Garbled Protocols and Two-Round MPC from Bilinear Maps*

Sanjam Garg
University of California, Berkeley
sanjamg@berkeley.edu

Akshayaram Srinivasan
University of California, Berkeley
akshayaram@berkeley.edu

Abstract

In this paper, we initiate the study of *garbled protocols* — a generalization of Yao's garbled circuits construction to distributed protocols. More specifically, in a garbled protocol construction, each party can independently generate a garbled protocol component along with pairs of input labels. Additionally, it generates an encoding of its input. The evaluation procedure takes as input the set of all garbled protocol components and the labels corresponding to the input encodings of all parties and outputs the entire transcript of the distributed protocol.

We provide constructions for garbling arbitrary protocols based on standard computational assumptions on bilinear maps (in the common random/reference string model). Next, using garbled protocols we obtain a general compiler that compresses any arbitrary round multiparty secure computation protocol into a two-round UC secure protocol. Previously, two-round multiparty secure computation protocols were only known assuming witness encryption or learning-with errors. Benefiting from our generic approach we also obtain two-round protocols (i) for the setting of random access machines (RAM programs) while keeping the (amortized) communication and computational costs proportional to running times, (ii) making only a black-box use of the underlying group, eliminating the need for any expensive non-black-box group operations and (iii) satisfying semi-honest security in the plain model.

Our results are obtained by a simple but powerful extension of the non-interactive zero-knowledge proof system of Groth, Ostrovsky and Sahai [Journal of ACM, 2012].

*Research supported in part from 2017 AFOSR YIP Award, DARPA/ARL SAFEWARE Award W911NF15C0210, AFOSR Award FA9550-15-1-0274, NSF CRII Award 1464397, and research grants by the Okawa Foundation, Visa Inc., and Center for Long-Term Cybersecurity (CLTC, UC Berkeley). The views expressed are those of the author and do not reflect the official policy or position of the funding agencies.

Contents

1 Introduction

Yao's garbled circuits [Yao86] (also see [AIK04, LP09, BHR12]) are enormously useful in cryptography. In a nutshell, Yao's construction on input a circuit C generates a garbled circuit $\widetilde{C}$ along with input labels $\{\mathsf{lab}_{i,0}, \mathsf{lab}_{i,1}\}$ such that $\widetilde{C}$ and $\{\mathsf{lab}_{i,x_i}\}$ can be used to compute $C(x)$ and nothing more. Over the years, Yao's construction has found numerous applications (to name a few [AF90, BMR90, FKN94, KO04, GKR08]) and several extensions [GHV10, AIK11, LO13] have been investigated. Furthermore, in light of their usefulness, substantial research has been invested to improve the practical efficiency of these constructions [BMR90, KS08, PSSW09, BHKR13, KMR14, ZRE15, GLNP15].

Garbled circuits, while tremendously useful in the two-party setting, when used in the multiparty setting lead to comparatively inferior solutions. For example, Yao's garbled circuits along with a two-round 1-out-of-2 oblivious transfer (OT) protocol [Rab81, AIR01, NP01, HK12] gives an easy solution to the problem of (semi-honest) two-round secure computation in the two-party setting. However, the same problem for the multiparty setting turns out to be much harder. Beaver, Micali and Rogaway [BMR90] show that garbled circuits can be used to realize a constant round multi-party computation protocol. However, unlike the two-party case, this protocol is not two rounds.

1.1 Garbled Protocols

In this paper, we introduce a generalization of Yao's construction from circuits to distributed protocols. We next elaborate on (i) what it means to garble a protocol, (ii) why this notion is interesting, and (iii) if we can realize this notion.

What does it mean to garble a protocol? Consider an arbitrary protocol Φ over n-parties $P_1, \ldots, P_n$ with inputs $x_1, \ldots, x_n$, respectively. Just as in garbled circuits, a garbled protocol construction allows each party P_i to independently generate a garbled protocol component $\widetilde{\Phi}_i$ along with input labels $\{\mathsf{lab}^i_{j,0}, \mathsf{lab}^i_{j,1}\}$. However, now the party P_i additionally generates an input encoding $\widetilde{x}_i$. Correctness requires that the set of all garbled protocol components $\{\widetilde{\Phi}_i\}_{i\in[n]}$ and the set of labels corresponding to the input encodings of all parties $\{\mathsf{lab}^i_{j,z_j}\}_{i\in[n],j\in[|z|]}$ where $z := \widetilde{x}_1\|\cdots\|\widetilde{x}_n$ can be used to generate the entire transcript of the protocol Φ. Detailing the security guarantee (for the semi-honest case), we require the existence of an efficient simulator Sim such that for any set $H \subseteq [n]$ of honest parties and inputs $\{x_i\}_{i\in[n]}$ of the parties we have that

$$\{\tilde{\Phi}_i, \{\mathsf{lab}^i_{j,z_j}\}, \widetilde{x}_i\}_{i\in[n]} \stackrel{c}{\approx} \mathsf{Sim}(H, \Phi(x_1, \ldots x_n), \{x_i\}_{i\notin H})$$

where $\stackrel{c}{\approx}$ denotes computational indistinguishability and $\Phi(x_1, \ldots, x_n)$ denotes the transcript of Φ.

Why consider Garbled Protocols? We illustrate the power of garbled protocols by showing how they can be used to realize a two-round (semi-honest) multiparty secure computation protocol. Looking ahead, our protocol is analogous to the construction of two-round, two party secure computation protocol using garbled circuits.

Take any n-party secure computation protocol Φ and let $x_1, \ldots, x_n$ be the respective inputs of the parties. Each party starts by independently generating $\left\{\widetilde{\Phi}_i, \{\mathsf{lab}^i_{j,0}, \mathsf{lab}^i_{j,1}\}, \widetilde{x}_i\right\}$. In the first round, each party distributes the generated values $\widetilde{x}_i$ to every other party. On receiving the first messages of all other parties, each party sends its second round message $\left(\widetilde{\Phi}_i, \{\mathsf{lab}^i_{j,z_j}\}\right)$ (with $z := \widetilde{x}_1\| \cdots \|\widetilde{x}_n$) to every other party. Finally, by correctness of garbled protocols we have that each party can locally execute the garbled protocol to obtain the output from the transcript $\Phi(x_1, \dots x_n)$. On the other hand, the security of the garbled protocols and Φ ensure that nothing else beyond the output is leaked.

Can we garble protocols? Our main result is a garbled protocols construction based on standard computational assumptions on bilinear maps [BF01, Jou04]. A bit more precisely:

Informal Theorem. *Assuming the subgroup decision assumption or the decision linear assumption on groups with bilinear maps there exists a garbled protocol construction with semi-malicious security (in the common reference/random string model).*[1]

We also show a modification of this construction such that it makes only black-box use of the underlying group and avoids any expensive non-black-box group operations.

1.2 Applications to Two-Round Multiparty Secure Computation

Using the above primitive, we obtain a general compiler that converts an arbitrary (polynomial) round (semi-honest) multi-party secure computation protocol into a two-round UC secure [Can01] protocol against static adversaries. Previously, such compilers [GGHR14, GLS15] were known under stronger computational assumptions such as indistinguishability obfuscation [BGI+01, GGH+13] or witness encryption [GGSW13].[2]

Furthermore, instantiating this compiler with any multi-party secure computation protocol (e.g., the one by Goldreich, Micali, and Wigderson [GMW87]) we obtain the first two-round multiparty computation protocol based on bilinear maps. Prior to this work, constructions of two-round multiparty computation protocols [MW16, PS16, BP16] were only known based on lattice assumptions such as the learning-with-errors [Reg05].[3] We also obtain the following extensions:

- *Black-Box Use of the Group:* With the goal of obtaining a two-round multiparty computation protocol that makes black-box use of the underlying cryptographic primitives, we modify our compiler from above. More specifically, building on the non-interactive OT protocol of Bellare and Micali [BM90] (based on the CDH assumption [DH76]), we obtain a compiler that converts any arbitrary round (malicious secure) protocol Φ^{OT} in the OT-hybrid model into a two-round UC secure protocol against static adversaries while only making black box use of the underlying group.

  Instantiating, this new compiler with an information theoretic protocol in the OT-hybrid model [Kil88, IPS08] yields a two-round multiparty computation protocol based on bilinear

[1] Semi-malicious security is a strengthening of semi-honest security where the parties follow the protocol but are allowed to choose an arbitrary string as its random tape.

[2] We note that the recent constructions of lockable obfuscation [GKW17, WZ17] based on standard assumptions such as learning with errors is insufficient to obtain such a compiler since these works assume that the lock value has some min-entropy.

[3] In two recent works, Boyle et al. [BGI16, BGI17] also obtain constructions of two-round multiparty computation based on DDH. However, their results are applicable only for the setting of constant number of parties — a special case of our result. Also, they assume the need for public-key infrastructure while we just assume a common random string.

maps while avoiding expensive non-black-box use of the underlying group.[4]

- *Semi-honest Protocol in Plain Model:* A simple modification in the construction of garbled protocols from Informal Theorem gives a construction with semi-honest security in the plain model. This readily gives a construction of two-round semi-honest secure MPC in the plain model from bilinear maps. Prior constructions required witness encryption to achieve the same result.

- *Extension to RAM programs:* Instantiating the above compilers with appropriate multi-party secure computation protocols for RAM programs [OS97, GKK+12], we also obtain the first two-round multiparty secure RAM computation protocol without first converting the RAM program to a circuit based on standard techniques [CR73, PF79].

  We note that the multi-key fully-homomorphic encryption [AJL+12, LTV12, CM15, MW16, PS16, BP16] based two-round secure computation techniques do not work for the setting of RAM programs. This is because fully-homomorphic encryption techniques need interaction for disclosing what locations are accessed by the oblivious RAM programs.[5] On the other hand, our use of garbled protocols does not suffer from this limitation.[6]

2 Technical Overview

At the heart of our garbled protocols construction is a simple but powerful extension of homomorphic proof commitments scheme. This primitive was first considered by Groth, Ostrovsky and Sahai [GOS06] who used it to realize a non-interactive zero-knowledge proof system based on bilinear maps. Below we start by (i) recalling GOS construction of homomorphic proof commitments, (ii) how we augment them, and (iii) use them to realize garbled protocols. Finally we give details on how to obtain two round, secure multiparty computation protocol making black-box use of the underlying group.

2.1 Starting Point: Homomorphic Proof Commitments

A homomorphic proof commitment scheme is a (non-interactive) commitment scheme com that supports homomorphic operations and provides some additional proof properties. In particular, it is additively homomorphic, i.e., $\mathrm{com}(b_0 + b_1; r_0 + r_1) = \mathrm{com}(b_0; r_0) \cdot \mathrm{com}(b_1; r_1)$ where the message space is over $\mathbb{Z}_p$. Furthermore, given a commitment $c = \mathrm{com}(b; r)$, the corresponding committed value b and randomness r, a prover can generate a NIZK proof proving that $b \in \{0, 1\}$ without leaking anything else about the value b.

GOS show that homomorophic proof commitments can be used to generate NIZK proofs for arbitrary NP-statements. This is done in two steps:

[4]However, unlike our non black-box protocol, the length of the common reference string of our black-box construction grows linearly with the number of parties.

[5]An oblivious RAM program is a RAM program compiled with an oblivious RAM scheme [Ost90, GO96].

[6]Another approach would be to use garbled RAM [LO13, GHL+14, GLOS15, GLO15]. However, those constructions suffer from the same limitation as Yao's garbled circuits in terms of supporting multiparty protocols. Specifically, garbled RAM can be used to construct two-round two-party secure computation protocol, but the multiparty protocol is only (larger than two) constant rounds [LO13, GGMP16].

1. First, GOS show that given three commitments $c_0 = \mathrm{com}(b_0; r_0), c_1 = \mathrm{com}(b_1; r_1)$, and $c_2 = \mathrm{com}(b_2; r_2)$ a prover given b_0, b_1, b_2 and r_0, r_1, r_2 can generate a NIZK proof proving that $b_2 = \mathsf{NAND}(b_0, b_1)$. This, in fact, can be done very simply by just proving that each one of b_0, b_1, b_2 and $b_0 + b_1 + 2b_2 - 2$ is in $\{0, 1\}$. In other words, the prover generates a proof showing that each one c_0, c_1, c_2 and $c_0 \cdot c_1 \cdot {c_2}^2 \cdot \mathrm{com}(-2; 0)$ is commitments to a value in $\{0, 1\}$. Looking at the table of a NAND gate (as GOS prove), it is not too hard to prove that these conditions are simultaneously satisfied if and only if values $b_2 = \mathsf{NAND}(b_0, b_1)$.

2. Using the above trick, Groth et al. provide NIZK proofs for arbitrary NP-statements by converting them to a circuit SAT instance. More specifically, given a circuit C composed entirely of NAND gates, a prover can prove that $\exists \mathsf{wit}$ such that $C(\mathsf{wit}) = 1$. The prover achieves this as follows: it commits to the value assigned to every wire of the circuit C on input wit and proves that (i) each of the committed values is in $\{0, 1\}$, (ii) each NAND gate in C has been computed correctly, and (iii) the output of the circuit is 1.

Now, we very briefly describe how the GOS construction works in the setting of composite order groups with bilinear maps. GOS commitments are generated with respect to a commitment key which can either be in the binding mode or in the hiding mode and keys generated in the two modes are computationally indistinguishable.[7] The commitment key ck consists of a description of a source group $\mathbb{G}$ (of order $n = pq$), a target group $\mathbb{G}_T$, a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ and a group element h. In the binding mode, h is chosen randomly from the subgroup[8] $\mathbb{G}_q$ and in the hiding mode h is chosen randomly from $\mathbb{G}$. The commitment keys in the two modes are indistinguishable from the sub-group decision assumption. The commitment c to a message $m \in \mathbb{Z}_p$ using randomness r is given by $g^m h^r$. When h is chosen randomly from $\mathbb{G}$, c information theoretically hides m and when h is chosen from the sub-group $\mathbb{G}_q$ there exists unique $(m, r) \in \mathbb{Z}_p \times \mathbb{Z}_n$ such that $c = g^m h^r$. The homomorphic property is easy to observe. The proof π certifying that c is a commitment to 0 or 1 is given by $(g^{2m-1}h^r)^r$. The verification procedure relies on fact that if c is of the form h^r or gh^r then either c or cg^{-1} have order 1 or q (when h is chosen in the binding mode). This is ensured by checking if $e(h, \pi) = e(c, cg^{-1})$.

2.2 New Technical Tool: Homomorphic Proof Commitments with Encryption

Armed with the above understanding of homomorphic proof commitments, we now explain how to augment them to support an encryption, decryption functionality. Specifically, an encryptor given a commitment c and a message msg can generate a ciphertext that can be efficiently decrypted using a proof π certifying the fact that c is a commitment to 0 or 1. Our security requirement is that if c is not a commitment to 0 or 1 then semantic security holds, i.e., for all $\mathsf{msg}, \mathsf{msg}'$ encryptions of msg are indistinguishable from encryptions of msg'. Note that if c is not a commitment to 0 or 1 then the prover cannot generate a proof certifying this fact. We call this primitive a homomorphic proof commitment with encryption. A careful reader might have noticed that the security provided by a homomorphic proof commitment with encryption is very similar to the security guarantee of a witness encryption [GGSW13]. Indeed, homomorphic proof commitment with encryption is a witness encryption scheme for a special language.

[7] In particular, the commitments generated using the binding key are perfectly binding whereas the ones generated using the hiding key are perfectly hiding.

[8] Recall that $\mathbb{G}_q$ is a sub-group of $\mathbb{G}$ with order q

Next, we describe how the above abstract notion can be realized. An elegant aspect of our work is that this augmentation to the homomorphic proof commitments of GOS can be done without changing their construction. The encryption procedure on input a commitment $c = g^m h^r$ and a message msg essentially outputs the ciphertext $(h^s, e(c^s, cg^{-1}) \cdot \mathsf{msg})$ for a randomly chosen $s \leftarrow \mathbb{Z}_n$.[9] To decrypt this ciphertext using a proof $\pi = (g^{2m-1}h^r)^r$, compute $e(h^s, \pi)$ and use it to unmask the message msg. The key idea while proving security is that when h is chosen in the binding mode, h^s "loses" some information about s — specifically, $s \mod p$ is uniformly distributed even given h^s. Furthermore, this entropy in s is transferred to the masking factor $e(c^s, cg^{-1}) = e(h^s, \pi)e(g, g)^{sm(m-1)}$ when m is not 0 or 1. This allows us to argue that the message msg remains hidden.

2.3 Realizing Garbled Protocols

In this subsection we highlight the key challenge in constructing garbled protocols for the multiparty setting and how homomorphic proof commitments with encryption can be used to overcome this barrier.

The key challenge. With the goal of explaining the challenge involved, we start by considering garbled protocols in the *easy* case of two parties. We will focus only on how P_1 generates its garbled protocol components as the components generated by P_2 will be analogous. For the case of two parties, P_1 can just garble the next message functions of the protocol Φ (using Yao's garbled circuits) and send them over to the P_2. The only issue with this approach is how does P_1's garbled next message functions read the messages generated by P_2 in the execution of Φ. A natural idea is to have P_2 commit to its input x_2 (and also its randomness in case Φ is a randomized protocol) in its input encoding $\tilde{x}_2$ which will then be hard-coded inside the garbled next-message functions. Next, P_1 can generate garblings of next message functions in a manner so that P_2 would be able to evaluate those garblings as long as it can prove to P_1's garbled circuit that it has been generating its own messages consistent with the committed input x_2. At a very high level this can be achieved by letting P_1's garbled next message functions output ciphertexts containing encryptions of certain labels that P_2 can decrypt only if it has been generating its own messages correctly.

However, the techniques from the literature for doing this based on standard assumptions involve P_2's secret state in the decryption step. Consequently, these techniques fail even for the three party setting because the third party, say, P_3 does not have access to P_2's secret state. Gordan et al. [GLS15] (building on Garg et al. [GGHR14]) observe that witness encryption [GGSW13] for NP can be used to solve this problem. The idea is: (i) P_1 outputs a witness encryption which allows decryption given just a NIZK proof certifying the correctness of computation, and (ii) P_2 outputs a proof for certifying this very fact. Next, using the proof, P_3 can decrypt P_1's ciphertext while secrecy of P_2's state is also maintained.

In this work, we show that the same intuition can be realized using homomorphic proof commitments with encryption. However, recall that homomorphic proof commitments with encryption are very weak. The encryption process cannot in "one-shot" verify that P_2 generated its messages correctly. Instead, our idea for this is that P_1 keeps P_2 on a "very tight leash," making sure that P_2 computes *every* NAND gate in the execution of Φ correctly.

[9]The actual construction uses a strong randomness extractor and we avoid this in the informal overview.

The rest of this subsection is organized as follows. (1) We start by making some assumptions on the structure of distributed protocol Φ. We note that these assumptions can be made without loss of generality. (2) Next, we give a garbling scheme for such structured protocols.

Structure of Φ. Let Φ be a n-party protocol. For the purposes of this informal overview, we will assume that Φ is deterministic. Let T be the round complexity of the protocol. We assume that each party P_i maintains a local state that is updated at the end of every round. The local state is a function of the input and the set of messages received from other parties.

At the beginning of the t^{th} round, every party P_i runs a program Φ_i on input t to obtain an output (i^*, f, g). [10] Here, i^* denotes the *active party* in round t. The active party P_{i^*} computes *one* NAND gate on a pair of bits of its state and writes the computed bit to its state. The inputs to the NAND gate are given by the bits in the indices f and g of the local state of P_{i^*}. Additionally, for a (pre-determined) subset of rounds $B_{i^*} \subseteq \{t \in [T] : (i^*, \cdot, \cdot) = \Phi_i(t)\}$, P_{i^*} outputs the computed bit to other parties. In this case, all the parties copy this bit to their state.

We note that any protocol can be compiled to follow this format at an additional cost of increasing the round complexity by a polynomial factor.

Garbling Scheme for Protocols. The garbled protocol component $\widetilde{\Phi}_i$ generated by P_i consists of a sequence of T garbled circuits and a set of labels for evaluating the first garbled circuit in the sequence. These garbled circuits have a special structure, namely, the t^{th} garbled circuit in the sequence outputs the labels for evaluating the $(t+1)^{th}$ garbled circuit and thus starting from the first garbled circuit we can execute every garbled circuit in the sequence. At a high level, the t^{th} garbled circuit corresponds to the computation done by party P_i in the t^{th} round of the protocol Φ. In a bit more details, the t^{th} garbled circuit takes as input the local state obtained after the first $t-1$ rounds, updates the local state and outputs the labels corresponding to the updated state for evaluating the next garbled circuit. This ensures that at the end of the T^{th} evaluation, we can obtain the transcript of the protocol from the final local state of party P_i. The encoding of an input x_i is given by a set of homomorphic commitments $\{c_{i,k}\}$ to each individual bit of the input x_i.

To look a bit more closely into the working of the t^{th} garbled circuit, let us assume that P_i is the active party in the t^{th} round. Our assumption on the structure of Φ implies that in the t^{th} round, P_i has to update its local state by computing a NAND of two bits in its current state and write the output to a specific location. Further, if $t \in B_i$, P_i has to communicate this bit to the other parties and the other parties have to copy this bit to their state. In particular, this means that the labels output by the t^{th} garbled circuit in every other protocol component $\widetilde{\Phi}_j$ for $j \neq i$ must reflect this communicated bit. The main technical challenge we solve is in designing a non-interactive method to realize this communication and also ensure at the same time that P_i computes *each* NAND gate correctly. This is done using homomorphic proof commitment with encryption. Let us start with a method to realize the communication.

Recall that by our assumption on Φ, the updated state of every party can only be one of two choices. This choice is determined by the output of the NAND computation done by the active party. Let the NAND computation done in round t take as input the bits in positions f and g of the local state of party P_i. For simplicity, let us assume that f, g correspond to indices where the input of P_i is written. Let d be a commitment to 0 using some fixed randomness (known to all parties)

[10] We assume that $\Phi_1(t) = \Phi_2(t) = \ldots = \Phi_n(t)$ for every $t \in [T]$.

and let $\overline{d}$ be a commitment to 1 (again using some fixed randomness). Applying the GOS trick, we deduce that if the output of the NAND computation is 0 then $e_0 = c_{i,f} \cdot c_{i,g} \cdot d^2 \cdot \mathrm{com}(-2; 0)$ is a commitment to $\{0,1\}$; else $e_1 = c_{i,f} \cdot c_{i,g} \cdot \overline{d}^2 \cdot \mathrm{com}(-2; 0)$ is a commitment to $\{0,1\}$. Now, we let every other garbled protocol component $\widetilde{\Phi}_j$ for $j \neq i$ output two zero-one encryptions: one under the commitment e_0 containing the set of labels of the updated state assuming that the communicated bit is 0; and the other under the commitment e_1 assuming that the communicated bit is 1. The active party outputs a zero-one proof that either e_0 or e_1 is a commitment to a message in $\{0,1\}$. Using this proof, every party can recover the correct set of labels corresponding to the updated state. This approach of letting a garbled circuit output a witness encryption of the labels of the next garbled circuit in a sequence is inspired by [**?**].

Note that the above described solution reveals the output of the NAND gate in the clear to the other parties. This is necessary for the case where the bit is communicated to other parties but is undesirable if the NAND is an internal computation as it might reveal some information about the secret state of party P_i. On the contrary, every other party must somehow ensure that P_i computes this NAND gate correctly. We solve this problem by augmenting the input encoding with a commitment to a string of random bits i.e., the input encoding will be a homomorphic commitment to every bit of $x_i \| r_i$ where r_i is a random string. To prove that an internal NAND computation is done correctly, the active party P_i generates a zero-one proof that either $e_0 = c_{i,f} \cdot c_{i,g} \cdot d^2 \cdot \mathrm{com}(-2; 0)$ or $e_1 = c_{i,f} \cdot c_{i,g} \cdot \overline{d} \cdot \mathrm{com}(-2; 0)$ is a commitment to $\{0,1\}$ where d is now a commitment to a random bit generated as a part of the input encoding. $\overline{d}$ denotes the commitment to the flipped bit. Now, a proof that either e_0 or e_1 contains a commitment to $\{0,1\}$ reveals the output of the NAND computation masked with the random bit committed in d and hence completely hides the output. Note that the homomorphic property of the commitment scheme enables every party to efficiently generate $\overline{d}$. A downside of this approach is that the size of the input encoding grows with the round complexity of Φ. But using techniques from the recent work of Cho et al. [**?**], we can make the size of the input encoding succinct i.e., grow only with the size of the input. We won't delve into the details.

2.4 Black-Box Two-Round MPC

Instantiating the above garbled protocols construction with a semi-honest secure Φ, we obtain a two-round multiparty computation protocol based on bilinear maps.[11] However, the protocol makes non-black box use of the underlying homomorphic proof commitment with encryption as well as cryptographic operations that Φ might invoke. In this subsection, we explain how to obtain a two-round MPC protocol by making black-box use of a homomorphic proof commitment with encryption as well as a DDH hard group.

Designing a protocol that makes black-box use of a homomorphic proof commitment with encryption is somewhat straightforward. We observe that the proofs and the ciphertexts computed within the garbled circuit can in fact be precomputed and hardwired in its description. Later, the garbled circuit chooses the appropriate pre-computed values based on its inputs. We note that this pre-computation is possible because the output of each garbled circuit depends only on a constant number of bits in its input.

[11] For technical reasons, we need the protocol Φ to be semi-malicious [AJL+12]. The semi-malicious security is a generalization of semi-honest security where the adversary is still restricted to follow the protocol but can choose its random coins arbitrarily. Note that the protocol described in [GMW87] is semi-maliciously secure.

We now explain how to obtain a protocol that makes black-box use of cryptographic operations invoked by Φ.

Suppose Φ was an information theoretic secure MPC then the compiled protocol already makes black-box use of the underlying cryptographic primitives. But information theoretic secure MPC protocols can exist only if a majority of the parties are honest [BOGW88] and secure channels are present between every pair of parties. However, the situation in the OT hybrid model is different. There exist constructions of information theoretic protocols tolerating dishonest majority and malicious behavior [Kil88, IPS08] in the OT hybrid model. We will be using such a protocol to design our black-box two round MPC.

Let Φ be an information theoretic secure protocol in the OT hybrid model tolerating malicious behavior. At a high level, our black-box two round MPC protocol generates OT correlations [12] in the first round and later hardwires these correlations in the garbled circuits to enable Φ perform information theoretic OTs. We now explain how to generate such OT correlations building on the non-interactive oblivious transfer by Bellare and Micali [BM90].

Let us first recall the OT protocol of Bellare and Micali in the common random string model. The crs consists of a random group element X. The sender samples a random exponent a and computes $A := g^a$ and sends it over to the receiver. The receiver samples a random exponent b and computes $B := g^b$. It then samples a random bit c and computes $C_0 := (1-c)B + c(\frac{X}{B})$ and $C_1 := cB + (1-c)(\frac{X}{B})$ and sends them over to A. Notice that by construction of C_0 and C_1, B knows the discrete log of C_c. The sender on receiving C_0 and C_1 sets the two random strings (s_0, s_1) to be (C_0^a, C_1^a) and the receiver sets (c, s_c) to be (c, A^b). Note that assuming the DDH assumption, the other string s_{1-c} is indistinguishable to a randomly distributed string. Building on this protocol and additionally using Groth-Sahai [GS12] proofs to obtain malicious security, we obtain a two round MPC protocol making black-box use a homomorphic proof commitment with encryption and a DDH hard group. [13]

3 Preliminaries

This section is devoted to recalling some well studied notions that we will need in this paper. Let λ denote the security parameter. A function $\mu(\cdot) : \mathbb{N} \to \mathbb{R}^+$ is said to be negligible if for any polynomial $\mathsf{poly}(\cdot)$ there exists λ_0 such that for all $\lambda > \lambda_0$ we have $\mu(\lambda) < \frac{1}{\mathsf{poly}(\lambda)}$. For a probabilistic algorithm A, we denote $A(x; r)$ to be the output of A on input x with the content of the random tape being r. When r is omitted, $A(x)$ denotes a distribution. For a finite set S, we denote $x \leftarrow S$ as the process of sampling x uniformly from the set S. We will use PPT to denote Probabilistic Polynomial Time algorithm. We denote $[k]$ to be the set $\{1, \ldots, k\}$ and $[a, b]$ to be the set $\{a, a+1, \ldots, b\}$ for $a \leq b$ and $a, b \in \mathbb{Z}$. For a binary string $x \in \{0,1\}^n$ we will denote the i^{th} bit of x by x_i. We assume without loss of generality that the length of the random tape used by all cryptographic algorithms is λ. We will use $\mathsf{negl}(\cdot)$ to denote an unspecified negligible function and $\mathsf{poly}(\cdot)$ to denote an unspecified polynomial function.

[12] Recall that OT correlations consists of a random pair of strings (s_0, s_1) provided to the sender and a pair (c, s_c) where c is a random bit provided to the receiver.

[13] We have been a little imprecise in this overview. In order to use Groth-Sahai proofs we cannot rely on DDH assumption as GS proofs assume the existence of an efficiently computable bilinear map. In the actual construction we assume CDH is hard.

3.1 Garbled Circuits

Below we recall the definition of garbling scheme for circuits [Yao82] (see Lindell and Pinkas [LP09] and Bellare et al. [BHR12] for a detailed proof and further discussion). A garbling scheme for circuits is a tuple of PPT algorithms $(\mathsf{GarbleCkt}, \mathsf{EvalCkt})$. Very roughly, $\mathsf{GarbleCkt}$ is the circuit garbling procedure and $\mathsf{EvalCkt}$ the corresponding evaluation procedure. More formally:

- $(\widetilde{\mathsf{C}}, \{\ell_{w,b}\}_{w\in\mathsf{inp}(C),b\in\{0,1\}}) \leftarrow \mathsf{GarbleCkt}\left(1^\lambda, C\right)$: $\mathsf{GarbleCkt}$ takes as input a security parameter λ, a circuit C, and outputs a *garbled circuit* $\widetilde{\mathsf{C}}$ along with labels $\ell_{w,b}$ where $w \in \mathsf{inp}(C)$ ($\mathsf{inp}(C)$ is the set of input wires to the circuit C) and $b \in \{0,1\}$.

- $y \leftarrow \mathsf{EvalCkt}\left(\widetilde{\mathsf{C}}, \{\ell_{w,x_w}\}_{w\in\mathsf{inp}(C)}\right)$: Given a garbled circuit $\widetilde{\mathsf{C}}$ and a sequence of input labels $\{\ell_{w,x_w}\}_{w\in\mathsf{inp}(C)}$ (referred to as the garbled input), $\mathsf{EvalCkt}$ outputs a string y.

Correctness. For correctness, we require that for any circuit C and input $x \in \{0,1\}^{|\mathsf{inp}(C)|}$ we have that:

$$\Pr\left[C(x) = \mathsf{EvalCkt}\left(\widetilde{\mathsf{C}}, \{\ell_{w,x_w}\}_{w\in\mathsf{inp}(C)}\right)\right] = 1$$

where $(\widetilde{\mathsf{C}}, \{\ell_{w,b}\}_{w\in\mathsf{inp}(C),b\in\{0,1\}}) \leftarrow \mathsf{GarbleCkt}\left(1^\lambda, C\right)$.

Security. For security, we require that there exists a PPT simulator Sim such that for any circuit C and input $x \in \{0,1\}^{|\mathsf{inp}(C)|}$, we have that

$$\left(\widetilde{\mathsf{C}}, \{\ell_{w,x_w}\}_{w\in\mathsf{inp}(C)}\right) \stackrel{c}{\approx} \mathsf{Sim}\left(1^\lambda, C(x)\right)$$

where $(\widetilde{\mathsf{C}}, \{\ell_{w,b}\}_{w\in\mathsf{inp}(C),b\in\{0,1\}}) \leftarrow \mathsf{GarbleCkt}\left(1^\lambda, C\right)$ and $\stackrel{c}{\approx}$ denotes that the two distributions are computationally indistinguishable.

3.2 Cryptographic Assumptions

We will state the cryptographic assumptions used in this paper.

Sub-Group Decision Assumption [BGN05]. The presentation here follows the notation given in [GOS12]. Let $\mathcal{G}_{\mathrm{BGN}}$ be a randomized algorithm that on security parameter λ outputs $(p, q, \mathbb{G}, \mathbb{G}_T, e, g)$ such that

- p, q are primes with $p < q$
- $\mathbb{G}, \mathbb{G}_T$ are descriptions of cyclic groups of order $n = pq$
- $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a bilinear map, i.e., $\forall u, v \in \mathbb{G}\ \forall a, b \in \mathbb{Z} : e(u^a, v^b) = e(u, v)^{ab}$
- g is a random generator for $\mathbb{G}$ and $e(g, g)$ generates $\mathbb{G}_T$
- Group operations, deciding group membership and the bilinear map are efficiently computable.

Let $\mathbb{G}_q$ be the subgroup of $\mathbb{G}$ of order q. The subgroup decision problem is to distinguish elements of $\mathbb{G}$ from elements of $\mathbb{G}_q$.

Definition 3.1 *The subgroup decision assumption holds for generator* $\mathcal{G}_{\mathrm{BGN}}$ *if for any non-uniform polynomial time adversary* $\mathcal{A}$ *we have*

$$\begin{aligned} &\Pr\left[(p,q,\mathbb{G},\mathbb{G}_T,e,g)\leftarrow\mathcal{G}_{\mathrm{BGN}}(1^\lambda); n=pq; r\leftarrow\mathbb{Z}_n^*; h=g^r:\mathcal{A}(n,\mathbb{G},\mathbb{G}_T,e,g,h)=1\right]\\ \overset{c}{\approx}\ &\Pr\left[(p,q,\mathbb{G},\mathbb{G}_T,e,g)\leftarrow\mathcal{G}_{\mathrm{BGN}}(1^\lambda); n=pq; r\leftarrow\mathbb{Z}_q^*; h=g^{pr}:\mathcal{A}(n,\mathbb{G},\mathbb{G}_T,e,g,h)=1\right]. \end{aligned}$$

Computational Diffie-Hellman Assumption [DH76]. Let $\mathsf{Setup}_{\mathrm{CDH}}$ be a randomized algorithm that takes a security parameter as input and outputs $(p,\mathbb{G},\mathbb{G}_T,e,g)$ such that

- p is a prime
- $\mathbb{G},\mathbb{G}_T$ are descriptions of groups of order p
- $e:\mathbb{G}\times\mathbb{G}\to\mathbb{G}_T$ is a bilinear map, i.e., $\forall u,v\in\mathbb{G}\ \forall a,b\in\mathbb{Z}: e(u^a,v^b)=e(u,v)^{ab}$
- g is a random generator of $\mathbb{G}$ and $e(g,g)$ generates $\mathbb{G}_T$
- Deciding group membership, group operations and the bilinear map are all efficiently computable.

Definition 3.2 (Computational Diffie-Hellaman Assumption) *We say the computational Diffie-Hellman holds for the bilinear group generator* $\mathsf{Setup}_{\mathrm{CDH}}$ *if for all non-uniform polynomial time adversaries* $\mathcal{A}$ *we have*

$$\Pr\left[(p,\mathbb{G},\mathbb{G}_T,e,g)\leftarrow\mathsf{Setup}_{\mathrm{CDH}}(1^\lambda); x,y\leftarrow\mathbb{Z}_p^*:\mathcal{A}(p,\mathbb{G},\mathbb{G}_T,e,g,g^x,g^y)=g^{xy}\right]\leq\mathsf{negl}(\lambda)$$

Decision Linear Assumption [BB04]. The presentation here follows the same notation given in [GOS12]. Let $\mathcal{G}_{\mathrm{DLIN}}$ be a randomized algorithm that takes a security parameter as input and outputs $(p,\mathbb{G},\mathbb{G}_T,e,g)$ such that

- p is a prime
- $\mathbb{G},\mathbb{G}_T$ are descriptions of groups of order p
- $e:\mathbb{G}\times\mathbb{G}\to\mathbb{G}_T$ is a bilinear map, i.e., $\forall u,v\in\mathbb{G}\ \forall a,b\in\mathbb{Z}: e(u^a,v^b)=e(u,v)^{ab}$
- g is a random generator of $\mathbb{G}$ and $e(g,g)$ generates $\mathbb{G}_T$
- Deciding group membership, group operations and the bilinear map are all efficiently computable.

Definition 3.3 (Decisional Linear Assumption) *We say the decisional linear assumption holds for the bilinear group generator* $\mathcal{G}_{\mathrm{DLIN}}$ *if for all non-uniform polynomial time adversaries* $\mathcal{A}$ *we have*

$$\begin{aligned} &\Pr\left[(p,\mathbb{G},\mathbb{G}_T,e,g)\leftarrow\mathcal{G}_{\mathrm{DLIN}}(1^k); x,y\leftarrow\mathbb{Z}_p^*; r,s\leftarrow\mathbb{Z}_p:\mathcal{A}(p,\mathbb{G},\mathbb{G}_T,e,g,g^x,g^y,g^{xr},g^{ys},g^{r+s})=1\right]\\ \approx\ &\Pr\left[(p,\mathbb{G},\mathbb{G}_T,e,g)\leftarrow\mathcal{G}_{\mathrm{DLIN}}(1^k); x,y\leftarrow\mathbb{Z}_p^*; r,s,d\leftarrow\mathbb{Z}_p:\mathcal{A}(p,\mathbb{G},\mathbb{G}_T,e,g,g^x,g^y,g^{xr},g^{ys},g^d)=1\right]. \end{aligned}$$

3.3 Homomorphic Proof Commitments

In this subsection we recall the definition of homomorphic proof commitments from Groth et al. [GOS12] for realizing non-interactive zero-knowledge proofs. Much of the description below has been taken verbatim from Groth et al. [GOS12]. We keep the notation identical to Groth et al. [GOS12, Section 3] for the sake of a reader familiar with Groth et al. [GOS12].

A homomorphic proof commitment scheme is a non-interactive commitment scheme with some special properties that we define below. Recall first that in a non-interactive commitment scheme there is a key generator, which generates a public commitment key ck. The commitment key ck defines a message space $\mathcal{M}_{ck}$, a randomizer space $\mathcal{R}_{ck}$ and a commitment space $\mathcal{C}_{ck}$. We will require that the key generation algorithm is probabilistic polynomial time and outputs keys of length $\theta(\lambda)$. It will in general be obvious which key we are using, so we will sometimes omit it in our notation. There is an efficient commitment algorithm com that takes as input the commitment key, a message and a randomizer and outputs a commitment, $c = \mathrm{com}(m; r)$. We call (m, r) an opening of c.

The commitment scheme must be binding and hiding. Binding means that it is infeasible to find two openings with different messages of the same commitment. Hiding means that given a commitment it is infeasible to guess which message is inside the commitment. We want a commitment scheme that has two different flavors of keys. The commitment key can be perfectly binding, in which case a valid commitment uniquely defines one possible message. Alternatively, the commitment key can be perfectly hiding, in which case the commitment reveals no information whatsoever about the message. We require that these two kinds of keys are computationally indistinguishable.

We will consider commitments, where both the message space $(\mathcal{M}, +, 0)$, the randomizer space $(\mathcal{R}, +, 0)$ and the commitment space $(\mathcal{C}, \cdot, 1)$ are finite abelian groups. The commitment scheme should be homomorphic, *i.e.*, for all messages and randomizers we have

$$\mathrm{com}(m_1 + m_2; r_1 + r_2) = \mathrm{com}(m_1; r_1)\mathrm{com}(m_2; r_2).$$

We will require that the message space has a generator 1, and also that it has at least order 4. The property that sets homomorphic proof commitments apart from other homomorphic commitments, is that there is a way to prove that a commitment contains a message belonging $\{0, 1\}$. More precisely, if the key is of the perfect binding type, then it is possible to prove that there exists an opening $(m, r) \in \{0, 1\} \times \mathcal{R}$. On the other hand, if it is a perfect hiding key, then the proof will be perfectly witness-indistinguishable, *i.e.*, it is impossible to tell whether the message is 0 or 1.

Homomorphic proof commitment. We say that $(K_{\text{binding}}, K_{\text{hiding}}, \mathrm{com}, \mathrm{Topen}, P_{01}, V_{01})$ is a homomorphic proof commitment scheme if it satisfies the following properties for all non-uniform polynomial time adversaries $\mathcal{A}$.

Key indistinguishability:

$$\Pr\Big[(ck, xk) \leftarrow K_{\text{binding}}(1^\lambda) : \mathcal{A}(ck) = 1\Big] \approx \Pr\Big[(ck, tk) \leftarrow K_{\text{hiding}}(1^k) : \mathcal{A}(ck) = 1\Big].$$

Homomorphic property:

$$\Pr\Big[\mathrm{mode} \leftarrow \{\mathrm{binding}, \mathrm{hiding}\}; (ck, *) \leftarrow K_{\mathrm{mode}}(1^\lambda) : \\ \forall (m_1, r_1), (m_2, r_2) \in \mathcal{M} \times \mathcal{R} : \mathrm{com}(m_1 + m_2; r_1 + r_2) = \mathrm{com}(m_1; r_1)\mathrm{com}(m_2; r_2)\Big] = 1.$$

Perfect binding:

$$\Pr\Big[(ck, xk) \leftarrow K_{\text{binding}}(1^\lambda) : \\ \exists (m_1, r_1), (m_2, r_2) \in \mathcal{M} \times \mathcal{R} \text{ such that } m_1 \neq m_2 \text{ and } \text{com}(m_1; r_1) = \text{com}(m_2; r_2)\Big] = 0.$$

Perfect extractability: We say the commitment scheme has perfect extractability if there is a polynomial time extraction algorithm Ext such that

$$\Pr\Big[(ck, xk) \leftarrow K_{\text{binding}}(1^k) : \forall (m, r) \in \{0,1\} \times \mathcal{R} : \text{Ext}_{xk}(\text{com}(m; r)) = m\Big].$$

Perfect trapdoor opening:

$$\Pr\Big[(ck, tk) \leftarrow K_{\text{hiding}}(1^\lambda); (m_1, m_2) \leftarrow \mathcal{A}(ck); r_1 \leftarrow \mathcal{R}; r_2 \leftarrow \text{Topen}_{tk}(m_1, r_1, m_2) : \\ \text{com}(m_1; r_1) = \text{com}(m_2; r_2) \text{ if } m_1, m_2 \in \mathcal{M}\Big] = 1.$$

Perfect trapdoor opening indistinguishability:

$$\Pr\Big[(ck, tk) \leftarrow K_{\text{hiding}}(1^k); (m_1, m_2) \leftarrow \mathcal{A}(ck); r_1 \leftarrow \mathcal{R}; r_2 \leftarrow \text{Topen}_{tk}(m_1, r_1, m_2) : \\ m_1, m_2 \in \mathcal{M} \text{ and } \mathcal{A}(r_2) = 1\Big] \\ = \Pr\Big[(ck, tk) \leftarrow K_{\text{hiding}}(1^k); (m_1, m_2) \leftarrow \mathcal{A}(ck); r_2 \leftarrow \mathcal{R} : m_1, m_2 \in \mathcal{M} \text{ and } \mathcal{A}(r_2) = 1\Big].$$

Perfect completeness:

$$\Pr\Big[\text{mode} \leftarrow \{\text{binding}, \text{hiding}\}; (ck, *) \leftarrow K_{\text{mode}}(1^\lambda); (m, r) \leftarrow \mathcal{A}(ck); \pi \leftarrow P_{01}(ck, m, r) : \\ V_{01}(ck, \text{com}(m; r), \pi) = 1 \text{ if } (m, r) \in \{0,1\} \times \mathcal{R}\Big] = 1.$$

Perfect soundness:

$$\Pr\Big[(ck, xk) \leftarrow K_{\text{binding}}(1^k); (c, \pi) \leftarrow \mathcal{A}(ck) : \\ \exists (m, r) \in \{0,1\} \times \mathcal{R} \text{ so } c = \text{com}(m; r) \;\text{ if }\; V_{01}(ck, c, \pi) = 1\Big] = 1.$$

Perfect witness indistinguishability:

$$\Pr\Big[(ck, tk) \leftarrow K_{\text{hiding}}(1^\lambda); (r_0, r_1) \leftarrow \mathcal{A}(ck); \pi \leftarrow P_{01}(ck, 0, r_0) : \\ r_0, r_1 \in \mathcal{R} \text{ and } \text{com}(0; r_0) = \text{com}(1; r_1) \text{ and } \mathcal{A}(\pi) = 1\Big] \\ = \Pr\Big[(ck, tk) \leftarrow K_{\text{hiding}}(1^\lambda); (r_0, r_1) \leftarrow \mathcal{A}(ck); \pi \leftarrow P_{01}(ck, 1, r_1) : \\ r_0, r_1 \in \mathcal{R} \text{ and } \text{com}(0; r_0) = \text{com}(1; r_1) \text{ and } \mathcal{A}(\pi) = 1\Big].$$

Remark 3.4 *Groth et al. [GOS12] in their definition of homomorphic proof commitments also define more properties such as perfect non-erasure witness indistinguishability. We do not need these properties and skip defining them here.*

4 Homomorphic Proof Commitments with Encryption

In this section we provide definitions of homomorphic proof commitments with encryption – namely, a homomorphic proof commitment scheme with some additional encryption and decryption functionality. We then give constructions of this primitive based on the sub-group decision and the decision linear assumptions.

4.1 The Definition

$(K_{\text{binding}}, K_{\text{hiding}}, \text{com}, \text{Topen}, P_{01}, V_{01}, E_{01}, D_{01})$ is a homomorphic proof commitments with encryption if $(K_{\text{binding}}, K_{\text{hiding}}, \text{com}, \text{Topen}, P_{01}, V_{01})$ is homomorphic proof commitment and E_{01}, D_{01} are PPT algorithms such that E_{01} on input a commitment key ck, a commitment $c = \text{com}(ck, m; r)$ and a message msg outputs a ciphertext ct and D_{01} given ck, the commitment c, the ciphertext ct and a proof π such that $V_{01}(ck, c, \pi) = 1$ outputs the encrypted message msg. In other words, given a proof π such that c is a commitment to a message in $\{0, 1\}$, we can decrypt the ciphertext ct. Formally, we require that E_{01} and D_{01} satisfy the following correctness and security properties.

Perfect Correctness. For any ck (in the support of $K_{\text{binding}}, K_{\text{hiding}}$), $m \in \{0, 1\}$, randomness r, and proof π generated by $P_{01}(ck, m, r)$ and message msg,

$$\Pr\Big[\mathsf{ct} \leftarrow E_{01}(ck, \text{com}(ck, m; r), \mathsf{msg}) \wedge D_{01}(ck, \mathsf{ct}, \pi) = \mathsf{msg}\Big] = 1.$$

Statistical Semantic-Security. For all (possibly unbounded) adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,

$$\Pr\Big[(ck, \cdot) \leftarrow K_{\text{binding}}(1^\lambda); (c, \mathsf{msg}_0, \mathsf{msg}_1, \mathsf{st}) \leftarrow \mathcal{A}_1(ck); b \leftarrow \{0, 1\}; \mathsf{ct} \leftarrow E_{01}(ck, c, \mathsf{msg}_b) : \mathcal{A}_2(ck, \mathsf{ct}, \mathsf{st}) = b \wedge \exists\, m \notin \{0, 1\}, r \in \mathcal{R} \text{ such that } c = \text{com}(ck, m; r)\Big] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

We say that scheme has *computational* semantic-security if the above requirement holds only against PPT $\mathcal{A}$.

Remark 4.1 *We note that homomorphic proof commitment with encryption is essentially a witness encryption [GGSW13] scheme for a special language.*

4.2 Construction from Sub-group Decision Assumption

In this subsection we give a construction of homomorphic proof commitment with encryption from the sub-group decision assumption.

At a high level, our construction $(K_{\text{binding}}, K_{\text{hiding}}, \text{com}, P_{01}, V_{01}, E_{01}, D_{01})$ is obtained by supplementing the homomorphic proof commitment scheme of Groth et al. [GOS12], namely $(K_{\text{binding}}, K_{\text{hiding}}, \text{com}, P_{01}, V_{01})$ with encryption E_{01} and decryption D_{01} operations. Below we start by recalling the Groth et al. construction and then explain how to supplement it with encryption and decryption operations.

The Groth et al. construction [GOS12, Section 4]. We describe the Groth et al. construction informally. A complete description (taken verbatim from [GOS12]) is provided in Figure 1.

The commitment key ck consists of a description of a source group $\mathbb{G}$ (of order $n = pq$), a target group $\mathbb{G}_T$, a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ and a group element h. In the binding mode, h is chosen randomly from the subgroup[14] $\mathbb{G}_q$ and in the hiding mode h is chosen randomly from $\mathbb{G}$. The commitment keys in the two modes are indistinguishable from the sub-group decision assumption. The commitment c to a message $m \in \mathbb{Z}_p$ using randomness r is given by $g^m h^r$. When h is chosen randomly from $\mathbb{G}$, c information theoretically hides m and when h is chosen from the sub-group $\mathbb{G}_q$ there exists unique $(m, r) \in \mathbb{Z}_p \times \mathbb{Z}_n$ such that $c = g^m h^r$. The homomorphic property is easy to observe. The proof π certifying that c is a commitment to 0 or 1 is given by $(g^{2m-1}h^r)^r$. The verification procedure relies on fact that if c is of the form h^r or gh^r then either c or cg^{-1} have order 1 or q (when h is chosen in the binding mode). This is ensured by checking if $e(h, \pi) = e(c, cg^{-1})$.

Supplemental Encryption and Decryption. We now describe the encryption and decryption procedures that we supplement homomorphic proof commitment. The encryption procedure on input a commitment-key ck, a commitment c and a message msg essentially outputs the ciphertext $(h^s, e(c^s, cg^{-1}) \cdot \mathsf{msg})$ for a randomly chosen $s \leftarrow \mathbb{Z}_n$.[15] To decrypt this ciphertext using a proof π, D_{01} computes $e(h^s, \pi)$ and use it to unmask the message msg. The key idea while proving security is that when h is chosen in the binding mode, h^s "loses" some information about s. This is later used to show that $e(c^s, cg^{-1})$ masks the message when c is not a commitment to 0 or 1.

The formal description is provided in Figure 2. The construction uses a $(\log p, \mathsf{negl}(\lambda))$-strong randomness extractor $\text{RandExt} : \mathbb{G} \times \{0,1\}^* \to \{0,1\}^{\frac{\log p}{2}}$.

Lemma 4.2 *Assuming the subgroup decision assumption, the construction described in Figures 1 and 2 is a homomorphic proof commitment with encryption.*

Proof We note that $(K_{\text{binding}}, K_{\text{hiding}}, \text{com}, P_{01}, V_{01}, \text{Ext})$ is a homomorphic proof commitment scheme as argued by Groth et al. [GOS12]. We now prove that (E_{01}, D_{01}) satisfy perfect correctness and statistical semantic-security.

Perfect Correctness. Let $c = \text{com}(ck, m; r)$ where $m \in \{0, 1\}$. Let $\mathsf{ct} = (v, h^s, \text{RandExt}(v, e(c^s, cg^{-1})) \oplus \mathsf{msg})$ and $\pi = (g^{2m-1}h^r)^r$. To prove correctness it is sufficient to show that $e(h^s, \pi) = e(c^s, cg^{-1})$.

$$\begin{aligned}
e(c^s, cg^{-1}) &= e(g^m h^r, g^{m-1} h^r)^s \\
&= e(g,g)^{sm(m-1)} e(h,g)^{sr(m-1)} e(g,h)^{smr} e(h,h)^{sr^2} \\
&= e(h,g)^{sr(m-1)} e(g,h)^{smr} e(h,h)^{sr^2} \qquad (\text{Since } m \in \{0,1\}) \\
&= e(h,g)^{sr(2m-1)} e(h,h)^{sr^2} \\
&= e(h^s, (g^{2m-1}h^r)^r) \\
&= e(h^s, \pi)
\end{aligned}$$

[14]Recall that $\mathbb{G}_q$ is a sub-group of $\mathbb{G}$ with order q

[15]The actual construction in Figure 2 uses a (strong) randomness extractor to extract random bits from $e(c^s, cg^{-1})$ and then use it to mask the message. We avoid this in the informal overview.

Perfectly binding key generation $K_{\text{binding}}(1^k)$**:**

1. $(p, q, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}_{\text{BGN}}(1^k)$. Let $n = pq$.
2. Sample $x \leftarrow \mathbb{Z}_q^*$ and compute $h = g^{px}$.
3. Let $ck = (n, \mathbb{G}, \mathbb{G}_T, e, g, h)$.
4. Let $xk = (ck, q)$.
5. Return (ck, xk).

Perfectly hiding key generation $K_{\text{binding}}(1^k)$**:**

1. $(p, q, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}_{\text{BGN}}(1^k)$. Let $n = pq$.
2. $x \leftarrow \mathbb{Z}_n^*$ and compute $h = g^x$.
3. Let $ck = (n, \mathbb{G}, \mathbb{G}_T, e, g, h)$.
4. Let $tk = (ck, x)$
5. Return (ck, tk)

Commitment $\text{com}_{ck}(m)$**:**

The key ck defines message space $\mathbb{Z}_p$, randomizer space $\mathbb{Z}_n$ and commitment space $\mathbb{G}$. To commit to message $m \in \mathbb{Z}_p$ do

1. $r \leftarrow \mathbb{Z}_n$
2. Return $\text{com}_{ck}(m; r) = g^m h^r$

WI proof $P_{01}(ck, m, r)$**:**

Given $(m, r) \in \{0, 1\} \times \mathbb{Z}_n$ we make the WI proof for commitment to 0 or 1 as $\pi = (g^{2m-1}h^r)^r$.

Verification $V_{01}(ck, c, \pi)$**:**

To verify a WI proof π of commitment c containing 0 or 1, check $e(c, cg^{-1}) = e(h, \pi)$.

Extraction $\text{Ext}_{xk}(c)$**:**

On a perfect binding key we can use $xk = (ck, q)$ to extract m of length $\mathcal{O}(\log k)$ from $c = g^m h^r$ as follows. Compute $c^q = (g^m h^r)^q = (g^q)^m$ and exhaustively search for m.

Trapdoor opening $\text{Topen}_{tk}(m, r, m')$**:**

Given a commitment $c = g^m h^r$ under a perfectly hiding commitment key we have $c = g^{m'} h^{r-(m'-m)/x}$. So we can create a perfectly hiding commitment and open it to any value we wish if we have the trapdoor key $tk = (ck, x)$. The trapdoor opening algorithm returns $r' = r - \frac{(m'-m)}{x} \bmod n$.

Figure 1: Homomorphic Proof Commitment from sub-group decision taken verbatim from [GOS12]

Encrypt $E_{01}(ck, c, \mathsf{msg})$**:** To encrypt $\mathsf{msg} \in \{0,1\}^{\frac{\log p}{2}}$,

1. Choose $s \leftarrow \mathbb{Z}_n$.
2. Choose $v \leftarrow \{0,1\}^*$ as the seed of RandExt.
3. Output $(v, h^s, \mathrm{RandExt}(v, e(c^s, cg^{-1})) \oplus \mathsf{msg})$.

Decrypt $D_{01}(ck, c, \pi, \mathsf{ct})$**:**

1. Parse ct as $(v, \mathsf{ct}_1, \mathsf{ct}_2)$.
2. Output $\mathrm{RandExt}(v, e(\mathsf{ct}_1, \pi)) \oplus \mathsf{ct}_2$.

Figure 2: Supplemental Encryption and Decryption.

Statistical Semantic Security. We first prove the following claim.

Claim 4.3 *Let $(ck, \cdot) \leftarrow K_{\text{binding}}(1^\lambda)$. Let S denote the random variable uniformly distributed in $\mathbb{Z}_n$. Then*

$$H_\infty(e(g,g)^S | (ck, h^S)) \geq \log p$$

Proof Let $q_1 \equiv q^{-1} \mod p$ and $p_1 = p^{-1} \mod q$. By Chinese remainder theorem, any $s \in \mathbb{Z}_n$ can be expressed as $s_q p p_1 + s_p q q_1$ where $s_p \equiv s \mod p$ and $s_q \equiv s \mod q$. As $(ck, \cdot) \leftarrow K_{\text{binding}}(1^\lambda)$, therefore we have that $h = g^{px}$ for some $x \in \mathbb{Z}_q^*$. Thus, for any $s \in \mathbb{Z}_n$, $h^s = g^{x(s_q p^2 p_1) \mod n}$.

Let S be uniformly distributed in $\mathbb{Z}_n$. By Chinese remainder theorem, $S_p \equiv S \mod p$ and $S_q \equiv S \mod q$ are uniform and independent random variables in $\mathbb{Z}_p$ and $\mathbb{Z}_q$ respectively. Also, $h^S = g^{xS_q p^2 p_1 \mod n}$. Therefore, conditioned on fixing h^S (which fixes S_q) and ck, g^S is still uniformly distributed over a set of size p since S_p is randomly distributed in $\mathbb{Z}_p$. Thus, $H_\infty(e(g,g)^S | (ck, h^S)) \geq \log p$. ■

Consider a commitment $c = \mathrm{com}(m; r)$ such that $m \notin \{0,1\}$. Let S be a random variable uniformly distributed in $\mathbb{Z}_n$. Then, we have that

$$e(c^S, cg^{-1}) = e(g,g)^{Sm(m-1)} e(h^S, g)^{r(m-1)} e(g, h^S)^{mr} e(h, h^S)^{r^2}$$

Since $m \notin \{0,1\}$, conditioned on fixing (h^S, ck), we infer from Claim 4.3 that $H_\infty(e(c^S, cg^{-1})) \geq \log p$. Now, relying on the fact that the output of randomness extractor is statistically close to uniform we conclude statistical semantic security for the scheme. ■

4.3 Construction from Decisional Linear Assumption

In this subsection we give our construction of homomorphic proof commitment with encryption from the decisional linear assumption [BB04]. We give the formal description of our construction in Figures 3,4.

Lemma 4.4 *Assuming the decisional linear assumption, the construction described in Figures 3 and 4 is a homomorphic proof commitment with encryption.*

Perfectly binding key generation $K_{\text{binding}}(1^k)$**:**

1. $(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}_{\text{DLIN}}(1^k)$
2. $x, y \leftarrow \mathbb{Z}_p^*$
3. $f = g^x, h = g^y$
4. $r_u, s_v \leftarrow \mathbb{Z}_p$
5. $(u, v, w) = (f^{r_u}, h^{s_v}, g^{r_u+s_v+z})$, where $z \leftarrow \mathbb{Z}_p^*$
6. Let $ck = (p, \mathbb{G}, \mathbb{G}_T, e, g, f, h, u, v, w)$
7. Let $xk = (ck, x, y, z)$ and return (ck, xk)

Perfectly hiding key generation $K_{\text{hiding}}(1^k)$**:**

1. $(p, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathcal{G}_{\text{DLIN}}(1^k)$
2. $x, y \leftarrow \mathbb{Z}_p^*$
3. $f = g^x, h = g^y$
4. $r_u, s_v \leftarrow \mathbb{Z}_p$
5. $(u, v, w) = (f^{r_u}, h^{s_v}, g^{r_u+s_v})$
6. Let $ck = (p, \mathbb{G}, \mathbb{G}_T, e, g, f, h, u, v, w)$
7. Let $tk = (ck, r_u, s_v)$ and return (ck, tk)

Commitment $\text{com}_{ck}(m)$**:** The key ck defines message space $\mathbb{Z}_p$, randomizer space $\mathbb{Z}_p \times \mathbb{Z}_p$ and commitment space $\mathbb{G}^3$. To commit to message $m \in \mathbb{Z}_p$ pick $(r, s) \leftarrow \mathbb{Z}_p \times \mathbb{Z}_p$ and return

$$c = (c_1, c_2, c_3) = \text{com}(m; r, s) = (u^m f^r, v^m h^s, w^m g^{r+s}).$$

Extraction $\text{Ext}_{xk}(c)$**:** On a perfect binding key we can extract m of length $\mathcal{O}(\log k)$ from $c = (c_1, c_2, c_3)$ as follows. Compute $(g^z)^m = c_3 c_1^{-1/x} c_2^{-1/y}$ and exhaustively search for m.

Trapdoor opening $\text{Topen}_{tk}(m, (r, s), m')$**:** Given a commitment $c = (u^m f^r, v^m h^s, w^m g^{r+s})$ under a perfectly hiding commitment key we have $c = (u^{m'} f^{r-(m'-m)r_u}, v^{m'} h^{s-(m'-m)s_v}, w^{m'} g^{r+s-(m'-m)(r_u+s_v)})$. So we can create a perfectly hiding commitment and open it to any value we wish if we have the trapdoor key $tk = (r_u, s_v)$ by returning (r', s') computed as $r' = r - (m' - m)r_u \bmod p$ and $s' = s - (m' - m)s_v \bmod p$.

WI proof $P_{01}(ck, m, (r, s))$**:** Given witness consisting of an opening $(m, r, s) \in \{0, 1\} \times \mathbb{Z}_p \times \mathbb{Z}_p$ we make a proof as follows. Choose $t \leftarrow \mathbb{Z}_p$ and let

$$\begin{array}{lll} \pi_{11} = (u^{2m-1} f^r)^r & \pi_{12} = v^{(2m-1)r} h^{rs-t} & \pi_{13} = w^{(2m-1)r} g^{(r+s)r+t} \\ \pi_{21} = u^{(2m-1)s} f^{rs+t} & \pi_{22} = (v^{2m-1} h^s)^s & \pi_{23} = w^{(2m-1)s} g^{(r+s)s-t} \end{array}$$

Return the proof $\pi = (\pi_{11}, \pi_{12}, \pi_{13}, \pi_{21}, \pi_{22}, \pi_{23})$.

Verification $V_{01}(ck, c, \pi)$**:** On input (ck, c, π) compute $\pi_{3j} = \pi_{1j}\pi_{2j}$ for $j = 1, 2, 3$. Accept the proof if and only if

$$\begin{array}{ll} e(f, \pi_{11}) = e(c_1, c_1 u^{-1}) & e(f, \pi_{12}) e(h, \pi_{21}) = e(c_1, c_2 v^{-1}) e(c_2, c_1 u^{-1}) \\ e(h, \pi_{22}) = e(c_2, c_2 v^{-1}) & e(f, \pi_{13}) e(g, \pi_{31}) = e(c_1, c_3 w^{-1}) e(c_3, c_1 u^{-1}) \\ e(g, \pi_{33}) = e(c_3, c_3 w^{-1}) & e(h, \pi_{23}) e(g, \pi_{32}) = e(c_2, c_3 w^{-1}) e(c_3, c_2 v^{-1}). \end{array}$$

Figure 3: Homomorphic proof commitment from decisional linear assumption taken verbatim from [GOS12].

Encrypt $E_{01}(ck, c, \mathsf{msg})$**:** To encrypt $\mathsf{msg} \in \{0,1\}^{\frac{\log p}{2}}$,

1. Choose $r_1, r_2, r_3, r_4, r_5, r_6 \leftarrow \mathbb{Z}_p^*$.
2. Compute

$$z_1 = \big(e(c_1, c_1 u^{-1})\big)^{r_1} \qquad z_4 = \big(e(c_1, c_2 v^{-1}) e(c_2, c_1 u^{-1})\big)^{r_4}$$
$$z_2 = \big(e(c_2, c_2 v^{-1})\big)^{r_2} \qquad z_5 = \big(e(c_1, c_3 w^{-1}) e(c_3, c_1 u^{-1})\big)^{r_5}$$
$$z_3 = \big(e(c_3, c_3 w^{-1})\big)^{r_3} \qquad z_6 = \big(e(c_2, c_3 w^{-1}) e(c_3, c_2 v^{-1})\big)^{r_6}.$$

3. Compute

$$\mathsf{ct}_{11} = f^{r_1} g^{r_5} \qquad \mathsf{ct}_{21} = h^{r_4} g^{r_5}$$
$$\mathsf{ct}_{12} = f^{r_4} g^{r_6} \qquad \mathsf{ct}_{22} = h^{r_2} g^{r_6}$$
$$\mathsf{ct}_{13} = f^{r_5} g^{r_3} \qquad \mathsf{ct}_{23} = h^{r_6} g^{r_3}$$

4. Choose $v \leftarrow \{0,1\}^*$ as the seed of RandExt.
5. Output $(v, \{\mathsf{ct}_{ij}\}, \mathrm{RandExt}(v, \prod z_i) \oplus \mathsf{msg})$.

Decrypt $D_{01}(ck, c, \pi, \mathsf{ct})$**:**

1. Parse ct as $(v, \{\mathsf{ct}_{ij}\}_{i\in[2],j\in[3]}, \overline{\mathsf{ct}})$.
2. Parse π as $\{\pi_{ij}\}_{i\in[2],j\in[3]}$.
3. Compute $\overline{z} = \prod_{i\in[2],j\in[3]} e(\mathsf{ct}_{ij}, \pi_{ij})$.
4. Output $\mathrm{RandExt}(v, \overline{z}) \oplus \overline{\mathsf{ct}}$.

Figure 4: Supplemental Encryption and Decryption.

Proof We note that $(K_{\text{binding}}, K_{\text{hiding}}, \mathrm{com}, P_{01}, V_{01}, \mathrm{Ext})$ is a homomorphic proof commitment scheme as argued by Groth et al. [GOS12]. We now prove that (E_{01}, D_{01}) satisfy perfect correctness and statistical semantic-security.

Perfect Correctness. Let $c = \mathrm{com}(ck, m; r)$ where $m \in \{0,1\}$. Let $\mathsf{ct} = (v, \{\mathsf{ct}_{ij}\}, \mathrm{RandExt}(v, \prod z_i) \oplus \mathsf{msg})$ and $\{\pi_{ij}\}$ be as described in Figure 3. To prove correctness it is sufficient to show that $\prod_{i\in[2],j\in[3]} e(\mathsf{ct}_{ij}, \pi_{ij}) = \prod z_i$. Note that:

$$e(\mathsf{ct}_{11}, \pi_{11}) = e(f^{r_1}, \pi_{11}) e(g^{r_5}, \pi_{11}) \qquad e(\mathsf{ct}_{21}, \pi_{21}) = e(h^{r_4}, \pi_{21}) e(g^{r_5}, \pi_{21})$$
$$e(\mathsf{ct}_{12}, \pi_{12}) = e(f^{r_4}, \pi_{12}) e(g^{r_6}, \pi_{12}) \qquad e(\mathsf{ct}_{22}, \pi_{22}) = e(h^{r_2}, \pi_{22}) e(g^{r_6}, \pi_{22})$$
$$e(\mathsf{ct}_{13}, \pi_{13}) = e(f^{r_5}, \pi_{13}) e(g^{r_3}, \pi_{13}) \qquad e(\mathsf{ct}_{23}, \pi_{23}) = e(h^{r_6}, \pi_{23}) e(g^{r_3}, \pi_{23})$$

Thus,

$$\prod_{i\in[2],j\in[3]} e(\mathsf{ct}_{ij},\pi_{ij}) = \Big(e(f^{r_1},\pi_{11})e(h^{r_2},\pi_{22})e(g^{r_3},\pi_{13}\pi_{23}) \\ e(f^{r_4},\pi_{12})e(h^{r_4},\pi_{21})e(f^{r_5},\pi_{13}) \\ e(g^{r_5},\pi_{11}\pi_{21})e(h^{r_6},\pi_{23})e(g^{r_6},\pi_{12}\pi_{22}) \Big)$$

$$= \Big(\left(e(c_1,c_1u^{-1})\right)^{r_1} \left(e(c_1,c_2v^{-1})e(c_2,c_1u^{-1})\right)^{r_4} \\ \left(e(c_2,c_2v^{-1})\right)^{r_2} \left(e(c_1,c_3w^{-1})e(c_3,c_1u^{-1})\right)^{r_5} \\ \left(e(c_3,c_3w^{-1})\right)^{r_3} \left(e(c_2,c_3w^{-1})e(c_3,c_2v^{-1})\right)^{r_6} \Big)$$

$$= \prod z_i$$

Statistical Semantic Security. We first prove the following claim.

Claim 4.5 *Let $(ck,\cdot) \leftarrow K_{\text{binding}}(1^\lambda)$. For every $i \in [6]$, let R_i denote the random variable uniformly distributed in $\mathbb{Z}_p$. Let*

$$\begin{array}{ll} \mathsf{CT}_{11} = f^{R_1}g^{R_5} & \mathsf{CT}_{21} = h^{R_4}g^{R_5} \\ \mathsf{CT}_{12} = f^{R_4}g^{R_6} & \mathsf{CT}_{22} = h^{R_2}g^{R_6} \\ \mathsf{CT}_{13} = f^{R_5}g^{R_3} & \mathsf{CT}_{23} = h^{R_6}g^{R_3} \end{array}$$

Then

$$H_\infty((R_1,R_2,R_3,R_4,R_5,R_6)|\{\mathsf{CT}_{i,j}\}) \geq \log p$$

Proof Let $S_{i,j} := \mathrm{DLOG}_g(\mathsf{CT}_{i,j})$. The proof follows directly from the observation that the following system of equations in $\{R_i\}$ is linearly dependent. [16]

$$\begin{array}{ll} S_{11} = xR_1 + R_5 & S_{21} = yR_4 + R_5 \\ S_{12} = xR_4 + R_6 & S_{22} = yR_2 + R_6 \\ S_{13} = xR_5 + R_3 & S_{23} = yR_6 + R_3 \end{array} \quad (4.1)$$

■

Consider a commitment $c = \mathrm{com}(m;r)$ such that $m \notin \{0,1\}$. Let R_i be a random variable

[16] Observe that the vectors $(0,0,1,0,x,0),(0,0,1,0,0,y),(0,0,0,x,0,1)$ and $(0,0,0,y,1,0)$ are linearly dependent.

uniformly distributed in $\mathbb{Z}_p$ for every $i \in [6]$. Then, we have that

$$\begin{aligned}\prod z_i &= \Big(\quad \left(e(c_1, c_1 u^{-1})\right)^{R_1} \left(e(c_1, c_2 v^{-1}) e(c_2, c_1 u^{-1})\right)^{R_4} \\ &\qquad \left(e(c_2, c_2 v^{-1})\right)^{R_2} \left(e(c_1, c_3 w^{-1}) e(c_3, c_1 u^{-1})\right)^{R_5} \\ &\qquad \left(e(c_3, c_3 w^{-1})\right)^{R_3} \left(e(c_2, c_3 w^{-1}) e(c_3, c_2 v^{-1})\right)^{R_6} \Big) \\ &= \underbrace{e(g,g)^{m(m-1)\left((xr_u)^2 R_1 + (ys_v)^2 R_2 + z'^2 R_3 + 2(xr_u ys_v) R_4 + 2(xr_u z') R_5 + 2(ys_v z') R_6\right)}}_{Z} \times \text{multiplicative terms}\end{aligned}$$

where $z' = z + r_u + s_v$. Since $m \notin \{0,1\}$, conditioned on fixing $\mathsf{ct}_1, \ldots, \mathsf{ct}_6$, we infer from Claim 4.3 that $H_\infty(Z) \geq \log p$ since the $\mathrm{DLOG}_{g_T}(Z)$ defines an equation in $\{R_i\}$ that is linearly independent of the system given in 4.1.[17] Now, relying on the fact that the output of randomness extractor is statistically close to uniform we conclude statistical semantic security for the scheme. ■

5 Garbling Protocols

In this section we give the definition of garbling scheme for protocols and give an instantiation based on a homomorphic proof commitment with encryption.

5.1 Definition

Let Φ be a n-party protocol.[18] Let x_i be the input of party i and let Φ_i be the next-message function for party i. We define the transcript of Φ to be the set of all messages exchanged between parties. The transcript is denoted by $\Phi(x_1, \ldots, x_n)$ when Φ is run with inputs $x_1, \ldots, x_n$. The transcript is also assumed to be the output of the protocol.

Definition 5.1 *A Garbling scheme for protocols is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Garble}, \mathsf{Eval})$ *with the following syntax, correctness and security properties.*

- $\mathsf{Setup}(1^\lambda)$: *It is a PPT algorithm that takes as input the security parameter (encoded in unary) and outputs a reference string σ.*
- $\mathsf{Garble}(\sigma, i, \Phi_i, x_i)$: *It is a PPT algorithm that takes as input a reference string σ, the index i of a party, the next message function Φ_i and the input x_i and outputs*
  - *A garbled protocol component $\widetilde{\Phi}_i$ of the next message function Φ_i.*
  - *An encoding $\widetilde{x}_i$ (of length ℓ_e) of the input x_i.*
  - *A set of encoding labels $\{\mathsf{lab}^i_{j,0}, \mathsf{lab}^i_{j,1}\}_{j \in [n \cdot \ell_e]}$ for the input encodings of all parties.*

[17] This can be verified by systematic elimination of every variable.

[18] For simplicity, we assume that Φ is deterministic. For the case where Φ is randomized, we extend the input string of each party to include its random coins so that Φ is a deterministic protocol in the inputs of the parties.

- $\mathsf{Eval}(\{\widetilde{\Phi}_i\}, \{\widetilde{x}_i\}, \{\mathsf{lab}^i_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}\})$: *It is a deterministic algorithm that takes as input the set of garbled protocol components* $\{\widetilde{\Phi}_i\}$, *a set of input encodings* $\{\widetilde{x}_i\}$ *and the encoding labels* $\{\mathsf{lab}^i_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}\}$ *corresponding to the input encodings* $\widetilde{x}_1\|\ldots\|\widetilde{x}_n$ *and outputs a string* y *or the symbol* $\perp$.

Correctness: *For every protocol* Φ *and every set of inputs* $\{x_i\}$,

$$\Pr\Big[\ \sigma \leftarrow \mathsf{Setup}(1^\lambda); (\widetilde{\Phi}_i, \widetilde{x}_i, \{\mathsf{lab}^i_{j,0}, \mathsf{lab}^i_{j,1}\}) \leftarrow \mathsf{Garble}(\sigma, i, \Phi_i, x_i)\ \forall\ i \in [n] : \\ \Phi(x_1, \ldots, x_n) = \mathsf{Eval}(\{\widetilde{\Phi}_i\}, \{\widetilde{x}_i\}, \{\mathsf{lab}^i_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}\})\Big] = 1$$

Semi-Honest Security: *There exists a PPT algorithm* Sim *such that for every protocol* Φ, *every subset* $H \subseteq [n]$ *of honest parties and every choice of inputs* $\{x_i\}_{i\in[n]}$ *of the parties we have that:*

$$\Big\{\sigma, \{\widetilde{\Phi}_i, \widetilde{x}_i, \mathsf{lab}^i_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}\}_{i\in[n]}\Big\} \overset{c}{\approx} \mathsf{Sim}(1^\lambda, \Phi, H, \{x_i\}_{i\notin H}, \Phi(x_1, \ldots, x_n))$$

where $\sigma \leftarrow \mathsf{Setup}(1^\lambda)$ *and for each* $i \in [n]$ *we have that* $(\widetilde{\Phi}_i, \widetilde{x}_i, \{\mathsf{lab}^i_{j,0}, \mathsf{lab}^i_{j,1}\}) \leftarrow \mathsf{Garble}(\sigma, i, \Phi_i, x_i)$.

Succinct Encodings. We say that a garbling scheme has succinct encodings if the size of the input encoding does not grow with the complexity of the protocol Φ. This is formally defined below.

Definition 5.2 *We say that a garbling scheme for protocols has succinct input encodings if there exists a fixed polynomial* p_{enc} *such that for every protocol* Φ *and every* $i \in [n]$ *the size of the input encoding* $|\widetilde{x}_i| = p_{enc}(|x_i|, \lambda)$.[19]

Semi-Malicious Security. For our specific application of designing two round multi-party computation protocols, the security guarantee in Definition 5.1 is not sufficient. We need a stronger form of security, namely semi-malicious security which is defined below. At a high-level this definition allows the adversary to choose its input encodings after seeing the common random string and the input encodings of the honest parties. We still assume that the input encodings are generated honestly but with arbitrary random coins. We call such adversaries as *admissible.*

Definition 5.3 (Semi-malicious Security) *There exists a PPT algorithm* $S = (S_1, S_2)$ *such that for every protocol* Φ, *and every subset* $H \subseteq [n]$ *of honest parties, and for every choice of inputs* $\{x_i\}_{i\in H}$ *for honest parties, we have that for every admissible PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,

$$\Big| \Pr\big[\mathsf{Real}[1^\lambda, \{x_i\}, H] = 1\big] - \Pr\big[\mathsf{Ideal}[1^\lambda, \{x_i\}, H] = 1\big] \Big| \leq \mathsf{negl}(\lambda)$$

where Real *and* Ideal *games are described in Figure 5.*

[19] For the case when Φ is a randomized protocol the length of the input x_i would now additionally grow with the randomness complexity of party P_i. In this case, the succinctness could be defined in stronger manner where the size of input encoding is independent of the randomness complexity as well.

$\mathsf{Real}[1^\lambda, \{x_i\}_{i\in H}, H]$	$\mathsf{Ideal}[1^\lambda, \{x_i\}_{i\in H}, H]$
1. $\sigma \leftarrow \mathsf{Setup}(1^\lambda)$ and for every $i \in H$, compute $(\widetilde{\Phi}_i, \widetilde{x}_i, \{\mathsf{lab}^i_{j,b}\}) \leftarrow \mathsf{Garble}(\sigma, i, \Phi_i, x_i)$	1. $(\sigma, \{\widetilde{x}_i\}_{i\in H}, \mathsf{st}_S) \leftarrow S_1(1^\lambda, H)$
2. $\{x_i, \tilde{x}_i\}_{i\notin H}, \mathsf{st}_{\mathcal{A}} \leftarrow \mathcal{A}_1(\sigma, \{\widetilde{x}_i\}_{i\in H})$.	2. $\{x_i, \tilde{x}_i\}_{i\notin H}, \mathsf{st}_{\mathcal{A}} \leftarrow \mathcal{A}_1(\sigma, \{\widetilde{x}_i\}_{i\in H})$.
3. Output $\mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \{\widetilde{\Phi}_i, \widetilde{x}_i, \mathsf{lab}^i_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}\}_{i\in H})$	3. Output $\mathcal{A}_2\left(\mathsf{st}_{\mathcal{A}}, S_2(\mathsf{st}_S, \{x_j, \widetilde{x}_j\}_{j\notin H}, \Phi(x_1, \ldots, x_n))\right)$

Figure 5: Semi-Malicious Real and Ideal world for Garbling Protocols

5.2 Construction

In this subsection we give a construction of a garbling scheme for protocols from a homomorphic proof commitment with encryption and a garbling scheme for circuits (which is implied by the existence of a commitment scheme). The main theorem that we prove in this section is:

Theorem 5.4 *Assuming the existence of a homomorphic proof commitment with encryption there exists a construction of garbling scheme for protocols satisfying semi-malicious security.*

From Lemma 4.2 and Lemma 4.4, this gives a construction of garbling scheme for protocols from the sub-group decision on composite order groups or the decision linear assumption on prime order groups.

Corollary 5.5 *Assuming the sub-group decision assumption or the decision linear assumption there exists a construction of garbling scheme for protocols satisfying semi-malicious security.*

Before describing the construction, we give some notation to describe the n-party protocol Φ and make additional assumptions on the structure of Φ. These assumptions can be made without loss of generality.

Notation for Φ. Recall that x_i denotes the input of party i and Φ_i denotes its next message function. We assume that the length of the input of each party is m. Let T be the round complexity of Φ.

Structure of Φ. We assume that each party P_i maintains a local state that is updated at the end of every round. The local state is a function of the input, the random tape and the set of messages received from other parties.

At the beginning of the t^{th} round, every party P_i runs the program Φ_i on input t to obtain an output (i^*, f, g). [20] Here, i^* denotes the *active party* in round t. The active party P_{i^*} computes *one* NAND gate on a pair of bits of its state and writes the computed bit to its state. The inputs to the NAND gate are given by the bits in the indices f and g of the local state of P_{i^*}. Additionally, for a (pre-determined) subset of rounds $B_{i^*} \subseteq \{t \in [T] : (i^*, \cdot, \cdot) = \Phi_i(t)\}$, P_{i^*} outputs the computed bit to other parties. In this case, all the parties copy this bit to their state. For the rest of the rounds

[20] We assume that $\Phi_1(t) = \Phi_2(t) = \ldots = \Phi_n(t)$ for every $t \in [T]$.

where P_{i^*} is active, it outputs the computed bit masked with a random bit. In those rounds, every other party ignores this message.

To describe this structure more formally, let $B := \cup_i B_i$. Let the initial state of the party P_i be $r_i \| (x_i, s_i)$ where $x_i \in \{0,1\}^m$ is the input, $s_i \in \{0,1\}^s$ be the random tape used in the computation of Φ and $r_i \in \{0,1\}^T$ are the masking bits. We will let r_i have the form

$$r_{i,k} := \begin{cases} 0 & \text{if } k \in [T] \cap B \\ \text{uniform in } \{0,1\} & \text{if } k \in [T] \setminus B \end{cases}$$

We consider $r_i \| (x_i, s_i)$ as the actual input of party P_i.

For every $i \in [n]$, let y_i be the state of party P_i before the beginning of round t. Let $(i^*, f, g) := \Phi_i(t)$. The parties compute their updated state y'_i at the end of round t as

$$\begin{aligned} y'_{i^*,k} \quad &:= \quad \begin{cases} y_{i^*,k} & k \neq t \\ \mathsf{NAND}(y_{i^*,f}, y_{i^*,g}) & k = t \end{cases} \\ \text{for } i \neq i^* \quad y'_i \quad &:= \quad \begin{cases} y_i & t \notin B_{i^*} \vee \mathsf{NAND}(y_{i^*,f}, y_{i^*,g}) = 0 \\ y_i \oplus e_t & t \in B_{i^*} \wedge \mathsf{NAND}(y_{i^*,f}, y_{i^*,g}) = 1 \end{cases} \end{aligned}$$

where e_k is the k-th unit vector. Finally, we let $\ell = T + m + r$ to denote the length of the local state of every party.

Remark 5.6 *We observe that any protocol Φ can be re-written to follow the above format at an additional cost of increasing the round complexity by a polynomial (in the computational complexity of Φ) factor.*

Construction. We give the formal description of our construction in Figure 6 and give an overview below. We make use of the following fact from [GOS12].

Fact 5.7 ([GOS12]) *Let $\mathcal{M}$ be the message space of a homomorphic proof commitment with encryption. Further, $\mathcal{M}$ is a finite cyclic group with neutral element 0 and generator 1. Let $b_0, b_1, b_2 \in \{0,1\}$. If the order of the group is at least 4, then*

$$b_2 = \neg(b_0 \wedge b_1) \qquad \textit{if and only if} \qquad b_0 + b_1 + 2b_2 - 2 \in \{0,1\}.$$

Overview. We start with the description of the input encoding. The encoding of an input x_i is given by a set of homomorphic commitments $\{c_{i,k}\}$ to each individual bit of the initial state $r_i \| (x_i, s_i)$ of party P_i in the computation of Φ. Recall that x_i is the input, s_i is the random tape used in the computation of Φ and r_i are the masking bits. Note that the homomorphic property of the commitment scheme implies that given $c_{i,k}$ which is a commitment to the bit $y_{i,k}$, one can efficiently compute the commitment $\bar{c}_{i,k}$ which is a commitment to the bit $1 - y_{i,k}$. The commitment $c_{i,k}$, of course hides the value of the bit $y_{i,k}$.

We now describe the garbled protocol component. The garbled protocol component $\widetilde{\Phi}_i$ consists of a sequence of T garbled circuits and a set of labels for evaluating the first garbled circuit in the sequence. These garbled circuits have a special structure, namely, the t^{th} garbled circuit in the sequence outputs the labels for evaluating the $(t+1)^{th}$ garbled circuit and thus starting from the first garbled circuit we can execute every garbled circuit in the sequence. We now give details of the t^{th} circuit in the sequence. At a high level, the t^{th} garbled circuit corresponds to the computation done by party P_i in the t^{th} round of the protocol Φ. In a bit more details, the t^{th} garbled circuit takes as input the local state obtained after the first $t-1$ rounds, updates the local state and outputs the labels corresponding to the updated state for evaluating the next garbled circuit. This ensures that at the end of the T^{th} evaluation, we can obtain the the transcript of the protocol from the final local state of party P_i. To explain in detail the working of the t^{th} garbled circuit, let us assume that P_i is the active party in the t^{th} round. This implies that P_i has to update its local state by computing the NAND of two bits in its current state and write the output to a specific location. Further, if $t \in B_i$ then P_i has to communicate this bit to the other parties and the other parties have to copy this bit to their state. This means that the labels output by the t^{th} garbled circuit in the protocol component $\widetilde{\Phi}_j$ for $j \neq i$ must reflect this communicated bit. The main technical challenge we solve is in designing a non-interactive method to realize this communication using homomorphic proof commitment with encryption. Let us explain how.

Recall that by our assumption on the structure of Φ, the updated state of every party can only be one of two choices. This choice is determined by the output of the NAND computation done by the active party. At a very high level, we achieve this communication by letting the active party give a proof π that it has correctly computed the NAND gate and every other party outputting two encryptions each containing a set of labels corresponding to one choice of the updated state. The guarantee we provide is that π can be used to decrypt exactly one of those two encryptions and the labels obtained as a result of the decryption procedure correspond to the correct updated state. Let us explain how this is achieved.

Let the NAND computation done in round t take as input the bits in position f and g of the local state of party P_i. For simplicity, let us assume that f, g correspond to indices where the input of P_i is written. The output of the NAND computation is written in position t. By our choice of the masking string r_i, $c_{i,t}$ is a commitment to 0 if $t \in B_i$. Further, by the homomorphic property of the commitment scheme, every party can compute $\overline{c}_{i,t}$ which is a commitment to 1. Fact 5.7 implies that if the output of the NAND computation is 0 then $e_0 = c_{i,f} \cdot c_{i,g} \cdot c_{i,t}^2 \mathrm{com}(ck, -2; 0)$ is a commitment to $\{0, 1\}$; else $e_1 = c_{i,f} \cdot c_{i,g} \cdot \overline{c}_{i,t}^2 \mathrm{com}(ck, -2; 0)$ is a commitment to $\{0, 1\}$. We let every other party output two zero-one encryptions: one under the commitment e_0 containing the set of labels of the updated state assuming the communicated bit is 0; and the other under the commitment e_1 assuming the communicated bit is 1. The active party outputs a zero-one proof that either e_0 or e_1 is a commitment to the message in $\{0, 1\}$. Using this proof every party can recover the correct set of labels corresponding to the updated state.

Correctness. To argue correctness, it is sufficient to show that the local state of each party is updated correctly at the end of every round number t. We show this by induction on the number of rounds. The base case is clear. Let us assume that the hypothesis is true for the first t rounds. Let y_i be the local state of party P_i at the end of round t. Let $(i^*, f, g) := \Phi_1(t+1)$. We consider two cases:

- **Case-1:** $t+1 \notin B_{i^*}$. In this case, the local state of parties $i \neq i^*$ does not change i.e., $y'_i = y_i$. The local state of party P_{i^*} is updated as $y'_{i^*,t+1} := \mathsf{NAND}(y_{i^*,f}, y_{i^*,g})$ and $y'_{i^*,k} = y_{i^*,k}$ for $k \neq t+1$. Notice that for the case where $t+1 \notin B_{i^*}$, program P_Φ outputs the labels corresponding to the string $\{y'_{i,k}\}_{k \neq t+1}$ in the clear and outputs two zero-one encryptions of the same label corresponding to $y'_{i,t+1}$ under the commitments e_0 and e_1 respectively. Thus, Let Φ be an n party protocol, $(K_{\text{binding}}, K_{\text{hiding}}, P_{01}, V_{01}, E_{01}, D_{01})$ be a homomorphic proof commitment with encryption, and $(\mathsf{GarbleCkt}, \mathsf{EvalCkt})$ be a garbling scheme for circuits.

$\mathsf{Setup}(1^\lambda)$**:** Sample $(ck, \cdot) \leftarrow K_{\text{binding}}(1^\lambda)$ and output $\sigma := ck$ as the reference string.

$\mathsf{Garble}(\sigma, i, \Phi_i, x_i)$**:** To generate the input encoding, garbled protocol component and encoding labels:

1. Compute $(\widetilde{x}_i, y_i, sk_i) \leftarrow \mathsf{Encode}(\sigma, i, x_i)$ where the function Encode is described in Figure 7.
2. Set $\mathsf{label}^{i,T+1} := \big((0,1), \ldots, (0,1)\big)$ where $(0,1)$ is repeated $\ell + n\ell_e + n\ell$ times and $\ell_e := |\widetilde{x}_i|$.
3. **for** each t from T down to 1,
$$\big(\widetilde{\mathsf{P}}^{i,t}, \mathsf{label}^{i,t}\big) \leftarrow \mathsf{GarbleCkt}(1^\lambda, \mathsf{P}_\Phi[i, t, sk_i, ck, \mathsf{label}^{i,t+1}])$$
where P_Φ is described in Figure 7.
4. Parse $\mathsf{label}^{i,1}$ as $\{\mathsf{st}^i_{k,0}, \mathsf{st}^i_{k,1}\}_{k\in[\ell]}, \{\mathsf{en}^i_{k,0}, \mathsf{en}^i_{k,1}\}_{k\in[n\ell_e]}, \{\mathsf{tr}^i_{k,0}, \mathsf{tr}^i_{k,1}\}_{k\in[n\ell]}$.
5. Set $\mathsf{st}^i := \{\mathsf{st}^i_{k,y_{i,k}}\}_{k\in[\ell]}$ and $\mathsf{tr}^i := \{\mathsf{tr}^i_{k,0}\}_{k\in[n\ell]}$.
6. Set the garbled protocol component $\widetilde{\Phi}_i := \big(\{\widetilde{\mathsf{P}}^{i,t}\}_{t\in[T]}, \mathsf{st}^i, \mathsf{tr}^i\big)$, the input encoding to $\widetilde{x}_i$ and the encoding labels to be $\{\mathsf{en}^i_{k,0}, \mathsf{en}^i_{k,1}\}_{k\in[n\ell_e]}$.

$\mathsf{Eval}(\{\widetilde{\Phi}_i\}, \{\overline{x}_i\}, \{\mathsf{en}^i_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}\})$**:** To compute the output of the protocol:

1. For every $i \in [n]$, parse $\widetilde{x}_i$ as $\{c_{i,k}\}_{k\in[\ell]}$. For every $k \in B$, check if $c_{i,k} := \text{com}(ck, 0; 0^\lambda)$. If not, output $\perp$.
2. Parse $\widetilde{\Phi}_i$ as $\big(\{\widetilde{\mathsf{P}}^{i,t}\}_{t\in[T]}, \mathsf{st}^i, \mathsf{tr}^i\big)$.
3. Set $\widetilde{\mathsf{label}}^i := \big(\mathsf{st}^i, \mathsf{en}^i_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}, \mathsf{tr}^i\big)$ and the initial tracking strings $u_i := 0^\ell$ for every $i \in [n]$.
4. **for** every round t from 1 to $T-1$ **do**:
   (a) Let $(i^*, f, g) := \Phi_1(t)$.
   (b) Compute $\big(\overline{\mathsf{label}}^{i^*}, \beta, \pi_{i^*,t}\big) \leftarrow \mathsf{EvalCkt}(\widetilde{\mathsf{P}}^{i^*,t}, \widetilde{\mathsf{label}}^{i^*})$ and $\overline{\mathsf{label}}^i \leftarrow \mathsf{EvalCkt}(\widetilde{\mathsf{P}}^{i,t}, \widetilde{\mathsf{label}}^i)$ for every $i \neq i^*$.
   (c) **for** every $i \in [n]$ **do**,
      i. Parse $\overline{\mathsf{label}}^i$ as $(\overline{\mathsf{st}}^i, \overline{\mathsf{en}}^i, \overline{\mathsf{tr}}^i)$.
      ii. Compute d_f, d_g, e_0, e_1 exactly as in P_Φ described in Figure 7 using the tracking string u_{i^*}.
      iii. Parse $\overline{\mathsf{st}}^i$ as $\big(\{\widehat{\mathsf{st}}^i_k\}_{k\neq t}, \mathsf{stct}^i_0, \mathsf{stct}^i_1\big)$ and compute $\widehat{\mathsf{st}}^i_t := D_{01}(ck, e_\beta, \mathsf{stct}^i_\beta, \pi_{i^*,t})$. Update $\mathsf{st}^i := \{\widehat{\mathsf{st}}^i_k\}_{k\in[\ell]}$.
      iv. Parse $\overline{\mathsf{tr}}^i$ as $\Big\{\{\widehat{\mathsf{tr}}^i_{(j-1)\ell+k}\}_{k\in[\ell]\setminus\{t\}}, \mathsf{trct}^i_{j,0}, \mathsf{trct}^i_{j,1}\Big\}_{j\in[n]}$. For every $j \in [n]$, compute $\widehat{\mathsf{tr}}^i_{(j-1)\ell+t} := D_{01}(ck, e_\beta, \mathsf{trct}^i_{j,\beta}, \pi_{i^*,t})$. Update $\mathsf{tr}^i := \{\widehat{\mathsf{tr}}^i_k\}_{k\in[n\ell]}$.
      v. Update $\widetilde{\mathsf{label}}^i := \big(\mathsf{st}^i, \mathsf{en}^i_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}, \mathsf{tr}^i\big)$.
      vi. Update $u_{i^*,t}$ to β. If $t \in B_{i^*}$, update every $u_{j,t}$ to β for all $j \in [n]$.
5. Compute $y := \mathsf{EvalCkt}(\widetilde{\mathsf{P}}^{i,T}, \widetilde{\mathsf{label}}^i)$ and output y.

Figure 6: Garbling Scheme for Protocols

$\mathsf{Encode}(\sigma, i, x_i)$

To generate an encoding of the input x_i do the following:

1. Choose $s_i \leftarrow \{0,1\}^s$ as the random tape of party P_i in the protocol Φ.
2. Let $B := \cup_i B_i$. Choose randomness $\{\omega_{i,k}\}_{k\in[\ell]}$ and the initial state $y_i := r_i \| (x_i, s_i)$ as:

$$r_{i,k} := \begin{cases} 0 & \text{if } k \in [T] \cap B \\ \text{uniform in } \{0,1\} & \text{if } k \in [T] \setminus B \end{cases} \qquad \omega_{i,k} := \begin{cases} 0^\lambda & \text{if } k \in B \\ \text{uniform in } \{0,1\}^\lambda & \text{otherwise} \end{cases}$$

3. For each $k \in [\ell]$, compute $c_{i,k} := \mathrm{com}(ck, y_{i,k}; \omega_{i,k})$.
4. Output $\widetilde{x}_i := \{c_{i,k}\}_{k\in[\ell]}$, the initial state y_i and the secret randomness $sk_i := \{\omega_{i,k}\}_{k\in[\ell]}$.

$\mathsf{P}_\Phi[i, t, sk_i, \{ck_i\}, \mathsf{label}]$

Input. The state y_i of party P_i, the set of encodings $\{\widetilde{x}_j\}$ and the set of tracking strings $\{u_j\}$
Hardcoded. The index i of the party, the round number t, the secret randomness sk_i, the commitment key ck and a set of labels $\mathsf{label} := \big\{\{\mathsf{st}_{k,0}, \mathsf{st}_{k,1}\}_{k\in[\ell]}, \{\mathsf{en}_{k,0}, \mathsf{en}_{k,1}\}_{k\in[n\ell_{enc}]}, \{\mathsf{tr}_{k,0}, \mathsf{tr}_{k,1}\}_{k\in[n\ell]}\big\}$.

1. Let $(i^*, f, g) := \Phi_i(t)$.
2. Parse $\widetilde{x}_{i^*}$ as $\{c_{i^*,k}\}_{k\in[\ell]}$.
3. Let d_f and d_g be the commitments to the bits $y_{i^*,f}$ and $y_{i^*,g}$ where y_{i^*} is the current state of the active party. These commitments are computed as follows: for $h \in \{f, g\}$, $d_h := c_{i^*,h}$ if $u_{i^*,h} = 0$; else, $d_h := \frac{\mathrm{com}(ck,1;0^\lambda)}{c_{i^*,h}}$.
4. Compute $e_0 := d_f d_g c_{i^*,t}^2 \mathrm{com}(ck, -2; 0^\lambda)$ and $e_1 := d_f d_g \left(\frac{\mathrm{com}(ck,1;0^\lambda)}{c_{i^*,t}}\right)^2 \mathrm{com}(ck, -2; 0^\lambda)$.
   Set $\alpha := \mathsf{NAND}(y_{i,f}, y_{i,g})$.
5. For $b \in \{0,1\}$, compute $\mathsf{stct}_b := \begin{cases} E_{01}(ck, e_b, \mathsf{st}_{t,b}) & \text{if } t \in B_{i^*} \\ E_{01}(ck, e_b, \mathsf{st}_{t,y_{i,t}}) & \text{if } t \notin B_{i^*} \wedge i \neq i^* \\ E_{01}(ck, e_b, \mathsf{st}_{t,\alpha}) & \text{if } t \notin B_{i^*} \wedge i = i^* \end{cases}$.
   Set $\overline{\mathsf{st}} := \{\mathsf{st}_{k,y_{i,k}}\}_{k\neq t}, \mathsf{stct}_0, \mathsf{stct}_1$.
6. Set $\overline{\mathsf{en}} := \{\mathsf{en}_{k,z_k}\}_{k\in[n\ell_{enc}]}$ where $z := \widetilde{x}_1 \| \ldots \| \widetilde{x}_n$.
7. For $b \in \{0,1\}$ and $j \in [n]$, compute $\mathsf{trct}_{j,b} := \begin{cases} E_{01}(ck, e_b, \mathsf{tr}_{(j-1)\ell+t,b}) & \text{if } (t \in B_{i^*}) \vee (j = i^*) \\ E_{01}(ck, e_b, \mathsf{tr}_{(j-1)\ell+t,u_{j,t}}) & \text{if } (t \notin B_{i^*}) \wedge (j \neq i^*) \end{cases}$.
   Set $\overline{\mathsf{tr}} := \big\{\{\mathsf{tr}_{(j-1)\ell+k,u_{j,k}}\}_{k\in[\ell]\setminus\{t\}}, \mathsf{trct}_{j,0}, \mathsf{trct}_{j,1}\big\}_{j\in[n]}$.
8. If $i = i^*$ then parse sk_i as $\{\omega_{i,k}\}_{k\in[\ell]}$. For $h \in \{f, g\}$, set $\omega'_{i,h} := \begin{cases} \omega_{i,h} & \text{if } u_{i,h} = 0 \\ -\omega_{i,h} & \text{otherwise} \end{cases}$
   Compute $\pi_{i,t} := P_{01}(ck, e_\beta, \rho_\beta)$ where $\beta := y_{i,t} \oplus \alpha, \rho_0 = \omega'_{i,f} + \omega'_{i,g} + 2\omega_{i,t}, \rho_1 = \omega'_{i,f} + \omega'_{i,g} - 2\omega_{i,t}$.
9. If $t \neq T$ then output $\overline{\mathsf{label}} := (\overline{\mathsf{st}}, \overline{\mathsf{en}}, \overline{\mathsf{tr}})$ and additionally output $(\beta, \pi_{i,t})$ if $i = i^*$.
   If $t = T$ then output the transcript of the protocol from the state as $\{y_{i,k}\}_{k\in B}$.

Figure 7: The programs Encode and P_Φ.

decrypting stct^i_β using the proof $\pi_{i^*,t+1}$ yields the label corresponding to $y'_{i,t+1}$ for every $i \in [n]$. Thus, the updated states of every party is correct as per the computation of Φ.

- **Case-2:** $t+1 \in B_{i^*}$. In this case, $y'_{i,t+1} = \mathsf{NAND}(y_{i^*,f}, y_{i^*,g})$ and $y'_{i^*,k} = y_{i^*,k}$ for $k \neq t+1$ for every party $i \in [n]$. The program P_Φ outputs the labels corresponding to the string $\{y'_{i,k}\}_{k \neq t+1}$ in the clear and outputs a zero-one encryption of the label $y'_{i,t+1}$ under the commitment $e_{y'_{i,t+1}}$ for every $i \in [n]$. Notice that by our construction $y_{i,t+1} = 0$ and thus $\beta := y'_{i,t+1}$. Thus, decrypting stct^i_β using the proof $\pi_{i^*,t+1}$ yields the label corresponding to $y'_{i,t+1}$ for every $i \in [n]$. Thus, even in this case the updated state of every party is correct as per the computation of Φ.

5.3 Security

In this subsection we prove that the construction given in Figure 6 satisfies the semi-malicious security (Definition 5.3). We start with the description of simulators (S_1, S_2).

S_1**:** On input 1^λ and the set H, S_1 generates the encodings of the honest parties as follows:

1. Sample $(ck, tk) \leftarrow K_{\text{hiding}}(1^\lambda)$ and set $\sigma := ck$.
2. **for** every $i \in H$ **do**:
   (a) Set the initial state of the party P_i to be $y_i := 0^T \| (0^m, 0^s)$. Choose the randomness $\{\omega_{i,k}\}$ as in the honest execution of Encode function.
   (b) Generate the commitments $c_{i,k} := \text{com}(ck, y_{i,k}; \omega_{i,k})$ for each $k \in [\ell]$.
   (c) Set the encoding $\widetilde{x}_i := \{c_{i,k}\}_{k \in [\ell]}$.
3. Set the secret state $\mathsf{st}_S := (tk, \{\omega_{i,k}\}_{i \in H, k \in [\ell]})$
4. Output $(\sigma, \{\widetilde{x}_i\}_{i \in H}, \mathsf{st}_S)$.

S_2**:** On input the secret state st_S, $\{\widetilde{x}_j, y_j\}_{j \notin H}$ [21] and the transcript $\Phi(x_1, \ldots, x_n)$, S_2 generates the garbled protocol components of the honest parties as follows:

1. For every $j \notin H$, construct the final state y^*_j using the initial state y_j and the transcript $\Phi(x_1, \ldots, x_n)$. Set the final tracking string corresponding to party P_j as $u^*_j := y_j \oplus y^*_j$.
2. For every $j \in H$, set the final tracking string $u^*_j := \begin{cases} 0 & \text{if } k > T \\ \text{uniform in } \{0,1\} & \text{if } k \in [T] \setminus B \\ \text{based on } \Phi(x_1, \ldots, x_n) & \text{if } k \in [T] \cap B \end{cases}$.
3. For every $i \in H$, compute $\left(\widetilde{P}^{i,T}, \widetilde{\mathsf{label}}^{i,T}\right) \leftarrow \mathsf{Sim}(1^\lambda, \Phi(x_1, \ldots, x_n))$.
4. **for** every t from $T-1$ down to 1 **do**:
   (a) For every $i \in H$, parse $\widetilde{\mathsf{label}}^{i,t+1}$ as $\{\mathsf{st}^i_k\}_{k \in [\ell]}, \{\mathsf{en}^i_k\}_{k \in [n\ell_e]}$ and $\{\mathsf{tr}^i_k\}_{k \in [\ell]}$.
   (b) Let $(i^*, f, g) := \Phi_1(t)$.
   (c) Compute d_f, d_g, e_0, e_1 as given in program P_Φ using the final tracking string u_{i^*}.
   (d) Let $\beta := u_{i^*,t}$.

[21] Recall that we consider y_i to be the actual input of party P_i

(e) For every $i \in H$, generate $\mathsf{stct}^i_\beta := E_{01}(ck, e_\beta, \mathsf{st}^i_t)$ and $\mathsf{stct}^i_{1-\beta} := E_{01}(ck, e_{1-\beta}, 0^\lambda)$.
For every $j \in [n]$, generate $\mathsf{trct}^i_{j,\beta} := E_{01}(ck, e_\beta, \mathsf{tr}^i_{(j-1)\ell+t})$ and $\mathsf{trct}^i_{j,1-\beta} := E_{01}(ck, e_{1-\beta}, 0^\lambda)$.

(f) Set $\overline{\mathsf{st}}^i$ as $\big(\{\mathsf{st}^i_k\}_{k\neq t}, \mathsf{stct}^i_0, \mathsf{stct}^i_1\big)$, $\overline{\mathsf{en}}^i$ as $\{\mathsf{en}^i_k\}_{k\in[n\ell_e]}$ and $\overline{\mathsf{tr}}^i$ as $\big\{\{\mathsf{tr}^i_{(j-1)\ell+k}\}_{k\in[\ell]\setminus\{t\}}, \mathsf{trct}^i_{j,0}, \mathsf{trct}^i_{j,1}\big\}_{j\in[n]}$.

(g) For every $i \in H \setminus \{i^*\}$ generate, $\widetilde{P}^{i,t}, \widetilde{\mathsf{label}}^{i,t} \leftarrow \mathsf{Sim}(1^\lambda, (\overline{\mathsf{st}}^i, \overline{\mathsf{en}}^i, \overline{\mathsf{tr}}^i))$.

(h) **if** $i^* \in H$ then:

  i. For $h \in \{f, g, t\}$, set $\omega'_{i^*,h} := \begin{cases} \omega_{i^*,h} & \text{if } u_{i^*,h} = 0 \\ -\omega_{i^*,h} & \text{otherwise} \end{cases}$.

  ii. Compute $v_1 := \mathrm{Topen}(tk, u_{i^*,f}, \omega'_{i^*,f}, 0)$, $v_2 := \mathrm{Topen}(tk, u_{i^*,g}, \omega'_{i^*,g}, 0)$ and $v_3 := \mathrm{Topen}(tk, u_{i^*,t}, \omega'_{i^*,t}, 1)$.

  iii. Compute $\pi_{i^*,t} := P_{01}(ck, 1, \rho)$ where $\rho = v_1 + v_2 - 2v_3$.

  iv. Generate, $\widetilde{P}^{i^*,t}, \widetilde{\mathsf{label}}^{i^*,t} \leftarrow \mathsf{Sim}(1^\lambda, (\overline{\mathsf{st}}^{i^*}, \overline{\mathsf{en}}^{i^*}, \overline{\mathsf{tr}}^{i^*}), (\beta, \pi_{i^*,t}))$

This completes the description of the simulators (S_1, S_2). We show that

Lemma 5.8 *Assuming the security of the garbling scheme for circuits and the homomorphic proof commitment with encryption, we have that for every protocol Φ, and every subset $H \subseteq [n]$ of honest parties, for every choice of inputs $\{x_i\}_{i\in H}$ for honest parties and for every PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,*

$$\Big|\Pr\big[\mathsf{Real}[1^\lambda, \{x_i\}, H] = 1\big] - \Pr\big[\mathsf{Ideal}[1^\lambda, \{x_i\}, H] = 1\big]\Big| \leq \mathsf{negl}(\lambda)$$

The proof of this lemma appears in Section 5.3.1.

5.3.1 Proof of Lemma 5.8

We prove the lemma through an hybrid argument. In a hybrid Hybrid_k, we define $\mathsf{Adv}(\mathsf{Hybrid}_k)$ to be the probability that $\mathcal{A}$ outputs 1 when given inputs as distributed in Hybrid_k.

For every $w \in [T]$, we define Hybrid_w as follows:

$\underline{\mathsf{Hybrid}_w}$: In this hybrid, we change how $\big(\widetilde{P}^{i,t}, \widetilde{\mathsf{label}}^{i,t}\big)$ for every $i \in H$ and $t < w$ are generated. In particular, we generate them as described in the modified garbling procedure given in Figure 8.

Notice that Hybrid_1 is distributed as in the real world.

Lemma 5.9 *Assuming the security of garbling scheme for circuits and the homomorphic proof commitments with encryption, we have that for every $w \in [T]$, $|\mathsf{Adv}(\mathsf{Hybrid}_w) - \mathsf{Adv}(\mathsf{Hybrid}_{w+1})| \leq \mathsf{negl}(\lambda)$.*

Proof We define a couple of intermediate hybrids.
$\underline{\mathsf{Hybrid}_{w,1}}$: Let $y^w_i := \{y^*_{i,k}\}_{k\in[w-1]}\|\{y_{i,k}\}_{k\in[w,\ell]}$ be the local state of party P_i at the beginning of the w-th round. Let $\{u^w_j\}$ be the set of tracking strings at the beginning of the w-th round. In this hybrid, for every $i \in H$ we generate

$$\big(\widetilde{P}^{i,w}, \widetilde{\mathsf{label}}^{i,w}\big) \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{P}_\Phi[i, w, ck, sk_i, \mathsf{label}^{i,w+1}](y^w_i, \{\widetilde{x}_j\}, \{u^w_j\}))$$

Garble': On additional inputs the hybrid number w, the final state y_i^* of party P_i, the set of encodings $\{\widetilde{x}_j\}$ and the final set of tracking strings $\{u_j^*\}$ for every P_j do:

1. Compute $(\widetilde{x}_i, y_i, sk_i) \leftarrow \mathsf{Encode}(\sigma, i, x_i)$ where the function Encode is described in Figure 7.
2. Set $\mathsf{label}^{i,T+1} := \big((0,1), \ldots, (0,1)\big)$ where $(0,1)$ is repeated $\ell + n\ell_e + n\ell$ times and $\ell_e := |\widetilde{x}_i|$.
3. **for** each t from T down to $\underline{w}$,

$$\big(\widetilde{\mathsf{P}}^{i,t}, \mathsf{label}^{i,t}\big) \leftarrow \mathsf{GarbleCkt}(1^\lambda, \mathsf{P}_\Phi[i, t, sk_i, ck, \mathsf{label}^{i,t+1}])$$

   where P_Φ is described in Figure 7.
4. Parse $\mathsf{label}^{i,w}$ as $\{\mathsf{st}^i_{k,0}, \mathsf{st}^i_{k,1}\}_{k\in[\ell]}, \{\mathsf{en}^i_{k,0}, \mathsf{en}^i_{k,1}\}_{k\in[n\ell_e]}, \{\mathsf{tr}^i_{k,0}, \mathsf{tr}^i_{k,1}\}_{k\in[n\ell]}$.
5. Set $\widetilde{\mathsf{label}}^{i,w} := \{\mathsf{st}^i_{k,y^*_{i,k}}\}_{k\in[w]}, \{\mathsf{st}^i_{k,y_{i,k}}\}_{k\in[w+1,\ell]}, \mathsf{en}_{\widetilde{x}_1\ldots\widetilde{x}_n}, \{\mathsf{tr}^i_{(j-1)\ell+k,u^*_{j,k}}\}_{j\in[n],k\in[w]}, \{\mathsf{tr}^i_{(j-1)\ell+k,0}\}_{j\in[n],k\in[w+1,\ell]}$.
6. **for** each t from $w-1$ down to 1:
   (a) Parse $\widetilde{\mathsf{label}}^{i,t+1}$ as $\{\mathsf{st}^i_k\}_{k\in[\ell]}, \{\mathsf{en}^i_k\}_{k\in[n\ell_e]}$ and $\{\mathsf{tr}^i_k\}_{k\in[\ell]}$.
   (b) Let $(i^*, f, g) := \Phi_1(t)$.
   (c) Compute d_f, d_g, e_0, e_1 as given in program P_Φ using the tracking string $u^*_{i^*}$.
   (d) Let $\beta := u^*_{i^*,t}$.
   (e) Generate $\mathsf{stct}^i_\beta := E_{01}(ck, e_\beta, \mathsf{st}^i_t)$ and $\mathsf{stct}^i_{1-\beta} := E_{01}(ck, e_{1-\beta}, \underline{0^\lambda})$. Generate $\mathsf{trct}^i_{j,\beta} := E_{01}(ck, e_\beta, \mathsf{tr}^i_{(j-1)\ell+t})$ and $\mathsf{trct}^i_{j,1-\beta} := E_{01}(ck, e_{1-\beta}, \underline{0^\lambda})$ for every $j \in [n]$.
   (f) Set $\overline{\mathsf{st}}^i$ as $\big(\{\mathsf{st}^i_k\}_{k\neq t}, \mathsf{stct}^i_0, \mathsf{stct}^i_1\big)$, $\overline{\mathsf{en}}^i$ as $\{\mathsf{en}^i_k\}_{k\in[n\ell_e]}$ and $\overline{\mathsf{tr}}^i$ as $\big\{\{\mathsf{tr}^i_{(j-1)\ell+k}\}_{k\in[\ell]\setminus\{t\}}, \mathsf{trct}^i_{j,0}, \mathsf{trct}^i_{j,1}\big\}_{j\in[n]}$.
   (g) Generate, $\widetilde{P}^{i,t}, \widetilde{\mathsf{label}}^{i,t} \leftarrow \mathsf{Sim}(1^\lambda, (\overline{\mathsf{st}}^i, \overline{\mathsf{en}}^i, \overline{\mathsf{tr}}^i))$.

Figure 8: Modified Garbling Procedure

Claim 5.10 *Assuming the security of garbling scheme for circuits,* $|\mathsf{Adv}(\mathsf{Hybrid}_w) - \mathsf{Adv}(\mathsf{Hybrid}_{w,1})| \leq \mathsf{negl}(\lambda)$

Proof Assume for the sake of contradiction that $|\mathsf{Adv}(\mathsf{Hybrid}_w) - \mathsf{Adv}(\mathsf{Hybrid}_{w,1})| > \frac{1}{\mathsf{poly}(\lambda)}$. We construct an adversary $\mathcal{B}$ breaking the security of garbling scheme for circuits.

$\mathcal{B}$ samples $(ck, \cdot) \leftarrow K_{\text{binding}}(1^\lambda)$ and computes the input encodings $\{\widetilde{x}_i\}_{i\in H}$ as per the honest procedure given in Figure 7. It then runs $\mathcal{A}_1(\sigma, \{\widetilde{x}_i\}_{i\in H})$ to obtain $\{\widetilde{x}_i\}_{i\notin H}$ and $\mathsf{st}_\mathcal{A}$. $\mathcal{B}$ generates the garbled circuits $\widetilde{\mathsf{P}}^{i,t}$ for t from T to $w+1$ as in the modified garbled procedure in Figure 8. $\mathcal{B}$ then interacts with the garbled circuits challenger and gives for every $i \in H$, $\mathsf{P}_\Phi[i, t, sk_i, ck, \mathsf{label}^{i,w+1}]$ as the challenge circuit and $y_i^w, \{\widetilde{x}_j\}, \{u_j^w\}$ as the challenge inputs. It obtains $\widetilde{\mathsf{P}}^{i,w}, \widetilde{\mathsf{label}}^{i,w}$. It then uses $\widetilde{\mathsf{label}}^{i,w}$ to generate the rest of the garbled circuits $\widetilde{\mathsf{P}}^{i,t}$ as in Figure 8. Finally, $\mathcal{B}$ runs $\mathcal{A}_2$ on the inputs $\mathsf{st}_\mathcal{A}, \{\widetilde{\Phi}_i, \widetilde{x}_i, \mathsf{lab}^i_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}\}_{i\in H}$ and outputs whatever $\mathcal{A}$ outputs.

Notice that if the garbling $\widetilde{\mathsf{P}}^{i,w}, \widetilde{\mathsf{label}}^{i,w}$ is generated using the honest procedure then the inputs $\mathcal{A}_2$ are distributed identically to Hybrid_w. Else, they are distributed identically to $\mathsf{Hybrid}_{w,1}$. Thus,

$\mathcal{B}$ breaks the security of garbling scheme for circuits which is a contradiction. ■

$\underline{\mathsf{Hybrid}_{w,2}}$: In this hybrid, we perform the modified garbling procedure with input $w+1$ instead of w. This hybrid is identically distributed to Hybrid_{w+1}.

Claim 5.11 *Assuming the statistical semantic security of homomorphic proof commitments with encryption,* $|\mathsf{Adv}(\mathsf{Hybrid}_{w,1}) - \mathsf{Adv}(\mathsf{Hybrid}_{w,2})| \leq \mathsf{negl}(\lambda)$

Proof Assume for the sake of contradiction that $|\mathsf{Adv}(\mathsf{Hybrid}_{w,1}) - \mathsf{Adv}(\mathsf{Hybrid}_{w,2})| > \frac{1}{\mathsf{poly}(\lambda)}$. We construct an adversary $\mathcal{B}$ breaking the semantic security of homomorphic proof commitments with encryption.

$\mathcal{B}$ obtains ck from the external challenger and computes the input encodings $\{\widetilde{x}_i\}_{i\in H}$ as per the honest procedure given in Figure 7. It then runs $\mathcal{A}_1(\sigma, \{\widetilde{x}_i\}_{i\in H})$ to obtain $\{\widetilde{x}_i\}_{i\notin H}$ and $\mathsf{st}_{\mathcal{A}}$. $\mathcal{B}$ generates the garbled circuits $\widetilde{\mathsf{P}}^{i,t}$ for t from T to $w+1$ as in the modified garbled procedure in Figure 8. Instead of generating the garbled circuit $\left(\widetilde{P}^{i,w}, \widetilde{\mathsf{label}}^{i,w}\right) \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{P}_\Phi[i, w, ck, sk_i, \mathsf{label}^{i,w+1}](y_i^w, \{\widetilde{x}_j\}, \{u_j^w\}))$ for every $i \in H$ as in $\mathsf{Hybrid}_{w,1}$, it generates it as follows:

1. Let y_i^{w+1} be the local state of party i and $\{u_j^{w+1}\}$ be the set of tracking strings at the end of the w-th round.
2. Let $(i^*, f, g) := \Phi_1(w)$.
3. Compute d_f, d_g, e_0, e_1 as given in program P_Φ using the tracking string $u_{i^*}^{w+1}$.
4. Let $\beta := u_{i^*,w}^{w+1}$, $\alpha := y_{i,w}^{w+1}$ and $\gamma_j := u_{j,w}^{w+1}$ for every $j \in [n]$.
5. Generate $\mathsf{stct}_\beta^i := E_{01}(ck, e_\beta, \mathsf{st}_{w,\alpha}^i)$ if $w \notin B_{i^*} \vee i = i^*$; else generate $\mathsf{stct}_\beta^i := E_{01}(ck, e_\beta, \mathsf{st}_{w,\beta}^i)$. Interact with the semantic security challenger by giving $e_{1-\beta}$ as the challenge commitment, $\mathsf{st}_{w,1-\alpha}^i$ [22] and 0^λ as the challenge messages. Receive the challenge ciphertext $\mathsf{stct}_{1-\beta}^i$.
6. Generate $\mathsf{trct}_{j,\beta}^i := E_{01}(ck, e_\beta, \mathsf{tr}_{(j-1)\ell+w,\beta}^i)$ if $w \in B_{i^*} \vee j = i^*$; else generate $\mathsf{trct}_{j,\beta}^i := E_{01}(ck, e_\beta, \mathsf{tr}_{(j-1)\ell+w,\gamma_j}^i)$. Interact with the semantic security challenger by giving $e_{1-\beta}$ as the challenge commitment and $\mathsf{tr}_{(j-1)\ell+w,1-\beta}^i$ [23] and 0^λ as the challenge messages for every $j \in [n]$. Receive the set of challenge ciphertexts $\{\mathsf{trct}_{j,1-\beta}^i\}$ for every $j \in [n]$.
7. Set $\overline{\mathsf{st}}^i$ as $\left(\{\mathsf{st}_{k,y_{i,k}^{w+1}}^i\}_{k\neq t}, \mathsf{stct}_0^i, \mathsf{stct}_1^i\right)$, $\overline{\mathsf{en}}^i$ as $\{\mathsf{en}_k^i\}_{k\in[n\ell_e]}$ and $\overline{\mathsf{tr}}^i$ as $\left\{\{\mathsf{tr}_{(j-1)\ell+k,u_{j,k}^{w+1}}^i\}_{k\in[\ell]\setminus\{t\}}, \mathsf{trct}_{j,0}^i, \mathsf{trct}_{j,1}^i\right\}_{j\in[n]}$.
8. Generate, $\widetilde{P}^{i,w}, \widetilde{\mathsf{label}}^{i,t} \leftarrow \mathsf{Sim}(1^\lambda, (\overline{\mathsf{st}}^i, \overline{\mathsf{en}}^i, \overline{\mathsf{tr}}^i))$.

Finally, $\mathcal{B}$ runs $\mathcal{A}_2$ on the inputs $\mathsf{st}_{\mathcal{A}}, \{\widetilde{\Phi}_i, \widetilde{x}_i, \mathsf{lab}_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}^i\}_{i\in H}$ and outputs whatever $\mathcal{A}$ outputs. Notice that the challenge commitment $e_{1-\beta}$ is not a commitment to zero-one message. Thus, $\mathcal{B}$ represents a valid challenger to the semantic security. If the challenge ciphertexts contain an encryption of the string 0^λ then the view of $\mathcal{A}_2$ is distributed identically to $\mathsf{Hybrid}_{w,2}$. Else, it is

[22] We need to give $\mathsf{st}_{w,\alpha}^i$ instead of $\mathsf{st}_{w,1-\alpha}^i$ if $w \notin B_{i^*}$

[23] If $w \notin B_{i^*} \wedge j = i^*$, we should then give $\mathsf{tr}_{(j-1)\ell+w,\gamma_j}^i$ instead of $\mathsf{tr}_{(j-1)\ell+w,1-\beta}^i$.

distributed identically to $\mathsf{Hybrid}_{w,1}$. Thus, $\mathcal{B}$ breaks the semantic security of homomorphic proof commitment with encryption. ■

This completes the proof of the lemma. ■

$\underline{\mathsf{Hybrid}_{T+1}}$: In this hybrid, we change how the reference string is generated. Instead of generating $(ck, \cdot) \leftarrow K_{\text{binding}}(1^\lambda)$, we generate it as $(ck, \cdot) \leftarrow K_{\text{hiding}}(1^\lambda)$.

Lemma 5.12 *Assuming key indisintinguishability property of homomorphic proof commitment with encryption,* $|\mathsf{Adv}(\mathsf{Hybrid}_T) - \mathsf{Adv}(\mathsf{Hybrid}_{T+1})| \leq \mathsf{negl}(\lambda)$

Proof If $|\mathsf{Adv}(\mathsf{Hybrid}_T) - \mathsf{Adv}(\mathsf{Hybrid}_{T+1})| > \frac{1}{\mathsf{poly}(\lambda)}$ we can get a straightforward reduction to the breaking key indisintguishability property. ■

$\underline{\mathsf{Hybrid}_{T+2}}$: For every $i \in H$, let y_i^* be the final local state, y_i be the initial local state and u_i^* be the final value of the tracking string corresponding to i. Notice that u_i^* is distributed uniformly on projection to the coordinates $[T] \setminus \{B\}$ and $y_i^* := y_i \oplus u_i^*$. Let $B_H := \cup_{i \in H} B_i$. In this hybrid, we change how $\pi_{i^*,t}$ (which is hardcoded while generating the simulated garbled circuit) is generated for every $t \in B_H$. In particular, we generate it as:

1. Let $(i^*, f, g) := \Phi_i(t)$.
2. For $h \in \{f, g, t\}$, set $\omega'_{i^*,h} := \begin{cases} \omega_{i^*,h} & \text{if } u^*_{i^*,h} = 0 \\ -\omega_{i^*,h} & \text{otherwise} \end{cases}$.
3. Compute v_1 such that $\text{com}(ck, y^*_{i^*,f}; \omega'_{i^*,f}) := \text{com}(0; v_1)$, v_2 such that $\text{com}(ck, y^*_{i^*,g}; \omega'_{i^*,g}) := \text{com}(0; v_2)$ and v_3 such that $\text{com}(ck, y^*_{i^*,t}; \omega'_{i^*,t}) := \text{com}(1; v_3)$. Notice that this step might take super-polynomial time.
4. Compute $\pi_{i^*,t} := P_{01}(ck, 1, \rho)$ where $\rho = v_1 + v_2 - 2v_3$.

Lemma 5.13 *Assuming perfect witness indistinguishability of homomorphic proof commitment with encryption we have,* $|\mathsf{Adv}(\mathsf{Hybrid}_{T+1}) - \mathsf{Adv}(\mathsf{Hybrid}_{T+2})| = 0$.

Proof Assume for the sake of contradiction that $|\mathsf{Adv}(\mathsf{Hybrid}_{w,1}) - \mathsf{Adv}(\mathsf{Hybrid}_{w,2})| > 0$. We construct an adversary $\mathcal{B}$ breaking the perfect witness indistinguishability of homomorphic proof commitment with encryption.

$\mathcal{B}$ interacts with the external challenger and obtains the commitment key ck. It sets $\sigma := ck$ and generates the encodings as in the honest procedure. For every $t \in B_H$, $\mathcal{B}$ submits the following sets of messages $(y^*_{i^*,f}, 0), (y^*_{i^*,g}, 0)$ and $(y^*_{i^*,t}, 1)$ and the following sets of witnesses $(\omega'_{i^*,f}, v_1)$, $(\omega'_{i^*,g}, v_2)$ and $(\omega'_{i^*,g}, v_3)$. It receives $\pi_{i^*,t}$ as the challenge proof. $\mathcal{B}$ generates the rest of the inputs to $\mathcal{A}_2$ and outputs whatever $\mathcal{A}_2$ outputs.

Notice that if $\pi_{i^*,t}$ has been generated using the honest commitment procedure then the view of $\mathcal{A}$ is identical to Hybrid_{T+1}. Else, it is distributed identically to Hybrid_{T+2}. Thus, $\mathcal{B}$ breaks the perfect witness indisintinguishability of homomorphic proof commitment with encryption. ■

$\underline{\mathsf{Hybrid}_{T+3}}$: In this hybrid, for every $i \in H$, we change how the encoding $\widetilde{x}_i$ is generated. In particular, for every $k \in [\ell] \setminus B$, we generate $c_{i,k} := \text{com}(ck, 0)$. As in the previous step, we find v_1, v_2, v_3 by running in possibly super-polynomial time.

Lemma 5.14 *Assuming perfect hiding of the homomorphic proof commitment with encryption we have,* $|\mathsf{Adv}(\mathsf{Hybrid}_{T+2}) - \mathsf{Adv}(\mathsf{Hybrid}_{T+3})| = 0$.

Proof It is easy to see that the only change between Hybrid_{T+2} and Hybrid_{T+3} is in the generation of commitments to the initial state. By perfect hiding property of the commitment scheme, we have $|\mathsf{Adv}(\mathsf{Hybrid}_{T+2}) - \mathsf{Adv}(\mathsf{Hybrid}_{T+3})| = 0$. ■

$\underline{\mathsf{Hybrid}_{T+4}}$: In this hybrid, we make the process of sampling from the distribution efficient by giving access to the trapdoor key tk. Notice that the distributions Hybrid_{T+3} and Hybrid_{T+4} are identical from the perfect trapdoor opening property of the homomorphic proof commitment with encryption. Hybrid_{T+4} is distributed identically to Ideal.

This completes the proof of Lemma 5.8.

6 Two Round Multi-Party Computation

In this section we give a construction of a UC secure two-round multi-party computation from a standalone multi-party semi-honest protocol Φ having arbitrary (but polynomial) round complexity using a garbling scheme for protocols. We give a construction in the $\mathcal{F}_{\text{NIZK}}$ hybrid model and we start by recall the NIZK ideal functionality.

NIZK Hybrid Model. We recall the functionality $\mathcal{F}_{\text{NIZK}}$ from [GOS12] in Figure 9. This figure is taken verbatim from [GOS12].

Parameterized with relation R and running with parties $P_1, \ldots, P_n$ and adversary $\mathcal{S}$.

Proof: On input (**prove**,sid, pid, x, w) from party P ignore if $(x, w) \notin R$. Send (**prove**,x) to $\mathcal{S}$ and wait for answer (**proof**, π). Upon receiving the answer store (x, π) and send (**proof**, sid, pid, π) to P.

Verification: On input (**verify**, sid, pid, x, π) from V check whether (x, π) is stored. If not send (**verify**,x, π) to $\mathcal{S}$ and wait for an answer (**witness**,w). Upon receiving the answer, check whether $(x, w) \in R$ and in that case, store (x, π). If (x, π) has been stored return (**verification**,sid, pid,1) to V, else return (**verification**,sid, pid,0).

Figure 9: NIZK argument functionality $\mathcal{F}_{\text{NIZK}}$.

We now describe the main theorem that we prove in this section:

Theorem 6.1 *Assuming the existence of a garbling scheme for protocols satisfying semi-malicious security and a standalone semi-honest secure n-party protocol Φ there exists a construction of two round UC secure MPC in the $\mathcal{F}_{\text{NIZK}}$ hybrid model.*

If we instantiate Φ with a semi-honest multiparty protocol for computing a garbled RAM program $\widetilde{P}$ (using the garbling function in say, [GLOS15]) we obtain the following corollary.

Corollary 6.2 *Assuming the existence of a garbling scheme for protocols satisfying the semi-malicious security and standalone semi-honest secure n-party protocol Φ for evaluating RAM programs there exists a construction of two round UC secure MPC for evaluating RAM programs in the $\mathcal{F}_{\text{NIZK}}$ hybrid model.*

We give definitions of universal composable security [Can01] in Appendix A.

Construction. We give the formal description of our two round MPC in the NIZK hybrid model in Figure 10. We assume that the standalone protocol is semi-malicious. In fact, the GMW semi-honest protocol [GMW87] can be shown to satisfy semi-malicious security.

Let Φ be a n-party stand alone semi-malicious secure protocol computing the function f and let $(\mathsf{Setup}, \mathsf{Garble}, \mathsf{Eval})$ be a garbling scheme for protocols. We describe a two-round protocol Π in the $\mathcal{F}_{\mathrm{NIZK}}$ model.

Ideal NIZK Functionality. The $\mathcal{F}_{\mathrm{NIZK}}$ ideal functionality is parameterized by the relation R defined as $R((\widetilde{x}_i, (\sigma, i, x_i, \widetilde{\Phi}_i, \{\mathsf{lab}_{j,0}, \mathsf{lab}_{j,1}\}, \omega_i)) = 1$ if and only if $(\widetilde{\Phi}_i, \widetilde{x}_i, \{\mathsf{lab}_{j,0}, \mathsf{lab}_{j,1}\}) \leftarrow \mathsf{Garble}(\sigma, i, x_i, \Phi_i; \omega_i)$.

Private Inputs: Party P_i for $i \in [n]$, receives its private input x_i, a session id sid.

Common Reference String: Let $\sigma \leftarrow \mathsf{Setup}(1^\lambda)$ and output σ as the common reference string.

Round 1: Each party P_i does the following:

1. Choose a uniform ω_i as the random tape for the Garble procedure.
2. Compute $(\widetilde{\Phi}_i, \widetilde{x}_i, \{\mathsf{lab}_{j,0}, \mathsf{lab}_{j,1}\}) \leftarrow \mathsf{Garble}(\sigma, i, x_i, \Phi_i; \omega_i)$.
3. Send $(\mathbf{prove}, sid, i, (\widetilde{x}_i, (\sigma, i, x_i, \widetilde{\Phi}_i, \{\mathsf{lab}_{j,0}, \mathsf{lab}_{j,1}\}, \omega_i))$ to $\mathcal{F}_{\mathrm{NIZK}}$. Receive the the message $(\mathbf{proof}, sid, i, \pi_i)$ from $\mathcal{F}_{\mathrm{NIZK}}$.
4. Send $(\widetilde{x}_i, \pi_i)$ to every other party.

Round 2: Each party P_i does the following:

1. For each $j \in [n] \setminus \{i\}$, verify if the proof π_j is valid by sending $(\mathbf{verify}, sid, j, \widetilde{x}_j, \pi_j)$ to $\mathcal{F}_{\mathrm{NIZK}}$. If $\mathcal{F}_{\mathrm{NIZK}}$ responds with $(\mathbf{verification}, sid, j, 0)$ then abort.
2. Send $\widetilde{\Phi}_i$ and $\{\mathsf{lab}^i_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}\}$ to every other party.

Evaluation: Every party P_i computes $y := \mathsf{Eval}(\{\widetilde{\Phi}_i\}, \{\widetilde{x}_i\}, \{\mathsf{lab}^i_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}\})$ and computes the output of the functionality f from y.

Figure 10: Two-round Multi-Party Computation Protocol

6.1 Description of the Simulator

In this subsection we give the description of the ideal world adversary $\mathcal{S}$ having access to the ideal functionality $\mathcal{F}_f$ that simulates the view of the real world adversary $\mathcal{A}$. $\mathcal{S}$ will internally use the simulators S_Φ for the semi-malicious security of Φ and (S_1, S_2) for the garbling scheme for protocols.

We assume that $\mathcal{A}$ is static and hence the set of honest parties H is known before the execution of the protocol.

Simulating the CRS. To simulate the common reference string, $\mathcal{S}$ runs S_1 on input 1^λ and H and obtains σ, the set of input encodings $\{\widetilde{x}_i\}_{i \in H}$ for the honest parties and the secret simulation state st_S. It set the common reference string to be σ and locally stores $\{\widetilde{x}_i\}_{i\in H}$ and st_S.

Simulating the interaction with $\mathcal{Z}$. For every input value for the set of corrupted parties that $\mathcal{S}$ receives from $\mathcal{Z}$, $\mathcal{S}$ writes that value to $\mathcal{A}$'s input tape. Similarly, the output of $\mathcal{A}$ is written as the output on $\mathcal{S}$'s output tape.

Simulating the interaction with $\mathcal{A}$: For every concurrent interaction with the session identifier sid that $\mathcal{A}$ may start, the simulator does the following:

- **Round-1 messages from $\mathcal{S}$ to $\mathcal{A}$:** $\mathcal{S}$ recovers $\{\widetilde{x}_i\}_{i\in H}$ from its local storage. For each $i \in H$, $\mathcal{S}$ samples π_i from the appropriate distribution and sends $(\widetilde{x}_i, \pi_i)$ to $\mathcal{A}$ on behalf of the honest party i. It then stores $(\widetilde{x}_,\pi_i)$ in its local storage.
- **Round-1 messages from $\mathcal{A}$ to $\mathcal{S}$:** $\mathcal{S}$ receives the message $(\mathbf{prove}, sid, i, (\widetilde{x}_i, (\sigma, i, x_i, \widetilde{\Phi}_i, \{\mathsf{lab}_{j,0}, \mathsf{lab}_{j,1}\}, \omega_i))$ that $\mathcal{A}$ sends to the ideal NIZK functionality for an $i \notin H$. $\mathcal{S}$ checks if $R((\widetilde{x}_i, (\sigma, i, x_i, \widetilde{\Phi}_i, \{\mathsf{lab}_{j,0}, \mathsf{lab}_{j,1}\}, \omega_i)) = 1$. If it is not the case, it ignores this message. Else it locally stores, $(\widetilde{x}_i, x_i, \omega_i)$ and sends the message $(\mathbf{proof}, \widetilde{x}_i)$ to $\mathcal{A}$ and receives a message $(\mathbf{proof}, \pi_i)$. It stores $(\widetilde{x}_i, \pi_i)$.
- **Round-2 messages from $\mathcal{S}$ to $\mathcal{A}$:**
  1. **for** every $i \in H$,
     (a) $\mathcal{S}$ receives the message $\{\widetilde{x}_j, \pi_j\}_{j\notin H}$ from $\mathcal{A}$ on behalf of i.
     (b) For every $j \notin H$, $\mathcal{S}$ checks if the value $(\widetilde{x}_j, \pi_j)$ is stored. If such a value does not exist, $\mathcal{S}$ locally stores $(i, \mathbf{abort})$.
  2. If for every $i \in H$, if $(i, \mathbf{abort})$ is stored, $\mathcal{S}$ aborts the execution.
  3. Else, for every $j \notin H$, $\mathcal{S}$ recovers the value $(\widetilde{x}_j, x_j, \omega_j)$ from its local storage and constructs the initial state y_j comprising of the input x_j and the randomness s_j of party j in the computation of Φ.
  4. $\mathcal{S}$ queries the ideal functionality $\mathcal{F}_f$ with the query $(\mathbf{input}, sid, j, x_j)$ for every $j \notin H$ and obtains the string z which is the output of the functionality.
  5. $\mathcal{S}$ then runs the simulator S_Φ on inputs $\{y_j\}_{j\notin H}$ and the output z and obtains the simulated transcript τ.
  6. It then runs S_2 on input the secret state st_S (recovered from its local storage), $\{y_j, \widetilde{x}_j\}_{j\notin H}$ and the simulated transcript τ to obtain $\{\widetilde{\Phi}_i, \mathsf{lab}_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}\}_{i\in H}$.
  7. For every i such that there does not exist $(i, \mathbf{abort})$ in its local storage, $\mathcal{S}$ forwards $\widetilde{\Phi}_i, \mathsf{lab}_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}$ on behalf of i.
- **Round-2 messages from $\mathcal{A}$ to $\mathcal{S}$:** For every $i \in H$, $\mathcal{S}$ obtains the second round message from $\mathcal{A}$ on behalf of the honest party. For every $i \in H$, if the set of message obtained from $\mathcal{A}$ is well formed, $\mathcal{S}$ sends $(\mathbf{generateOutput}, sid, i)$ to the trusted party $\mathcal{F}_f$.

We show that:

Lemma 6.3 *Assuming the semi-malicious security of Φ and the security of garbling scheme for protocols for any environment $\mathcal{Z}$ that obeys the rules of interaction for UC security we have* $\mathrm{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.

6.2 Proof of Lemma 6.3

We prove the lemma via a hybrid argument.

$\underline{\mathsf{Hybrid}_1}$: This corresponds to the real world execution where the environment $\mathcal{Z}$ is interacting with the real world adversary $\mathcal{A}$. Alternatively, we can view this hybrid in the ideal world where the ideal world adversary $\mathcal{S}$ additionally has access to the private inputs of the honest parties and interacts with $\mathcal{A}$. $\mathcal{S}$ generates the messages of the honest parties as given in the description of the protocol. It also simulates the ideal world functionality $\mathcal{F}_{\text{NIZK}}$ to $\mathcal{A}$.

$\underline{\mathsf{Hybrid}_2}$: In this hybrid the ideal world adversary $\mathcal{S}$ invokes the simulator of the garbling scheme for protocols instead of generating the round-1 and round-2 messages honestly. To give more details, $\mathcal{S}$ runs S_1 on input 1^λ and H, to obtain the common reference string σ, the set of input encodings $\{\widetilde{x}_i\}_{i\in H}$ corresponding to the honest parties and the secret simulation state st_S. It sets the common reference string as σ and forwards $\{\widetilde{x}_i\}_{i\in H}$ to $\mathcal{A}$ on behalf of the honest parties. It obtains $\{\widetilde{x}_j\}_{j\notin H}$ and recovers the initial input state y_j and the input x_j for each $j \in H$ through its simulation of the ideal NIZK functionality. It executes Φ in "its head" using its knowledge of the honest parties inputs and using the extracted inputs of the corrupted parties to obtain the transcript $\Phi(x_1, \ldots, x_n)$. It then runs the simulator S_2 on input the secret state st_S, $\{y_j, \widetilde{x}_j\}_{j\notin H}$ and the transcript $\Phi(x_1, \ldots, x_n)$ to obtain $\{\widetilde{\Phi}_i, \mathsf{lab}_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}\}_{i\in H}$. It then forwards this to $\mathcal{A}$.

Notice that the view of the adversary in Hybrid_1 and Hybrid_2 is computationally indistinguishable from the semi-malicious security of garbling scheme for protocols.

$\underline{\mathsf{Hybrid}_3}$: In this hybrid, the ideal world adversary uses the simulator for Φ to generate the transcript of the protocol. The ideal world adversary $\mathcal{S}$ recovers the the initial input state y_j and the input x_j for each $j \in H$ as in the previous hybrid. It queries the ideal world functionality $\mathcal{F}_f$ on input $(\mathbf{input}, sid, j, x_j)$ for each $j \in [n]$ and obtains the output z. It then runs the simulator S_Φ on inputs $\{y_j\}_{j\notin H}$ and z to obtain the simulated transcript τ. It then uses τ as input while running S_2. The rest of the simulation is exactly as in the previous hybrid.

Notice that the view of the adversary in Hybrid_2 and Hybrid_3 is computationally indistinguishable from the semi-malicious security of Φ. Hybrid_3 is distributed identically to $\text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

7 Black-box Construction of Two-Round MPC

In this section we explain how to augment the construction of two-round MPC so that it makes black-box use of the underlying group. The main theorem we prove in this section is:

Theorem 7.1 *Assuming the sub-group decision assumption and the computational Diffie-Hellman assumption there exists a construction of UC secure two round MPC that makes black-box use of the underlying groups.*

Before proceeding we define a stronger security property of the garbling scheme for protocols which will be used in the construction of black-box MPC.

$\mathsf{extReal}[1^\lambda, \{x_i\}_{i\in H}, H]$	$\mathsf{extIdeal}[1^\lambda, \{x_i\}_{i\in H}, H]$
1. $\sigma \leftarrow \mathsf{Setup}(1^\lambda)$ and for every $i \in H$, compute $(\widetilde{\Phi}_i, \widetilde{x}_i, \{\mathsf{lab}^i_{j,b}\}) \leftarrow \mathsf{Garble}(\sigma, i, \Phi_i, x_i)$	1. $(\sigma, \{\widetilde{x}_i\}_{i\in H}, \mathsf{st}_S) \leftarrow S_1(1^\lambda, H)$
2. $\{x_i, \tilde{x}_i\}_{i\notin H}, \mathsf{st}_\mathcal{A} \leftarrow \mathcal{A}_1(\sigma, \{\widetilde{x}_i\}_{i\in H})$.	2. $\{x_i, \tilde{x}_i\}_{i\notin H}, \mathsf{st}_\mathcal{A} \leftarrow \mathcal{A}_1(\sigma, \{\widetilde{x}_i\}_{i\in H})$.
3. Output $\mathcal{A}_2(\mathsf{st}_\mathcal{A}, \{\widetilde{\Phi}_i, \widetilde{x}_i, \mathsf{lab}^i_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}\}_{i\in H})$	3. Output $\mathcal{A}_2\left(\mathsf{st}_\mathcal{A}, S_2(\mathsf{st}_S, \{\widetilde{x}_j\}_{j\notin H}, \Phi(x_1, \ldots, x_n))\right)$

Figure 11: Semi-Malicious Real and Ideal world for Garbling Protocols

7.1 Extractable Semi-Malicious Security

In this subsection we define a stronger security notion for garbling scheme for protocols, namely, extractable semi-malicious security. The difference between this notion and the semi-malicious security (Definition 5.3) is that the simulator is only provided with the encodings $\{\widetilde{x}_i\}_{i\notin H}$ of the corrupted parties and not provided with their input $\{x_i\}_{i\notin H}$. We define this notion formally below.

Definition 7.2 (Extractable Semi-malicious Security) *A garbling scheme for protocols is said to satisfy extractable semi-malicious security if there exists a PPT algorithm $S = (S_1, S_2)$ such that for every protocol Φ, and every subset $H \subseteq [n]$ of honest parties, and for every choice of inputs $\{x_i\}_{i\in H}$ for honest parties, we have that for every admissible PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,*

$$\left| \Pr\left[\mathsf{extReal}[1^\lambda, \{x_i\}, H] = 1\right] - \Pr\left[\mathsf{extIdeal}[1^\lambda, \{x_i\}, H] = 1\right] \right| \leq \mathsf{negl}(\lambda)$$

where extReal *and* extIdeal *games are described in Figure 11.*

In Appendix B we give a construction of garbling scheme for protocols that satisfies extractable semi-malicious security and additionally makes black-box use of a homomorphic proof commitment with encryption. A key difference between this construction and the construction given in Figure 6 is in the structure of the input encoding $\widetilde{x}_i$. Recall that in the previous construction $\widetilde{x}_i$ consists of commitments $\{c_{i,k}\}_{k\in[\ell]}$. Now, the input encoding contains an additional component $\{\pi_{i,k}\}_{k\in[\ell]}$ where $\pi_{i,k}$ is a proof that $c_{i,k}$ is a commitment to a message in $\{0, 1\}$. Additionally, the size of the common reference string σ grows with the number of parties.

Remark 7.3 *We note that the construction given in Appendix B can be modified to satisfy semi-honest security in the plain model by asking every party to generate the common random/reference string in the binding mode. Later, in the proof of security we will change the reference strings of the honest parties to the hiding mode.*

7.2 $\mathcal{F}_{\text{NIOT}}$ functionality

In this section we give a protocol for realizing the non-interactive oblivious transfer $\mathcal{F}_{\text{NIOT}}$ functionality defined in Figure 12. Looking ahead, the $\mathcal{F}_{\text{NIOT}}$ functionality will be used to generate OT correlations which then can be used to realize information theoretic MPC protocols [Kil88, IPS08] with security against malicious behavior. We will use such protocols in our black-box MPC construction.

Parametrized with parties $P_1, \ldots, P_n$ and adversary $\mathcal{S}$ controlling a subset of the parties. Let H be the set of parties not controlled by the adversary.

On receiving (sid, pid_1, pid_2) from a party with id pid_1, check if pid_1 or pid_2 is in H.

If both $pid_1, pid_2 \in H$, sample $(s_0, s_1, c) \leftarrow \{0,1\}$, send (s_0, s_1) to the party pid_1 and (c, s_c) to the party pid_2.

If $pid_1 \notin H$ but $pid_2 \in H$ then send the message $(\textbf{sender}, pid_1)$ to $\mathcal{S}$ and receive (s_0, s_1) from $\mathcal{S}$. Sample $c \leftarrow \{0,1\}$ and send (c, s_c) to the party pid_2.

If $pid_1 \in H$ but $pid_2 \notin H$, send the message $(\textbf{receiver}, pid_2)$ to $\mathcal{S}$ and receive (c, s_c) from $\mathcal{S}$. Sample $s_{1-c} \leftarrow \{0,1\}$ and send (s_0, s_1) to the party pid_1.

if both $pid_1, pid_2 \notin H$, ignore the message.

Figure 12: Non-Interactive Oblivious Transfer functionality $\mathcal{F}_{\text{NIOT}}$.

Construction. We give a protocol for realizing the $\mathcal{F}_{\text{NIOT}}$ functionality in Figure 13. At a high level, we augment the non-interactive oblivious transfer protocol of Bellare and Micali [BM90] with Groth-Sahai proofs [GS12] that enables the simulator to extract the sender bits or the receiver's choice bit in the simulation. We use Groth-Sahai proofs so that it enables us to give proof about equations over groups while making black-box use of the underlying group.

Let $(K_{GS,\text{binding}}, K_{GS,\text{hiding}}, P, V)$ be the Groth-Sahai proof system for proving equations over bilinear groups. Let $\text{Setup}_{\text{CDH}}$ on input 1^λ give the description of groups $\mathbb{G}, \mathbb{G}_T$ with prime order p, a generator g for g and a bilinear map $e : \mathbb{G}\times\mathbb{G} \to \mathbb{G}_T$ such that the computational Diffie-Hellman is hard to solve in $\mathbb{G}$.

Inputs: Party P_i for $i \in [n]$, receives a session id sid.

Common Reference String: Let $(\mathbb{G}, \mathbb{G}_T, g, p, e) \leftarrow \text{Setup}_{\text{CDH}}(1^\lambda)$. Let $x \leftarrow \mathbb{Z}$ and set $X := g^x$. For each party $i \in [n]$, sample $\sigma_i \leftarrow K_{GS,\text{binding}}(1^\lambda)$. The common reference string consists of $(\mathbb{G}, \mathbb{G}_T, p, g, e, X, \sigma_1, \ldots, \sigma_n)$.

Let us assume that P_i is the sender and P_j is the receiver.

Message sent by $P_i \to P_j$**:** Sample $a \leftarrow \mathbb{Z}_p^*$ and compute $A := g^a$. Let π_A be the GS proof for the equation that there exists an a such that $g^a = A$ using σ_i as the crs. Send (A, π_a) to P_j.

Message sent by $P_j \to P_i$**:** Sample $b \leftarrow \mathbb{Z}_p^*$ and $c \leftarrow \{0,1\}$. Compute $g_0 := (1-c)g^b + c(\frac{X}{g^b})$ and $g_1 := cg^b + (1-c)(\frac{X}{g^b})$. Let π_{g_0,g_1} be a GS proof for the equation that there exists a group element B such that $e(g_0 B, g_1 B) = 1$ using σ_j as the crs. Send $(g_0, g_1, \pi_{g_0,g_1})$ to P_i.

Computation: P_i verifies the proof π_{g_0,g_1} and sets $(s_0, s_1) := (h(g_0^a), h(g_1^a))$ where h is the hardcore predicate. P_j verifies the proof π_A and sets $(c, s_c) := (c, h(A^b))$.

Figure 13: Non-Interactive Oblivious Transfer

Description of the Simulator. We assume that $\mathcal{A}$ is static and hence the set of honest parties H is known before the execution of the protocol.

Simulating the CRS. To simulate the common reference string, $\mathcal{S}$ samples $(\mathbb{G}, \mathbb{G}_T, g, p, e) \leftarrow \text{Setup}_{\text{CDH}}(1^\lambda)$. It then chooses $x \leftarrow \mathbb{Z}_p^*$ and sets $X := g^x$. For every $i \in H$, it chooses $(\sigma_i, tk_i) \leftarrow K_{GS,\text{hiding}}(1^\lambda)$ and for every $i \notin H$ it chooses $(\sigma_i, xk_i) \leftarrow K_{GS,\text{binding}}(1^\lambda)$. It sets the common reference string to be $(\mathbb{G}, \mathbb{G}_T, p, g, e, X, \sigma_1, \ldots, \sigma_n)$

Simulating the interaction with $\mathcal{Z}$. For every input value for the set of corrupted parties that $\mathcal{S}$ receives from $\mathcal{Z}$, $\mathcal{S}$ writes that value to $\mathcal{A}$'s input tape. Similarly, the output of $\mathcal{A}$ is written as the output on $\mathcal{S}$'s output tape.

Simulating the interaction with $\mathcal{A}$: For every concurrent interaction with the session identifier sid that $\mathcal{A}$ may start and for every choice of sender and the receiver P_i and P_j respectively, the simulator does the following:

1. **Both P_i and P_j are honest:**
   (a) Simulator chooses a random $a \leftarrow \mathbb{Z}_p^*$ and compute $A := g^a$. It generates a simulated proof π_A for the GS equation that there exists an a such that $g^a = A$ using σ_i as the crs and tk_i as the trapdoor information. It sends (A, π_A) to P_j on behalf of the honest P_j.
   (b) Simulator samples $b \leftarrow \mathbb{Z}_p^*$ and $c \leftarrow \{0,1\}$. It computes $g_0 := (1-c)g^b + c(\frac{X}{g^b})$ and $g_1 := cg^b + (1-c)(\frac{X}{g^b})$. It generates a simulated proof π_{g_0,g_1} for the equation that there exists a group element B such that $e(g_0B, g_1B) = 1$ using σ_j as the crs and tk_j as the trapdoor information. It then sends $(g_0, g_1, \pi_{g_0,g_1})$ to P_i on behalf of the honest party P_j.

2. **P_i is corrupted and P_j is honest:**
   (a) Simulator samples $b \leftarrow \mathbb{Z}_p^*$ and $c \leftarrow \{0,1\}$. It computes $g_0 := (1-c)g^b + c(\frac{X}{g^b})$ and $g_1 := cg^b + (1-c)(\frac{X}{g^b})$. It generates a simulated proof π_{g_0,g_1} for the equation that there exists a group element B such that $e(g_0B, g_1B) = 1$ using σ_j as the crs and tk_j as the trapdoor information. It then sends $(g_0, g_1, \pi_{g_0,g_1})$ to P_i on behalf of the honest party P_j.
   (b) It receives the element A and the proof π_A that $\mathcal{A}$ sends on behalf of the corrupted party P_i. Using the extraction key xk_i, simulator recovers the witness a used in the computation of π_A. It computes $(h(g_0^a), h(g_1^a))$ and sends them to the ideal functionality $\mathcal{F}_{\text{NIOT}}$.

3. **P_j is corrupted but P_i is honest:**
   (a) Simulator chooses a random $a \leftarrow \mathbb{Z}_p^*$ and computes $A := g^a$. It generates a simulated proof π_A for the GS equation that there exists an a such that $g^a = A$ using σ_i as the crs and tk_i as the trapdoor information. It sends (A, π_a) to P_j on behalf of the honest party P_i.

(b) Simulator receives the elements g_0, g_1, π_{g_0,g_1} from the adversary $\mathcal{A}$. Using the extraction key xk_j simulator recovers the witness B such that $e(g_0 B, g_1 B) = 1$. It sets c to be the bit in $\{0,1\}$ such that $g_c B = 1$. It computes g_c^a and sends $(c, h(g_c^a))$ to the ideal functionality $\mathcal{F}_{\text{NIOT}}$.

We show that:

Lemma 7.4 *Assuming the computational Diffie-Hellman assumption and the security of the GS proof system, for every $\mathcal{Z}$ that obeys the rules of interaction for UC security we have* $\text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.

Proof of Lemma 7.4. We now show that no environment can distinguish the real world execution with adversary $\mathcal{A}$ and an ideal world execution with adversary $\mathcal{S}$. We consider three cases.

1. **Case-1:** Both P_i and P_j are honest. In this case the real world and the ideal world distributions are identical except that the proof π_A and π_{g_0,g_1} are simulated and the crs for the honest parties is generated in the hiding mode.
   $\underline{\mathsf{Hybrid}_1}$: In this hybrid, we change the crs for the honest parties $i \in H$ to be the hiding mode. Indistinguishability from the real world execution follows from the crs indistinguishability property of the GS proof systems.
   $\underline{\mathsf{Hybrid}_2}$: In this hybrid, we change the proofs generated by the honest parties to be simulated rather than being generated honestly. Indistinguishability from the previous hybrid follows from the zero-knowledge proof of the GS proof systems.

   Notice that in Hybrid_2 the view of the adversary is computationally independent of the bits (s_0, s_1) from the computational Diffie-Hellman assumption and is statistically independent of the bit c.

2. **Case-2:** P_i is corrupted and P_j is honest. In this case the real world and the ideal world distributions are identical except that the proof π_{g_0,g_1} is simulated and the crs for the honest parties is generated in the hiding mode.
   $\underline{\mathsf{Hybrid}_1}$: In this hybrid, we change the crs for the honest parties $i \in H$ to be the hiding mode. Indistinguishability from the real world execution follows from the crs indistinguishability property of the GS proof systems.
   $\underline{\mathsf{Hybrid}_2}$: In thus hybrid we change the the proof π_{g_0,g_1} to be simulated instead of generating it honestly. Indistinguishability from the previous hybrid follows from the zero-knowledge proof of the GS proof systems.
   Note that the extracted bits $(h(g_0^a), h(g_1^a))$ in Hybrid_2 is computationally indistinguishable to the bits in the real world view of the adversary.

3. **Case-3:** P_j is corrupted and P_i is honest. In this case the real world and the ideal world distributions are identical except that the proof π_A is simulated and the crs for the honest parties is generated in the hiding mode.
   $\underline{\mathsf{Hybrid}_1}$: In this hybrid, we change the crs for the honest parties $i \in H$ to be the hiding mode. Indistinguishability from the real world execution follows from the crs indistinguishability property of the GS proof systems.
   $\underline{\mathsf{Hybrid}_2}$: In thus hybrid we change the the proof π_A to be simulated instead of generating it honestly. Indistinguishability from the previous hybrid follows from the zero-knowledge proof

of the GS proof systems.

Note that the extracted bit c in Hybrid_2 is computationally indistinguishable to the bit in the real world view of the adversary. Now in Hybrid_2, $h(g_{1-c}^a)$ is indistinguishable from a random bit from the Computational Diffie-Hellman assumption.

7.3 Black-box Construction of Two-round MPC

We give the construction of two round MPC that makes black-box use of the underlying group in Figure 14. The protocol uses an extractable semi-malicious secure garbling scheme for protocols ($\mathsf{Setup}, \mathsf{Garble}, \mathsf{Eval}$). For the sake of exposition, we split the Garble procedure into ($\mathsf{Encode}, \mathsf{GarbProt}$) each having a separate random tape. Such a construction of garbling scheme appears in Figure 18 and Figure 19 in Appendix B. Furthermore, the construction in Figure 18 is augmented to make black-box use of the underlying group as described in Section B.3.

Security. The description of the simulator and the hybrid arguments is almost identical to the previous construction. We give the details in Appendix C for the sake of completeness.

Let Φ be a n-party stand alone semi-malicious secure protocol computing the function f in the OT hybrid model and let $(\mathsf{Setup}, \mathsf{Garble}, \mathsf{Eval})$ be a garbling scheme for protocols satisfying the extractable semi-malicious security. We describe a two-round protocol Π computing f.

Ideal $\mathcal{F}_{\mathrm{NIOT}}$ functionality. Parameterized by the parties $P_1, \ldots, P_n$

Private Inputs: Party P_i for $i \in [n]$, receives its private input x_i, a session id sid.

Common Reference String: Let $\sigma \leftarrow \mathsf{Setup}(1^\lambda)$ and output σ as the common reference string.

Round 1: Each party P_i does the following:

1. Choose a uniform ω_i as the random tape for Encode procedure.
2. Compute $\widetilde{x}_i \leftarrow \mathsf{Encode}(\sigma, i, x_i; \omega_i)$.
3. Send $\widetilde{x}_i$ to every other party. For every OT invocation in Φ where P_i acts as the sender, send (sid, i, pid_2) where pid_2 is the receiver id in the OT interaction.

Round 2: Each party P_i does the following:

1. Receive the set of bits $\{(s_{k,0}, s_{k,1})\}$ for every OT invocation where P_i acts as the sender and the set $\{c, s_{l,c}\}$ for every OT invocation where P_i acts as the receiver from the ideal functionality $\mathcal{F}_{\mathrm{NIOT}}$.
2. Parse σ as $\{ck_i\}$. For each $j \in [n] \setminus \{i\}$, parse $\widetilde{x}_j$ as $\{c_{j,k}, \pi_{j,k}\}_{k \in [\ell]}$. Check if for every $j \in [n] \setminus \{i\}$ and $k \in [\ell]$, $V_{01}(ck_j, c_{j,k}, \pi_{j,k}) = 1$ and for every $k \in B$, if $c_{j,k} := \mathrm{com}(ck_j, 0; 0^\lambda)$. If any of the checks fail, abort.
3. Compute $(\widetilde{\Phi}_i, \mathsf{lab}^i_{\widetilde{x}_1 \| \ldots \| \widetilde{x}_n}) \leftarrow \mathsf{GarbProt}(\sigma, i, \Phi_i, \{\widetilde{x}_j\})$ where Φ_i has the correlations $\{(s_{k,0}, s_{k,1})\}$ and $\{c, s_{l,c}\}$ hardwired.
4. Send $\widetilde{\Phi}_i$ and $\{\mathsf{lab}^i_{\widetilde{x}_1 \| \ldots \| \widetilde{x}_n}\}$ to every other party.

Evaluation: Every party P_i computes $y := \mathsf{Eval}(\{\widetilde{\Phi}_i\}, \{\widetilde{x}_i\}, \{\mathsf{lab}^i_{\widetilde{x}_1 \| \ldots \| \widetilde{x}_n}\})$ and computes the output of the functionality from y.

Figure 14: Two-round Multi-Party Computation Protocol making black-box use of the underlying group

References

[AF90] Martín Abadi and Joan Feigenbaum. Secure circuit evaluation. *Journal of Cryptology*, 2(1):1–12, 1990.

[AIK04] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th FOCS*, pages 166–175, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press.

[AIK11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 120–129, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press.

[AIR01] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 119–135, Innsbruck, Austria, May 6–10, 2001. Springer, Heidelberg, Germany.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[BB04] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 443–459, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany.

[BCL+05] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 361–377, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Heidelberg, Germany.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 509–539, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[BGI17] Elette Boyle, Niv Gilboa, and Yuval Ishai. Group-based secure computation: Optimizing rounds, communication, and computation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 163–193, Paris, France, May 8–12, 2017. Springer, Heidelberg, Germany.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 325–341, Cambridge, MA, USA, February 10–12, 2005. Springer, Heidelberg, Germany.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy*, pages 478–492, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 784–796, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

[BM90] Mihir Bellare and Silvio Micali. Non-interactive oblivious transfer and applications. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 547–557, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10, Chicago, IL, USA, May 2–4, 1988. ACM Press.

[BP16] Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 190–213, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.

[Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, page 219, 2004.

[CDG+17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 33–65, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503, Montréal, Québec, Canada, May 19–21, 2002. ACM Press.

[CLP10] Ran Canetti, Huijia Lin, and Rafael Pass. Adaptive hardness and composable security in the plain model from standard assumptions. In *51st FOCS*, pages 541–550, Las Vegas, NV, USA, October 23–26, 2010. IEEE Computer Society Press.

[CM15] Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled FHE from learning with errors. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 630–656, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[CR73] Stephen A. Cook and Robert A. Reckhow. Time bounded random access machines. *J. Comput. Syst. Sci.*, 7(4):354–375, 1973.

[DG17] Nico Döttling and Sanjam Garg. Identity-based encryption from the diffie-hellman assumption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 537–569, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *26th ACM STOC*, pages 554–563, Montréal, Québec, Canada, May 23–25, 1994. ACM Press.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 74–94, San Diego, CA, USA, February 24–26, 2014. Springer, Heidelberg, Germany.

[GGMP16] Sanjam Garg, Divya Gupta, Peihan Miao, and Omkant Pandey. Secure multiparty RAM computation in constant rounds. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 491–520, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[GHL+14] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled RAM revisited. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 405–422, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-Hop homomorphic encryption and rerandomizable Yao circuits. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 155–172, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany.

[GKK+12] S. Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure two-party computation in sublinear (amortized) time. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 513–524, Raleigh, NC, USA, October 16–18, 2012. ACM Press.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 39–56, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany.

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. Cryptology ePrint Archive, Report 2017/274, 2017. http://eprint.iacr.org/2017/274.

[GLNP15] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 567–578, Denver, CO, USA, October 12–16, 2015. ACM Press.

[GLO15] Sanjam Garg, Steve Lu, and Rafail Ostrovsky. Black-box garbled RAM. In Venkatesan Guruswami, editor, *56th FOCS*, pages 210–229, Berkeley, CA, USA, October 17–20, 2015. IEEE Computer Society Press.

[GLOS15] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled RAM from one-way functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 449–458, Portland, OR, USA, June 14–17, 2015. ACM Press.

[GLS15] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 63–82, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, Cambridge, UK, 2001.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for np. In *Proceedings of Eurocrypt 2006, volume 4004 of LNCS*, pages 339–358. Springer, 2006.

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3):11:1–11:35, 2012.

[GS12] Jens Groth and Amit Sahai. Efficient noninteractive proof systems for bilinear groups. *SIAM J. Comput.*, 41(5):1193–1232, 2012.

[HK12] Shai Halevi and Yael Tauman Kalai. Smooth projective hashing and two-message oblivious transfer. *Journal of Cryptology*, 25(1):158–193, January 2012.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany.

[Jou04] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276, September 2004.

[Kil88] Joe Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31, Chicago, IL, USA, May 2–4, 1988. ACM Press.

[KMR14] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 440–457, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[KO04] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 335–354, Santa Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498, Reykjavik, Iceland, July 7–11, 2008. Springer, Heidelberg, Germany.

[LO13] Steve Lu and Rafail Ostrovsky. How to garble RAM programs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 719–734, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 1219–1234, New York, NY, USA, May 19–22, 2012. ACM Press.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th SODA*, pages 448–457, Washington, DC, USA, January 7–9, 2001. ACM-SIAM.

[OS97] Rafail Ostrovsky and Victor Shoup. Private information storage (extended abstract). In *29th ACM STOC*, pages 294–303, El Paso, TX, USA, May 4–6, 1997. ACM Press.

[Ost90] Rafail Ostrovsky. Efficient computation on oblivious RAMs. In *22nd ACM STOC*, pages 514–523, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

[PF79] Nicholas Pippenger and Michael J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979.

[PS16] Chris Peikert and Sina Shiehian. Multi-key FHE from LWE, revisited. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 217–238, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 250–267, Tokyo, Japan, December 6–10, 2009. Springer, Heidelberg, Germany.

[Rab81] Michael O. Rabin. How to exchange secrets with oblivious transfer, 1981.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under lwe. Cryptology ePrint Archive, Report 2017/276, 2017. http://eprint.iacr.org/2017/276.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164, Chicago, Illinois, November 3–5, 1982. IEEE Computer Society Press.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

A UC Security

In this section we briefly review UC security. For full details see [Can01]. A large part of this introduction has been taken verbatim from [CLP10].

A.1 The basic model of execution

Following [GMR88, Gol01], a protocol is represented as an interactive Turing machine (ITM), which represents the program to be run within each participant. Specifically, an ITM has three tapes that can be written to by other ITMs: the input and subroutine output tapes model the inputs from and the outputs to other programs running within the same "entity" (say, the same physical computer), and the incoming communication tapes and outgoing communication tapes model messages received from and to be sent to the network. It also has an identity tape that cannot be written to by the ITM itself. The identity tape contains the program of the ITM (in some standard encoding) plus additional identifying information specified below. Adversarial entities are also modeled as ITMs.

We distinguish between ITMs (which represent static objects, or programs) and *instances of ITMs*, or ITIs, that represent interacting processes in a running system. Specifically, an ITI is an ITM along with an identifer that distinguishes it from other ITIs in the same system. The identifier consists of two parts: A session-identifier (SID) which identifies which protocol instance the ITM belongs to, and a party identifier (PID) that distinguishes among the parties in a protocol instance. Typically the PID is also used to associate ITIs with "parties", or clusters, that represent some administrative domains or physical computers.

The model of computation consists of a number of ITIs that can write on each other's tapes in certain ways (specified in the model). The pair (SID,PID) is a unique identifier of the ITI in the system.

With one exception (discussed within) we assume that all ITMs are probabilistic polynomial time (PPT). An ITM is PPT if there exists a constant $c > 0$ such that, at any point during its run, the overall number of steps taken by M is at most m^c, where m is the overall number of bits written on the *input tape* of M in this run. (In fact, in order to guarantee that the overall protocol execution process is bounded by a polynomial, we define m as the total number of bits written to the input tape of M, *minus the overall number of bits written by M to input tapes of other ITMs.*; see [Can01].)

A.2 Security of protocols

Protocols that securely carry out a given task (or, protocol problem) are defined in three steps, as follows. First, the process of executing a protocol in an adversarial environment is formalized. Next, an "ideal process" for carrying out the task at hand is formalized. In the ideal process the parties do not communicate with each other. Instead they have access to an "ideal functionality," which is essentially an incorruptible "trusted party" that is programmed to capture the desired functionality of the task at hand. A protocol is said to securely realize an ideal functionality if the process of running the protocol amounts to "emulating" the ideal process for that ideal functionality. Below we overview the model of protocol execution (called the *real-life model*), the ideal process, and the notion of protocol emulation.

The model for protocol execution. The model of computation consists of the parties running an instance of a protocol Π, an adversary $\mathcal{A}$ that controls the communication among the parties, and an *environment* $\mathcal{Z}$ that controls the inputs to the parties and sees their outputs. We assume that all parties have a security parameter $\lambda \in \mathbb{N}$. (We remark that this is done merely for convenience and is not essential for the model to make sense). The execution consists of a sequence of *activations*, where in each activation a single participant (either $\mathcal{Z}$, $\mathcal{A}$, or some other ITM) is activated, and may write on a tape of at most *one* other participant, subject to the rules below. Once the activation of a participant is complete (i.e., once it enters a special waiting state), the participant whose tape was written on is activated next. (If no such party exists then the environment is activated next.)

The environment is given an external input z and is the first to be activated. In its first activation, the environment invokes the adversary $\mathcal{A}$, providing it with some arbitrary input. In the context of UC security, the environment can from now on invoke (namely, provide input to) only ITMs that consist of a single instance of protocol Π. That is, all the ITMs invoked by the environment must have the same SID and the code of Π.

Once the adversary is activated, it may read its own tapes and the outgoing communication tapes of all parties. It may either deliver a message to some party by writing this message on the party's incoming communication tape or report information to $\mathcal{Z}$ by writing this information on the subroutine output tape of $\mathcal{Z}$. For simplicity of exposition, in the rest of this paper we assume authenticated communication; that is, the adversary may deliver only messages that were actually sent. (This is however not essential as shown in [Can04, BCL+05].)

Once a protocol party (i.e., an ITI running Π) is activated, either due to an input given by the environment or due to a message delivered by the adversary, it follows its code and possibly writes a local output on the subroutine output tape of the environment, or an outgoing message on the adversary's incoming communication tape.

The protocol execution ends when the environment halts. The output of the protocol execution is the output of the environment. Without loss of generality we assume that this output consists of only a single bit.

Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(n,z,r)$ denote the output of the environment $\mathcal{Z}$ when interacting with parties running protocol Π on security parameter λ, input z and random input $r = r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1, r_2, \ldots$ as described above (z and $r_{\mathcal{Z}}$ for $\mathcal{Z}$; $r_{\mathcal{A}}$ for $\mathcal{A}$, r_i for party P_i). Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(n,z)$ random variable describing $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(n,z,r)$ where r is uniformly chosen. Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(n,z)\}_{n\in\mathbb{N},z\in\{0,1\}^*}$.

Ideal functionalities and ideal protocols. Security of protocols is defined via comparing the protocol execution to an *ideal protocol* for carrying out the task at hand. A key ingredient in the ideal protocol is the *ideal functionality* that captures the desired functionality, or the specification, of that task. The ideal functionality is modeled as another ITM (representing a "trusted party") that interacts with the parties and the adversary. More specifically, in the ideal protocol for functionality $\mathcal{F}$ all parties simply hand their inputs to an ITI running $\mathcal{F}$. (We will simply call this ITI $\mathcal{F}$. The SID of $\mathcal{F}$ is the same as the SID of the ITIs running the ideal protocol. (the PID of $\mathcal{F}$ is null.)) In addition, $\mathcal{F}$ can interact with the adversary according to its code. Whenever $\mathcal{F}$ outputs a value to a party, the party immediately copies this value to its own output tape. We call the parties in the ideal protocol dummy parties. Let $\Pi(\mathcal{F})$ denote the ideal protocol for functionality $\mathcal{F}$.

Securely realizing an ideal functionality. We say that a protocol Π *emulates* protocol ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running Π, or it is interacting with S and parties running ϕ. This means that, from the point of view of the environment, running protocol Π is 'just as good' as interacting with ϕ. We say that Π *securely realizes* an ideal functionality $\mathcal{F}$ if it emulates the ideal protocol $\Pi(\mathcal{F})$. More precise definitions follow. A distribution ensemble is called *binary* if it consists of distributions over $\{0, 1\}$.

Definition A.1 *Let Π and ϕ be protocols. We say that Π UC-emulates ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ that obeys the rules of interaction for UC security we have* $\mathrm{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.

Definition A.2 *Let $\mathcal{F}$ be an ideal functionality and let Π be a protocol. We say that Π UC-realizes $\mathcal{F}$ if Π UC-emulates the ideal process $\Pi(\mathcal{F})$.*

A.3 Hybrid protocols

Hybrid protocols are protocols where, in addition to communicating as usual as in the standard model of execution, the parties also have access to (multiple copies of) an ideal functionality. Hybrid protocols represent protocols that use idealizations of underlying primitives, or alternatively make *trust assumptions* on the underlying network. They are also instrumental in stating the universal composition theorem. Specifically, in an $\mathcal{F}$-hybrid protocol (i.e., in a hybrid protocol with access to an ideal functionality $\mathcal{F}$), the parties may give inputs to and receive outputs from an unbounded number of copies of $\mathcal{F}$.

The communication between the parties and each one of the copies of $\mathcal{F}$ mimics the ideal process. That is, giving input to a copy of $\mathcal{F}$ is done by writing the input value on the input tape of that copy. Similarly, each copy of $\mathcal{F}$ writes the output values to the subroutine output tape of the corresponding party. It is stressed that the adversary does not see the interaction between the copies of $\mathcal{F}$ and the honest parties.

The copies of $\mathcal{F}$ are differentiated using their SIDs. All inputs to each copy and all outputs from each copy carry the corresponding SID. The model does not specify how the SIDs are generated, nor does it specify how parties "agree" on the SID of a certain protocol copy that is to be run by them. These tasks are left to the protocol. This convention seems to simplify formulating ideal functionalities, and designing protocols that securely realize them, by freeing the functionality from the need to choose the SIDs and guarantee their uniqueness. In addition, it seems to reflect common practice of protocol design in existing networks.

The definition of a protocol securely realizing an ideal functionality is extended to hybrid protocols in the natural way.

The universal composition operation. We define the universal composition operation and state the universal composition theorem. Let ρ be an $\mathcal{F}$-hybrid protocol, and let Π be a protocol that securely realizes $\mathcal{F}$. The composed protocol ρ^{Π} is constructed by modifying the code of each ITM in ρ so that the first message sent to each copy of $\mathcal{F}$ is replaced with an invocation of a new copy of Π with fresh random input, with the same SID, and with the contents of that message as input. Each subsequent message to that copy of $\mathcal{F}$ is replaced with an activation of the corresponding copy of Π, with the contents of that message given to Π as new input. Each output value generated by a copy of Π is treated as a message received from the corresponding copy of $\mathcal{F}$. The copy of Π will start sending and receiving messages as specified in its code. Notice that if Π is a $\mathcal{G}$-hybrid protocol (i.e., ρ uses ideal evaluation calls to some functionality $\mathcal{G}$) then so is ρ^{Π}.

The universal composition theorem. Let $\mathcal{F}$ be an ideal functionality. In its general form, the composition theorem basically says that if Π is a protocol that UC-realizes $\mathcal{F}$ then, for any $\mathcal{F}$-hybrid protocol ρ, we have that an execution of the composed protocol ρ^{Π} "emulates" an execution of protocol ρ. That is, for any adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that no environment machine $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and protocol ρ^{Π} or with $\mathcal{S}$ and protocol ρ, in a UC interaction. As a corollary, we get that if protocol ρ UC-realizes $\mathcal{F}$, then so does protocol ρ^{Π}. [24]

Theorem A.3 (Universal Composition [Can01].) *Let $\mathcal{F}$ be an ideal functionality. Let ρ be a $\mathcal{F}$-hybrid protocol, and let Π be a protocol that UC-realizes $\mathcal{F}$. Then protocol ρ^{Π} UC-emulates ρ.*

An immediate corollary of this theorem is that if the protocol ρ UC-realizes some functionality $\mathcal{G}$, then so does ρ^{Π}.

A.4 The Common Reference/Random String Model

In the common reference string (CRS) model [CF01, CLOS02], all parties in the system obtain from a trusted party a reference string, which is sampled according to a pre-specified distribution D. The reference string is referred to as the *CRS*. In the UC framework, this is modeled by an ideal functionality $\mathcal{F}_{CRS}^{D}$ that samples a string ρ from a pre-specified distribution D and sets ρ as the CRS. $\mathcal{F}_{CRS}^{D}$ is described in Figure 15.

Functionality $\mathcal{F}_{\mathbf{CRS}}^{\mathbf{D}}$

1. Upon activation with session id *sid* proceed as follows. Sample $\rho = D(r)$, where r denotes uniform random coins, and send $(\texttt{crs}, sid, \rho)$ to the adversary.
2. On receiving $(\texttt{crs}, sid)$ from some party send $(\texttt{crs}, sid, \rho)$ to that party.

Figure 15: The Common Reference String Functionality.

When the distribution D in $\mathcal{F}_{CRS}^{D}$ is sent to be the uniform distribution (on a string of appropriate length) then we obtain the common random string model.

[24] The universal composition theorem in [Can01] applies only to "subroutine respecting protocols", namely protocols that do not share subroutines with any other protocol in the system.

A.5 General Functionality

We consider the general-UC functionality $\mathcal{F}$, which securely evaluates any polynomial-time (possibly randomize) function $f : (\{0,1\}^{\ell_{in}})^n \to (\{0,1\}^{\ell_{out}})^n$. The functionality $\mathcal{F}_f$ is parameterized with a function f and is described in Figure 16. In this paper we will only be concerned with the *static* corruption model.

Functionality $\mathcal{F}_\mathbf{f}$

$\mathcal{F}_f$ parameterized by an (possibly randomized) n-ary function f, running with parties $\mathcal{P} = \{P_1, \ldots P_n\}$ (of which some may be corrupted) and an adversary $\mathcal{S}$, proceeds as follows:

1. Each party P_i (and $\mathcal{S}$ on behalf of P_i if P_i is corrupted) sends $(\mathsf{input}, \mathsf{sid}, \mathcal{P}, P_i, x_i)$ to the functionality.

2. Upon receiving the inputs from all parties, evaluate $(y_1, \ldots y_n) \leftarrow f(x_1, \ldots, x_n)$. For every P_i that is corrupted send adversary $\mathcal{S}$ the message $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, P_i, y_i)$.

3. On receiving $(\mathsf{generateOutput}, \mathsf{sid}, \mathcal{P}, P_i)$ from $\mathcal{S}$ the ideal functionality outputs $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, P_i, y_i)$ to P_i. (And ignores the message if inputs from all parties in $\mathcal{P}$ have not been received.)

Figure 16: General Functionality.

B Garbling Protocols with Extractable Semi-malicious Security

In this section we give a construction of garbling scheme for protocols satisfying extractable semi-malicious security. We first describe a construction making non-black box use of a homomorphic proof commitment with encryption and then we will explain how to make the construction black-box.

The key difference between this construction and the construction in Section 5.2 is that the length of the CRS in this construction grows with the number of parties. Essentially, we have a separate commitment key for each party in the CRS. The rest of the components are almost identical to the construction in Section 5.2.

Construction. We give the description of the construction in Figure 18. The differences between the two constructions are underlined.

B.1 Description of Simulators

We now give the description of simulators (S_1, S_2) that satisfy the extractable semi-malicious security.

S_1**:** On input 1^λ and the set H, S_1 generates the encodings of the honest parties as follows:

Functionality $\mathcal{F}_\mathbf{f}$

$\mathcal{F}_f$ parameterized by an n-ary deterministic single output function f, running with parties $\mathcal{P} = \{P_1, \ldots P_n\}$ (of which some may be corrupted) and an adversary $\mathcal{S}$, proceeds as follows:

1. Each party P_i (and $\mathcal{S}$ on behalf of P_i if P_i is corrupted) sends $(\mathsf{input}, \mathsf{sid}, \mathcal{P}, P_i, x_i)$ to the functionality.
2. Upon receiving the inputs from all parties, evaluate $y \leftarrow f(x_1, \ldots, x_n)$. Send adversary $\mathcal{S}$ the message $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, y)$.
3. On receiving $(\mathsf{generateOutput}, \mathsf{sid}, \mathcal{P}, P_i)$ from $\mathcal{S}$ the ideal functionality outputs $(\mathsf{output}, \mathsf{sid}, \mathcal{P}, y)$ to P_i. (And ignores the message if inputs from all parties in $\mathcal{P}$ have not been received.)

Figure 17: General Functionality for Deterministic Single Output Functionalities.

1. For each $i \in H$, sample $(ck_i, tk_i) \leftarrow K_{\text{hiding}}(1^\lambda)$ and for each $i \notin H$, sample $(ck_i, xk_i) \leftarrow K_{\text{binding}}(1^\lambda)$. Set $\sigma := \{ck_i\}_i$.
2. **for** every $i \in H$ **do**:
   (a) Set the initial state of the party P_i to be $y_i := 0^T \| (0^m, 0^s)$. Choose the randomness $\{\omega_{i,k}\}$ as in the honest execution of Encode function.
   (b) Generate the commitments $c_{i,k} := \text{com}(ck_i, y_{i,k}; \omega_{i,k})$ for each $k \in [\ell]$.
   (c) Set the encoding $\widetilde{x}_i := \{c_{i,k}\}_{k\in[\ell]}$.
3. Set the secret state $\mathsf{st}_S := (\{tk_i\}_{i\in H}, \{xk_i\}_{i\notin H}, \{\omega_{i,k}\}_{i\in H, k\in[\ell]})$
4. Output $(\sigma, \{\widetilde{x}_i\}_{i\in H}, \mathsf{st}_S)$.

S_2**:** On input the secret state st_S, $\{\widetilde{x}_j\}_{j\notin H}$ and the transcript $\Phi(x_1, \ldots, x_n)$ S_2 generates the garbled protocol components of the honest parties as follows:

1. For every $j \notin H$, use the extraction key xk_j to extract the value of the initial state y_j from $\{c_{j,k}\}_{k\in[\ell]}$.
2. For every $j \notin H$, construct the final state y_j^* using the initial state y_j and the transcript $\Phi(x_1, \ldots, x_n)$. Set the final tracking string corresponding to party P_j as $u_j^* := y_j \oplus y_j^*$.
3. For every $j \in H$, set the final tracking string $u_j^* := \begin{cases} 0 & \text{if } k > T \\ \text{uniform in } \{0,1\} & \text{if } k \in [T] \setminus B \\ \text{based on } \Phi(x_1, \ldots, x_n) & \text{if } k \in [T] \cap B \end{cases}$.
4. For every $i \in H$, compute $\left(\widetilde{P}^{i,T}, \widetilde{\mathsf{label}}^{i,T}\right) \leftarrow \mathsf{Sim}(1^\lambda, \Phi(x_1, \ldots, x_n))$.
5. **for** every t from $T-1$ down to 1 **do**:
   (a) For every $i \in H$, parse $\widetilde{\mathsf{label}}^{i,t+1}$ as $\{\mathsf{st}_k^i\}_{k\in[\ell]}, \{\mathsf{en}_k^i\}_{k\in[n\ell_e]}$ and $\{\mathsf{tr}_k^i\}_{k\in[\ell]}$.
   (b) Let $(i^*, f, g) := \Phi_1(t)$.

Let Φ be an n party protocol, $(K_{\text{binding}}, K_{\text{hiding}}, P_{01}, V_{01}, E_{01}, D_{01})$ be a homomorphic proof commitment with encryption, and $(\mathsf{GarbleCkt}, \mathsf{EvalCkt})$ be a garbling scheme for circuits.

$\mathsf{Setup}(1^\lambda)$**:** Sample $\underline{(ck_i, \cdot) \leftarrow K_{\text{binding}}(1^\lambda)}$ for each $i \in [n]$ and output $\sigma := \underline{\{ck_i\}_{i\in[n]}}$ as the reference string.

$\mathsf{Garble}(\sigma, i, \Phi_i, x_i)$**:** To generate the input encoding, garbled protocol component and encoding labels:

1. Compute $(\widetilde{x}_i, y_i, sk_i) \leftarrow \mathsf{Encode}(\sigma, i, x_i)$ where the function Encode is described in Figure 19.
2. Set $\mathsf{label}^{i,T+1} := \big((0,1), \ldots, (0,1)\big)$ where $(0,1)$ is repeated $\ell + n\ell_e + n\ell$ times and $\ell_e := |\widetilde{x}_i|$.
3. **for** each t from T down to 1,
$$\big(\widetilde{\mathsf{P}}^{i,t}, \mathsf{label}^{i,t}\big) \leftarrow \mathsf{GarbleCkt}(1^\lambda, \mathsf{P}_\Phi[i, t, sk_i, \underline{\{ck_i\}}, \mathsf{label}^{i,t+1}])$$
where P_Φ is described in Figure 19.
4. Parse $\mathsf{label}^{i,1}$ as $\{\mathsf{st}^i_{k,0}, \mathsf{st}^i_{k,1}\}_{k\in[\ell]}, \{\mathsf{en}^i_{k,0}, \mathsf{en}^i_{k,1}\}_{k\in[n\ell_e]}, \{\mathsf{tr}^i_{k,0}, \mathsf{tr}^i_{k,1}\}_{k\in[n\ell]}$.
5. Set $\mathsf{st}^i := \{\mathsf{st}^i_{k,y_{i,k}}\}_{k\in[\ell]}$ and $\mathsf{tr}^i := \{\mathsf{tr}^i_{k,0}\}_{k\in[n\ell]}$.
6. Set the garbled protocol component $\widetilde{\Phi}_i := \big(\{\widetilde{\mathsf{P}}^{i,t}\}_{t\in[T]}, \mathsf{st}^i, \mathsf{tr}^i\big)$, the input encoding to $\widetilde{x}_i$ and the encoding labels to be $\{\mathsf{en}^i_{k,0}, \mathsf{en}^i_{k,1}\}_{k\in[n\ell_e]}$.

$\mathsf{Eval}(\{\widetilde{\Phi}_i\}, \{\overline{x}_i\}, \{\mathsf{en}^i_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}\})$**:** To compute the output of the protocol:

1. For every $i \in [n]$, parse $\widetilde{x}_i$ as $\{c_{i,k}, \pi_{i,k}\}_{k\in[\ell]}$. Check if $\underline{V_{01}(ck_i, c_{i,k}, \pi_{i,k}) = 1}$ for every $k \in [\ell]$. Additionally, for every $k \in B$, check if $c_{i,k} := \mathrm{com}(\underline{ck_i}, 0; 0^\lambda)$. If any of the checks fail, output $\perp$.
2. Parse $\widetilde{\Phi}_i$ as $\big(\{\widetilde{\mathsf{P}}^{i,t}\}_{t\in[T]}, \mathsf{st}^i, \mathsf{tr}^i\big)$.
3. Set $\widetilde{\mathsf{label}}^i := \big(\mathsf{st}^i, \mathsf{en}^i_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}, \mathsf{tr}^i\big)$ and the initial tracking strings $u_i := 0^\ell$ for every $i \in [n]$.
4. **for** every round t from 1 to $T-1$ **do**:
   (a) Let $(i^*, f, g) := \Phi_1(t)$.
   (b) Compute $\big(\overline{\mathsf{label}}^{i^*}, \beta, \pi_{i^*,t}\big) \leftarrow \mathsf{EvalCkt}(\widetilde{\mathsf{P}}^{i^*,t}, \widetilde{\mathsf{label}}^{i^*})$ and $\overline{\mathsf{label}}^i \leftarrow \mathsf{EvalCkt}(\widetilde{\mathsf{P}}^{i,t}, \widetilde{\mathsf{label}}^i)$ for every $i \neq i^*$.
   (c) **for** every $i \in [n]$ **do**,
      i. Parse $\overline{\mathsf{label}}^i$ as $(\overline{\mathsf{st}}^i, \overline{\mathsf{en}}^i, \overline{\mathsf{tr}}^i)$.
      ii. Compute d_f, d_g, e_0, e_1 exactly as in P_Φ described in Figure 7 using the tracking string u_{i^*}.
      iii. Parse $\overline{\mathsf{st}}^i$ as $\big(\{\widehat{\mathsf{st}}^i_k\}_{k\neq t}, \mathsf{stct}^i_0, \mathsf{stct}^i_1\big)$ and compute $\widehat{\mathsf{st}}^i_t := D_{01}(\underline{ck_{i^*}}, e_\beta, \mathsf{stct}^i_\beta, \pi_{i^*,t})$. Update $\mathsf{st}^i := \{\widehat{\mathsf{st}}^i_k\}_{k\in[\ell]}$.
      iv. Parse $\overline{\mathsf{tr}}^i$ as $\Big\{\{\widehat{\mathsf{tr}}^i_{(j-1)\ell+k}\}_{k\in[\ell]\setminus\{t\}}, \mathsf{trct}^i_{j,0}, \mathsf{trct}^i_{j,1}\Big\}_{j\in[n]}$. For every $j \in [n]$, compute $\widehat{\mathsf{tr}}^i_{(j-1)\ell+t} := D_{01}(\underline{ck_{i^*}}, e_\beta, \mathsf{trct}^i_{j,\beta}, \pi_{i^*,t})$. Update $\mathsf{tr}^i := \{\widehat{\mathsf{tr}}^i_k\}_{k\in[n\ell]}$.
      v. Update $\widetilde{\mathsf{label}}^i := \big(\mathsf{st}^i, \mathsf{en}^i_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}, \mathsf{tr}^i\big)$.
      vi. Update $u_{i^*,t}$ to β. If $t \in B_{i^*}$, update every $u_{j,t}$ to β for all $j \in [n]$.
   (d) Compute $y := \mathsf{EvalCkt}(\widetilde{\mathsf{P}}^{i,T}, \widetilde{\mathsf{label}}^i)$ and output y.

Figure 18: Garbling Scheme for Protocols

$\mathsf{Encode}(\sigma, i, x_i)$

To generate an encoding of the input x_i do the following:

1. Choose $s_i \leftarrow \{0,1\}^s$ as the random tape of party P_i in the protocol Φ.
2. Let $B := \cup_i B_i$. Choose randomness $\{\omega_{i,k}\}_{k\in[\ell]}$ and the initial state $y_i := r_i \| (x_i, s_i)$ as:
$$r_{i,k} := \begin{cases} 0 & \text{if } k \in [T] \cap B \\ \text{uniform in } \{0,1\} & \text{if } k \in [T] \setminus B \end{cases} \qquad \omega_{i,k} := \begin{cases} 0^\lambda & \text{if } k \in B \\ \text{uniform in } \{0,1\}^\lambda & \text{otherwise} \end{cases}$$
3. For each $k \in [\ell]$, compute $c_{i,k} := \mathrm{com}(\underline{ck_i}, y_{i,k}; \omega_{i,k})$ and $\underline{\pi_{i,k} := P_{01}(ck_i, y_{i,k}, \omega_{i,k})}$
4. Output $\widetilde{x}_i := \{c_{i,k}, \underline{p_{i,k}}\}_{k\in[\ell]}$, the initial state y_i and the secret randomness $sk_i := \{\omega_{i,k}\}_{k\in[\ell]}$.

$\mathsf{P}_\Phi[i, t, sk_i, \{ck_i\}, \mathsf{label}]$

Input. The state y_i of party P_i, the set of encodings $\{\widetilde{x}_j\}$ and the set of tracking strings $\{u_j\}$
Hardcoded. The index i of the party, the round number t, the secret randomness sk_i, the set of commitment keys $\{ck_i\}$ and a set of labels $\mathsf{label} := \left\{\{\mathsf{st}_{k,0}, \mathsf{st}_{k,1}\}_{k\in[\ell]}, \{\mathsf{en}_{k,0}, \mathsf{en}_{k,1}\}_{k\in[n\ell_{enc}]}, \{\mathsf{tr}_{k,0}, \mathsf{tr}_{k,1}\}_{k\in[n\ell]}\right\}$.

1. Let $(i^*, f, g) := \Phi_i(t)$.
2. Parse $\widetilde{x}_{i^*}$ as $\{c_{i^*,k}\}_{k\in[\ell]}$.
3. Let d_f and d_g be the commitments to the bits $y_{i^*,f}$ and $y_{i^*,g}$ where y_{i^*} is the current state of the active party. These commitments are computed as follows: for $h \in \{f, g\}$, $d_h := c_{i^*,h}$ if $u_{i^*,h} = 0$; else, $d_h := \frac{\mathrm{com}(\underline{ck_{i^*}}, 1; 0^\lambda)}{c_{i^*,h}}$.
4. Compute $e_0 := d_f d_g c_{i^*,t}^2 \mathrm{com}(\underline{ck_{i^*}}, -2; 0^\lambda)$ and $e_1 := d_f d_g \left(\frac{\mathrm{com}(\underline{ck_{i^*}}, 1; 0^\lambda)}{c_{i^*,t}}\right)^2 \mathrm{com}(\underline{ck_{i^*}}, -2; 0^\lambda)$.
Set $\alpha := \mathsf{NAND}(y_{i,f}, y_{i,g})$.
5. For $b \in \{0,1\}$, compute $\mathsf{stct}_b := \begin{cases} E_{01}(\underline{ck_{i^*}}, e_b, \mathsf{st}_{t,b}) & \text{if } t \in B_{i^*} \\ E_{01}(\underline{ck_{i^*}}, e_b, \mathsf{st}_{t,y_{i,t}}) & \text{if } t \notin B_{i^*} \wedge i \neq i^* \\ E_{01}(\underline{ck_{i^*}}, e_b, \mathsf{st}_{t,\alpha}) & \text{if } t \notin B_{i^*} \wedge i = i^* \end{cases}$.
Set $\overline{\mathsf{st}} := \{\mathsf{st}_{k,y_{i,k}}\}_{k\neq t}, \mathsf{stct}_0, \mathsf{stct}_1$.
6. Set $\overline{\mathsf{en}} := \mathsf{en}_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}$.
7. For $b \in \{0,1\}$, $j \in [n]$, compute $\mathsf{trct}_{j,b} := \begin{cases} E_{01}(\underline{ck_{i^*}}, e_b, \mathsf{tr}_{(j-1)\ell+t,b}) & \text{if } (t \in B_{i^*}) \vee (j = i^*) \\ E_{01}(\underline{ck_{i^*}}, e_b, \mathsf{tr}_{(j-1)\ell+t,u_{j,t}}) & \text{if } (t \notin B_{i^*}) \wedge (j \neq i^*) \end{cases}$.
Set $\overline{\mathsf{tr}} := \left\{\{\mathsf{tr}_{(j-1)\ell+k,u_{j,k}}\}_{k\in[\ell]\setminus\{t\}}, \mathsf{trct}_{j,0}, \mathsf{trct}_{j,1}\right\}_{j\in[n]}$.
8. If $i = i^*$ then parse sk_i as $\{\omega_{i,k}\}_{k\in[\ell]}$. For $h \in \{f, g\}$, set $\omega'_{i,h} := \begin{cases} \omega_{i,h} & \text{if } u_{i,h} = 0 \\ -\omega_{i,h} & \text{otherwise} \end{cases}$
Compute $\pi_{i,t} := P_{01}(\underline{ck_i}, e_\beta, \rho_\beta)$ where $\beta := y_{i,t} \oplus \alpha, \rho_0 = \omega'_{i,f} + \omega'_{i,g} + 2\omega_{i,t}, \rho_1 = \omega'_{i,f} + \omega'_{i,g} - 2\omega_{i,t}$.
9. If $t \neq T$ then output $\overline{\mathsf{label}} := (\overline{\mathsf{st}}, \overline{\mathsf{en}}, \overline{\mathsf{tr}})$ and additionally output $(\beta, \pi_{i,t})$ if $i = i^*$.
If $t = T$ then output the transcript of the protocol from the state as $\{y_{i,k}\}_{k\in B}$.

Figure 19: The programs Encode and P_Φ.

(c) Compute d_f, d_g, e_0, e_1 as given in program P_Φ using the final tracking string u_{i^*}.

(d) Let $\beta := u_{i^*,t}$.

(e) For every $i \in H$, generate $\mathsf{stct}^i_\beta := E_{01}(ck_{i^*}, e_\beta, \mathsf{st}^i_t)$ and $\mathsf{stct}^i_{1-\beta} := E_{01}(ck_{i^*}, e_{1-\beta}, 0^\lambda)$.
For every $j \in [n]$, generate $\mathsf{trct}^i_{j,\beta} := E_{01}(ck_{i^*}, e_\beta, \mathsf{tr}^i_{(j-1)\ell+t})$ and $\mathsf{trct}^i_{j,1-\beta} := E_{01}(ck_{i^*}, e_{1-\beta}, 0^\lambda)$.

(f) Set $\overline{\mathsf{st}}^i$ as $\left(\{\mathsf{st}^i_k\}_{k\neq t}, \mathsf{stct}^i_0, \mathsf{stct}^i_1\right)$, $\overline{\mathsf{en}}^i$ as $\{\mathsf{en}^i_k\}_{k\in[n\ell_e]}$ and $\overline{\mathsf{tr}}^i$ as $\left\{\{\mathsf{tr}^i_{(j-1)\ell+k}\}_{k\in[\ell]\setminus\{t\}}, \mathsf{trct}^i_{j,0}, \mathsf{trct}^i_{j,1}\right\}_{j\in[n]}$.

(g) For every $i \in H \setminus \{i^*\}$ generate, $\widetilde{P}^{i,t}, \widetilde{\mathsf{label}}^{i,t} \leftarrow \mathsf{Sim}(1^\lambda, (\overline{\mathsf{st}}^i, \overline{\mathsf{en}}^i, \overline{\mathsf{tr}}^i))$.

(h) **if** $i^* \in H$ then:

   i. For $h \in \{f, g, t\}$, set $\omega'_{i^*,h} := \begin{cases} \omega_{i^*,h} & \text{if } u_{i^*,h} = 0 \\ -\omega_{i^*,h} & \text{otherwise} \end{cases}$.

   ii. Compute $v_1 := \text{Topen}(tk_{i^*}, u_{i^*,f}, \omega'_{i^*,f}, 0)$, $v_2 := \text{Topen}(tk_{i^*}, u_{i^*,g}, \omega'_{i^*,g}, 0)$ and $v_3 := \text{Topen}(tk_{i^*}, u_{i^*,t}, \omega'_{i^*,t}, 1)$.

   iii. Compute $\pi_{i^*,t} := P_{01}(ck_{i^*}, 1, \rho)$ where $\rho = v_1 + v_2 - 2v_3$.

   iv. Generate, $\widetilde{P}^{i^*,t}, \widetilde{\mathsf{label}}^{i^*,t} \leftarrow \mathsf{Sim}(1^\lambda, (\overline{\mathsf{st}}^{i^*}, \overline{\mathsf{en}}^{i^*}, \overline{\mathsf{tr}}^{i^*}), (\beta, \pi_{i^*,t}))$

We show that

Lemma B.1 *Assuming the security of the garbling scheme for circuits and the homomorphic proof commitment with encryption for every protocol Φ, and every subset $H \subseteq [n]$ of honest parties, and for every choice of inputs $\{x_i\}_{i\in H}$ for honest parties, we have that for every PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,*

$$\left| \Pr\left[\mathsf{extReal}[1^\lambda, \{x_i\}, H] = 1\right] - \Pr\left[\mathsf{extIdeal}[1^\lambda, \{x_i\}, H] = 1\right] \right| \leq \mathsf{negl}(\lambda)$$

B.2 Proof of Lemma B.1

The proof of this lemma proceeds in a similar way as the proof of Lemma 5.8. As before, in hybrid Hybrid_k, we define $\mathsf{Adv}(\mathsf{Hybrid}_k)$ to be the probability that $\mathcal{A}$ outputs 1 when given inputs as distributed in Hybrid_k.

For every $w \in [T]$, we define Hybrid_w as follows:

$\underline{\mathsf{Hybrid}_w}$: In this hybrid, we change how $\left(\widetilde{P}^{i,t}, \widetilde{\mathsf{label}}^{i,t}\right)$ for every $i \in H$ and $t < w$ are generated. In particular, we generate them as given in the modified garbling procedure given in Figure 8.

Notice that Hybrid_1 is distributed as in the real world.

Lemma B.2 *Assuming the security of garbling scheme for circuits and the homomorphic proof commitments with encryption, we have that for every $w \in [T]$, $|\mathsf{Adv}(\mathsf{Hybrid}_w) - \mathsf{Adv}(\mathsf{Hybrid}_{w+1})| \leq \mathsf{negl}(\lambda)$.*

Proof We define a couple of intermediate hybrids.
$\underline{\mathsf{Hybrid}_{w,1}}$: Let $y^w_i := \{y^*_{i,k}\}_{k\in[w-1]} \| \{y_{i,k}\}_{k\in[w,\ell]}$ be the local state of party P_i at the beginning of the w-th round. We use the extraction key xk_j to extract the value of the initial state $\{y_j\}_{j\notin H}$ from $\{\widetilde{x}_j\}_{j\notin H}$. Later using the transcript $\Phi(x_1, \ldots, x_n)$, we can construct y^w_j for every $j \notin H$. For every

Garble''**:** On additional inputs the hybrid number w, the final state y_i^* of party P_i, the set of encodings $\{\widetilde{x}_j\}$ and the final set of tracking strings $\{u_j^*\}$ for every P_j do:

1. Compute $(\widetilde{x}_i, y_i, sk_i) \leftarrow \mathsf{Encode}(\sigma, i, x_i)$ where the function Encode is described in Figure 19.
2. Set $\mathsf{label}^{i,T+1} := \big((0,1),\ldots,(0,1)\big)$ where $(0,1)$ is repeated $\ell + n\ell_e + n\ell$ times and $\ell_e := |\widetilde{x}_i|$.
3. **for** each t from T down to $\underline{w}$,
$$\big(\widetilde{\mathsf{P}}^{i,t}, \mathsf{label}^{i,t}\big) \leftarrow \mathsf{GarbleCkt}(1^\lambda, \mathsf{P}_\Phi[i,t,sk_i,\{ck_i\},\mathsf{label}^{i,t+1}])$$
where P_Φ is described in Figure 19.
4. Parse $\mathsf{label}^{i,w}$ as $\{\mathsf{st}^i_{k,0}, \mathsf{st}^i_{k,1}\}_{k\in[\ell]}, \{\mathsf{en}^i_{k,0}, \mathsf{en}^i_{k,1}\}_{k\in[n\ell_e]}, \{\mathsf{tr}^i_{k,0}, \mathsf{tr}^i_{k,1}\}_{k\in[n\ell]}$.
5. Set $\widetilde{\mathsf{label}}^{i,w} := \{\mathsf{st}^i_{k,y^*_{i,k}}\}_{k\in[w]}, \{\mathsf{st}^i_{k,y_{i,k}}\}_{k\in[w+1,\ell]}, \mathsf{en}_{\widetilde{x}_1\ldots\widetilde{x}_n}, \{\mathsf{tr}^i_{(j-1)\ell+k,u^*_{j,k}}\}_{j\in[n],k\in[w]}, \{\mathsf{tr}^i_{(j-1)\ell+k,0}\}_{j\in[n],k\in[w+1,\ell]}$.
6. **for** each t from $w-1$ down to 1:
   (a) Parse $\widetilde{\mathsf{label}}^{i,t+1}$ as $\{\mathsf{st}^i_k\}_{k\in[\ell]}, \{\mathsf{en}^i_k\}_{k\in[n\ell_e]}$ and $\{\mathsf{tr}^i_k\}_{k\in[\ell]}$.
   (b) Let $(i^*, f, g) := \Phi_1(t)$.
   (c) Compute d_f, d_g, e_0, e_1 as given in program P_Φ using the tracking string $u^*_{i^*}$.
   (d) Let $\beta := u^*_{i^*,t}$.
   (e) Generate $\mathsf{stct}^i_\beta := E_{01}(ck_{i^*}, e_\beta, \mathsf{st}^i_t)$ and $\mathsf{stct}^i_{1-\beta} := E_{01}(ck_{i^*}, e_{1-\beta}, \underline{0^\lambda})$. Generate $\mathsf{trct}^i_{j,\beta} := E_{01}(ck_{i^*}, e_\beta, \mathsf{tr}^i_{(j-1)\ell+t})$ and $\mathsf{trct}^i_{j,1-\beta} := E_{01}(ck_{i^*}, e_{1-\beta}, \underline{0^\lambda})$ for every $j \in [n]$.
   (f) Set $\overline{\mathsf{st}}^i$ as $\big(\{\mathsf{st}^i_k\}_{k\neq t}, \mathsf{stct}^i_0, \mathsf{stct}^i_1\big)$, $\overline{\mathsf{en}}^i$ as $\{\mathsf{en}^i_k\}_{k\in[n\ell_e]}$ and $\overline{\mathsf{tr}}^i$ as $\big\{\{\mathsf{tr}^i_{(j-1)\ell+k}\}_{k\in[\ell]\setminus\{t\}}, \mathsf{trct}^i_{j,0}, \mathsf{trct}^i_{j,1}\big\}_{j\in[n]}$.
   (g) Generate, $\widetilde{P}^{i,t}, \widetilde{\mathsf{label}}^{i,t} \leftarrow \mathsf{Sim}(1^\lambda, (\overline{\mathsf{st}}^i, \overline{\mathsf{en}}^i, \overline{\mathsf{tr}}^i))$.

Figure 20: Modified Garbling Procedure

$i \in H$, y_i can be constructed from $\Phi(x_1,\ldots,x_n)$. Let $\{u_j^w\}$ be the set of tracking strings at the beginning of the w-th round. In this hybrid, for every for every $i \in H$ we generate

$$\big(\widetilde{P}^{i,w}, \widetilde{\mathsf{label}}^{i,w}\big) \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{P}_\Phi[i,w,ck,sk_i,\mathsf{label}^{i,w+1}](y_i^w, \{\widetilde{x}_j\}, \{u_j^w\}))$$

Claim B.3 *Assuming the security of garbling scheme for circuits,* $|\mathsf{Adv}(\mathsf{Hybrid}_w) - \mathsf{Adv}(\mathsf{Hybrid}_{w,1})| \leq \mathsf{negl}(\lambda)$

Proof The proof of this claim follows via an identical argument in the proof of Claim 5.10. ■

$\underline{\mathsf{Hybrid}_{w,2}}$: In this hybrid, we perform the modified garbling procedure with input $w+1$ instead of w. This hybrid is identically distributed to Hybrid_{w+1}. Additionally, instead of using the extraction key xk_j to extract out the value of the initial state y_j for every $j \notin H$ we brute force search for a y_j using $\{c_{j,k}\}_{k\in[\ell]}$. Note that this search procedure might take super-polynomial time.

Claim B.4 *Assuming the statistical semantic security of homomorphic proof commitments with encryption,* $|\mathsf{Adv}(\mathsf{Hybrid}_{w,1}) - \mathsf{Adv}(\mathsf{Hybrid}_{w,2})| \leq \mathsf{negl}(\lambda)$

Proof Assume for the sake of contradiction that $|\mathsf{Adv}(\mathsf{Hybrid}_{w,1}) - \mathsf{Adv}(\mathsf{Hybrid}_{w,2})| > \frac{1}{\mathsf{poly}(\lambda)}$. We construct an adversary $\mathcal{B}$ breaking the semantic security of homomorphic proof commitments with encryption.

$\mathcal{B}$ obtains $\{ck_i\}_{i\in[n]}$ from the external challenger and computes the input encodings $\{\widetilde{x}_i\}_{i\in H}$ as per the honest procedure given in Figure 7. It then runs $\mathcal{A}_1(\sigma, \{\widetilde{x}_i\}_{i\in H})$ to obtain $\{\widetilde{x}_j\}_{j\notin H}$ and $\mathsf{st}_{\mathcal{A}}$. $\mathcal{B}$ does a brute force search on $\{\widetilde{x}_j\}_{j\notin H}$ to obtain the initial states $\{y_j\}_{j\notin H}$. $\mathcal{B}$ generates the garbled circuits $\widetilde{\mathsf{P}}^{i,t}$ for t from T to $w+1$ as in the modified garbled procedure in Figure 20. Instead of generating the garbled circuit $\left(\widetilde{P}^{i,w}, \widetilde{\mathsf{label}}^{i,w}\right) \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{P}_\Phi[i, w, \{ck_i\}, sk_i, \mathsf{label}^{i,w+1}](y_i^w, \{\widetilde{x}_j\}, \{u_j^w\}))$ for every $i \in H$ as in $\mathsf{Hybrid}_{w,1}$, it generates it as follows:

1. Let y_i^{w+1} be the local state of party i and $\{u_j^{w+1}\}$ be the set of tracking strings at the end of the w-th round.
2. Let $(i^*, f, g) := \Phi_1(w)$.
3. Compute d_f, d_g, e_0, e_1 as given in program P_Φ using the tracking string $u_{i^*}^{w+1}$.
4. Let $\beta := u_{i^*,w}^{w+1}$, $\alpha := y_{i,w}^{w+1}$ and $\gamma_j := u_{j,w}^{w+1}$ for every $j \in [n]$.
5. Generate $\mathsf{stct}^i_\beta := E_{01}(ck_{i^*}, e_\beta, \mathsf{st}^i_{w,\alpha})$ if $w \notin B_{i^*} \vee i = i^*$; else generate $\mathsf{stct}^i_\beta := E_{01}(ck_{i^*}, e_\beta, \mathsf{st}^i_{w,\beta})$. Interact with the semantic security challenger by giving $e_{1-\beta}$ as the challenge commitment, $\mathsf{st}^i_{w,1-\alpha}$ [25] and 0^λ as the challenge messages. Receive the challenge ciphertext $\mathsf{stct}^i_{1-\beta}$.
6. Generate $\mathsf{trct}^i_{j,\beta} := E_{01}(ck_{i^*}, e_\beta, \mathsf{tr}^i_{(j-1)\ell+w,\beta})$ if $w \in B_{i^*} \vee j = i^*$; else generate $\mathsf{trct}^i_{j,\beta} := E_{01}(ck_{i^*}, e_\beta, \mathsf{tr}^i_{(j-1)\ell+w,\gamma_j})$. Interact with the semantic security challenger by giving $e_{1-\beta}$ as the challenge commitment and $\mathsf{tr}^i_{(j-1)\ell+w,1-\beta}$ [26] and 0^λ as the challenge messages for every $j \in [n]$. Receive the set of challenge ciphertexts $\{\mathsf{trct}^i_{j,1-\beta}\}$ for every $j \in [n]$.
7. Set $\overline{\mathsf{st}}^i$ as $\left(\{\mathsf{st}^i_{k,y^{w+1}_{i,k}}\}_{k\neq t}, \mathsf{stct}^i_0, \mathsf{stct}^i_1\right)$, $\overline{\mathsf{en}}^i$ as $\{\mathsf{en}^i_k\}_{k\in[n\ell_e]}$ and $\overline{\mathsf{tr}}^i$ as $\big\{\{\mathsf{tr}^i_{(j-1)\ell+k,u^{w+1}_{j,k}}\}_{k\in[\ell]\setminus\{t\}}, \mathsf{trct}^i_{j,0}, \mathsf{trct}^i_{j,1}\big\}_{j\in[n]}$.
8. Generate, $\widetilde{P}^{i,w}, \widetilde{\mathsf{label}}^{i,t} \leftarrow \mathsf{Sim}(1^\lambda, (\overline{\mathsf{st}}^i, \overline{\mathsf{en}}^i, \overline{\mathsf{tr}}^i))$.

Finally, $\mathcal{B}$ runs $\mathcal{A}_2$ on the inputs $\mathsf{st}_{\mathcal{A}}, \{\widetilde{\Phi}_i, \widetilde{x}_i, \mathsf{lab}^i_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}\}_{i\in H}$ and outputs whatever $\mathcal{A}$ outputs. Notice that the challenge commitment $e_{1-\beta}$ is not a commitment to zero-one message. Thus, $\mathcal{B}$ represents a valid challenger to the semantic security. If the challenge ciphertexts contain an encryption of the string 0^λ then the view of $\mathcal{A}_2$ is distributed identically to $\mathsf{Hybrid}_{w,2}$. Else, it is distributed identically to $\mathsf{Hybrid}_{w,1}$. Thus, $\mathcal{B}$ breaks the semantic security of homomorphic proof commitment with encryption. ■

This completes the proof of the lemma. ■

$\underline{\mathsf{Hybrid}_{T+1}}$: In this hybrid, we change how the reference string is generated. Instead of generating $(ck_i, \cdot) \leftarrow K_{\text{binding}}(1^\lambda)$, we generate it as $(ck_i, \cdot) \leftarrow K_{\text{hiding}}(1^\lambda)$ for every $i \in H$. We still generate

[25] We need to give $\mathsf{st}^i_{w,\alpha}$ instead of $\mathsf{st}^i_{w,1-\alpha}$ if $w \notin B_{i^*}$

[26] If $w \notin B_{i^*} \wedge j = i^*$, we should then give $\mathsf{tr}^i_{(j-1)\ell+w,\gamma_j}$ instead of $\mathsf{tr}^i_{(j-1)\ell+w,1-\beta}$.

$(ck_j, xk_j) \leftarrow K_{\text{binding}}(1^\lambda)$ for every $j \in H$ and use the extraction key xk_j to extract the value of the initial state y_j.

The following lemma directly follows from the key indistinguishability of the homomorphic proof commitment.

Lemma B.5 *Assuming key indisintinguishability property of homomorphic proof commitment with encryption,* $|\mathsf{Adv}(\mathsf{Hybrid}_T) - \mathsf{Adv}(\mathsf{Hybrid}_{T+1})| \leq \mathsf{negl}(\lambda)$

$\underline{\mathsf{Hybrid}_{T+2}}$: For every $i \in H$, let y_i^* be the final local state, y_i be the initial local state and u_i^* be the final value of the tracking string corresponding to i. Notice that u_i^* is distributed uniformly on projection to the coordinates $[T] \setminus \{B\}$ and $y_i^* := y_i \oplus u_i^*$. Let $B_H := \cup_{i \in H} B_i$. In this hybrid, we change how $\pi_{i^*,t}$ (which is hardcoded while generating the simulated garbled circuit) is generated for every $t \in B_H$. In particular, we generate it as:

1. Let $(i^*, f, g) := \Phi_i(t)$.
2. For $h \in \{f, g, t\}$, set $\omega'_{i^*,h} := \begin{cases} \omega_{i^*,h} & \text{if } u^*_{i^*,h} = 0 \\ -\omega_{i^*,h} & \text{otherwise} \end{cases}$.
3. Compute v_1 such that $\text{com}(ck_{i^*}, y^*_{i^*,f}; \omega'_{i^*,f}) := \text{com}(0; v_1)$, v_2 such that $\text{com}(ck_{i^*}, y^*_{i^*,g}; \omega'_{i^*,g}) := \text{com}(0; v_2)$ and v_3 such that $\text{com}(ck_{i^*}, y^*_{i^*,t}; \omega'_{i^*,t}) := \text{com}(1; v_3)$. Notice that this step might take super-polynomial time.
4. Compute $\pi_{i^*,t} := P_{01}(ck_{i^*}, 1, \rho)$ where $\rho = v_1 + v_2 - 2v_3$.

The following lemma follows via an identical argument to Lemma 5.13.

Lemma B.6 *Assuming perfect witness indistinguishability of homomorphic proof commitment with encryption we have,* $|\mathsf{Adv}(\mathsf{Hybrid}_{T+1}) - \mathsf{Adv}(\mathsf{Hybrid}_{T+2})| = 0$.

$\underline{\mathsf{Hybrid}_{T+3}}$: In this hybrid, for every $i \in H$, we change how the encoding $\widetilde{x}_i$ is generated. In particular, for every $k \in [\ell] \setminus B$, we generate $c_{i,k} := \text{com}(ck_i, 0)$. As in the previous hybrid, we additionally find v_1, v_2, v_3 by running in possibly super-polynomial time.

Lemma follows from Lemma 5.14

Lemma B.7 *Assuming perfect hiding of the homomorphic proof commitment with encryption we have,* $|\mathsf{Adv}(\mathsf{Hybrid}_{T+2}) - \mathsf{Adv}(\mathsf{Hybrid}_{T+3})| = 0$.

$\underline{\mathsf{Hybrid}_{T+4}}$: In this hybrid, we make the process of sampling from the distribution efficient by giving access to the trapdoor key tk. Notice that the distributions Hybrid_{T+3} and Hybrid_{T+4} are identical from the perfect trapdoor opening property of the homomorphic proof commitment with encryption. Hybrid_{T+4} is distributed identically to $\mathsf{extIdeal}$.

B.3 Black-Box Construction

In this subsection we explain how to transform the non-black box construction given in Figure 18 to a fully black-box construction. Notice that the only non black-box use the underlying primitive happens in the program P_Φ described in Figure 19 that uses the circuit for encryption and proof generation. We now describe a way to make the program P_Φ not use the circuit for computing encryption and proof generation by pre-computing the appropriate values and hardwiring them in the program.

Since the output of Φ_i on an input is fixed and does not depend on the messages sent in the protocol, we can pre-compute the value of this output as (i^*, f, g) for every round t. For each value of the bit $u_{i^*,h}$ where $h \in \{f, g\}$, we can pre-compute $d_{h,0}$ if $u_{i^*,h} = 0$ and $d_{h,1}$ if $u_{i^*,h} = 1$. We can then hardwire these pre-computed commitments in the program P_Φ and use them as and when needed within the program. For each of the four possible choices of $d_{f,a}$ and $d_{g,b}$ where $a, b, \in \{0, 1\}$, we pre-compute the values $e_{a,b,0}$ and $e_{a,b,1}$ and hardwire them in the program P_Φ. Similarly, for every choice of $a, b, c \in \{0, 1\}$, we can pre-compute the encryptions $\mathsf{stct}_{a,b,c,d}$ by encrypting $\mathsf{st}_{t,d}$ for $d \in \{0, 1\}$. Similarly, we can pre-compute $\mathsf{trct}_{j,a,b,c,d}$ encrypting $\mathsf{tr}_{(j-1)\ell+t,d}$ for $d \in \{0, 1\}$ and hardwire them in the program P_Φ. We can also pre-compute the proof $\pi_{i^*,t,a,b,c}$ for every value of $a, b, c \in \{0, 1\}$. Now the role of the program P_Φ is just to identify the appropriate commitments, the ciphertexts and the proofs based on the state y_i and the tracking strings $\{u_j\}$. Notice that the number of hardwired components in the new program P_Φ is polynomial in the number of parties. The new program P_Φ makes black-box use of the underlying homomorphic proof commitment with encryption.

C Description of the Simulator and Hybrid argument for Black-box MPC

In this section we give the description of the simulator for the protocol described in Figure 14 and prove indistinguishability of the simulated distribution from the real world execution.

C.1 Description of the Simulator

In this subsection we give the description of the ideal world adversary $\mathcal{S}$ having access to the ideal functionality $\mathcal{F}_f$ that simulates the view of the real world adversary $\mathcal{A}$. $\mathcal{S}$ will internally use the simulators S_Φ for the semi-malicious security of Φ and (S_1, S_2) for the garbling scheme for protocols.

We assume that $\mathcal{A}$ is static and hence the set of honest parties H is known before the execution of the protocol.

Simulating the CRS. To simulate the common reference string, $\mathcal{S}$ runs S_1 on input 1^λ and H and obtains σ, the set of input encodings $\{\widetilde{x}_i\}_{i\in H}$ for the honest parties and the secret simulation state st_S. It set the common reference string to be σ and locally stores $\{\widetilde{x}_i\}_{i\in H}$ and st_S. Note that st_S consists of a set of extraction keys xk_j for every $j \notin H$.

Simulating the interaction with $\mathcal{Z}$. For every input value for the set of corrupted parties that $\mathcal{S}$ receives from $\mathcal{Z}$, $\mathcal{S}$ writes that value to $\mathcal{A}$'s input tape. Similarly, the output of $\mathcal{A}$ is written as the output on $\mathcal{S}$'s output tape.

Simulating the interaction with $\mathcal{A}$: For every concurrent interaction with the session identifier sid that $\mathcal{A}$ may start, the simulator does the following:

- **Round-1 messages from $\mathcal{S}$ to $\mathcal{A}$:** $\mathcal{S}$ recovers $\{\widetilde{x}_i\}_{i\in H}$ from its local storage. For each $i \in H$, $\mathcal{S}$ sends $\widetilde{x}_i$ to $\mathcal{A}$ on behalf of the honest party i.

- **Round-1 messages from $\mathcal{A}$ to $\mathcal{S}$:** $\mathcal{S}$ receives $\{\widetilde{x}_j\}_{j\notin H}$ on behalf of each honest party $i \in H$. $\mathcal{S}$ also simulates the ideal functionality $\mathcal{F}_{\text{NIOT}}$ to $\mathcal{A}$ and stores all the queries that $\mathcal{A}$ makes to $\mathcal{F}_{\text{NIOT}}$ and answers them according to $\mathcal{F}$.

- **Round-2 messages from $\mathcal{S}$ to $\mathcal{A}$:**

  1. **for** every $i \in H$,
     (a) For each $j \notin H$, $\mathcal{S}$ parses $\widetilde{x}_j$ received by i as $\{c_{j,k}, \pi_{j,k}\}_{j\notin H}$.
     (b) For every $j \notin H$ and every $k \in [\ell]$, $\mathcal{S}$ checks if the proof $\pi_{j,k}$ is a valid zero-one proof that $c_{j,k}$ is a commitment to a message in $\{0, 1\}$. If any of the checks fail, $\mathcal{S}$ locally stores $(i, \textbf{abort})$.
  2. If for every $i \in H$, $(i, \textbf{abort})$ is stored, $\mathcal{S}$ aborts the execution.
  3. Else, for every $j \notin H$, $\mathcal{S}$ recovers the initial state y_j from the commitments $\{c_{j,k}\}$ using the extraction key xk_j. Note that the initial state y_j consists of the input x_j and the randomness s_j of party j in the computation of Φ.
  4. $\mathcal{S}$ queries the ideal functionality $\mathcal{F}_f$ with the query $(\textbf{input}, sid, j, x_j)$ for every $j \notin H$ and obtains the string z which is the output of the functionality.
  5. $\mathcal{S}$ then runs the simulator S_Φ on inputs $\{y_j\}_{j\notin H}$, the output z and the extracted OT correlations queried by $\mathcal{A}$ to obtain the simulated transcript τ.
  6. It then runs S_2 on input the secret state st_S (recovered from its local storage), $\{\widetilde{x}_j\}_{j\notin H}$ and the simulated transcript τ to obtain $\{\widetilde{\Phi}_i, \mathsf{lab}_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}\}_{i\in H}$.
  7. For every i such that there does not exist $(i, \textbf{abort})$ in its local storage, $\mathcal{S}$ forwards $\widetilde{\Phi}_i, \mathsf{lab}_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}$ on behalf of i.

- **Round-2 messages from $\mathcal{A}$ to $\mathcal{S}$:** For every $i \in H$, $\mathcal{S}$ obtains the second round message from $\mathcal{A}$ on behalf of the honest party. For every $i \in H$, if the set of message obtained from $\mathcal{A}$ is well formed, $\mathcal{S}$ sends $(\textbf{generateOutput}, sid, i)$ to the trusted party $\mathcal{F}_f$.

We show that:

Lemma C.1 *Assuming the semi-malicious security of Φ and the extractable semi-malicious security of garbling scheme for protocols for any environment $\mathcal{Z}$ that obeys the rules of interaction for UC security we have* $\text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.

C.2 Proof of Lemma C.1

We prove the lemma via a hybrid argument.

$\underline{\mathsf{Hybrid}_1}$: This corresponds to the real world execution where the environment $\mathcal{Z}$ is interacting with the real world adversary $\mathcal{A}$. Alternatively, we can view this hybrid in the ideal world where the ideal world adversary $\mathcal{S}$ additionally has access to the private inputs of the honest parties and interacts with $\mathcal{A}$. $\mathcal{S}$ generates the messages of the honest parties as given in the description of the protocol. It also simulates the ideal world functionality $\mathcal{F}_{\text{NIOT}}$ to $\mathcal{A}$.

$\underline{\mathsf{Hybrid}_2}$: In this hybrid the ideal world adversary $\mathcal{S}$ invokes the simulator of the garbling scheme for protocols instead of generating the round-1 and round-2 messages honestly. To give more details, $\mathcal{S}$ runs S_1 on input 1^λ and H, to obtain the common reference string σ, the set of input encodings $\{\widetilde{x}_i\}_{i\in H}$ corresponding to the honest parties and the secret simulation state st_S. It sets the common reference string as σ and forwards $\{\widetilde{x}_i\}_{i\in H}$ to $\mathcal{A}$ on behalf of the honest parties. It obtains $\{\widetilde{x}_j\}_{j\notin H}$ and recovers the initial input state y_j and the input x_j for each $j \in H$ using the extraction key xk_j in st_S. It executes Φ in "its head" using its knowledge of the honest parties inputs and using the extracted inputs of the corrupted parties to obtain the transcript $\Phi(x_1, \ldots, x_n)$. It then runs the simulator S_2 on input the secret state st_S, $\{\widetilde{x}_j\}_{j\notin H}$ and the transcript $\Phi(x_1, \ldots, x_n)$ to obtain $\{\widetilde{\Phi}_i, \mathsf{lab}_{\widetilde{x}_1\|\ldots\|\widetilde{x}_n}\}_{i\in H}$. It then forwards this to $\mathcal{A}$.

Notice that the view of the adversary in Hybrid_1 and Hybrid_2 is computationally indistinguishable from the extractable semi-malicious security of garbling scheme for protocols.

$\underline{\mathsf{Hybrid}_3}$: In this hybrid, the ideal world adversary uses the simulator for Φ to generate the transcript of the protocol. The ideal world adversary $\mathcal{S}$ recovers the the initial input state y_j and the input x_j for each $j \in H$ as in the previous hybrid. It queries the ideal world functionality $\mathcal{F}_f$ on input $(\mathbf{input}, sid, j, x_j)$ for each $j \in [n]$ and obtains the output z. It then runs the simulator S_Φ on inputs $\{x_j\}_{j\notin H}$, the output z and the extracted OT correlations to obtain the simulated transcript τ. It then uses τ as input while running S_2. The rest of the simulation is exactly as in the previous hybrid.

Notice that the view of the adversary in Hybrid_2 and Hybrid_3 is computationally indistinguishable from the semi-malicious security of Φ. Hybrid_3 is distributed identically to $\mathrm{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.