Silent Two-party Computation Assisted by Semi-trusted Hardware

Yibiao Lu
Zhejiang University
3160103186@zju.edu.cn

Bingsheng Zhang
Zhejiang University
bingsheng@zju.edu.cn

Weiran Liu
Alibaba Group
weiran.lwr@alibaba-inc.com

Lei Zhang
Alibaba Group
zongchao.zl@taobao.com

Kui Ren
Zhejiang University
kuiren@zju.edu.cn

***Abstract*—With the advancement of the *trusted execution environment* (TEE) technologies, hardware-supported secure computing becomes increasingly popular due to its efficiency. During the protocol execution, typically, the players need to contact a third-party server for remote attestation, ensuring the validity of the involved trusted hardware component, such as Intel SGX, as well as the integrity of the computation result. When the hardware manufacturer is not fully trusted, sensitive information may be leaked to the third-party server through backdoors, side-channels, steganography, and kleptography, etc. In this work, we introduce a new security notion called *semi-trusted hardware model*, where the adversary is allowed to passively and/or maliciously corrupt the hardware component. Therefore, she can learn the input of the hardware component and might also tamper the output. We show that two-party computation can still be significantly sped up in this new model. When the semi-trusted hardware is instantiated by Intel SGX, to generate 10k random OT's, our protocol is 50X and 270X faster than the EMP-ROT in the LAN and WAN setting, respectively. For the AES, SHA-1, and SHA-256 evaluation, our protocol is 4-5X and 40-50X faster than the EMP-SH2PC in the LAN and WAN setting, respectively.**

1. Introduction

In secure multi-party computation (MPC), two or more players want to collectively compute a function and receive its output without revealing their inputs to the other players. In the past decades, MPC has gradually transitioned from theory to practice, and it has been widely used in many security critical real-world applications, such as private set intersection and secure auction. In spite of its success, MPC is still not efficient for complicated real-time tasks due to its computational overhead and high communication cost. Meanwhile, recent development of trusted execution environment (TEE) technologies, such as Intel SGX and ARM TrustZone, enables a new approach for privacy-preserving computation. Hardware-supported secure computing can greatly accelerate an MPC process by avoiding expensive cryptographic operations. However, this kind of constructions introduce additional hardware setup assumptions that require new trust roots, e.g. Intel. Recent exposure of Intel source code [1] raises a security concern on possible backdoors contained in its design. Moreover, many side-channel and micro-architecture attacks [2]–[6] have been discovered to compromise the security guarantees provided by trusted hardware components.

When the hardware manufacturer is not fully trusted, sensitive information may be leaked through backdoors, side-channels, steganography and kleptography, etc. For instance, Intel SGX utilizes the remote attestation mechanism to ensure the validity of the enclave execution environment and the integrity of the computation result. More specifically, Intel's (anonymous) attestation is based on an anonymous group signature scheme called Intel Enhanced Privacy ID (EPID) [7]. To verify that an outcome is computed by a pre-agreed program in a genuine SGX, Quoting Enclave (QE) will produce a quote by signing the report with the group signature. The users then need to contact the remote Intel Attestation Service (IAS) (or some other alternative servers) for verification. If Intel is malicious, sensitive information may be leaked from the SGX component to the IAS through the signatures, using for example kleptography techniques. (Currently, Intel SGX uses 4000-bit RSA signatures.) That means the input of SGX might be revealed to the adversary during the protocol execution.

When the hardware provider is not allied with the MPC participants, is it possible to still use potentially malicious leaky hardware components to accelerate MPC executions with privacy assurance?[1] In this work, we answer this question affirmatively.

New model. We introduce a new semi-trusted hardware model, where the adversary $\mathcal{A}$ is allowed to passively or maliciously corrupt the hardware ideal functionality $\mathcal{F}_{\text{HW}}$. $\mathcal{F}_{\text{HW}}$ is parameterized with a PPT ITM M, which specifies its functionality. When the hardware functionality $\mathcal{F}_{\text{HW}}$ is passively corrupted, the adversary $\mathcal{A}$ can learn all the incoming messages received by $\mathcal{F}_{\text{HW}}$; when $\mathcal{F}_{\text{HW}}$ is maliciously corrupted, in addition to leaning the incoming messages, the adversary $\mathcal{A}$ can replace the original M with an arbitrary ITM M^*; namely the adversary $\mathcal{A}$ can fully control the execution of $\mathcal{F}_{\text{HW}}$.

We formalize our model in the Universal Composibility (UC) framework [8]. In this security framework, the adversary $\mathcal{A}$ is allowed to control the network and corrupt some machines (ideal functionalities and/or MPC players). In this work, we focus on two-party computation, and we circumvent some impossibility results, we introduce some restrictions to the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ to enable efficient constructions. More precisely, we assume the hardware manufacturer will not collude with the

Bingsheng Zhang is the corresponding author.

1. In this work, we don't address the information leakage problem from SGX to the host PC during the execution via the side-channel attacks.

MPC players; therefore, we restrict the adversary $\mathcal{A}$ to only corrupt either the semi-trusted hardware functionality $\mathcal{F}_{\text{HW}}$ or the player(s) P_1 (and/or P_2).

Our constructions. We propose a new type of two-party computation (2PC) protocols called *silent MPC* that uses semi-trusted hardware to significantly reduce the communication between the 2PC players. The main idea is to use semi-trusted hardware for those MPC computation that does not depend on the actual protocol inputs; thus no sensitive information is leaked to the hardware components. Take random OT (ROT) generation as an example, assume the Receiver uses an SGX-enabled machine, while there is no special hardware requirement to the Sender. During the ROT protocol, the Sender only needs to exchange a random seed ω with the Receiver's SGX enclave via a secure channel. Both parties can then generate polynomially many ROT copies without any further communication. Namely, for each ROT copy, the Sender locally computes $R_{\mathsf{ctr}}^{(0)} \leftarrow \mathsf{PRF}_{\omega}(\mathsf{ctr}, 0)$ and $R_{\mathsf{ctr}}^{(1)} \leftarrow \mathsf{PRF}_{\omega}(\mathsf{ctr}, 1)$ from the seed ω using some pseudo-random function PRF, where ctr is the counter; meanwhile, the SGX picks a random bit $b_{\mathsf{ctr}} \leftarrow \{0, 1\}$, and then it generates $R_{\mathsf{ctr}}^{(b_{\mathsf{ctr}})} \leftarrow \mathsf{PRF}_{\omega}(\mathsf{ctr}, b_{\mathsf{ctr}})$. The SGX locally outputs $\langle b_{\mathsf{ctr}}, R^{(b_{\mathsf{ctr}})_{\mathsf{ctr}}} \rangle$ to the Receiver.

For garbled circuit (GC) evaluation, the communication between the 2PC players can also be dramatically reduced. Similarly, we assume the GC Evaluator uses an SGX-enabled machine, while there is no special hardware requirement to the GC Garbler. Note that, the main cost of a GC-based 2PC protocol is the transmission of the garbled tables of the entire circuit. Analogously, during the GC protocol, the Garbler exchanges a random seed k with the Evaluator's SGX enclave via a secure channel. The SGX can then internally generate the garbled tables and locally outputs them to the Evaluator without network communication. The only communication needed is for transmitting the input labels from the Garbler to the Evaluator. Hence, the overall communication is linear to the input size and independent of the circuit size.

Remark. We would like to emphasize that naively using the secure hardware components, such as SGX, to prepare the ROT copies and GC tables in the offline phase won't result in a (UC) simulatable 2PC protocol. This is because the simulator cannot extract the malicious Evaluator's input in the offline phase, yet it needs to learn the MPC output (from the ideal functionality) to simulate the (fake) GC tables in the real/hybrid world. As described in Sec. 4 later, the protocol should invoke the secure hardware component at the right moment along with the 2PC protocol execution.

To handle malicious adversaries, the 2PC players need to check the correctness of the garbled circuit(s) generated by $\mathcal{F}_{\text{HW}}$, we adopt the *cut-and-choose* technique, i.e., $\mathcal{F}_{\text{HW}}$ generates a number of garbled circuits; P_2 checks some of them and evaluates the others, and the majority of the output of the evaluation circuits will be considered as the final output. Note that there are two subtle problems should be addressed when adopting the cut-and-choose technique.

- A malicious party (P_1 and/or P_2) may use inconsistent inputs for different garbled circuits.
- P_1 may carry out a selective failure attack on P_2's input, e.g., P_1 may selectively provide incorrect wire labels for some of P_2's input; according to whether the protocol aborts, P_1 can learn some information about P_2's input.

To address P_1's input inconsistency problem, we utilize the *XOR-gadget* technique introduced by Mohassel and Riva [10].

To address P_2's input inconsistency problem, we use single-choice OT (SCROT) to ensure the choice bit of P_2 for the same input bit is consistent among all the garbled circuits.

To address the selective failure problem, we adopt the cut-and-choose OT technique introduced by Lindell and Pinkas [11] to our scenario.

Efficiency. Table. 1 shows the performance comparison between the well-known EMP-ROT [9] protocol and our silent ROT protocol. We perform the experiments on an SGX-enabled Dell OptiPlex 7080 equipped with an Intel Core 8700 CPU @ 3.20 GHz with 32 GB RAM. In the LAN setting (Bandwidth: 1Gbps, Delay: 1ms), our silent ROT protocol is up to 207X faster w.r.t. the sender's running time and up to 50X faster w.r.t. the receiver's running time than the EMP-ROT [9]. In the WAN setting (Bandwidth: 100Mbps, Delay: 25ms), our silent ROT protocol is up to 1151X faster w.r.t. the sender's running time and up to 269X faster w.r.t. the receiver's running time than the EMP-ROT.

Table. 2 shows the performance comparison between the well-known EMP-SH2PC [9] protocol and our semi-honest setting silent 2PC protocol. We perform the experiments on this same machine as above. We test the garbling time, the garbled tables transmission time, and the evaluation time separately. Since in our protocol, the garbling process is also performed in the SGX enclave at the evaluator side, we split the evaluator running time of our protocol into two parts: (i) the SGX running time and (ii) normal mode CPU running time. We take the AES-non-expanded, SHA-1, SHA-256 circuit evaluation as benchmarks. In the LAN setting, our silent 2PC protocol is 4-5X faster than the EMP-SH2PC [9]. In the WAN setting, our silent 2PC protocol is 40-50X faster than the EMP-SH2PC.

Roadmaps. In Sec. 2 we provide notations and the necessary background knowledge for garbling scheme and Intel SGX. In Sec. 3 we describe the 2PC functionality under the UC framework and introduce the semi-trusted hardware model. In Sec. 4, we present our silent 2PC protocols both in the semi-honest setting and malicious setting. We then examine the security of our protocols in Sec. 5. Further, we provide our implementation details and benchmarks in Sec. 6. Finally, we discuss the related works in Sec. 7 and give a conclusion.

2. Preliminaries

Notation. Throughout this paper, we use the following notations and terminologies. Let $\lambda \in \mathbb{N}$ be the security parameter. Denote the set $\{a, a+1, \ldots, b\}$ by $[a, b]$, let $[b]$ denote $[1, b]$, and let empty set denote $\emptyset$. When A is an array, $|A|$ stands for the size of A in terms of the number of entries. We abbreviate *probabilistic polynomial time* as PPT. When S is a set, $s \leftarrow S$ stands for sampling s uniformly at random from S. When A

TABLE 1: Performance comparison of the ROT protocol (Result obtained from SGX-enabled Dell OptiPlex 7080 (Intel Core 8700 CPU @ 3.20 GHz, 32 GB RAM). Our protocol is running on Windows 10 2004, and the EMP-ROT is running on Ubuntu 18.04 LTS).

# ROT	Network setting	Sender's running time (in ms)		Receiver's running time (in ms)	
		EMP-ROT [9]	Our ROT	EMP-ROT [9]	Our ROT
1×10^4	LAN (Bandwidth: 1Gbps, Delay: 1ms)	14.953	0.072	11.879	0.224
	WAN (Bandwidth: 100Mbps, Delay: 25ms)	87.482	0.076	60.286	0.237
1×10^5	LAN (Bandwidth: 1Gbps, Delay: 1ms)	27.504	0.649	14.595	2.408
	WAN (Bandwidth: 100Mbps, Delay: 25ms)	229.526	0.684	70.298	2.409
1×10^6	LAN (Bandwidth: 1Gbps, Delay: 1ms)	176.531	5.905	148.909	19.230
	WAN (Bandwidth: 100Mbps, Delay: 25ms)	1431.493	5.953	1218.493	21.709
1×10^7	LAN (Bandwidth: 1Gbps, Delay: 1ms)	1588.928	58.151	1420.503	181.545
	WAN (Bandwidth: 100Mbps, Delay: 25ms)	13266.754	58.766	13059.677	189.718
1×10^8	LAN (Bandwidth: 1Gbps, Delay: 1ms)	15789.809	567.430	14132.705	1739.177
	WAN (Bandwidth: 100Mbps, Delay: 25ms)	132417.995	571.185	130772.327	1758.607

TABLE 2: Performance comparison of the semi-honest setting 2PC protocol (Result obtained from SGX-enabled Dell OptiPlex 7080 (Intel Core 8700 CPU @ 3.20 GHz, 32 GB RAM, OS: Ubuntu 18.04 LTS). It shows the running time (in ms) for evaluating AES-non-expanded, SHA-1, and SHA-256 circuits 1000 times, respectively.

Circuit	Network setting	EMP-SH2PC [9] running time (in ms)			Our 2PC protocol running time (in ms)		
		Garbler	Transmission	Evaluator	Garbler	Transmission	Evaluator (SGX+PC)
AES-non-expanded	LAN (1Gbps, 1ms)	252.854	1842.682	237.081	≈ 0	≈ 0	266.184 + 198.824
	WAN (100Mbps, 25ms)	268.459	19259.730	258.939	≈ 0	≈ 0	272.311 + 202.442
SHA-1	LAN (1Gbps, 1ms)	1477.546	10125.825	1435.283	≈ 0	≈ 0	1264.749 + 1086.122
	WAN (100Mbps, 25ms)	1481.897	104323.883	1439.263	≈ 0	≈ 0	1267.931 + 1085.896
SHA-256	LAN (1Gbps, 1ms)	3738.750	24493.608	3570.436	≈ 0	≈ 0	3051.586 + 2662.728
	WAN (100Mbps, 25ms)	3523.648	257959.467	3341.378	≈ 0	≈ 0	3052.418 + 2665.292

is a randomised algorithm, $y \leftarrow A(x)$ stands for running A on input x with a fresh random coin r. When needed, we denote $y := A(x; r)$ as running A on input x with the explicit random coin r. Let $\mathrm{poly}(\cdot)$ and $\mathsf{negl}(\cdot)$ be a polynomially-bounded function and negligible function, respectively. We assume each party has a unique PID, and for readability, we refer P_i as the PID for the party P_i.

Garbling Scheme. As defined in [12], a garbling scheme GC consists of the following PPT algorithms $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$.

- $\mathsf{Gb}(1^\lambda, f)$ is the garbling algorithm that takes input as the security parameter $\lambda \in \mathbb{N}$ and a circuit f, and it returns a garbled circuit F, encoding information e, and decoding information d.
- $\mathsf{En}(e, x)$ is the encoding algorithm that takes input as the encoding information e and an input x, and it returns a garbled input X.
- $\mathsf{Ev}(F, X)$ is the evaluation algorithm that takes input as the garbled circuit F and the garbled input X, and it returns a garbled output Y.
- $\mathsf{De}(d, Y)$ is the decoding algorithm that takes input as the decoding information d and the garbled output Y, and it returns the plaintext output y.

A garbling scheme $\mathsf{GC} := (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ is called projective if e consists of $2n$ wire labels, where n is the number of input bits. For notation simplicity, we denote n_1 and n_2 as the input size of x_1 and x_2, respectively, and $n_1 + n_2 = n$. For the i-th input bit, we denote the corresponding wire labels as $(X_i^{(0)}, X_i^{(1)})$. Let $e := \{(X_i^{(0)}, X_i^{(1)})\}_{i\in[n]}$; the encoding algorithm $\mathsf{En}(e, x)$ simply outputs $X_i^{(x[i])}, i \in [n]$, where $x[i]$ stands for the i-th bit of x.

Analogously, a garbling scheme is called output-projective if d consists of 2 labels for each output bits, which can be denoted as $(Z_i^{(0)}, Z_i^{(1)})$. we use m to represent the length of the output. Let $d := \{(Z_i^{(0)}, Z_i^{(1)})\}_{i\in[m]}$; the decoding algorithm $\mathsf{De}(d, Y)$ outputs $y[i], i \in [m]$, where $y[i]$ is the i-th bit of y such that $Z_i^{y[i]} = Y_i$, simple equality test works for this check.

In this work, we assume GC is both projective and output-projective.

Definition 1 (Correctness [12])**.** *We say a garbling scheme* $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *is correct if for all functions* f *and input* x*:*

$$\Pr[(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, f) : \mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, x))) = f(x)] = 1 \ .$$

Definition 2 (Simulatable Privacy [12])**.** *We say a garbling scheme* $(\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *is simulatable private if for all functions* f *and input* x*, there exists a PPT simulator* Sim *such that for all PPT adversary* $\mathcal{A}$ *the following holds:*

$$\Pr\left[\begin{array}{l}(F_0, e_0, d_0) \leftarrow \mathsf{Gb}(1^\lambda, f); X_0 \leftarrow \mathsf{En}(e, x);\\ (F_1, X_1, d_1) \leftarrow \mathsf{Sim}(1^\lambda, f(x), \Phi(f));\\ b \leftarrow \{0,1\}; b^* \leftarrow \mathcal{A}(F_b, X_b, d_b) : b = b^*\end{array}\right] = \mathsf{negl}(\lambda) \ .$$

where Φ *is the side-information function.*

Yao's GC Optimizations and Our Choice. Throughout the past decades, several optimization techniques have been proposed to improve the efficiency of Yao's garbled circuit (GC). In this section, we examine a few Yao's GC optimizations and analyze their suitability for our work to achieve the best performance.

In the classical garbling scheme, the GC generator needs to invoke PRF 4 times for each gate to create a garbled table consists of 4 ciphertexts. The GC evaluator also needs to invoke PRF up to 4 times for each gate to decrypt all these ciphertexts and obtains an output wire label.

Beaver *et al.* [13] introduced a technique called *point-and-permute*. By appending a select bit to each wire label, one can easily determine the places of the corresponding ciphertexts. Therefore, for a garbled table, the GC evaluator can decide which ciphertext to decrypt according to the select bit and only invoke PRF once. Nevertheless, each garbled table still contains 4 ciphertexts, and it takes 4 PRF calls to generate. We adopt this technique in our design, as it greatly reduces the GC evaluator's computational cost, and it is compatible with other optimizations.

Naor *et al.* [14] introduced a *garbled row-reduction* technique known as GRR3 to reduce the garbled table size. The main idea is to fix 1 of the 4 ciphertexts, e.g. the top one, in each garbled table to be 0, and thus can be eliminated. In our construction, the memory of the enclave is limited, and this technique can reduce memory usage of GC generation.

Kolesnikov *et al.* [15] introduced the *free-XOR* technique. This technique allows us to garble and evaluate XOR gates for free. To do this, the offset between a wire's 0-label and its 1-label of the entire circuit is fixed to Δ. Therefore, one can generate or evaluate an XOR gate via a simple XOR operation. This technique can greatly improve the performance of our scheme.

We note that, in a conventional 2PC setting, the other optimization techniques, such as *GRR2* [16] and *half-gates* [17], may be helpful to further improve scheme performance. However, GRR2 is not compatible with free-XOR. Although half-gates is compatible with the aforementioned three optimizations, it is not ideal for our construction. The reason is that the main benefit of half-gates is to reduce the non-XOR gate garbled table size to 2, but it needs 2 PRF invocations to evaluate. Whereas, in our design, the GC size is not the bottleneck of our overall performance, because the GC is transmitted between the SGX enclave and the host locally. While, without half-gates, each non-XOR gate garbled table only needs 1 PRF invocation to evaluate.

Intel SGX. Intel Software Guard Extensions (SGX) is a widely used technology that enhances security of data and code. It allows developers to create guarded private region called enclave in processor reserved memory (PRM) and execute programs in the enclave. The enclave is a isolated execution environment, high-level softwares, including operating system and BIOS, can't break down the integrity and confidentiality guarantees of its computation. In execution, a party can remotely attest the genuinity of an enclave, provide private information to the enclave and verify the outcome is computed by a pre-agreed program with an advanced feature of Intel SGX called remote attestation. More specifically, Intel's (anonymous) attestation is based on an anonymous group signature scheme called Intel Enhanced Privacy ID (EPID) [7]. The enclave to be attested first invoke the EREPORT instruction to create a locally verifiable report of its attributes and measurement, and send this report to a special enclave named Quoting Enclave (QE). The QE verifies the report and produce a remotely verifiable quote by signing the quote with the group signature. The enclave then forwards the quote to the challenge party, and the party can contact with the remote Intel Attestation Service (IAS) server for verification. The IAS will first verify the group signature and then create a attestation verification report as a response.

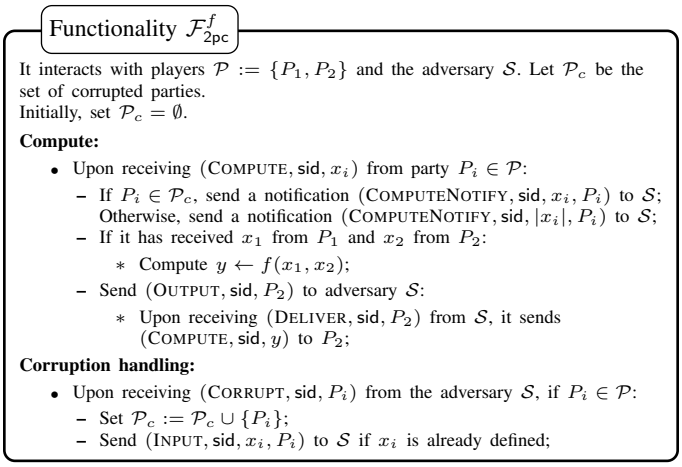

Figure 1: Functionality $\mathcal{F}_{\mathsf{2pc}}^{f}$

3. Security Model

Universal Composibility. Our security model is based on the Universal Composibility (UC) framework [8], which lays down a solid foundation for designing and analyzing protocols secure against attacks in an arbitrary *network* execution environment (therefore it is also known as *network aware security model*). Roughly speaking, in the UC framework, protocols are carried out over multiple interconnected machines; to capture attacks, a network adversary $\mathcal{A}$ is introduced, which is allowed to corrupt some machines (i.e., have the full control of all physical parts of some machines); in addition, $\mathcal{A}$ is allowed to partially control the communication tapes of all uncorrupted machines, that is, it sees all the messages sent from and to the uncorrupted machines and controls the sequence in which they are delivered. Then, a protocol ρ is a UC-secure implementation of a functionality $\mathcal{F}$, if it satisfies that for every network adversary $\mathcal{A}$ attacking an execution of ρ, there is another adversary $\mathcal{S}$—known as the simulator—attacking the ideal process that uses $\mathcal{F}$ (by corrupting the same set of machines), such that, the executions of ρ with $\mathcal{A}$ and that of $\mathcal{F}$ with $\mathcal{S}$ makes no difference to any network execution environment.

The idea world execution. In the ideal world, P_1 and P_2 only communicate with an ideal functionality $\mathcal{F}_{\mathsf{2pc}}^{f}$ during the execution. As depicted in Fig. 1, party $P_i \in \mathcal{P}$ sends $(\textsc{Compute}, \mathsf{sid}, x_i)$ to the functionality $\mathcal{F}_{\mathsf{2pc}}^{f}$, and $\mathcal{F}_{\mathsf{2pc}}^{f}$ sends a notification $(\textsc{ComputeNotify}, \mathsf{sid}, x_i, P_i)$ to the adversary $\mathcal{S}$ if P_i is corrupted; Otherwise, $\mathcal{F}_{\mathsf{2pc}}^{f}$ leaks the input size $(\textsc{ComputeNotify}, \mathsf{sid}, |x_i|, P_i)$ to $\mathcal{S}$. When both parties' inputs are received, $\mathcal{F}_{\mathsf{2pc}}^{f}$ computes $y \leftarrow f(x_1, x_2)$. It then sends $(\textsc{Compute}, \mathsf{sid}, y)$ to P_2 if the adversary $\mathcal{S}$ allows. For corruption handling, if the adversary $\mathcal{S}$ corrupts party $P_i \in \mathcal{P}$, $\mathcal{F}_{\mathsf{2pc}}^{f}$ adds P_i to the set of corrupted parties, $\mathcal{P}_c$, and leaks P_i's input x_i to $\mathcal{S}$ if it is already defined.

The real world execution. The real/hybrid world protocol Π utilises a semi-trusted hardware components, which are modeled

Semi-trusted Hardware Functionality $\mathcal{F}_{\text{HW}}[\mathsf{M}]$

It interacts with players $\mathcal{P} := \{P_1, P_2\}$ and the adversary $\mathcal{A}$. It is parameterized with a PPT ITM M and a Boolean flag corrupted.
Initially, set corrupted := false;

- Upon receiving $(\text{CORRUPT}, \mathsf{sid}, \mathsf{M}^*)$ from $\mathcal{A}$:
  - Set corrupted := true;
  - If $\mathsf{M}^* \neq \emptyset$, replace $\mathsf{M} := \mathsf{M}^*$;
- Upon receiving $(\text{RUN}, \mathsf{sid}, x_i)$ from party $P_i \in \mathcal{P}$:
  - If corrupted = true:
    - Send leakage message $(\text{RUNNOTIFY}, \mathsf{sid}, x_i, P_i)$ to $\mathcal{A}$;
  - If corrupted = false:
    - Send notification message $(\text{RUNNOTIFY}, \mathsf{sid}, P_i)$ to $\mathcal{A}$;
  - When $(\text{RUN}, \mathsf{sid}, x_1)$ and $(\text{RUN}, \mathsf{sid}, x_2)$ are both received:
    - Run $(y_1, y_2) \leftarrow \mathsf{M}(x_1, x_2)$;
    - For $i \in \{1, 2\}$, send $(\mathsf{Run}, \mathsf{sid}, y_i)$ to P_i;

Figure 2: The semi-trusted hardware functionality $\mathcal{F}_{\text{HW}}[\mathsf{M}]$

Description of M^{s}

$\mathsf{M}^{\mathsf{s}}(x_1, x_2)$:

- Parse $x_1 = \langle \mathsf{CMD}_1, x_1' \rangle$ and $x_2 = \langle \mathsf{CMD}_2, x_2' \rangle$;
- If $\mathsf{CMD}_1 = \mathsf{CMD}_2 = \text{GC}$:
  - Parse $x_1' = f_1$ and $x_2' = f_2$;
  - Assert $f_1 = f_2$;
  - Pick random $k \leftarrow \{0,1\}^\lambda$;
  - Generate $(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, f_1; k)$;
  - Return $y_1 := k$ and $y_2 := (F, d)$;
- If $\mathsf{CMD}_1 = \mathsf{CMD}_2 = \text{ROT}$:
  - Parse $x_1' = \ell_1$ and $x_2' = \ell_2$;
  - Assert $\ell_1 = \ell_2$;
  - Pick random $\omega \leftarrow \{0,1\}^\lambda$;
  - For $i \in [\ell_1]$:
    - Pick random $b_i \leftarrow \{0,1\}$;
    - Generate $R_i^{(b_i)} \leftarrow \mathsf{PRF}_\omega(i, b_i)$;
  - Return $y_1 := \omega$ and $y_2 := \{\langle b_i, R_i^{(b_i)} \rangle\}_{i \in [\ell_2]}$;

Figure 3: Description of M^{s}

as the ideal functionality $\mathcal{F}_{\text{HW}}$. Later, we will discuss how $\mathcal{F}_{\text{HW}}$ is instantiated by Intel SGX in practice. For notation simplicity, we define $\mathcal{F}_{\text{HW}}$ as a template, and specify the required functionalities in the description of a PPT Turing machine M. Our semi-honest setting protocol $\Pi^{\mathsf{s}}_{\mathsf{2pc}}$ uses $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{s}}]$; whereas, our malicious setting protocol $\Pi^{\mathsf{m}}_{\mathsf{2pc}}$ uses $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$.

3.1. Semi-trusted Hardware Model

We introduce a new notion, called *semi-trusted hardware model*. Unlike the conventional trusted hardware model, the semi-trusted hardware functionality $\mathcal{F}_{\text{HW}}[\mathsf{M}]$ shown in Fig. 2 can be corrupted by the adversary $\mathcal{A}$. The functionality $\mathcal{F}_{\text{HW}}[\mathsf{M}]$ is parameterized with a PPT Turing machine M and a Boolean flag corrupted to indicate whether the hardware is corrupted. The parties P_1 and P_2 can invoke $\mathcal{F}_{\text{HW}}[\mathsf{M}]$ to compute $(y_1, y_2) \leftarrow \mathsf{M}(x_1, x_2)$ by sending the input x_1 and x_2 respectively to $\mathcal{F}_{\text{HW}}$. The input x_i is in the form of $\langle \mathsf{CMD}, \mathsf{Data} \rangle$, where CMD is the command that specifies which function M is going to execute, and Data is the function's input.

However, the adversary $\mathcal{A}$ is allowed to corrupt $\mathcal{F}_{\text{HW}}$ via the $(\text{CORRUPT}, \mathsf{sid}, \mathsf{M}^*)$ command. When $\mathcal{A}$ is a semi-honest adversary, it sets $\mathsf{M}^* = \emptyset$. In execution, if $\mathcal{F}_{\text{HW}}$ is corrupted, it will leak each party's input to $\mathcal{A}$. When $\mathcal{A}$ is a malicious adversary, M^* can be arbitrarily defined by $\mathcal{A}$ (not necessarily PPT), and $\mathcal{F}_{\text{HW}}$ computes $(y_1, y_2) \leftarrow \mathsf{M}^*(x_1, x_2)$ instead. After the computation, $\mathcal{F}_{\text{HW}}$ sends the output y_1 to the party P_1 and y_2 to the party P_2.

Description of M^{s}. We now define the Turing machine M^{s} for $\mathcal{F}_{\text{HW}}$ that will be used for our 2PC protocol in the semi-honest adversarial setting. As depicted in Fig. 3, M^{s} accepts two commands – GC (to generate garbled circuit) and ROT (to generate random OT's).

When both P_1 and P_2 submit $\langle \text{GC}, f \rangle$, M^{s} picks a random seed $k \leftarrow \{0,1\}^\lambda$ and generates the garbled circuit $(F, e, d) \leftarrow \mathsf{Gb}(1^\lambda, f; k)$. After that, M^{s} outputs $y_1 := k$ and $y_2 := (F, d)$.

When both P_1 and P_2 submit $\langle \text{ROT}, \ell \rangle$, M^{s} first picks a random key $\omega \leftarrow \{0,1\}^\lambda$. For $i \in [\ell]$, M^{s} picks a random bit $b_i \leftarrow \{0,1\}$, and then it generates $R_i^{(b_i)} \leftarrow \mathsf{PRF}_\omega(i, b_i)$. After that, M^{s} outputs $y_1 := \omega$ and $y_2 := \{\langle b_i, R_i^{(b_i)} \rangle\}_{i \in [\ell]}$.

Instantiation of M^{s}. In practice, M^{s} can be instantiated by just running an SGX enclave on the P_2 side. P_1 will remotely interact with P_2's SGX enclave via a secure channel established by remote attestation. We use 128-bit AES-NI as the PRF algorithm. As introduced in Sec. 2, we adopt three GC optimizations, respectively are point-and-permute, GRR3 and free-XOR. For the GRR3 optimization, we set the 0-label of the output wire as the first row of the garbled table, and XOR each row with this 0-label, then the first row becomes an all 0 string and thus can be eliminated.

Description of M^{m}. The Turing machine M^{m} will be used for our 2PC protocol in the malicious adversarial setting. As depicted in Fig. 4, M^{m} accepts three commands – GC (to generate garbled circuits), ROT (to generate random OT's), and SCROT (to generate single-choice random OT's). Different from M^{s}, all the three commands need an extra argument to specify the output directions of the protocols, i.e., P_1 and P_2 can be the sender (or the receiver) of the oblivious transfer protocol, and the evaluator (or the garbler) of the garbling scheme, respectively. For notation simplicity, we use $\psi \in \{1, 2\}$ to specify the output directions.

When both P_1 and P_2 submit $\langle \text{GC}, f, \ell, \psi \rangle$, M^{m} generates ℓ copies of the garbled circuits – for $j \in [\ell]$: (i) pick a random seed $\omega_j \leftarrow \{0,1\}^\lambda$ and (ii) generate the garbled circuit $(F_j, e_j, d_j) \leftarrow \mathsf{Gb}(1^\lambda, f; \omega_j)$. After that, M^{m} outputs $y_\psi := \{\omega_j\}_{j \in [\ell]}$ and $y_{3-\psi} := \{F_j, d_j\}_{j \in [\ell]}$.

When both P_1 and P_2 submit $\langle \text{ROT}, \ell, \eta, \psi \rangle$, M^{m} generates ℓ batches of the random OT's, and each batch contains η copies of random OT's. For $j \in [\ell]$, it picks a random key $\omega_j \leftarrow \{0,1\}^\lambda$ and computes $\sigma_j \leftarrow \mathsf{hash}(\omega_j)$. In the j-th batch, M^{m} for $i \in [\eta]$, (i) pick random $b_{j,i} \leftarrow \{0,1\}$ and (ii) generate $R_{j,i}^{(b_{j,i})} \leftarrow \mathsf{PRF}_{\omega_j}(b_{j,i})$. After that, M^{m} outputs $y_\psi := \{\omega_j\}_{j \in [\ell]}$ and $y_{3-\psi} := \left\{\sigma_j, \left\{\left\langle b_{j,i}, R_{j,i}^{(b_{j,i})} \right\rangle\right\}_{i \in [\eta]}\right\}_{j \in [\ell]}$.

When both P_1 and P_2 submit $\langle \text{SCROT}, \ell, \eta, \psi \rangle$, M^{m} gener-

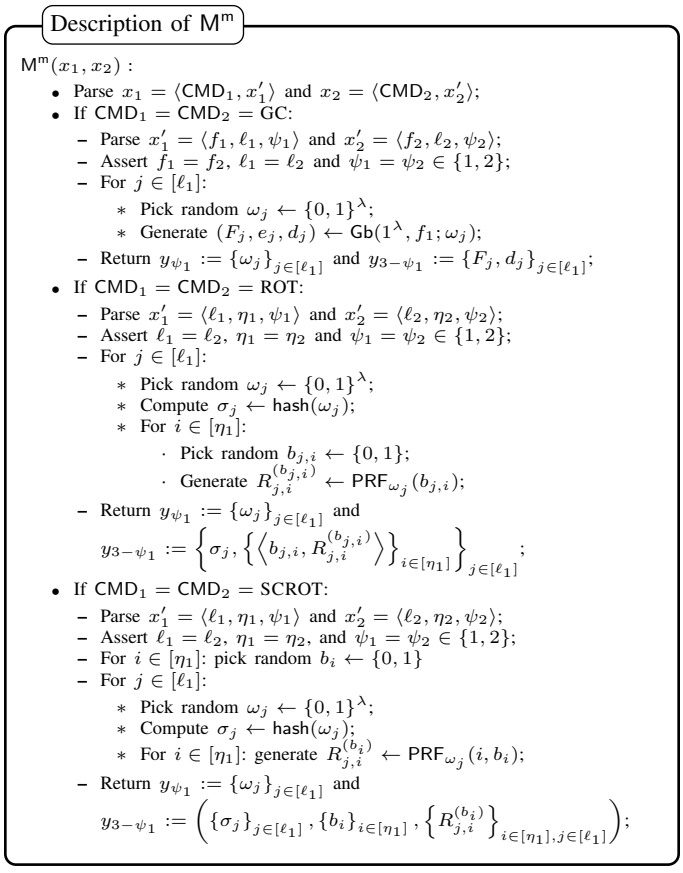

Figure 4: Description of M^{m}

ates ℓ batches of the single-choice random OT's, and each batch contains η copies of single-choice random OT's. The procedure is similar to the random OT generation. The only difference is that in single-choice random OT's, the i-th ROT's in each batch use the same choice bit $b_i \in \{0,1\}$.

Instantiation of M^{m}. In practice, M^{m} can instantiated by running SGX enclaves on both P_1 and P_2 sides. Upon initialization, two enclaves need to jointly pick a common random seed through a coin flipping protocol. We then utilize pseudorandomness instead of picking true random coins to save communication between P_1 and P_2. The hash function is defined as $\mathsf{hash}(x) := \mathsf{PRF}_k(x) \oplus x$, where k is a fresh random key. The GC optimizations are the same as M^{s}.

4. Silent 2PC Protocols

4.1. The semi-honest setting protocol

Let f be the function that P_1 and P_2 want to jointly compute. Denote n_1 and n_2 as the input size of P_1 and P_2 in f, respectively. As depicted in Fig. 5, in the semi-honest setting protocol, both parties first sends $(\mathsf{Run}, \mathsf{sid}, \langle \mathsf{ROT}, n_2 \rangle)$ to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ to generate n_2 ROT copies. After that, P_1 receives ω as the seed of ROT, and it generates $R_i^{(0)} \leftarrow \mathsf{PRF}_\omega(i, 0)$, $R_i^{(1)} \leftarrow \mathsf{PRF}_\omega(i, 1)$, $i \in [n_2]$. Meanwhile, P_2 receives the ROT copies $\{\langle b_i, R_i^{(b_i)} \rangle\}_{i\in[n_2]}$ from $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$, it then computes and sends the choice bits $\{c_i := b_i \oplus x_{2,i}\}_{i\in[n_2]}$ to P_1.

Next, both parties send $(\mathsf{Run}, \mathsf{sid}, \langle \mathsf{GC}, f \rangle)$ to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$. After that, P_1 obtains the GC seed k. P_1 then generates the GC: $(F, e, d) \leftarrow \mathsf{GC.Gb}(1^\lambda, f; k)$[2]. Subsequently, P_1 computes the OT responses $W_i^{(0)} := X_{n_1+i}^{(0)} \oplus R_i^{(c_i)}$, $W_i^{(1)} := X_{n_1+i}^{(1)} \oplus R_i^{(c_i \oplus 1)}$, $i \in [n_2]$, and P_1 sends $\{W_i^{(0)}, W_i^{(1)}\}_{i\in[n_2]}$ along with its input labels $\{Z_i := X_i^{(x_{1,i})}\}_{i\in[n_1]}$ to P_2. P_2 receives the garbled tables, decoding information (F, d) from $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$, and $\{Z_i\}_{i\in[n_1]}$, $\{W_i^{(0)}, W_i^{(1)}\}_{i\in[n_2]}$ from P_1. P_2 computes $Z_{n_1+j} := W_j^{(x_{2,j})} \oplus R_j^{(b_j)}$ for $j \in [n_2]$. At the end, P_2 evaluates $Y \leftarrow \mathsf{GC.Ev}(F, (Z_1, \ldots, Z_{n_1+n_2}))$, and decodes $y \leftarrow \mathsf{GC.De}(d, Y)$.

4.2. The malicious setting protocol

In our security setting, the hardware functionality $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ may be maliciously corrupted. Therefore, our protocol also need to verify the validity of the output of $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$. Before presenting the formal protocol description, we first give some intuition in the following.

Intuition. In the spirit of our design, the evaluator shall receive the garbled circuit(s) from $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ instead of the garbler. Hence, when $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ is instantiated by a local SGX, there is no communication between P_1 and P_2. Since the $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ can be maliciously corrupted, it's necessary to verify if the $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ correctly generates the ROT copies the GC tables. Roughly speaking, we adopt the cut-and-choose technique, which require the $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ to generate $p(\lambda)$ copies of the garbled circuit and allows the evaluator the check some of them, where $p()$ is some polynomial function. After the check is finished, the evaluator will evaluate the unchecked copies and consider the majority of the outputs of the evaluation circuits as the final output. Note that naive cut-and-choose GC tables can only ensure the correctness of the garbled circuits. To achieve active security, in addition, we need to guarantee the followings.

- The players (the garbler and the evaluator) should use the same input when evaluating different garbled circuits.
- There should be a mechanism to prevent a malicious the garbler from carrying out the selective failure attack.

To address the problem that the evaluator uses inconsistent inputs, we introduce a new type of ROT functionality to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ called single-choice random OT (SCROT). That is a batch of ROT's using the same choice bit. This can be used to force the evaluator to use the same choice bit (i.e., input wire value) among different GC copies.

We use the cut-and-choose OT technique [11] to (i) check the validity of the ROT and SCROT copies generated by potentially malicious $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$, and (ii) the selective failure attacks launched by P_1 or P_2. Further, to ensure the garbler's input consistency, we adopt the *XOR-gadget* technique introduced by Mohassel and Riva [10]. In this technique, both the garbler and the evaluator needs to evaluate $x_1 \oplus r$, where r is a random value

2. In practice, P_1 only needs to generate the corresponding encoding information of the GC, e, i.e. the wire labels; therefore, $\mathsf{GC.Gb}$ is only partially executed for efficiency.

Protocol $\Pi^{\mathsf{s}}_{\mathsf{2pc}}$

Let n_1 and n_2 be f's input size of P_1 and P_2, respectively. Denote $n := n_1 + n_2$ as the overall input size.

Protocol description:

- Upon receiving $(\textsc{Compute}, \mathsf{sid}, x_i := (x_{i,1}, \ldots, x_{i,n_i}))$ from the environment $\mathcal{Z}$, the party $P_i \in \mathcal{P}$:
  - Send $(\mathsf{Run}, \mathsf{sid}, \langle \mathrm{ROT}, n_2 \rangle)$ to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$, which will reply $(\mathsf{Run}, \mathsf{sid}, \alpha_i)$;
- Upon receiving $(\mathsf{Run}, \mathsf{sid}, \alpha_1)$ from $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$, the party P_1:
  - Parse $\alpha_1 = \omega$;
  - For $i \in [n_2]$: generate $R_i^{(0)} \leftarrow \mathsf{PRF}_\omega(i, 0)$ and $R_i^{(1)} \leftarrow \mathsf{PRF}_\omega(i, 1)$;
- Upon receiving $(\mathsf{Run}, \mathsf{sid}, \alpha_2)$ from $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$, the party P_2:
  - Parse $\alpha_2 = \{\langle b_i, R_i^{(b_i)} \rangle\}_{i \in [n_2]}$;
  - Send $\{c_i := b_i \oplus x_{2,i}\}_{i \in [n_2]}$ to P_1;
  - Send $(\mathsf{Run}, \mathsf{sid}, \langle \mathsf{GC}, f \rangle)$ to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$;
- Upon receiving $\{c_i\}_{i \in [n_2]}$ from P_2, the party P_1:
  - Send $(\mathsf{Run}, \mathsf{sid}, \langle \mathsf{GC}, f \rangle)$ to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$, and obtain $(\mathsf{Run}, \mathsf{sid}, k)$;
  - Generate $(F, e, d) \leftarrow \mathsf{GC.Gb}(1^\lambda, f; k)$;
  - Parse $e = \{X_i^{(0)}, X_i^{(1)}\}_{i \in [n]}$;
  - For $i \in [n_2]$: compute $W_i^{(0)} := X_{n_1+i}^{(0)} \oplus R_i^{(c_i)}$ and $W_i^{(1)} := X_{n_1+i}^{(1)} \oplus R_i^{(c_i \oplus 1)}$;
  - Send $\{Z_i := X_i^{(x_{1,i})}\}_{i \in [n_1]}$ and $\{W_i^{(0)}, W_i^{(1)}\}_{i \in [n_2]}$ to P_2;
- Upon receiving $(\mathsf{Run}, \mathsf{sid}, (F, d))$ from $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$, and $\{Z_i\}_{i \in [n_1]}$, $\{W_i^{(0)}, W_i^{(1)}\}_{i \in [n_2]}$ from P_1, the party P_2:
  - For $j \in [n_2]$, compute $Z_{n_1+j} := W_j^{(x_{2,j})} \oplus R_j^{(b_j)}$;
  - Evaluate $Y \leftarrow \mathsf{GC.Ev}(F, (Z_1, \ldots, Z_{n_1+n_2}))$;
  - Decode $y \leftarrow \mathsf{GC.De}(d, Y)$;
  - Return $(\textsc{Compute}, \mathsf{sid}, y)$ to the environment $\mathcal{Z}$;

Figure 5: The semi-honest setting protocol $\Pi^{\mathsf{s}}_{\mathsf{2pc}}$ in the $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$-hybrid model

chosen by the garbler. The garbler evaluates the garbled circuits for the function $\hat{f}(x, r) := x \oplus r$ which are called XOR-gadgets, and the evaluator needs to evaluate the garbled circuits for the function $f'((x_1, r), x_2) := (f(x_1, x_2), x_1 \oplus r)$.

As in the work [10], the evaluator use three checks to ensure the garbler use the same input x_1 in the majority of the evaluation circuits: (i) checking if the garbler use the same input value in the evaluation of the XOR-gadgets, we use single-choice OT to guarantee this consistency; (ii) checking if the result of the j-th XOR-gadget equals to the $x_1 \oplus r_j$ from the j-th garbled circuit for the evaluation circuits, we ask the garbler to commit to its evaluation results of the evaluation XOR-gadgets and decommit some of them later to audit; (iii) checking if the same r_j is used in the j-th garbled circuit and the j-th XOR-gadget, we require the garbler to send all the r_j labels for the garbled circuits at the beginning of the protocol, and for the circuits to be checked, the garbler sends the seeds of the garbled circuits and the r_j labels for XOR-gadgets to the evaluator. Then the evaluator can generate the wire labels of those garbled circuits, since the evaluator already has the wire labels of the XOR-gadgets, it can computes the value of r_j in both the j-th garbled circuit and the j-th XOR-gadget and perform the check.

Protocol description. Let n_1 and n_2 be the input size of P_1 and P_2 in circuit f, respectively. Denote $n := n_1 + n_2$ as the overall input size. Let $p()$ be a polynomial function. Define argument function $f'((x_1, r), x_2) := (f(x_1, x_2), x_1 \oplus r)$ and the XOR-gadget function $f_2(x, r) := x \oplus r$.

We use hash-based commitment scheme. Namely, to commit $m \in \{0,1\}^*$, $\mathsf{Com}(m)$, the committer picks a random string $\rho \leftarrow \{0,1\}^\lambda$ and computes the commitment as $\mathsf{hash}(m, \rho)$. ρ is also known as the decommitment value.

As depicted in Fig. 6 and Fig. 7, in the malicious setting protocol, both parties first sends $(\mathsf{Run}, \mathsf{sid}, \langle \mathrm{ROT}, n_1, 2 \rangle)$ and $(\mathsf{Run}, \mathsf{sid}, \langle \mathrm{SCROT}, n_1, 2 \rangle)$ to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ to generate the random OTs that will be used for the XOR-gadgets, then they sends $(\mathsf{Run}, \mathsf{sid}, \langle \mathrm{SCROT}, n_2, 1 \rangle)$ to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ to generate the random OTs that will be used for the garbled circuits.

After that, P_1 obtains the ROT copies and the SCROT copies for the XOR-gadgets, the hash values of the corresponding seeds, along with the seeds of the SCROT copies for the garbled circuits. P_1 then picks $p(\lambda)$ random values $r_j \in \{0,1\}^{n_1}$, and it generates n_2 SCROT copies $\tilde{R}_{j,i}^{(0)} \leftarrow \mathsf{PRF}_{\tilde{\omega}_j}(i, 0)$ and $\tilde{R}_{j,i}^{(1)} \leftarrow \mathsf{PRF}_{\tilde{\omega}_j}(i, 1)$. Subsequently, P_1 sends the select bits $\{\hat{c}_{j,i} := \hat{b}_{j,i} \oplus r_{j,i}\}_{i \in [n_1], j \in [p(\lambda)]}$ and $\{\bar{c}_i := \bar{b}_i \oplus x_{1,i}\}_{i \in [n_1]}$ to P_2, and it sends $(\mathsf{Run}, \mathsf{sid}, \langle \mathsf{GC}, \langle f_2, p(\lambda), 2 \rangle \rangle)$ to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$.

Next, P_2 obtains the ROT seeds and the SCROT seeds for the XOR-gadgets, and the SCROT copies for the garbled circuits from $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$, it also receives the selects bits from P_1. It then sends $(\mathsf{Run}, \mathsf{sid}, \langle \mathsf{GC}, \langle f_2, p(\lambda), 2 \rangle \rangle)$ to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$, which will reply the seeds of the XOR-gadgets $\dot{k}_j$. P_2 uses these seeds to generate $p(\lambda)$ XOR-gadget copies $(\ddot{F}_j, \ddot{e}_j, \ddot{d}_j) \leftarrow \mathsf{GC.Gb}(1^\lambda, f_2; \dot{k}_j)$, where $\ddot{e}_j = \{\ddot{X}_{j,i}^{(0)}, \ddot{X}_{j,i}^{(1)}\}_{i \in [2n_1]}$ are the input wire labels. Subsequently, it uses the ROT seeds to generate $\hat{R}_{j,i}^{(0)} \leftarrow \mathsf{PRF}_{\hat{\omega}_j}(i, 0)$ and $\hat{R}_{j,i}^{(1)} \leftarrow \mathsf{PRF}_{\hat{\omega}_j}(i, 1)$, and it computes $\hat{W}_{j,i}^{(0)} := \ddot{X}_{j,n_1+i}^{(0)} \oplus \hat{R}_{j,i}^{(\hat{c}_{j,i})}$ and $\hat{W}_{j,i}^{(1)} := \ddot{X}_{j,n_1+i}^{(1)} \oplus \hat{R}_{j,i}^{(\hat{c}_{j,i} \oplus 1)}$, it also generates the SCROT copies $\bar{R}_{j,i}^{(0)}, \bar{R}_{j,i}^{(1)}$ and $\bar{W}_{j,i}^{(0)}, \bar{W}_{j,i}^{(1)}$ in the same way. After that, it sends the select bits used in garbled circuits $\{\tilde{c}_i := \tilde{b}_i \oplus x_{2,i}\}_{i \in [n_2]}$ and the encrypted XOR-gadget wire labels $\{\hat{W}_{j,i}^{(0)}, \hat{W}_{j,i}^{(1)}, \bar{W}_{j,i}^{(0)}, \bar{W}_{j,i}^{(1)}\}_{i \in [n_1], j \in [p(\lambda)]}$ to P_1, it also sends $(\mathsf{Run}, \mathsf{sid}, \langle \mathsf{GC}, \langle f_1, p(\lambda), 1 \rangle \rangle)$ to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ to generate the garbled circuit copies.

P_1 then receives the garbled tables and the decoding in-

Protocol $\Pi^{\mathsf{m}}_{\mathsf{2pc}}$

Let n_1 and n_2 be the input size of P_1 and P_2 in circuit f, respectively. Denote $n := n_1 + n_2$ as the overall input size. Let $p()$ be a polynomial function. Define functions $f_1((x_1, r), x_2) := (f(x_1, x_2), x_1 \oplus r)$ and $f_2(x, r) := x \oplus r$.

Protocol Description:

- Upon receiving $(\textsc{Compute}, \mathsf{sid}, x_i := (x_{i,1}, \ldots, x_{i,n_i}))$ from the environment $\mathcal{Z}$, the party $P_i \in \mathcal{P}$:
  - Send $(\mathsf{Run}, \mathsf{sid}, \langle \mathsf{ROT}, n_1, 2\rangle)$ to $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$, which will reply $(\mathsf{Run}, \mathsf{sid}, \alpha_i)$ to P_i;
  - Send $(\mathsf{Run}, \mathsf{sid}, \langle \mathsf{SCROT}, n_1, 2\rangle)$ to $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$, which will reply $(\mathsf{Run}, \mathsf{sid}, \beta_i)$ to P_i;
  - Send $(\mathsf{Run}, \mathsf{sid}, \langle \mathsf{SCROT}, n_2, 1\rangle)$ to $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$, which will reply $(\mathsf{Run}, \mathsf{sid}, \gamma_i)$ to P_i;
- Upon receiving $(\mathsf{Run}, \mathsf{sid}, \alpha_1)$, $(\mathsf{Run}, \mathsf{sid}, \beta_1)$ and $(\mathsf{Run}, \mathsf{sid}, \gamma_1)$ from $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$, the party P_1:
  - Parse $\alpha_1 = \left\{\hat{\sigma}_j, \left\{\left\langle \hat{b}_{j,i}, \hat{R}_{j,i}^{(\hat{b}_{j,i})}\right\rangle\right\}_{i\in[n_1]}\right\}_{j\in[p(\lambda)]}$, $\beta_1 = \left(\{\bar{\sigma}_j\}_{j\in[p(\lambda)]}, \{\bar{b}_i\}_{i\in[n_1]}, \left\{\bar{R}_{j,i}^{(\bar{b}_i)}\right\}_{i\in[n_1],j\in[p(\lambda)]}\right)$, and $\gamma_1 = \{\tilde{\omega}_j\}_{j\in[p(\lambda)]}$;
  - For $j \in [p(\lambda)]$:
    - Pick random $r_j \leftarrow \{0,1\}^{n_1}$;
    - For $i \in [n_2]$, generate $\tilde{R}_{j,i}^{(0)} \leftarrow \mathsf{PRF}_{\tilde{\omega}_j}(i, 0)$ and $\tilde{R}_{j,i}^{(1)} \leftarrow \mathsf{PRF}_{\tilde{\omega}_j}(i, 1)$;
  - Send $\{\hat{c}_{j,i} := \hat{b}_{j,i} \oplus r_{j,i}\}_{i\in[n_1],j\in[p(\lambda)]}$ and $\{\bar{c}_i := \bar{b}_i \oplus x_{1,i}\}_{i\in[n_1]}$ to P_2;
  - Send $(\mathsf{Run}, \mathsf{sid}, \langle \mathsf{GC}, \langle f_2, p(\lambda), 2\rangle\rangle)$ to $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$;
- Upon receiving $(\mathsf{Run}, \mathsf{sid}, \alpha_2)$, $(\mathsf{Run}, \mathsf{sid}, \beta_2)$, $(\mathsf{Run}, \mathsf{sid}, \gamma_2)$ from $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$ and $\{\hat{c}_{j,i}\}_{i\in[n_1],j\in[p(\lambda)]}$, $\{\bar{c}_i\}_{i\in[n_1]}$ from P_1, the party P_2:
  - Send $(\mathsf{Run}, \mathsf{sid}, \langle \mathsf{GC}, \langle f_2, p(\lambda), 2\rangle\rangle)$ to $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$, and obtain $(\mathsf{Run}, \mathsf{sid}, \{\ddot{k}_j\}_{j\in[p(\lambda)]})$ from $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$;
  - Parse $\alpha_2 = \{\hat{\omega}_j\}_{j\in[p(\lambda)]}$, $\beta_2 = \{\bar{\omega}_j\}_{j\in[p(\lambda)]}$, and $\gamma_2 = \left(\{\tilde{\sigma}_j\}_{j\in[p(\lambda)]}, \left\{\tilde{b}_i\right\}_{i\in[n_2]}, \left\{\tilde{R}_{j,i}^{(\tilde{b}_i)}\right\}_{i\in[n_2],j\in[p(\lambda)]}\right)$;
  - For $j \in [p(\lambda)]$:
    - Generate $(\ddot{F}_j, \ddot{e}_j, \ddot{d}_j) \leftarrow \mathsf{GC.Gb}(1^\lambda, f_2; \ddot{k}_j)$, and parse $\ddot{e}_j = \{\ddot{X}_{j,i}^{(0)}, \ddot{X}_{j,i}^{(1)}\}_{i\in[2n_1]}$;
    - For $i \in [n_1]$:
      - Generate $\hat{R}_{j,i}^{(0)} \leftarrow \mathsf{PRF}_{\hat{\omega}_j}(i, 0)$ and $\hat{R}_{j,i}^{(1)} \leftarrow \mathsf{PRF}_{\hat{\omega}_j}(i, 1)$;
      - Generate $\bar{R}_{j,i}^{(0)} \leftarrow \mathsf{PRF}_{\bar{\omega}_j}(i, 0)$ and $\bar{R}_{j,i}^{(1)} \leftarrow \mathsf{PRF}_{\bar{\omega}_j}(i, 1)$;
      - Compute $\hat{W}_{j,i}^{(0)} := \ddot{X}_{j,n_1+i}^{(0)} \oplus \hat{R}_{j,i}^{(\hat{c}_{j,i})}$ and $\hat{W}_{j,i}^{(1)} := \ddot{X}_{j,n_1+i}^{(1)} \oplus \hat{R}_{j,i}^{(\hat{c}_{j,i}\oplus 1)}$;
      - Compute $\bar{W}_{j,i}^{(0)} := \ddot{X}_{j,i}^{(0)} \oplus \bar{R}_{j,i}^{(\bar{c}_{j,i})}$ and $\bar{W}_{j,i}^{(1)} := \ddot{X}_{j,i}^{(1)} \oplus \bar{R}_{j,i}^{(\bar{c}_{j,i}\oplus 1)}$;
  - Send $\{\tilde{c}_i := \tilde{b}_i \oplus x_{2,i}\}_{i\in[n_2]}$, $\{\hat{W}_{j,i}^{(0)}, \hat{W}_{j,i}^{(1)}, \bar{W}_{j,i}^{(0)}, \bar{W}_{j,i}^{(1)}\}_{i\in[n_1],j\in[p(\lambda)]}$ to P_1;
  - Send $(\mathsf{Run}, \mathsf{sid}, \langle \mathsf{GC}, \langle f_1, p(\lambda), 1\rangle\rangle)$ to $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$;
- Upon receiving $(\mathsf{Run}, \mathsf{sid}, \{(\ddot{F}_j, \ddot{d}_j)\}_{j\in[p(\lambda)]})$ from $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$, and $\{\tilde{c}_i\}_{i\in[n_2]}$, $\{\hat{W}_{j,i}^{(0)}, \hat{W}_{j,i}^{(1)}, \bar{W}_{j,i}^{(0)}, \bar{W}_{j,i}^{(1)}\}_{i\in[n_1],j\in[p(\lambda)]}$ from P_2, the party P_1:
  - Send $(\mathsf{Run}, \mathsf{sid}, \langle \mathsf{GC}, \langle f_1, p(\lambda), 1\rangle\rangle)$ to $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$, and obtain $(\mathsf{Run}, \mathsf{sid}, \{k_j\}_{j\in[p(\lambda)]})$ from $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$;
  - For $j \in [p(\lambda)]$:
    - Generate $(F_j, e_j, d_j) \leftarrow \mathsf{GC.Gb}(1^\lambda, f_1; k_j)$, and parse $e_j = \{X_{j,i}^{(0)}, X_{j,i}^{(1)}\}_{i\in[2n_1+n_2]}$;
    - For $i \in [n_2]$, compute $\tilde{W}_{j,i}^{(0)} := X_{j,i}^{(0)} \oplus \tilde{R}_{j,i}^{(\tilde{c}_{j,i})}$ and $\tilde{W}_{j,i}^{(1)} := X_{j,i}^{(1)} \oplus \tilde{R}_{j,i}^{(\tilde{c}_{j,i}\oplus 1)}$;
    - For $i \in [n_1]$, compute $\ddot{Z}_{j,i} := \bar{W}_{j,i}^{(x_{1,i})} \oplus \bar{R}_{j,i}^{(\bar{b}_{j,i})}$ and $\ddot{Z}_{j,n_1+i} := \hat{W}_{j,i}^{(r_{j,i})} \oplus \hat{R}_{j,i}^{(\hat{b}_{j,i})}$;
  - Send $\{Z_{j,n_1+i} := X_{j,n_1+i}^{(r_{j,i})}\}_{i\in[n_1],j\in[p(\lambda)]}$ and $\{\tilde{W}_{j,i}^{(0)}, \tilde{W}_{j,i}^{(1)}\}_{i\in[n_2],j\in[p(\lambda)]}$ to P_2;
- Upon receiving $(\mathsf{Run}, \mathsf{sid}, \{(F_j, d_j)\}_{j\in[p(\lambda)]})$ from $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{s}}]$, and $\{Z_{j,n_1+i}\}_{i\in[n_1],j\in[p(\lambda)]}$, $\{\tilde{W}_{j,i}^{(0)}, \tilde{W}_{j,i}^{(1)}\}_{i\in[n_2],j\in[p(\lambda)]}$ from P_1, the party P_2:
  - For $i \in [n_2]$, compute $Z_{j,2n_1+i} := \tilde{W}_{j,i}^{(x_{2,i})} \oplus \tilde{R}_{j,i}^{(\tilde{b}_{j,i})}$;

Figure 6: The malicious setting protocol $\Pi^{\mathsf{m}}_{\mathsf{2pc}}$ in the $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$-hybrid model (Part I)

formation of the XOR-gadgets from $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$, and $\{\tilde{c}_i\}_{i\in[n_2]}$, $\{\hat{W}_{j,i}^{(0)}, \hat{W}_{j,i}^{(1)}, \bar{W}_{j,i}^{(0)}, \bar{W}_{j,i}^{(1)}\}_{i\in[n_1],j\in[p(\lambda)]}$ from P_2. It sends the command $(\mathsf{Run}, \mathsf{sid}, \langle \mathsf{GC}, \langle f_1, p(\lambda), 1\rangle\rangle)$ to $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$, and obtains $(\mathsf{Run}, \mathsf{sid}, \{k_j\}_{j\in[p(\lambda)]})$. Next, it uses the garbled circuit seeds to generate $(F_j, e_j, d_j) \leftarrow \mathsf{GC.Gb}(1^\lambda, f_1; k_j)$, where $e_j = \{X_{j,i}^{(0)}, X_{j,i}^{(1)}\}_{i\in[2n_1+n_2]}$ are the input wire labels in garbled circuits. It then computes $\tilde{W}_{j,i}^{(0)} := X_{j,i}^{(0)} \oplus \tilde{R}_{j,i}^{(\tilde{c}_{j,i})}$ and $\tilde{W}_{j,i}^{(1)} := X_{j,i}^{(1)} \oplus \tilde{R}_{j,i}^{(\tilde{c}_{j,i}\oplus 1)}$ as the encrypted wire labels in garbled circuits. Subsequently, it decrypts its input wire labels in XOR-gadgets by computing e $\ddot{Z}_{j,i} := \bar{W}_{j,i}^{(x_{1,i})} \oplus \bar{R}_{j,i}^{(\bar{b}_{j,i})}$ and $\ddot{Z}_{j,n_1+i} := \hat{W}_{j,i}^{(r_{j,i})} \oplus \hat{R}_{j,i}^{(\hat{b}_{j,i})}$. After that, P_1 sends the wire labels corresponding to r_j in the garbled circuits $\{Z_{j,n_1+i} := X_{j,n_1+i}^{(r_{j,i})}\}_{i\in[n_1],j\in[p(\lambda)]}$ and $\{\tilde{W}_{j,i}^{(0)}, \tilde{W}_{j,i}^{(1)}\}_{i\in[n_2],j\in[p(\lambda)]}$ to P_2, and P_2 computes $Z_{j,2n_1+i} := \tilde{W}_{j,i}^{(x_{2,i})} \oplus \tilde{R}_{j,i}^{(\tilde{b}_{j,i})}$ to decrypts its wire labels in the garbled circuits.

P_1 and P_2 then jointly choose a random set $\mathcal{I} \subseteq [p(\lambda)]$ such that $|\mathcal{I}| = p(\lambda)/2$ as the check set; $[p(\lambda)] \setminus \mathcal{I}$ is called evaluation set. They then perform the following cut-and-choose procedure.

P_1 evaluates all the evaluation XOR-gadgets, generates commitments for the outputs and sends these commitments to P_2. P_1 also sends the seeds of the check garbled circuits along with the SCROT seeds used for these garbled circuits, and the wire labels of r_j in the check XOR-gadgets to P_2. Subsequently, for the check set, P_2 computes the hash values of the SCROT seeds and asserts these values equals to the hash values obtained from $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$. P_2 then generates the garbled circuits copies by invoking $\mathsf{GC.Gb}$, and it asserts these copies equals to those obtained from $\mathcal{F}_{\mathsf{HW}}[\mathsf{M}^{\mathsf{m}}]$. Next, P_2 generates the SCROT copies $R_{j,i}^{*(0)} \leftarrow \mathsf{PRF}_{\tilde{\omega}_j}(0)$ and $R_{j,i}^{*(1)} \leftarrow \mathsf{PRF}_{\tilde{\omega}_j}(1)$, and it decrypts P_1's inputs for the SCROT's, it then asserts P_1's inputs are exactly the wire labels corresponding to P_2's inputs in garbled circuits. After that, P_2 extracts the value of r_j that P_1 used in garbled

Protocol $\Pi^{\mathsf{m}}_{\mathsf{2pc}}$

▷ *Cut and choose*

- The party P_1 and P_2 use coin-flipping protocol to jointly choose a random set $\mathcal{I} \subseteq [p(\lambda)]$ such that $|\mathcal{I}| = p(\lambda)/2$;
- The party P_1:
  - For $j \in [p(\lambda)] \setminus \mathcal{I}$:
    - Evaluate $\ddot{Y}_j \leftarrow \mathsf{GC.Ev}\left(\ddot{F}_j, \left(\ddot{Z}_{j,1}, \ldots, \ddot{Z}_{j,2n_1}\right)\right)$;
    - Generate a commitment $(\mathfrak{C}_j, \mathfrak{D}_j) \leftarrow \mathsf{Com}(\ddot{Y}_j)$;
  - Sends $\left\{\ddot{Z}_{j,n_1+i} := \ddot{X}^{(r_{j,i})}_{j,n_1+i}\right\}_{i\in[n_1],j\in\mathcal{I}}$, $\{k_j\}_{j\in\mathcal{I}}$, $\{\tilde{\omega}_j\}_{j\in\mathcal{I}}$ and $\{\mathfrak{C}_j\}_{j\in[p(\lambda)]\setminus\mathcal{I}}$ to P_2;
- Upon receiving $\left\{\ddot{Z}_{j,i}\right\}_{i\in[n_1+1,2n_1],j\in\mathcal{I}}$, $\{k_j\}_{j\in\mathcal{I}}$, $\{\tilde{\omega}_j\}_{j\in\mathcal{I}}$ and $\{\mathfrak{C}_j\}_{j\in[p(\lambda)]\setminus\mathcal{I}}$ from P_1, the party P_2:
  - For $j \in \mathcal{I}$:
    - Assert $\mathsf{hash}(\tilde{\omega}_j) = \tilde{\sigma}_j$;
    - Generate $(F^*_j, e^*_j, d^*_j) \leftarrow \mathsf{GC.Gb}(1^\lambda, f_1; k_j)$, and parse $e^*_j = \left\{X^{(0)*}_{j,i}, X^{(1)*}_{j,i}\right\}_{i\in[2n_1+n_2]}$;
    - Assert $F^*_j = F_j$ and $d^*_j = d_j$;
    - For $i \in [n_2]$:
      - Generate $R^{*(0)}_{j,i} \leftarrow \mathsf{PRF}_{\tilde{\omega}_j}(0)$ and $R^{*(1)}_{j,i} \leftarrow \mathsf{PRF}_{\tilde{\omega}_j}(1)$;
      - Compute $X'^{(0)}_{j,2n_1+i} := \tilde{W}^{(0)}_{j,i} \oplus R^{*(b_i\oplus x_{2,i})}_{j,i}$ and $X'^{(1)}_{j,2n_1+i} := \tilde{W}^{(1)}_{j,i} \oplus R^{*(b_i\oplus x_{2,i}\oplus 1)}_{j,i}$;
      - Assert $X^{*(0)}_{j,2n_1+i} = X'^{(0)}_{j,2n_1+i}$ and $X^{*(1)}_{j,2n_1+i} = X'^{(1)}_{j,2n_1+i}$.
    - For $i \in [n_1]$:
      - Find $r^{(1)}_{j,i}$ such that $Z_{j,n_1+i} = X^{*\left(r^{(1)}_{j,i}\right)}_{j,n_1+i}$ and $r^{(2)}_{j,i}$ such that $\ddot{Z}_{j,n_1+i} = \ddot{X}^{\left(r^{(2)}_{j,i}\right)}_{j,n_1+i}$;
      - Assert $r^{(1)}_{j,i} = r^{(2)}_{j,i}$;
  - Send $\left\{\ddot{k}_j\right\}_{j\in[p(\lambda)]\setminus\mathcal{I}}$, $\{\hat{\omega}_j\}_{j\in[p(\lambda)]\setminus\mathcal{I}}$ and $\{\bar{\omega}_j\}_{j\in[p(\lambda)]\setminus\mathcal{I}}$ to P_1;
- Upon receiving $\left\{\ddot{k}_j\right\}_{j\in[p(\lambda)]\setminus\mathcal{I}}$, $\{\hat{\omega}_j\}_{j\in[p(\lambda)]\setminus\mathcal{I}}$ and $\{\bar{\omega}_j\}_{j\in[p(\lambda)]\setminus\mathcal{I}}$ from P_2, the party P_1:
  - For $j \in [p(\lambda)] \setminus \mathcal{I}$:
    - Assert $\mathsf{hash}(\hat{\omega}_j) = \hat{\sigma}_j$ and $\mathsf{hash}(\bar{\omega}_j) = \bar{\sigma}_j$;
    - Generate $(\ddot{F}^*_j, \ddot{e}^*_j, \ddot{d}^*_j) \leftarrow \mathsf{GC.Gb}(1^\lambda, f_2; \ddot{k}_j)$, and parse $\ddot{e}^*_j = \{\ddot{X}^{*(0)}_{j,i}, \ddot{X}^{*(1)}_{j,i}\}_{i\in[2n_1]}$;
    - Assert $\ddot{F}^*_j = \ddot{F}_j$ and $\ddot{d}^*_j = \ddot{d}_j$;
    - For $i \in [n_1]$:
      - Generate $\hat{R}^{*(0)}_{j,i} \leftarrow \mathsf{PRF}_{\hat{\omega}_j}(0)$ and $\hat{R}^{*(1)}_{j,i} \leftarrow \mathsf{PRF}_{\hat{\omega}_j}(1)$;
      - Generate $\bar{R}^{*(0)}_{j,i} \leftarrow \mathsf{PRF}_{\bar{\omega}_j}(0)$ and $\bar{R}^{*(1)}_{j,i} \leftarrow \mathsf{PRF}_{\bar{\omega}_j}(1)$;
      - Compute $\ddot{X}'^{(0)}_{j,n_1+i} := \hat{W}^{(0)}_{j,i} \oplus \hat{R}^{*(\hat{b}_i\oplus r_{j,i})}_{j,i}$ and $\ddot{X}'^{(1)}_{j,n_1+i} := \hat{W}^{(1)}_{j,i} \oplus \hat{R}^{*(\hat{b}_i\oplus r_{j,i}\oplus 1)}_{j,i}$;
      - Compute $\ddot{X}'^{(0)}_{j,i} := \bar{W}^{(0)}_{j,i} \oplus \bar{R}^{*(\bar{b}_i\oplus x_{1,i})}_{j,i}$ and $\ddot{X}'^{(1)}_{j,i} := \bar{W}^{(1)}_{j,i} \oplus \bar{R}^{*(\bar{b}_i\oplus x_{1,i}\oplus 1)}_{j,i}$;
      - Assert $\ddot{X}^{*(0)}_{j,i} = \ddot{X}'^{(0)}_{j,i}$, $\ddot{X}^{*(1)}_{j,i} = \ddot{X}'^{(1)}_{j,i}$, $\ddot{X}^{*(0)}_{j,n_1+i} = \ddot{X}'^{(0)}_{j,n_1+i}$ and $\ddot{X}^{*(1)}_{j,n_1+i} = \ddot{X}'^{(1)}_{j,n_1+i}$.
  - Send decommitments $\{\mathfrak{D}_j\}_{j\in[p(\lambda)]\setminus\mathcal{I}}$ and $\left\{Z_{j,i} := X^{(x_{1,i})}_{j,i}\right\}_{i\in[n_1],j\in[p(\lambda)]\setminus\mathcal{I}}$ to P_2;
- Upon receiving decommitments $\{\mathfrak{D}_j\}_{j\in[p(\lambda)]\setminus\mathcal{I}}$ and $\{Z_{j,i}\}_{i\in[n_1],j\in[p(\lambda)]\setminus\mathcal{I}}$, the party P_2:
  - For $j \in [p(\lambda)] \setminus \mathcal{I}$:
    - Evaluate $Y_j \leftarrow \mathsf{GC.Ev}(F_j, (Z_{j,1}, \ldots, Z_{j,2n_1+n_2}))$;
    - Decode $y_j \leftarrow \mathsf{GC.De}(d_j, Y_j)$;
    - Parse $y_j = (y_{j,1}, y_{j,2}) = (f(x_1, x_2), x_1 \oplus r_j)$;
    - Check if P_1's commitment corresponding to labels of value $x_1 \oplus r_j = y_{j,2}$.
  - Record the majority of $\{y_{j,1}\}_{j\notin\mathcal{I}}$ as y_{out}, and return $(\textsc{Compute}, \mathsf{sid}, y_{\mathsf{out}})$ to the environment $\mathcal{Z}$;

Figure 7: The malicious setting protocol $\Pi^{\mathsf{m}}_{\mathsf{2pc}}$ in the $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$-hybrid model (Part II)

circuits and XOR-gadgets, and asserts P_1 uses same r_j in the j-th garbled circuit and the j-th XOR-gadget. If the check passes, P_2 sends the seeds of XOR-gadgets and the corresponding seeds of ROT's and SCROT's for the evaluation circuits to P_1.

After receiving the seeds, P_1 first computes the hash of the ROT seeds and the SCROT seeds, and it asserts these values equals to those values it obtains from $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$. P_1 then generates the evaluation XOR-gadgets along with the related SCROT's and ROT's, and it checks if P_2 honestly use the random OTs to transfer the wire labels of XOR-gadgets. If all the checks pass, P_1 decommits to the results of the evaluation XOR-gadgets and sends its wire labels in the evaluation garbled circuits. P_2 then evaluates all the evaluation garbled circuits. Finally, P_2 checks if the value of $r_j \oplus x_1$ from the outputs of garbled circuits are identical to the outputs from the XOR-gadgets. If the check passes, P_2 assumes P_1's input is consistent in most garbled circuits, and it records the majority outputs of the evaluation garbled circuits as the final protocol result.

5. Security

In practice, we assume the hardware manufacturer will not collude with the MPC players; otherwise, when P_1 (or P_2) is colluding with $\mathcal{F}_{\mathrm{HW}}$, no input privacy can be guaranteed. We first examine why our protocols are secure at the high level, and then formally state the security of our semi-honest setting

protocol $\Pi^{\mathsf{s}}_{\mathsf{2pc}}$ and malicious setting protocol $\Pi^{\mathsf{m}}_{\mathsf{2pc}}$ in Thm. 1 and Thm. 2, respectively, where we restrict the adversary $\mathcal{A}$ to only corrupt either the semi-trusted hardware functionality $\mathcal{F}_{\text{HW}}$ or the player(s) P_1 (and/or P_2).

In the semi-honest setting, the view of $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{s}}]$ is the MPC function f and some public parameters, which is already known to the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$; therefore, no additional information would be leaked through $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{s}}]$ to the adversary $\mathcal{A}$. Since $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{s}}]$ could only be passively corrupted, the correctness of GC tables and ROT copies are preserved. The input privacy of protocol $\Pi^{\mathsf{s}}_{\mathsf{2pc}}$ is guaranteed by the simulatable privacy property of the underlying GC garbling scheme.

In the malicious setting, $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$, P_1, and P_2 may be maliciously corrupted. The main design principle is as follows. In P_1's point of view, either $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$ or P_2 could be corrupted. Similarly, in P_2's point of view, either $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$ or P_1 could be corrupted. Note that our protocol does not provide accountability, i.e., when the protocol abort, we are not required to identify which party is guilty. Therefore, P_1 (or P_2) could treat the messages received from $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$ as if they are generated and delivered by P_2 (or P_1).

$\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$ combines (a) the *XOR-gadget* technique [10] to ensure the input consistency of P_1, (b) single-choice OT (SCROT) to ensure the choice bit of P_2 for the same input bit is consistent among all the garbled circuits, and (c) the cut-and-choose OT technique [11] to prevent selective failure attacks.

In total, $p_1(\lambda)$ copies of GC are generated. Currently, we consider the majority of the evaluation result as the final result. A malicious $\mathcal{F}_{\text{HW}}$ can only influence the evaluation result by generating incorrect garbled tables and incorrect ROT copies. It can success, i.e., influences the result, only when the check to the check circuits passes and the majority of the evaluation circuits are incorrect. According to the computation in the work [11], its success probability is about $2^{-0.311\lambda}$. In other words, we need to use $p_1(\lambda) \approx 132$ to achieve a 40-bit security guarantee.

Remark. There are other better cut-and-choose techniques can be found in the literature, but they are not compatible with our protocols. For instance, Lindell [18] and Huang *et al.* [19] propose protocols achieving a cheating probability of $2^{-\lambda}$. The Lindell's work [18] introduces a new technique such that, if the party P_1 is caught cheating, P_2 is able to learn P_1's input and computes the function by itself. However, in our design principle, $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$ should not learn P_1's input, and thus this technique is not suitable for our protocols. The work of Huang *et al.* [19] uses symmetric cut-and-choose where both parties need to generate garbled circuits and check the garbled circuits generated by the other party, and the final output is determined jointly. In our work, we use the malicious $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$ to generate the garbled circuits for both parties, and $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$ can easily manipulate the final result by providing same incorrect garbled circuits to both parties.

Theorem 1. *If* $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ *is a secure PRF function, and* $\mathsf{GC} := (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *is a secure simulatable private garbling scheme, protocol* $\Pi^{\mathsf{s}}_{\mathsf{2pc}}$ *described in Fig. 5 UC-realizes* $\mathcal{F}^f_{\mathsf{2pc}}$ *as described in Fig. 1 in the* $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{s}}]$*-hybrid model against any PPT semi-honest adversaries who can corrupt either* $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{s}}]$ *or the player(s)* P_1 *(and/or* P_2*) with static corruption.*

The proof is provided in App. A.

Theorem 2. *If* $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ *is a secure PRF function, and* $\mathsf{GC} := (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *is a secure simulatable private garbling scheme, protocol* $\Pi^{\mathsf{m}}_{\mathsf{2pc}}$ *described in Fig. 6 and Fig. 7 UC-realizes* $\mathcal{F}^f_{\mathsf{2pc}}$ *as described in Fig. 1 in the* $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$*-hybrid model against any PPT malicious adversaries who can corrupt either* $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$ *or the player(s)* P_1 *(and/or* P_2*) with static corruption.*

The proof is provided in App. B.

TABLE 3: Performance of our malicious setting 2PC protocol with 40-bit security guarantee (Result obtained from SGX-enabled Dell OptiPlex 7080 (Intel Core 8700 CPU @ 3.20 GHz, 32 GB RAM, OS: Ubuntu 18.04 LTS). The LAN setting: 1Gbps bandwidth, 1ms delay; the WAN setting: 100Mbps bandwidth, 25ms delay.

Circuit	Network setting	Running time (in ms)	
		Garbler	Evaluator
AES-non-expanded	LAN	133.410	135.101
	WAN	796.561	805.641
SHA-1	LAN	443.719	451.660
	WAN	1404.330	1428.117
SHA-256	LAN	844.175	861.919
	WAN	1850.553	1890.025

6. Implementation and Benchmarks

Our protocol is implemented in C++ using Intel SGX SDK on Windows. We use AES-NI for the PRF algorithm. To efficiently generate ROT's in the SGX enclave, we carefully analyze the performance bottleneck and notice that if we just generate one copy of ROT at once, then the Receiver needs to enter the enclave for many times, and the enter/exit may cause performance loss. Because this, we group ROT's into batches, and choose the optimal batch size according to the test result.

We already explained our choice of GC optimizations in Sec. 2, and here we provide more details. Denote the seed of the garbled circuit as k, to generate the wire labels, we first compute the $\mathsf{PRF}_k(0)$ and force it's least significant bit to be 1, and the result is the Δ in the free-XOR optimization. Subsequently, we invoke the PRF for n times in the form $\mathsf{PRF}_k(i)$ to generate the 0-label of the i-th input, then we compute $\Delta \oplus \mathsf{PRF}_k(i)$ to get the 1-label of the i-th input. After obtaining all the wire labels, we computes $k' := \mathsf{PRF}_k(n+1)$ as the seed for generation of garbled tables.

With regard to the generation of the garbled circuits, we assume the order of the gates in the circuit description is layer-designed such that, for a gate to be garbled, it's input wire won't be the output wire of a gate that hasn't been garbled. Hence, we can garble the gates as this order. For a XOR gate, since free-XOR is used, its garbled tables is eliminated, and we simply XOR the two input wires' 0-label to obtain the 0-label of the output wire. For each non-XOR gate, we invoke the hash algorithm for 4 times for different input value combinations

to generate 4 ciphertexts. After that, we can determine each ciphertext's place in the garbled table according to the select bit, i.e., the least significant bit of the wire labels, as desribed in the point-and-permute optimization. Denote the first input wire label's select bit as s_a and the second input wire label's select bit as s_b, the ciphertext derived from these two wire labels will be placed in the $(s_a + 2 * s_b + 1)$-th row of the garbled table. Furthermore, since we adopt the GRR3 optimization, we set the value of the first row as the output wire's 0-label, and XOR each row with it, then the first row becomes an all 0 string and thus can be eliminated.

The evaluation process use the same seed as the garbling process, since the SGX enclave runs on the Evaluator's machine, this seed can be locally transferred. The evaluation order is also as in the circuit description. For each XOR gate, the Receiver only has two wire labels and we XOR these two label to obtain the result. For each non-XOR gate, we use the input wire labels to compute 1 ciphertext using PRF and determine its place according to the input wire labels' select bits.

We perform the experiments of our protocol on an SGX-enabled Dell OptiPlex 7080 equipped with an Intel Core 8700 CPU @ 3.20 GHz with 32.0 GB RAM. Our protocols are running on Windows 10 2004, and the EMP protocols are running on Ubuntu 18.04 LTS.

We evaluate all protocols in two simulated network settings: (i) a LAN setting with 1Gbps bandwidth and 0.1ms delay and (ii) a WAN setting with 100Mbps bandwidth and 25ms delay.

To test the performance of our ROT generation protocol, we compared our protocol with the implementation in EMP-ROT [9]. Table. 1 shows the performance comparison for generating 10^4 to 10^8 copies of ROT, where the result is the average of 10 tests.

To test the performance of the semi-honest 2PC protocols, our benchmarks use three Bristol circuits, which consists of only AND gates, XOR gates and inverters, respectively are AES-non-expanded circuit, SHA-1 circuit and SHA-256 circuit. The AES-non-expanded circuit contains 33872 wires and 33616 gates, including 6800 AND gates; in this circuit, the party P_1's input size, the party P_2's input size and the output size are all 128 bits. The SHA-1 circuit contains 107113 wires and 106601 gates, including 37300 AND gates; in this circuit, the party P_1's input size is 512 bits while the party P_2 doesn't have a input, and the output size is 160 bits. The SHA-256 circuit contains 236624 wires and 236112 gates, including 90825 AND gates; in this circuit, the party P_1's input size is 512 bits while the party P_2 doesn't have a input, and the output size is 256 bits. For the semi-honest setting protocol, we compared our protocol with EMP-SH2PC [9].

Table. 2 shows the performance comparison for evaluating the aforementioned benchmark circuits for 1000 times, and the results are the average of 10 tests. We measure the garbling time, transmission time, and the evaluation time separately. For our protocol, since the garbling procedure is executed in the SGX enclave at the evaluator side, we present the evaluator running time of our protocol in two parts: (i) the SGX running time and (ii) normal mode CPU running time.

Table. 3 shows the performance of our malicious setting 2PC protocol with 40-bit security guarantee, i.e., $p_1(\lambda) = 132$ copies of GC tables are generated for cut-and-choose. It takes 135 ms to securely evaluate the AES-non-expanded circuit once in the LAN setting, and 805 ms in the WAN setting. The numbers for SHA-1 and SHA-256 are 451 ms and 861 ms in the LAN setting, respectively.

7. Related Work

Felsen *et al.* [20] proposed an Intel SGX-based secure function evaluation (SFE) approach in which private inputs are sent to enclave. In their protocol, only the inputs and the outputs need to be transferred, the communication complexity of their protocol is optimal up to an additive constant. They evaluate the Boolean circuit representation of the function in enclave to provide security with regards to software side-channel attacks. In addition, they reduce the problem of private function evaluation (PFE) to the problem of SFE by using universal circuits and are the first to address PFE problem via TEEs. They give a prototype implementation of their protocol and compare its performance with state-of-art implementations of Yao's GC and the GMW protocols, the result shows their protocol's efficiency.

In our work, we also use Intel SGX to accelerate computation. However, we keep the enclave away from private inputs to guarantee privacy even when the enclave is corrupted. To use a enclave that does not know private inputs, we propose a modified edition of Yao's GC protocol. The hardware component only generates the ROT copies and garbled tables in the pre-computation process, and all the processes related to the private input is executed outside the enclave

Bahmani *et al.* [21] proposed an intuitive approach in which the program in an isolated execution environment (IEE) plays the role of a trusted third party and the major part of computational load is left to the untrusted machine. In this way, they reach a minimum communication complexity that only depends on number of inputs and outputs. Obviously, the trust to IEE and hardware manufacturer is crucial. They introduced a novel notion of labelled attested computation (LAC) and give a LAC-based solution with rigorous security guarantees. They implement Intel SGX-based version of their protocol and compare it with the ABY framework, and their solution is hundreds of times faster than ABY.

Gupta *et al.* [22] proposed a protocol using Intel SGX for SFE problem which is secure in the semi-honest model, they also show how to improve their protocol's security. They notice the problem that the developers need to trust hardware and hard supplier when using Intel SGX, but don't propose a feasible solution. Because SGX-enabled CPU was not available at the time when they finished their research, they did not give the implementation of their protocol. However, they estimate the amounts of cryptographic operations that their protocol and Yao's garbled circuit protocol need, and expect a huge improvement of performance.

In our work, we focus on the problem that the hardware component can also be corrupted, and propose a semi-honest setting protocol and a malicious setting protocol. In these protocols, we isolate the hardware component from the private inputs, and only use the hardware to execute some pre-computation processes.

Choi *et al.* [23] consider the possibility of SGX being compromised and want to protect the most sensitive data in any case. They propose a hybrid SFE-SGX protocol which consists of calculation in SGX enclave and standard cryptographic techniques. The function to be evaluated is partitioned into several round functions, the odd rounds are executed in enclave and the even rounds are done using a scheme based on garbled circuit. They claim that, if the partition scheme is proper, which means no private inputs is leaked by intermediate values, their hybrid approach ensures security against semi-hones adversary. They also notice that there are numerous side-channel attacks against SGX that can extract information from enclave, so they deploy corresponding mitigation techniques to protect privacy. They present how to use this hybrid protocol to solve privacy-preserving retrieval and privacy-preserving navigation, and the evaluation shows that the hybrid protocol achieves up to 38 times of performance improvement over the pure garbled circuit protocol.

In our work, the computation is also done both in the enclave and out of the enclave. In Choi's work, the enclave gets part of the private input, while we ensure the enclave is isolated with any private data and only some pre-computation work need to be done in enclave, which guarantees privacy even if the enclave is compromised. Choi's work is based on the assumption that the partition of function is properly done, however, partitioning itself is a hard problem which requires careful consideration. While our work is based on the garbled circuit protocol, and we don't rely on this additional requirement.

Chakraborty *et al.* [24] use the trusted hardware to enable intellectual property protection. Only users who possesses authorized hardware can use the deep learning model to predict. For the unauthorized users, the model accuracy will greatly decrease such that they can hardly obtain an accurate result.

For the cut-and-choose technique, Lindell and Pinkas [11] notice that the selective failure attack is possible because only garbled circuits are checked. They introduced a new primitive called cut-and-choose oblivious transfer, which allows the GC evaluator to simultaneously check garbled circuits and oblivious transfers. Furthermore, they augment their cut-and-choose OT protocol to a single-choice cut-and-choose OT protocol, which ensures the input consistency of the GC evaluator. The GC generator uses zero-knowledge proof to prove that its inputs are consistent. They use majority voting to decide the output, resulting in an error probability of $2^{-0.311\lambda}$. In other words, to achieve an error probability of 2^{-40}, about 132 garbled circuits should be used.

In our work, we borrow their idea of using cut-and-choose to prevent selective failure attack. The different thing is, in our protocol, the OT protocol is based on the ROT generated on a seed by the trusted hardware SGX, so we let SGX compute the hash of the seed and transfer to P_2, and P_2 can catch a cheating P1 that wants to carry out selective failure attack by verifying these hash values. The input consistency of P_2 is also guaranteed by the single-choice OT protocol.

To further improve efficiency, Lindell [18] proposed another protocol based on the primitive cut-and-choose OT. In this protocol, a malicious adversary can only successfully cheat with a probability of probability of $2^{-\lambda}$, in comparison with the previous protocol, the number of garbled circuits needed to achieve a cheating probability of 2^{-40} is reduced from 132 to 40. In addition to the protocol based on cut-and-choose, he adds another secure evaluation which enables the GC evaluator to obtain the input of a cheat GC garbler who uses inconsistent inputs. Then, the GC evaluator can compute the result by itself. The output computed by the GC evaluator and the output computed by the GC protocol seems no difference to the GC garbler, therefore, a malicious GC generator can cheat only when all the check circuits are correct and all the evaluation circuits are incorrect, resulting in a cheating probability of $2^{-\lambda}$.

In the same period, Huang *et al.* [19] also proposed a protocol based on cut-and-choose that achieves a cheating probability of $2^{-\lambda}$. They design the protocol in a symmetric setting, where both parties generate garbled circuits and evaluate those generated by the other party. In their protocol, a party outputs value $\mathbf{v}$ for an output wire if and only if for each party, at least one of the evaluation circuits is evaluated to $\mathbf{v}$ on that wire. Since an honest party always generate correct garbled circuits, the correctness won't be broken down unless all the check circuits generated by the other party are correct and all the evaluation circuits generated by the other party are incorrect, so a cheating probability of $2^{-\lambda}$ is achieved.

In our work, however, we find it difficult to further reduce the cheat probability and improve the efficiency as described in Sec. 5, and we leave this possible improvement as future work.

8. Conclusion

In this work, we investigate the problem where the trusted hardware manufacturer are not fully trusted, and the hardware components may leak sensitive information to the remote servers through backdoors, side-channels, steganography, and kleptography, etc. We first present a new security notion called semi-trusted hardware model, where the adversary is allowed to passively and/or maliciously corrupt the hardware component. We then propose a new type of two-party computation (2PC) protocols called *silent MPC* that uses semi-trusted hardware to significantly reduce the communication between the 2PC players. Our constructs use semi-trusted hardware for pre-computation that does not depend on the actual protocol inputs; thus no sensitive information is leaked to the hardware components. We implemented our protocols and compared it with the EMP-toolkit. When the semi-trusted hardware is instantiated by Intel SGX, our ROT protocol is several magnitude times faster than than the EMP-ROT, and our semi-honest 2PC protocol is also significantly faster than the EMP-SH2PC in both LAN and WAN setting. We emphasize this line of research is far from being completed, and we will generalize our technique to multi-party computation scenarios as the future work.

References

[1] G. Dan and S. Jim, "More than 20gb of intel source code and proprietary data dumped online," [EB/OL], https://arstechnica.com/information-technology/2020/08/intel-is-investigating-the-leak-of-20gb-of-its-source-code-and-private-data/ Accessed August 30, 2020.

[2] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, "Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx," in *CCS'17*, 2017, pp. 2421–2434.

[3] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure:SGX cache attacks are practical," in *WOOT'17*, 2017.

[4] J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx, "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution," in *USENIX Security'17*, 2017, pp. 1041–1056.

[5] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution," in *USENIX Security'18*, 2018, pp. 991–1008.

[6] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T.-H. Lai, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," *IEEE Security & Privacy*, vol. 18, no. 3, pp. 28–37, 2020.

[7] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen, "Intel® software guard extensions: Epid provisioning and attestation services," *White Paper*, vol. 1, no. 1-10, p. 119, 2016.

[8] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *FOCS'01*. IEEE, 2001, pp. 136–145.

[9] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient MultiParty computation toolkit," https://github.com/emp-toolkit, 2016.

[10] P. Mohassel and B. Riva, "Garbled circuits checking garbled circuits: More efficient and secure two-party computation," in *CRYPTO'13*. Springer, 2013, pp. 36–53.

[11] Y. Lindell and B. Pinkas, "Secure two-party computation via cut-and-choose oblivious transfer," *Journal of cryptology*, vol. 25, no. 4, pp. 680–722, 2012.

[12] M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," in *CCS'12*, 2012, pp. 784–796.

[13] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols," in *STOC'90*, 1990, pp. 503–513.

[14] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," in *EC'99*, 1999, pp. 129–139.

[15] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *ICALP'08*. Springer, 2008, pp. 486–498.

[16] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *ASIACRYPT'09*. Springer, 2009, pp. 250–267.

[17] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole," in *EUROCRYPT'15*. Springer, 2015, pp. 220–250.

[18] Y. Lindell, "Fast cut-and-choose-based protocols for malicious and covert adversaries," *Journal of Cryptology*, vol. 29, no. 2, pp. 456–490, 2016.

[19] Y. Huang, J. Katz, and D. Evans, "Efficient secure two-party computation using symmetric cut-and-choose," in *CRYPTO'13*. Springer, 2013, pp. 18–35.

[20] S. Felsen, Á. Kiss, T. Schneider, and C. Weinert, "Secure and private function evaluation with intel sgx," in *CCSW@CCS'19*, 2019, pp. 165–181.

[21] R. Bahmani, M. Barbosa, F. Brasser, B. Portela, A.-R. Sadeghi, G. Scerri, and B. Warinschi, "Secure multiparty computation from sgx," in *FC'17*. Springer, 2017, pp. 477–497.

[22] D. Gupta, B. Mood, J. Feigenbaum, K. Butler, and P. Traynor, "Using intel software guard extensions for efficient two-party secure function evaluation," in *FC'16*. Springer, 2016, pp. 302–318.

[23] J. I. Choi, D. Tian, G. Hernandez, C. Patton, B. Mood, T. Shrimpton, K. R. Butler, and P. Traynor, "A hybrid approach to secure function evaluation using sgx," in *AsiaCCS'19*, 2019, pp. 100–113.

[24] A. Chakraborty, A. Mondal, and A. Srivastava, "Hardware-assisted intellectual property protection of deep learning models," in *DAC'20*, 2020.

Appendix

1. Proof of Theorem. 1

Theorem 1. *If* $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ *is a secure PRF function, and* $\mathsf{GC} := (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *is a secure simulatable private garbling scheme, protocol* $\Pi^{\mathsf{s}}_{\mathsf{2pc}}$ *described in Fig. 5 UC-realizes* $\mathcal{F}^f_{\mathsf{2pc}}$ *as described in Fig. 1 in the* $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$*-hybrid model against any PPT semi-honest adversaries who can corrupt either* $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ *or the player(s)* P_1 *(and/or* P_2*) with static corruption.*

Proof. To prove Thm. 1, we construct a simulator $\mathcal{S}$ such that no non-uniform PPT environment $\mathcal{Z}$ can distinguish between (i) the real execution $\mathrm{EXEC}^{\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]}_{\Pi^{\mathsf{s}}_{\mathsf{2pc}},\mathcal{A},\mathcal{Z}}$ where the parties $\mathcal{P} := \{P_1, P_2\}$ run protocol $\Pi^{\mathsf{s}}_{\mathsf{2pc}}$ in the $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$-hybrid model and the corrupted parties are controlled by a dummy adversary $\mathcal{A}$ who simply forwards messages from/to $\mathcal{Z}$, and (ii) the ideal execution $\mathrm{EXEC}_{\mathcal{F}^f_{\mathsf{2pc}},\mathcal{S},\mathcal{Z}}$ where the parties P_1 and P_2 interact with functionality $\mathcal{F}^f_{\mathsf{2pc}}$ in the ideal world, and corrupted parties are controlled by the simulator $\mathcal{S}$. We consider following cases.

Case 1: $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ is corrupted; P_1 and P_2 are honest.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. $\mathcal{S}$ simulates the interface of $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ as well as honest parties P_1 and P_2. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- Upon receiving $(\textsc{ComputeNotify}, \mathsf{sid}, |x_i|, P_i)$ for an honest party P_i from the external $\mathcal{F}^f_{\mathsf{2pc}}$, the simulator $\mathcal{S}$ sends $(\mathsf{Run}, \mathsf{sid}, \langle \mathrm{ROT}, n_2 \rangle)$ to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ on behave of P_i.
- Upon receiving $(\mathsf{Run}, \mathsf{sid}, Q_i)$ from the party $P_i \in \mathcal{P}$ via the interface of $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$, $\mathcal{S}$ acts as $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ to send $(\textsc{RunNotify}, \mathsf{sid}, Q_i, P_i)$ to $\mathcal{A}$. $\mathcal{S}$ then simulates the $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ functionality as defined.
- When the simulated party P_2 receive $\{\langle b_i, R_i^{(b_i)} \rangle\}_{i\in[n_2]}$ from $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$, $\mathcal{S}$ acts as P_2 to send $\{c_i := b_i\}_{i\in[n_2]}$ to the simulated party P_1.
  $\mathcal{S}$ then acts as both P_1 and P_2 to send $(\mathsf{Run}, \mathsf{sid}, \langle \mathrm{GC}, f \rangle)$ to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$. It then simulates the rest communication between P_1 and P_2 according to the protocol description as if both P_1 and P_2 receive $(\textsc{Compute}, \mathsf{sid}, 0)$ from $\mathcal{Z}$.

Indistinguishability. Assume the communication between P_1 and P_2 is via the secure channel functionality $\mathcal{F}_{\mathrm{SC}}$, the views of $\mathcal{A}$ and $\mathcal{Z}$ in $\mathrm{EXEC}^{\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]}_{\Pi^{\mathsf{s}}_{\mathsf{2pc}},\mathcal{A},\mathcal{Z}}$ and $\mathrm{EXEC}_{\mathcal{F}^f_{\mathsf{2pc}},\mathcal{S},\mathcal{Z}}$ are identical. Therefore, it is perfectly indistinguishable.

Case 2: P_1 is corrupted; P_2 and $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ are honest.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. $\mathcal{S}$ simulates the interface of $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ as well as honest P_2. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- Upon receiving $(\textsc{ComputeNotify}, \mathsf{sid}, |x_2|, P_2)$ from the external $\mathcal{F}^f_{\mathsf{2pc}}$, the simulator $\mathcal{S}$ sends $(\mathsf{Run}, \mathsf{sid}, \langle \mathrm{ROT}, n_2 \rangle)$ to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ on behave of P_2.
- When the simulated party P_2 receive $\{\langle b_i, R_i^{(b_i)} \rangle\}_{i\in[n_2]}$ from $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$, $\mathcal{S}$ acts as P_2 to send $\{c_i := b_i\}_{i\in[n_2]}$ to the simulated party P_1. $\mathcal{S}$ then acts as P_2 to send $(\mathsf{Run}, \mathsf{sid}, \langle \mathrm{GC}, f \rangle)$ to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$.
- When P_2 receives $\{Z_i\}_{i\in[n_1]}$ and $\{W_i^{(0)}, W_i^{(1)}\}_{i\in[n_2]}$ from P_1, $\mathcal{S}$ uses the internal GC label information (F, e, d) of the simulated $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ to extract P_1's input x_1^*. It then sends $(\textsc{Compute}, \mathsf{sid}, x_1^*)$ to the external $\mathcal{F}^f_{\mathsf{2pc}}$ on behave of P_1.
- Upon receiving $(\textsc{Output}, \mathsf{sid}, P_2)$ from the external $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$, if $\mathcal{A}$ allows P_2 to finish the protocol execution and obtains y, $\mathcal{S}$ sends $(\textsc{Deliver}, \mathsf{sid}, P_2)$ to the external $\mathcal{F}^f_{\mathsf{2pc}}$.

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \ldots, \mathcal{H}_1$.

Hybrid $\mathcal{H}_0$: It is the real protocol execution $\mathrm{EXEC}^{\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]}_{\Pi^{\mathsf{s}}_{\mathsf{2pc}},\mathcal{A},\mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that in $\mathcal{H}_1$, P_2 sends $\{c'_i := b_i\}_{i\in[n_2]}$ to P_1, instead of $\{c_i := b_i \oplus x_{2,i}\}_{i\in[n_2]}$.

Claim 1. *$\mathcal{H}_1$ and $\mathcal{H}_0$ are perfectly indistinguishable.*

Proof. Since b_i are the ROT select bits randomly picked by $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$, the distribution of $\{c'_i\}_{i\in[n_2]}$ and $\{c_i\}_{i\in[n_2]}$ are identical. Therefore, $\mathcal{H}_1$ and $\mathcal{H}_0$ are perfectly indistinguishable. □

The adversary's view of $\mathcal{H}_1$ is identical to the simulated view $\mathrm{EXEC}_{\mathcal{F}^f_{\mathsf{2pc}},\mathcal{S},\mathcal{Z}}$. Therefore, it is perfectly indistinguishable.

Case 3: P_2 is corrupted; P_1 and $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ are honest.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. $\mathcal{S}$ simulates the interface of $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ as well as honest P_1. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- Upon receiving $(\textsc{ComputeNotify}, \mathsf{sid}, |x_1|, P_1)$ from the external $\mathcal{F}^f_{\mathsf{2pc}}$, the simulator $\mathcal{S}$ sends $(\mathsf{Run}, \mathsf{sid}, \langle \mathrm{ROT}, n_2 \rangle)$ to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ on behave of P_1.
- When P_1 receives $\{c_i\}_{i\in[n_2]}$ from P_2, $\mathcal{S}$ uses the internal GC label information (F, e, d) of the simulated $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ to extract P_2's input x_2^*. It then sends $(\textsc{Compute}, \mathsf{sid}, x_2^*)$ to the external $\mathcal{F}^f_{\mathsf{2pc}}$ on behave of P_2.
- Upon receiving $(\textsc{Compute}, \mathsf{sid}, y)$ from the external $\mathcal{F}^f_{\mathsf{2pc}}$ for P_2, the simulator $\mathcal{S}$ uses the GC simulator to generate $(F', X', d') \leftarrow \mathsf{Sim}(1^\lambda, y, \Phi(f))$.
- Upon receiving $(\mathsf{Run}, \mathsf{sid}, \langle \mathrm{GC}, f \rangle)$ from both parties P_1 and P_2 to $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$, $\mathcal{S}$ sends (F', d') as the GC tables and decode information to P_2 on behave of $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$.
  $\mathcal{S}$ then uses X' as the wire labels to generate $\{Z_i\}_{i\in[n_1]}$ and $\{W_i^{(0)}, W_i^{(1)}\}_{i\in[n_2]}$ as follows:
  1. For $i \in [n_1]$, set $Z_i := X'_i$;
  2. For $i \in [n_2]$: set $W_i^{(x_{2,i})} := X'_{n_1+i} \oplus R_i^{(b_i)}$ and $W_i^{(x_{2,i}\oplus 1)} := R_i^{(b_i\oplus 1)}$;
  $\mathcal{S}$ then acts as P_1 to send those messages to P_2.

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \ldots, \mathcal{H}_2$.

Hybrid $\mathcal{H}_0$: It is the real protocol execution $\mathrm{EXEC}^{\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]}_{\Pi^{\mathsf{s}}_{\mathsf{2pc}},\mathcal{A},\mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that $\mathcal{H}_1$ uses true random numbers $R_i^{(0)}, R_i^{(1)} \leftarrow \{0,1\}^\lambda$ instead of $R_i^{(b)} \leftarrow \mathsf{PRF}_\omega(i,b)$, $b \in \{0,1\}$.

Claim 2. *$\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ is a secure PRF function with adversarial distinguishing advantage $\mathsf{Adv}_{\mathsf{PRF}}(\mathcal{A},\lambda)$, then $\mathcal{H}_1$ and $\mathcal{H}_0$ are indistinguishable with distinguishing advantage $n_2 \cdot \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{A},\lambda)$.*

Proof. It is a straightforward reduction; namely, by hybrid argument, we can show that if there exists an adversary $\mathcal{A}$ who can distinguish $\mathcal{H}_1$ from $\mathcal{H}_0$, then we can construct an adversary $\mathcal{B}$ who can break the PRF. □

Hybrid $\mathcal{H}_2$: $\mathcal{H}_2$ is the same as $\mathcal{H}_1$ except that $\mathcal{H}_2$ generates $(F', X', d') \leftarrow \mathsf{Sim}(1^\lambda, y, \Phi(f))$, and then it uses X' as the wire labels to generate $\{Z_i\}_{i\in[n_1]}$ and $\{W_i^{(x_{2,i})}\}_{i\in[n_2]}$. $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ also sends (F', d') as the GC tables and decoding information to P_2.

Claim 3. *If GC is simulatable private with adversarial distinguishing advantage $\mathsf{Adv}_{\mathsf{GC}}^{\mathsf{prv.sim},\Phi,\mathsf{Sim}}(\mathcal{A},\lambda)$, then $\mathcal{H}_2$ and $\mathcal{H}_1$ are indistinguishable with distinguishing advantage $\mathsf{Adv}_{\mathsf{GC}}^{\mathsf{prv.sim},\Phi,\mathsf{Sim}}(\mathcal{A},\lambda)$.*

Proof. First of all, by the requirement of simulatable privacy in Def. 2, $(F', X', d') \leftarrow \mathsf{Sim}(1^\lambda, y, \Phi(f))$ should be indistinguishable from the real one. Moreover, since P_2 does not know $R_i^{(b_i \oplus 1)}$, if there is an adversary $\mathcal{A}$ who can distinguish the distribution of $\{W_i^{(0)}, W_i^{(1)}\}_{i\in[n_2]}$ from the real one with probability ε, then we can construct an adversary $\mathcal{B}$ who has the same distinguishing advantage $\mathsf{Adv}_{\mathsf{GC}}^{\mathsf{prv.sim},\Phi,\mathsf{Sim}}(\mathcal{B},\lambda) = \varepsilon$. □

The adversary's view of $\mathcal{H}_2$ is identical to the simulated view $\mathrm{EXEC}_{\mathcal{F}_{\mathsf{2pc}}^f,\mathcal{S},\mathcal{Z}}$. Therefore, if GC is simulatable private, the views of $\mathcal{A}$ and $\mathcal{Z}$ in $\mathrm{EXEC}^{\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]}_{\Pi^{\mathsf{s}}_{\mathsf{2pc}},\mathcal{A},\mathcal{Z}}$ and $\mathrm{EXEC}_{\mathcal{F}_{\mathsf{2pc}}^f,\mathcal{S},\mathcal{Z}}$ are indistinguishable with distinguishing advantage

$$n_2 \cdot \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{A},\lambda) + \mathsf{Adv}_{\mathsf{GC}}^{\mathsf{prv.sim},\Phi,\mathsf{Sim}}(\mathcal{A},\lambda) = \mathsf{negl}(\lambda) \ .$$

Case 4: P_1 and P_2 are corrupted; $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$ is honest.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. The simulator $\mathcal{S}$ simulates the functionality $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{s}}]$.

Indistinguishability. This is a trivial case. Since both P_1 and P_2 are controlled by the adversary $\mathcal{A}$, no message is simulated by $\mathcal{S}$.

This concludes the proof. □

2. Proof of Theorem. 2

Theorem 2. *If $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ is a secure PRF function, and $\mathsf{GC} := (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ is a secure simulatable private garbling scheme, protocol $\Pi^{\mathsf{m}}_{\mathsf{2pc}}$ described in Fig. 6 and Fig. 7 UC-realizes $\mathcal{F}_{\mathsf{2pc}}^f$ as described in Fig. 1 in the $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$-hybrid model against any PPT malicious adversaries who can corrupt either $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ or the player(s) P_1 (and/or P_2) with static corruption.*

Proof. To prove Thm. 2, we construct a simulator $\mathcal{S}$ such that no non-uniform PPT environment $\mathcal{Z}$ can distinguish between (i) the real execution $\mathrm{EXEC}^{\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]}_{\Pi^{\mathsf{m}}_{\mathsf{2pc}},\mathcal{A},\mathcal{Z}}$ where the parties $\mathcal{P} := \{P_1, P_2\}$ run protocol $\Pi^{\mathsf{m}}_{\mathsf{2pc}}$ in the $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$-hybrid model and the corrupted parties are controlled by a dummy adversary $\mathcal{A}$ who simply forwards messages from/to $\mathcal{Z}$, and (ii) the ideal execution $\mathrm{EXEC}_{\mathcal{F}_{\mathsf{2pc}}^f,\mathcal{S},\mathcal{Z}}$ where the parties P_1 and P_2 interact with functionality $\mathcal{F}_{\mathsf{2pc}}^f$ in the ideal world, and corrupted parties are controlled by the simulator $\mathcal{S}$. We consider following cases.

Case 1: $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ is corrupted; P_1 and P_2 are honest.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. $\mathcal{S}$ simulates the interface of $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ as well as honest parties P_1 and P_2. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- Upon receiving $(\textsc{ComputeNotify}, \mathsf{sid}, |x_i|, P_i)$ for an honest party P_i from the external $\mathcal{F}_{\mathsf{2pc}}^f$, the simulator $\mathcal{S}$ acts as P_i to perform according to the protocol description as if P_i receives $(\textsc{Compute}, \mathsf{sid}, 0)$ from $\mathcal{Z}$.
- Upon receiving $(\mathsf{Run}, \mathsf{sid}, Q_i)$ from the party $P_i \in \mathcal{P}$ via the interface of $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$, $\mathcal{S}$ acts as $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ to send $(\textsc{RunNotify}, \mathsf{sid}, Q_i, P_i)$ to $\mathcal{A}$. $\mathcal{S}$ then simulates the $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ functionality as defined.
- Upon receiving $(\textsc{Output}, \mathsf{sid}, P_2)$ from the external $\mathcal{F}_{\mathsf{2pc}}^f$, the simulator $\mathcal{S}$ returns $(\textsc{Deliver}, \mathsf{sid}, P_2)$ if the simulated P_2 does not abort and obtains the protocol output y.

Indistinguishability. Assume the communication between P_1 and P_2 is via the secure channel functionality $\mathcal{F}_{\mathrm{SC}}$, the views of $\mathcal{A}$ and $\mathcal{Z}$ in $\mathrm{EXEC}^{\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]}_{\Pi^{\mathsf{m}}_{\mathsf{2pc}},\mathcal{A},\mathcal{Z}}$ and $\mathrm{EXEC}_{\mathcal{F}_{\mathsf{2pc}}^f,\mathcal{S},\mathcal{Z}}$ are identical. Therefore, it is perfectly indistinguishable.

Case 2: P_1 is corrupted; P_2 and $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ are honest.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. $\mathcal{S}$ simulates the interface of $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ as well as honest P_2. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- Upon receiving $(\textsc{ComputeNotify}, \mathsf{sid}, |x_2|, P_2)$ from the external $\mathcal{F}_{\mathsf{2pc}}^f$, the simulator $\mathcal{S}$ acts as P_2 to perform according to the protocol description as if P_2 receives $(\textsc{Compute}, \mathsf{sid}, 0)$ from $\mathcal{Z}$.
- When the simulated party P_2 receive $\{\bar{c}_i := \bar{b}_i \oplus x^*_{1,i}\}_{i\in[n_1]}$ from P_1, $\mathcal{S}$ uses the knowledge of $\bar{b}_i$ from the internal state of $\mathcal{F}_{\mathrm{HW}}[\mathsf{M}^{\mathsf{m}}]$ to extract x^*_1. $\mathcal{S}$ then sends $(\textsc{Compute}, \mathsf{sid}, x^*_1)$ to the external $\mathcal{F}_{\mathsf{2pc}}^f$ on behave of P_1.
- The simulator $\mathcal{S}$ aborts, if the simulated P_2 completes the protocol execution, yet the majority of the $r^{(1)}_{j,i}$ used in the evaluation set of the garbled circuits F_j are inconsistent with the $r^{(2)}_{j,i}$ used in the evaluation set of the XOR-gadget circuits $\ddot{F}_j$, $j \in [p(\lambda)] \setminus \mathcal{I}$.
- Upon receiving $(\textsc{Output}, \mathsf{sid}, P_2)$ from the external $\mathcal{F}_{\mathsf{2pc}}^f$, the simulator $\mathcal{S}$ returns $(\textsc{Deliver}, \mathsf{sid}, P_2)$ if the simulated P_2 does not abort and obtains the protocol output y.

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \ldots, \mathcal{H}_2$.

Hybrid $\mathcal{H}_0$: It is the real protocol execution $\text{EXEC}^{\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]}_{\Pi^{\mathsf{m}}_{\mathsf{2pc}},\mathcal{A},\mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that in $\mathcal{H}_1$, P_2 sends $\{\tilde{c}'_i := \tilde{b}_i\}_{i\in[n_2]}$ to P_1, instead of $\{\tilde{c}_i := \tilde{b}_i \oplus x_{2,i}\}_{i\in[n_2]}$.

Claim 4. *$\mathcal{H}_1$ and $\mathcal{H}_0$ are perfectly indistinguishable.*

Proof. Since b_i are the ROT select bits randomly picked by $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{s}}]$, the distribution of $\{\tilde{c}'_i\}_{i\in[n_2]}$ and $\{\tilde{c}_i\}_{i\in[n_2]}$ are identical. Therefore, $\mathcal{H}_1$ and $\mathcal{H}_0$ are perfectly indistinguishable. □

Hybrid $\mathcal{H}_2$: $\mathcal{H}_2$ is the same as $\mathcal{H}_1$ except that in $\mathcal{H}_2$, the simulator $\mathcal{S}$ aborts, if P_1 and P_2 completes the protocol execution, yet the majority of the $r^{(1)}_{j,i}$ used in the evaluation set of the garbled circuits F_j are inconsistent with the $r^{(2)}_{j,i}$ used in the evaluation set of the XOR-gadget circuits $\ddot{F}_j$, $j \in [p(\lambda)] \setminus \mathcal{I}$.

Claim 5. *The distinguishing advantage between $\mathcal{H}_2$ and $\mathcal{H}_1$ is less than $2^{-(p(\lambda)/4-1)}$.*

Proof. The probability that the adversary $\mathcal{A}$ can distinguish $\mathcal{H}_2$ from $\mathcal{H}_1$ is exactly the probability where all the $r^{(1)}_{j,i}$ used in the check set of the garbled circuits F_j are the same with the $r^{(2)}_{j,i}$ used in the check set of the XOR-gadget circuits $\ddot{F}_j$, $j \in \mathcal{I}$; however, the majority of the $r^{(1)}_{j,i}$ used in the evaluation set of the garbled circuits F_j are inconsistent with the $r^{(2)}_{j,i}$ used in the evaluation set of the XOR-gadget circuits $\ddot{F}_j$, $j \in [p(\lambda)] \setminus \mathcal{I}$. For every security parameter $\lambda \in \mathbb{N}$, the above probability is

$$
\begin{aligned}
p &= \sum_{i=p(\lambda)/4}^{p(\lambda)/2} \frac{\binom{p(\lambda)-i}{p(\lambda)/2}}{\binom{p(\lambda)}{p(\lambda)/2}} = \frac{1}{\binom{p(\lambda)}{p(\lambda)/2}} \cdot \sum_{i=p(\lambda)/4}^{p(\lambda)/2} \binom{p(\lambda)-i}{p(\lambda)/2} \\
&= \frac{1}{\binom{p(\lambda)}{p(\lambda)/2}} \cdot \sum_{i=0}^{3/4\cdot p(\lambda)} \binom{i}{p(\lambda)/2} = \frac{1}{\binom{p(\lambda)}{p(\lambda)/2}} \cdot \binom{3/4\cdot p(\lambda)+1}{p(\lambda)/2+1} \\
&< \frac{1}{2^{p(\lambda)/4-1}}
\end{aligned}
$$

□

The adversary's view of $\mathcal{H}_2$ is identical to the simulated view $\text{EXEC}_{\mathcal{F}^f_{\mathsf{2pc}},\mathcal{S},\mathcal{Z}}$. Therefore, the probability that the adversary can distinguish $\text{EXEC}_{\mathcal{F}^f_{\mathsf{2pc}},\mathcal{S},\mathcal{Z}}$ from $\text{EXEC}^{\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]}_{\Pi^{\mathsf{m}}_{\mathsf{2pc}},\mathcal{A},\mathcal{Z}}$ is bounded by $2^{-(p(\lambda)/4-1)}$, which is negligible in λ.

Case 3: P_2 is corrupted; P_1 and $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$ are honest.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. $\mathcal{S}$ simulates the interface of $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$ as well as honest party P_1. In addition, the simulator $\mathcal{S}$ simulates the following interactions with $\mathcal{A}$.

- Upon receiving $(\text{COMPUTENOTIFY}, \mathsf{sid}, |x_1|, P_1)$ from the external $\mathcal{F}^f_{\mathsf{2pc}}$, the simulator $\mathcal{S}$ acts as P_1 to perform according to the protocol description as if P_1 receives $(\text{COMPUTE}, \mathsf{sid}, 0)$ from $\mathcal{Z}$.
- When the simulated party P_1 receive $\{\tilde{c}_i := \tilde{b}_i \oplus x^*_{2,i}\}_{i\in[n_2]}$ from P_2, $\mathcal{S}$ uses the knowledge of $\tilde{b}_i$ from the internal state of $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$ to extract x^*_2. $\mathcal{S}$ then sends $(\text{COMPUTE}, \mathsf{sid}, x^*_2)$ to the external $\mathcal{F}^f_{\mathsf{2pc}}$ on behave of P_2.
- Upon receiving $(\text{OUTPUT}, \mathsf{sid}, P_2)$ from the external $\mathcal{F}^f_{\mathsf{2pc}}$, the simulator $\mathcal{S}$ returns $(\text{DELIVER}, \mathsf{sid}, P_2)$ if the simulated P_1 does not abort.
- Upon receiving $(\text{COMPUTE}, \mathsf{sid}, y)$ from the external $\mathcal{F}^f_{\mathsf{2pc}}$ for P_2, the simulator $\mathcal{S}$ picks a random check set $\mathcal{I}$. For $j \in \mathcal{I}$, $\mathcal{S}$ generates $(F'_j, X'_j, d'_j) \leftarrow \mathsf{GC.Gb}(1^\lambda, f_1; k_j)$. For $j \in [p(\lambda)/\mathcal{I}]$, $\mathcal{S}$ uses the GC simulator to generate $(F'_j, X'_j, d'_j) \leftarrow \mathsf{Sim}(1^\lambda, (y, r_j), \Phi(f_1))$.
- Upon receiving $(\mathsf{Run}, \mathsf{sid}, \langle \text{GC}, \langle f_1, p(\lambda), 1\rangle\rangle)$ from both P_1 and P_2 to $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$, $\mathcal{S}$ sends $\{(F'_j, d'_j)\}_{j\in[p(\lambda)]}$ as the GC tables and decode information to P_2 on behave of $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$.
- For $i \in [n_2], j \in [p(\lambda)]$, $\mathcal{S}$ then sets $\tilde{W}^{(x_{2,i})}_{j,i} := X'_{j,n_1+i} \oplus \tilde{R}^{(b_i)}_{j,i}$ and $\tilde{W}^{(x_{2,i}\oplus 1)}_{j,i} := \tilde{R}^{(b_i\oplus 1)}_{j,i}$; It then acts as P_1 to send $\{\tilde{W}^{(0)}_{j,i}, \tilde{W}^{(1)}_{j,i}\}_{i\in[n_2],j\in[p(\lambda)]}$ to P_2;
- The simulator $\mathcal{S}$ simulates the coin-flipping protocol to fix the check set as $\mathcal{I}$ as chosen before.
- $\mathcal{S}$ then uses X'_j as the wire labels to generate $\{Z_{j,i} := X'_{j,i}\}_{i\in[n_1],j\in[p(\lambda)]\setminus\mathcal{I}}$. It then acts as P_1 to send $\{Z_{j,i}\}_{i\in[n_1],j\in[p(\lambda)]\setminus\mathcal{I}}$ to P_2.

Indistinguishability. The indistinguishability is proven through a series of hybrid worlds $\mathcal{H}_0, \ldots, \mathcal{H}_5$.

Hybrid $\mathcal{H}_0$: It is the real protocol execution $\text{EXEC}^{\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]}_{\Pi^{\mathsf{m}}_{\mathsf{2pc}},\mathcal{A},\mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: $\mathcal{H}_1$ is the same as $\mathcal{H}_0$ except that in $\mathcal{H}_1$, $\mathcal{S}$ simulates the coin-flipping protocol to fix a random check set $\mathcal{I}$.

Claim 6. *$\mathcal{H}_1$ and $\mathcal{H}_0$ are indistinguishable if the underlying coin-flipping protocol is UC-secure.*

Hybrid $\mathcal{H}_2$: $\mathcal{H}_2$ is the same as $\mathcal{H}_1$ except that in the random OT generation, $\mathcal{H}_2$ uses λ-bit true random numbers $\{\hat{R}^{(0)}_{j,i}, \hat{R}^{(1)}_{j,i}, \bar{R}^{(0)}_{j,i}, \bar{R}^{(1)}_{j,i}\}_{i\in[n_1],j\in[p(\lambda)]}, \{\tilde{R}^{(0)}_{j,i}, \tilde{R}^{(1)}_{j,i}\}_{i\in[n_2],j\in[p(\lambda)]}$ instead of using PRF.

Claim 7. *$\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ is a secure PRF function with adversarial distinguishing advantage $\mathsf{Adv}_{\mathsf{PRF}}(\mathcal{A}, \lambda)$, then $\mathcal{H}_1$ and $\mathcal{H}_0$ are indistinguishable with distinguishing advantage $(2n_1 + n_2) \cdot p(\lambda) \cdot \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{A}, \lambda)$.*

Proof. It is a straightforward reduction; namely, by hybrid argument, we can show that if there exists an adversary $\mathcal{A}$ who can distinguish $\mathcal{H}_1$ from $\mathcal{H}_0$, then we can construct an adversary $\mathcal{B}$ who can break the PRF. □

Hybrid $\mathcal{H}_3$: $\mathcal{H}_3$ is the same as $\mathcal{H}_2$ except that in $\mathcal{H}_3$, when the simulated party P_1 receive $\{\tilde{c}_i := \tilde{b}_i \oplus x^*_{2,i}\}_{i\in[n_2]}$ from P_2, $\mathcal{S}$ uses the knowledge of $\tilde{b}_i$ from the internal state of $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$ to extract x^*_2. $\mathcal{S}$ then sends $(\text{COMPUTE}, \mathsf{sid}, x^*_2)$ to the external $\mathcal{F}^f_{\mathsf{2pc}}$ on behave of P_2. After receiving $(\text{COMPUTE}, \mathsf{sid}, y)$ from the external $\mathcal{F}^f_{\mathsf{2pc}}$ for P_2, the simulator $\mathcal{S}$ uses the GC simulator to generate $(F'_j, X'_j, d'_j) \leftarrow \mathsf{Sim}(1^\lambda, (y, r_j), \Phi(f_1))$, $j \in [p(\lambda)/\mathcal{I}]$.

For $i \in [n_2], j \in [p(\lambda)]$, $\mathcal{S}$ then sets $\tilde{W}^{(x_{2,i})}_{j,i} := X'_{j,n_1+i} \oplus \tilde{R}^{(b_i)}_{j,i}$ and $\tilde{W}^{(x_{2,i}\oplus 1)}_{j,i} := \tilde{R}^{(b_i\oplus 1)}_{j,i}$; It then acts as P_1 to send

$\{\tilde{W}_{j,i}^{(0)}, \tilde{W}_{j,i}^{(1)}\}_{i\in[n_2],j\in[p(\lambda)]}$ to P_2; $\mathcal{S}$ then uses X'_j as the wire labels to generate $\{Z_{j,i} := X'_{j,i}\}_{i\in[n_1],j\in[p(\lambda]]\setminus\mathcal{I}}$.

Claim 8. *If* GC *is simulatable private with adversarial distinguishing advantage* $\mathsf{Adv}_{\mathsf{GC}}^{\mathsf{prv.sim},\Phi,\mathsf{Sim}}(\mathcal{A},\lambda)$*, then* $\mathcal{H}_3$ *and* $\mathcal{H}_2$ *are indistinguishable with distinguishing advantage*

$$p(\lambda)/2 \cdot \mathsf{Adv}_{\mathsf{GC}}^{\mathsf{prv.sim},\Phi,\mathsf{Sim}}(\mathcal{A},\lambda) \ .$$

Proof. First of all, by the requirement of simulatable privacy in Def. 2, $(F', X', d') \leftarrow \mathsf{Sim}(1^\lambda, y, \Phi(f))$ should be indistinguishable from the real one. Moreover, since P_2 does not know $\tilde{R}_{j,i}^{(b_i\oplus 1)}$, by hybrid argument, if there is an adversary $\mathcal{A}$ who can distinguish the distribution of $\{Z_{j,i}\}_{i\in[n_1],j\in[p(\lambda]]\setminus\mathcal{I}}$ and $\{\tilde{W}_{j,i}^{(x_{2,i})}\}_{i\in[n_2],j\in[p(\lambda)]\setminus\mathcal{I}}$ from the real one, then we can construct an adversary $\mathcal{B}$ who can break the simulation privacy of GC. □

Hybrid $\mathcal{H}_4$: $\mathcal{H}_4$ is the same as $\mathcal{H}_3$ except that in $\mathcal{H}_4$, the simulator $\mathcal{S}$ sends $\{\bar{c}'_i := \bar{b}_i\}_{i\in[n_1]}$ to P_2, instead of $\{\bar{c}_i := \bar{b}_i \oplus x_{1,i}\}_{i\in[n_1]}$ for XOR-gadgets.

Claim 9. *$\mathcal{H}_4$ and $\mathcal{H}_3$ are perfectly indistinguishable.*

Proof. Since $\bar{b}_i$ are the ROT select bits randomly picked by $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$, the distribution of $\{\bar{c}'_i\}_{i\in[n_1]}$ and $\{\bar{c}_i\}_{i\in[n_1]}$ are identical. Therefore, $\mathcal{H}_4$ and $\mathcal{H}_3$ are perfectly indistinguishable. □

The adversary's view of $\mathcal{H}_4$ is identical to the simulated view $\text{EXEC}_{\mathcal{F}_{\mathsf{2pc}}^f,\mathcal{S},\mathcal{Z}}$. Therefore, the views of $\mathcal{A}$ and $\mathcal{Z}$ in $\text{EXEC}^{\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]}_{\Pi_{\mathsf{2pc}}^{\mathsf{m}},\mathcal{A},\mathcal{Z}}$ and $\text{EXEC}_{\mathcal{F}_{\mathsf{2pc}}^f,\mathcal{S},\mathcal{Z}}$ are negligible indistinguishable with distinguishing advantage

$$(2n_1+n_2)\cdot p(\lambda)\cdot \mathsf{Adv}_{\mathsf{PRF}}(\mathcal{A},\lambda)+p(\lambda)/2\cdot \mathsf{Adv}_{\mathsf{GC}}^{\mathsf{prv.sim},\Phi,\mathsf{Sim}}(\mathcal{A},\lambda) \ .$$

Case 4: P_1 and P_2 are corrupted; $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$ is honest.

Simulator. The simulator $\mathcal{S}$ internally runs $\mathcal{A}$, forwarding messages to/from the environment $\mathcal{Z}$. The simulator $\mathcal{S}$ simulates the functionality $\mathcal{F}_{\text{HW}}[\mathsf{M}^{\mathsf{m}}]$.

Indistinguishability. This is a trivial case, since both P_1 and P_2 are controlled by the adversary $\mathcal{A}$.

This concludes the proof. □