Efficiency Improvements for Two-party Secure Computation

Vladimir Kolesnikov[1] and Ranjit Kumaresan[2*]

[1] Bell Labs Research, Alcatel-Lucent
600 Mountain Avenue, Murray Hill, NJ 07974, USA
[2] University of Maryland,
College Park, MD 20740, USA

Abstract. We optimize the communication (and, indirectly, computation) complexity of two-party secure function evaluation (SFE). We propose a new approach, which relies on the *information-theoretic* (IT) Garbled Circuit (GC), which is more efficient than Yao's GC on shallow circuits. When evaluating a large circuit, we "slice" it into thin layers and evaluate them with IT GC. Motivated by the client-server setting, we propose two variants of our construction: one for semi-honest model (relatively straightforward), and one secure against a semi-honest server and covert client (more technically involved). One of our new building blocks, String-selection Oblivious Transfer (SOT), may be of independent interest.
Our approach, asymptotically, offers communication and computation improvement factor logarithmic in security parameter κ, over standard state-of-the-art GC. In practical terms, already for today's $\kappa \in \{128, 256\}$ our (unoptimized) algorithm offers approximately a factor 2 communication improvement in the semi-honest model, and is only a factor ≈ 1.5 more costly in setting with covert client.

1 Introduction

We propose efficiency improvements of two-party Secure Function Evaluation (SFE). SFE allows two parties to evaluate any function on their respective inputs x and y, while maintaining privacy of both x and y. SFE is justifiably a subject of an immense amount of research. Efficient SFE algorithms enable a variety of electronic transactions, previously impossible due to mutual mistrust of participants. Examples include auctions, contract signing, set intersection, etc. As computation and communication resources have increased, SFE of many useful functions has become practical for common use.

Still, SFE of most of today's functions of interest is either completely out of reach of practicality, or carries costs sufficient to deter would-be adopters, who instead choose stronger trust models, entice users to give up their privacy with incentives, or use similar crypto-workarounds. We believe that truly practical efficiency is required for SFE to see use in real-life applications.

[*] Work partly done while the author was visiting Bell Labs.

On the cost of SFE rounds. This work is mainly motivated by the client-server setting, and its specific scalability and performance issues. We argue that in this setting, the number of communication rounds in SFE often plays an insignificant role in practice.

Of course, additional rounds may cause somewhat increased *latency* of an individual computation – a possible inconvenience to the user of interactive applications. However, many SFE protocols allow for a significant precomputation and also for streaming, where message transmission may begin (and even a response may be received) before the sender completes the computation of the message. Thus, even in the peer-to-peer setting round-related latency need not be a wasted time. And this is certainly true in the client-server environment, where the idle server can always busy itself with the next client.

Further, in the server environment, where computation and communication resources are provisioned as close to demand as possible, we are particularly interested in *throughput* (rather than latency) of the total computation. This is because of the high cost of SFE, and the possibility of its adoption (vs the use of a crypto-workaround) only at an engineer- and business-acceptable price point, which includes hardware costs and electricity consumption. (Today's U.S. data centers already consume approximately 2% of U.S. electricity [13]. What would happen if they all ran SFE?)

1.1 Our Setting

As justified above, we aim to optimize computation and, mainly, communication of two-party SFE, without particular worry about round complexity. Further, our main algorithm is in an asymmetric setting: one player (presumably, server) is semi-honest, while the other (presumably, client), is covert [2]. This is in line with our goal of achieving maximal performance, while providing appropriate security guarantees. We argue that it is reasonable that a server (a business) would not deviate from a prescribed protocol for fear of lawsuits and bad publicity, and the client – for fear of being caught with probability guaranteed by the covert-secure protocol. While we also give a simpler protocol in the semi-honest model, the hybrid protocol (semi-honest client and covert server) is our main focus.

Finally, we remark that we are interested in the scalable client-server setting, since we believe it to be the setting most likely to pioneer practical use of SFE.

For simplicity, we present our protocols in the Random Oracle (RO) model. We note that we can reduce this assumption to correlation-robust hash functions, needed for OT extension [7], by encrypting more carefully in our SOT protocol.

1.2 Our Contributions, Outline of the Work, and Results

We optimize computation and communication complexity of two-party SFE for the important practical settings of semi-honest server and semi-honest or covert client. Our Garbled-Circuit (GC)-like protocol, built on consecutive secure evaluation of "slices" of the original circuit, takes advantage of the high efficiency of underlying information-theoretic (IT) GC-variant of Kolesnikov [11].

The main technical challenge with this approach is efficient wire key translation between the circuit slices, secure against covert GC evaluator. Natural but expensive use of committed OT [4, 10] would negate IT GC performance gain. Instead, we introduce and efficiently implement a new OT variant of independent interest, which we call String-selection OT (SOT). Here, the receiver submits a selection string (instead of just one bit); he obtains corresponding OT output only if he submitted one of the two sender-specified (presumably secret) strings. Our second contribution is a construction of SFE slice-evaluation protocol, which also has several subtleties in the covert-client setting.

We start with presenting detailed overview of our entire solution in Section 1.4, and cover preliminaries in Section 2. In Section 3.1, we define SOT and build SFE protocol secure against covert client $\mathcal{C}$, assuming a SOT protocol in the same model. Next, in Section 3.2 we present an efficient SOT protocol, based on OT extension [7, 6]. Our SOT protocol is actually secure against a malicious $\mathcal{C}$, since we are able to get this advantage at comparable cost to that with the covert $\mathcal{C}$. In Section 3.1 we remark on shortcuts our SFE protocol could take when both $\mathcal{S}$ and $\mathcal{C}$ are semi-honest.

Finally, in Section 4, we calculate the costs of our protocol, and compare them with that of state-of-the-art Yao GC protocols, such as [22]. We achieve asymptotic log factor improvement in security parameter κ in communication and computation in both covert and semi-honest $\mathcal{C}$ settings. In practical terms, for today's $\kappa \in \{128, 256\}$ we offer approximately a factor 2 communication improvement in the semi-honest model, and are a factor ≈ 1.5 more costly in setting with covert client. We note that our protocols can be further optimized, resulting in even better concrete performance, while GC protocols we are comparing with have been highly fine-tuned for performance.

1.3 Related Work

We survey efficient SFE state-of-the-art, and discuss how it relates to our work.

Most relevant to us is a comparatively small body of work, which provides improvements to the SFE core techniques, which address the semi-honest model. We mention, but do not discuss in detail here the works that specifically concentrate on the malicious setting, such as [14, 8, 21, 16, 23, 20]. This is because their improvement techniques cannot be transferred into the semi-honest world, and, further, malicious-secure protocols are *much* more costly than the protocols we are considering.

After Yao's original GC work [25], we are aware of few improvements to the core algorithm. Naor et al. [19] mentioned that it was possible to reduce the number of entries (each of size security parameter κ) in the GC garbled table to 3 from 4. Kolesnikov [11] introduced the GESS construction, which can be viewed as information-theoretic (IT) GC, and is much more efficient than standard GC on shallow circuits. Using a GESS block in GC, Kolesnikov and Schneider [12] showed how to get XOR gates "for free" in GC. Finally, Pinkas et al. [22] showed how to reduce the garbled table size to 3 entries, while preserving the free-XOR compatibility, or to two entries, but disallowing free-XOR technique.

GMW [5], a non-constant-round SFE protocol like ours, is very communication-efficient, if given cheap oblivious transfer (OT) (e.g., of cost independent of security parameter). However, using today's best two-party OT [7], GMW's communication cost, 4κ per gate, is slightly worse than GC. Further, GMW is not secure against covert client.

In this work, we are building on [11], and demonstrate communication cost improvement over [22], as a trade-off with round complexity. Our costs are correspondingly better than that of GMW, and we need fewer rounds than GMW.

Finally, we note the theoretical work of Naor and Nissim [17], which uses indexing to perform SFE of branching programs (BP), and achieves costs polynomial in the communication complexity of the *insecurely* computed function. We note that, firstly, [17] is not performance-optimized. However, more importantly, BP function representation often carries dramatic overhead (exponential for integer multiplication), as compared to circuits.

1.4 Overview of our solution

As discussed and justified above, our main goal is communication and computational complexity reduction of the solution. We allow ourselves additional communication rounds.

Our main idea is to build on the information-theoretic version of GC of Kolesnikov [11], which, due to its avoidance of encryption, is *more efficient* than computational GC for small-depth circuits. We capitalize on this by "slicing" our original circuit C into a sequence of shallow circuits C_1,...C_n, which we then evaluate and obtain corresponding efficiency improvement. There are several technical problems that need to be solved.

First, recall that Kolesnikov's scheme, GESS, does not require generation or sending of garbled tables. It does use wire keys, which may start with 1-bit-long strings for output wires, and grow in size approximately quadratically with the depth d of the fan-out-one circuit. For generic fan-out-2 circuits, thus, total wire keys at depth d is up to $O(2^d d^2)$.

The first problem is allowing for piece-wise secure circuit evaluation, given the circuit's slicing. In the semi-honest model, this can be achieved as follows. Consider any slicing of C, where some wire w_j of C is an output wire of C_i, and is an input wire of C_{i+1}. Now, when a slice C_i is evaluated, C_i's 1-bit wire key for w_j is computed by the evaluator, and then used, via OT, to obtain the wire key for the corresponding input wire of C_{i+1}. This process repeats until C's output wire keys are computed by the evaluator. In order to prevent the evaluator from learning the intermediate wire values of C, the 1-bit wire keys of slices' output wires are randomly assigned to wire values.

While secure against passive adversaries, above construction is easily compromised by an active evaluator. Indeed, he can influence the output of the computation simply by flipping the 1-bit key of w_j before using it in OT, which will result in flipping the underlying bit on that wire.

To efficiently resolve this problem and achieve covert security against the evaluator, we introduce String-selection OT (SOT), a variant of 1-out-of-2 OT, where the receiver submits a selection string (instead of a selection bit). Naturally, in SOT, receiver obtains OT output corresponding to his selection string; submission of a string not expected by the sender $\mathcal{S}$ results in error output. In our construction, we will use multi-bit wire keys on output wires of each slice; Client $\mathcal{C}$ will submit them to SOT to obtain input wire keys for next slice. Now, a malicious $\mathcal{C}$ wishing to flip a wire value must guess, in the on-line fashion, the multi-bit wire key corresponding to the opposite wire value. We will show that this results in covert security against the evaluator.

Efficient OT is a critical component in our construction. Another technical contribution of this paper is an efficient SOT protocol for arbitrary-length selection strings, secure against malicious $\mathcal{C}$.

Finally, we discuss how to optimally slice a given circuit, evaluate and compare efficiency of our approach to previous solutions.

2 Preliminaries and Notation

2.1 Garbled Circuits (GC)

Yao's Garbled Circuit approach [25], excellently presented in [15], is the most efficient method for one-round secure evaluation of a boolean circuit C. We summarize its ideas in the following. The circuit *constructor* (server $\mathcal{S}$) creates a *garbled circuit* $\widetilde{C}$: for each wire w_i of the circuit, he randomly chooses two garblings $\widetilde{w}_i^0, \widetilde{w}_i^1$, where $\widetilde{w}_i^j$ is the *garbled value* of w_i's value j. (Note: $\widetilde{w}_i^j$ does not reveal j.) Further, for each gate G_i, $\mathcal{S}$ creates a *garbled table* $\widetilde{T}_i$ with the following property: given a set of garbled values of G_i's inputs, $\widetilde{T}_i$ allows to recover the garbled value of the corresponding G_i's output, but nothing else. $\mathcal{S}$ sends these garbled tables, called *garbled circuit* $\widetilde{C}$ to the *evaluator* (client $\mathcal{C}$). Additionally, $\mathcal{C}$ obliviously obtains the *garbled inputs* $\widetilde{w}_i$ corresponding to inputs of both parties: the garbled inputs $\widetilde{x}$ corresponding to the inputs x of $\mathcal{S}$ are sent directly and $\widetilde{y}$ are obtained with a parallel 1-out-of-2 oblivious transfer (OT) protocol [18, 1]. Now, $\mathcal{C}$ can evaluate the garbled circuit $\widetilde{C}$ on the garbled inputs to obtain the *garbled outputs* by evaluating $\widetilde{C}$ gate by gate, using the garbled tables $\widetilde{T}_i$. Finally, $\mathcal{C}$ determines the plain values corresponding to the obtained garbled output values using an output translation table received by $\mathcal{S}$. Correctness of GC follows from the way garbled tables $\widetilde{T}_i$ are constructed.

2.2 GESS: Efficient Information-Theoretic GC for Shallow Circuits

We review the Gate Evaluation Secret Sharing (GESS) scheme of Kolesnikov [11], which is the most efficient information-theoretic analog of GC. Because encryption there is done with bitwise XOR and bit shufflings, rather than with standard primitives such as AES, GESS is significantly more efficient than standard GC, both in computation and communication, for shallow circuits.

At a high level, GESS is a secret sharing scheme, designed to match with the gate function g, as follows. The output wire keys are the secrets, from which the *constructor* produces four secret shares, one for each of the wire keys of the two input wires. GESS guarantees that a combination of shares, corresponding to any of the four possible gate inputs, reconstructs the corresponding key of the output wire. This secret sharing can be applied recursively, enabling reduction of SFE to OT (to transfer the wire secrets on the input wires). One result of [11] is the SFE protocol for a boolean formula F of communication complexity $\approx \sum d_i^2$, where d_i is the depth of the i-th leaf of F. This improvement (prior best construction – [9] combined with [3] – cost $\approx \sum 2^{\theta(\sqrt{d_i})}$) will allow us to outperform the standard GC by evaluating thin slices of the circuit. We note that other IT GC variants could also be used in our work, with corresponding performance disadvantage.

2.3 Covert Security

In this work, we consider a semi-honest Server, and a stronger Client-adversary who may deviate from the protocol specification in an attempt to cheat. While cheating attempts may be successful, the covert model [2] guarantees that any attempted cheating is caught with a certain minimal probability ϵ.

Aumann and Lindell [2] give three formalizations of this notion; we consider their strongest definition, the strong explicit-cheat formulation. Informally, this variant of the covert-security definition guarantees that, if caught, the adversary does not learn honest player's input. If not caught, the adversary may succeed either in learning the honest player's input or influencing the computation (or both). The definition is given in the standard ideal-real paradigm. Intuitively, the difference with the malicious model is that covert ideal world allows the `cheat` request: if successful (based on the coin flip by a trusted party), the adversary is allowed to win, if not, honest players are informed of cheat attempt.

We refer the reader to [2] for details and formal definitions.

2.4 Notation

Let κ be the computational security parameter.

Our SFE protocol is given a circuit C which represents a function f that a server $\mathcal{S}$ (with input x) and a client $\mathcal{C}$ (with input y) wish to compute. Let d denote the depth of C. Our protocol proceeds by dividing the circuit C into horizontal slices. Let ℓ denote the number of such slices, and let $C_1, \ldots, C_\ell$ denote these ℓ slices of C. We let d' denote the depth of each slice C_i.

In circuit slice C_i, we let $u_{i,j}$ (resp. $v_{i,j}$) denote the jth input (resp. output) wire. For a wire $u_{i,j}$ (resp. $v_{i,j}$), we refer to the garbled values corresponding to 0 and 1 by $\widetilde{u}_{i,j}^0, \widetilde{u}_{i,j}^1$ (resp. $\widetilde{v}_{i,j}^0, \widetilde{v}_{i,j}^1$) respectively. In our protocol, let k (resp. k') denote the length of input (resp. output) wire garblings (k' will be related to the covert deterrent factor as $\epsilon = \frac{1}{2^{k'}-1}$). While evaluating the garbled circuit, $\mathcal{C}$ will possess only one of two garbled values for each wire in the circuit. We let $\widetilde{u}'_{i,j}$ (resp. $\widetilde{v}'_{i,j}$) denote the garbled value on wire $u_{i,j}$ (resp. $v_{i,j}$) that is possessed by $\mathcal{C}$.

In Section 3, we introduce the primitive $\mathrm{SOT}_{k,k'}$ which requires a receiver $\mathcal{R}$, with input a k'-bit selection string, to select one of two k-bit strings held by sender $\mathcal{S}$. Our protocol for $\mathrm{SOT}_{k,k'}$ uses calls to the standard 1-out-of-2 OT primitive (where receiver $\mathcal{R}$, with input a selection bit, selects one of two k-bit strings held by sender $\mathcal{S}$).

3 Our Protocol for Secure Two-Party Computation

We describe our protocol for secure two-party computation against a semi-honest server and a covert receiver in a hybrid model with ideal access to String-selection OT (SOT), defined below:

Definition 1. String-selection OT, $\mathrm{SOT}_{k,k'}$, *is the following functionality:*

Inputs: *$\mathcal{S}$ holds two pairs $(x_0, r_0), (x_1, r_1)$, where each x_0, x_1 are k-bit strings, and r_0, r_1 are k'-bit strings (with $r_0 \neq r_1$). $\mathcal{R}$ holds k'-bit selection string r.*

Outputs: *If $r = r_i$ for some $i \in \{0, 1\}$, then $\mathcal{R}$ outputs x_i, and $\mathcal{S}$ outputs empty string λ. Otherwise, $\mathcal{R}$ and $\mathcal{S}$ both output error symbol $\perp$.*

3.1 Our Protocol

We now present our protocol for securely computing a function f, represented by a circuit C, where semi-honest $\mathcal{S}$ has input x and covert $\mathcal{C}$ has input y.

Our protocol uses OT, the standard 1-out-of-2 OT protocol, and $\mathrm{SOT}_{k,k'}$, a SOT protocol, as defined in Definition 1. Assume both OT and $\mathrm{SOT}_{k,k'}$ are secure against a semi-honest sender and a covert receiver with deterrent ϵ (the required value of ϵ will depend on the parameters of the SFE protocol and is stated in the security theorems below). We prove security in the strongest covert formulation of [2], the strong explicit cheat formulation.

We evaluate C slice-by-slice. Further, each slice is viewed as a fan-out-1 circuit, as needed for the GESS scheme. We will discuss performance-optimal ways of generating the slices in Appendix C.

Protocol 1 *1.* Circuit Preparation:

(a) Slicing. *Given d, d', server $\mathcal{S}$ divides circuit C of depth d into horizontal sub-circuit layers, or* slices, *$C_1, \ldots, C_\ell$ of depth d'.*

(b) Preparing each slice. *In this step, $\mathcal{S}$ randomly generates the output secrets of the slice, and, applying the* GESS *sharing scheme, obtains corresponding input secrets, as follows.*
Denote by $u_{i,j}$ the input wires and by $v_{i,j}$ the output wires of the slice C_i. For each wire $v_{i,j}$, $\mathcal{S}$ picks two random garblings $\tilde{v}^0_{i,j}, \tilde{v}^1_{i,j}$ of length $k' > 1$ (conditioned on $\tilde{v}^0_{i,j} \neq \tilde{v}^1_{i,j}$). $\mathcal{S}$ then (information-theoretically) computes the GESS *garblings for each input wire in the subcircuit, as described in [11]. Let k be the maximal length of the garblings $\tilde{u}^0_{i,j}, \tilde{u}^1_{i,j}$ of the input wires $u_{i,j}$. Recall, GESS does not use garbled tables.*

2. Circuit Evaluation:
For $1 \leq i \leq \ell$, in round i do:

(a) Oblivious transfer of keys.
   i. *For the top slice (the sub-circuit C_1), do the standard SFE garblings transfer:*
   For each client input wire $u_{1,j}$ (representing the bits of $\mathcal{C}$'s input y), $\mathcal{S}$ and $\mathcal{C}$ execute a 1*-out-of-*2 *OT protocol, where $\mathcal{S}$ plays the role of a sender, with inputs $\tilde{u}^0_{1,j}, \tilde{u}^1_{1,j}$, and $\mathcal{C}$ plays the role of the receiver, directly using his inputs for OT.*
   For each server input wire $u_{1,j}$ (representing the bits of $\mathcal{S}$'s input x), $\mathcal{S}$ sends to $\mathcal{C}$ one of $\tilde{u}^0_{1,j}, \tilde{u}^1_{1,j}$, corresponding to its input bits.
   ii. *For slices $C_i, i \neq 1$, transfer next slice's input keys based on the current output keys:*
   For each input wire $u_{i,j}$ of the slice C_i, $\mathcal{S}$ uniformly at random chooses a string $r_{i,j}$ of length k (this will mask the transferred secrets[3]). $\mathcal{S}$ then acts as sender in $\mathrm{SOT}_{k,k'}$ with input $((\tilde{u}^0_{i,j} \oplus r_{i,j}, \tilde{v}^0_{i-1,j})$, $(\tilde{u}^1_{i,j} \oplus r_{i,j}, \tilde{v}^1_{i-1,j}))$, and $\mathcal{C}$ acts as receiver with input $\tilde{v}'_{i-1,j}$ (this is the output wire secret of the one-above slice C_{i-1}, which $\mathcal{C}$ obtains by executing Step 2b described next). Let $\mathcal{C}$ obtain $\tilde{u}'_{i,j} \oplus r_{i,j}$ as the output from $\mathrm{SOT}_{k,k'}$.
   Once all $\mathrm{SOT}_{k,k'}$ had completed, $\mathcal{S}$ sends all $r_{i,j}$ to $\mathcal{C}$, who then computes all $\tilde{u}'_{i,j}$.

(b) Evaluating the slice. *$\mathcal{C}$ evaluates the* GESS *sub-circuit C_i, using garbled input values $\tilde{u}'_{i,j}$ to obtain the output values $\tilde{v}'_{i,j}$.*

3. Output of the computation. *Recall, w.l.o.g., only $\mathcal{C}$ receives the output. $\mathcal{S}$ now sends the output translation tables to $\mathcal{C}$. For each output wire of C, $\mathcal{C}$ outputs the bit corresponding to the wire secret obtained in evaluation of the last slice C_ℓ.*

Observation 1 *We note that technically the reason for sending masked wire values via $\mathrm{SOT}_{k,k'}$ in Step 2(a)ii is to facilitate the simulation proof, as follows. When simulating covert $\mathcal{C}^*$ without the knowledge of the input, the simulator $\mathsf{Sim_C}$'s messages to $\mathcal{C}^*$ "commit" $\mathsf{Sim_C}$ to certain randomized representation of players' inputs. When, in SOT of the i-th slice C_i, a covert $\mathcal{C}^*$ successfully cheats, he will be given* both *wire keys for some wire of C_i. Without the masking, this knowledge, combined with the knowledge of the gate function and a key on a sibling wire, allows $\mathcal{C}^*$ to infer the wire value encrypted by the simulation, which might differ from the expected value. We use the mask to hide the encrypted value even when both wire keys are revealed. Mask can be selected to "decommit" transcript seen by $\mathcal{C}^*$ to either of wire values.*

Observation 2 *We note that in the semi-honest-$\mathcal{C}$ case the above protocol can be simplified and made more efficient. In particular, k' is set to* 1*, in Step 2(a)ii, it is sufficient to use OT (vs SOT), and offset strings r_j are not needed.*

We prove security of Protocol 1 against a semi-honest server $\mathcal{S}$ and a *covert* client $\mathcal{C}$.

[3] The reason for the masking is to enable simulation of $\mathcal{C}$. See Observation 1 for details.

Theorem 3. *Let OT be a* 1*-out-of-*2 *OT, and* $\mathrm{SOT}_{k,k'}$ *be a string-selection OT (cf. Definition 1), both secure against semi-honest sender. Then Protocol 1 is secure against a semi-honest server* $\mathcal{S}$*.*

Further, let k' *be a parameter upper-bounded by* $\mathrm{poly}(n)$*, and set* $\epsilon = \frac{1}{2^{k'}-1}$*. Let* f *be any probabilistic polynomial-time function. Assume that the underlying OT and SOT protocols are secure in the presence of covert receiver with* ϵ*-deterrent, in the strong explicit cheat formulation of covert security. Then, Protocol 1 securely computes* f *in the presence of covert client* $\mathcal{C}$ *with* ϵ*-deterrent, in the strong explicit cheat formulation.*

The proof is presented in Appendix A.

3.2 A Protocol for $\mathbf{SOT}_{k,k'}$

In this section, we introduce an efficient string-selection OT protocol, secure against semi-honest sender and malicious receiver, which works for selection strings of arbitrary length k'. We build it from k' standard 1-out-of-2 OTs. We note that, while Protocol 1 requires only covert-$\mathcal{C}$ security, we provide a stronger building block (at no or very little extra cost).

The intuition behind our construction is as follows. We will split each of the two secrets x_0, x_1 into k' random shares x_i^j (i.e. $2k'$ total shares), with the restriction that the sets of x_i^j, indexed by each of the two selection strings, reconstructs the corresponding secret. Then, in the semi-honest model, performing k' standard OTs allows receiver to reconstruct one of the two secrets, corresponding to his selection string.

To protect against malicious or covert receiver, we, firstly, assume that underlying OTs are secure against such receiver. Further, we allow the sender to confirm that the receiver indeed received one of the two secrets, as follows. Denote by h_0 (resp. h_1) the hash of the vector of secret shares corresponding to the secret x_0 (resp. x_1). To confirm that receiver obtained shares to reconstruct at least one of x_0, x_1, the sender will send $h_0 \oplus h_1$ to $\mathcal{R}$, and expect to receive both h_0 and h_1 back (actually, it is sufficient to receive just one of h_0, h_1 selected by R at random).

The above check introduces a subtle vulnerability: the value $h_0 \oplus h_1$ leaks information if selection strings differ in a single position. Indeed, then there is only one secret share unknown to malicious $\mathcal{R}$, and secrets' values can be verified by checking against received $h_0 \oplus h_1$. (If we restricted the selection strings to differ in at least two positions, this approach can be made to work. However, such restriction is less natural.) As a result, we now can not transfer the $\mathrm{SOT}_{k,k'}$ secrets directly. To address this, we transfer $\mathrm{SOT}_{k,k'}$ secrets by encrypting each with a random key, and then OT-transferring one of the two keys as above, which is secure for OT of long random secrets.

Our Protocol. Let sender $\mathcal{S}$ have input $(x_0, r_0), (x_1, r_1)$ with $|x_0| = |x_1| = k$, and $|r_0| = |r_1| = k'$. Let receiver $\mathcal{R}$ have input $r \in \{r_0, r_1\}$. Let κ be a security parameter, OT be a standard 1-out-of-2 OT, and $H : \{0,1\}^* \to \{0,1\}^\kappa$ be a random oracle.

Protocol 2 STRING-SELECTION OT

1. *Let* $r_0 = r_{01}, r_{02}, ..., r_{0k'}$, *and* $r_1 = r_{11}, r_{12}, ..., r_{1k'}$, *where* r_{ij} *are bits.* $\mathcal{S}$ *chooses* $s_i^j \in_R \{0,1\}^\kappa$, *for* $i \in \{0,1\}, j \in \{1, \ldots, k'\}$. $\mathcal{S}$ *sets keys* $s_0 = \bigoplus_j s_j^{r_{0j}}$, *and* $s_1 = \bigoplus_j s_j^{r_{1j}}$.
2. $\mathcal{S}$ *and* $\mathcal{R}$ *participate in* k' *OTs as follows. For* $j = 1$ *to* k'*:*
   (a) $\mathcal{S}$ *with input* (s_j^0, s_j^1), *and* $\mathcal{R}$ *with input* r_j *send their inputs to OT.* $\mathcal{R}$ *receives* $s_j^{r_j}$.
3. $\mathcal{S}$ *computes hashes of shares corresponding to the two secrets/selection strings:* $h_0 = H(s_1^{r_{01}}, s_2^{r_{02}}, ..., s_{k'}^{r_{0k'}})$, *and* $h_1 = H(s_1^{r_{11}}, s_2^{r_{12}}, ..., s_{k'}^{r_{1k'}})$. $\mathcal{S}$ *then sends* $h = h_0 \oplus h_1$ *to* $\mathcal{R}$.
4. $\mathcal{R}$ *computes* $h_\mathcal{R} = H(s_{r_1}, ..., s_{r_{k'}})$ *and sends to* $\mathcal{S}$ *either* $h_\mathcal{R}$ *or* $h \oplus h_\mathcal{R}$, *equiprobably.*
5. $\mathcal{S}$ *checks that the hash received from* $\mathcal{R}$ *is equal to either* h_0 *or* h_1. *If so, it sends, in random order,* $H(s_0) \oplus x_0, H(s_1) \oplus x_1$ *and terminates with output* λ*; if not,* $\mathcal{S}$ *sends* $\perp$ *to* $\mathcal{R}$ *and outputs failure symbol* $\perp$.
6. $\mathcal{R}$ *computes key* $s_r = \bigoplus_j \ s_{r_j}$ *and recovers the secret by canceling out* $H(s_r)$.

For readability, we omitted simple technical details, such as adding redundancy to x_i which allows $\mathcal{R}$ to identify the recovered secret. We also slightly abuse notation and consider the output length of H sufficiently stretched when we use it to mask secrets x_i.

Theorem 4. *Assume that the underlying OT is secure in the presence of semi-honest sender and malicious receiver. When* $k' > 1$*, Protocol 2 is a secure SOT protocol in the presence of semi-honest sender and malicious receiver.*

The proof is presented in Appendix B.

4 Performance Analysis

For the lack of space, we present all the calculations in Appendix C. Here we summarize the results.

Consider a fan-out 2 circuit C with c gates. To simplify our cost calculation and w.l.o.g., we assume C is a rectangular circuit of constant width, where each gate has fan-out 2. Let C be divided into ℓ slices, each of depth d'. Let k (resp. k') be key length on input (resp. output) wires of each slice. k' is effectively the covert deterrent parameter, and k grows as $O(2^{d'})$ due to fan-out 2.

Then communication cost (measured in bits) of GC [22] is $\mathsf{cost}(\mathsf{Yao}) = 2\kappa c$, and of GMW is $\mathsf{cost}(\mathsf{GMW}) = 4c + 4\kappa c$. Our costs are $O(\kappa c / \log \kappa)$ in the semi-honest model, and $O(k'\kappa c / \log(\kappa/k'))$ in the covert-client setting, with small constants. The computation costs (the number of hash function evaluations) of all protocols, including ours is proportional to the communication cost; hence our asymptotic improvements translate into computation as well.

In concrete terms, we get up to factor 2 improvement in the semi-honest model for today's typical parameters. Setting $d' = 3$ and $\kappa = 256$, our cost is

$\approx 208c$, as compared to $\approx 512c$ of [22]. In the setting with covert client ($k' = 2$, deterrent 1/3), our protocol has cost $\approx 804c$, at less than factor 2 disadvantage; it surpasses [22] for $\kappa \approx 4800$.

Finally, while the GC protocol has been highly fine-tuned for performance, we note that our protocols have room for optimization, resulting in even better concrete advantage.

References

1. Aiello, W., Ishai, Y., and Reingold, O. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology – EUROCRYPT 2001* (May 2001), B. Pfitzmann, Ed., vol. 2045 of *Lecture Notes in Computer Science*, Springer, pp. 119–135.
2. Aumann, Y., and Lindell, Y. Security against covert adversaries: Efficient protocols for realistic adversaries. In *TCC 2007: 4th Theory of Cryptography Conference* (Feb. 2007), S. P. Vadhan, Ed., vol. 4392 of *Lecture Notes in Computer Science*, Springer, pp. 137–156.
3. Cleve, R. Towards optimal simulations of formulas by bounded-width programs. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing* (New York, NY, USA, 1990), STOC '90, ACM, pp. 271–277.
4. Crépeau, C. Verifiable disclosure of secrets and applications (abstract). In *Advances in Cryptology – EUROCRYPT'89* (Apr. 1990), J.-J. Quisquater and J. Vandewalle, Eds., vol. 434 of *Lecture Notes in Computer Science*, Springer, pp. 150–154.
5. Goldreich, O., Micali, S., and Wigderson, A. How to play any mental game, or a completeness theorem for protocols with honest majority. In *19th Annual ACM Symposium on Theory of Computing* (May 1987), A. Aho, Ed., ACM Press, pp. 218–229.
6. Harnik, D., Ishai, Y., Kushilevitz, E., and Nielsen, J. B. OT-combiners via secure computation. In *TCC 2008: 5th Theory of Cryptography Conference* (Mar. 2008), R. Canetti, Ed., vol. 4948 of *Lecture Notes in Computer Science*, Springer, pp. 393–411.
7. Ishai, Y., Kilian, J., Nissim, K., and Petrank, E. Extending oblivious transfers efficiently. In *Advances in Cryptology – CRYPTO 2003* (Aug. 2003), D. Boneh, Ed., vol. 2729 of *Lecture Notes in Computer Science*, Springer, pp. 145–161.
8. Jarecki, S., and Shmatikov, V. Efficient two-party secure computation on committed inputs. In *Advances in Cryptology – EUROCRYPT 2007* (May 2007), M. Naor, Ed., vol. 4515 of *Lecture Notes in Computer Science*, Springer, pp. 97–114.
9. Kilian, J. Founding cryptography on oblivious transfer. In *STOC* (1988), pp. 20–31.
10. Kiraz, M. S., and Schoenmakers, B. An efficient protocol for fair secure two-party computation. In *Topics in Cryptology – CT-RSA 2008* (Apr. 2008), T. Malkin, Ed., vol. 4964 of *Lecture Notes in Computer Science*, Springer, pp. 88–105.
11. Kolesnikov, V. Gate evaluation secret sharing and secure one-round two-party computation. In *Advances in Cryptology – ASIACRYPT 2005* (Dec. 2005), B. K. Roy, Ed., vol. 3788 of *Lecture Notes in Computer Science*, Springer, pp. 136–155.
12. Kolesnikov, V., and Schneider, T. Improved garbled circuit: Free XOR gates and applications. In *ICALP 2008: 35th International Colloquium on Automata,*

Languages and Programming, Part II (July 2008), L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, Eds., vol. 5126 of *Lecture Notes in Computer Science*, Springer, pp. 486–498.

13. Koomey, J. Growth in data center electricity use 2005 to 2010. `http://www.koomey.com/post/8323374335`. Retrieved Sept 22, 2011, July 2011.
14. Lindell, Y., and Pinkas, B. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology – EUROCRYPT 2007* (May 2007), M. Naor, Ed., vol. 4515 of *Lecture Notes in Computer Science*, Springer, pp. 52–78.
15. Lindell, Y., and Pinkas, B. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology 22*, 2 (Apr. 2009), 161–188.
16. Lindell, Y., and Pinkas, B. Secure two-party computation via cut-and-choose oblivious transfer. In *TCC 2011: 8th Theory of Cryptography Conference* (Mar. 2011), Y. Ishai, Ed., vol. 6597 of *Lecture Notes in Computer Science*, Springer, pp. 329–346.
17. Naor, M., and Nissim, K. Communication preserving protocols for secure function evaluation. In *33rd Annual ACM Symposium on Theory of Computing* (July 2001), ACM Press, pp. 590–599.
18. Naor, M., and Pinkas, B. Efficient oblivious transfer protocols. In *12th Annual ACM-SIAM Symposium on Discrete Algorithms* (Jan. 2001), ACM-SIAM, pp. 448–457.
19. Naor, M., Pinkas, B., and Sumner, R. Privacy preserving auctions and mechanism design. In *ACM Conference on Electronic Commerce* (1999), pp. 129–139.
20. Nielsen, J. B., Nordholt, P. S., Orlandi, C., and Burra, S. S. A new approach to practical active-secure two-party computation. Cryptology ePrint Archive, Report 2011/091, 2011. `http://eprint.iacr.org/`.
21. Nielsen, J. B., and Orlandi, C. LEGO for two-party secure computation. In *TCC 2009: 6th Theory of Cryptography Conference* (Mar. 2009), O. Reingold, Ed., vol. 5444 of *Lecture Notes in Computer Science*, Springer, pp. 368–386.
22. Pinkas, B., Schneider, T., Smart, N. P., and Williams, S. C. Secure two-party computation is practical. In *Advances in Cryptology – ASIACRYPT 2009* (Dec. 2009), M. Matsui, Ed., vol. 5912 of *Lecture Notes in Computer Science*, Springer, pp. 250–267.
23. Shelat, A., and Shen, C.-H. Two-output secure computation with malicious adversaries. In *Advances in Cryptology – EUROCRYPT 2011* (May 2011), K. G. Paterson, Ed., vol. 6632 of *Lecture Notes in Computer Science*, Springer, pp. 386–405.
24. Yao, A. C. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science* (Nov. 1982), IEEE Computer Society Press, pp. 160–164.
25. Yao, A. C.-C. How to generate and exchange secrets (extended abstract). In *FOCS* (1986), pp. 162–167.

A Proof of Theorem 3

Proof. **Security against semi-honest $\mathcal{S}$.** Given input x, the semi-honest $\mathcal{S}^*$ is simulated as follows. Sim_S chooses a random input y' for $\mathcal{C}$ and plays honest $\mathcal{C}$ interacting with $\mathcal{S}^*$. He then outputs whatever $\mathcal{S}^*$ outputs.

It is easy to see that $\mathcal{S}^*$ does not receive any messages – all protocol messages related to $\mathcal{C}$'s input are delivered inside OT and $\text{SOT}_{k,k'}$, the protocols that do not return output to $\mathcal{S}^*$ (other than possible error symbols, which will never be output in this simulation, since it presumes that $\mathcal{C}$ is acting honestly, and $\mathcal{S}^*$ is semi-honest). Hence, this is a perfect simulation (however, the function calls to underlying OT primitives will not be perfectly simulated).

The proof of security in the client-corruption case is somewhat more complex.

Security against covert $\mathcal{C}$. This part of the proof is somewhat more involved. We present the simulator Sim_C of a covert attacker $\mathcal{C}^*$, and argue that it produces a good simulation.

Sim_C starts $\mathcal{C}^*$ and interacts with it, sending $\mathcal{C}^*$ messages it expects to receive, and playing the role of the trusted party for the OT and $\text{SOT}_{k,k'}$ oracle calls that $\mathcal{C}^*$ may make, where $\mathcal{C}^*$ plays the role of the receiver. Unless terminating early (e.g. due to simulating abort), Sim_C will first simulate processing the top slice C_1, then all internal slices (using same procedure for each internal slice), and then the last, output slice C_ℓ.

For each output wire of each slice C_i, Sim_C samples two random strings of length k' (with the restriction that these two strings are different per wire). Based on these, Sim_C computes C_i's input wire garblings by applying the GESS algorithm.

For the top slice C_1, Sim_C plays OT trusted party, where $\mathcal{C}^*$ is the receiver, and receives OT input from $\mathcal{C}^*$.

1. If the input is abort or $\mathsf{corrupted}$, then Sim_C sends abort or $\mathsf{corrupted}$ (respectively) to the trusted party computing f, simulates $\mathcal{C}$ aborting and halts (outputting whatever $\mathcal{C}^*$ outputs).
2. If the input is cheat, then Sim_C sends cheat to the trusted party. If it receives back $\mathsf{corrupted}$, then it hands $\mathcal{C}^*$ the message $\mathsf{corrupted}$ as if it received it from the trusted party, simulates $\mathcal{C}$ aborting and halts (outputting whatever $\mathcal{C}^*$ outputs). If it receives back $\mathsf{undetected}$ (and thus $\mathcal{S}$'s input x as well), then Sim_C works as follows. First, it hands $\mathcal{C}^*$ the string $\mathsf{undetected}$ together with all the input wire keys that were part of server's input in the OT ($\mathcal{C}^*$ expects to receive OT inputs in this case). Next, Sim_C uses the input x of $\mathcal{S}$ that it received in order to perfectly emulate $\mathcal{S}$ in the rest of the execution. This is easily done, since so far Sim_C had not delivered to $\mathcal{C}^*$ any messages "from $\mathcal{S}$" that depended on $\mathcal{S}$'s input.
3. If the input is a representation of $\mathcal{C}^*$'s input to OT, then Sim_C hands $\mathcal{C}^*$ the input wire garbling keys that are "chosen" by the $\mathcal{C}^*$'s OT input, and proceeds with the simulation below.

Sim_C now also sends $\mathcal{C}^*$ the wire secrets corresponding to a random input of $\mathcal{S}$. Now, presumably, $\mathcal{C}^*$ will evaluate the slice and use the output wire secrets in $\text{SOT}_{k,k'}$ oracles.

For all internal slices $C_2, ..., C_{\ell-1}$ Sim_C plays $\text{SOT}_{k,k'}$ trusted party, where $\mathcal{C}^*$ is the receiver, and receives $\text{SOT}_{k,k'}$ input from $\mathcal{C}^*$.

1. If the input is abort or corrupted, then Sim_C sends abort or corrupted (respectively) to the trusted party computing f, simulates $\mathcal{C}$ aborting and halts (outputting whatever $\mathcal{C}^*$ outputs).
2. If the input is cheat, then Sim_C sends cheat to the trusted party.
   If it receives back corrupted, then it hands $\mathcal{C}^*$ the message corrupted as if it received it from the trusted party, simulates $\mathcal{C}$ aborting and halts (outputting whatever $\mathcal{C}^*$ outputs).
   If it receives back undetected (and thus $\mathcal{S}$'s input x as well), then Sim_C works as follows. First, it hands $\mathcal{C}^*$ the string undetected together with all the masked input wire keys $\tilde{u}^0_{i,j} \oplus r_{i,j}, \tilde{u}^1_{i,j} \oplus r_{i,j}$ and the corresponding selection-string keys $\tilde{v}^0_{i-1,j}, \tilde{v}^1_{i-1,j}$ that were part of sender's input in current $\mathrm{SOT}_{k,k'}$ ($\mathcal{C}^*$ expects to receive OT inputs in this case). These input wire keys were generated by Sim_C by running GESS wire key generation for the current slice, the selection-string keys are the output keys from preceding slice, and the wire-key to selection-string correspondence is set at random at this time (we will reconcile it when needed later).
   Next, Sim_C uses the input x of $\mathcal{S}$ that it just received and the input y of $\mathcal{C}^*$ that it received in OT in slice C_1, to perfectly emulate $\mathcal{S}$ in the rest of the execution. We note that Sim_C had already sent messages to $\mathcal{C}^*$ which should depend on $\mathcal{S}$'s input, which we now need to reconcile with the real input x received. First, we observe that the view of $\mathcal{C}^*$ of the prior slices' evaluation is consistent with any input of $\mathcal{S}$, since $\mathcal{C}^*$ is never given both secrets on any wire. ($\mathcal{C}^*$ only sees both secrets on the immediately preceding slice's output wires since Sim_C gave him both string-selection keys for current slice. However, this is allowed in the underlying GESS scheme, and hence is also consistent with any input of $\mathcal{S}$.) At the same time, both $\mathrm{SOT}_{k,k'}$ secrets are revealed to $\mathcal{C}^*$ in the current slice, and $\mathcal{C}^*$ can obtain both wire encodings for each of the slice's input wires, which, in turn, may reveal correspondence between wire values and encodings. Since $\mathcal{C}^*$'s $\mathrm{SOT}_{k,k'}$ selection string determines the "active" $\mathrm{SOT}_{k,k'}$ value, which is offset by $r_{i,j}$ into the "active" wire encoding, we now have to be careful that it is consistent with the players' input into the function. This reconciliation is easily achieved by appropriately setting the simulated offset string $r'_{i,j}$ for each wire j. The simulated $r'_{i,j}$ is selected so that the "active" (resp. "inactive") encoding corresponding to the function's inputs x and y is obtained by XORing $r'_{i,j}$ with the "active" (resp. "inactive") string-selection OT value. In other words, for each wire where Sim_C chose incorrect wire-key to selection-string correspondence, this correspondence is flipped by choosing the right offset $r'_{i,j}$. This flipping selection is computed as follows: for GESS input keys u_0, u_1, offset r (and $\mathrm{SOT}_{k,k'}$ inputs $u_0 \oplus r, u_1 \oplus r$) the correspondence is reversed by applying offset $r' = u_0 \oplus u_1 \oplus r$.
   Sim_C simulates the rest of the interaction simply by playing the honest $\mathcal{S}$.
3. If the input is a representation of $\mathcal{C}^*$'s input to $\mathrm{SOT}_{k,k'}$ then Sim_C proceeds as follows. We stress that Sim_C knows exactly the garblings of the input wires of C_{i-1} sent to $\mathcal{C}^*$ (and hence the garblings of the output wires of C_{i-1} that $\mathcal{C}^*$ should reconstruct and use as inputs to $\mathrm{SOT}_{k,k'}$ oracles for slice C_i).

(a) If $\mathcal{C}^*$ submits $\mathrm{SOT}_{k,k'}$ inputs as expected, then $\mathsf{Sim_C}$ sends $\mathcal{C}^*$ the corresponding garblings of the C_i input wires.

(b) Otherwise, we deal with the cheating attempt by $\mathcal{C}^*$. $\mathsf{Sim_C}$ then sends cheat to the trusted party. By the definition of the ideal model, with probability $\epsilon = \frac{1}{2^{k'}-1}$ $\mathsf{Sim_C}$ receives back the message undetected , and with probability $1 - \epsilon = 1 - \frac{1}{2^{k'}-1}$ it receives the message corrupted together with $\mathcal{S}$'s input x.
If it receives back corrupted, then it simulates $\mathcal{S}$ aborting due to detected cheating and outputs whatever $\mathcal{C}^*$ outputs.
If it receives back undetected, then it receives $\mathcal{S}$'s input, simulates honest $\mathcal{S}$, and outputs whatever $\mathcal{C}^*$ outputs. We stress that the $\mathcal{C}^*$'s view of execution so far is independent of $\mathcal{S}$'s input, since $\mathcal{C}^*$ never receives more than one wire key per wire. (The only exception to this is the information inferred by $\mathcal{C}^*$ based on the fact that he was undetected in attempting to submit another string to string-selection OT. Specifically, this gives $\mathcal{C}^*$ the knowledge of both wire keys on one of the the output wires of the preceding slice (importantly, he will not learn both input garblings of the corresponding input wire of the current slice). This information is allowed in the GESS protocol, and will not allow to correlate wire keys.) Therefore, the simulation goes through in this case.

To simulate the output slice C_ℓ, $\mathsf{Sim_C}$ first performs the simulation steps of an internal slice described above. Upon completion, he additionally has to reconcile the view with the output of the function. If after the $\mathrm{SOT}_{k,k'}$ step $\mathcal{C}^*$ still has not attempted to cheat, $\mathsf{Sim_C}$ provides to the trusted party the input that was provided by $\mathcal{C}^*$ in the OT of slice C_1, and gets back the output of the computation. Now $\mathsf{Sim_C}$ simply provides $\mathcal{C}^*$ the output translation tables with the mapping of the output values, which would map to the output received from the trusted party. ■

B Proof of Theorem 4

Proof. (sketch) **Security against semi-honest $\mathcal{S}$.** We start with showing that the protocol is secure against the semi-honest sender $\mathcal{S}$. The information received by $\mathcal{S}$ are transcripts of the underlying OTs, and the hash of one of the two shares sequences. Neither leaks information. Firstly, OT's are secure against semi-honest $\mathcal{S}$. Further, an honest receiver uses $r \in \{r_0, r_1\}$ selection string, hence the hash sent to $\mathcal{S}$ will in fact be of one of the two sequences corresponding to the selection strings. The simulator $\mathsf{Sim_S}$ follows naturally.

Security against malicious receiver $\mathcal{R}$. We present the simulator $\mathsf{Sim_R}$ of a malicious $\mathcal{R}^*$, and argue that it produces a good simulation.

$\mathsf{Sim_R}$ starts $\mathcal{R}^*$ and interacts with it, sending it messages it expects to receive, and playing the role of the trusted party for the OT oracle calls that $\mathcal{R}^*$ makes, in which $\mathcal{R}^*$ plays the role of the receiver.

$\mathsf{Sim_R}$ starts by playing OT trusted party k' times, where $\mathcal{R}^*$ is the receiver; as such, $\mathsf{Sim_R}$ receives all k' OT selection bits from $\mathcal{R}^*$ and each time uses a random string s'_j to hand to $\mathcal{R}^*$ as his OT output. If any of the underlying OT's input is abort, then $\mathsf{Sim_R}$ sends abort to the trusted party computing $\mathrm{SOT}_{k,k'}$ and halts, outputting whatever $\mathcal{R}^*$ outputs.

$\mathsf{Sim_R}$ then sends a random κ-bit string h' to $\mathcal{R}^*$, simulating the message of Step 3. $\mathsf{Sim_R}$ then receives a hash value from $\mathcal{R}^*$ as Step 4 message. If this value was *not* computed correctly from the simulated OT strings s'_j and hash h', then $\mathsf{Sim_R}$ sends abort to the trusted party, and terminates outputting whatever $\mathcal{R}^*$ outputs. Otherwise, $\mathsf{Sim_R}$ feeds $\mathcal{R}^*$'s selection bits to the trusted party of $\mathrm{SOT}_{k,k'}$. $\mathsf{Sim_R}$ gets back from the trusted party either:

– string x', one of $\mathcal{S}$'s secrets. In this case, $\mathcal{R}^*$ submitted a valid selection string, and we simulate successful completion. $\mathsf{Sim_R}$ sends, in random order, $x' \oplus H(\oplus_j s'_j)$ and a random string of equal length, and then terminates outputting whatever $\mathcal{R}^*$ outputs.
– $\perp$. In this case, $\mathcal{R}^*$ did not submit a valid selection string, and we simulate abnormal termination. $\mathsf{Sim_R}$ sends $\perp$ to $\mathcal{R}^*$ and terminates outputting whatever $\mathcal{R}^*$ outputs.

We now argue that $\mathsf{Sim_R}$ produces view indistinguishable from the real execution. We first note that $\mathsf{Sim_R}$'s interaction with $\mathcal{R}^*$ is indistinguishable from that of honest $\mathcal{S}$. Indeed, OT secrets delivered to $\mathcal{R}^*$ are distributed identically to real execution. Further, since non-selected OT secrets remain hidden, the string h' sent is also indistinguishable from real execution. Finally, the simulation of Step 5 is indistinguishable from real, since it is infeasible to search for a preimage of H to check whether it satisfies both simulation of Steps 3 and 5. ■

C Performance Analysis Calculation

Consider a circuit C with fan-out 2 of size c gates. We will ignore the costs of n inputs, as usually $c \gg n$. We compare the communication complexity of our protocol against existing solutions, mainly the state-of-the-art GC approach [24, 22]. For completeness, we include the (higher) costs of the GMW approach [5].

We let all solutions use OT extension [7], which reduces the total number of public-key operations to $O(\kappa)$. The communication cost of extending m OTs on ℓ-bit strings in the semi-honest setting [7] is

$$\mathsf{cost}\,(\mathrm{OT}_\ell^m) = \mathsf{cost}\,(\mathrm{OT}_\kappa^\kappa) + 2m\ell + 2\kappa m. \tag{1}$$

We ignore $\mathsf{cost}(\mathrm{OT}_\kappa^\kappa)$, which is independent of the size of the circuit.

Yao Garbled Circuit. Standard GC is secure against a semi-honest $\mathcal{S}$ and covert $\mathcal{C}$. State-of-the-art GC [22] sends at least 2 encryptions per gate, i.e., a total of $2\kappa c$ bits. However, if using free-XOR, this increases to 3 encryptions per non-XOR gate [22], i.e., a total of $3\kappa c$ bits. For simplicity, set $\mathsf{cost}(\mathsf{Yao}) = 2\kappa c$.

GMW Approach. Each AND/OR gate in C triggers a *1-out-of-4 OT* invocation. In the semi-honest setting, one invocation of a 1-out-of-4 OT can be reduced to two invocations of 1-out-of-2 OT. Each OT in GMW approach requires the sender's input to be of length 1 bit. Using Equation 1, the total communication complexity of SFE is $(2 \cdot (2c) \cdot 1) + (2\kappa \cdot (2c))$. Thus, $\mathsf{cost}(\mathsf{GMW}) = 4c + 4\kappa c$.

Our Approach. The wiring of the circuit plays an important role in the actual cost of our approach. For simplicity and w.l.o.g., we assume C is a rectangular circuit of constant width, where each gate has fan-out 2. Let C divided into ℓ slices, each of depth d'. Let k (resp. k') be key length on input (resp. output) wires of each slice. Recall, that k grows as $O(2^{d'})$ due to fan-out 2.

The costs of the underlying $\mathrm{SOT}_{k,k'}$ instances dominate the cost of our protocol. Our protocol requires a single instance of $\mathrm{SOT}_{k,k'}$ (which translates to k' 1-out-of-2 OT invocations) for each input gate in a slice. This amounts to a total of $k' \cdot (c/d')$ 1-out-of-2 OT invocations in a rectangular circuit. When using OT extension, we may first execute 1-out-of-2 OTs using random inputs, and then correct them later when OT inputs are available. (We omit the details of this correction.) When executing a very wide circuit, or a number of smaller ones in parallel, this correction can be avoided whatsoever simply by running OT extension on each layer of the circuit when $\mathcal{C}$'s inputs are known.

Semi-honest Setting. Here we set $k' = 1$ and note that the random pads used to mask the secrets corresponding to the input wires are not necessary. From Equation 1 and assuming no correction, we obtain cost $(2k + 2\kappa)c/d'$.

Setting $k = \kappa$, and hence $d' = O(\log k)$, we obtain the cost $O(\kappa c/\log \kappa)$. We also calculate concrete efficiency improvements (see Table 1).

	k	cost as a function of κ	cost when $\kappa = 128$	cost when $\kappa = 256$
$d' = 1$	4	$(8 + 2\kappa)c$	$264c$	$520c$
$d' = 2$	16	$(32 + 2\kappa)c/2$	$144c$	$272c$
$d' = 3$	56	$(112 + 2\kappa)c/3$	$122.67c$	$208c$
$d' = 4$	180	$(360 + 2\kappa)c/4$	$154c$	$218c$
$d' = 5$	542	$(1084 + 2\kappa)c/5$	$268c$	$319.2c$

Table 1. Concrete cost of semi-honest SFE for different slide depth values d' (k is calculated from d' assuming rectangular fan-out 2 circuit). GC costs at least $2\kappa c$. Thus when $\kappa = 128$ GC cost is $256c$, and when $\kappa = 256$, GC cost is $512c$.

Covert Client Setting. Compared to semi-honest cost, here we need to add κ bits per instance of $\mathrm{SOT}_{k,k'}$ for the random pad of Step 2(a)ii of Protocol 1, i.e., a total of $\kappa \cdot (c/d')$ bits for the entire protocol. Further, each instance of $\mathrm{SOT}_{k,k'}$ translates to k' 1-out-of-2 OT instances, an needs additional 2κ bits that confirm the honest behavior of the receiver (Steps 3 and 4 of Protocol 2), and an additional 2κ bits to transfer the masked secrets (Step 5 of Protocol 2). Using equation 1, we obtain the cost $(2kk' + 2\kappa k' + 5\kappa)c/d'$

Setting $k = \kappa$ and hence $d' = O(\log(k/k'))$, the cost is $O(k'\kappa c/\log(\kappa/k'))$. For $k' = 2$ and today's practical $\kappa = \{128, 256\}$, we show the concrete cost of our protocol in Table 2.

	k	cost as a function of κ	cost when $\kappa = 128$	cost when $\kappa = 256$
$d' = 1$	7	$(28 + 9\kappa)c$	$1180c$	$2332c$
$d' = 2$	24	$(96 + 9\kappa)c/2$	$624c$	$1200c$
$d' = 3$	76	$(304 + 9\kappa)c/3$	$485.33c$	$869.33c$
$d' = 4$	228	$(912 + 9\kappa)c/4$	$516c$	$804c$
$d' = 5$	654	$(2616 + 9\kappa)c/5$	$753.6c$	$984c$
$d' = 6$	1802	$(7208 + 9\kappa)c/6$	$1393.33c$	$1585.33c$

Table 2. Concrete cost of our SFE protocol with a covert client ($\epsilon = 1/3, k' = 2$) for different slide depth values d' (k is calculated from d' assuming rectangular fan-out 2 circuit). GC costs at least $2\kappa c$. Thus when $\kappa = 128$ GC cost is $256c$, and when $\kappa = 256$, GC cost is $512c$.

C.1 Slicing Circuits

W.l.o.g we assume that the given circuit C contains at most n gates at each depth and that gates at any given depth receive input only from the output wires of the gates at the previous level. We bound the size c of the circuit by $(n \cdot d)$, and make an observation regarding the number of distinct wires that connect gates at a given depth to the next.

Note that the n gates at any given depth have $2n$ associated input wires. Since all these $2n$ input wires comes from the n gates at the previous level. Therefore, the number of distinct wires that connect gates at a given depth to the next equals n. Therefore, when we slice the circuit C into d' slices, the total number of SOT invocations required per slice is at most $n = c/d'$.

Our next discussion is on how to apply the GESS algorithm to each slice. Specifically we cope with the following issues: (1) GESS construction is applicable only to circuits representing boolean formulae, and (2) GESS assigns input keys of different lengths to the left and right input wires of any gate.

Given a slice C_i, we derive a tree-circuit C'_i (representing a boolean formulae) from C_i such that the GESS algorithm can be applied to C'_i. Naturally such a derivation results in a $2^{d'}$ expansion factor on the number of input gates (and thus on the input wires) for each slice in C'_i relative to C_i. Recall that each slice C_i has n distinct output wires and n distinct input wires. So even though each of the n output wires now correspond to $2^{d'}$ input wires in the tree-circuit C'_i, these $2^{d'}$ wires are roughly divided uniformly between n distinct input wires. Furthermore, we argue that it is possible to *rebalance* the tree-circuit C'_i such that the $2^{d'}$ duplicates of each wire appears in all $2^{d'}$ positions in the tree exactly once. Given the above, we derive the length of the input keys required for each of the n distinct input wires of C as the sum of the length of the input keys required for a single output wire in a depth d' tree-circuit C'_i. When the output wire carries a single bit, the sum of the length of the input keys is bounded by $2^{d'}(d'^2 \log(d'+1) + d' \log(d'+1) + d' + 1) \approx 2^{d'} d'^2 \log d'$, dominated by the $2^{d'}$ term [11]. Thus we conclude that the depth of a slice d' is related to the length of the input keys k by the equation $d' = O(\log k)$. More generally when the output wire is assigned a k' bit value, the slice depth $d' = O(\log(k/k'))$. Table 3 shows the *exact* length of the GESS input keys required for each distinct input wire as a function of the depth d' and k'.

k'	$d' = 1$	$d' = 2$	$d' = 3$	$d' = 4$
1	4	16	56	180
2	7	24	76	228
3	10	32	96	276
4	13	40	116	324
5	16	48	136	372

Table 3. Dependence of the length of input keys on depth d and on-line security parameter k'. Example: for a slice of depth 3 and for $k' = 2$, the length of the input keys (i.e., k) is 76.

Finally, note that the expansion factor applies only to the length of the *input keys* to slice C_i. Although the output wires are also duplicated in the tree-circuit C'_i, any one of these k' bit outputs can be used in our protocol to obtain the corresponding keys for the next slice C'_{i+1}.