# Combining Secret Sharing and Garbled Circuits for Efficient Private IEEE 754 Floating-Point Computations

Pille Pullonen[1,2] and Sander Siim[1,2]

[1] Cybernetica AS
[2] University of Tartu
{pille.pullonen, sander.siim}@cyber.ee

**Abstract.** Two of the major branches in secure multi-party computation research are secret sharing and garbled circuits. This work succeeds in combining these to enable seamlessly switching to the technique more efficient for the required functionality. As an example, we add garbled circuits based IEEE 754 floating-point numbers to a secret sharing environment achieving very high efficiency and the first, to our knowledge, fully IEEE 754 compliant secure floating-point implementation.

## 1 Introduction

Secure multi-party computation (MPC) enables parties to securely compute some function on their secret inputs and receive the secret outputs, without leaking anything to other parties. The fastest MPC protocols for integer arithmetic, like Sharemind [11][8] and SPDZ [13], rely on additive secret sharing. Additive sharing supports efficient addition and multiplication due to the algebraic properties of the scheme. However, floating-point arithmetic is much more sophisticated and contains a composition of different operations, both integer arithmetic as well as bitwise operations. Existing implementations based on secret sharing provide only near approximations to the IEEE 754 standard [1][20][27]. Although [14] proposes IEEE 754 protocols, no implementation is provided.

Another MPC approach is based on the garbled circuits method (GC) attributed to Yao [35] and detailed in [26]. A good overview of the recent advances can be found in [6][4], especially in the full versions. The baseline method is applicable to the two-party setting, however it can be extended to the case with more parties [7]. State-of-the-art garbling methods are already very efficient [4] and, in addition, means to derive optimized circuits from existing programs have been developed [17][23]. This allows secure protocols for arbitrary computations to be built with small effort using a general GC approach. However, in many cases the obtained protocols are less efficient than their secret-sharing-based alternatives. In practice, it would be useful to choose the more efficient technique, either sharing or GC based, for each particular subprotocol, but this requires interleaving secret sharing and garbling based methods in one computation.

In this paper, we present a *hybrid protocol*, which enables arbitrary secure computations through a combination of GC and secret sharing protocols[1]. In our protocol, GC gives the power to do bit-level operations in a compact manner, whereas secret sharing complements the construction with a fast oblivious transfer as well as composability with other secret-sharing-based protocols. Thereby, large and complex algorithms can be implemented by composition of the most efficient basic primitives. We illustrate the benefits by extending the Sharemind MPC framework [11][8] with, to our knowledge, the first secure floating-point protocol suite fully conforming to the IEEE 754 standard.

## 2 Preliminaries

In MPC, parties $\mathcal{P}_1, \ldots, \mathcal{P}_m$ want to securely compute a function $f$ on secret inputs $x_1, \ldots, x_m$ to learn $f(x_1, \ldots, x_m) = (y_1, \ldots, y_m)$, without leaking anything about inputs $x_i$ to other parties. Secret sharing

[1] Part of this work has been previously published in a Cybernetica technical report [33]

This research was, in part, funded by the U.S. Government. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. This work has also received funding from the Estonian Research Council through grant IUT27-1, and ERDF through EXCS.

is a mechanism of distributing data between participants without giving any of them direct access to the data, but enabling computations [32]. We denote a secret-shared vector $\mathbf{x}$ of $n$ elements shared between parties $\mathcal{P}_1,\ldots,\mathcal{P}_m$ by $[\![\mathbf{x}]\!] = [\![x_1, x_2, \ldots, x_n]\!] = [\![x_1]\!], \ldots, [\![x_n]\!]$ where party $\mathcal{P}_i$ holds $[\![\mathbf{x}]\!]_i = [\![x_1, x_2, \ldots, x_n]\!]_i$. An ordered tuple of regular non-secret-shared values $a_1, \ldots, a_n$ is denoted with $(a_1, \ldots, a_n)$. We focus on the additive secret sharing scheme, where sharing is defined with $\sum_{i=1}^{m} [\![x]\!]_i = x$. However, other schemes may also be used to implement our proposed protocol. We minimally require that the sharing scheme supports efficient integer addition and multiplication which remain secure in the presence of one passively corrupted party.

In the garbled circuits protocol [35][26], two parties called garbler and evaluator securely compute a known function $f(x, y)$ on their joint inputs. Essentially, the garbler encrypts a Boolean circuit of $g = f(a, \cdot)$ and sends the *garbled* truth tables of gates to the evaluator. The evaluator then uses oblivious transfer to obtain the keys corresponding to its input to decrypt the garbled circuit and evaluate $g(b)$. We refer to these keys as *tokens*. The garbler generates two uniformly random tokens $X^0, X^1$ for all the circuit's wires, including input and output wires, one corresponding to a 0-bit and the other to a 1-bit. The evaluation of the garbled circuit is completely oblivious, as the evaluator does not learn the circuit's output nor any intermediary results, only random tokens, which correspond to the actual values. The garbler can later deduce the actual output of the computation from the output tokens received from the garbled circuit evaluation. The output tokens only reveal the circuit output and no information about the evaluator's input. Thus, the inputs $a$ and $b$ are not revealed to the other party. In practice, symmetric cryptographic primitives such as a secure block cipher or hash function are used for garbling the circuit.

We use the notation from [4] to describe a circuit $f$ as a tuple $(n, m, q, A, B, G)$, where $n$, $m$ and $q$ respectively denote the number of external input wires, external output wires and gates in $f$. All wires are labeled by indexes. Namely, 1 to $n$ are input wires, $n + 1$ to $n + q$ mark gate output wires and $n + q - m + 1$ to $n + q$ are circuit outputs. Functions $A$ and $B$, respectively, identify the first and second input wire of any gate. The function $G$ determines the functionality of each gate, especially for a gate $g \in Gates$, the function $G(g) : \{0,1\}^2 \to \{0,1\}$ denotes the functionality of $g$. We use $X_j^b \in \{0,1\}^k$ to denote the token of the $j$-th wire corresponding to bit $b \in \{0,1\}$, where $k$ is the length of the generated tokens. We say $X_j^b$ has the *semantics* of $b$ and *type* $\mathtt{lsb}(X_j^b)$.

Here we only emphasize the important aspects of the used security proof framework, for details we refer to [9] or Appendix A. A protocol is said to be *input private* if, for any collection of allowed corrupted parties, there exists a simulator that can simulate the view of the adversary based on the inputs of corrupted parties. The *ordered composition* of an input private and a secure protocol, where all outputs are provided by the secure protocol, is secure if it is *output predictable*. The latter means that the composed protocols are correct and the final protocol does not leak information about its input shares to ensure the privacy of the first part.

Garbled circuits have two important security definitions: privacy and obliviousness [6]. Respectively, we consider prv.ind, prv.sim and obv.ind, obv.sim for either indistinguishability or simulation based versions of these definitions. Both properties are formalized via the *side-information function* $\Phi$ that captures the information that is revealed by the garbled circuit. We consider $\Phi_{topo}$ and $\Phi_{xor}$ that leak the topology and XOR operations. These functions are both efficiently invertible [4]. Therefore, by equivalence relations from [6], indistinguishability and simulation-based definitions coincide for both privacy and obliviousness. For a longer discussion of these properties see Appendix B.

## 3 Combining Garbled Circuits with Secret Sharing

Our goal is to construct an efficient protocol for securely evaluating Boolean circuits on bitwise secret-shared input, thereby allowing secret sharing protocols to be composed with computations more suitable for GC. Thus, each subprotocol can use the method more suitable for the given functionality and inputs, which can ultimately increase the efficiency of a larger computation. A similar approach is also used in the TASTY framework that combines GC and additively homomorphic encryption in a two-party setting [16]. As in our construction, TASTY relies on secure protocols to convert between different representations of the secret values in order to combine secure computation methods. They use two-party protocols for conversion between GC tokens and encrypted values [21], whereas our construction is based on a multi-party setting with secret sharing instead of homomorphic encryption.

The idea of our protocol is to set up GC to accept secret-shared inputs and to produce shared outputs. Thus, our protocol in Alg. 1 consists of three important steps. First, we require an efficient

**Algorithm 1:** Hybrid protocol for processing bitwise secret-shared data with a garbled circuit

**Input**: Shared bit vector $[\![\mathbf{x}]\!] = [\![x_1, \ldots, x_n]\!]$
Boolean circuit $f$ that calculates $f : \{0,1\}^n \to \{0,1\}^m$
**Output**: Shared bit vector $[\![\mathbf{y}]\!] = [\![y_1, \ldots, y_m]\!]$ such that $[\![\mathbf{y}]\!] = f([\![\mathbf{x}]\!])$

1 **foreach** input wire $i \in \{1, \ldots, n\}$ **do**
2 $\quad \mathcal{CP}_1$ generates a token pair $(X_i^0, X_i^1) \in \{0,1\}^k \times \{0,1\}^k$
3 The computing parties initiate an OT protocol which results in $\mathcal{CP}_2$ receiving $\mathbf{X} = (X_1^{x_1}, \ldots, X_n^{x_n})$ (the input tokens corresponding to the actual input bits)
4 $\mathcal{CP}_1$ garbles circuit $f$ and sends the garbled truth tables to $\mathcal{CP}_2$
5 $\mathcal{CP}_2$ evaluates garbled $f$ using $\mathbf{X}$ to get output tokens $\mathbf{Y} = (X_{n+q-m+1}^{y_1}, \ldots, X_{n+q}^{y_m})$
6 The garbled output is converted to secret-shared form to receive $[\![\mathbf{y}]\!]$
7 **return** $[\![\mathbf{y}]\!]$

garbling scheme to provide secure evaluation of circuits. Suppose we have $\mathcal{CP}_1, \ldots, \mathcal{CP}_m$ who hold some secret-shared data. We will let computing parties $\mathcal{CP}_1$ and $\mathcal{CP}_2$ respectively perform the computations of garbler and evaluator from the GC protocol. Note that independently of the properties of the secret sharing scheme we require that $\mathcal{CP}_1$ and $\mathcal{CP}_2$ are not colluding. Second, the GC protocol requires an oblivious transfer (OT) to provide the input tokens to the evaluator based on the secret-shared inputs. Third, we must convert the garbled outputs to the appropriate secret-shared form.

### 3.1 An Implementation of the Hybrid Protocol

Generally, the hybrid protocol can be implemented using various secret sharing and garbling schemes, provided their composition retains the necessary security properties (see Sec. 3.2 for details). However, we will focus on our instantiation built into the Sharemind MPC platform [8]. We chose Sharemind because it already provides an optimized multi-party computation environment based on secret sharing, which could easily be extended with our GC based protocol.

Our protocol extends Sharemind's `additive3pp` protection domain, which implements various secure computation protocols using a 3-out-of-3 additive secret sharing scheme [10]. This allows us to easily compose the hybrid protocol with the fast existing primitives for integer arithmetic. Note that since we are using the two-party Yao protocol, we can achieve security against at most one corrupted party. However, this fits well with Sharemind's `additive3pp` security model, which also provides security against a single passively corrupted party and is thus the optimal secret sharing environment to use in this case.

To securely evaluate garbled circuits in the presence of colluding parties, protocols from [3][7] may be used. However, it is unclear if there is anything to be gained performance-wise from extending this with our approach, since secret sharing is already used as a sub-primitive in the multi-party garbled circuit evaluation. Also, for security against two colluding parties, we would require at least 5 computing parties [7], which considerably increases the complexity of the secure computation environment. Consequently, we fix a setting with three computing parties $\mathcal{CP}_1$, $\mathcal{CP}_2$ and $\mathcal{CP}_3$ and additive secret sharing for the rest of this paper.

Note that different data types provided by Sharemind can be efficiently converted to shared bit vectors required in our construction using the bit extraction protocol from [11], which allows us to compose all `additive3pp` protocols with the hybrid protocol.

**Oblivious Transfer** The garbler $\mathcal{CP}_1$ generates a pair $(X_i^0, X_i^1)$ of tokens for every bit $x_i$ in the beginning of the garbling process. We need to transfer the tokens that correspond to the protocol inputs to the evaluator $\mathcal{CP}_2$. Clearly, if we have a subprotocol that calculates the necessary secret-shared tokens $[\![X_1^{x_1}, \ldots, X_n^{x_n}]\!]$, then we can complete the transfer by sending all result shares to $\mathcal{CP}_2$. This subprotocol can be easily implemented using secret-sharing-based multiplication and addition protocols. The resulting `OT` protocol is given in Alg. 2. An OT protocol is secure if the secret choice of the receiver remains private and the receiver is only able to learn one of the two messages of the sender. In addition to basic OT security properties, we also require that the choice bits $x_i$ are not leaked to any of the participants, including the receiver.

**Algorithm 2:** Oblivious transfer of input tokens (OT)

**Input**: $\mathcal{CP}_1$ holds the input tokens $(X_1^0, \ldots, X_n^0, X_1^1, \ldots, X_n^1)$
The input bit vector $[\![\mathbf{x}]\!] = [\![x_1, \ldots, x_n]\!]$ is shared between all parties
**Output**: $\mathcal{CP}_2$ receives input tokens $(X_1^{x_1}, \ldots, X_n^{x_n})$

1 $[\![\mathbf{X^0}]\!] = [\![X_1^0, \ldots, X_n^0]\!]$ and $[\![\mathbf{X^1}]\!] = [\![X_1^1, \ldots, X_n^1]\!]$ are instantiated as shared values, with shares of $\mathcal{CP}_2$ and $\mathcal{CP}_3$ initialized to 0
2 $[\![\mathbf{X}]\!] \leftarrow [\![\mathbf{X^0}]\!] \cdot ([\![\mathbf{1}]\!] - [\![\mathbf{x}]\!]) + [\![\mathbf{X^1}]\!] \cdot [\![\mathbf{x}]\!]$
3 $\mathcal{CP}_1$ and $\mathcal{CP}_3$ send their shares of $[\![\mathbf{X}]\!]$ to $\mathcal{CP}_2$
4 $\mathcal{CP}_2$ combines the shares of $[\![\mathbf{X}]\!]$ to get $(X_1^{x_1}, \ldots, X_n^{x_n})$
5 **return** $(X_1^{x_1}, \ldots, X_n^{x_n})$

The computations on line 2 of Alg. 2 are performed using the secure and input private multiplication and addition protocols from [11]. As a result, $[\![\mathbf{X}]\!] = [\![X_1^{x_1}, \ldots, X_n^{x_n}]\!]$ and the inputs $[\![\mathbf{x}]\!]$ remain private. Essentially, we perform an *oblivious choice* on secret-shared values, similarly to [25]. Note that each $x_i$ can easily be extended to the length of the tokens by creating a bit string consisting of $k$ copies of $x_i$. Therefore, the operations are performed in $\mathbb{Z}_{2^k}$. On line 3, the shares of $[\![\mathbf{X}]\!]$ are sent to $\mathcal{CP}_2$ who combines them to receive the input tokens.

**Garbling** The emphasis for achieving efficient GC is on reducing network communication, as this is ultimately the bottleneck for all GC protocols. Recent garbling schemes already bring the cost of local computations to a minimum as demonstrated in [4]. Due to these considerations, we chose the GaXR scheme with the A4 *dual-key cipher* instantiation from [4] for our protocol, which is one of the fastest to date and results in the smallest amount of network communication among alternatives proposed in [4].

The dual-key-cipher (DKC) is a formalization of the underlying encryption primitive of the garbling process [6]. A DKC is a deterministic function $\mathbb{E} : \{0,1\}^k \times \{0,1\}^k \times \{0,1\}^\tau \times \{0,1\}^k \to \{0,1\}^k$. It takes secret wire tokens $A$ and $B$ and a tweak $T$ to encrypt a wire token $X$. The function $\mathbb{E}$ for the A4 DKC instantiation is defined as

$$\mathbb{E}(A, B, T, X) = \pi(K \parallel T)_{[1:k]} \oplus K \oplus X$$

where $K = 2A \oplus 4B$. Here $\pi(K \parallel T)_{[1:k]}$ denotes the first $k$ bits of the result. It is easy to see that the decryption function $\mathbb{D}$ is completely symmetric to $\mathbb{E}$.

The function $\pi : \{0,1\}^{k+\tau} \to \{0,1\}^{k+\tau}$ denotes a random permutation, as the security of GaXR is shown in the *random permutation model*. We use fixed-key AES-128 with $k = 80$ and $\tau = 48$ to instantiate $\pi$, which provides reasonable security guarantees for this garbling scheme [4]. Tweak $T$ is the index of the garbled gate encoded as a $\tau$-bit integer. For the doubling function denoted by $2A$ we use multiplication with element $x$ over finite field $GF(2^k)$, as it provides the best security guarantees over other possible alternatives [4]. Note that here $k$ also corresponds to bit-length of the wire tokens. Our implementation uses the irreducible polynomial $x^{80} + x^9 + x^4 + x^2 + 1$ from [31] for defining the finite field.

The GaXR scheme incorporates *free-XOR* [22] and *garbled row reduction* [30] optimizations, both of which significantly reduce network communication of GC. Also, the A4 DKC instantiation allows us to use a smaller token size (80 bits as opposed to 128 bits for example), which drastically scales down the size of the resulting garbled circuit. Naturally one could use other garbling schemes for instantiating the hybrid protocol. However, for security we require that the scheme retains the obliviousness property [6] (see also Section 3.2 about security of hybrid protocol).

Fig. 1 summarizes the hybrid protocol. The garbler $\mathcal{CP}_1$ first generates a token pair $(X_i^0, X_i^1)$ for each input wire, with $X_i^0$ and $X_i^1$ having the semantics of 0 and 1 respectively. Then all three computing parties synchronously execute the OT protocol in Alg 2. As a result $\mathcal{CP}_2$ receives the correct input tokens needed for evaluation. Next, $\mathcal{CP}_1$ garbles the circuit according to the GaXR scheme and sends the garbled truth tables $P$ to $\mathcal{CP}_2$. The evaluator $\mathcal{CP}_2$ can then evaluate the garbled circuit using the transferred input tokens to receive the garbled output.

As an implementation detail, we have parallelized our protocol on two levels. First, the garbled tables are streamed by fixed-size batches from garbler to evaluator, similarly to [18]. The evaluator can then start evaluating the circuit while the garbler encrypts the next batch. This is especially relevant

performance-wise for large circuits. The batch size can be fixed for different circuits separately and fine-tuned to match the Sharemind instance's network and hardware capabilities.

In addition, our implementation allows both garbler and evaluator to run several threads to evaluate the same circuit with different inputs simultaneously. This can be thought of as using a number of garbler-evaluator pairs, similarly to the *cut-and-choose* implementation of [24] for actively secure GC. Besides parallel garbling, this allows a joint `OT` to be done for all the scheduled evaluations. This parallelization greatly reduces the cost of a single circuit evaluation when evaluating the circuit on a vector of inputs.

**Resharing** The final step in the protocol is resharing the output between all three computing parties using perfectly secure `Reshare` protocol Alg. 1 from [8]. This protocol rerandomizes the output shares held by $\mathcal{CP}_1$ and $\mathcal{CP}_2$ as $\mathbf{y} = [\![\mathbf{y}]\!]'_1 + [\![\mathbf{y}]\!]'_2$ to a uniformly secret-shared output $[\![\mathbf{y}]\!]$ and ensures that we can securely compose our protocol with all `additive3pp` protocols, which is vital for efficient computations that would benefit from both GC and secret sharing.

**Algorithm 3:** Hybrid protocol algorithm for $\mathcal{CP}_1$

**Input**: $[\![\mathbf{x}]\!]_1 = [\![x_1, \ldots, x_n]\!]_1$ and circuit $f = (n, m, q, A, B, G)$
**Output**: $[\![\mathbf{y}]\!]_1 = [\![y_1, \ldots, y_m]\!]_1$ such that $[\![\mathbf{y}]\!] = f([\![\mathbf{x}]\!])$

$R \xleftarrow{\$} \{0,1\}^{k-1} \parallel 1$
**for** $i \leftarrow 1$ **to** $n$ **do**
  $t \xleftarrow{\$} \{0,1\}$
  $X_i^0 \xleftarrow{\$} \{0,1\}^{k-1} \parallel t, \quad X_i^1 \leftarrow X_i^0 \oplus R$
`OT` $((X_1^0, \ldots, X_n^0), (X_1^1, \ldots, X_n^1), [\![\mathbf{x}]\!]_1)$
**for** $g \leftarrow n+1$ **to** $n+q$ **do**
  $a \leftarrow A(g), \quad b \leftarrow B(g)$
  **if** $G(g) = XOR$ **then**
    $X_g^0 \leftarrow X_a^0 \oplus X_b^0, \quad X_g^1 \leftarrow X_g^0 \oplus R$
  **else**
    **for** $i \leftarrow 0$ **to** $1, j \leftarrow 0$ **to** $1$ **do**
      $u \leftarrow i \oplus \mathtt{lsb}(X_a^0)$
      $v \leftarrow j \oplus \mathtt{lsb}(X_b^0)$
      $r \leftarrow G(g, u, v)$
      **if** $i = 0$ **and** $j = 0$ **then**
        $X_g^r \leftarrow \mathbb{E}(X_a^u, X_b^v, g, 0^k)$
        $X_g^{r-1} \leftarrow X_g^r \oplus R$
      **else**
        $P[g, i, j] \leftarrow \mathbb{E}(X_a^u, X_b^v, g, X_g^r)$
Send $P$ to $\mathcal{CP}_2$
**for** $i \leftarrow 1$ **to** $m$ **do**
  $y'_{i1} \leftarrow \mathtt{lsb}(X^0_{n+q-m+i})$
$[\![\mathbf{y}]\!]'_1 \leftarrow (y'_{11}, \ldots, y'_{m1})$
$[\![y_1, \ldots, y_m]\!]_1 \leftarrow$ `Reshare`$([\![\mathbf{y}]\!]'_1)$
**return** $[\![\mathbf{y}]\!]_1 = [\![y_1, \ldots, y_m]\!]_1$

**Algorithm 4:** Hybrid protocol algorithm for $\mathcal{CP}_2$

**Input**: $[\![\mathbf{x}]\!]_2 = [\![x_1, \ldots, x_n]\!]_2$ and circuit $f = (n, m, q, A, B, G)$
**Output**: $[\![\mathbf{y}]\!]_2 = [\![y_1, \ldots, y_m]\!]_2$ such that $[\![\mathbf{y}]\!] = f([\![\mathbf{x}]\!])$

$(X_1, \ldots, X_n) \leftarrow$ `OT` $(0^{k \cdot n}, 0^{k \cdot n}, [\![\mathbf{x}]\!]_2)$
Receive $P$ from $\mathcal{CP}_1$
**for** $g \leftarrow n+1$ **to** $n+q$ **do**
  $a \leftarrow A(g), \quad b \leftarrow B(g)$
  $i \leftarrow \mathtt{lsb}(X_a), \quad j \leftarrow \mathtt{lsb}(X_b)$
  **if** $G(g) = XOR$ **then**
    $X_g \leftarrow X_a \oplus X_b$
  **else if** $i = 0$ **and** $j = 0$ **then**
    $X_g \leftarrow \mathbb{E}(X_a, X_b, g, 0^k)$
  **else**
    $X_g \leftarrow \mathbb{D}(X_a, X_b, g, P[g, i, j])$
**for** $i \leftarrow 1$ **to** $m$ **do**
  $y'_{i2} \leftarrow \mathtt{lsb}(X_{n+q-m+i})$
$[\![\mathbf{y}]\!]'_2 \leftarrow (y'_{12}, \ldots, y'_{m2})$
$[\![y_1, \ldots, y_m]\!]_2 \leftarrow$ `Reshare`$([\![\mathbf{y}]\!]'_2)$
**return** $[\![\mathbf{y}]\!]_2 = [\![y_1, \ldots, y_m]\!]_2$

**Algorithm 5:** Hybrid protocol algorithm for $\mathcal{CP}_3$

**Input**: $[\![\mathbf{x}]\!]_3 = [\![x_1, \ldots, x_n]\!]_3$
**Output**: $[\![\mathbf{y}]\!]_3 = [\![y_1, \ldots, y_m]\!]_3$ such that $[\![\mathbf{y}]\!] = f([\![\mathbf{x}]\!])$

`OT` $(0^{k \cdot n}, 0^{k \cdot n}, [\![\mathbf{x}]\!]_3)$
$[\![\mathbf{y}]\!]'_3 \leftarrow 0^m$
$[\![y_1, \ldots, y_m]\!]_3 \leftarrow$ `Reshare`$([\![\mathbf{y}]\!]'_3)$
**return** $[\![\mathbf{y}]\!]_3 = [\![y_1, \ldots, y_m]\!]_3$

**Fig. 1.** Detailed algorithms of the hybrid protocol for all computing parties.

### 3.2 Security of the Hybrid Protocol

Our main security proof for the hybrid protocol is based on the proof scheme of [9]. Using this framework, we need to prove that the protocol up until the `Reshare` function is passively input private and then apply the composition result from [9]. For this, we need to establish the output predictability of the

composition. We denote the part of the hybrid protocol on Fig. 1 before final `Reshare` protocol as `Hybrid'`. This section gives an overview of the important building blocks of the proof, a full proof can be found in Appendix C. Especially, this section gives a general overview of combining the input privacy based view of [9] and the properties of the garbled circuits [6] and some intuition about the security of such construction. We consider only passive corruption and static adversaries that can corrupt at most one of the participants.

The description of basic Sharemind protocols as well as the proofs of their input privacy or security can be found in [11],[8]. They also use passive static adversaries that corrupt at most one of the three parties. In the following, we require privacy of addition and multiplication operations as well as the security of the resharing and multiplication protocols.

Output predictability is a notion that allows to capture the correctness of the composition of private and secure system. This is necessary as the privacy definition does not ensure that the protocol computes correct outputs. In addition, if we compose input private and secure protocols then we must ensure that the outputs of the private protocol are hidden. This is also captured by the notion of joint output predictability. This is defined for `Hybrid'` and the ideal functionality corresponding to `Reshare`. This ideal functionality of `Reshare` simply outputs uniformly random sharing of the input value.

**Theorem 1.** *`Hybrid'` protocol is correct.*

*Proof (Proof idea).* The correctness follows from the correctness of the sub-protocols used in the `OT` part and the correctness of the GAXR garbling scheme.

**Corollary 1.** *The ordered composition of `Hybrid'` and `Reshare` is jointly output predictable.*

*Proof (Proof idea).* The output predictability follows from Lemma 2 in [9] and Thm. 1.

Note, that the obliviousness of the garbling scheme [6] is quite like the input privacy [9] and is necessary for the input privacy of the `Hybrid'` protocol. However, only prv.sim security of GAXR is shown in the full version [5] of [4]. Hence, we also need to establish the obliviousness of the garbling scheme used in the hybrid protocol.

**Theorem 2.** GAXR *scheme is computationally obv.ind secure.*

*Proof (Proof idea).* The types of the input wires are independent of the semantics as they are generated independently on line 3 of Alg. 3 by $\mathcal{CP}_1$. In short, the obv.ind security follows from the fact that the keys of the outputs are generated the same way as the intermediate keys. Therefore, if the obliviousness property would be invalidated then we could build an adversary that adds some computations to the circuits that invalidate obliviousness to also break the privacy property that is proved in [5].

In the following we need to provide a privacy simulator for the `Hybrid'` protocol. However, for that we also need to have a simulator for the obliviousness property of the garbling scheme. Hence, besides obv.ind security we also require obv.sim security.

**Corollary 2.** GAXR *is computationally obv.sim secure.*

*Proof (Proof idea).* It follows directly from Thm. 2 as the indistinguishability and simulatability based definitions are equivalent.

The main security property required for using [9] is the input privacy. For our security proof we need to establish that `Hybrid'` protocol is input private.

**Theorem 3.** *Protocol `Hybrid'` is perfectly input private for passively corrupted $\mathcal{CP}_1$ or $\mathcal{CP}_3$ and computationally input private against passively corrupted $\mathcal{CP}_2$.*

*Proof (Proof idea).* We have to show the existence of the privacy simulator that can simulate the view of the corrupted party based on its inputs.

*Corrupted $\mathcal{CP}_1$ or $\mathcal{CP}_3$.* The perfect input privacy of these parties is ensured by the perfect input privacy of the addition and multiplication protocol and the composability of input privacy (Thm. 3 in [9]).

*Corrupted* $\mathcal{CP}_2$. From obv.sim security in Cor. 2 we know that there exists a simulator $\mathcal{S}$ such that it outputs garblings indistinguishable from those output by the real garbler. The privacy simulator $\mathcal{P}$ for $\mathcal{CP}_2$ can be built from the simulator $\mathcal{S}$. This $\mathcal{P}$ knows the circuit $f$ and can run $\mathcal{S}$ to obtain the simulated garbling. Next, it has to simulate the `OT` that can be done perfectly by using the privacy simulator for `OT`.

Using all the previous results we are ready to establish the fact that the proposed hybrid protocol is a secure multi-party computation protocol that computes the function $f$. Especially, we require the main result of [9] which states that the composition of a private and a secure system is secure if all outputs come only from the secure system and the built composition is output predictable.

**Corollary 3.** *Hybrid protocol algorithm is perfectly secure against passively corrupted* $\mathcal{CP}_1$ *and* $\mathcal{CP}_3$ *and computationally secure against passively corrupted* $\mathcal{CP}_2$.

*Proof (Proof idea).* From Cor. 1 we know that the composition is jointly output predictable. Thm. 3 also showed that `Hybrid'` being the first part of the ordered composition is input private. Therefore, we can use the composition result in Thm. 2 in [9] to conclude that the full hybrid protocol is secure.

In general, an analogous secure construction can be obtained from any oblivious correct garbling scheme, a secure protocol to convert garbled outputs to shares, and a sharing-based secure oblivious transfer that is input private for all parties except the evaluator. For example, any sharing scheme could be used as long as the additional requirement that the evaluator and garbler are not corrupted together holds and there exists a protocol to convert the garbled outputs to a shared bit vector.

Note that following the ideas of [9] it would also be meaningful to combine `Hybrid'` directly with other input private secure computation protocols and only perform the final `Reshare` step after all desired computations are finished. At first we would then combine the private protocols into the composed private protocol according to the composability of input privacy. The final `Reshare` step can be added analogously to Cor. 3 to make the composed private protocol secure.

## 4 Using the Hybrid Protocol for Efficient Computations

Sharemind's `additive3pp` protocols enable fast integer operations. On the other hand, bit-level operations are more costly. However, in practical applications we are also interested in more complex primitives that rely heavily on different non-linear operations. A very relevant example of this is floating-point computations.

The *de facto* standard today for binary floating-point arithmetic is the IEEE Standard for Floating-Point Arithmetic (IEEE 754) [19]. Although existing secure implementations of floating-point operations resemble IEEE 754 [1][20][27], they do not always produce identical results compared to regular hardware implementations. The main shortcomings are in not rounding inexact results to nearest representable floating-point numbers, lack of support for gradual underflow and missing error handling [15]. The previous implementations rely on secret-sharing the sign, exponent and significand of the floating-point number separately for efficiency of computations, which makes it difficult to cover all the finer details of the IEEE 754 standard.

Using the CBMC-GC circuit compiler [17] (v.0.9.3 [12]), we were able to implement an efficient and fully IEEE 754 compliant floating-point protocol suite based on our hybrid protocol. The CBMC-GC compiler transforms C programs directly to highly optimized circuits usable in a GC protocol. This allowed us to use exact IEEE 754 software implementations as a basis for our protocols, thereby achieving compliance with the standard. We chose CBMC-GC for this task, since its latest version tends to produce smaller circuits than other similar compilers, such as [23]. Minimal circuit size, or specifically, the number of non-XOR gates, is paramount for the performance of any garbled circuits protocol. However, CBMC-GC does not scale well to very large circuits in terms of compilation time, as our results in Sec. 4.1 show. Nevertheless, CBMC-GC currently seems to be the best choice for implementing small efficient primitives using a garbled circuits method.

We implemented both single and double precision secret-shared floating-point data types. The `float` and `double` types are represented as 32-bit and 64-bit bitwise secret-shared integers that correspond exactly to the IEEE 754 standard. Our construction guarantees bit-by-bit identical results to those of

regular hardware floating-point procedures, excluding non-standardized details such as the significand bits of a NaN. We empirically verified this claim for the four arithmetic operations and square root for both precisions on a machine with Intel Core i7-870 2.93 GHz processor against equivalent C programs compiled with GCC 4.8.1-2. For all operations, we tested both the secure protocols and corresponding C programs on a representative sample of inputs to cover all corner cases and verified that the results were identical, while considering all NaN representations as equivalent.

### 4.1 Circuits for IEEE 754 Primitives

The circuits used in our protocol suite are listed and described in Table 1. We list circuit sizes as well as the number of garbled tables batches sent during one evaluation of the circuit. The circuits were compiled on a workstation with 16 GB RAM and an Intel Core i7-870 2.93 GHz processor. Although it would have much reduced the circuit sizes, we were unable to use the SAT-minimization functionality of CBMC-GC for larger circuits due to high compilation times. We note that compiling the largest circuit float64_erf used a maximum of ~10 GB of RAM. Also, CBMC-GC used only one processor core for compilation, which means the order of magnitude of these compilation times is unavoidable in practice with the current version of CBMC-GC, since it does not leverage the multi-core architecture of modern processors.

We used the efficient SoftFloat [34] IEEE 754 software implementation for compiling addition, multiplication, division and square root circuits. We additionally used musl libc [28] for double precision $e^x$ and error function (erf) as an example of more complex operations and the flexibility of our approach to implement arbitrary primitives. Only minor syntactic modifications of the source code were required to compile it with CBMC-GC. Although SoftFloat supports all four rounding modes described in the IEEE 754 standard, we hard-coded rounding to the default "Round to nearest, ties to even" mode, since this is most used in practice and provides the best bounds on rounding errors [15]. Alternatively, we could compile a different circuit for each rounding mode, or a single circuit that takes the rounding mode as an input.

**Table 1.** IEEE 754 floating-point operation circuits compiled with CBMC-GC

| Circuit | Non-XOR Gates | Total Gates | No of Batches | Used SAT-minimization | Compilation Time |
|---|---|---|---|---|---|
| float_add | 5671 | 7052 | 1 | + | 8 min 32 s |
| float_sub | 5671 | 7052 | 1 | + | 5 min 28 s |
| float_mul | 5138 | 7701 | 1 | + | 5 min 1 s |
| float_div | 12851 | 21384 | 1 | - | 58 s |
| float_sqrt | 35987 | 66003 | 2 | - | 2 min 40 s |
| double_add | 13129 | 15882 | 1 | + | 1 h 8 min |
| double_sub | 13129 | 15882 | 1 | + | 1 h 11 min |
| double_mul | 13104 | 25276 | 1 | + | 3 h 46 min |
| double_div | 36133 | 73684 | 2 | - | 5 min 35 s |
| double_sqrt | 85975 | 169932 | 4 | - | 10 min 11 s |
| double_exp | 393807 | 579281 | 8 | - | 1 h 13 min |
| double_erf | 2585188 | 3979603 | 52 | - | 47 h 4 min |

We chose to ignore all floating-point exceptions that may be raised during computations, since in an MPC environment, we cannot guarantee that raising an exception (e.g. division by zero) in the middle of a computation would not leak information about inputs. As our protocols correctly handle all special cases defined in the IEEE 754 standard such as NaNs, infinities and denormalized numbers, any exceptions will be reflected in the final result. For example, overflows and underflows will result in infinities, multiplying 0 with infinity results in NaN etc. Previous implementations [1][20][27] did not explicitly handle such cases and produced valid but meaningless results in error situations. Nonetheless, our approach would easily support raising standardized exception flags also in between operations, for example by adding an exception flag variable as an input and output to all circuits.

### 4.2 Performance Analysis

We measured the performance of all primitives implemented with our hybrid protocol and compared the results to existing approximation-based floating-point operations [20] in Sharemind. Note that Sharemind has undergone a complete rewrite of its network layer since [20] was published. Because of this, we also performed new benchmarks for the approximation-based floating-point. We used the division protocol based on Chebyshev polynomials and $e^x$ protocol with Taylor polynomials, since the precision of the Taylor polynomial version is better comparable with the IEEE 754 protocol.

The benchmarks were performed on a cluster of three nodes hosting Sharemind. All nodes had 48 GB of RAM and a 12-core 3GHz Intel CPU supporting AES-NI and HyperThreading. The nodes were connected to a LAN with 1 Gbps full duplex links. All tests were executed with a maximum of 24 concurrent garbler-evaluator pairs, as the hardware supports up to 24 parallel threads.

The performance results are shown in Table 2 for single precision and Table 3 for double precision. All measurements are presented in operations per second (ops) as the mean of 5 to 1000 iterations depending on the circuit size. The measurements depict the whole running time of the protocol including oblivious transfer, garbling and evaluation. Circuits are parsed and cached in an offline phase, however. The input size refers to the number of respective operations computed in one test using the parallelization techniques described in Sec. 3.1.

**Table 2.** Performance of single precision floating-point operations (ops)

| | | **Input size in elements** | | | | |
|---|---|---|---|---|---|---|
| | | **1** | **10** | **100** | **1000** | **10000** |
| Add | Approx. | 2.43 | 24.1 | 228.1 | 1496 | 3790 |
| | IEEE 754 | 24.99 | 134.5 | 477.2 | 583.6 | 597 |
| Multiply | Approx. | 7.76 | 77.94 | 751.6 | 5413 | 16 830 |
| | IEEE 754 | 26.17 | 135.5 | 506 | 632.9 | 632.9 |
| Divide | Approx. | 0.53 | 5.25 | 46.48 | 237 | 432.6 |
| | IEEE 754 | 14.53 | 88.2 | 233.6 | 279.1 | 284.5 |
| Square root | Approx. | 0.34 | 3.26 | 28.07 | 126.1 | 206.1 |
| | IEEE 754 | 7.83 | 44 | 92.9 | 105.1 | 106.6 |

Our measurements show that hybrid protocol IEEE 754 operations, excluding error function, are faster than approximation-based operations for smaller input sizes. The error function clearly illustrates the substantial overhead for evaluating very large circuits compared to a secret-sharing-based protocol, thereby motivating the composition of small but efficient primitives as opposed to full circuit programs. The IEEE 754 division and square root perform very well compared to approximation-based versions, whereas the error function and multiplication are slower on larger input sizes. Our protocols well outperform the results from [27] and our double precision addition and division are faster than the implementation of [1], however, multiplication is slightly slower. The latter is expected, since it is efficient to implement floating-point multiplication using secret sharing and less is gained from a GC approach. Overall, we see that for smaller input sizes, our IEEE 754 operations tend to be faster than secret-sharing-based alternatives.

The results also show that IEEE 754 operations do not benefit much from parallelization already after inputs of size ~100, while the approximation-based operations parallelize well to 10000 elements. This is due to the large size of the garbled tables that are transmitted over the network. Notice that since XOR-gates can be evaluated without sending the corresponding garbled tables to the evaluator, then, in practice, performance depends only on the number of non-XOR circuits in the gate.

In all larger tests with the IEEE 754 operations, the network link was constantly saturated, which introduced an inevitable upper bound on performance. This demonstrates the trade-off between GC and secret sharing, as GC generally requires more network communication, but has better round-complexity. For example, in our instantiation, the garbled circuit for `float` addition has size ~175 KB, whereas the approximation-based protocol uses at most 12 KB of one-way network communication for a single operation over a series of communication rounds. In both cases, this amount scales linearly with the input size, and consequently the sharing-based protocols have better amortized performance for larger inputs.

In practice, the input size can be used to dynamically choose between GC or sharing-based protocols. However, the overhead of converting between additive and bitwise shared values must also be taken into account. This overhead is relatively small when operations in one domain are grouped together, but may become significant when many conversions back-and-forth from bitwise to additive shares are required.

The fact that the network link was saturated for the IEEE protocols means that the performance of our implementation is close to the theoretical maximum when considering only communication overhead and disregarding the computational complexity entirely. If we consider only transferring the garbled tables over the network, the maximum performance for the `float` square root protocol using 1 Gbps network links is ~116 ops, since one garbled table of the square root circuit has size ~1.08 MB. Our implementation achieves 106.6 ops on input size 10000, which is very close to the theoretical maximum with this garbling scheme. The results are similar also for the other operations. The `float` square root protocol also achieved the highest amortized rate for the number of non-XOR circuit gates garbled and evaluated per second, which was a little over 3.8 million non-XOR gates per second.

The IEEE 754 protocols used significantly more memory and processing power, as all processor cores of the garbler node were nearly constantly working at maximum capacity. The effect of the high-speed parallel garbling on overall performance was nevertheless ultimately dominated by the network bandwidth, since the garbled tables were generated faster than they could be transferred to the evaluator. This was evident, as the garbler node finished processing much earlier (in larger tests, minutes earlier) than the evaluator, which suggests that less powerful hardware could have been used for similar results. The approximation-based counterparts used only ~10% of the hardware capability and the network saturation point arrived at much larger input sizes.

**Table 3.** Performance of double precision floating-point operations (ops)

| | | **Input size in elements** | | | |
|---|---|---|---|---|---|
| | | **1** | **10** | **100** | **1000** |
| Add | Approx. | 2.29 | 22.22 | 188.2 | 857.7 |
| | IEEE 754 | 16 | 103 | 228 | 260 |
| Multiply | Approx. | 7.17 | 71.98 | 647.9 | 3560 |
| | IEEE 754 | 13.74 | 90.8 | 221 | 259 |
| Divide | Approx. | 0.5 | 4.78 | 35.22 | 115.7 |
| | IEEE 754 | 7.31 | 46 | 89.2 | 101 |
| Square root | Approx. | 0.26 | 2.4 | 14.23 | 31 |
| | IEEE 754 | 3.57 | 23.3 | 39.5 | 43.4 |
| $e^x$ | Approx. | 0.28 | 2.57 | 14.7 | 31.1 |
| | IEEE 754 | 1.1 | 6.38 | 9 | 9.5 |
| Error function | Approx. | 0.3 | 2.92 | 19.8 | 55.4 |
| | IEEE 754 | 0.18 | 0.95 | 1.35 | 1.47 |

### 4.3 Private Satellite Collision Analysis with IEEE 754

To see the effects of different floating-point implementations on the performance of larger applications, we also carried out benchmarks on the satellite conjunction analysis algorithm from [20] using both our IEEE 754 and Sharemind's current approximation-based floating-point operations. The algorithm's running time and precision of the result depends on the number of iterations done in the integral approximation routine using Simpson's rule. We used 40 iterations similarly to [20]. The tests were performed on the same cluster as benchmarks in the previous section using double precision operations. The results of the tests are presented in Table 4.

**Table 4.** Collision analysis performance with 40 iterations for the integral approximation step. We include the total running time and also the running time per satellite pair in parentheses.

| | Number of satellite pairs | | |
|---|---|---|---|
| | **1** | **2** | **4** |
| Approx.-based | 77 (77) sec | 102 (51) sec | 153 (38) sec |
| IEEE 754 | 154 (154) sec | 292 (146) sec | 567 (142) sec |

We can see that the approximation-based floating-point outperforms the IEEE 754 implementation and also parallelizes better, as the cost for analyzing a single satellite pair greatly decreases when analyzing multiple pairs at once. However, examining the breakdown of the running times of different operations (Fig. 2 and Fig. 3), we see that the IEEE 754 version is slower mainly due to the low performance of its error function implementation. For a single satellite pair, the error function operation took up 76.9% of the whole analysis using the IEEE 754 floating-point. With the approximation-based floating-point, the percentage of the error function was only 7.7%.

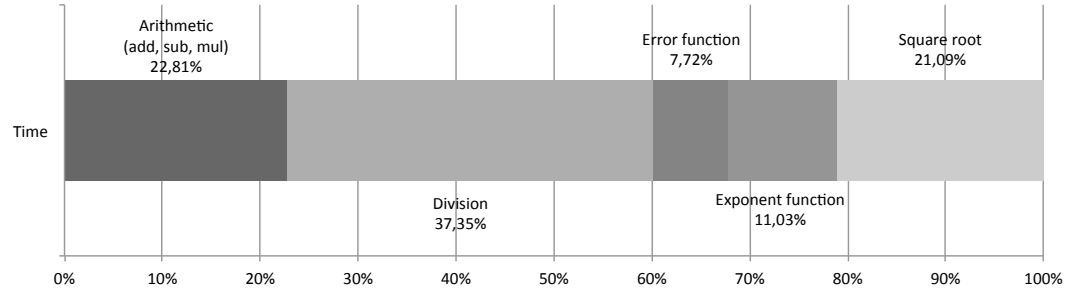


**Fig. 2.** Breakdown of operation runtime in collision analysis with approximation-based floating-point.

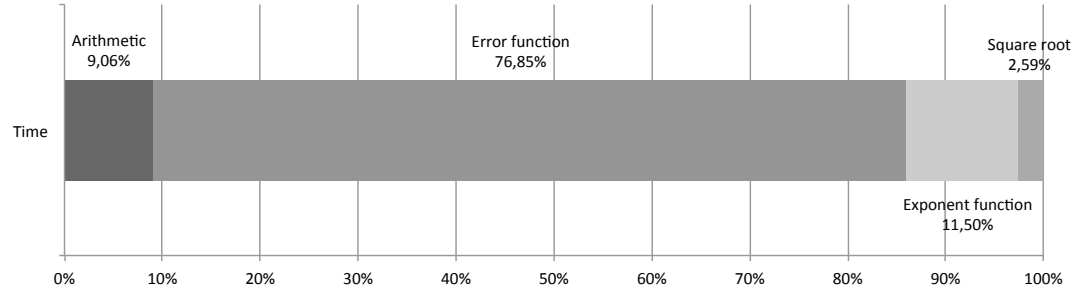


**Fig. 3.** Breakdown of operation runtime in collision analysis with IEEE 754 floating-point.

This suggests that the running time of the collision analysis could be much improved by using the IEEE 754 operations, but converting to the approximation-based floating-point format only for calculating the error function. This conversion is easily implemented using the bit extraction protocol from [11]. The error function is calculated only once in the algorithm on a vector of inputs, therefore only one conversion between floating-point formats is required, which will have negligible overhead compared to the gain of a much faster error function computation. Specifically, for 1 satellite pair, the error function is computed on an input vector of size 162. The conversion of 162 floating-point numbers to and from the secret-shared format takes time less than ~1 second. Thus, in this case, processing one satellite pair would

take ~43 seconds, which is almost two times faster than using only approximation-based floating-point. This demonstrates first-hand how our approach of combining different MPC methods can significantly increase the performance of applications, as the same operation implemented with different techniques can have very different performance profiles.

## 5 Conclusion

This work provided a protocol for combining GC with secret sharing. For this we consider a setting where the oblivious transfer for the garbled evaluation inputs can work for secret-shared inputs rather than the inputs known to the evaluator. In addition, it is required that the outputs of the garbled evaluation remain private. This allows us to combine the strengths of both approaches. Especially, efficient secret-sharing-based computation protocols can be augmented with easily generated GC based protocols for the functionalities where no known efficient sharing-based protocol exists. As an example, we added a very efficient first fully IEEE 754 compliant secure floating-point implementation to Sharemind. We also analyzed the performance of a private satellite collision analysis algorithm and showed that much greater performance can be achieved with our approach by using the more efficient method—either secret sharing or GC—for each subprotocol.

## Acknowledgments

We would like to thank the authors of the CBMC-GC circuit compiler for supporting us in our efforts to generate the circuits described in this paper.

## References

[1] Aliasgari, M., Blanton, M., Zhang, Y., Steele, A.: Secure computation on floating point numbers. In: Proc. of NDSS'13. The Internet Society (2013)
[2] Backes, M., Pfitzmann, B., Waidner, M.: The reactive simulatability (RSIM) framework for asynchronous systems. Information and Computation 205(12), 1685–1720 (2007)
[3] Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: Ortiz, H. (ed.) Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA. pp. 503–513. ACM (1990), http://doi.acm.org/10.1145/100216.100287
[4] Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: Proc. of SP'13. pp. 478–492. IEEE Computer Society, Washington, DC, USA (2013)
[5] Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. Cryptology ePrint Archive (2013)
[6] Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Proc. of CCS'12. pp. 784–796. ACM, New York, USA (2012)
[7] Ben-David, A., Nisan, N., Pinkas, B.: FairplayMP: a system for secure multi-party computation. In: Proc. of CCS'08. pp. 257–266. ACM (2008)
[8] Bogdanov, D.: Sharemind: programmable secure computations with practical applications. Ph.D. thesis, University of Tartu (2013)
[9] Bogdanov, D., Laud, P., Laur, S., Pullonen, P.: From input private to universally composable secure multi-party computation. In: Proc. of CSF'14. IEEE Computer Society (2014)
[10] Bogdanov, D., Laud, P., Randmets, J.: Domain-polymorphic programming of privacy-preserving applications. In: Proc. of PETShop'13. pp. 23–26. ACM (2013)
[11] Bogdanov, D., Niitsoo, M., Toft, T., Willemson, J.: High-performance secure multi-party computation for data mining applications. IJIS 11(6), 403–418 (2012)
[12] CBMC-GC. http://forsyte.at/software/cbmc-gc/
[13] Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Proc. of CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer (2012)
[14] Franz, M., Katzenbeisser, S.: Processing encrypted floating point signals. In: Proc. of MM&Sec'11. pp. 103–108. ACM, New York, NY, USA (2011)

[15] Goldberg, D.: What every computer scientist should know about floating-point arithmetic. ACM Computing Surveys 23(1), 5–48 (1991)
[16] Henecka, W., Kögl, S., Sadeghi, A.R., Schneider, T., Wehrenberg, I.: TASTY: Tool for automating secure two-party computations. In: Proc. of CCS'10. pp. 451–462. ACM, New York, NY, USA (2010)
[17] Holzer, A., Franz, M., Katzenbeisser, S., Veith, H.: Secure two-party computations in ANSI C. In: Proc. of CCS'12. pp. 772–783. ACM (2012)
[18] Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: Proc. of SEC'11. USENIX Association (2011)
[19] 754-2008 - IEEE standard for floating-point arithmetic. `http://ieeexplore.ieee.org/servlet/opac?punumber=4610933` (2008)
[20] Kamm, L., Willemson, J.: Secure floating-point arithmetic and private satellite collision analysis. IJIS (to appear 2015)
[21] Kolesnikov, V., Sadeghi, A., Schneider, T.: A systematic approach to practically efficient general two-party secure function evaluation protocols and their modular design. Journal of Computer Security 21(2), 283–315 (2013), `http://dx.doi.org/10.3233/JCS-130464`
[22] Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Proc. of ICALP (2). LNCS, vol. 5126, pp. 486–498. Springer (2008)
[23] Kreuter, B., Mood, B., Shelat, A., Butler, K.: PCF: A portable circuit format for scalable two-party secure computation. In: Proc. of SEC'13. pp. 321–336. USENIX Association, Berkeley, CA, USA (2013)
[24] Kreuter, B., Shelat, A., Shen, C.: Billion-gate secure computation with malicious adversaries. In: Proc. of Security'12. USENIX Association (2012)
[25] Laur, S., Willemson, J., Zhang, B.: Round-Efficient Oblivious Database Manipulation. In: Proceedings of the 14th International Conference on Information Security. ISC'11. pp. 262–277 (2011)
[26] Lindell, Y., Pinkas, B.: A proof of security of Yao's protocol for two-party computation. J. Cryptology 22(2), 161–188 (2009)
[27] Liu, Y.C., Chiang, Y.T., Hsu, T.S., Liau, C.J., Wang, D.W.: Floating point arithmetic protocols for constructing secure data analysis application. Procedia Computer Science 22(0), 152 – 161 (2013)
[28] musl libc. `http://www.musl-libc.org/`
[29] Pfitzmann, B., Waidner, M.: A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. In: Proc. of SP'01. pp. 184– (2001)
[30] Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Proc. of ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer (2009)
[31] Seroussi, G.: Table of low-weight binary irreducible polynomials. `http://www.hpl.hp.com/techreports/98/HPL-98-135.html` (1998)
[32] Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (1979)
[33] Siim, S., Bogdanov, D.: A general mechanism for implementing secure operations on secret shared data. Tech. Rep. T-4-21, Cybernetica, `http://research.cyber.ee/`. (2014)
[34] SoftFloat. `http://www.jhauser.us/arithmetic/SoftFloat.html`
[35] Yao, A.C.: Protocols for secure computations. In: Proc. of SFCS'82. pp. 160–164. IEEE Computer Society, Washington, DC, USA (1982)

## A Input Privacy Based Security

Our security proofs are based on the proof framework that allows to securely combine input private protocols with secure protocols to obtain universally composable secure protocols as defined in [9]. In this section, we give a general overview of this framework and introduce the more important definitions and results. For a longer introduction see [9]. The general idea is that for most MPC protocols it is sufficient to show that they are input private and that a secure protocol can be obtained via specific compositions of input private and secure protocols. The formalization is defined based on the reactive simulatability (RSIM) framework [29],[2], but here we only consider the limited versions required for [9] and focus on giving an overview rather than all the details. Each party is modelled as a machine $M$ and a protocol is executed by a group $\hat{\mathbf{M}}$ of machines whose program corresponds to the protocol description of the respective party. For example, each party in a protocol can be represented by one machine and the collection $\hat{\mathbf{M}}$ then consists of all the parties described. Besides the machines described by the protocol,

there are two important machines to consider: the honest user (or the environment) $H$ and the adversary $\mathcal{A}$. The adversary can corrupt parties, especially for passive security, the adversary sees everything that the corrupted party sees in the protocol. The honest user represents all previous, simultaneous and following computations and is responsible for giving protocol inputs as well as learning the outcomes.

Machines communicate with each other over different ports. Particularly, for a collection $\hat{\mathbf{M}}$ we specify the set of ports $S$ that is used to communicate with the honest user $H$. In addition, all ports not in $S$ are forbidden for $H$ to use to communicate with $\hat{\mathbf{M}}$. Moreover, ports of $\hat{\mathbf{M}}$ not in $S$ and not used to communicate between machines in $\hat{\mathbf{M}}$ are used for communicating with the adversary $\mathcal{A}$. Analogously, there are ports that allow communication between $H$ and $\mathcal{A}$. We call $(\hat{\mathbf{M}}, S)$ a system $Sys$. Note that the system describes our protocol by fixing the codes of machines in $\hat{\mathbf{M}}$ as well as the intended input and output ports that are contained in $S$. Especially, for each system the interface that it uses to communicate with either $H$ or $\mathcal{A}$ is well defined by $S$. Based on this we define a set of configurations $Conf(Sys)$ that contains tuples $(\hat{\mathbf{M}}, S, H, \mathcal{A})$ for each valid $H$ and $\mathcal{A}$ with respect to the interface specified for the system $Sys$ so that this set does not have any open ports. A configuration is said to be polynomial time, if respective $H$ and $\mathcal{A}$ are polynomial time machines. The communication is illustrated on Fig. 4 where all arrows denote one way communication defined by a set of respective ports.

If the protocols are executed then the machines in $Sys$ obtain their inputs from $H$, interact together to compute the required functionality, and return outputs to $H$. In addition, for passive corruption all corrupted parties always forward their view of the protocol as well as their current state to the adversary. A view of the party is the set of all messages it receives in the communication.

**Definition 1 (Simulatability).** *We say that $Sys_1 = (\hat{\mathbf{M}}_1, S)$ is perfectly as secure as $Sys_2 = (\hat{\mathbf{M}}_2, S)$ if, for every configuration $conf_1 = (\hat{\mathbf{M}}_1, S, H, \mathcal{A}_1) \in Conf(Sys_1)$, there exists a configuration $conf_2 = (\hat{\mathbf{M}}_2, S, H, \mathcal{A}_2) \in Conf(Sys_2)$ with the same $H$ such that the environments have coinciding views*

$$view_{conf_1}(H) = view_{conf_2}(H) \ .$$

Similar definitions can be given for computational and statistical security. For computational security, views created by all polynomial configurations $Conf(Sys_1)$ should be computationally indistinguishable from some views of configurations in $Conf(Sys_2)$. Analogously, statistical indistinguishability of views for all configurations is required for statistical security.

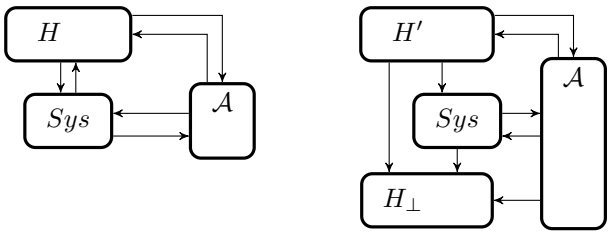


**Fig. 4.** Security configuration with $H$, $\mathcal{A}$ and our system and privacy configuration with $H = H' \cup H_\perp$. Arrows denote the communication directions.

For secret-sharing-based information theoretic secure multi-party computation we require that the ideal functionality is such that the output shares are uniformly random shares of the computed result. The only case where we can allow for a relaxation is when no communication occurs in the protocol. This captures the intuition that the output shares should not leak anything until the output value is published and even then they should not leak more than is revealed by the output value. If the shares are not uniformly random then they might give extra information about the input shares or performed computations. For example when the final shares are opened when declassifying a value then all that should be obtained is the output of the computation. However, non-uniform shares might reveal information about the flow of the computation.

The main goal of MPC protocol security is that nothing is leaked about the inputs other than what can be learned from the outputs. This is well captured by the simulatability security definition. However, in secret-sharing-based MPC for example, it is also interesting to consider protocols that have secret

shared output, meaning that the output each party gets, should leak nothing about the inputs or the output value. This is formalized as the input privacy notion, because in such protocols no information should be leaked about the protocol inputs. Input privacy is a special security notion in the sense that it only protect the inputs of honest parties. It is not sufficient for ensuring the correctness or secrecy of the outputs of the protocol. However, it is sufficient for secure computation as long as the parties do not publish their private outputs.

**Definition 2 (Input privacy).** *We say that* $Sys_1 = (\hat{\mathbf{M}}_1, S)$ *is perfectly at least as input-private as* $Sys_2 = (\hat{\mathbf{M}}_2, S)$ *if, for every privacy configuration* $conf_1 = (\hat{\mathbf{M}}_1, S, H' \cup H_\perp, \mathcal{A}_1) \in Conf(Sys_1)$*, there exists a privacy configuration* $conf_2 = (\hat{\mathbf{M}}_2, S, H' \cup H_\perp, \mathcal{A}_2) \in Conf(Sys_2)$ *with the same* $H' \cup H_\perp$*, such that the restricted views coincide* $view_{conf_1}(H') = view_{conf_2}(H')$*.*

The important difference from the simulatability definition is that the environment $H$ consist of two distinct parts $H = H' \cup H_\perp$ where only $H_\perp$ learns the joint output of the protocol, but does not help $H'$ in distinguishing the real and the simulated worlds. This $H$ is also illustrated on Fig. 4 and means that in the formalization we do not care about the consistency of the outputs as they are in some sense discarded to $H_\perp$ only that the inputs and consecutive communication do not give the adversary the power to distinguish real world and the ideal world.Analogously to the simulatability security definition, we can also consider the computational and statistical flavors of the definition.

In addition, an *ordered composition* of protocols is considered, where two systems $Sys_1$ and $Sys_2$ are said to be in ordered composition $Sys_1 \to Sys_2$ if $Sys_1$ can give inputs to $Sys_2$, but not vice versa. This is required to easily specify which sub-protocols can provide the outputs of the final protocol. Finally, as the input privacy definition does not ensure the correctness of the input private protocol with respect to the specified ideal functionality, we also have to restrict which protocols can be securely and correctly composed. For this purpose, the *output predictability* (Definitions 9 and 10 in [9]) is defined for a composition to capture the correctness as well as the fact that the final protocol does not leak too much information about its inputs. Notably, two properties can ensure predictability of $Sys_1 \to Sys_2$, $Sys_1$ has to be correct and $Sys_2$ should not leak the values of the input shares. Furthermore, for Sharemind, the `Reshare` protocol is well suitable for $Sys_2$ as it just sends uniformly random values and does not reveal anything about the input shares. Note that `Reshare` is a special protocol as it is secure as well as input private.

The two main results of [9] are that the ordered composition of two input private systems is input private (Thm. 3 in [9]) and that the ordered and output predictable composition of input private and secure system is secure (Thm. 2 in [9]). These establish that we can prove that some main component of the protocol is input private and then we can combine input private protocols for specific operations to form an input private protocol for the required computations. Only the full protocol needs to be finished with some secure protocol to obtain a secure resulting protocol for the desired computations.

## B Security Definitions for Garbled Circuits

A good formalization of garbled circuits security is given in [6]. The two security notions relevant to this work are privacy and obliviousness. In fact, given the previous treatment of MPC, the simulatability property of MPC is similarities to the privacy property of garbled circuits and likewise connection exists between input privacy and obliviousness that both rely on the construction not leaking the outputs and hiding the inputs.

Both of the GC security properties have two flavors of definitions: indistinguishability based and simulation based. In general, the two are not equivalent and in such cases the simulatability based definition is considered to be the correct one. However, for our purposes the two versions of definitions coincide. Both flavors are defined via security games. The ones we need for this work, are given on Fig. 5. We require PrvInd for indistinguishability based privacy definition as this is the property proved for the garbling scheme that we use. In addition, we need to examine both versions of obliviousness as this is the property required for the security of our hybrid scheme. In the games we consider a security parameter $k$, garbled circuit $F$, garbled inputs $X$, garbling function `Gb`, encoding information $e$, encoding function $\mathbb{E}$ and output decoding information $d$. Additionally, $\mathcal{S}$ is a simulator. Each game starts and finishes with the described initial and final procedures respectively. Initializing procedure picks the challenge bit and the finalizing phase checks if the adversary correctly distinguished between the two challenges.

**Algorithm 6:** Game PrvInd, function $\textsc{Garble}(f_0, f_1, x_0, x_1)$

**if** $\Phi(f_0) \neq \Phi(f_1) \vee f_0(x_0) \neq f_1(x_1) \vee \{x_0, x_1\} \not\subseteq \{0,1\}^{f_0.n}$ **then**
  **return** $\perp$
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f_b)$
$X \leftarrow \mathbb{E}(e, x_b)$
**return** $(F, X, d)$

**Algorithm 7:** Game ObvInd, function $\textsc{Garble}(f_0, f_1, x_0, x_1)$

**if** $\Phi(f_0) \neq \Phi(f_1) \vee \{x_0, x_1\} \not\subseteq \{0,1\}^{f_0.n}$ **then**
  **return** $\perp$
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f_b)$
$X \leftarrow \mathbb{E}(e, x_b)$
**return** $(F, X)$

**Algorithm 8:** Game ObvSim, function $\textsc{Garble}(f, x)$

**if** $x \not\subseteq \{0,1\}^{f_0.n}$ **then**
  **return** $\perp$
**if** $b = 1$ **then**
  $(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$
  $X \leftarrow \mathbb{E}(e, x)$
**else**
  $(F, X) \leftarrow \mathcal{S}(1^k, \Phi(f))$
**return** $(F, X)$

**Algorithm 9:** Initializing procedure, function $\textsc{Initialize}()$

$b \xleftarrow{\$} \{0,1\}$
**return** $b$

**Algorithm 10:** Finalizing procedure, function $\textsc{Finalize}(b')$

**return** $b == b'$

**Fig. 5.** Games for defining prv.ind, obv.ind and obv.sim security of a garbling scheme with garbling algorithm $\mathsf{Gb}$ and encoding algorithm $\mathbb{E}$.

The main difference of the two security properties is that in case of privacy the adversary obtains the whole information needed to evaluate the garbled circuit and decode the result. However, for obliviousness the decoding information is missing meaning that the adversary can evaluate the circuit, but can only obtain the tokens of the output wires. Moreover, as the two functions in the obliviousness games do not need to have the same output, then this definition guarantees that as long as the decoding information is unknown the circuit does not reveal the output value. However, if the decoding becomes known then something besides the output value might also be revealed about the evaluation process. On the contrary, for the private scheme the decoding value $d$ is present, but the output of the garbling scheme may be decodable even without the $d$. Still, the definition of privacy is sufficient to ensure that nothing that can not be learned from the output value is leaked in the protocol. Hence, these notions are both interesting in different use-cases.

As one can not achieve absolute security then the notion of side-information function $\Phi$ is used to specify what the garbling construction leaks. The most common part of $\Phi$ is a function $\Phi_{topo}$ that leaks the topology of the circuit. In this work, we also consider $\Phi_{xor}$ because using free-XOR technique reveals which gates are XOR. From [4] we know that these functions are both efficiently invertible and, therefore, by equivalence relations from [6], indistinguishability and simulation based definitions of privacy and obliviousness coincide. In the following, we use the GaXR garbling scheme from [4] that is known to be private and for which $\Phi$ is efficiently invertible. However, for fitting well with the general proof framework of [9] where most of the protocols need to be input private, we need to show that this garbling scheme is also oblivious.

The GaXR garbling scheme relies on a $\sigma$-derived dual key cipher (DKC) for a mapping $\sigma$ that satisfies several properties from [4]. Note that the DKC that we used is defined for random permutations as well as the security of the GaXR scheme is defined in the *random permutation model*. In our instantiations, we use AES as described in Sec. 3.1. Our security proofs are based on the original privacy proof of the GaXR scheme that can be found in [5] and holds in the random permutation model. However, for our protocol we also need to establish that the used garbling scheme is oblivious.

## C Security Proof of the Hybrid Protocol

We use the general proof framework of [9] introduced in Appendix A to show that the hybrid protocol is secure against passive static adversaries that corrupt at most one participant. Lets denote the part of the hybrid protocol on Fig. 1 before final `Reshare` protocol as `Hybrid'`. In this case, the total protocol

can be seen as an ordered composition of `Hybrid'` and `Reshare`. For this we need to prove that the protocol up until the `Reshare` function is input private and then apply the composition result from [9]. However, for applying this theorem we also need to establish the output predictability of the composition in addition to the the privacy of the main protocol `Hybrid'`. Likewise, note that `Reshare` (Alg. 1 in [8]) is secure and is well suitable for the second protocol in the output predictability definition as it does not use the input shares for any of the protocol messages and totally rerandomizes the inputs.

We use the garbling scheme GaXR from [4] and we use a version of the correct $\sigma$-derived DKC from the same paper. The indistinguishability based privacy and obliviousness proofs are formalized as games PrvInd and ObvInd on Fig 5.

In order to obtain the output predictability of the composition, we at first require that the private composed protocol is correct. This is necessary to ensure that the private protocol computes the same functionality as the associated ideal functionality, currently defined by the function $f$ in the hybrid protocol.

The ideal functionality corresponding to `Reshare` is such that it takes the input as shares and outputs a uniformly random sharing of the same secret value that was encoded by the inputs. For `Hybrid'` the ideal functionality is such that it takes as inputs the shares $[\![\mathbf{x}]\!]$ and outputs the value $\mathbf{y} = f(\mathbf{x})$ as shares $[\![\mathbf{y}]\!]$ where $[\![\mathbf{y}]\!]_3 = (0, \ldots, 0)$ but the other shares $[\![\mathbf{y}]\!]_1$ and $[\![\mathbf{y}]\!]_2$ are uniformly random. It is easy to see that the corresponding ideal functionality for the hybrid protocol is such that it outputs uniformly random shares of $f(\mathbf{x})$. Note that this corresponds to the definition of an ideal functionality for a secret-sharing-based secure multi-party computation protocol.

**Theorem 1.** *`Hybrid'` protocol is correct.*

*Proof.* The correctness follows from the correctness of the sub-protocols used in the `OT` part and the correctness of the GaXR garbling scheme. We require that if all participants follow the protocol, then the outputs are correct. We can consider `Hybrid'` in two parts: `OT` and garbling scheme. If oblivious transfer transmits the correct keys then the rest follows from the correctness of the algorithm pair for garbling and evaluation.

*The `OT`* is trivially correct if the used subprotocols are correct. It is easy to see that if the secret input bit $x_i = 0$ then $[\![X_i]\!] = [\![X_i^0]\!] \cdot (1 - 0) + [\![X_i^1]\!] \cdot 0 = [\![X_i^0]\!]$ and analogously $[\![X_i]\!] = [\![X_i^1]\!]$ if $x_i = 1$. Hence, we always have $[\![X_i]\!] = [\![X_i^{x_i}]\!]$ and the `OT` protocol is correct.

*The garbling scheme.* The DKC that we use is correct, meaning that given the correct keys the decoding can recover the encoded message. The rest of the correctness follows from the usage of standard techniques for garbling in GaXR, including free-XOR and garbled row reduction.

**Corollary 1.** *The ordered composition of `Hybrid'` and `Reshare` is jointly output predictable.*

*Proof.* The output predictability follows from Lemma 2 in [9] and Thm. 1. For a correct protocol `Hybrid'` we know that it is sufficient to show an output predictor for the composition of `Hybrid'` and ideal `Reshare`. However, this is simple as the first part `Hybrid'` is correct and the ideal `Reshare` protocol is such that on input $[\![x]\!]$ its behaviour, by definition, depends only on the value $x$ and not on the shares $[\![x]\!]_i$. Therefore, the output predictor can just compute the functionality `Hybrid'` based on the input circuit $f$ and output uniformly random shares of the result.

In order to establish the privacy of `Hybrid'`, we need to analyze the combination of secret-sharing-based `OT` and the garbling scheme. Note that the obliviousness property of the garbling scheme [6] is quite like the input privacy property [9] and, in fact, the obliviousness of the garbling scheme is required for the composed system to be input private. Intuitively, it is clear that we can securely continue the computation after `Hybrid'` only if the output tokens obtained by the evaluator $\mathcal{CP}_2$ do not reveal the output to $\mathcal{CP}_2$. Hence, we have to establish that the used GaXR scheme that is known to be private is also oblivious. This can be done analogously to obtaining obliviousness proof of Garble2 from privacy proof of Garble1 in [6], however, in the following we consider a reduction based proof that is easier to follow just considering the games on Fig. 5.

**Theorem 2.** GaXR *scheme is computationally obv.ind secure.*

*Proof.* The only thing we need to consider here is if the types (least significant bits) of the output wire tokens are independent of the semantics of these wires. The rest of the proof follows the proof of privacy

of GaXR from the full version [5] of [4] except that the procedure Garble returns only $(F, X)$ and not the decoding information.

Clearly the types of the input wires are independent of the semantics as they are generated independently on line 3 by $\mathcal{CP}_1$ in Alg. 3. In short, the rest of the independence follows from the fact that the tokens of the circuit outputs are generated the same way as the intermediate tokens in the circuit. Therefore, if the output types would reveal information about the semantics then also some intermediate values on the same (or other circuits that contain this circuit) would leak information about the state of the evaluation and invalidate the privacy property. Following is a longer specification of this observation.

Assume, by contradiction, that there exists some adversary $\mathcal{A}$ that is good at the ObvInd game, meaning that it wins with a non-negligible probability. In addition, without loss of generality, we assume that $\mathcal{A}$ always queries Garble with inputs that pass the initial checks. Consider an adversary $\mathcal{B}$ against the game PrvInd. We can construct the adversary $\mathcal{B}$ such that it interacts with the adversary $\mathcal{A}$. If $\mathcal{A}$ queries $(f_0, f_1, x_0, x_1)$ from Garble, then $\mathcal{B}$ takes a constant function $f^*$ and concatenates this with $f_0$ and $f_1$ to obtain $f_0^* = f^* \circ f_0$ and $f_1^* = f^* \circ f_1$. All outputs of $f_i$ are inputs to $f^*$ that then outputs a constant value $c$. Trivially $f_0^*(x_0) = c = f_1^*(x_1)$ and this is a valid input to PrvInd. Especially, the addition of $f^*$ affects the $\Phi_{topo}$ and $\Phi_{xor}$ outcomes in the same way giving $\Phi(f_1^*) = \Phi(f_0^*)$ (as we had $\Phi(f_1) = \Phi(f_0)$ from the checks in the game) and trivially the inputs still belong to the correct range, therefore making it a useful query for $\mathcal{B}$.

Clearly $f_1$ and $f_0$ must have the same number of outputs as $\Phi_{topo}(f_1) = \Phi_{topo}(f_0)$. Assume, that they have two output bits $x_1$ and $x_2$, then we could have $f^*(x_1, x_2) = x_1 \wedge x_2 \wedge (x_1 \oplus x_2) = 0$ that can be extended as $f^*(x_1, \ldots, x_n) = x_1 \wedge x_2 \wedge \ldots \wedge x_n \wedge (x_1 \oplus x_2) = 0$ for more than two outputs. For one bit output $x_1$ the chosen $f^*$ has to be a bit more invasive and, for example, can also define an additional input bit $y_1$ to compute $f^*(x_1, y_1) = x_1 \wedge y_1 \wedge (x_1 \oplus y_1) = 0$. In such case, also the token corresponding to this extra bit must be stripped by the adversary $\mathcal{B}$ to forward the garbled function to $\mathcal{A}$. In total, $f^*$ can always be added with linear time in the output length of $f_0$ and $f_1$ which is trivially at most linear in the original circuit size.

Adversary $\mathcal{B}$ receives $(F^*, X, d)$ from the game. It then computes $F$ by removing the gates from $F^*$ that correspond to $f^*$ and outputs $(F, X)$ to $\mathcal{A}$. The garbling scheme proceeds in topological order and the garbling of gates corresponding to the final part $f^*$ does not affect the garbled output corresponding to $f$. In addition, these gates can be removed as the output wires are generated in the same manner as the intermediate wires of the circuit, hence also the intermediate wires in $F^*$ that correspond to outputs of $F$ are valid output tokens for $F$. Adversary $\mathcal{B}$ outputs the same final value as $\mathcal{A}$. As argued before $f^*$ adds at most linear overhead, therefore, the running-time of $\mathcal{B}$ is at most a linear factor longer that $\mathcal{A}$ that has to prepare the initial circuits. For a polynomial time $\mathcal{A}$ the new adversary $\mathcal{B}$ is also polynomial time.

We clearly have $\mathbf{Adv}(\mathcal{B}) = \mathbf{Adv}(\mathcal{A})$ because $\mathcal{B}$ outputs correct answer exactly if $\mathcal{A}$ gave a correct answer as both of these cases had the same challenge bit. In addition, $\mathcal{A}$ had the same view as it usually does in the ObvInd game because adding $f^*$ and removing it from $F^*$ later gives the same output as just garbling $f$ to obtain $F$. Furthermore, the inputs $X$ are generated in the same way and connected to $x_b$ for challenge $b$. In total, this is a contradiction because we know that $\mathbf{Adv}(\mathcal{B})$ is negligible because the scheme is prv.ind secure, but we assumed $\mathbf{Adv}(\mathcal{A})$ to be non-negligible. Therefore there can exist no such adversary $\mathcal{A}$ that breaks the obv.ind security.

**Corollary 2.** GaXR *is computationally obv.sim secure.*

*Proof.* We know that $\Phi_{topo}$ and $\Phi_{xor}$ are efficiently invertible and therefore obv.ind security and obv.sim security are equivalent as shown in [6]. Hence, it follows directly from Thm. 2.

From Cor. 2 we know that there exists a simulator $\mathcal{S}$ such that for inputs $\mathcal{S}(1^k, \Phi(f))$ it outputs $(F, X)$ indistinguishable from those output by the garbling scheme. This simulator is defined by game ObvSim in [6] and here on Fig. 5. In the following, we use this simulator to prove the input privacy of the `Hybrid'` protocol.

In the following, we call the computation part of the `OT` protocol in Alg. 2 the oblivious choice to distinguish it from the final part where the values of the input tokens are opened for the evaluator $\mathcal{CP}_2$.

**Theorem 3.** *Protocol* `Hybrid'` *is perfectly input private for passively corrupted* $\mathcal{CP}_1$ *or* $\mathcal{CP}_3$ *and computationally input private against passively corrupted* $\mathcal{CP}_2$.

*Proof.* We have to consider the possibility of each of the three parties being corrupted by the adversary. For each of these cases we have to show the existence of the privacy simulator that can, based on the inputs of the corrupted party, simulate the view of the corrupted party.

*Party $\mathcal{CP}_1$ or $\mathcal{CP}_3$ is corrupted.* The only incoming communication for these parties in `Hybrid'`occurs during the `OT` phase, especially during the computation of the oblivious choice, but not during declassification. Therefore, the perfect input privacy of these parties is ensured by the perfect input privacy of the addition and multiplication protocol and the composability of input privacy (Thm. 3 in [9]).

*Party $\mathcal{CP}_2$ is corrupted.* The privacy simulator for $\mathcal{CP}_2$ can be built from the simulator $\mathcal{S}$ in the obv.sim privacy definition of the garbling scheme (game ObvSim on Fig. 5). The privacy simulator knows the circuit $f$ that is to be computed and also has the security parameter $k$, therefore, it can run $\mathcal{S}(1^k, \Phi(f))$ to obtain $(F, X)$. Next, it has to simulate the oblivious transfer with output $X$ that can be done perfectly by using the privacy simulator for the computation part of the `OT` protocol and finally, simulating the declassifying procedure with output $X$, by choosing suitable $[\![X]\!]_1$ and $[\![X]\!]_3$. In addition, it has to simulate $\mathcal{CP}_1$ sending garbled circuit to $\mathcal{CP}_2$ that can be done by sending $F$. In the following, we analyze the correctness of this simulator construction.

First, consider the simulation of the `OT` protocol in more detail to understand why it can be simulated perfectly. In short, if the output $X$ is known then `OT` is also secure against corrupted receiver and there exists a simulator that can simulate the protocol run with that output. This simulation works as follows. It is clear that the oblivious choice part of the protocol is perfectly input private because it is a composition of perfectly input private protocols. We can use the privacy simulator for this part. For the declassification protocol we still need to analyze further why opening the output to $\mathcal{CP}_2$ can be perfectly simulated in this protocol. Note that, as analyzed in [9], the full multiplication protocol of Sharemind is secure because of the finishing `Reshare` step. Hence, the output shares of the multiplication protocol are uniformly random. In addition, the final addition protocol in the oblivious choice protocol then adds two uniformly random shares locally and also obtains a uniformly random output. Therefore, oblivious choice protocol is also secure and has a uniformly random output share for each party. In the opening, $[\![X]\!]_1$ and $[\![X]\!]_3$ can be simulated by choosing $[\![X]\!]_1$ uniformly at random and fixing $[\![X]\!]_3 = X - [\![X]\!]_1 - [\![X]\!]_2$ where the simulator had to compute $[\![X]\!]_2$ as corrupted party's output in oblivious choice. This simulation is perfect.

Assume, by contradiction, that the previous construction is not a good privacy simulation for `Hybrid'` and there exists some configuration with environment $H = H' \cup H_\perp$ and adversary $\mathcal{A}$ (that has corrupted $\mathcal{CP}_2$) for which the real and simulated worlds of some functionality $f$ are distinguishable with some non-negligible probability. Denote this combination as $\mathcal{A}^H$. Consider an adversary $\mathcal{B}$ against the obv.sim security of the garbling scheme that functions as a *Sys* that interacts with $\mathcal{A}^H$ (as in the privacy definition on Fig. 4) and also interacts with the ObvSim game. At first the adversary $\mathcal{B}$ runs $\mathcal{A}^H$ and the respective needed simulation of the oblivious choice, until it has received all shares of $[\![\mathbf{x}]\!]$ from $H$. Specifically, the parties that have already received their input shares can send the first round of messages of the multiplication protocol as part of the `OT` and $\mathcal{B}$ can simulate these messages using the privacy simulator of the `OT`. Upon receiving all input shares $[\![\mathbf{x}]\!]$, the adversary $\mathcal{B}$ restores the value $\mathbf{x}$ and queries GARBLE of the ObvSim game with input $(f, \mathbf{x})$ for the circuit $f$ computed in this instance of the hybrid protocol. It receives $(F, X)$ and continues the simulation of the `OT` protocol with output $X$ and simulation of the `Hybrid'` protocol with the garbled tables $F$. In the end, it outputs the choice $b$ of the honest user $H'$.

For computational security, we require $H$ and $\mathcal{A}$ as well as the simulator $\mathcal{S}$ and real garbling algorithm to be polynomial time by definition. In addition, the `OT` protocol as well as its simulation is polynomial time. Therefore the new adversary $\mathcal{B}$ is also polynomial time because it is a combination of polynomial time machines.

We know that the rest of the privacy simulation, except for the choice of $(F, X)$, is perfect. Especially, this means that this part of the view of $H'$ in the real and simulated world is indistinguishable for the cases where $X$ and $F$ are indistinguishable in the two worlds. Therefore, $\mathcal{B}$ wins exactly if $H'$ successfully distinguishes the real and the ideal world, because this is exactly when $H'$ distinguished simulated $(F, X)$ from the real ones. This is a contradiction, as the garbling scheme is obv.sim secure, but we assumed that the total protocol is not input private. Hence, there can not exist any such configurations that provide $\mathcal{A}$ and $H$ for our $\mathcal{A}^H$ and invalidate the input privacy with respect to the described simulator.

**Corollary 3.** *Hybrid protocol algorithm is perfectly secure against passively corrupted* $\mathcal{CP}_1$ *and* $\mathcal{CP}_3$ *and computationally secure against passively corrupted* $\mathcal{CP}_2$*.*

*Proof.* From Cor. 1 we know that the composition is jointly output predictable. Clearly `Hybrid'` and `Reshare` are in ordered composition because all outputs of `Hybrid'` are inputs to the secure `Reshare` protocol. There is no data dependency from `Reshare` to `Hybrid'`. Thm 3 also showed that `Hybrid'`, being the first part of the ordered composition, is input private. Finally, `Reshare` is also secure. Therefore, we can use the composition result that ordered composition of input private and secure protocols is secure if the composition has predictable outcome [9] to conclude that the full hybrid protocol is secure.