathsf{vk}_A}((v_b, key(v_b), C, D, \sigma, \{(c_i, \mathsf{aux}_i, \sigma_A^i)\}_{i \in I}), \perp)$$

   where (a) $I = \mathsf{wit}(v_b)$, if v_b is not part of the output, (b)σ is a signature on a message of the form $(v_b, \cdot)$, otherwise. If CC outputs 0, then Alice outputs $corrupt_B$. If CC returns v_b, Bob stores v_b in store $OutCheck_B$.
3. If $OutCheck_A$ is not equal to $c_{\leq \ell}$ minus the elements that are part of Bob's input, then Bob outputs $corrupt_A$.
4. If $OutCheck_B$ is not equal to $c_{\leq \ell}$ minus the elements that are part of Alice's input, then Alice outputs $corrupt_A$.

Fig. 7. The generic structure of a secure iterative protocol with covert security

the proof presented in Section 3.1. Below we prove security when one of the parties are corrupted. We argue for the case when Alice is corrupted

and the case for Bob follows symmetrically since the protocol is symmetric in both parties.

Alice is corrupted. On a high-level, by our assumptions there is a unique solution and a unique order in which the output is revealed in every iteration. More precisely, given the optimal solution, the output of each iteration using $\mathbf{LT}_f$ is determined. Furthermore, this output is either an element in Alice's input or Bob's input. The simulator S fixes A's random tape unfiromly at random and proceeds as follows:

1. S executes the Input Commitment Phase playing the role of Bob. For all commitments made by Alice, S runs the extractor algorithm E provided by the Π_{Ext} protocol to create a transcript and extract all of Alice's input. For all of Bob's, S commits to the all 0 string.
2. Now S has Alice's input which it feeds to the ideal functionality computing f and receives the output $c_1, \ldots, c_\ell$. Next S interacts with Alice in the Secure Computation Phase. In iteration i, S receives Alice's input $(u_a, x, \mathsf{aux}_a, \mathsf{sk}_A)$ for $\mathbf{LT}_f$. S can check if Alice's input is correct, by performing the computation of $\mathbf{LT}_f$ with Alice's input as (u_a, x). If $u_a = c_i$ then S simply outputs c_i as the output of the computation. Otherwise, c_i must be Bob's input and the simulator checks if the computation with Bob's input as c_i results in c_i. If it is not, S outputs BAD_i and halts.
3. If S successfully completes the Secure Computation Phase, it proceeds to simulate the Consistency Check Phase. In this phase, Alice first proves consistency for every commitment C it made in the Input Commitment Phase by providing input to the CC_{vk_B} functionality. The simulator evaluates the input using the procedure honestly and sends $corrupt_A$ if the procedure returns 0.
4. Finally, for every Commitment made by Bob that is not part of the input, S simply sends what the CC_{vk} functionality should send if Bob is honest, namely, it sends 1 to Alice.

This concludes the description of the simulator S. We now proceed to prove covert-security. First, we prove the following claim.

Consider an adversarial Alice A^*. We prove indistinguishability in a hybrid experiment H where we construct another simulator S' that knows Bob's input. In this experiment S' proceeds identically to S with the exception that in the Input Commitment Phase it commits to the real inputs of Bob instead of the all 0 string as S would. Indistinguishability of the output of S and S' follows directly from the hiding property of

the commitment scheme and the fact that the views are in the $\mathbf{LT}, CC$-hybrid. More precisely,

$$\text{IDEAL}^{\mathbf{LT},CC}_{f,S(z),I}(x_1, x_2, k) \approx \text{IDEAL}^{\mathbf{LT},CC}_{f,S'(z),I}(x_1, x_2, k)$$

Next, we argue indistinguishability of the hybrid experiment H and the real experiment. First, we observe that both these experiment proceed identically in the Input Commitment Phase. Fix a partial transcript τ of the view at the end of Input Commitment Phase. Let $c_1, \ldots, c_\ell$ be the output obtained by S'. It now follows that, conditioned on the Secure Computation Phase outputting $c_1, \ldots, c_\ell$, the views in H and the real experiment are identically distributed. This is because the simulator honestly computes the $\mathbf{LT}$-functionality and the output is completely determined by $c_{\leq i}$ (c_i is the output of the iteration and the other previous outputs are required to determine aux_b). For any view v of the experiment, let $\mathsf{Success}(v)$ denote this event. Let BAD denote the union of all events BAD_i. It follows that

$$\begin{aligned}&\Pr[v \leftarrow \text{IDEAL}^{\mathbf{LT},CC}_{f,S'(z),I}(x_1, x_2, k) : D(v) = 1 \ \wedge \ \neg\text{BAD}] \\ &\quad = \Pr[v \leftarrow \text{REAL}^{\mathbf{LT},CC}_{f,A^*(z),I}(x_1, x_2, k)) : D(v) = 1 \ \wedge \ \mathsf{Success}(v)] \quad (1)\end{aligned}$$

Assume for contradiction, the simulation did not satisfy covert security with 1-deterrence. Then there exists an adversary A, distinguisher D and polynomial $p(\cdot)$ such that

$$\begin{aligned}&\Big|\Pr[D(\text{IDEAL}^{\mathbf{LT},CC}_{f,S'(z),I}(x_1, x_2, k)) = 1] - \Pr[D(\text{REAL}^{\mathbf{LT},CC}_{f,A^*(z),I}(x_1, x_2, k))) = 1]\Big| \\ &\qquad \geq \Pr[out_B(\text{REAL}_{f,A^*(z),I}(x_1, x_2, k)) = \mathsf{corrupt}_A] + \frac{1}{p(k)}\end{aligned}$$

Using Equation 1, we rewrite the above equation as follows:

$$\begin{aligned}&\Big|\Pr[v \leftarrow \text{IDEAL}^{\mathbf{LT},CC}_{f,S'(z),I}(x_1, x_2, k) : D(v) = 1 \ \wedge \ \text{BAD}] \\ &\qquad - \Pr[v \leftarrow \text{REAL}^{\mathbf{LT},CC}_{f,A^*(z),I}(x_1, x_2, k)) : D(v) = 1 \ \wedge \ \neg\mathsf{Success}(v)]\Big| \\ &\qquad\quad \geq \Pr[out_B(\text{REAL}_{f,A^*(z),I}(x_1, x_2, k)) = \mathsf{corrupt}_A] + \frac{1}{p(k)} \quad (2)\end{aligned}$$

Below in Claim 1, we show that $\Pr[\text{BAD}]$ is negligible close to $\Pr[\neg\mathsf{Success}]$. Furthemore, if $\neg\mathsf{Success}$ occurs, then it must be the case that Bob outputs

$corrupt_A$. Therefore,

$$\begin{aligned}
&\Big|\Pr[v \leftarrow \text{IDEAL}^{\mathbf{LT},CC}_{f,S'(z),I}(x_1,x_2,k) : D(v)=1 \ \wedge \ \text{BAD}] \\
&\qquad - \Pr[v \leftarrow \text{REAL}^{\mathbf{LT},CC}_{f,A^*(z),I}(x_1,x_2,k)) : D(v)=1 \ \wedge \ \neg\mathsf{Success}(v)]\Big| \\
&\leq \Pr[\neg\mathsf{Success}] - \mu_1(n) \\
&= \Pr[out_B(\text{REAL}_{f,A^*(z),I}(x_1,x_2,k)) = \mathsf{corrupt}_A] - \mu_1(k)
\end{aligned}$$

This is a contradiction to Equation 2.

Claim 1. $\Big|\Pr[\text{BAD}] - \Pr[\neg\mathsf{Success}]\Big| < \mu_1(k)$
where the first probability is over the experiment in H and the second is over the real experiment.

Proof: Observe that if BAD occurs, then S' outputs BAD_i for some i. This means that in iteration i, the result of the computation using **LT** was not c_i. Since, we consider unique inputs, it must be the case that the output was something different from c_i. There are two cases:

c_i **was part of Alice's input** In this case, Alice must not have used the input corresponding to c_i in iteration i. Hence, the output of the i^{th} iteration must have been different. Since there is a unique order in which the outputs are revealed, the computation between Alice and Bob must have resulted in an output different from $c_{\leq \ell}$. Then by the the consistency-check property of f, it will follow that Alice cannot convince Bob in the Consistency-Check Phase for the commitment corresponding to C_i on the same transcript output by S'. This means that Bob would output $corrupt_A$ in Step 1 on such a transcript.

c_i **was part of Bob's input** Suppose that Alice used an input u in iteration i that was chosen by the greedy procedure. In this case, it cannot be that Alice committed to u in the Input Commitment Phase. This is because, S' extracted all the inputs from Alice and the output is unique given Alice's input. In this case, we show that Alice will fail in the Consistency-Check Procedure. First, we observe that Alice cannot produce an input to CC such that the output is u. Recall that it would have to produce both a signature using Bob's key for u and a commitment C from the input commitment phase containing the input u, since there is no such commitment, it cannot achieve this. Hence, $OutCheck_A$ computed by Bob will not contain u but $c_{\leq \ell}$ does. Therefore, Bob will output $corrupt_A$ in Step 3.

We recall that the view in H and the real experiments are identically distributed up until the iteration where BAD_i occurs. Therefore, from the above argument, it follows that if BAD_i was output by S_i on any partial transcript τ in hybrid experiment H up until iteration i, then Bob must have output $corrupt_A$ on a continuation from τ in the real experiment except with negligible probability. This concludes the proof of the Claim and the Theorem. □

5.2 Convex Hull

We present a consistency-check for the convex-hull problem that will provide covert-security with 1-deterrence. Recall from the general covert-secure protocol that an adversary can fail to consider some of its input committed in the Commitment Phase. In the Consistency-Check Phase, the adversary needs to show that a particular point p committed to is not part of the convex hull. Towards this, it will choose three points p_1, p_2 and p_3 on the convex-hull that was output and prove that p lies strictly in the interior of the triangle formed by the p_1, p_2 and p_3. As before we assume that no two points share the same x or y coordinate and no three points are collinear.

We describe below how to instantiate this problem in our framework. First we show that convex-hull is covert-greedy compatible:

greedy compatible From Section 4.1 we know this is greedy compatible.

Consistency Check: We define the $\mathsf{key}(u) = \perp$ and $\mathsf{aux}(i, u) = \perp$. The function wit on input u is defined to be the index of three points in the output $c_1, \ldots, c_\ell$ such that u resides within the triangle formed by the three points. Observe that if a particular point u is not on the convex-hull, it lies inside and there must be three points on the hull for which u is contained in the triangle formed by the three points. Moreover, for any point outside the hull, there exists no set of three points for which this conditions will be true. The function CC on input $(u, C, D, (u_1, u_2, u_3))$ outputs 1 only if u is contained in the triangle formed by u_1, u_2 and u_3.

Theorem 4. *The* Generic Iterative Secure Computation with Covert Security *protocol instantiated with the Consistency Check Procedure CC, functions* aux, key *and* wit *described above securely computes the convex hull of the union of inputs of Alice and Bob, in the presence of covert-adversaries with* 1*-deterrence, assuming all inputs of Alice and Bob are distinct, no two of which share the same x-coordinate and no three points are collinear.*

Overhead: The total number of rounds of communication is $O(|Z| + |I|)$. This is because the secure computation phase take $O(|Z|)$-rounds and the consistency check phase is $O(|I|)$-rounds. In each round of the secure computation phase, the protocol performs at most one secure comparison of $\log M$-bit integers and one signature computation on a $\log M$ bit string. As mentioned before, the signatures need not be securely computed and can be computed locally and sent to the other party. In particular, for the case of convex-hull, the message to be signed is the point output in the current iteration. Therefore, the communication complexity of each round of iteration in this phase is $O(\log M) + O(k)$. In each round of the consistency check phase, the protocol performs (a) One decommitment verification that will cost $poly(k)$ (b) $O(1)$ signature verifications that will cost $poly(k)$ (c) $O(1)$ subtractions of $\log M$-bit integers (this will require $O(\log M)$ gates) and $O(1)$ multiplications of $\log M$-bit integers (this will require $O(\log^2 M)$ gates). This is for checking if a point is in a triangle. Since all the circuits need to be securely computed against malicious adversaries there will be a $poly(k)$ overhead. The overall communication complexity will be $O(|Z| \log M + |I| \log^2 M)$ times $poly(k)$. In comparison, the naive circuit implementation will have at least n memory accesses which will result in a circuit size of $\Omega(|I|^2 \log M)$ times $poly(k)$.

5.3 Matroids

We begin with a simple consistency-check for matroids that will yield covert-security with 1-deterrence. The communication complexity of implementing this check would be $O(|S|)$ where the matroid is $(S, \mathcal{I})$.

We recall some basic notions regarding matroids (see [13]). All sets in $\mathcal{I}$ are referred to as *independent sets* and any set not in $\mathcal{I}$ is called a *dependent set*. A *cycle* of a matroid $(S, \mathcal{I})$ is a setwise minimal dependent set. A *cut* in $(S, \mathcal{I})$ is a setwise minimal subset of S intersecting all maximal independent sets. The names have been so chosen to maintain the intuitive connection with the special case of MST. [1] Suppose B is an independent set and $B \cup \{x\}$ is dependent. Then $B \cup \{x\}$ contains a cycle. We refer to this cycle as the *fundamental cycle* of x and B. The cycle must contain x, since $C - \{x\} \subseteq B$ and hence independent. The following proposition follows directly form the properties of a matroid.

Proposition 1. *An element $x \in S$ is in no minimum weight maximal independent set iff it has the largest weight on some cycle.*

[1] Note however, the notion of cuts in graphs are a union of cuts as defined here since the notion of a cut in a graph need not necessarily be setwise minimal.

This proposition will form the basis of the consistency-check as for every element x not in the minimum weight maximal independent set B, there is a unique cycle, i.e. the fundamental cycle of x and B which can be computed by both parties given B and x. Then this cycle can be used to demonstrate that x is not part of the solution. In the full version, we consider other examples such as the (a) minimum spanning tree,[2] (b) unit job scheduling, and (c) single source shortest distance examples.

6 Computing the Median: Revisiting the AMP protocol

To incorporate the median protocol to our framework we need to make a few modifications; the output of each iteration is not part of the final output. The output of the final iteration is the output of the protocol.

We define $\mathbf{LT}_f(x, y)$ function simply returns either 0 or 1 depending on whether $x > y$. The definition of F is slightly more complicated and subsumes the *pruning* of the input sets that is explicit in the AMP protocol. Specifically, we define

$$F_A(i, m, x, S) = \begin{cases} n/2 & \text{if } i = 0 \\ x/2 & \text{if } m = 0 \ \wedge \ i > 0 \\ x + x/2 & \text{if } m = 1 \ \wedge \ i > 0 \end{cases}$$
$$F_B(i, m, x, S) = F_A(i, \overline{m}, x, S)$$

In words, Alice begins the protocol using her $n/2^{\text{th}}$ (i.e. her median) element in comparison with Bob's median element. If Alice's element is larger, then she updates her index to the median of the smaller half of her array (i.e., the $n/4$ rank element) and conversely Bob updates his index to the median of the larger half of his array (i.e., his $n/2+n/4$ rank element). The loop is repeated $\ell(|U|, |V|) = \lceil \log(|U|) \rceil = \lceil \log(n) \rceil$ times and at the end of the looping, the output O.

Malicious Security for aborting adversaries The security proof in [1] for the median protocol does not handle the case when the adversary aborts during extraction. We improve their simulation to handle this case.

As in [1], we construct a simulator in a hybrid model where all parties have access to an ideal functionality that computes comparisons. In the malicious protocol, at every step a lower bound l and an upper bound u is maintained by secret sharing the values between the two parties. This enforces the input provided by the parties to lie between l and u. Consider

[2] It is possible to achieve better efficiency then the general matroid approach for MST.

an adversary A for the protocol. We can visualize the actions of A in each iteration as either going left or right in a binary tree depending on the output of the comparison at that iteration. In the malicious setting, the simulator is required to extract the adversary's input to feed it to the ideal functionality. Since the actual inputs are revealed at each node, the simulator needs a mechanism to visit every node. Since every node can be arrived by a sequence of left or right traversal to a child, the simulator can reach every node by providing the appropriate sequence of outputs for the comparison operation and then allowing the adversary to reveal the value at that node. This value is given as input to the next iteration. Since we are in the hybrid where the simulator can see every input that the adversary sends, the simulator can extract the entire tree. Recall that the adversary is required to provide inputs that respect the lower and upper bound. Hence after extraction, an *in-order* traversal of the binary tree gives a sorted set of values. Using this set as the adversary's input, the simulator feeds it to the ideal functionality and obtains the output, and then traverses the tree again giving the actual inputs as dictated by the median.

We show how to construct a simulator that also allows the adversary to abort. If the adversary aborts when the execution reaches some particular node, then the simulator will not be able to extract the values in the entire tree rooted at that node. Our simulation proceeds as follows. It explores the tree just as in [1] and marks the nodes on which the adversary aborted. At the end of the extraction, the simulator has a partial binary tree. However, the simulator needs to send some input the ideal functionality on behalf of the adversary to obtain the output. Towards that, the simulator extends the partial tree to a full binary tree by adding dummy nodes. Next, the simulator assigns values for the dummy nodes. To do that every dummy node is given a unique label and then following [1], we perform an in-order traversal of the tree (excluding leaves) to obtain the adversary's input. Then each label is assigned a value equal to the first value in the sequence before the label. Then the sequence is sanitized to handle duplicate items as in [1].

The following observations follows from [1]. When the computation reaches a leaf, the adversary provides a single value to the comparison. For the rightmost leaf, the value is the largest value among all nodes. For the other nodes, the value is the same value on the lowermost internal node of the path from the root to the leaf, for which the comparison result returned true. Each item in the input of Alice appears exactly once in an internal node and exactly once in a leaf node. Finally, the

values for the labels have so been chosen so that if for the actual (some hidden) inputs of the adversary the median is revealed in a particular leaf node, the median calculated by the input constructed by simulator will have the property that the leaf node corresponding to the median on the simulator's reconstructed input and the actual node corresponding to the adversary's input will be part of the same subtree of all dummy nodes. Thus, any traversal to either of these nodes from the root will result on the same node on which the adversary aborted.

Proof sketch. When S receives the output from the trusted party, it simulates the route that the execution takes in the tree, and performs any additional operation that Alice might apply to its view of the protocol. There are two cases depending on the median: Either the median is the value revealed in one of the comparisons in a leaf where the adversary aborted in some ancestor of that node, or it is the value revealed in a leaf that is not a dummy node. In the latter case, the simulation traverses the route the execution would take using the median and reach the leaf node. In the former case, the simulation will result in a traversal to the leaf node corresponding to the input computed by the simulator. However, by construction, the actual median lies in the same subtree as this leaf node and the simulator will abort at the same place as it would have with the actual inputs of the adversary. Hence the simulation proceeds identical to real experiment.

6.1 On achieving malicious security in our general framework

Recall that in our general iterative framework the outputs are gradually released and this can allow a malicious adversary to alter its input to an intermediate iteration of the protocol based on the results from previous iterations. As in the covert security protocol, this can be circumvented by requiring the adversary to commit to their input before the protocol starts and the simulation extract these inputs at the beginning to be fed to the ideal functionality. However, this is not enough, as an adversary can selectively abort or selectively use his committed inputs in the iterative protocol based on the intermediate results of the computation. One approach would be to simply require each party to prove in zero-knowledge that it used the correct input. While this can be made communication efficient (by relying on universal arguments or SNARKs) it will blow up the computational complexity. This might be efficient if a short witness can establish correctness (analogous to our consistency checks for covert security) but seems unlikely for the applications discussed in this work.

For the specific case of the median, as explained above, we are able to obtain malicious security without significant blow up in the communication or computation complexity. Towards obtaining efficient simulation of malicious adversaries, we next try to identify what property of the median protocol enables such efficient simulation.

Towards this, define $trace_i$ in our general framework to be the trace of the outputs in the first i iterations, namely, $c_1, \ldots, c_i$ which are the outputs of $\mathbf{LT}_f$ in each iteration. Let Alice's input be $A = \{a_1, \ldots, a_n\}$. For any particular data element a_j held by Alice (analogously for Bob) define $T(a_j)$ to be the set containing all traces $trace_i$ such that there is some input set B for Bob such that a_j is Alice's input in the $i+1$'st iteration when Alice and Bob interact with inputs A and B and $trace_i$ is the trace for the first i iterations.

When we consider the secure median protocol the first property we observe is that $T(a_j)$ is exactly 1. To see this consider two traces $trace_i = (c_1, \ldots, c_i)$ and $\widetilde{trace}_i = \hat{c}_1, \ldots, \hat{c}_i$ such that the maximum prefix length that they share is k, namely, for $t = 1, \ldots, k$, $c_t = \hat{c}_t$ and $c_{k+1} \neq \hat{c}_{k+1}$. Since the outputs of each iteration are results of comparisons, let us assume without loss of generality that $c_{k+1} = 0$ (meaning Alice's input was less than Bob's input in that iteration) and $\hat{c}_{k+1} = 1$. Since they share the same prefix until iteration k, the input fed by Alice in the $k+1^{st}$ iteration in both the traces must be identical. Call this element a_{j_1}. Let a_{j_2} and a_{j_3} be the input used by Alice in iteration $i+1$ at the end of traces $trace_i$ and $\widetilde{trace}_i$. It follows from the AMP protocol that $a_{j_2} < a_{j_1} < a_{j_3}$. This is because at iteration $k+1$ the result of the comparisons either prunes the data of elements less than a_{j_1} or greater than a_{j_1}. Hence, each input a_j cannot be in two different traces. A second property of the median protocol is that given an adversary every possible trace can be simulated in polynomial time. This is because the number of iterations is $O(\log n)$ and in each iteration there are at most two outcomes. We next give a rough intuition as to why these two properties are necessary.

If the first property fails to hold, then for a particular input element there are two different traces. This means the adversary can decide to selectively abort in one of the traces and such an adversary cannot be simulated without knowing the honest parties input. If the second property fails to hold, the adversary can selectively use its data. Since the all traces cannot enumerated in polynomial time, the adversary's input needs to be extracted by other means. If we relied on extractable commitments as in the case of covert security, the simulator will not know what to feed to the ideal functionality as some of the committed inputs may never be

used by the adversary. Another example besides the median that satisfies this would be the bisection method to find roots of a polynomial where Alice and Bob hold parts of a polynomial and wish to find a root in a prescribed interval. In future work, we plan to explore more examples that admit malicious security in our framework.

References

1. Gagan Aggarwal, Nina Mishra, and Benny Pinkas. Secure computation of the median (and other elements of specified ranks). *J. Cryptology*, 23(3):373–401, 2010.
2. Yonatan Aumann and Yehuda Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. *Journal of Cryptology*, 23:281–343, April 2010.
3. Boaz Barak and Oded Goldreich. Universal arguments and their applications. In *IEEE Conference on Computational Complexity*, pages 194–203, 2002.
4. Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: A System for Secure Multi-party Computation. In *ACM Conference on Computer and Communications Security*, 2008.
5. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 326–349, 2012.
6. Justin Brickell and Vitaly Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In *ASIACRYPT'05*, 2005.
7. Kai-Min Chung, Zhenming Liu, and Rafael Pass. Statistically-secure oram with $\tilde{O}(\log^2 n)$ overhead. *arXiv preprint arXiv:1307.3699*, 2013.
8. Giovanni Di Crescenzo and Helger Lipmaa. Succinct NP proofs from an extractability assumption. In *Logic and Theory of Algorithms, 4th Conference on Computability in Europe, CiE 2008, Athens, Greece, June 15-20, 2008, Proceedings*, pages 175–185, 2008.
9. S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure two-party computation in sublinear (amortized) time. In *CCS*, pages 513–524, 2012.
10. Chih hao Shen and Abhi Shelat. Fast two-party secure computation with minimal assumptions. In *ACM CCS 2013*, 2012.
11. Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster Secure Two-Party Computation Using Garbled Circuits. In *USENIX Security Symposium*, 2011.
12. Marcel Keller and Peter Scholl. Efficient, oblivious data structures for mpc. Cryptology ePrint Archive, Report 2014/137, 2014. http://eprint.iacr.org/.
13. Dexter C. Kozen. *Design and Analysis of Algorithms*. Texts and Monographs in Computer Science. Springer, 1992.
14. Benjamin Kreuter, Benjamin Mood, Abhi Shelat, and Kevin Butler. PCF: A Portable Circuit Format for Scalable Two-Party Secure Computation. In *USENIX Security Symposium*, 2013.
15. Benjamin Kreuter, Abhi Shelat, and Chih hao Shen. Billion-Gate Secure Computation with Malicious Adversaries. In *USENIX Security Symposium*, 2012.

16. Chang Liu, Yan Huang, Elaine Shi, Jonathan Katz, and Michael Hicks. Automating efficient ram-model secure computation. *IEEE S & P*, 2014.
17. Philip MacKenzie, Alina Oprea, and Michael Reiter. Automatic Generation of Two-party Computations. In *ACM Conference on Computer and Communications Security*, 2003.
18. Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay: A secure two-party computation system. In *USENIX Security*, 2004.
19. Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.
20. Rafail Ostrovsky and Victor Shoup. Private information storage. In *STOC'97*, pages 294–303, 1997.
21. Aseem Rastogi, Matthew A. Hammer, and Michael Hicks. Wysteria: A programming language for generic, mixed-mode multiparty computations. *IEEE S & P*, 2014.
22. Elaine Shi, T.-H. Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious RAM with $O((\log N)^3)$ worst-case cost. In *ASIACRYPT*, 2011.
23. Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious ram protocol. In *In CCS*, 2013.
24. Xiao Shaun Wang, Yan Huang, T-H. Hubert Chan, Abhi Shelat, and Elaine Shi. Scoram: Oblivious ram for secure computation. In *CCS'14*, 2014.
25. Andrew Chi-Chih Yao. How to generate and exchange secrets. In *FOCS*, 1986.