Gate Evaluation Secret Sharing and Secure One-Round Two-Party Computation

Vladimir Kolesnikov

Department of Computer Science,
University of Toronto, Canada
vlad@cs.utoronto.ca

Abstract. We propose *Gate Evaluation Secret Sharing* (GESS) – a new kind of secret sharing, designed for use in secure function evaluation (SFE) with minimal interaction. The resulting simple and powerful GESS approach to SFE is a generalization of Yao's garbled circuit technique.

We give efficient GESS schemes for evaluating binary gates and prove (almost) matching lower bounds. We give a more efficient information-theoretic reduction of SFE of a boolean formula F to oblivious transfer. Its complexity is $\approx \sum d_i^2$, where d_i is the depth of the i-th leaf of F.

1 Introduction

The main motivation for this work is one-round secure function evaluation (SFE). SFE is one of the core problems of cryptography. We consider the following one-round two semi-honest parties setting. Alice and Bob wish to compute a function f of their inputs x and y respectively: Alice sends the first message to Bob, Bob replies, and Alice computes $f(x, y)$. Both parties follow the prescribed protocol, but try to infer additional information from the messages they receive. This problem has been extensively studied, and very efficient solutions (with cost linear in the circuit representing f) exist (Yao's garbled circuit [3,21,24,25,27]), when Alice is polytime bounded. When Alice is computationally unlimited, only much less efficient algorithms are known [4,9,18,19,20,26].

One-round SFE is particularly interesting for several reasons. Firstly, from a practical point of view, interaction necessarily involves latencies in message deliveries, and in many practical situations waiting for messages dominates the entire computation time. Secondly, a large volume of research, e.g. [8,12,18,19], aims specifically at reducing round complexity of multiparty protocols. Investigating the two-party one-round model may help increase our understanding of general secure multiparty computation. Finally, the recently popular area of secure autonomous agent computing (see, e.g. [1,8]) relies on one-round protocols, commonly implemented via encrypted circuit constructions. A variety of very useful mobile agents computing simple functions may benefit from our improvements. One such example, discussed in [1], is that of a shopping agent that would accept a sales offer if it is below a certain threshold.

We approach the problem in a general way by reducing SFE to oblivious transfer (OT). OT is a powerful primitive, and is the subject of a vast amount of

B. Roy (Ed.): ASIACRYPT 2005, LNCS 3788, pp. 136–155, 2005.

research. It has been studied in many settings; for example, OT is instantiable with information-theoretic (IT) security (e.g. with noisy and quantum channels or a distributed sender [23]). Our SFE constructions automatically apply to all of the above (and many other) settings and will benefit from future OT research.

1.1 Our Contributions and Outline of the Work

Our main idea is a new *simple* way of evaluating circuit gates securely by using a new type of secret sharing, which we call *Gate Evaluation Secret Sharing* (GESS). Our method can be viewed as a generalization of Yao's garbled gate evaluation procedure, offering a simple and powerful approach for designing efficient SFE protocols. Our method is flexible, and not limited to $\vee, \wedge, \neg$ gates. Circuits with special purpose (e.g. non-binary) gates may be designed and implemented via GESS to achieve better efficiency for specific functions (see, e.g., Sect. 2.6).

We show how a composition of GESS schemes can be used to efficiently reduce SFE to (parallel executions of) 1-out of-2 OT. Given a boolean formula, we obtain a *one-round* reduction, meaning that an instantiation of OT results in a SFE protocol, the security and round complexity of which are that of the underlying OT. Our reduction is very efficient. Previous approaches in part suffer from the exponential (in depth) cost of evaluation of a gate, which has intuitievely appeared necessary. We break this intuition by providing a scheme for gate evaluation whose cost is only *quadratic* in the depth of the gate. Further, in our reduction, we don't "pay" for the internal gates of the formula. For a depth d circuit, this results in a factor of approximately $2^{O(\sqrt{d})}$ improvement over previous solutions: $O(2^d d^2)$ vs $\Theta(2^d 2^{\Theta(\sqrt{d})})$. (Like all other approaches, ours suffers from the fact that the number of gates may be exponential in depth. Thus, we offer polytime reduction of only NC^1 circuits.) We prove non-trivial lower bounds, showing that our constructions are almost optimal in the GESS framework.

The GESS approach is especially efficient on small circuits, since it does not use encryption. In Sect. 2.6, we demonstrate this by a new efficient protocol for the Two Millionaires problem. This protocol also serves as an example of designing and implementing custom GESS gates.

We start with describing previous approaches and giving conceptual and performance comparisons to our work (Sect. 1.2). We then present intuition for our approach and introduce the necessary formal definitions in Sect. 2 and 2.1. We present our constructions, lower bounds and performance analysis in Sect. 2.3 – 2.5. In Sect. 2.6 we present a new solution of the Two Millionaires problem.

In Sect. 3, we show how to use GESS to allow polytime SFE of polysize circuits, when Alice is polytime. In effect, we obtain another implementation of Yao's garbled circuit approach for the model with polytime Alice, offering essentially the same computational and communication complexity as its best implementations. The natural and efficient handling of the computational setting demonstrates the generality of the GESS approach. We mention that the efficiency of Yao's garbled circuit technique in the standard model can be (slightly) improved by using IT GESS on "the bottom part" of the circuit (see discussion in Sect. 3).

1.2 Comparisons with Related Previous Work

General discussion. Note the frequent use of a variety of secret sharing schemes in secure multiparty computation. They are always used, however, to share secrets *among players*. We contrast this with our novel use, where secrets are shared *among wires* and given to the player who performs reconstruction.

We note that some of the previous approaches (e.g. [9,18,19,20]) are applicable to more general representations of functions (e.g. by arithmetic formulas or branching programs (BP)). Many functions may have especially efficient representations when not restricted to boolean formulas (the setting we consider); such functions may not benefit from our constructions.

Although our reductions are efficient for polysize boolean formulas of arbitrary depth, they perform better on balanced formulas. For the latter, the complexity is quasi-linear (vs. cubic for highly unbalanced formulas) in the size of the formula. Note that it is possible ([7,6]) to rebalance any formula to obtain an equivalent log-depth balanced formula, at the cost of small increase in its size (see end of Sect. 2.4 for more discussion).

Therefore, for the remainder of this section, assume that we are given a boolean formula (or an NC^1 circuit, which can be viewed as one), which is rebalanced if it benefits the approach considered.

Let d be the depth of the formula or the circuit.

Comparing our reduction to previous constant-round approaches.

Kilian [20] was the first to show a one-round IT reduction (of complexity $\Theta(4^d)$) of SFE to OT. Kilian relies on Barrington's [2] representation of NC^1 circuits as permutation BPs. It is possible to replace Barrington's representation in Kilian's construction with a more efficient construction of Cleve [9] (see, e.g. Cramer et al. [10]). The resulting complexity is $\Theta(2^d 2^{\Theta(\sqrt{d})})$, which is the best previously known for NC^1 circuits and (re)balanced formulas.

Ishai and Kushilevitz [18,19] suggested a way of representing a circuit as a predicate on a vector of degree 3 (degree of the input variables x_i is 1) *randomizing polynomials*. Their construction assigns an (exponential in d in size) polynomial representation to each wire of the corresponding fan-out 1 circuit, and implies a one-round SFE-to-OT reduction, of complexity $\Theta(4^d)$. They also previously suggested a related *Private Simultaneous Messages* (PSM) model [17] of computation. They showed how to evaluate functions computed by BPs in the PSM model (and also in our SFE-to-OT reduction model) with resources quadratic in the size of the BP. (Recall, BPs are more powerful than permutation BPs or formulas.) For our setting, their approach implies a one-round SFE-to-OT reduction of cost $\Theta(4^d)$, using an (almost) linear in size transformation of a formula to a BP [14].

Our reduction of boolean formulas is simpler and more efficient (costing $O(2^d d^2)$) than the above approaches.

Yao's garbled circuit approach can also be used for such reduction (see, e.g. [19]). The idea is to use an IT-secure two-time encryption scheme (e.g. using one-time pad) in Yao's garbled circuit. The keys of such a scheme must be more

than twice the size of the secret, causing an exponential (in d) growth of the size of secrets, even in fan-in 1 circuits[1]. The complexity of such a scheme is about $\Theta(4^d)$ (up to 2^d leaves, each of size up to 2^d). Our approach is a generalization and an improvement of this approach.

Sander, Young and Yung (SYY) ([26]) present a "fully homomorphic" encryption scheme and apply it to SFE. The encryption size grows exponentially with the number of the applied OR operations, resulting in $\Theta(8^d)$ cost of SFE. Beaver [4] suggests an optimization of the SYY pyramid and extends the approach to the multiparty setting, achieving complexity $\Theta(4^d)$. Further, using the representation of Feige, Kilian and Naor [11] of NLOGSPACE as a product of polysize matrices, he shows how to compute it in one round, bootstrapping the SYY approach, also achieving complexity $\Theta(4^d)$. Our approach is conceptually different, simpler, more composable, uses fewer assumptions, and offers complexity of at most $O(2^d d^2)$. Also, unlike SYY, we do not have the requirement of a layered circuit, which further increases our performance improvement.

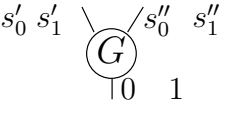
Finally, we mention (but do not discuss) a variety of non-constant round solutions (e.g. [22] and [16]).

1.3 Our Setting

We are working in a setting with two semi-honest participants who use randomness in their computation. A large part of our work concerns reductions of various problems to the OT oracle. In the semi-honest model, secure reductions result in secure protocols when the called oracles are replaced by their secure implementations. Further, the oracles' implementations may be run in parallel, which, with natural OT implementations, results in secure one-round protocols. See Goldreich [15] for definitions, discussion and the composition theorem.

2 The GESS Approach

The intuition behind the GESS approach. Suppose first that the circuit C consists of a single binary gate G with two inputs, one held by Alice, and one by Bob. To transfer the value of the output wire to Alice, Bob encodes possible values of each of the two input wires and transfers to Alice two of the four encodings – one for each wire. Encoding of Alice's wire value is sent via OT. Each pair of encodings that can be possibly sent, has to allow the recovery of the corresponding to G value of the output wire, and cannot carry any other useful information. Consider the following example.



[1] Note the distinction between this flavour of Yao's approach and its standard version for evaluation of polysize circuits (e.g. [3,25,24,21]). The latter is not a reduction to OT; e.g, it cannot be used to construct one-round protocols IT-secure against Alice.

Given the possible output values $0, 1$ and the semantics of the gate G, Bob generates encodings of the input wires' values $(s'_0, s'_1), (s''_0, s''_1)$, such that each possible pair of encodings s'_i, s''_j, where $i, j \in \{0, 1\}$, allows to reconstruct $G(i, j)$, and carries no other information. Now, if Bob sends Alice shares corresponding to their inputs, Alice would be able to reconstruct the value of the output wire, and nothing else.

This mostly corresponds to our intuition of secret sharing schemes. Indeed, the possible gate outputs play the role of secrets, which are shared and then reconstructed from the input wires encodings (shares).

Our next observation is that Bob need not share the *values* of the output wire, but instead can share their *encodings*, which, in turn, may be input shares of another gate. Thus, Alice and Bob can recursively apply the GESS approach to multi-gate circuits. For each wire, Alice will only be able to obtain one secret – the one corresponding the the value of the wire on the parties' inputs.

2.1 The Definition of Gate Evaluation Secret Sharing

We now formally state the desired properties of the secret sharing scheme. While the idea of the definition is quite simple, it is somewhat burdened with notation due to the necessary level of formalism. For simplicity, we present the definition for the case of a gate with two binary inputs and a binary output, postponing the presentation of its most general form to Appendix A (Def. 2). A simple instructive example of a GESS scheme is Constr. 2 in Sect. 2.3.

Let G be a gate with two binary inputs and a binary output. Also denote by $G : \{0, 1\} \times \{0, 1\} \mapsto \{0, 1\}$ the function computed by gate G. Let SEC be the domain of secrets. Suppose we've associated a secret $s_i \in SEC$ with each of the two possible values i of the output wire of G. In general, distributions of s_0 and s_1 may be dependent, so we talk about a *tuple* of secrets $\langle s_0, s_1 \rangle$ from a domain of tuples $TSEC \subset SEC^2$ associated with the output wire. We want to assign a share to each value of the two input wires, such that each combination of shares allows reconstruction of (only) the "right" secret. As do secrets, shares on a wire form a tuple: $\langle sh_{10}, sh_{11} \rangle \in TSH_1 \subset (SH_1)^2$ on wire 1, and $\langle sh_{20}, sh_{21} \rangle \in TSH_2 \subset (SH_2)^2$ on wire 2. In our notation, $sh_{ij} \in SH_i$ is the share of the i-th input wire ($i \in \{1, 2\}$), corresponding to the value $j \in \{0, 1\}$.

Definition 1. *(Gate evaluation Secret Sharing) A* gate evaluation *secret sharing scheme (*GESS*) for evaluating G as above (we also say GESS* implementing *G) is a pair of algorithms (Shr, Rec) (with implicitly defined secrets domain SEC, secrets tuples domain $TSEC$, two share domains SH_1 and SH_2 and two share tuples domains TSH_1, TSH_2), such that the following holds.*

The probabilistic share generation algorithm Shr takes as input a two-tuple of secrets $\langle s_0, s_1 \rangle \in TSEC$ and outputs two tuples of shares (one for each wire), where, $\forall i \in \{1, 2\}$, the i-th tuple $t_i \in TSH_i$ consists of two shares $sh_{ij} \in SH_i$. The deterministic share reconstruction algorithm Rec takes as input two elements $sh_1 \in SH_1$ and $sh_2 \in SH_2$ and outputs $s \in SEC$.

Let $v = \langle v_1, v_2 \rangle \in \{0,1\} \times \{0,1\}$ be a selection vector. Define the selection function $Sel(\langle sh_{10}, sh_{11} \rangle, \langle sh_{20}, sh_{21} \rangle, v) = \langle sh_{1v_1}, sh_{2v_2} \rangle$. Write $V_1 \equiv V_2$ to denote that V_1 and V_2 are distributed identically.

Shr and Rec satisfy the following conditions:

- *correctness: for all random inputs of Shr and secrets tuples $\langle s_0, s_1 \rangle \in TSEC$, $\forall v \in \{0,1\}^2, Rec(Sel(Shr(\langle s_0, s_1 \rangle)), v)) = s_{G(v)}$*
- *privacy (selected shares contain no information other than the value $s_{G(v)}$): There exists a simulator Sim, such that $\forall \langle s_0, s_1 \rangle \in TSEC$ and any $v \in \{0,1\}^2$: $Sim(s_{G(v)}) \equiv Sel(Shr(\langle s_0, s_1 \rangle), v)$*

Observation 1. *A simple generalization of this definition (required for discussion in Sect. 2.3 and 2.4) considers the identity gate G_I with a four-valued output wire, where each output corresponds to a pair of inputs. In this case, the secrets form a 4-tuple $\langle s_{00}, ..., s_{11} \rangle$, while there are still two two-tuples of shares. Note that we can convert GESS implementing G_I into GESS implementing any other binary gate by simply restricting some of the secrets to be equal. Denote the correspondence between a secret $s \in SEC$ and the wire value $v \in \{0,1\}$ by $s \leftrightarrow v$. Then setting $s_{01} = s_{10} = s_{11} \leftrightarrow 1, s_{00} \leftrightarrow 0$ gives the implementation of the OR, and $s_{00} = s_{01} = s_{10} \leftrightarrow 0, s_{11} \leftrightarrow 1$ – of the AND gates. NOT gates can be implemented "for free" by simply eliminating them and inverting the correspondence of the appropriate wire's values and secrets.*

Observation 2. *We note that, in contrast with the traditional approach of multi-secret sharing schemes, our definition allows the possibility that a single share gives out some information about a secret. It is easy to see, however, that this information must be common to* every *secret, since otherwise it is possible to determine whether a corresponding combination of secret/share occurred, which allows to easily construct a distinguisher breaking the privacy requirement of GESS. Further, shares of the same wire, corresponding to different values, must be distributed identically (otherwise a distinguisher exists).*

The definition is given for specific input and output domains, and therefore we do not talk about polynomial bounds on *Shr* and *Rec*. However, in practice, we are interested in *ensembles* of schemes and want them to be uniform polytime algorithms. We won't insist on an ensemble of *efficient* simulators, because an efficient simulator exists if any one exists. Indeed, an efficient simulator can simply output $Sel(Shr(\langle s_0, s_1 \rangle), v)$, where at least one of the secrets s_i is equal to s, and v is any selection vector, such that $G(v) = i$.

2.2 Reduction of SFE to OT Using GESS

Suppose Alice and Bob have a circuit C, consisting of fan-out 1 gates $G_1, G_2, ...$ We formally describe a reduction of securely evaluating C on their inputs to calls to OT, resulting in a one-round protocol. Again, for simplicity of presentation we assume that all gates G_i are fan-in 2 binary gates.

Assume that for every gate G of C, there exists a GESS $GESS_G : (Shr_G, Rec_G)$ of Def. 1 with appropriate secret domains (as described below). We give explicit

constructions (e.g. Constr. 2 in Sect. 2.3) of such schemes for all gates with two binary inputs. We note that GESS for every other gate can be constructed (e.g. from Constr. 1 instantiated with GESS of Constr. 2).

Construction 1. *(Reducing SFE to OT)* **Bob's precomputation.** *Bob starts with the output gate. He sets the secrets domain SEC of it to be $\{0,1\}$ and sets the secrets tuple to $\langle 0,1\rangle$. He proceeds through gates of C recursively as follows.*

Consider a gate G. Let $TSEC$ and a secrets tuple $t = \langle s_0, s_1\rangle \in TSEC$ are given for G. Let $GESS_G$ be a GESS scheme implementing G with secrets tuples domain $TSEC \subset SEC^2$. Bob runs Shr_G on the secrets tuple t and obtains two tuples of shares $t_1 \in TSH_1$ and $t_2 \in TSH_2$, corresponding to the first and second input wires of G respectively. Let G'_i be the i-th input gate of G ($i \in \{0,1\}$). Then Bob processes G'_i as follows. He treats the tuple of shares $t_i \in TSH_i$ of G's input wire as the tuple of secrets of G'_i, and TSH_i – as the secrets tuples domain of G'_i. Bob now applies the algorithm of this paragraph to G'_i.

Eventually, Bob obtains secrets tuples for all input wires of C. Note that Bob's choices of instances of GESS schemes for the gates of C are deterministic and built into the protocol; this explicates the corresponding Rec procedures.

Interaction. *For each input wire associated with Alice, she and Bob make (parallel) calls to OT oracles. Alice has the wire's input and Bob has the tuple of secrets as their inputs of each of the calls. For each input wire associated with Bob, Bob sends Alice the corresponding secret from that wire's tuple of secrets*[2].

Alice's computation. *Alice obtains results of the OT and the secrets corresponding to Bob's inputs. Alice proceeds, from the top down on the circuit C, as follows. For each gate, Alice knows the secrets corresponding to the inputs of the gate, and the corresponding Rec procedure. She runs Rec on the input secrets and obtains the output secret. She proceeds in this manner until she obtains the secret corresponding to the output wire. Alice outputs this secret.*

Theorem 1. *Constr. 1 is a non-cryptographic reduction (thus unconditionally secure against both Alice and Bob) of SFE of C to OT, in the semi-honest model.*

The proof of Theorem 1 is intuitive and is presented in Appendix B.

Observation 3. *A circuit C with fan-out greater than* 1 *can be converted into a corresponding (potentially very large) tree-circuit C' by duplicating C's subtrees where appropriate. Equivalently, one can view the secrets as being computed and propagated by Bob* in parallel *on the same wire. Note that we, however, need not increase the* number *of corresponding OT instances due to the growth of C' relative to C (until a certain efficiency threshold is reached). Rather, Bob's inputs to OT will be longer (without the increase in the total number of bits transferred). This will often result in significant computational and communication savings.*

[2] This message is appended to Bob's messages of the n-round instantiations of OT oracles to form an n-round protocol.

2.3 GESS for Gates with Two Binary Inputs

We now present an efficient ensemble of GESS schemes (indexed by the secrets domains) implementing any binary gate with two binary inputs. This construction is a building block of a more efficient Constr. 3. We present GESS for the 1-to-1 gate function $G : \{0,1\}^2 \mapsto \{00, 01, 10, 11\}$, where $G(0,0) = 00, G(0,1) = 01, G(1,0) = 10, G(1,1) = 11$ (see Observation 1 for justification).

Let the secrets domain be $SEC = \{0,1\}^n$, and four (not necessarily distinct) secrets $s_{00}, ...s_{11} \in SEC$ are given; the secret s_{ij} corresponds to the value $G(i,j)$ of the output wire. Note that $|SEC| \geq 4$ need not hold; our scheme is interesting even when $|SEC| \geq 2$.

The intuition for the design of the GESS scheme is as follows. We first randomly choose two strings $R_0, R_1 \in_R SEC$ to be the shares sh_{10} and sh_{11} (corresponding to 0 and 1 of the first input wire). Now consider sh_{20} – the share corresponding to 0 of the second input wire. We want this share to produce either s_{00} (when combined with sh_{10}) or s_{10} (when combined with sh_{11}). Thus, the share $sh_{20} = B_{00}B_{10}$ will consist of two blocks. One, $B_{00} = s_{00} \oplus R_0$, is designed to be combined with R_0 and reconstruct s_{00}. The other, $B_{10} = s_{10} \oplus R_1$, is designed to be combined with R_1 and reconstruct s_{10}. Share $sh_{21} = B_{01}B_{11}$ is constructed similarly, setting $B_{01} = s_{01} \oplus R_0$ and $B_{11} = s_{11} \oplus R_1$. Note the indexing notation – the secret s_{ij} is always reconstructed using B_{ij}.

Both leftmost blocks B_{00} and B_{01} are designed to be combined with the same share R_0, and both rightmost blocks B_{10} and B_{11} are designed to be combined with R_1. Therefore, we append a 0 to R_0 to tell *Rec* to use the left block of the second share for reconstruction, and append a 1 to R_1 to tell *Rec* to use the right block of the second share for reconstruction. Finally, to hide information leaked by the order of blocks in shares, we perform the following. We randomly choose a bit b; if $b = 1$, we reverse the order of blocks in *both* shares of wire 2 and invert the appended pointer bits of the shares of wire 1. More formally:

Construction 2. *(GESS ensemble for gates with two binary inputs.) Let* $SEC = \{0,1\}^n$ *and* $TSEC = SEC^4$ *be the secrets domains. Let the secrets tuple* $\langle s_{00}, ..., s_{11} \rangle \in TSEC$ *be given. The domains of shares are:* $SH_1 = \{0,1\} \times SEC$ *and* $SH_2 = SEC^2$*. Note that* $TSH_1 = SH_1^2$ *and* $TSH_2 = SH_2^2$*.*

Shr chooses $b \in_R \{0,1\}, R_0, R_1 \in_R SEC$ *and sets blocks*
$B_{00} = s_{00} \oplus R_0$, $B_{01} = s_{01} \oplus R_0$, $B_{10} = s_{10} \oplus R_1$, $B_{11} = s_{11} \oplus R_1$.
Shr sets the tuples of shares $\langle sh_{10}, sh_{11} \rangle \in SH_1, \langle sh_{20}, sh_{21} \rangle \in SH_2$ *as follows*

	wire 1	*wire* 2, *if* $b = 0$	*wire* 2, *if* $b = 1$
wire value 0	$sh_{10} = bR_0$	$sh_{20} = B_{00}B_{10}$	$sh_{20} = B_{10}B_{00}$
wire value 1	$sh_{11} = \bar{b}R_1$	$sh_{21} = B_{01}B_{11}$	$sh_{21} = B_{11}B_{01}$

Rec proceeds as follows. On input $Sh_1 = b'r$, $Sh_2 = a_0a_1$, *Rec outputs* $r \oplus a_{b'}$.

Theorem 2. *For each* $n \in \mathbb{N}$, *Constr. 2 is a GESS scheme.*

Proof. (Sketch): To prove correctness, we need to show that no matter what the random choices of *Shr* and the wire values i_1, i_2 are, *Rec* always reconstructs $s_{G(i_1,i_2)}$. Verification of correctness is simple and is moved to Appendix D.

We now prove security. Suppose secrets $s_{00}, ..., s_{11}$ are given. This determines the distribution on the Shr generated shares. Let the input wire values i_1, i_2 be given. Then the distribution P on the corresponding pair of shares $\langle sh_{1i_1}, sh_{2i_2}\rangle$ and the secret $s = s_{G(i_1,i_2)}$ shared by the pair are determined. The goal of the simulator is, given only s, to generate a pair of shares distributed identically to P. Note that this exactly corresponds to the privacy condition $Sim(s_{G(i_1,i_2)}) \equiv Sel(Shr(s_{00}, ..., s_{11}), \langle i_1, i_2\rangle)$ of Def. 1.

The following natural simulator $Sim(s)$ suffices. On input $s \in SEC$, Sim chooses a random bit $d \in_R \{0,1\}$ and random strings $p, q \in_R SEC$. If $d = 0$, he outputs $(\langle d, p\rangle, \langle p \oplus s, q\rangle)$, otherwise he outputs $(\langle d, p\rangle, \langle q, p \oplus s\rangle)$. The simple proof by case analysis is presented in Appendix D. □

The Permute and Point (PP) Technique. We note the application of the following technique: we permuted the blocks of the shares of the second wire, and appended pointers to the shares of the first wire, hiding information contained in the order of blocks. We use the same idea in all other constructions in this paper (of Sect. 2.4 and 2.6). We believe this technique is likely to be useful in many other GESS constructions; it may also have other applications.

Observation 4. *We note that the simulator Sim of Theorem 2 is the same for every gate function – it is only the secrets semantics that defines the semantics of the gate. Therefore, Sim can simulate gates without knowing what they are. Therefore, when this secret sharing scheme is plugged into the protocol of Sect. 2.2, semantics of all gates are unconditionally hidden from Alice - she only knows the wire connections of C.*

2.4 The Main Construction – GESS for AND/OR/NOT Circuits

Note the inefficiency of Constr. 2, causing the shares corresponding to the second input wire be double the size of the gate's secrets. While, in some circuits, we could avoid the exponential (in depth) secret growth by balancing the direction of greater growth toward more shallow parts of the circuit, a more efficient solution is desirable. We discuss only AND/OR circuits, since NOT gates are given for "free" (see Observation 1).

Recall, in Constr. 2 each of the two shares of the second wire consists of two blocks. Observe that in the case of OR and AND gates either left or right blocks of these two shares are equal. We use this property to reduce (relative to Constr. 2) the size of the shares when the secrets are of the above form. Our key idea is to view the shares of the second wire as being equal, except for one block.

Suppose each of the four secrets consists of n blocks and the secrets differ only in the j^{th} block, as follows:

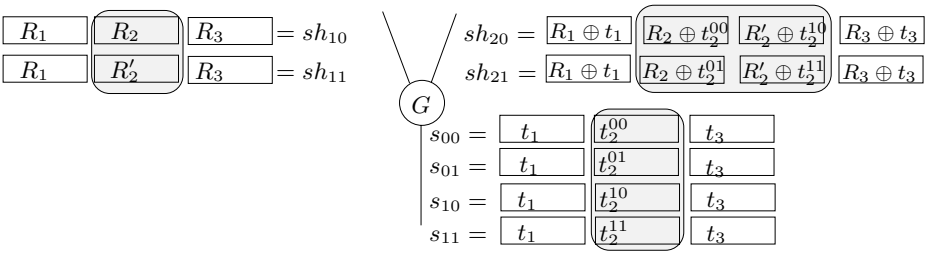
$$\begin{array}{lllllllll} s_{00} = (& t_1 & \ldots & t_{j-1} & t_j^{00} & t_{j+1} & \ldots & t_n &), \quad ... \\ s_{11} = (& t_1 & \ldots & t_{j-1} & t_j^{11} & t_{j+1} & \ldots & t_n &), \end{array}$$

where $\forall i = 1..n$: $t_i, t_j^{00}, t_j^{01}, t_j^{10}, t_j^{11} \in D$, for some domain D of size k. It is

convenient to consider the *columns* of blocks, spanning across the shares. Every column (with the exception of the j-th) consists of four equal blocks. We stress that the index j is only determined by the secrets, and must not be recovered at reconstruction.We construct a GESS for gates with two binary inputs, where the size of each share of the first wire is $n(k + \lceil \log(n+1) \rceil)$ and of the second wire is $(n+1)k$. Further, each share of the first wire consists of n blocks of size $|D| + \lceil \log(n+1) \rceil$, and all but one pair of corresponding blocks are equal between the shares. Each share of the second wire consists of $n+1$ blocks of size $|D|$ and, for OR and AND gates, all but one pair of corresponding blocks are equal between the shares. Since the generated shares satisfy the above conditions on secrets, repeated application of this GESS for OR and AND gates is possible.

The scheme's intuition. For simplicity of presentation, we do not present the GESS scheme in full generality here (this is postponed to Appendix C). We show its main ideas by considering the case where the four secrets consist of $n = 3$ blocks each, and $j = 2$ is the index of the column of distinct blocks.

Our idea is to share the secrets "column-wise", that is to treat each of the three columns of blocks of the secrets as a tuple of subsecrets and share this tuple separately, producing the corresponding subshares. Consider sharing the 1-st column. All four subsecrets are equal (to $t_1 \in D$), and we share them trivially by setting both subshares of the first wire to a random string $R_1 \in_R D$, and both subshares of the second wire to be $R_1 \oplus t_1$. Column 3 is shared similarly. We share column 2 as in Constr. 2 (highlighted on the diagram), omitting the last step of appending the pointers and permutation. This preliminary assignment of shares (still leaking information due to order of blocks) is shown on the diagram.



Note that the reconstruction of secrets is done by XOR'ing the corresponding blocks of the shares, and, importantly, the procedure is the same for both types of sharing we use. For example, given sh_{10} and sh_{21}, we reconstruct the secret $(R_1 \oplus (R_1 \oplus t_1), \quad R_2 \oplus (R_2 \oplus t_2^{01}), \quad R_3 \oplus (R_3 \oplus t_3)) = s_{01}$.

The remaining (PP) step (not shown on the diagram) is to randomly permute the order of the four columns of both shares of wire 2 and to append $(\log 4)$-bit pointers to each block of the shares of wire 1, telling *Rec* which block of the second share to use. Note that the pointers appended to both blocks of column 1 of wire 1 are the same. The same holds for column 3. Pointers appended to blocks of column 2 are different. For example, if the identity permutation was applied, then we will append "1" to both blocks R_1, "2" to R_2, "3" to R'_2, and "4" to both blocks R_3. Because G is either an OR or an AND gate, both tuples

of shares maintain the property that all but one pairs of corresponding blocks are equal between the shares of the tuple. Note that it is not a problem that the index of the column with different entries on input wire 1 is the same as that on the output wire: since the adversary never sees both shares of any wire, this index remains unconditionally hidden.

Construction 3. *(GESS for AND/OR gates) The presented construction can be naturally generalized for an arbitrary number of blocks n of size k and for arbitrary index j of the column with differing blocks. The formal presentation of this general construction is postponed to Appendix C (Constr. 6).*

Theorem 3. *For each $n, k, j \in \mathbb{N}$, Constr. 3 is a GESS scheme as defined by (a generalization of) Def. 1.*

We give the intuition of the proof and refer the reader to Appendix C for details. First, the correctness of the reconstruction is easily verifiable. Further, each of the four pairs of shares, reconstructing their corresponding secret $s \in \{s_{00}, .., s_{11}\}$, has the following structure. Let $s = (t_1, ..., t_n)$. The second share in each pair of shares is a sequence of $n + 1$ randomly chosen blocks r_i from D: $sh_2 = (r_1, ..., r_{n+1})$. The first share in each pair is a sequence of n "blocks with pointers" $sh_1 = (B_1, ..., B_n)$, as follows. $\forall i \in \{1..n\}, B_i = \langle p_i, b_i \rangle$, where $p_1, ..., p_n$ is a random permutation of a random n-element subset of $\{1..n + 1\}$, and $b_i = t_i \oplus r_{p_i} \in D$. This implies the simulator $Sim(s)$, required by Def. 1.

GESS' performance. From above, if the secrets of the output wire of G consist of n blocks of size k, then the secrets of G's inputs consist of no more than $n+1$ blocks of size $k+\lceil \log(n+1) \rceil$. Similarly, d levels deeper, wires' secrets consist of no more than $n + d$ blocks of size $k + \sum_{i=1..d} \lceil \log(n + i) \rceil$. Therefore, starting with one-bit secrets ($n = 1, k = 1$), a tree circuit will have at depth d secrets of size at most $(d+1)(d \log(d+1)+1) = d^2 \log(d+1)+d \log(d+1)+d+1$. The shares grow very slowly: as $d \to \inf$, the "share expansion factor" — the ratio of sizes of shares to sizes of secrets of a GESS scheme for a gate G at depth d — approaches 1. Since the gates have exactly two inputs, there are at most 2^d input wires to the circuit, and the total size of Bob's secrets to be sent to Alice is $2^d(d^2 \log(d+1) + d \log(d+1) + d + 1) \approx 2^d d^2 \log d$, dominated by the 2^d term.

Rebalancing C prior to applying the above reduction may result in substantial performance improvement. Bonet and Buss [6] and Bshouty, Cleve and Eberly [7] prove the following fact (and exhibit the rebalancing procedure).

Let C be a $\{\vee, \wedge, \neg\}$-formula of leaf size m. Then for all $k \geq 2$, there is an equivalent $\{\vee, \wedge, \neg\}$-formula C', such that $\text{depth}(C') \leq (3k \ln 2) \cdot \log m$, and $\text{leafsize}(C') \leq m^{\alpha}$, where $\alpha = 1 + \frac{1}{1+\log(k-1)}$.

Consider a highly unbalanced C of size m. Direct application of our reduction costs $\Theta(m^3)$, more than BP based approaches [17,18,19] of cost $O(m^2)$. Rebalancing C as above, even suboptimally setting $k = 9$, results in a formula C' of size $m^{1.25}$ and depth $\approx 18.5 \log m$. Applying the reduction to C' yields a much better cost $O(m^{1.25} \log^2 m)$. An optimal (w.r.t. the cost of the GESS reduction) choice of k or better rebalancing will further improve our (but not BP's) performance.

2.5 Lower Bounds for GESS – The Optimality of Our Constructions

Let $i, j \in \{0,1\}$. Denote by A_i (resp. B_i) the random variable of the share corresponding to the wire value i of the first (resp. second) input wire. Denote by S_{ij} the random variable of the secret corresponding to the gate output value $G(i,j)$. Let $H(\cdot)$ be Shannon entropy. We start with proving a technical lemma.

Lemma 1. *For any GESS scheme implementing a gate with binary inputs,* $H(A_i)+H(B_j) \geq H(S_{i(1\text{-}j)}|B_{1\text{-}j})+H(S_{(1\text{-}i)j}|A_{1\text{-}i})+H(S_{ij}|S_{i(1\text{-}j)}S_{(1\text{-}i)j}S_{(1\text{-}i)(1\text{-}j)})$.

Proof. For simplicity, prove the lemma for $i = j = 0$, i.e that $H(A_0) + H(B_0) \geq H(S_{01}|B_1) + H(S_{10}|A_1) + H(S_{00}|S_{01}S_{10}S_{11})$. Other cases are analogous.

First, since $H(S_{01}|A_0B_1) = 0$, and using the chain rule twice, obtain $H(A_0|B_1) = H(A_0S_{01}|B_1) - H(S_{01}|A_0B_1) = H(A_0S_{01}|B_1) = H(S_{01}|B_1) + H(A_0|B_1S_{01})$. Similarly, $H(B_0|A_1) = H(S_{10}|A_1) + H(B_0|A_1S_{10})$.

By definition, A_1, B_1 do not reveal anything about S_{00} (other than what's implied by S_{11}), and, further, A_0, B_0 recover S_{00}. Then $H(S_{00}|S_{01}S_{10}S_{11}) \leq H(S_{00}|A_1B_1S_{01}S_{10}) \leq H(A_0B_0|A_1B_1S_{01}S_{10}) \leq H(A_0|A_1B_1S_{01}S_{10}) + H(B_0|A_1B_1S_{01}S_{10}) \leq H(A_0|B_1S_{01}) + H(B_0|A_1S_{10})$.

Thus, $H(A_0)+H(B_0) \geq H(A_0|B_1)+H(B_0|A_1) \geq H(S_{01}|B_1)+H(A_0|B_1S_{01})+H(S_{10}|A_1) + H(B_0|A_1S_{10}) \geq H(S_{01}|B_1) + H(S_{10}|A_1) + H(S_{00}|S_{01}S_{10}S_{11})$. □

Because all shares corresponding to the same wire must be distributed identically (Observation 2), their entropies must be equal. Thus Lemma 1 implies that $\forall i_1, i_2 \in \{0,1\} : H(A_{i_1}) + H(B_{i_2}) \geq MAX_{i,j\in\{0,1\}}(H(S_{i(1-j)}|B_{1-j}) + H(S_{(1-i)j}|A_{1-i}) + H(S_{ij}|S_{i(1-j)}S_{(1-i)j}S_{(1-i)(1-j)}))$.

Consider non-trivial gates – those that depend on both (binary) inputs. Note that the gate output need not be binary. We show the optimality of constructions for the natural case when the secrets are drawn independently at random from the same domain (with only the restrictions of secrets equality imposed by the semantics of G). In that case, by Observation 2, $H(S_{i(1-j)}|B_{1-j}) = H(S_{i(1-j)})$ and $H(S_{(1-i)j}|A_{1-i}) = H(S_{(1-i)j})$. Consider the two possible cases.

Case 1: there exist gate inputs i, j, s.t. $G(i,j)$ is not equal to the gate value on any other inputs. This is the case for most non-trivial gates (including AND and OR). In this case, $H(S_{ij}|S_{i(1-j)}S_{(1-i)j}S_{(1-i)(1-j)}) = H(S_{ij})$ and thus $\forall i_1, i_2 \in \{0,1\} : H(A_{i_1}) + H(B_{i_2}) \geq H(S_{i(1-j)}) + H(S_{(1-i)j}) + H(S_{ij})$. This matches (within 1 bit) the upper bound given by Constr. 2.

Case 2: such i, j don't exist. Then the only non-trivial gates are XOR and ¬ XOR. GESS of Constr. 4 implements XOR and matches the lower bound of $H(S_{i(1-j)}) + H(S_{(1-i)j})$ for this case.

Construction 4. *(GESS ensemble for XOR gates.) Let* $SEC = \{0,1\}^n$ *and* $TSEC = SEC^2$ *be the secrets domains. Let the secrets tuple* $\langle s_0, s_1\rangle \in TSEC$ *be given. The domains of shares are set as follows:* $SH_1 = SH_2 = SEC$.

Shr chooses $R \in_R SEC$ *and sets* $sh_{10} = R, sh_{11} = s_0 \oplus s_1 \oplus R, sh_{20} = s_0 \oplus R, sh_{21} = s_1 \oplus R$.

Rec proceeds as follows. On input sh_1, sh_2, *Rec outputs* $sh_1 \oplus sh_2$.

Theorem 4. *For each $n \in \mathbb{N}$, Constr. 4 is a GESS as defined by Def. 1.*

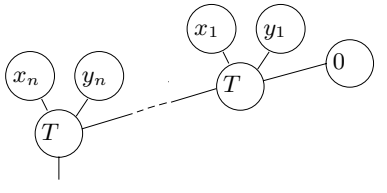
The proof of Thm. 4 is very simple and is omitted.
In conclusion, for the shares A_i and B_j of the two input wires, we proved

Theorem 5. *For every GESS scheme implementing an OR or an AND gate, when all secrets are chosen at random from the same domain SEC and each has entropy H_S, $\forall i, j \in \{0,1\}: \ H(A_i) + H(B_j) \geq 3H_S$.*

Of course, the entropy of each share must be at least H_S. Then all possible gates with two binary inputs are (almost) optimally implemented by either Constr. 2 or 4. Our Constr. 3 beats the above lower bound by exploiting common information among secrets. We leave open the question of exact lower bounds for this interesting case. We stress that the share-size-to-secret-size ratio approaching 1, achieved by Constr. 3, is "near optimal".

2.6 Application of GESS: Efficient Practical Two Millionaires

We apply the GESS approach to give a new efficient solution to the two millionaires problem. We design a GESS scheme for a new type of gate and use it to compute the Greater Than (GT) predicate. We use the intuitive circuit C (below) that compares bits of the parties' inputs x and y, starting with the most significant, and sets the answer bit when it encounters the difference.

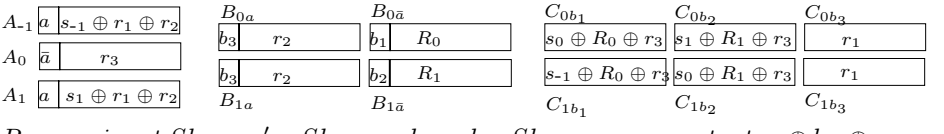


where $T(j, x_i, y_i) = \begin{cases} j, \text{if} j \in \{-1, 1\}, \\ -1, \text{if} j = 0 \wedge x_i < y_i, \\ 0, \text{if} j = 0 \wedge x_i = y_i, \\ 1, \text{if} j = 0 \wedge x_i > y_i. \end{cases}$

Here j is ternary input and x_i and y_i are bits. It is easy to see that C indeed computes GT: once a ternary wire is set to -1 or 1, that value is propagated to the output wire. We aim to minimize the expansion of the share corresponding to the input j. Note the double application of permute and point in Constr. 5.

Construction 5. *(GESS ensemble for T-gates.) Let $SEC = \{0,1\}^n$ and $TSEC = SEC^3$ be the secrets domains. Let the secrets tuple $\langle s_{-1}, s_0, s_1 \rangle \in TSEC$ is given. The domains of shares are set as follows: $SH_1 = \{0,1\} \times SEC$, $SH_2 = (\{0,1\}^2 \times SEC)^2$ and $SH_3 = SEC^3$.*

Shr chooses $R_0, R_1, r_1, r_2, r_3 \in_R SEC$, $a \in_R \{0,1\}$ and $b = \{b_1, b_2, b_3\}$ - a random permutation of $\{0,1,2\}$, where each b_i is suitably represented by 2 bits. Shr sets the shares $sh_{1i} = A_i, sh_{2i} = \langle B_{i0}, B_{i1} \rangle, sh_{3i} = \langle C_{i0}, C_{i1}, C_{i2} \rangle$, as shown on the following diagram.



Rec, on input $Sh_1 = a'r$, $Sh_2 = p_0\, b_0\, p_1\, b_1$, $Sh_3 = c_0 c_1 c_2$, outputs $r \oplus b_{a'} \oplus c_{p_{a'}}$.

Theorem 6. *For each $n \in \mathbb{N}$, Constr. 5 is a GESS as defined by Def. 1.*

Proof. (Sketch): Correctness of the scheme is easily verified. The simulator $Sim(s)$ chooses random $\alpha \in_R \{0,1\}, r'_0, ..., r'_4 \in_R SEC, \beta_0, \beta_1 \in_R \{0,1,2\}$, where $\beta_0 \neq \beta_1$. Let β'_i be suitable 2-bit representations of β_i. Sim outputs shares $\langle(\alpha r'_2), (\beta'_0 r'_0 \beta'_1 r'_1), (\gamma_0\gamma_1\gamma_2)\rangle$, where $\gamma_{\beta_\alpha} = s \oplus r'_2 \oplus r'_\alpha$, and the other two γ_i are assigned r'_3 and r'_4. The proof of equality of the generated distribution to the real execution is similar to that of previous two theorems, and is omitted. □

Performance. Let n be the length in bits of the compared numbers. The secrets corresponding to the T-gate at level i are of length i, and thus the secrets corresponding to the corresponding x_i and y_i are of lengths $3i$ and $2i+4$. Thus, Bob needs to send $\sum_{i=1..n} 3i = 1.5n(n+1)$ bits and perform n 1-out of-2 OT's with secrets of sizes $2+4, ..., 2n+4$.

The asymptotic complexity of this GT solution is worse than that of the best currently known for either setting with limited Alice (Yao's approach, see, e.g. [24]) or unlimited Alice [5,13]. Still, our solution performs better for comparing smaller numbers ($n \approx 60..70$), since we do not use encryption[3].

We note that a reduction with a complexity similar to ours (quadratic) can be obtained by using BP-based techniques of [19].

3 Extension to Evaluating Polysize Circuits

When Alice is assumed to be polynomially bounded, all polytime computable functions can be efficiently evaluated. Beaver, Micali and Rogaway [3,25], Naor, Pinkas and Sumner [24] and Lindell and Pinkas [21] suggested one-round protocols following Yao's [27] garbled circuit approach.

As discussed, the OT reduction does not allow polytime evaluation of general polysize circuits, due to the exponential growth of combined secrets size for each level of general circuits. We now informally describe a natural extension that handles this problem in the standard model. This demonstrates the generality and applicability of the GESS approach. The resulting solution is conceptually very clean, although slightly less efficient than the best known approach.

The protocol is essentially Constr. 1, with the following amendment. Bob will not propagate the secrets "up the circuit". Instead, for a gate G with output wires $w_1, ..., w_n$ and their (already computed) corresponding secrets tuples $(s^1_0, s^1_1), ..., (s^n_0, s^n_1)$, he encrypts all the secrets corresponding to each gate value *together*. More formally, he chooses two random keys k', k'' of a semantically secure private-key encryption scheme E. He computes $e_0 = E_{k'}(\langle s^1_0, ..., s^n_0\rangle), e_1 = E_{k''}(\langle s^1_1, ..., s^n_1\rangle)$ and assigns G's labels to be a random permutation of e_0, e_1. He then treats the keys as the secrets to be propagated, letting k' and k'' correspond to wire values 0 and 1 respectively. When Bob is done, he will have

[3] This advantage is minute with standard (public-key primitive based) OT implementations; it may be significant in other settings.

assigned secrets to each of the input wires and associated labels with each of the gates. He sends the secrets to Alice as before, additionally sending her the gate labels.

Alice obtains the secret shares for the input wires and proceeds evaluation similarly to the previous solution. The difference now is that, after having recovered a gate's secret (which is the key for one of the associated encryptions), she decrypts the corresponding encryption to recover the outgoing wires' secrets. To ensure that only one decryption succeeds, we impose an additional requirement on the encryption scheme. Informally, we need the ranges of encryptions under different keys be distinct, and that Alice is able to tell which decryption succeeded. This is a rather weak requirement, satisfied, for example, by schemes with *elusive* and *efficiently verifiable* ranges, formalized in [21]. Alice then uses the recovered secrets as shares in computing the child gate's secrets, and so on. Finally, she outputs the value of the output wire.

Theorem 7. *The above construction securely (against computationally unlimited Bob and limited Alice) reduces SFE of polysize circuits to OT, in the semi-honest model.*

The proof of the theorem is rather intuitive and is presented in Appendix E.

The performance of the resulting approach is very similar to that of the currently best known solutions (e.g. [21,24]). Indeed, our wire secrets are of the same size as theirs, and thus the only difference in performance is caused by the size of the gate labels. In [24], each gate has four labels of size N each[4], where N is the security parameter. It is easy to see that each gate of our solution adds up to $6N$ bits to the collection of all gate labels (two secrets of length N expand into two shares of length $N+1$ and two shares of length $2N$, which then are encrypted and stored as labels.). Some optimization of this number is also possible. For example, we need not encrypt (and thus add the corresponding labels) for the secrets that are just larger than N. This can reduce the gate induced label size gate by up to $2N$ bits.

We further note that in our scheme we only need to use encryptions once the secret sizes grow too large (i.e some threshold larger than encryption keys). Thus our method improves the performance of the evaluation of "the bottom part" of every circuit, and can be combined with Yao's garbled circuit implementations.

Acknowledgements. The author is very grateful to Ian F. Blake, Steven Myers, Berry Schoenmakers, and, most of all, to Charles Rackoff for many criticisms and insightful discussions both on the content and presentation of this work. Special thanks to Yuval Ishai for critically reading earlier versions of this paper and for pointing out a lot of related work and interesting applications. Finally, thanks to all referees of this paper for their comments and suggestions. This research was partially supported by Ontario Graduate Scholarship (OGS).

[4] The authors also mention an optimization that allows using only three labels.

References

1. Joy Algesheimer, Christian Cachin, Jan Camenisch, and Gunter Karjoth. Cryptographic security for mobile code. In *SP '01: Proceedings of the IEEE Symposium on Security and Privacy*, page 2. IEEE Computer Society, 2001.
2. D A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In *Proc. 18th ACM Symp. on Theory of Computing*, pages 1–5, New York, NY, USA, 1986. ACM Press.
3. D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proc. 22nd ACM Symp. on Theory of Computing*, pages 503–513, 1990.
4. Donald Beaver. Minimal-latency secure function evaluation. In *Proc. EUROCRYPT 2000*, pages 335–350. Springer, 2000. Lecture Notes in Computer Science, vol. 1807.
5. Ian F. Blake and Vladimir Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *Proc. ASIACRYPT 2004*, pages 515–529, 2004.
6. Maria Luisa Bonet and Samuel R. Buss. Size-depth tradeoff for boolean formulae. *Information Processing Letters*, 11(1994):151–155.
7. N. H. Bshouty, R. Cleve, and W. Eberly. Size-depth tradeoffs for algebraic formulae. In *Proc. 32nd IEEE Symp. on Foundations of Comp. Science*, pages 334–341. IEEE, 1991.
8. Christian Cachin, Jan Camenisch, Joe Kilian, and Joy Muller. One-round secure computation and secure autonomous mobile agents. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, 2000.
9. R. Cleve. Towards optimal simulations of formulas by bounded-width programs. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 271–277, New York, NY, USA, 1990. ACM Press.
10. Ronald Cramer, Serge Fehr, Yuval Ishai, and Eyal Kushilevitz. Efficient multiparty computation over rings. In *EUROCRYPT*, pages 596–613, 2003.
11. Uri Feige, Joe Killian, and Moni Naor. A minimal model for secure computation (extended abstract). In *Proc. 26th ACM Symp. on Theory of Computing*, pages 554–563. ACM Press, 1994.
12. Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.
13. Marc Fischlin. A cost-effective pay-per-multiplication comparison method for millionaires. In *RSA Security 2001 Cryptographer's Track*, pages 457–471. Springer-Verlag, 2001. Lecture Notes in Computer Science, vol. 2020.
14. Oliver Giel. Branching program size is almost linear in formula size. *J. Comput. Syst. Sci.*, 63(2):222–235, 2001.
15. Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press, 2004.
16. Oded Goldreich and Ronen Vainish. How to solve any protocol problem - an efficiency improvement. In *CRYPTO '87: A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, pages 73–86, London, UK, 1988. Springer-Verlag.
17. Yuval Ishai and Eyal Kushilevitz. Private simultaneous messages protocols with applications. In *ISTCS '97: Proceedings of the Fifth Israel Symposium on the Theory of Computing Systems (ISTCS '97)*, page 174, Washington, DC, USA, 1997. IEEE Computer Society.
18. Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Proc. 41th IEEE Symp. on Foundations of Comp. Science*, page 294. IEEE Computer Society, 2000.

19. Yuval Ishai and Eyal Kushilevitz. Perfect constant-round secure computation via perfect randomizing polynomials. In *ICALP*, pages 244–256, 2002.
20. J. Kilian. Founding cryptography on oblivious transfer. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 20–31, Chicago, 1988. ACM.
21. Yehuda Lindell and Benny Pinkas. A proof of Yao's protocol for secure two-party computation. Cryptology ePrint Archive, Report 2004/175, 2004. `http://eprint.iacr.org/`.
22. Moni Naor and Kobbi Nissim. Communication preserving protocols for secure function evaluation. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 590–599, New York, NY, USA, 2001. ACM Press.
23. Moni Naor and Benny Pinkas. Distributed oblivious transfer. In *Proc. ASIACRYPT 2000*, volume 1976, pages 200–219. Springer-Verlag, 2000. Lecture Notes in Computer Science, vol. 293.
24. Moni Naor, Benny Pinkas, and Reuben Sumner. Privacy preserving auctions and mechanism design. In *1st ACM Conf. on Electronic Commerce*, pages 129–139, 1999.
25. Phillip Rogaway. *The round complexity of secure protocols.* PhD thesis, MIT, 1991.
26. Tomas Sander, Adam Young, and Moti Yung. Non-interactive cryptocomputing for NC^1. In *Proceedings 40th IEEE Symposium on Foundations of Computer Science*, pages 554–566, New York, 1999. IEEE.
27. A. C. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE Symp. on Foundations of Comp. Science*, pages 162–167, Toronto, 1986. IEEE.

A The General Definition of GESS

We give a general definition of a GESS scheme that allows to share a tuple of secrets. Let G be a gate with k inputs from domain $D_I = D_{I_1} \times ... \times D_{I_k}$ and one output from domain D_O. We also denote by $G : D_I \mapsto D_O$ the function computed by gate G. Let SEC be the domain of secrets and $TSEC \subset SEC^{|D_O|}$ be the domain of tuples of secrets to be shared. For simplicity of presentation and without loss of generality, assume that all domains D_{I_i} and D_O are initial sequences of non-negative numbers, e.g. $D_{I_1} = \{0, 1, 2, ..., |D_{I_1}| - 1\}$.

Definition 2. *(Gate evaluation Secret Sharing) A* gate evaluation *secret sharing scheme (*GESS*) for evaluating G (we also say GESS* implementing *G) is a pair of algorithms (Shr, Rec) (with implicitly defined secrets domain SEC, secrets tuples domain $TSEC$, k share domains $SH_1, ..., SH_k$ and k share tuples domains $TSH_1, ..., TSH_k$), such that the following holds.*

The probabilistic share generation algorithm Shr takes as input a $d_O = |D_O|$-tuple of secrets
$\langle s_0, ..., s_{d_O-1} \rangle \in TSEC$ *and outputs a sequence of k* tuples *of shares, where the i-th tuple $t_i \in TSH_i$ consists of $|D_{I_i}|$ shares $sh_{ij} \in SH_i$. The deterministic share reconstruction algorithm Rec takes as input a sequence of k elements $sh_i \in SH_i$, one from each domain, and outputs $s \in SEC$.*

Let $b = \langle b_1, ..., b_k \rangle \in D_I$ be a selection vector. Define the selection function $Sel(\langle sh_{10}, ..., sh_{1|D_{I_1}|-1} \rangle, ..., \langle sh_{k0}, ..., sh_{k|D_{I_k}|-1} \rangle, b) = \{sh_{1b_1}, ..., sh_{kb_k}\}$.

Shr and Rec satisfy the following conditions:

- *correctness: for all random inputs of Shr and secrets tuples* $\langle s_0, ..., s_{d_O-1}\rangle \in TSEC$, $\forall b \in D_I, Rec(Sel(Shr(\langle s_0, ..., s_{d_O-1}\rangle), b)) = s_{G(b)}$
- *privacy (selected shares contain no information other than the value* $s_{G(b)}$*): There exists a simulator Sim, such that* $\forall\langle s_0, ..., s_{d_O-1}\rangle \in TSEC$ *and any* $b \in D_I$*:* $Sim(s_{G(b)}) \equiv Sel(Shr(\langle s_0, ..., s_{d_O-1}\rangle), b)$

B Proof of Theorem 1

Proof. (Sketch): Security against Bob is trivial since he does not receive any messages. The intuition for the scheme's security against Alice is that none of the GESS implementations leak any information. To prove security, we show how to construct Sim_A, perfectly simulating the following ensemble (view of Alice): $\mathrm{VIEW}_A(x, a) = \{x, m_{OT}, m\}$, where x and a are Alice's input and output, m_{OT} is the sequence of messages received from the OT oracles and m is the message received from Bob directly.

Sim_A first simulates wire secrets assignment as follows. He starts with the output wire, assigns its value to be a, and proceeds through gates from the bottom up as follows. Given gate G, its $GESS_G$, simulator Sim_G, and G's output wire value v, Sim_A assigns values to G's input wires according to $Sim_G(v)$.

Eventually, Sim_A assigns secrets to all input wires of C. Sim_A outputs $\{x, m'_{OT}, m'\}$, where x is Alice's input, m'_{OT} and m' are (proper representations of) the sequences of C's input wires assignments corresponding to Alice and to Bob respectively.

It is intuitive that the proposed simulator perfectly simulates Alice's view. Indeed, the vector of inputs to C defines a value assignment to each wire of the circuit, which, in turn, defines a distribution on shares/secrets obtained (received or computed) by Alice for each wire. We prove that wire assignment of Sim_A perfectly simulates the obtained secret for each wire. It is clear that Sim_A perfectly assigns the secret corresponding to the output wire by setting it to the output of the computation he obtained as its input. Further, Sim_A assigns secrets to the input wires of the output gate G. These secrets are distributed identically to the secrets that Alice reconstructs for these wires, because of the perfect simulation of Sim_G. Proceeding upward to the input wires, it is clear that Sim_A perfectly simulates all the wire assignments that Alice sees and reconstructs in the real execution. □

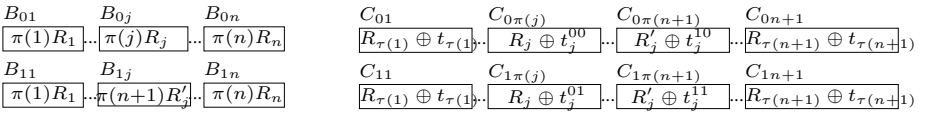
C The General Construction of GESS for AND/OR Gates

Construction 6. *(Improved GESS for gates with two binary inputs.) Let* $D = \{0,1\}^k$ *and* $SEC = D^n$*. Let secrets* $s_{00}, ..., s_{11} \in SEC$ *consist of* n *blocks of length* k*, and differ only in the* j*-th block. That is, let*

$$s_{00} = (\quad t_1 \quad \ldots \quad t_{j-1} \quad t_j^{00} \quad t_{j+1} \quad \ldots \quad t_n \quad),$$
$$\ldots$$
$$s_{11} = (\quad t_1 \quad \ldots \quad t_{j-1} \quad t_j^{11} \quad t_{j+1} \quad \ldots \quad t_n \quad),$$

where $\forall i = 1..n$*:* $t_i, t_j^{00}, t_j^{01}, t_j^{10}, t_j^{11} \in D$*, and the index* j *is determined only by the secrets. Let* $TSEC \subset SEC^4$ *be the space of all tuples of the above form.*

Shr chooses $R_1, ...R_n, R'_j \in_R D$ *and a random permutation*[5] $\pi : \{1..n+1\} \mapsto \{1..n+1\}$*. Let* $\tau = \pi^{-1}$ *be the inverse of* π*. For* $m \in \{0,1\}$*, Shr sets the shares* $sh_{1m} = \langle B_{m1}, ..., B_{mn} \rangle$ *and* $sh_{2m} = \langle C_{m1}, ..., C_{mn+1} \rangle$*, as shown on the following diagram.*



More specifically, the blocks of both shares of the first wire will be assigned $R_1, \ldots, R_n$*, with the exception of the* j^{th} *block of the share corresponding to* 1*, which will be assigned* R'_j*. Shr then, for all* i*, prepends* $\pi(i)$ *to the* i^{th} *block of both shares of the first wire, with the exception of the* j^{th} *block of the second share, which gets prepended* $\pi(n+1)$*.*

Each $\pi(i)$*-th block of both shares of the second wire will be set to* $R_i \oplus t_i$*, with the exception of blocks* $\pi(j), \pi(n+1)$*. Those blocks assignment is motivated by Construction 2. Specifically, we set the* $\pi(j)$*-th block of the share corresponding to* 0 *to* $R_j \oplus t_j^{00}$ *and that block of the share corresponding to* 1 *– to* $R_j \oplus t_j^{01}$*. We set the* $\pi(n+1)$*-st block of the share corresponding to* 0 *to* $R'_j \oplus t_j^{10}$ *and that block of the share corresponding to* 1 *– to* $R_j \oplus t_j^{11}$*. This completes the description of Shr.*

Rec proceeds as follows. He obtains two shares $sh_1 = (ind_1, r_1, ..., ind_n, r_n)$ *and* $sh_2 = (a_1, ..., a_{n+1})$*. He reconstructs the secret* $s = (\sigma_1, ..., \sigma_n)$ *by setting* $\sigma_i = r_i \oplus a_{ind_i}$*.*

Theorem 8. *For each* $n, k, j \in \mathbb{N}$*, Construction 6 is a GESS scheme as defined by Def. 1. (Note that security and correctness hold w.r.t.* $TSEC$*.)*

Proof. (Sketch): The correctness of the reconstruction is easily verifiable. To prove security, we construct a simulator $Sim(s)$. On input $s = \sigma_1, ..., \sigma_n$, $Sim(s)$ does the following. He chooses random $r'_1, ..., r'_{n+1} \in_R D$ and a random permutation $\rho : \{1..n+1\} \mapsto \{1..n+1\}$. He outputs the shares $sh_1 = (\rho(1)r'_1, \ldots, \rho(n)r'_n)$ and $sh_2 = (\sigma_{\rho^{-1}(1)} \oplus r'_{\rho^{-1}(1)}, \ldots, \sigma_{\rho^{-1}(n+1)} \oplus r'_{\rho^{-1}(n+1)})$

We now prove that Sim perfectly simulates the real-life generated shares. The first share is distributed identically to both of the real-life generated shares of the first vector. Indeed, each r_i is distributed identically to each R_i, R_j and R'_j and $\rho(1), ..., \rho(n)$ is distributed identically to $\pi(1), ..., \pi(n)$ and to $\pi(1), ..., \pi(j-1), \pi(n+1), \pi(j+1), ..., \pi(n)$, for any j.

As for the second share, all blocks (and their positions) are generated identically to the real execution, with the exception of blocks in positions $\rho(j)$ and $\rho(n+1)$. Proof of the equality of their distribution to the corresponding blocks of the real distribution closely follows that of Construction 2 and is omitted. □

[5] This permutation specifies which block of the second tuple is XOR'ed with the i^{th} block of the first tuple to obtain the i^{th} block of the reconstructed secret.

D Case Analysis for the Proof of Theorem 2

Proof. (Sketch): We need to consider the four possible combinations of gate input values $i_1, i_2 \in \{0,1\}$. We show that Sim perfectly simulates the corresponding truly generated shares. Denote random variables $\langle sh_1, sh_2\rangle = \langle b'r, a_0a_1\rangle = Sel(Shr(s_{00}, ..., s_{11}), \langle i_1, i_2\rangle)$. We write out only one case; others are analogous.

Case $i_1 = 0, i_2 = 0$**.** Thus $s = s_{G(0,0)}$.
Correctness: If $b = 0$, then $b' = 0, sh_1 = 0R_0, sh_2 = (s_{00} \oplus R_0, s_{10} \oplus R_1)$. $Rec(sh_1, sh_2) = R_0 \oplus (s_{00} \oplus R_0) = s_{00} = s$. If $b = 1$, then $b' = 1, sh_1 = 1R_0, sh_2 = (s_{10} \oplus R_1, s_{00} \oplus R_0)$. $Rec(sh_1, sh_2) = R_0 \oplus (s_{00} \oplus R_0) = s_{00} = s$.
Security: Clearly, $Sim(s)$ perfectly simulates sh_1. Further, sh_2 consists of two blocks $B_{00} = s \oplus R_0$ and $B_{10} = s_{10} \oplus R_1$. Observe that $B_{10} = s_{10} \oplus R_1$ is distributed uniformly randomly on SEC (since R_1 is random on SEC and secret). Therefore, sh_2 consists of two blocks from SEC, where one block is random on SEC and the other is equal to $s \oplus R_0$, where the non-random block is pointed by the bit b' of sh_1, Therefore $Sim(s)$ also perfectly simulates sh_2 and the pair $\langle sh_1, sh_2\rangle$, since d is distributed identically to b'. □

E Proof of Theorem 7

Proof. (Sketch): The reduction is trivially secure against Bob, since he does not receive any messages from Alice. To prove security against Alice, we will show how to simulate the input wires' secrets and gate labels that Bob sends to Alice, given the output of the computation. We present the proof for binary fan-in 2 circuits; a more general argument is readily obtained by natural generalization.

The simulator $Sim(x, b)$ proceeds as follows. First, it (perfectly) simulates the secret of the output wire by s.

Then, for each level of the circuit, starting from the bottom, for each gate G of the current level: given the (previously simulated) G's output wires' secrets $s_0, ..., s_{k-1}$, it simulates G's input wires' secrets and gate labels as follows. It chooses two random keys s', s'' from the key domain of the employed encryption scheme. Then it computes $e_0 = Enc_{s'}(\langle s_0, ..., s_{k-1}\rangle), e_1 = Enc_{s''}(\langle 0, ..., 0\rangle)$ and assigns G's labels to be a random permutation of e_0, e_1. Then Sim runs the the simulator $S_G(s')$ of the secret sharing scheme of G. The simulator $S_G(s')$ produces two shares (distributed identically to real execution), each of which is the simulation of the secret of the corresponding wire.

Sim runs the above procedure on C "from the bottom up", and eventually obtains the simulations of the input wires and gate labels, which he outputs, suitably formatted.

We note the true randomness of all encryption keys and the perfect simulations of secret sharing schemes. Intuitively, the only way for an adversary to distinguish the simulation from the real execution is by distinguishing the sets of non-decrypted gate labels. However, learning anything "substantial" that way would mean breaking the semantic security of the employed encryption scheme, which can be shown by a simple hybrid argument. □