Honorific Security: Efficient Two-Party Computation with Offloaded Arbitration and Public Verifiability

Tianxiang Dai[1], Yufan Jiang[2,5], Yong Li[3], Jörn Müller-Quade[2,5], and Andy Rupp[4,5]

[1] Lancaster University Leipzig, Leipzig
t.dai@lancaster.ac.uk
[2] Karlsruhe Institute of Technology, Germany
{yufan.jiang, joern.mueller-quade}@kit.edu
[3] Huawei European Research Center, Germany
yong.li1@huawei.com
[4] University of Luxembourg, Luxembourg
andy.rupp@uni.lu
[5] KASTEL Security Research Labs, Germany

Abstract. Secure two-party computation (2PC) allows two distrustful parties to jointly compute some functions while keeping their private secrets unrevealed. Adversaries are often categorized as semi-honest and malicious, depending on whether they follow the protocol specifications or not. While a semi-honest secure protocol is fast but strongly assumed that all participants will follow the protocol, a malicious protocol often requires heavy verification steps and preprocessing phase to prohibit any cheat. Covert security [10] was first introduced by Aumann and Lindell, which looks into the "middle ground" between semi-honest security and malicious security, such that active adversaries who cheat will be caught with a predefined probability. However, it is still an open question that how to properly determine such a probability before protocol execution, and the misbehavior detection must be made by other honest participants with cut-and-choose in current constructions. To achieve public verifiability and meanwhile outsource the verification steps, [12] presented publicly auditable security to enable an external auditor to verify the result correctness. Essentially, an additional existence assumption of a bulletin board functionality is required to keep tracking the broadcasted messages for the auditor. And moreover, the auditor cannot identify the cheater, but only points out the incorrect result. The (*robust*) *accountability* family [40, 62, 76] achieves both output delivery guarantee and public verifiability, which relies on heavy offline and online constructions with zero knowledge proofs.

In this paper, we propose a new security notion called *honorific security*, where an external arbiter can find the cheater with overwhelming probability under the malicious corruption. With *honorific security*, we do not prohibit cheat of a corrupted party during the online stage, but enable the honest party to detect and punish the cheater later on in public. We show that a maliciously secure garbled circuit (GC) [83] protocol can

thus be constructed with only slightly more overhead than a passively secure protocol. Our construction performs up to 2.37 times and 13.30 times as fast as the state-of-the-art protocols with covert and malicious security, respectively.

Keywords: Two-party computation, Security notion, Efficient protocols, Multi-party computation, Honorific security

1 Introduction

In secure two-party computation, two parties are willing to jointly compute a function f without revealing their private input $\{x_1, x_2\}$ to each other. 2PC protocols should guarantee that besides the output of the given function $\{y_1, y_2\} = f(x_1, x_2)$, nothing else can be learned (privacy), and the output $\{y_1, y_2\}$ is distributed correctly (correctness).

In different settings, 2PC protocols can be designed against various types of adversaries, making trade-offs between efficiency and security. Up to now, two main categories of adversaries, i.e., *semi-honest adversaries* and *malicious adversaries*, are considered. Semi-honest adversaries can be seen as protocol participants who do not violate the protocols but attempt to learn more than the predefined output of the functions, where malicious adversaries may deviate arbitrarily from the protocol by taking actions to manipulate the result and messages. Protocols secure against semi-honest adversaries offer quite limited privacy, while the ones with malicious security [8, 27, 54, 56, 67, 81] are usually too inefficient for high-throughput applications in practice [32, 42]. Covert security [9, 26, 31, 39, 57] has targeted the middle ground between semi-honest and malicious security. These adversaries may actively cheat like malicious adversaries, but they can be caught with a constant probability ϵ and they are afraid of being caught. Protocols secure against such adversaries allow successful cheating, but guarantee that honest parties can detect such behaviors with the given probability ϵ. However, it is still an open question that how ϵ can be properly determined before the protocol execution.

In the meantime, applications that have large economic or political consequences such as auction [2, 21] and e-voting system [3, 61] all require that the result must be correctly computed, and the correctness must be *publicly verifiable*. Publicly auditable security [12] was proposed to address this issue, where an external auditor is introduced to take over all verifications and provides a publicly verifiable audit result. A small subtlety in [12] is that the auditor is not able to identify the cheater, even if it finds out that the protocol result is incorrect. Thus, the deterrence provided in [12] may not be sufficient to prevent a potential cheat, although it is publicly verifiable. *Accountability* [40, 62] and its variant *robust accountability* [76] provides more security guarantee than [12]. [76] guarantees that once the protocol terminates, parties output either the correct result or a subset of cheaters[6] (no honest party is falsely blamed), if the num-

[6] If a party is corrupted, it may behave honestly during the protocol execution.

ber of the corrupted parties is below a pre-defined threshold value. Additionally, once any cheater is detected, it can be publicly verified. To achieve such a strong notion, parties have to compute (verify) *non-interactive zero knowledge proof*s (**NIZKP**) and commitments in the online stage.

1.1 Outsourced Verification and Public Verifiability without Bulletin Board

In both maliciously and covert-secure protocols, we notice that functional computation and misbehavior detection are always performed simultaneously by the protocol participants. Such as in [40, 62, 76], the protocol terminates immediately, if the cheating is found. The *public verifiability* is often regarded as a by-product. For applications such as cloud services [4, 6, 17, 74], privacy preserving machine learning (PPML) tasks [64, 69, 79] and blockchain platforms [2, 23, 35, 68, 85], efficiency issue is always an important concern to be addressed beyond the security. In this paper, we use the existence of an external party (like in [12]) and a potential arbitration as a deterrence to force parties to behave honestly. While the deterrence provided in [12] is not sufficient, and meanwhile the line of work [40, 62, 76] relies on a heavy online (and offline) computation and verification, we propose a novel 2PC security notion *honorific security* by introducing an arbiter $\mathbf{P_{Ar}}$, which can identify the cheater (and is thus powerful than [12]). Like any law or regulation, the notion *honorific security* itself does not technically prohibit any cheat during execution, but provides the honest party with the ability to detect and punish the cheater later on in public.

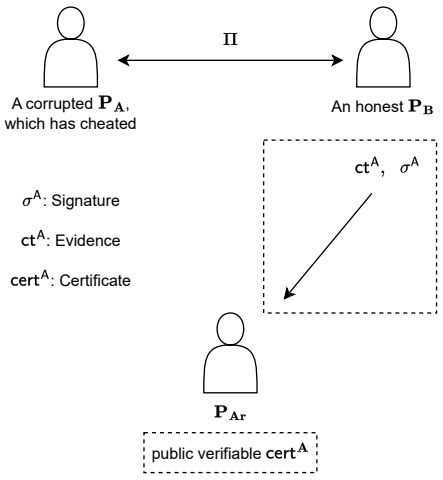

Fig. 1. Workflow of a potential arbitration

Previous work [12, 40, 62, 76] simply assumes that an external auditor has access to all transcripts published on a *bulletin board* ideal functionality $\mathcal{F}_{\mathsf{Bulletin}}$.

We point out that building protocols on top of $\mathcal{F}_{\mathsf{Bulletin}}$ is tricky, since the implementation of $\mathcal{F}_{\mathsf{Bulletin}}$ doubles the communication overhead (the same message will be transmitted to the auditor once again!). In addition, [12, 40, 62, 76] emphasizes that the audit can be started at anytime after the protocol execution. In reality, it means that protocol participants must keep the **entire** message transcripts for a potential audit even after the protocol termination. In this paper, we provide a more realistic implementation without relying on $\mathcal{F}_{\mathsf{Bulletin}}$.

As shown in Fig. 1, after a protocol Π is executed, both $\mathbf{P_A}$ and $\mathbf{P_B}$ have already collected decisive evidences and signatures. Suppose a corrupted $\mathbf{P_A}$ has cheated during the protocol execution, if $\mathbf{P_B}$ sends evidence ct^{A} and signature σ^{A} to $\mathbf{P_{Ar}}$, a publicly verifiable certificate $\mathsf{cert}^{\mathsf{A}}$ will be generated to identify the cheater $\mathbf{P_A}$. Similar to *accountability*, we require that an honest party cannot be falsely blamed. Note that such a condition is not trivial in *honorific security*, if $\mathbf{P_{Ar}}$ is maliciously corrupted[7]. Since $\mathbf{P_{Ar}}$ can be activated only after the main protocol is finished, we emphasize that protocols achieving *honorific security* are different from the traditional three-party computation, where all three parties should stay active during the whole protocol execution.

1.2 Application Scenarios

In this section, we introduce potential applications that could benefit from *honorific security*.

Privacy-Preserving Machine Learning Google [59] proposed the federated learning (FL) scenes where a (global) machine learning model is trained by different data owners without leaking their database. Later some works [34, 36, 46, 51] pointed out that the intermediate leakage in FL can be used by adversaries to infer sensitive information. Thus, frameworks such as [28, 47, 52, 71, 72, 75] solve this issue by letting parties execute 2PC protocols following a mix protocol approach. Regarding *machine learning as a service* (MLaaS) scenario, recent works [22, 25, 30, 63] consider applying machine learning inference as a service between a client holding the private data input and a server holding the model. [63] introduces a *model-extraction attack*, where a malicious client tries to extract the private model of a service provider (a server) by violating the passively secure 2PC protocol. From another point of view, a malicious service provider may also deviate from the protocol and thus cause the client to output some incorrect results [30]. Both [30, 63] design protocols to prevent above misbehavior in the online stage. Suppose there is another server acting as a validation service provider, which stays offline during the online protocol execution above but can be activated by any of the protocol participants for a validation service, either the malicious client or the malicious server is thus forced to act honestly to avoid a public accusation later on. For a PPML training application, previous work such as [18, 60] already consider bringing extra validation nodes (parties) to

[7] In [12, 40, 62, 76], the auditor is considered as an external party which can not be corrupted.

achieve a better security guarantee. In [18], two of the four parties are responsible only for the functional computation, where another two parties are assigned to check whether the computation is performed correctly. Remind that [18] achieves exactly *honorific security*, if two validation parties are regarded as $\mathbf{P_{Ar}}$ and the validation part is postponed to the latter phase.

Applications with Trusted Hardware In the meantime, trusted execution environment (TEE) technologies, such as Intel SGX and ARM TrustZone are widely used to accelerate MPC online computations or achieve additional security guarantees. [50] uses trusted hardware to achieve malicious security for graph neural networks training and inference. [33] enables a remote server equipped with an SGX to securely evaluate garbled circuits for input data provider. MPC constructions using hardware tokens such as [11, 38] also imply an independent party for attestation and sealing. Besides modelling trusted hardware as *tamper-proof* hardware tokens, [70] proposed a semi-trusted hardware model, where a TEE (e.g. SGX) can be maliciously corrupted and thus deviates from the protocol. [70] designs three-party GC protocols, where an SGX is responsible for the functional computation and activated during the entire online runtime. Obviously, we observe that the arbiter in *honorific security* can be realized by a TEE. Depending on how we model the TEE (e.g. *tamper-proof*, semi-honest or malicious) and where the TEE is located (e.g. only at one party, at both parties or at a remote server), 2PC protocols achieving *honorific security* can be constructed differently.

1.3 Technical Overview

In this paper, we flesh out the GC [83] instantiation, and we benchmark our construction against state-of-the-art GC protocols. We stress that *honorific security* does not have to be realized by a GC-based protocol only.

In a GC protocol, the garbler has to send a garbled circuit to the evaluator and meanwhile perform an OT protocol to hand over the wire labels. The garbler could cheat by sending an incorrect GC, or handing over some incorrect wire labels to the OT protocol, and then observing whether the evaluator aborts (*selective failure attack*). However, if the evaluator holds the encrypted $\mathsf{seed}^{\mathsf{A}}$ as an evidence, which is used by the garbler to derive randomness, it can forward this evidence to $\mathbf{P_{Ar}}$, enabling $\mathbf{P_{Ar}}$ to reconstruct the whole circuit and the correct wire labels. Thus, the evidence held by their protocol partners can be a strong incentive for parties to behave honestly to avoid punishment in public after a potential arbitration. We refer the reader to Section 5 for more details.

1.4 Our Contribution

In this work, we formalize a new security notion called ***honorific security*** and then present an efficient GC protocol and an efficient oblivious transfer (OT) protocol with *honorific security* in the universal composability (UC) framework [19]. More specifically, we achieve the following goals:

- ***New Ideal Functionalities in UC.*** We require correctness and privacy when there is at most one maliciously corrupted party (*honest majority*). This is achieved by introducing an extra, maliciously corruptible third party (the **arbiter $\mathbf{P_{Ar}}$**) to the 2PC ideal functionality, which can check and verify the misbehavior of protocol participants. We point out, that it is sometimes not sufficient for $\mathbf{P_{Ar}}$ to only receive the arbitration result from the functionality internally, if a participant is required to use a certain input to the functionality or use the exact output received from the functionality for further computation. Thus, we define an ideal functionality of two-party computation in two modes. In Lazy-Arbiter mode ($\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$), the functionality sends only the arbitration result to $\mathbf{P_{Ar}}$. And in Busy-Arbiter mode ($\mathcal{F}_{\mathsf{BA}}^{\mathsf{2pc}}$), the functionality also forwards the input (and the output if needed) of the arbitrated party to $\mathbf{P_{Ar}}$, enabling $\mathbf{P_{Ar}}$ to check the input consistency beyond the ideal functionality. We also present an OT ideal functionality in Lazy-Auditor mode ($\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$).
- ***Practical Constructions.*** We construct protocols that realize $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ and $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$ based on symmetric key encryption, garbled circuits and digital signatures, and we prove that all of our constructions are secure in the UC framework. Specificallly, we do not rely on the *bulletin board* funcitonality $\mathcal{F}_{\mathsf{Bulletin}}$. The general idea behind our constructions is that participants are responsible for exchanging encrypted evidences including input-independent randomness they have used with each other. To achieve *public verifiability*, participants also have to sign the evidences and the transcript hash to be compared with. Once a misbehavior is discovered during an arbitration, $\mathbf{P_{Ar}}$ can simply publish the decrypted evidences along with the signed hash of the transcripts, enabling the public to identify the cheater.
- ***High Efficiency.*** We provide a fair comparison of our protocol against state-of-the-art GC-based protocols with covert and malicious security in Section 8. The intuition behind *honorific security* is that both parties should behave honestly even if the arbiter **does not** interfere. 2PC protocols are then accelerated by offloading non-functional computations to an potential arbitration section, which can be executed separately. To highlight the power of our notion and practical constructions, we always let the arbitration take place and count its cost in the experiments. We show, that even with the arbitration cost added, our protocols are almost as efficient as those with semi-honest security and up to 2.37 times and 13.30 times as fast as protocols with covert and malicious security, respectively.

2 Related Work

Security Notions beyond Semi-honest. The formal definition of malicious security can be found in Goldreich's seminal two volume classics [37]. Protocols with malicious security [8, 24, 27, 29, 43, 54, 56, 67, 81, 82] ensure that even if an adversary $\mathcal{A}$ deviates from protocol definition arbitrarily, $\mathcal{A}$ cannot learn anything about other parties' inputs, except that $\mathcal{A}$ may cause other parties to abort (*security with abort*) [66]. Typical constructions and analysis can be found in

[54, 67, 81]. Covert security was first introduced by Aumann and Lindell [10] in 2007 against rational adversaries, targeting the middle ground between semi-honest and malicious security. In principle, in a covert-secure protocol, cheating behavior can succeed with probability $1 - \epsilon$, and will be detected by other parties with the remaining probability ϵ, which is also called the deterrence factor. Follow-up works [9, 26, 39, 65] confirmed that protocols with reasonable ϵ have a clear advantage in efficiency over ones with malicious security. Among them, authors of [9] highlight another critical feature that a covert-secure MPC protocol may need: the *public verifiability* (PVC). However, existing instantiations in [9, 10, 14, 26, 31, 39, 45, 57, 78] are still much heavier than the semi-honest ones in terms of computational resources and bandwidth consumption.

Distinction from Other Notions. Bringing in an arbiter has already made our model distinct from pure two party computation, where each party takes care of its own privacy all by itself after setting up. Compared to covert security [10], we require the probability of catching cheaters to be overwhelming instead of being a constant probability. A similar approach [12] introduces an external auditor beyond protocol participants and achieves *publicly auditable security*. However, the auditor is not able to identify the cheater. The notion *identifiable abort* [13, 49, 73] allows all honest parties to identify at least one corrupted party which causes the protocol to abort *during* execution. *Accountability* [40, 62] and its variant *robust accountability* [76] achieves both output delivery guarantee and public verifiability by performing **NIZKP**s and commitments in the online stage. As mentioned in the previous section, all notions require either a broadcast channel, or a bulletin board functionality (or both) to achieve an additional security guarantee. The most important aspect regarding these notions is that the functional computation and the verification computation are performed simultaneously by protocol participants themselves.

Role of an Extra Party. The idea of delegating part of the MPC tasks to an independent party roots in Beaver's Commodity-based Cryptography in 1997 [15]. Independently generated correlated randomness, such as Beaver's multiplicative triples [15], has been extremely helpful in performance improvement of sharing-based MPC frameworks, such as [12, 40, 56, 62]. Another role of the extra party is for security. MPC constructions using hardware tokens [11, 38] also imply an independent party for attestation and sealing, which can persuade both Alice and Bob to believe the correctness and non-malleability of logic in the hardware [5]. Prior works such as [18, 60] also consider including extra parties as verification nodes instead of participating in the functional part (in the honest majority setting).

3 Preliminaries

We summarize notations used in this paper in Table 1. We use "a party uses randomness derived from `seed`" as a convention for the action that a party uses

seed as the key of a pseudorandom function (PRF) to obtain a sufficiently long series of pseudorandomness.

$\{(x_j^0, x_j^1)\}_{j\in[m]}$	m pairs messages (x^0, x^1)
$\{\hat{B}_{i,j,b}\}$	a set of correct evaluator's input label extracted from $\hat{\mathsf{GC}}_i$
$\{\mathsf{pk}_i, \mathsf{sk}_i\}$	public key and private key of $\mathbf{P_i}$ for a signature scheme
$\{A_{i,b}\}$	a set of garbler's input label $A_{i,b}$ for i-th wire, label b
$\{B_{i,b}\}$	a set of evaluator's input label $B_{i,b}$ for i-th wire, label b
$\{O_{i,b}\}$	a set of output label $O_{i,b}$ for i-th wire, label b
o^{B}	output of $\mathbf{P_B}$
$\{y_j^0, y_j^1\}_{j\in[m]}$	m pairs encrypted messages
GCS	garbling scheme $\mathsf{GCS} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$
$\hat{\mathsf{GC}}$	Garbled circuit computed by $\mathbf{P_{Ar}}$ using decrypted $\mathsf{seed}^{\mathsf{A}}$
GC	Garbled circuit computed by $\mathbf{P_A}$
κ	the security parameter
key	Symmetric encryption-decryption key
$\mathbf{P}_i$	$\mathbf{P_i}$
$\mathbf{r}$	choice bit of $\mathbf{P_B}$
$\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$	ideal functionality 2pc with honorific security in Lazy-Arbiter mode
$\mathcal{F}_{\mathsf{BA}}^{\mathsf{2pc}}$	ideal functionality 2pc with honorific security in Busy-Arbiter mode
$\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$	ideal functionality $\mathsf{m}\times\mathsf{OT}$ with honorific security in Busy-Arbiter mode against sender
$\mathcal{F}_{\mathsf{BA,R}}^{\mathsf{m}\times\mathsf{OT}}$	ideal functionality $\mathsf{m}\times\mathsf{OT}$ with honorific security in Busy-Arbiter mode against receiver
$\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$	ideal functionality $\mathsf{m}\times\mathsf{OT}$ with honorific security in Layz-Arbiter mode
$\mathsf{cheatParty}$	a flag documenting cheated party
$\mathsf{cheated}$	notification that the arbitrated party cheated
$\mathsf{ct}^{\mathsf{A,OT}}$	evidence computed by $\mathbf{P_A}$ for an OT protocol
$\mathsf{ct}^{\mathsf{A,GC}}$	evidence computed by $\mathbf{P_A}$ for a GC protocol
Π^{GC}	GC Protocol
$\Pi_{\mathsf{BA,S}}^{\mathsf{OTE}}$	OTE Protocol realizes $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$
$\Pi_{\mathsf{LA}}^{\mathsf{OTE}}$	OTE Protocol realizes $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$
$\mathcal{S}$	the simulator
$\mathcal{Z}$	the environment machine
$\mathsf{seed}^{\mathsf{A}}$	seed used for garbling ith circuit by $\mathbf{P_A}$
sid	session identifier
$\sigma^{\mathsf{A,OT}}$	signature computed by $\mathbf{P_A}$ for an OT protocol
$\sigma^{\mathsf{A,GC}}$	signature computed by $\mathbf{P_A}$ for a GC protocol
table	decryption table of ith garbled circuit
$a \xleftarrow{\$} V$	sampling an element from V uniformly at random
H	hash function modeled as random oracle
$\mathcal{H}$	hash value of some messages
$\mathbf{h}$	commitment
$x_{\mathbf{i}}$	$\mathbf{P_i}$'s valid input

Table 1. Notations.

Definition 1 (Garbling Scheme). *A circuit garbling scheme* $\mathsf{GCS} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *consists of the following algorithms.*

- $\mathsf{Gb}(1^\kappa, \mathcal{C})$ *denotes the garbling algorithm. It takes the security parameter* 1^κ *and the circuit* $\mathcal{C}$ *as input. It returns a garbled circuit* GC*, encoding information* e*, and decoding information* d*.*
- $\mathsf{En}(w, e)$ *denotes the encoding algorithm. It takes the input* w *and encoding information* e *as input. It returns the garbled input* $\{W_{i,b}\}$*.*
- $\mathsf{Ev}(\mathsf{GC}, W)$ *denotes the evaluation algorithm. It takes the garbled circuit* GC *and garbled input* $\{W_{i,b}\}$ *as input. It returns a garbled output* $\{O_i\}$*.*
- $\mathsf{De}(d, \{O_i\})$ *denotes the decoding algorithm. It takes the decoding information* d *and garbled output* $\{O_i\}$ *as input. It returns the output* $\{o\}$*.*

Definition 2 (Correctness [16]). *A garbling scheme* $\mathsf{GCS} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *is correct, if for all functions* $\mathcal{C}$ *and input* w*:*

$$\Pr\left[\begin{matrix}\mathsf{De}(d, \mathsf{Ev}(\mathsf{GC}, \mathsf{En}(e, w))) = \mathcal{C}(w) : \\ (\mathsf{GC}, e, d) \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C})\end{matrix}\right] = 1$$

Definition 3 (Simulatable Privacy [16, 70]). *A garbling scheme* $\mathsf{GCS} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *is simulatable private, if for all functions* $\mathcal{C}$ *and input* w*, there exists a probabilistic polynomial time (PPT) simulator* $\mathbf{Sim}$ *such that for all PPT adversary* $\mathcal{A}$*:*

$$\Pr\left[\begin{matrix}b = b^\star : \\ (\mathsf{GC}_0, e_0, d_0) \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C}); W_0 \leftarrow \mathsf{En}(e, w); \\ (\mathsf{GC}_1, W_1, d_1) \leftarrow \mathbf{Sim}(1^\kappa, \mathcal{C}(w), \Phi(\mathcal{C})); \\ b \leftarrow \{0,1\}; b^\star \leftarrow \mathcal{A}(\mathsf{GC}_b, W_b, d_b);\end{matrix}\right] \leq \mathsf{negl}(\kappa),$$

where Φ *denotes the side-information function.*

Definition 4. ***(Signature scheme)*** *A signature scheme* $\mathsf{SIG} = (\mathsf{SIG.Gen}, \mathsf{SIG.Sign}, \mathsf{SIG.Vfy})$ *is described as below:*

- $\mathsf{SIG.Gen}(1^\kappa) \xrightarrow{\$} (\mathsf{pk}, \mathsf{sk})$*. The non-deterministic key generation algorithm* $\mathsf{SIG.Gen}()$ *takes the security parameter* 1^κ *as the input and outputs the public key* pk *and the private key* sk*.*
- $\mathsf{SIG.Sign}(\mathsf{sk}, m) \xrightarrow{\$} \sigma$*. The (non-deterministic) message signing algorithm* $\mathsf{SIG.Sign}()$ *takes the private key* sk *and a message* m *as the input and outputs the a signature* σ*.*
- $\mathsf{SIG.Vfy}(\mathsf{pk}, m, \sigma) = b$*. The deterministic signature verification algorithm* $\mathsf{SIG.Vfy}()$ *takes the public key* pk*, a message* m *a signature* σ *as input and outputs a boolean value* b*.*

Definition 5 (Symmetric key encryption scheme). *A symmetric key encryption scheme* $\Pi = (\mathsf{KGen}, \mathsf{ENC}, \mathsf{DEC})$ *is described as below.*

- $\Pi.\mathsf{KGen}(1^\kappa) \xrightarrow{\$} \mathsf{key}$*. The non-deterministic key generation algorithm* $\mathsf{KGen}()$ *takes the security parameter* 1^κ *as the input and outputs one encryption-decryption key* key*.*

- $\Pi.\mathsf{ENC}(\mathsf{key}, m) \xrightarrow{\$} c$. *The (non-deterministic) encryption algorithm* $\mathsf{ENC}()$ *takes the key* key *and a message* m *as the input and outputs a ciphertext* c.
- $\Pi.\mathsf{DEC}(\mathsf{key}, c) = m'$. *The deterministic decryption algorithm* $\mathsf{DEC}()$ *takes the key* key*, a ciphertext* c *as input and outputs a plaintext* m'.

Due to the page limitation, we refer the reader to cryptography texts such as [53] for definitions of correctness and security of hash function, PRF and other primitives.

4 Ideal Functionalities with Honorific Security in UC

In this section, we formally define the ideal functionality with *honorific security*, see Fig. 2 and the Definition 6 for more details.

4.1 Ideal Functionalities $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ and $\mathcal{F}_{\mathsf{BA}}^{\mathsf{2pc}}$

Let $\mathbf{P_A}$, $\mathbf{P_B}$ and $\mathbf{P_{Ar}}$ denote the participating parties $\mathcal{P} = \{\mathbf{P_A}, \mathbf{P_B}, \mathbf{P_{Ar}}\}$, $\mathcal{P}_c \subseteq \{\mathbf{P_A}, \mathbf{P_B}, \mathbf{P_{Ar}}\}$ denote the corrupted parties controlled by an adversary. We include the following basic queries in our new ideal functionality:

Inputs: Let x_i denote party $\mathbf{P_i}$'s input, z denote adversary $\mathcal{S}$'s auxiliary input.

Sends input to ideal functionality: The honest party $\mathbf{P_j}$ sends its input to the ideal functionality. The corrupted parties may either send their predefined input, some other input of the same length, or abort (by replacing the input with a special $(\mathbf{abort}, \mathbf{P_i}, \mathsf{sid})$ to the ideal functionality.

Early Abort Option: If the ideal functionality receives the message $(\mathbf{abort}, \mathbf{P_i}, \mathsf{sid})$, it sends $(\mathbf{abort}, \mathbf{P_i}, \mathsf{sid})$ to all parties and the ideal execution terminates. Otherwise, the ideal execution proceeds.

Ideal functionality sends outputs to adversary: The ideal functionality computes $f_i(x'_\mathsf{A}, x'_\mathsf{B})$ and sends $f_i(x'_\mathsf{A}, x'_\mathsf{B})$ to party $\mathbf{P_i}$ for all $i \in \mathcal{P}_c$ (i.e. to all corrupted parties).

Adversary instructs ideal functionality to continue or halt: $\mathcal{S}$ sends either $(\mathbf{continue}, \mathbf{P_i}, \mathsf{sid})$ or $(\mathbf{abort}, \mathbf{P_i}, \mathsf{sid})$ to the ideal functionality. If received $(\mathbf{continue}, \mathbf{P_i}, \mathsf{sid})$, the ideal functionality sends $f_j(x'_\mathsf{A}, x'_\mathsf{B})$ to the honest party $\mathbf{P_j}$ for all $j \notin P_c$ (i.e. to all honest parties). Otherwise, the ideal functionality sends $(\mathbf{abort}, \mathbf{P_i}, \mathsf{sid})$ to party $\mathbf{P_j}$.

Outputs: An honest party always outputs what the ideal functionality sends to it. The corrupted parties output nothing. The adversary $\mathcal{S}$ outputs any (probabilistic polynomial time computable) function of the initial inputs $\{x_\mathrm{i}\}_{i \in P_c}$, the auxiliary input z, and the messages $\{f_i(x'_\mathsf{A}, x'_\mathsf{B})\}_{i \in P_c}$ received from the ideal functionality.

Cheat flag. Besides basic queries defined in [19, 20], the ideal functionality now has an additional flag **cheatParty** internally, recording cheating parties (or none) during the execution. Another internal state **arbiterReady** denotes whether the flag **cheatParty** has been set properly.

Functionality $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ interacts with players $\mathcal{P} := \{\mathbf{P_A}, \mathbf{P_B}, \mathbf{P_{Ar}}\}$ and the adversary $\mathcal{S}$. It has three internal states: a set of corrupted parties $\mathcal{P}_c \subseteq \{\mathbf{P_A}, \mathbf{P_B}, \mathbf{P_{Ar}}\}$, a set of cheated parties $\mathbf{cheatParty} \subseteq \{\mathbf{P_A}, \mathbf{P_B}\}$, and a state $\mathbf{arbiterReady} \in \{\mathbf{true}, \mathbf{false}\}$. Initially, $\mathcal{P}_c = \emptyset$, $\mathbf{cheatParty} = \emptyset$, $\mathbf{arbiterReady} = \mathbf{false}$.

Corrupt: Upon receiving $(\mathbf{corrupt}, \mathbf{P_i}, \mathsf{sid})$ from the adversary $\mathcal{S}$:

- If $\mathbf{P_i} \in \mathcal{P}$ and $\mathcal{P}_c = \emptyset$, set $\mathcal{P}_c := \{\mathbf{P_i}\}$. Send $(\mathbf{corrupt\_success}, \mathbf{P_i}, \mathsf{sid})$ to $\mathcal{S}$.
- Otherwise, send $(\mathbf{corrupt\_failed}, \mathbf{P_i}, \mathsf{sid})$ to $\mathcal{S}$.

Compute: Upon receiving $(\mathbf{compute}, x_{\mathrm{A}}, \mathbf{P_A}, \mathsf{sid})$ from party $\mathbf{P_A}$ and $(\mathbf{compute}, x_{\mathrm{B}}, \mathbf{P_B}, \mathsf{sid})$ from party $\mathbf{P_B}$:

- Compute $f(x_{\mathrm{A}}, x_{\mathrm{B}})$ and send it to $\mathbf{P_B}$.
- Set $\mathbf{arbiterReady} = \mathbf{true}$.

Cheat: Upon receiving $(\mathbf{cheat}, \mathbf{P_i}, \mathsf{sid})$ from party $\mathbf{P_i}$, where $\mathbf{P_i} \in \{\mathbf{P_A}, \mathbf{P_B}\}$:

- If $\mathbf{P_i} \in \mathcal{P}_c$, send a message $(\mathbf{cheat\_success}, x_{\mathrm{j}}, \mathsf{sid})$ to $\mathcal{S}$, wait to receive o^{j} from $\mathbf{P_i}$ and send o^{j} to $\mathbf{P_j}$. Set $\mathbf{cheatParty} = \mathbf{P_i} \cup \mathbf{cheatParty}$ and $\mathbf{arbiterReady} = \mathbf{true}$.
- Otherwise, send a message $(\mathbf{cheat\_failed}, \mathbf{P_i}, \mathsf{sid})$ to $\mathcal{S}$.

Arbitrate: Upon receiving $((\mathbf{arbitrate}, \mathbf{P_j}), \mathbf{P_i}, \mathsf{sid})$ from party $\mathbf{P_i}$ intended to arbitrate $\mathbf{P_j}$:

- If $\mathbf{arbiterReady} = \mathbf{false}$, ignore this query.
- **Lazy-Arbiter mode**: If $\mathbf{P_j} \in \mathbf{cheatParty}$, send $(\mathbf{cheated}, \mathbf{P_j}, \mathsf{sid})$ to $\mathbf{P_{Ar}}$ and halt. Otherwise, send $(\mathbf{honest}, \mathbf{P_j}, \mathsf{sid})$ to $\mathbf{P_{Ar}}$.
- **Busy-Arbiter mode**: If $\mathbf{P_j} \in \mathbf{cheatParty}$, send $(\mathbf{cheated}, \mathbf{P_j}, \mathsf{sid})$ to $\mathbf{P_{Ar}}$ and halt. Otherwise, send $\mathsf{info} \subseteq \{x_{\mathrm{j}}, o^{\mathsf{j}}\}$ to $\mathbf{P_{Ar}}$.

Fig. 2. Two Party Functionality $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ and $\mathcal{F}_{\mathsf{BA}}^{\mathsf{2pc}}$.

Cheat query. We then extend the ideal functionality by adding a new instruction, such that a simulator $\mathcal{S}$ can send to the ideal functionality. Similar to the ideal functionalities with covert security, $\mathcal{S}$ is able to send a **cheat** query to the ideal functionality, and this cheat decision "must be made before the adversary learns anything" [10]. Remark that $\mathbf{P_{Ar}}$ can only cheat by framing a party up with an incorrect arbitration result, since $\mathbf{P_{Ar}}$ will be activated only for the arbitration. Although such a cheat will be prevented by our protocol construction, we still provide the cheat option to $\mathbf{P_{Ar}}$ for further extensions.

Arbirate query. Generally, we allow any party to send an arbitrate query ((**arbirate**, $\mathbf{P_j}$), $\mathbf{P_i}, \mathsf{sid}$) to the ideal functionality, including $\mathbf{P_A}$, $\mathbf{P_B}$ and $\mathbf{P_{Ar}}$. From practical point of view, allowing $\mathbf{P_{Ar}}$ to request arbitration requires $\mathbf{P_{Ar}}$ to be aware of every protocol execution, which might be unrealistic. In this paper, we focus on the case that only $\mathbf{P_A}$ and $\mathbf{P_B}$ send this query to the ideal functionality.

Arbiter mode. On the other hand, sometimes $\mathbf{P_{Ar}}$ may also have to arbitrate whether a protocol participant has handed in the *correct* input to the ideal functionality, or used the *correct* output for the further execution received from an ideal functionality, even if it behaved honestly *during* the protocol execution. We thus split the ideal functionality into two modes:

- **Lazy-Arbiter mode**: $\mathbf{P_{Ar}}$ only receives the arbitration result from the ideal functionality, whether the arbitrated party (say $\mathbf{P_j}$ in the following context) cheated or not *during* the protocol execution.
- **Busy-Arbiter mode**: If $\mathbf{P_j}$ already cheated during the protocol execution, $\mathbf{P_{Ar}}$ receives $(\mathbf{cheated}, \mathbf{P_j}, \mathsf{sid})$ from the ideal functionality. Otherwise, instead of receiving the notification, $\mathbf{P_{Ar}}$ obtains $\mathsf{info} \subseteq \{x_\mathsf{j}, o^\mathsf{j}\}$.

We point out that the input of a sub-protocol does **not** have to be the private input of any party, and $\mathbf{P_{Ar}}$ only receives such information when it has to be arbitrated.

Definition 6 (Honorific Security). *Let $\mathcal{F}^{\mathsf{2pc}}$ be a two-party functionality and $\mathcal{F}^{\mathsf{2pc}}_{\mathsf{LA}}$, $\mathcal{F}^{\mathsf{2pc}}_{\mathsf{BA}}$ be the corresponding functionality with honorific security in Lazy-Arbiter Mode and Busy-Arbiter Mode. We say that a protocol Π UC-realizes $\mathcal{F}^{\mathsf{2pc}}$ with honorific security, if Π UC-realizes $\mathcal{F}^{\mathsf{2pc}}_{\mathsf{LA}}$, or $\mathcal{F}^{\mathsf{2pc}}_{\mathsf{BA}}$.*

4.2 Ideal Functionalities $\mathcal{F}^{\mathsf{m}\times\mathsf{OT}}_{\mathsf{BA,S}}$ and $\mathcal{F}^{\mathsf{m}\times\mathsf{ROT}}_{\mathsf{BA,S}}$

The OT ideal functionality $\mathsf{m}\times\mathsf{OT}$ and the random OT ideal functionality $\mathsf{m}\times\mathsf{ROT}$ with *honorific security* shown in Fig. 3 are constructed in Busy-Arbiter mode. And this is exactly the case when parties perform a GC protocol Π^{GC} using $\mathcal{F}^{\mathsf{m}\times\mathsf{OT}}_{\mathsf{BA,S}}$ as a sub-protocol, where $\mathbf{P_A}$'s input to $\mathcal{F}^{\mathsf{m}\times\mathsf{OT}}_{\mathsf{BA,S}}$ is just randomness generated during the intermediate steps within Π^{GC}. Similarly, if $\mathcal{F}^{\mathsf{m}\times\mathsf{ROT}}_{\mathsf{BA,S}}$ is chosen as a sub-protocol, the output delivered to $\mathbf{P_A}$ does **not** relate to any party's real input. Importantly, $\mathbf{P_B}$'s input to $\mathcal{F}^{\mathsf{m}\times\mathsf{OT}}_{\mathsf{BA,S}}$ and $\mathcal{F}^{\mathsf{m}\times\mathsf{ROT}}_{\mathsf{BA,S}}$ is the real private input regarding Π^{GC}, and does not need to be arbitrated beyond the scope of an OT protocol. For this reason, if $\mathbf{P_B}$ is arbitrated, the ideal functionality should only deliver the arbitration result to $\mathbf{P_{Ar}}$.

5 Protocol UC-realizing $\mathcal{F}^{\mathsf{2pc}}_{\mathsf{LA}}$

In this section, we show how to realize $\mathcal{F}^{\mathsf{2pc}}_{\mathsf{LA}}$ based on GC. We introduce a high-level overview of the protocol Π^{GC}, and provide a formal definition in Fig. 4.

Using a signature scheme SIG, all three parties $\mathbf{P_A}$, $\mathbf{P_B}$ and $\mathbf{P_{Ar}}$ run $\mathsf{SIG.Gen}$ to obtain their public-private key pairs. We assume that all parties know the public keys of each other as the common reference string (CRS) before running the protocol. In the CRS model, this will allow the simulator $\mathcal{S}$ to simulate the key pair for the signature scheme.

In the key setup phase, $\mathbf{P_{Ar}}$ calls $\Pi.\mathsf{KGen}$ of a symmetric key encryption scheme to generate a symmetric key $\mathsf{key}^{\mathsf{Ar}}$. $\mathbf{P_{Ar}}$ then computes a commitment

Functionality $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$ interacts with players $\mathcal{P} := \{\mathbf{S}, \mathbf{R}, \mathbf{P_{Ar}}\}$ and the adversary $\mathcal{S}$. It has three internal states: a set of corrupted parties $\mathcal{P}_c \subseteq \{\mathbf{S}, \mathbf{R}, \mathbf{P_{Ar}}\}$, a set of cheated parties **cheatParty** $\subseteq \{\mathbf{S}, \mathbf{R}\}$, and a state **arbiterReady** $\in \{\mathbf{true}, \mathbf{false}\}$.
Initially, $\mathcal{P}_c = \emptyset$, **cheatParty** $= \emptyset$, **arbiterReady** $=$ **false**.

Corrupt: same as $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ and $\mathcal{F}_{\mathsf{BA}}^{\mathsf{2pc}}$.
Compute ($\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$ and $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$): Upon receiving $(\{x^0, x^1\}^m, \mathbf{S}, \mathsf{sid})$ from $\mathbf{S}$ and $(x_{\mathsf{B}}, \mathbf{R}, \mathsf{sid})$ from $\mathbf{R}$, send $\{x^{x_{\mathsf{B}}[i]}\}^m$ to $\mathbf{R}$, set **arbiterReady** $=$ **true**.
Compute ($\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{ROT}}$): Upon receiving $(x_{\mathsf{B}}, \mathbf{R}, \mathsf{sid})$ from $\mathbf{R}$:

- If $\mathbf{S} \notin \mathcal{P}_c$, sample random $\{x^0, x^1\}^m$, send $\{x^0, x^1\}^m$ to $\mathbf{S}$ and $\{x^{x_{\mathsf{B}}[i]}\}^m$ to $\mathbf{R}$.
- Otherwise, wait for $\mathbf{S}$ to input $\{x^0, x^1\}^m$, then output as above using these values.
- Set **arbiterReady** $=$ **true**.

Cheat: same as $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ and $\mathcal{F}_{\mathsf{BA}}^{\mathsf{2pc}}$.
Arbitrate($\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$ and $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{ROT}}$):

- If **arbiterReady** $=$ **false**, ignore the following queries.
- Upon receiving $((\mathbf{arbitrate}, \mathbf{S}), \mathbf{R}, \mathsf{sid})$ from party $\mathbf{R}$ intended to arbitrate $\mathbf{S}$: If $\mathbf{S} \in$ **cheatParty** send $(\mathbf{cheated}, \mathbf{S}, \mathsf{sid})$ to $\mathbf{P_{Ar}}$ and halt. Otherwise, send $\{x^0, x^1\}^m$ to $\mathbf{P_{Ar}}$.
- Upon receiving $((\mathbf{arbitrate}, \mathbf{R}), \mathbf{S}, \mathsf{sid})$ from party $\mathbf{S}$ intended to arbitrate $\mathbf{R}$: If $\mathbf{R} \in$ **cheatParty** send $(\mathbf{cheated}, \mathbf{R}, \mathsf{sid})$ to $\mathbf{P_{Ar}}$ and halt. Otherwise, send $(\mathbf{honest}, \mathbf{R}, \mathsf{sid})$ to $\mathbf{P_{Ar}}$.

Arbitrate($\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$):

- If **arbiterReady** $=$ **false**, ignore the following queries.
- Upon receiving $((\mathbf{arbitrate}, \mathbf{P_j}), \mathbf{P_i}, \mathsf{sid})$ from party $\mathbf{P_i}$ intended to arbitrate $\mathbf{P_j}$: If $\mathbf{P_j} \in$ **cheatParty** send $(\mathbf{cheated}, \mathbf{P_j}, \mathsf{sid})$ to $\mathbf{P_{Ar}}$ and halt. Otherwise, send $(\mathbf{honest}, \mathbf{P_j}, \mathsf{sid})$ to $\mathbf{P_{Ar}}$.

Fig. 3. OT Functionality $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$, $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{ROT}}$ and $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$.

$\mathbf{h}$ on $\mathsf{key}^{\mathsf{Ar}}$, sends $\mathsf{key}^{\mathsf{Ar}}$, $\mathbf{h}$ and decom to $\mathbf{P_A}$ along with a signature σ^{Ar} on $\mathbf{h}$. Afterward, $\mathbf{P_B}$ receives $\mathbf{h}$ and σ^{Ar} from $\mathbf{P_A}$. This setup is mainly considered for $\mathbf{P_A}$ to prepare its encrypted evidences in the main part of the protocol. Obviously, $\mathbf{P_B}$ can obtain another symmetric key with the same setup steps simultaneously if needed, having $\mathbf{P_A}$ holding the signed commitment.

To achieve *honorific security*, both party $\mathbf{P_A}$ and $\mathbf{P_B}$ have to send evidence and signature to each other. In GC-based 2PC, if $\mathbf{P_B}$ wants to check whether $\mathbf{P_A}$ has cheated during the protocol execution, $\mathbf{P_B}$ can send the commitment $\mathbf{h}$, the evidence $\mathsf{ct}^{\mathsf{A,GC}}$, the signature $\sigma^{\mathsf{A,GC}}$, and the corresponding hash value $\mathcal{H}$ on GC to $\mathbf{P_{Ar}}$, which can open the evidence and check all $\mathbf{P_A}$'s behaviors during the protocol execution. Recall that $\mathbf{P_{Ar}}$ is allowed to hold $\mathsf{seed}^{\mathsf{A}}$ since it does not leak any information about the private input of both $\mathbf{P_A}$ and $\mathbf{P_B}$. Meanwhile, the wire labels of both parties are required to be kept secret from $\mathbf{P_{Ar}}$ (ensured by the honest majority setting). Note that $\mathbf{P_B}$ has already proved

Private inputs: $\mathbf{P_A}$ has input $x_\mathsf{A} \in \{0,1\}^{n_1}$ and a key pair $\{\mathsf{pk_A}, \mathsf{sk_A}\}$ for the signature scheme. $\mathbf{P_B}$ has input $x_\mathsf{B} \in \{0,1\}^{n_2}$ and a key pair $\{\mathsf{pk_B}, \mathsf{sk_B}\}$ for the signature scheme. $\mathbf{P_{Ar}}$ has a key pair $\{\mathsf{pk_{Ar}}, \mathsf{sk_{Ar}}\}$ for the signature scheme.
Public inputs: $\mathbf{P_A}$ and $\mathbf{P_B}$ agree on a circuit $\mathcal{C}$ and a parameter κ. All three parties know the public key pk_i of each other and a session ID sid.
CRS: $(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{pk_{Ar}})$.

Key Setup:

1. $\mathbf{P_{Ar}}$ generates $\mathsf{key}^{\mathsf{Ar}}$ and computes a commitment $\mathbf{h} \leftarrow \mathsf{Com}(\mathsf{key}^{\mathsf{Ar}})$, then signs it with a signature $\sigma^{\mathsf{Ar}} \leftarrow \mathsf{SIG.Sign}(\mathsf{sk_{Ar}}, \mathbf{h})$.
2. $\mathbf{P_{Ar}}$ sends $(\mathsf{key}^{\mathsf{Ar}}, \mathbf{h}, \mathsf{decom}, \sigma^{\mathsf{Ar}})$ to $\mathbf{P_A}$, which verifies whether $\mathbf{h}$ and σ^{Ar} are both valid, aborts with output $\perp$ if not.
3. $\mathbf{P_A}$ sends $(\mathbf{h}, \sigma^{\mathsf{Ar}})$ to $\mathbf{P_B}$, which verifies whether σ^{Ar} is valid, and aborts with output $\perp$ if not.

Protocol:

1. $\mathbf{P_A}$ garbles the circuit $\mathcal{C}$ using randomness derived from $\mathsf{seed}^{\mathsf{A}}$. The garbled circuit is denoted as GC, as well as $\mathbf{P_A}$'s input wire labels $\{A_{i,b}\}_{i\in[n_1],b\in\{0,1\}}$, $\mathbf{P_B}$'s input wire labels $\{B_{i,b}\}_{i\in[n_2],b\in\{0,1\}}$, and output wire labels $\{O_{i,b}\}_{i\in[n_3],b\in\{0,1\}}$. $\mathbf{P_A}$ then computes a decoding table $\mathsf{table} \leftarrow \{\mathsf{Label}_i^0, \mathsf{Label}_i^1\}_{i\in[n_3]}$, where $\mathsf{Label}_i^0 \leftarrow \mathsf{H}(O_{i,0})$ and $\mathsf{Label}_i^1 \leftarrow \mathsf{H}(O_{i,1})$.
2. $\mathbf{P_A}$ and $\mathbf{P_B}$ call $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$, where $\mathbf{P_A}$ uses $\{B_{i,b}\}_{i\in[n_2],b\in\{0,1\}}$ as input and $\mathbf{P_B}$ uses x_B as input.
3. $\mathbf{P_A}$ computes an evidence $\mathsf{ct}^{\mathsf{A,GC}} \leftarrow \Pi.\mathsf{ENC}(\mathsf{key}^{\mathsf{Ar}}, \mathsf{seed}^{\mathsf{A}})$. Then $\mathbf{P_A}$ computes a hash value $\mathcal{H} \leftarrow \mathsf{H}(\mathcal{C}||\mathsf{GC}||\mathsf{table}||\mathsf{sid})$ and a signature of this message $\sigma^{\mathsf{A,GC}} \leftarrow \mathsf{SIG.Sign}(\mathsf{sk_A}, \mathbf{h}||\mathcal{H}||\mathsf{ct}^{\mathsf{A,GC}}||\mathsf{sid})$. $\mathbf{P_A}$ then sends $(\mathsf{GC}, \mathsf{table}, \mathsf{ct}^{\mathsf{A,GC}}, \sigma^{\mathsf{A,GC}}, \mathsf{sid})$ to $\mathbf{P_B}$.
4. $\mathbf{P_B}$ computes $\mathcal{H}$ and checks whether $\sigma^{\mathsf{A,GC}}$ is a valid signature for $(\mathbf{h}, \mathcal{H}, \mathsf{ct}^{\mathsf{A,GC}}, \mathsf{sid})$, and aborts with output $\perp$ if not.
5. $\mathbf{P_A}$ sends $\{A_{i,x_\mathsf{A}[i]}\}_{i\in[n_1]}$ to $\mathbf{P_B}$.
6. $\mathbf{P_B}$ evaluates GC using $\{A_{i,x_\mathsf{A}[i]}\}_{i\in[n_1]}$ and $\{B_{i,x_\mathsf{B}[i]}\}_{i\in[n_2]}$, and obtains $\{O_{i,o^\mathsf{B}[i]}\}_{i\in[n_3]}$. Then, $\mathbf{P_B}$ computes $\{\mathsf{H}(O_{i,o^\mathsf{B}[i]})\}_{i\in[n_3]}$. If any $\mathsf{H}(O_{i,o^\mathsf{B}[i]}) \notin \{\mathsf{Label}_i^0, \mathsf{Label}_i^1\}$, $\mathbf{P_B}$ aborts with output $\perp$, otherwise $\mathbf{P_B}$ outputs o^B.

* **Arbitrate:**

1. $\mathbf{P_B}$ sends an arbitrate query $(\mathcal{C}, \mathcal{H}, \mathsf{ct}^{\mathsf{A,GC}}, \sigma^{\mathsf{A,GC}}, \mathsf{sid})$ to $\mathbf{P_{Ar}}$, which checks whether $\sigma^{\mathsf{A,GC}}$ is valid, and aborts with output $\perp$ if not.
2. $\mathbf{P_B}$ sends an arbitrate query $((\mathbf{arbitrate}, \mathbf{P_A}), \mathbf{P_B}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$. If $\mathbf{P_{Ar}}$ receives $(\mathbf{cheated}, \mathbf{P_A}, \mathsf{sid})$, then the arbitration ends here. If $\mathbf{P_{Ar}}$ receives $\{B_{i,b}\}$, the arbitration proceeds.
3. $\mathbf{P_{Ar}}$ retrieves $\mathsf{seed}^{\mathsf{A}} \leftarrow \Pi.\mathsf{DEC}(\mathsf{key}^{\mathsf{Ar}}, \mathsf{ct}^{\mathsf{A,GC}})$, computes $\{\hat{B}_{i,b}\}$, $\hat{\mathsf{GC}}$ and $\hat{\mathsf{table}}$ using the randomness derived from $\mathsf{seed}^{\mathsf{A}}$.
4. $\mathbf{P_{Ar}}$ computes $\hat{\mathcal{H}} \leftarrow \mathsf{H}(\mathcal{C}||\hat{\mathsf{GC}}||\hat{\mathsf{table}}||\mathsf{sid})$. If $\hat{\mathcal{H}} = \mathcal{H}$, and $\{\hat{B}_{i,j,b}\} = \{B_{i,b}\}$, then $\mathbf{P_{Ar}}$ locally outputs $(\mathbf{honest}, \mathbf{P_A}, \mathsf{sid})$. Otherwise, $\mathbf{P_{Ar}}$ outputs $(\mathbf{cheated}, \mathbf{P_A}, \mathsf{sid})$.

Fig. 4. Full description of GC based Π^{GC} that UC-realizes $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$.

whether the signed commitment $\mathbf{h}$ is indeed provided by $\mathbf{P_{Ar}}$ by verifying the signature of $\mathbf{P_{Ar}}$ in the key setup stage. It ensures that $\mathbf{P_A}$ must encrypts the correct $\mathsf{seed}^{\mathsf{A}}$ with the symmetric key $\mathsf{key}^{\mathsf{Ar}}$ provided by $\mathbf{P_{Ar}}$ (since $\mathbf{P_{Ar}}$ will use the correct $\mathsf{key}^{\mathsf{Ar}}$ to decrypt), otherwise the arbitration will fail except for negligible probability.

The above idea can prevent any $\mathbf{P_A}$ from cheating during the GC generation phase, since $\mathbf{P_B}$ can verify $\mathbf{P_A}$'s behavior at any time by sending this evidence to $\mathbf{P_{Ar}}$. But during the OT protocol, where $\mathbf{P_B}$ learns the receiver's input wire labels, $\mathbf{P_A}$ can still perform the *selective failure attack* by providing a pair of true and false labels, and then observing if $\mathbf{P_B}$ aborts to obtain $\mathbf{P_B}$'s one-bit input. In our model, if $\mathbf{P_A}$ performs such an attack and $\mathbf{P_B}$ prosecutes, $\mathbf{P_{Ar}}$ should be able to detect such dishonest behavior. However, we notice that an original OT functionality (or even $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$) does not provide $\mathbf{P_{Ar}}$ with such an ability or other materials to perform the check. The reason is that when parties call a functionality as a sub-protocol, the input requirement is "out of scope" of this functionality description. As an example, we suppose that parties execute $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$ as the sub-protocol within Π^{GC}. The OT sender $\mathbf{P_A}$ does not cheat by sending the cheat option to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$, but performs the *selective failure attack* described as above. We observe that $\mathbf{P_{Ar}}$ will receive the notification of $\mathbf{P_A}$'s honesty from $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$, since $\mathbf{P_A}$ has not cheated internally during the OT protocol execution, although $\mathbf{P_A}$ does not input the **correct** labels to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$ as expected. Interestingly, this cheat can be captured by the simulator during simulation, since the simulator can decrypt $\mathsf{seed}^{\mathsf{A}}$ and thus be aware that the labels sent from $\mathbf{P_A}$ to the simulated OT ideal functionality are incorrect. In reality, $\mathbf{P_{Ar}}$ is not given such an ability yet. To solve this problem, we require an improved OT functionality shown in Fig. 3 to be used within our protocol:

- $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$ outputs directly (**cheated**, $\mathbf{P_A}, \mathsf{sid}$) to $\mathbf{P_{Ar}}$ if $\mathbf{P_A}$ cheated internally in $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$ already, or it outputs $\mathbf{P_A}$'s input $\{B_{i,b}\}$ after receiving the arbitrate query from $\mathbf{P_B}$. This allows $\mathbf{P_{Ar}}$ to perform the consistency check, by comparing the real input $\{B_{i,b}\}$ with the claimed input computed by the $\mathsf{seed}^{\mathsf{A}}$.
- $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{ROT}}$ works similarly compared to $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$, it just outputs $\mathbf{P_A}$'s output $\{B_{i,b}\}$ to $\mathbf{P_{Ar}}$ instead of the input. In this case, only $\{A_{i,b}\}$ are generated by $\mathbf{P_A}$ using randomness derived from the $\mathsf{seed}^{\mathsf{A}}$. As for $\mathbf{P_B}$'s input wire labels, $\mathbf{P_A}$ is supposed to use the messages obtained from $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{ROT}}$. To check whether $\mathbf{P_A}$ has cheated, $\mathbf{P_{Ar}}$ only has to compare the hashes of two circuits, where one of them is computed by $\mathbf{P_{Ar}}$ itself using $\{A_{i,b}\}$ generated by $\mathsf{seed}^{\mathsf{A}}$ and $\{B_{i,b}\}$ received from $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{ROT}}$, and the another one is sent from $\mathbf{P_B}$ along with the signature of $\mathbf{P_A}$.

Finally, $\mathbf{P_A}$ sends the garbler's input $\{A_{i,x_{\mathsf{A}}[i]}\}$ to $\mathbf{P_B}$, allowing $\mathbf{P_B}$ to compute the output o^{B}. Although $\mathbf{P_A}$ can send some invalid $\{A_{i,x_{\mathsf{A}}[i]}\}$ and cause $\mathbf{P_B}$ to abort. But as already mentioned in [66, 81], any such abort occurs independently of $\mathbf{P_B}$'s private input, and thus does not help $\mathbf{P_A}$ to learn a single bit of $\mathbf{P_B}$'s input.

Functionality $\mathcal{F}_{\mathsf{BA,R}}^{\mathsf{m}\times\mathsf{OT}}$ interacts with players $\mathcal{P} := \{\mathbf{S}, \mathbf{R}, \mathbf{P_{Ar}}\}$ and the adversary $\mathcal{S}$. It has three internal states: a set of corrupted parties $\mathcal{P}_c \in \{\mathbf{S}, \mathbf{R}, \mathbf{P_{Ar}}\}$, a set of cheated parties $\mathbf{cheatParty} \in \{\mathbf{S}, \mathbf{R}\}$, and a state $\mathbf{arbiterReady} \in \{\mathbf{true}, \mathbf{false}\}$.
Initially, $\mathcal{P}_c = \emptyset$, $\mathbf{cheatParty} = \emptyset$, $\mathbf{arbiterReady} = \mathbf{false}$.

Corrupt: same as $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$.
Compute:: same as $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$.
Cheat: same as $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$.
Arbitrate:

- If $\mathbf{arbiterReady} = \mathbf{false}$, ignore the following queries.
- Upon receiving $((\mathbf{arbitrate}, \mathbf{S}), \mathbf{R}, \mathsf{sid})$ from party $\mathbf{R}$ intended to arbitrate $\mathbf{S}$: If $\mathbf{S} \in \mathbf{cheatParty}$ send $(\mathbf{cheated}, \mathbf{S}, \mathsf{sid})$ to $\mathbf{P_{Ar}}$ and halt. Otherwise, send $(\mathbf{honest}, \mathbf{S}, \mathsf{sid})$ to $\mathbf{P_{Ar}}$.
- Upon receiving $((\mathbf{arbitrate}, \mathbf{R}), \mathbf{S}, \mathsf{sid})$ from party $\mathbf{S}$ intended to arbitrate $\mathbf{R}$: If $\mathbf{R} \in \mathbf{cheatParty}$ send $(\mathbf{cheated}, \mathbf{R}, \mathsf{sid})$ to $\mathbf{P_{Ar}}$ and halt. Otherwise, send $(x_{\mathrm{B}}, \{x^{x_{\mathrm{B}}[i]}\})$ to $\mathbf{P_{Ar}}$.

Fig. 5. $\mathsf{m} \times \mathsf{OT}$ ideal functionality $\mathcal{F}_{\mathsf{BA,R}}^{\mathsf{m}\times\mathsf{OT}}$.

Specifically, the above protocol ensures that no one is framed up. Again, we consider the case that $\mathbf{P_A}$ is arbitrated by $\mathbf{P_{Ar}}$. In order to successfully frame $\mathbf{P_A}$ up, either $\mathbf{P_B}$ or $\mathbf{P_{Ar}}$ has to forge an incorrect evidence (or transcript hash), which can be successfully verified with $\mathbf{P_A}$'s signature. If SIG is existentially unforgeable under chosen-message attacks (see Section 7 for more details), this can be achieved only with negligible probability.

Due to efficiency issues, we choose to discuss our Π^{GC} using $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$ for further proofs and implementation sections.

6 Oblivious Transfer with Honorific Security

6.1 Protocol UC-realizing $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$

In this section, we show how to convert an OT Extension protocol such as [55, 77] to a protocol which UC-realizes $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$. We take the improved OT Extension protocol [55] (which is now a simple instantiation of SoftSpokenOT [77]) as an example, and we show that the modified protocol $\Pi_{\mathsf{BA,S}}^{\mathsf{OTE}}$ shown in Appendix A implements $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$ in the $\mathcal{F}_{\mathsf{BA,R}}^{\mathsf{l}\times\mathsf{OT}}$-hybrid model.

Since the original protocol [55] is secure against both malicious $\mathbf{P_A}$ and $\mathbf{P_B}$, we focus on how to enable $\mathbf{P_{Ar}}$ to receive $\mathbf{P_A}$'s input as an additional output (and nothing else), if $\mathbf{P_B}$ sends the arbitrate query to $\mathbf{P_{Ar}}$ (and $\mathbf{P_A}$ is honest). Recall that a base OT protocol is executed as a sub-protocol in [55], where $\mathbf{P_A}$ uses $\mathbf{s}$ as its input and receives $\{k_{s_i}\}$ as output. Since both $\mathbf{s}$ and $\{k_{s_i}\}$ are just randomness generated by $\mathbf{P_A}$ and $\mathbf{P_B}$, allowing $\mathbf{P_{Ar}}$ to hold this information is harmless. Note that holding both $\mathbf{s}$ and $\{k_{s_i}\}$ along with the exact messages $\mathbf{P_A}$

has received from $\mathbf{P_B}$ enables $\mathbf{P_{Ar}}$ to reconstruct $\mathbf{P_A}$'s view, including $\mathbf{P_A}$'s real input to the OT Extension protocol. Again, we are facing the same problem as in Π^{GC}, since a traditional OT ideal functionality will not forward $\mathbf{s}$ and $\{k_{s_i}\}$ to $\mathbf{P_{Ar}}$. Thus, $\mathbf{P_A}$ and $\mathbf{P_B}$ have to call $\mathcal{F}_{\mathsf{BA,R}}^{\mathsf{l}\times\mathsf{OT}}$ shown in Fig. 5 ($\mathcal{F}_{\mathsf{BA,R}}^{\mathsf{m}\times\mathsf{OT}}$ with $\mathbf{m}=\mathbf{l}$), enabling $\mathbf{P_{Ar}}$ to receive such information. If $\mathbf{P_B}$ (the OT sender in $\mathcal{F}_{\mathsf{BA,R}}^{\mathsf{l}\times\mathsf{OT}}$) sends the arbitrate query to $\mathcal{F}_{\mathsf{BA,R}}^{\mathsf{l}\times\mathsf{OT}}$, it then forwards $\mathbf{P_A}$'s input $\mathbf{s}$ and output $\{k_{s_i}\}$ to $\mathbf{P_{Ar}}$ (if $\mathbf{P_A}$ is honest). In addition, we let $\mathbf{P_A}$ sign the message transcripts and send the signature $\sigma^{\mathsf{A,OT}}$ to $\mathbf{P_B}$ in step $**$. This ensures that $\mathbf{P_B}$ cannot frame $\mathbf{P_A}$ up by sending incorrect message transcripts to $\mathbf{P_{Ar}}$.

$\mathcal{F}_{\mathsf{BA,R}}^{\mathsf{l}\times\mathsf{OT}}$ can be easily implemented by running any modified maliciously secure OT protocol l times, where $\mathbf{P_A}$ (the OT receiver) additionally sends its input encrypted by the symmetric key $\mathsf{key}^{\mathsf{Ar}}$ and a signature (on the encrypted input and the message transcripts) to $\mathbf{P_B}$ (the OT sender). We notice that such an implementation requires an additional communication round for the last message sent from $\mathbf{P_A}$, if $\mathcal{F}_{\mathsf{BA,R}}^{\mathsf{l}\times\mathsf{OT}}$ is separately executed. However, if $\mathcal{F}_{\mathsf{BA,R}}^{\mathsf{l}\times\mathsf{OT}}$ is called as a sub-protocol (as in $\Pi_{\mathsf{BA,S}}^{\mathsf{OTE}}$), the additional communication round is omitted.

6.2 Protocol UC-realizing $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$

We further modify the OT Extension protocol [7] to a protocol $\Pi_{\mathsf{LA}}^{\mathsf{OTE}}$, which UC-realizes $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$ described in Fig. 3 in the $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{l}\times\mathsf{OT}}$-hybrid model ($\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$ with $\mathbf{m}=\mathbf{l}$). More details can be found in Fig. 6.

Since [7] is already secure against a malicious $\mathbf{P_A}$, we focus on the security against a malicious $\mathbf{P_B}$. In [7], a malicious $\mathbf{P_B}$ may use inconsistent $\mathbf{r}$ to compute $\{\mathbf{u}^i\}^l$ then extract $\mathbf{P_A}$'s secret input $\mathbf{s}$. As long as we can prevent $\mathbf{P_B}$ from doing so, we are done. We notice that $\{k_0,k_1\}^l$ used by $\mathbf{P_B}$ in base OT stage are only randomness, which can thus be received by $\mathbf{P_{Ar}}$ as evidence for further arbitration. It only remains the issue about how to provide $\mathbf{P_{Ar}}$ with the ability to check the input consistency of $\mathbf{r}$, without revealing it. Our solution is to let $\mathbf{P_B}$ compute a masked version of $\mathbf{u}^i$ with a random string Δ, say $\mathbf{v}^i$, along with a signature generated by $\mathbf{P_B}$. $\mathbf{P_A}$ is supposed to receive all $\mathbf{u}^i$, it can simply compute $\mathbf{v}^i$ by XORing all $\mathbf{u}^i$ with the received Δ, and then checks whether the signature is valid. Remark that by completing the signature verification, $\mathbf{P_A}$ has already checked the consistency of Δ. In the arbitration part, $\mathbf{P_{Ar}}$ receives all $\mathbf{v}^i$ from $\mathbf{P_A}$ along with $\{k_0,k_1\}^l$ received from $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{l}\times\mathsf{OT}}$, which allows $\mathbf{P_{Ar}}$ to compute $\mathbf{R}^i$, where each $\mathbf{R}^i$ is supposed to be $\mathbf{r}\oplus\Delta$. Thus, $\mathbf{P_{Ar}}$ only have to verify the consistency of $\mathbf{R}^i$.

7 Security Analysis

Let $\mathbf{vrfy}()$ denote the arbitration algorithm that $\mathbf{P_{Ar}}$ performs. Let $\mathsf{cert}^{\mathbf{Ar,j}}$ denote a certificate, which consists of the transcript $\mathbf{P_{Ar}}$ receives from $\mathbf{P_i}$ to arbitrate $\mathbf{P_j}$ along with $(\mathsf{key}^{\mathsf{Ar}},\mathsf{decom})$. We first define ***Public Verifiability***.

Private inputs: $\mathbf{P_A}$ has input $\{(x_j^0, x_j^1)\}_{j\in[m]}$ and key pair $\{\mathsf{pk_A}, \mathsf{sk_A}\}$ for a signature scheme. $\mathbf{P_B}$ has input $\mathbf{r} = (r_1, ..., r_m)$ and key pair $\{\mathsf{pk_B}, \mathsf{sk_B}\}$ for a signature scheme. $\mathbf{P_{Ar}}$ has a key pair $\{\mathsf{pk_{Ar}}, \mathsf{sk_{Ar}}\}$ for the signature scheme.
Public inputs: All three parties know the public key pk_i of each other.
CRS: $(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{pk_{Ar}})$.

Protocol:

1. $\mathbf{P_A}$ initializes a random vector $\mathbf{s} \in \{0,1\}^l$ and $\mathbf{P_B}$ chooses random $\{k_0, k_1\}^l$ each of size κ.
2. Parties proceed with a $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{l\times OT}}$ ($\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m\times OT}}$ with $\mathbf{m} = \mathbf{l}$), where $\mathbf{P_A}$ acts as the receiver with input $\mathbf{s}$ and $\mathbf{P_B}$ acts as the sender with input $\{k_0, k_1\}^l$.
3. For $i \in [l]$, let $\mathbf{t}^i = \mathsf{G}(k_0^i)$. Let $T = [t^1|...|t^l]$ denote the $m \times l$ matrix where ith column is $\mathbf{t}^i$. Let $\mathbf{t}_j$ denote the jth row of T. $\mathbf{P_B}$ uses input choice bits $\mathbf{r}$ to compute $\mathbf{t}^i \leftarrow \mathsf{G}(k_0^i)$ and $\mathbf{u}^i \leftarrow \mathbf{t}^i \oplus \mathsf{G}(k_1^i) \oplus \mathbf{r}$.
4. $\mathbf{P_B}$ sends $\{\mathbf{u}^i\}^l$ to $\mathbf{P_A}$.

** $\mathbf{P_B}$ then chooses a random string $\Delta \xleftarrow{\$} \mathbb{Z}_{2^m}$, then computes $\{\mathbf{v}^i\}_{i\in[l]}$, where $\mathbf{v}^i \leftarrow \mathbf{u}^i \oplus \Delta$. $\mathbf{P_B}$ generates the signature $\sigma^{\mathsf{B,OT}} \leftarrow \mathsf{SIG.Sign}(\mathsf{sk_B}, \{\mathbf{v}^i\}^l || \mathsf{sid})$

** $\mathbf{P_B}$ sends $(\Delta, \sigma^{\mathsf{B,OT}}, \mathsf{sid})$ to $\mathbf{P_A}$, which checks if the signature $\sigma^{\mathsf{B,OT}}$ is valid, and aborts with output $\perp$ if not.

5. For $i \in [l]$, $\mathbf{P_A}$ defines $\mathbf{q}^i = (s_i \cdot \mathbf{u^i}) \oplus \mathsf{G}(k_{s_i}^i)$. Let $Q = [q^1|...|q^l]$ denote the $m \times l$ bit matrix with $\mathbf{q^i}$ represents its ith column. Let $\mathbf{q}_j$ denote the jth row of the matrix Q. Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$.
6. $\mathbf{P_A}$ computes and sends $\{y_j^0, y_j^1\}_{j\in[m]}$, where:

$$y_j^0 = x_j^0 \oplus \mathsf{H}(j, \mathbf{q}_j) \quad \text{and} \quad y_j^1 = x_j^1 \oplus \mathsf{H}(j, \mathbf{q}_j \oplus \mathbf{s})$$

7. $\mathbf{P_B}$ outputs $(x_1^{r_1}, ..., x_m^{r_m})$ with $x_j^{r_j} = y_j^{r_j} \oplus \mathsf{H}(j, \mathbf{t}_j)$.

* **Arbitrate:**

1. $\mathbf{P_A}$ sends an arbitrate query $(\{\mathbf{v}^i\}^l, \sigma^{\mathsf{B,OT}}, \mathsf{sid})$ to $\mathbf{P_{Ar}}$, which checks whether $\sigma^{\mathsf{B,OT}}$ is valid, and aborts with output $\perp$ if not.
2. $\mathbf{P_A}$ sends an arbitrate query $((\mathbf{arbitrate}, \mathbf{P_B}), \mathbf{P_A}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{l\times OT}}$. If $\mathbf{P_{Ar}}$ receives $(\mathbf{cheated}, \mathbf{P_B}, \mathsf{sid})$, then the arbitration ends here. If $\mathbf{P_{Ar}}$ receives $\{k_0, k_1\}^l$, the arbitration proceeds.
3. $\mathbf{P_{Ar}}$ computes $\mathbf{R}^i \leftarrow \mathbf{v}^i \oplus \mathsf{G}(k_0^i) \oplus \mathsf{G}(k_1^i)$ for all $i \in [l]$. If all $\mathbf{R}^i$ are consistent, $\mathbf{P_{Ar}}$ locally outputs $(\mathbf{honest}, \mathbf{P_B}, \mathsf{sid})$. Otherwise, $\mathbf{P_{Ar}}$ outputs $(\mathbf{cheated}, \mathbf{P_B}, \mathsf{sid})$.

Fig. 6. A secure OT Extension protocol $\Pi_{\mathsf{LA}}^{\mathsf{OTE}}$ that UC-realizes $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m\times OT}}$.

Definition 7 (Public Verifiability). *If any protocol participant* $\mathbf{P_j}$ *cheats and an honest participant* $\mathbf{P_i}$ *sends the arbitrate query,* $\mathbf{P_{Ar}}$ *always outputs an arbitration result* $(\mathbf{cheated}, \mathbf{P_j}, \mathsf{sid})$ *with a* $\mathsf{cert}^{\mathbf{Ar,j}}$*, except with negligible probability. If* $\mathbf{P_{Ar}}$ *sends* $\mathsf{cert}^{\mathbf{Ar,j}}$ *to any party* $\mathbf{P_k}$*, then* $\mathbf{P_k}$ *always outputs* $(\mathbf{cheated}, \mathbf{P_j}, \mathsf{sid})$ *by executing* $\mathbf{vrfy}(\mathsf{cert}^{\mathbf{Ar,j}})$*, except with negligible probability.*

7.1 Security in Π^{GC}

Theorem 1. *Assume* $\mathsf{GCS} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ *is a simulatably private and correct garbling scheme,* $\mathsf{H}()$ *is a correlation-robust cryptographic hash function,* SIG *is existentially unforgeable under chosen-message attacks (EUF-CMA), and* SKE *Π is secure under chosen-plaintext attacks (IND-CPA). Protocol Π^{GC} described in Fig.4 UC-realizes $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ described in Fig.2 with public verifiablity in the $\mathcal{F}^{\mathsf{CRS}}$, $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m\times OT}}$-hybrid model in the presence of a malicious adversary who can corrupt either $\mathbf{P_A}$, $\mathbf{P_B}$ or $\mathbf{P_{Ar}}$, with static corruption.*

Let $\mathcal{A}$ be a malicious, static adversary that interacts with parties performing the protocol Π^{GC} shown in Fig. 4. We construct an adversary $\mathcal{S}$ for the ideal process for $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ such that no environment $\mathcal{Z}$ can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and the protocol Π^{GC} or with $\mathcal{S}$ in the ideal process for $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$.

Initialization step: The common reference string (CRS) is chosen by $\mathcal{S}$ in the following way (recall that $\mathcal{S}$ chooses the CRS for the simulated $\mathcal{A}$ by itself):

– $\mathcal{S}$ runs $\mathsf{SIG.Gen}(1^\kappa)$, obtaining three pair $(\mathsf{pk_A}, \mathsf{sk_A})$, $(\mathsf{pk_B}, \mathsf{sk_B})$ and $(\mathsf{pk_{Ar}}, \mathsf{sk_{Ar}})$.

Then, $\mathcal{S}$ sets the CRS to equal $(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{pk_{Ar}})$, and stores $\mathsf{sk_A}$, $\mathsf{sk_B}$ and $\mathsf{sk_{Ar}}$.

Simulating the communication with $\mathcal{Z}$: Every input value that $\mathcal{S}$ receives from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape (as if coming from $\mathcal{A}$'s environment). Every output value written by $\mathcal{A}$ on its output tape is copied to $\mathcal{S}$'s own output tape (to be read by $\mathcal{S}$'s environment $\mathcal{Z}$).

Simulating the case $\mathbf{P_B}$ is corrupted. $\mathcal{S}$ simulates a real execution in which the corrupted $\mathbf{P_B}$ controlled by $\mathcal{A}$ delivers message to uncorrupted $\mathbf{P_A}$ in the internal (simulated) interaction. The $\mathcal{S}$ works as follows:

1. In the key setup stage, plays the role of an honest $\mathbf{P_A}$, receives $(\mathsf{key}^{\mathsf{Ar}}, \mathbf{h}, \mathsf{decom}, \sigma^{\mathsf{Ar}})$ from $\mathbf{P_{Ar}}$. Then sends $(\mathbf{h}, \sigma^{\mathsf{Ar}})$ to $\mathcal{A}$.
2. Chooses a κ-bit $\mathsf{seed}^{\mathsf{A}}$ uniformly at random. Uses pseudorandomness derived from the $\mathsf{seed}^{\mathsf{A}}$ to generate pseudorandom keys $\{B_{i,b}\}_{i\in[n_2],b\in\{0,1\}}$.
3. Plays the $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m\times OT}}$ with $\mathcal{A}$ playing $\mathbf{P_B}$:
   - If the input is $(\mathbf{cheat}, \mathbf{P_B}, \mathsf{sid})$, sends $(\mathbf{cheat}, \mathbf{P_B}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ and receives back $\mathbf{P_A}$'s input x_{A}. Then uses the input x_{A} of $\mathbf{P_A}$ to perfectly emulate $\mathbf{P_A}$ in rest of the execution. Sends o^A to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$, which will be received by $\mathbf{P_A}$ as output. The simulation ends here for this case.
   - If the input is x_{B}, hands $\mathcal{A}$ the wire labels $\{B_{i,j,x_{\mathsf{B}}[j]}\}$ that are chosen by $\mathcal{A}$, proceeds with the simulation below.
4. Sends the $\mathcal{A}$'s input x_{B} as $\mathbf{P_B}$'s input to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$, receives back the output o^{B}.
5. Uses the appropriate $\mathbf{P_B}$'s input wire labels from above, generates a garbled GC and the corresponding table, which are always evaluated to o^{B}.

6. Computes $\mathsf{ct}^{\mathsf{A},\mathsf{GC}}$ using the $\mathsf{seed}^{\mathsf{A}}$, then the hash value $\mathcal{H}$ using the garbled GC.
7. Computes the corresponding $\sigma^{\mathsf{A},\mathsf{GC}}$, sends GC, table, $\mathsf{ct}^{\mathsf{A},\mathsf{GC}}$, $\sigma^{\mathsf{A},\mathsf{GC}}$, sid to $\mathcal{A}$.
8. Hands $\mathcal{A}$ arbitrary $\mathbf{P_A}$'s input wire labels and halts.

Proof. We now prove that $\mathcal{Z}$ cannot distinguish an interaction of protocol Π^{GC} with $\mathcal{A}$ (denoted by $\mathsf{REAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}_{\Pi^{\mathsf{GC}},\mathcal{A},\mathcal{Z}}$) from an interaction in the ideal process with $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ and $\mathcal{S}$ (denoted by $\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}_{\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}},\mathcal{S},\mathcal{Z}}$):

Case 1: $\mathcal{A}$ sends $(\mathbf{cheat}, \mathbf{P_B}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$. We point out that $\mathcal{Z}$ cannot distinguish $\mathsf{REAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}_{\Pi^{\mathsf{GC}},\mathcal{A},\mathcal{Z}}$ with $\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}_{\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}},\mathcal{S},\mathcal{Z}}$ in this case. Note that the oblivious transfer is the first step of the protocol. After sending the cheat option to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$, $\mathcal{S}$ receives back the exact honest $\mathbf{P_A}$'s input x_{A} and can thus perfectly emulate the $\mathbf{P_A}$'s behavior during the simulated execution of the protocol. Besides, $\mathcal{S}$ can easily simulate the messages sent from $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$ to $\mathbf{P_{Ar}}$ and perform the exact arbitration as an honest $\mathbf{P_{Ar}}$ by holding $\mathsf{key}^{\mathsf{Ar}}$. Thus, the simulation of $\mathbf{P_{Ar}}$ for $\mathcal{A}$ is also perfect.

Case 2: $\mathcal{A}$ sends $(x_{\mathsf{B}}, \mathbf{P_B}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$. We examine several hybrid experiments:

Hybrid $\mathcal{H}_0$: It is the real protocol execution $\mathsf{REAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}_{\Pi^{\mathsf{GC}},\mathcal{A},\mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: We consider a simulator $\mathcal{S}_0$, which holds the real $\mathbf{P_A}$'s input x_{A} and the exact $\mathsf{seed}^{\mathsf{A}}$, denoted by $\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}_{\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}},\mathcal{S}_0(x_{\mathsf{A}},\mathsf{seed}^{\mathsf{A}}),\mathcal{Z}}$.

It is trivial to verify that $\mathcal{H}_0$ and $\mathcal{H}_1$ are indistinguishable. Since $\mathcal{S}_0$ holds x_{A} and $\mathsf{seed}^{\mathsf{A}}$, it can perfectly emulate $\mathbf{P_A}$'s behavior.

Hybrid $\mathcal{H}_2$: We consider a simulator $\mathcal{S}_1$, that works exactly as $\mathcal{S}_0$, except that it aborts if it receives a modified ciphertext $\mathsf{ct}^{\mathsf{A},\mathsf{GC}}$, denoted by $\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}_{\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}},\mathcal{S}_1(x_{\mathsf{A}},\mathsf{seed}^{\mathsf{A}}),\mathcal{Z}}$.

The indistinguishability of $\mathcal{H}_2$ and $\mathcal{H}_1$ is derived from the security of the signature scheme SIG with the adversarial distinguishing advantage $\mathbf{Adv}^{\mathsf{SIG}}$. An environment $\mathcal{Z}$, which can distinguishes between $\mathcal{H}_2$ and $\mathcal{H}_1$, can be turned into an adversary $\mathcal{B}_0$ against the security of the signature scheme SIG scheme.

Hybrid $\mathcal{H}_3$: We consider a simulator $\mathcal{S}_2$ that works exactly as $\mathcal{S}_1$, except using $\mathbf{0}$ instead of using the correct $\mathsf{seed}^{\mathsf{A}}$ to compute the evidence $\mathsf{ct}^{\mathsf{A},\mathsf{GC}}$, denoted by $\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}_{\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}},\mathcal{S}_2(x_{\mathsf{A}},\mathsf{seed}^{\mathsf{A}}),\mathcal{Z}}$. Note that $\mathcal{S}$ still computes GC exactly as in the real execution.

The indistinguishability of $\mathcal{H}_3$ and $\mathcal{H}_2$ is derived from the security of the SKE scheme Π with adversarial distinguishing advantage $\mathbf{Adv}^{\mathsf{SKE}}$. An environment $\mathcal{Z}$, which can distinguishes between $\mathcal{H}_3$ and $\mathcal{H}_2$, can be turned into an adversary

$\mathcal{B}_1$ against the security of the SKE Π scheme.

Hybrid $\mathcal{H}_4$: We consider a simulator $\mathcal{S}_3$ that works exactly as $\mathcal{S}_2$, except that $\mathcal{S}^3$ does not hold x_A and seed^A, but instead garbles the circuit $\mathcal{C}$ in such a way that this circuit will always be evaluated to o^B, using the appropriate input wire labels. We denote such case as $\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}_{\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}},\mathcal{S}_3,\mathcal{Z}}$, which is identical to $\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}_{\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}},\mathcal{S},\mathcal{Z}}$.

The $\mathcal{H}_4$ and $\mathcal{H}_3$ are indistinguishable, if GCS is simulatable private with adversarial distinguishing advantage $\mathbf{Adv}^{\mathsf{GCS}}$. Again, an environment $\mathcal{Z}$, which can distinguishes between $\mathcal{H}_4$ and $\mathcal{H}_3$, can be turned into an adversary $\mathcal{B}_2$ against the security of the GCS scheme.

Recall that the simulation of $\mathbf{P_{Ar}}$ by $\mathcal{S}$ is perfect in the CRS model. This completes the proof. □

Simulating the case $\mathbf{P_A}$ is corrupted. $\mathcal{S}$ simulates a real execution in which the corrupted $\mathbf{P_A}$ controlled by $\mathcal{A}$ delivers message to uncorrupted $\mathbf{P_B}$ in the internal (simulated) interaction. The $\mathcal{S}$ works as follows:

1. In the key setup stage, plays the role of an honest $\mathbf{P_{Ar}}$, sends $(\mathsf{key}^\mathsf{Ar}, \mathbf{h}, \mathsf{decom}, \sigma^\mathsf{Ar})$ to $\mathcal{A}$. Then plays role of an honest $\mathbf{P_B}$, receives $(\mathbf{h}, \sigma^\mathsf{Ar})$, checks whether σ^Ar is valid, aborts if not.
2. Plays the $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$ with $\mathcal{A}$ playing $\mathbf{P_A}$:
   - If the input is $(\mathbf{cheat}, \mathbf{P_A}, \mathsf{sid})$, sends $(\mathbf{cheat}, \mathbf{P_A}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ and receives back $\mathbf{P_B}$'s input x_B. Then uses the input x_B of $\mathbf{P_B}$ to perfectly emulate $\mathbf{P_B}$ in rest of the execution. Sends o^B to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$, which will be received by $\mathbf{P_B}$ as output. The simulation ends here for this case.
   - If the input is $\{B_{i,b}\}_{i\in[n_2],b\in\{0,1\}}$, then proceeds with the simulation below.
3. Plays the role as an honest $\mathbf{P_B}$, receives GC, table, $\mathsf{ct}^{\mathsf{A,GC}}$ and $\sigma^{\mathsf{A,GC}}$ from $\mathcal{A}$, compute $\mathcal{H}$.
4. Checks if the signature $\sigma^{\mathsf{A,GC}}$ is invalid, sends $\perp$ to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ and halts.
5. Extracts seed^A with $\Pi.\mathsf{DEC}(\mathsf{key}^\mathsf{Ar}, \mathsf{ct}^{\mathsf{A,GC}})$. Computes $\hat{\mathsf{GC}}$ and $\hat{\mathsf{table}}$ using the pseudorandomness derived from seed^A.
6. Computes the claimed hash value $\hat{\mathcal{H}}$.
7. For the following cases:
   - If $\hat{\mathcal{H}} \neq \mathcal{H}$, sends $(\mathbf{cheat}, \mathbf{P_A}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ and receives back $\mathbf{P_B}$'s input x_B. Then uses the input x_B of $\mathbf{P_B}$ to perfectly emulate $\mathbf{P_B}$ in rest of the execution. Sends o^B to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$, which will be received by $\mathbf{P_B}$ as output. The simulation ends here for this case.
   - If any of $\{B_{i,b}\}_{i\in[n_2],b\in\{0,1\}}$ is not a consistent label corresponding to $\hat{\mathsf{GC}}$, sends $(\mathbf{cheat}, \mathbf{P_A}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ and receives back $\mathbf{P_B}$'s input x_B. Then runs perfect emulation using $\mathbf{P_B}$'s input. Sends o^B to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$, which will be received by $\mathbf{P_B}$ as output. The simulation ends here for this case.
8. Upon receiving $\{A_{i,j,x_\mathsf{A}[j]}\}$ from $\mathcal{A}$, checks if any $\{A_{i,j,x_\mathsf{A}[j]}\}$ are invalid, sends ξ to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ and halts (cause $\mathbf{P_B}$ to receive $\perp$). Otherwise, derives the exact $\mathcal{A}$'s input x_A, sends to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ and halts.

Proof. We now prove that $\mathcal{Z}$ cannot distinguish an interaction of protocol Π^{GC} with $\mathcal{A}$ (denoted by $\mathsf{REAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}_{\Pi^{\mathsf{GC}},\mathcal{A},\mathcal{Z}}$) from an interaction in the ideal process with $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$ and $\mathcal{S}$ (denoted by $\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}_{\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}},\mathcal{S},\mathcal{Z}}$):

Case 1: $\mathcal{A}$ sends $(\mathbf{cheat}, \mathbf{P_B}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$. It is the same to the previous proof of **Case 1**, that $\mathcal{Z}$ cannot distinguish $\mathsf{REAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}_{\Pi^{\mathsf{GC}},\mathcal{A},\mathcal{Z}}$ with $\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}_{\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}},\mathcal{S},\mathcal{Z}}$, because $\mathcal{S}$ receives back the exact honest $\mathbf{P_B}$'s input x_{B} at the very beginning and can thus perfectly emulate the $\mathbf{P_B}$'s behavior ($\mathbf{P_{Ar}}$'s behavior as well).

Case 2: $\mathcal{A}$ sends an incorrect GC or hands any incorrect labels $\{B_{i,b}\}$ to $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$. In this case, both $\mathsf{REAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}_{\Pi^{\mathsf{GC}},\mathcal{A},\mathcal{Z}}$ and $\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}_{\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}},\mathcal{S},\mathcal{Z}}$ are still indistinguishable to $\mathcal{Z}$. Recall that $\mathcal{S}$ can decrypt the evidence sent from $\mathcal{A}$ and can thus perfectly emulate the arbitration performed by $\mathbf{P_{Ar}}$ and the messages sent from $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$ to $\mathbf{P_{Ar}}$. Note that $\mathcal{A}$ does not have output during the interaction with $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$ and is not aware of $\mathbf{P_B}$'s input x_{B} sent to $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$, neither does $\mathcal{Z}$. After receiving $\mathbf{P_B}$'s input x_{B} from $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$, $\mathcal{S}$ can still perfectly emulate the $\mathbf{P_B}$'s behavior.

Case 3: $\mathcal{A}$ bahaves honestly. In this case, recall again that $\mathcal{S}$ can decrypt the evidence sent by $\mathbf{P_A}$ and thus extract the $\mathcal{A}$'s input x_{A} by comparing the wire labels sent by $\mathcal{A}$ and the wire labels of garbled circuit using the decrypted $\mathsf{seed}^{\mathsf{A}}$.

This completes the proof. □

Simulating the case $\mathbf{P_{Ar}}$ is corrupted. For simplicity, we consider the case that $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$ is implemented by a protocol such that $\mathbf{P_B}$'s misbehavior can be detected by $\mathbf{P_A}$ within the ideal functionality $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$. So from $\mathbf{P_{Ar}}$'s point of view, we only focus on whether $\mathbf{P_A}$'s dishonest behavior takes place in the rest of the proof. Besides, $\mathbf{P_{Ar}}$ only receives messages from $\mathbf{P_B}$ and does not forward any message to both $\mathbf{P_A}$ and $\mathbf{P_B}$, so $\mathcal{S}$ only have to simulate the GC process proceeded by both simulated $\mathbf{P_A}$ and $\mathbf{P_B}$. The $\mathcal{S}$ works as follows:

1. In the key setup stage, $\mathcal{S}$ acts an honest $\mathbf{P_A}$ and receives $(\mathsf{key}^{\mathsf{Ar}}, \mathbf{h}, \mathsf{decom}, \sigma^{\mathsf{Ar}})$ from $\mathcal{A}$, checks whether σ^{Ar} and $\mathbf{h}$ are both valid, aborts if not. Then $\mathcal{S}$ plays the role of an honest $\mathbf{P_B}$ and receives $(\mathbf{h}, \sigma^{\mathsf{Ar}})$.
2. As $\mathbf{P_A}$, $\mathcal{S}$ chooses uniform distributed κ-bit $\mathsf{seed}^{\mathsf{A}}$ to compute $\{B_{i,b}\}_{i\in[n_2],b\in\{0,1\}}$.
3. $\mathcal{S}$ plays $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$ between $\mathbf{P_A}$ and $\mathbf{P_B}$, where $\mathbf{P_A}$ uses $\{B_{i,b}\}$ and $\mathbf{P_B}$ uses a uniformly distributed x_{B} as input.
4. $\mathcal{S}$ uses $\mathsf{seed}^{\mathsf{A}}$ to garble circuit $\mathcal{C}$ and thus receives GC and table. Then $\mathcal{S}$ computes $\mathsf{ct}^{\mathsf{A,GC}}$ and $\sigma^{\mathsf{A,GC}}$.
5. $\mathcal{S}$ plays the role of $\mathbf{P_B}$ and sends the arbitrate query $(\mathcal{C}, \mathcal{H}, \mathsf{ct}^{\mathsf{A,GC}}, \sigma^{\mathsf{A,GC}}, \mathsf{sid})$ to $\mathcal{A}$.

6. $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$ by receiving $\mathbf{P_B}$'s arbitrate query, which sends $\mathbf{P_A}$'s input $\{B_{i,b}\}$ to $\mathcal{A}$ and halts.

Proof. We argue that $\mathsf{REAL}_{\Pi^{\mathsf{GC}},\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}$ is indistinguishable from $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}},\mathcal{S},\mathcal{Z}}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}}$. Although $\mathbf{P_{Ar}}$ is corrupted by a malicious $\mathcal{A}$, it however does not have any input to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{2pc}}$. In both ideal simulation and real protocol execution, $\mathcal{S}$ and $\mathbf{P_A}$ use a uniformly random κ-bit $\mathsf{seed}^{\mathsf{A}}$ to garble the circuit, and meanwhile generate $\mathbf{P_B}$'s wire label $\{B_{i,b}\}$ as input to $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$. The generated hash value on GC is indistinguishable based on $\mathsf{H}()$. In addition, $\mathcal{S}$ can simulate $\mathbf{P_A}$'s signature under the CRS model, we conclude both ideal and real executions are indistinguishable. □

7.2 Security in $\Pi_{\mathsf{BA,S}}^{\mathsf{OTE}}$

Theorem 2. *Assume* $\mathsf{H}()$ *is a correlation-robust cryptographic hash function,* G *is a pseudo random generator,* SIG *is existentially unforgeable under chosen-message attacks (EUF-CMA). Protocol* $\Pi_{\mathsf{BA,S}}^{\mathsf{OTE}}$ *described in Fig. 7 UC-realizes* $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$ *described in Fig. 3 in the* $\mathcal{F}^{\mathsf{CRS}}$, $\mathcal{F}_{\mathsf{BA,R}}^{\mathsf{l}\times\mathsf{OT}}$*-hybrid model in the presence of a malicious adversary who can corrupt either* $\mathbf{P_A}$, $\mathbf{P_B}$ *or* $\mathbf{P_{Ar}}$, *with static corruption.*

Due to space limitation, we provide a proof sketch in this section.

Malicious $\mathbf{P_A}$ The original paper [55] already provides security against malicious $\mathbf{P_A}$ without the steps * and **. We observe that $\mathbf{P_A}$ does not receive any additional message during these two steps. It can only cause $\mathbf{P_B}$ to abort by sending an incorrect signature, which does not help $\mathbf{P_A}$ to learn any single bit of $\mathbf{P_B}$'s input. Thus, we do not provide further proof and directly derive security against malicious $\mathbf{P_A}$ from the original paper.

Malicious $\mathbf{P_B}$ Same as above, the original proof shows that the protocol without the steps * and ** is secure against malicious $\mathbf{P_B}$ [55]. Since SIG is unforgeable under CMA, receiving $\sigma^{\mathsf{A,OT}}$ does not provide malicious $\mathbf{P_B}$ with more information or new ability. Specifically, malicious $\mathbf{P_B}$ cannot cause $\mathbf{P_{Ar}}$ to output an incorrect result, since $\mathbf{P_B}$ cannot forge $\mathbf{P_A}$'s signature and force $\mathbf{P_{Ar}}$ to accept some unmatched $\{\mathbf{u}^i\}$. Again, we conclude with the malicious security from the original paper.

Malicious $\mathbf{P_{Ar}}$ In the CRS model, the simulator $\mathcal{S}$ can forge $\mathbf{P_B}$'s signature by replacing the signing key pair of $\mathbf{P_B}$ with its own key pair. Upon receiving $\mathbf{P_A}$'s input $\{(x_j^0, x_j^1)\}_{j\in[m]}$ from $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m}\times\mathsf{OT}}$, $\mathcal{S}$ can perfectly simulate the arbitrate query sent from $\mathbf{P_B}$ to $\mathcal{A}$ by choosing $\mathbf{s}$, $\{k_0, k_1\}^l$, and $\mathbf{r}$ uniformly at random, then computing the corresponding $\{\mathbf{u}^i\}$ and $\{y_j^0, y_j^1\}_{j\in[m]}$.

7.3 Security in $\Pi_{\mathsf{LA}}^{\mathsf{OTE}}$

Theorem 3. *Assume* $\mathsf{H}()$ *is a correlation-robust cryptographic hash function,* G *is a pseudo random generator,* SIG *is existentially unforgeable under chosen-message attacks (EUF-CMA). Protocol* $\Pi_{\mathsf{LA}}^{\mathsf{OTE}}$ *described in Fig.6 UC-realizes* $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$ *in the* $\mathcal{F}^{\mathsf{CRS}}$, $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{l}\times\mathsf{OT}}$*-hybrid model in the presence of a malicious adversary who can corrupt either* $\mathbf{P_A}$, $\mathbf{P_B}$ *or* $\mathbf{P_{Ar}}$*, with static corruption.*

In this section, we prove that our protocol $\Pi_{\mathsf{LA}}^{\mathsf{OTE}}$ is secure against malicious $\mathbf{P_B}$. And we provide a proof sketch for both malicious $\mathbf{P_A}$ and $\mathbf{P_{Ar}}$ in Appendix B.

Malicious $\mathbf{P_B}$ We construct $\mathcal{S}$ simulating a real execution in which the corrupted $\mathbf{P_B}$ controlled by $\mathcal{A}$ delivers message to uncorrupted $\mathbf{P_A}$ in the internal (simulated) interaction. The $\mathcal{S}$ works as follows:

1. $\mathcal{S}$ plays the $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{l}\times\mathsf{OT}}$ with $\mathcal{A}$ playing $\mathbf{P_B}$:
   - If input is $(\mathbf{cheat}, \mathbf{P_B}, \mathsf{sid})$, $\mathcal{S}$ sends $(\mathbf{cheat}, \mathbf{P_B}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{l}\times\mathsf{OT}}$ and receives back $\mathbf{P_A}$'s input $\{x_j^0, x_j^1\}^m$. Then $\mathcal{S}$ uses the real $\mathbf{P_A}$'s input to perfectly emulate $\mathbf{P_A}$ in the rest of execution. $\mathcal{S}$ sends o^A to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$, which will be received by $\mathbf{P_A}$ as output. The simulation ends here for this case.
   - If input is $\{k_0, k_1\}^l$, $\mathcal{S}$ proceeds with the simulation below.
2. $\mathcal{S}$ plays the role as an honest $\mathbf{P_A}$, receives $\{\mathbf{u}^i\}^l$, Δ, $\sigma^{\mathsf{B,OT}}$, checks whether $\sigma^{\mathsf{B,OT}}$ is invalid, sends $\perp$ to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$ and halt.
3. For all $i \in [l]$, $\mathcal{S}$ computes:
$$\mathbf{r}^i \leftarrow \mathbf{u}^i \oplus \mathsf{G}(k_0^i) \oplus \mathsf{G}(k_1^i)$$
4. For the following case:
   - If any $\mathbf{r}^i$ is inconsistent with others, $\mathcal{S}$ sends $(\mathbf{cheat}, \mathbf{P_B}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$ and receives back $\mathbf{P_A}$'s input $\{x_j^0, x_j^1\}^m$. Then $\mathcal{S}$ uses the real $\mathbf{P_A}$'s input to perfectly emulate $\mathbf{P_A}$ in the rest of execution. $\mathcal{S}$ sends o^A to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$, which will be received by $\mathbf{P_A}$ as output. The simulation ends here for this case.
   - If all $\mathbf{r}^i$ are consistent, $\mathcal{S}$ proceeds with the simulation below.
5. $\mathcal{S}$ extracts $\mathcal{A}$'s input $\mathbf{r}$ and sends to $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$. Upon receiving $\{x^{r_j}\}^m$, $\mathcal{S}$ uses $\{u^i\}^l$, s, $\{k_{s_i}\}^l$ and received $\{x^{r_j}\}^m$ to compute $\{y^{r_j}\}^m$, sets $\{y^{\overline{r_j}}\}^m$ uniformly at random. $\mathcal{S}$ sends $\{y^b\}^m$ to $\mathcal{A}$ and halts.

Proof. We now prove that $\mathcal{Z}$ cannot distinguish an interaction of $\Pi_{\mathsf{LA}}^{\mathsf{OTE}}$ described in 4 with $\mathcal{A}$ (denoted by $\mathsf{REAL}_{\Pi_{\mathsf{LA}}^{\mathsf{OTE}},\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{l}\times\mathsf{OT}}}$) from an interaction in the ideal process with $\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}}$ and $\mathcal{S}$ (denoted by $\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}},\mathcal{S},\mathcal{Z}}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{l}\times\mathsf{OT}}}$):

Case 1: Either when $\mathcal{A}$ sends $(\mathbf{cheat}, \mathbf{P_B}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{l}\times\mathsf{OT}}$, or $\mathcal{S}$ finds that $\mathcal{A}$ cheats by using inconsistent $\mathbf{r}^i$, $\mathcal{S}$ receives back the exact honest $\mathbf{P_A}$'s input

$\{(x_j^0, x_j^1)\}_{j\in[m]}$. For any of above cases, $\mathcal{A}$ hasn't received any messages yet during the protocol, thus $\mathcal{S}$ can perfectly emulate the $\mathbf{P_A}$'s behavior in the rest of the protocol execution ($\mathbf{P_{Ar}}$'s behavior as well). This yields the indistinguishability of $\mathsf{REAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{l}\times\mathsf{OT}}}_{\Pi_{\mathsf{LA}}^{\mathsf{OTE}},\mathcal{A},\mathcal{Z}}$ and $\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{l}\times\mathsf{OT}}}_{\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}},\mathcal{S},\mathcal{Z}}$ to $\mathcal{Z}$.

Case 2: For the second case when $\mathcal{A}$ behaves honestly, we consider the following hybrid worlds:

Hybrid $\mathcal{H}_0$: It is the real protocol execution $\mathsf{REAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{l}\times\mathsf{OT}}}_{\Pi_{\mathsf{LA}}^{\mathsf{OTE}},\mathcal{A},\mathcal{Z}}$.

Hybrid $\mathcal{H}_1$: We consider a simulator $\mathcal{S}^0$, which holds the real $\mathbf{P_A}$'s private input $\{(x_j^0, x_j^1)\}_{j\in[m]}$, denoted by $\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{l}\times\mathsf{OT}}}_{\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}},\mathcal{S}^0(\{(x_j^0,x_j^1)\}),\mathcal{Z}}$.

Hybrid $\mathcal{H}_2$: We consider a simulator $\mathcal{S}^1$ that works exactly as $\mathcal{S}^0$, except using the received $\{x^{r_j}\}$ instead of using the real $\mathbf{P_A}$'s private input $\{(x_j^0, x_j^1)\}_{j\in[m]}$, denoted by $\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{l}\times\mathsf{OT}}}_{\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}},\mathcal{S}^1,\mathcal{Z}}$, which is exactly the same as $\mathsf{IDEAL}^{\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{l}\times\mathsf{OT}}}_{\mathcal{F}_{\mathsf{LA}}^{\mathsf{m}\times\mathsf{OT}},\mathcal{S},\mathcal{Z}}$.

It is trivial to verify that $\mathcal{H}_0$ and $\mathcal{H}_1$ are indistinguishable. Since $\mathcal{S}$ holds $\{(x_j^0, x_j^1)\}_{j\in[m]}$, it can perfectly emulate $\mathbf{P_A}$'s behavior.

Now we show that $\mathcal{H}_2$ and $\mathcal{H}_1$ are computationally indistinguishable to $\mathcal{A}$. Since $\mathcal{A}$ does not know the choice bit $\mathbf{s}$, for each $j \in [l]$, due to the functionality of protocol, $\mathcal{A}$ can only compute either $\mathsf{H}(j, q_j)$ or $\mathsf{H}(j, q_j \oplus \mathbf{s})$ depends on its choice bit r_j, and the remaining hash is uniform distributed to $\mathcal{A}$. Since $\mathcal{A}$ does not know $x^{\overline{r_j}}$ either, $y^{\overline{r_j}} \leftarrow x^{\overline{r_j}} \oplus \mathsf{H}^{\overline{r_j}}$ is uniform distributed to $\mathcal{A}$ as well.

This completes the proof. □

8 Evaluation

8.1 Evalution Setup

Testbed environment. All experiments are executed in a single server with separate processes for $\mathbf{P_A}$ and $\mathbf{P_B}$, with an additional process for $\mathbf{P_{Ar}}$. The server runs Ubuntu Server 22.04 LTS and has two Intel Xeon CPUs (8360Y @ 2.40GHz). All programs run with a single thread. In the LAN setting, the network bandwidth is 1 Gbps and the average latency is 0.2 ms. In the WAN setting, the network bandwidth is 100 Mbps and the average latency is 40 ms. Both are simulated with `tc` [44]. We have never met any issue with the memory usage.

Baseline. To validate the efficiency of our protocols, we implement them in the open source framework `emp-toolkit` [80], and compare them against baseline implementations included in `emp-toolkit`. More specifically, `emp-sh2pc` (*Semi-honest*) at commit `61589f5`, `emp-pvc` (*PVC*) at commit `7c75a85` and a modified version of `emp-ag2pc` (*Malicious*) at commit `eddb6bf`.

Experiment parameters. We set the security parameter $\kappa = 128$ in our implementation. We implement our protocol Π^{GC} and Π^{OT} with the state-of-the-art techniques for garbling [58] [84]. We use SHA-256 for the hash function provided by openssl [1] instead of Free Hash mentioned in PVC [45], as Guo et al. [41] pointed out lately that this instantiation of the hash function was not collision resistant. As for signature scheme, we choose the standard ECDSA implementation provided by openssl as well.

Benchmark. To benchmark the running time, we perform each protocol for 10 times. Each time, we use the longest time of all parties as the running time of that run. The average running time among the experiments is presented in Table 4 and Table 5. We also count the total communication volume of $\mathbf{P_B}$, which includes both inbound and outbound traffic. In our case, it sums up $\mathbf{P_B}$'s communication with $\mathbf{P_A}$, as well as with $\mathbf{P_{Ar}}$. The statistics is shown in Table 6.

Experiment circuits. The circuits used for evaluation are listed in Table 2, where n_1 denotes the number of $\mathbf{P_A}$'s input wires, n_2 the number of $\mathbf{P_B}$'s input wires, n_3 the number of output wires, and $|\mathcal{C}|$ the number of AND gates.

Table 2. Circuits for evaluation. Overall n_2 OTs are required for each circuit.

Circuit	n_1	n_2	n_3	$\lvert\mathcal{C}\rvert$
AES-128	128	128	128	6,800
SHA-256	512	256	256	22,573
SHA-512	1,024	512	512	57,947
Mult.	2,048	2,048	2,048	4,192K
Hamming dist. (Ham.)	1,048K	1,048K	22	10,223K

Table 3. Relative slowdown or speedup between our protocol and other protocols in LAN setting and WAN setting.

Circuit	Slowdown Se.-ho.		Speedup PVC [45]		Speedup Malicious [54]	
	LAN	WAN	LAN	WAN	LAN	WAN
AES-128	41.17%	28.69%	1.53×	2.37×	2.62×	3.16×
SHA-256	27.64%	23.57%	1.88×	2.34×	5.63×	4.89×
SHA-512	18.88%	17.56%	1.52×	2.09×	9.56×	7.32×
Mult.	23.80%	4.42%	1.11×	1.04×	9.65×	13.30×
Ham.	73.43%	45.58%	1.09×	1.39×	4.55×	6.04×

Table 4. Comparison of the running time (in milliseconds) of all protocols in **LAN** setting.

Circuit	Se.-ho.	**This paper**	PVC [45]	Malicious [54]
AES-128	23.73	**33.50**	51.32	87.78
SHA-256	28.26	**36.07**	67.68	202.95
SHA-512	37.77	**44.90**	68.18	429.36
Mult.	1,513	**1,874**	2,078	18,089
Ham.	1,354	**2349**	2,550	10,682

Table 5. Comparison of the running time (in milliseconds) of all protocols in **WAN** setting.

Circuit	Se.-ho.	**This paper**	PVC [45]	Malicious [54]
AES-128	309.17	**397.87**	942.93	1,257
SHA-256	347.45	**429.35**	1,006	2,097
SHA-512	447.26	**525.81**	1,098	3,849
Mult.	11,429	**11,934**	12,452	158,778
Ham.	10,169	**14,804**	20,573	89,464

Table 6. Communication complexity in MB.

Circuit	Se.-ho.	**This paper**	PVC [45]	Malicious [54]
AES-128	0.21	**0.24**	0.75	0.27
SHA-256	0.71	**0.75**	1.27	0.93
SHA-512	1.80	**1.85**	2.40	2.39
Mult.	128.04	**128.16**	128.67	170.03
Ham.	112.01	**160.04**	176.54	129.00

8.2 Comparisons

Compared to the semi-honest protocol. In Table 4 and Table 5, we show the running time of our protocol for each circuit compared with that against semi-honest adversaries in LAN and WAN settings. For the semi-honest protocol, the results contain the running time of a base OT protocol and a passively secure OT extension protocol [7]. For our protocol, we follow the constructions described in Section 5 with an OT extension protocol $\Pi^{\mathsf{OTE}}_{\mathsf{BA,S}}$ described in Section 6.1. As shown in Table 3, the slowdown factor of our protocol comparing to the semi-honest protocol never exceeds 2.

Compared to the PVC protocol. hen we compare the running time of our protocol to the PVC protocol [45] with a deterrence factor $\epsilon = 1/2$. We show in both Table 4 and Table 5 that achieving *honorific security* costs much less than achieving covert security. Our protocol performs up to 1.88 times faster in LAN setting and 2.37 times faster in WAN setting than the PVC protocol. To run a PVC protocol, both garbler and evaluator have to jointly perform $2 * \lambda - 1$ times garbling scheme and $\lambda * n_2$ OTs (recall that $\epsilon = 1 - \frac{1}{\lambda}$), while only two times

garbling scheme and n_2 OTs are needed in our protocol. We point out that if we choose to execute the PVC protocol with some lager ϵ, the boost factor brought by honorific security will be more effective.

Compared to the malicious protocol. In Table 4 and Table 5 we list the performance of running a state-of-the-art malicious protocol [54] ([24] is proposed without implementation). As shown in Table 3, our protocol beats the malicious protocol with at least a 2.85 times acceleration in LAN setting and a 3.56 times acceleration in WAN setting. Remark that if we omit the computation and communication overhead of an arbitration by letting the parties only hold the received evidences as a deterrence, we can achieve a even better performance.

Communication overhead The communication overhead of this paper compared against the PVC protocol [45] and the malicious protocol [54] is documented in Table 6. As we expected, the communication volume of our protocol is much closer to the semi-honest protocol.

9 Conclusion and Future Work

In this paper, we propose a new security notion *honorific security* in the UC framework. By constructing an efficient OT protocol and an efficient GC-based 2PC protocol with provable security, we show that this notion provides sufficient security guarantee and implies high efficiency. We believe that it is extremely meaningful to construct other 2PC protocols to achieve honorific security in the future, such as sharing-based 2PC protocols.

Bibliography

[1] Openssl (2018), https://github.com/openssl/openssl/tree/OpenSSL_1_1_1-stable

[2] Supporting private data on hyperledger fabric with secure multiparty computation. IBM Journal of Research and Development **63**(2/3), 3–1 (2019)

[3] Adida, B.: Helios: Web-based open-audit voting. In: USENIX security symposium. vol. 17, pp. 335–348 (2008)

[4] Alexandru, A.B., Morari, M., Pappas, G.J.: Cloud-based mpc with encrypted data. In: 2018 IEEE conference on decision and control (CDC). pp. 5014–5019. IEEE (2018)

[5] Anati, I., Gueron, S., Johnson, S., Scarlata, V.: Innovative technology for cpu based attestation and sealing. In: Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy. vol. 13, p. 7. Citeseer (2013)

[6] Archer, D.W., Bogdanov, D., Lindell, Y., Kamm, L., Nielsen, K., Pagter, J.I., Smart, N.P., Wright, R.N.: From keys to databases—real-world applications of secure multi-party computation. The Computer Journal **61**(12), 1749–1771 (2018)

[7] Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 535–548 (2013)

[8] Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer extensions with security for malicious adversaries. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 673–701. Springer (2015)

[9] Asharov, G., Orlandi, C.: Calling out cheaters: Covert security with public verifiability. In: Wang, X., Sako, K. (eds.) Advances in Cryptology – ASIACRYPT 2012. pp. 681–698. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)

[10] Aumann, Y., Lindell, Y.: Security against covert adversaries: Efficient protocols for realistic adversaries. vol. 23, pp. 281–343. Springer (2010)

[11] Badrinarayanan, S., Jain, A., Ostrovsky, R., Visconti, I.: Uc-secure multiparty computation from one-way functions using stateless tokens. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 577–605. Springer (2019)

[12] Baum, C., Damgård, I., Orlandi, C.: Publicly auditable secure multi-party computation. In: International Conference on Security and Cryptography for Networks. pp. 175–196. Springer (2014)

[13] Baum, C., Orsini, E., Scholl, P.: Efficient secure multiparty computation with identifiable abort. In: Theory of Cryptography: 14th International Conference, TCC 2016-B, Beijing, China, October 31-November 3, 2016, Proceedings, Part I 14. pp. 461–490. Springer (2016)

[14] Baum, C., Orsini, E., Scholl, P., Soria-Vazquez, E.: Efficient constant-round mpc with identifiable abort and public verifiability. In: Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II. pp. 562–592. Springer (2020)
[15] Beaver, D.: Commodity-based cryptography. In: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing. pp. 446–455 (1997)
[16] Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Proceedings of the 2012 ACM conference on Computer and communications security. pp. 784–796 (2012)
[17] Bestavros, A., Lapets, A., Varia, M.: User-centric distributed solutions for privacy-preserving analytics. Communications of the ACM **60**(2), 37–39 (2017)
[18] Byali, M., Chaudhari, H., Patra, A., Suresh, A.: Flash: fast and robust framework for privacy-preserving machine learning. Cryptology ePrint Archive (2019)
[19] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. pp. 136–145. IEEE (2001)
[20] Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. pp. 494–503 (2002)
[21] Cartlidge, J., Smart, N.P., Talibi Alaoui, Y.: Mpc joins the dark side. In: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security. pp. 148–159 (2019)
[22] Chandran, N., Gupta, D., Obbattu, S.L.B., Shah, A.: {SIMC}:{ML} inference secure against malicious clients at {Semi-Honest} cost. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 1361–1378 (2022)
[23] Cordi, C., Frank, M.P., Gabert, K., Helinski, C., Kao, R.C., Kolesnikov, V., Ladha, A., Pattengale, N.: Auditable, available and resilient private computation on the blockchain via mpc. In: International Symposium on Cyber Security, Cryptology, and Machine Learning. pp. 281–299. Springer (2022)
[24] Cui, H., Wang, X., Yang, K., Yu, Y.: Actively secure half-gates with minimum overhead under duplex networks. In: Advances in Cryptology–EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23–27, 2023, Proceedings, Part II. pp. 35–67. Springer (2023)
[25] Dalskov, A., Escudero, D., Keller, M.: Secure evaluation of quantized neural networks. Proceedings on Privacy Enhancing Technologies **4**, 355–375 (2020)
[26] Damgård, I., Geisler, M., Nielsen, J.B.: From passive to covert security at low cost. In: Theory of Cryptography Conference. pp. 128–145. Springer (2010)
[27] Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Annual Cryptology Conference. pp. 643–662. Springer (2012)

[28] Demmler, D., Schneider, T., Zohner, M.: Aby-a framework for efficient mixed-protocol secure two-party computation. In: NDSS (2015)
[29] Dittmer, S., Ishai, Y., Lu, S., Ostrovsky, R.: Authenticated garbling from simple correlations. In: Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part IV. pp. 57–87. Springer (2022)
[30] Dong, C., Weng, J., Liu, J., Zhang, Y., Tong, Y., Yang, A., Cheng, Y., Hu, S.: Fusion: Efficient and secure inference resilient to malicious servers. In: 30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023. The Internet Society (2023)
[31] Evans, D., Kolesnikov, V., Rosulek, M.: A pragmatic introduction to secure multi-party computation. Foundations and Trends® in Privacy and Security **2**(2-3) (2017)
[32] Evans, D., Kolesnikov, V., Rosulek, M., et al.: A pragmatic introduction to secure multi-party computation. Foundations and Trends® in Privacy and Security **2**(2-3), 70–246 (2018)
[33] Felsen, S., Kiss, Á., Schneider, T., Weinert, C.: Secure and private function evaluation with intel sgx. In: Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop. pp. 165–181 (2019)
[34] Fu, C., Zhang, X., Ji, S., Chen, J., Wu, J., Guo, S., Zhou, J., Liu, A.X., Wang, T.: Label inference attacks against vertical federated learning. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 1397–1414 (2022)
[35] Gao, H., Ma, Z., Luo, S., Wang, Z.: Bfr-mpc: a blockchain-based fair and robust multi-party computation scheme. IEEE access **7**, 110439–110450 (2019)
[36] Geiping, J., Bauermeister, H., Dröge, H., Moeller, M.: Inverting gradients-how easy is it to break privacy in federated learning? Advances in Neural Information Processing Systems **33**, 16937–16947 (2020)
[37] Goldreich, O.: Foundations of cryptography: volume 2, basic applications. Cambridge university press (2009)
[38] Goyal, V., Ishai, Y., Sahai, A., Venkatesan, R., Wadia, A.: Founding cryptography on tamper-proof hardware tokens. In: Theory of Cryptography Conference. pp. 308–326. Springer (2010)
[39] Goyal, V., Mohassel, P., Smith, A.: Efficient two party and multi party computation against covert adversaries. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 289–306. Springer (2008)
[40] Graf, M., Küsters, R., Rausch, D.: Auc: Accountable universal composability. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 1148–1167. IEEE (2023)
[41] Guo, C., Katz, J., Wang, X., Yu, Y.: Efficient and secure multiparty computation from fixed-key block ciphers. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 825–841. IEEE (2020)

[42] Hastings, M., Hemenway, B., Noble, D., Zdancewic, S.: Sok: General purpose compilers for secure multi-party computation. In: 2019 IEEE symposium on security and privacy (SP). pp. 1220–1237. IEEE (2019)
[43] Hazay, C., Scholl, P., Soria-Vazquez, E.: Low cost constant round mpc combining bmr and oblivious transfer. Journal of Cryptology **33**(4), 1732–1786 (2020)
[44] Hemminger, S., et al.: Network emulation with netem. In: Linux conf au. vol. 5, p. 2005. Citeseer (2005)
[45] Hong, C., Katz, J., Kolesnikov, V., Lu, W.j., Wang, X.: Covert security with public verifiability: Faster, leaner, and simpler. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 97–121. Springer (2019)
[46] Huang, Y., Gupta, S., Song, Z., Li, K., Arora, S.: Evaluating gradient inversion attacks and defenses in federated learning. Advances in Neural Information Processing Systems **34**, 7232–7241 (2021)
[47] Huang, Z., Lu, W.j., Hong, C., Ding, J.: Cheetah: Lean and fast secure two-party deep neural network inference. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 809–826 (2022)
[48] Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Annual International Cryptology Conference. pp. 145–161. Springer (2003)
[49] Ishai, Y., Ostrovsky, R., Zikas, V.: Secure multi-party computation with identifiable abort. In: Annual Cryptology Conference. pp. 369–386. Springer (2014)
[50] Jie, Y., Ren, Y., Wang, Q., Xie, Y., Zhang, C., Wei, L., Liu, J.: Multi-party secure computation with intel sgx for graph neural networks. In: ICC 2022-IEEE International Conference on Communications. pp. 528–533. IEEE (2022)
[51] Jin, X., Chen, P.Y., Hsu, C.Y., Yu, C.M., Chen, T.: Cafe: Catastrophic data leakage in vertical federated learning. Advances in Neural Information Processing Systems **34**, 994–1006 (2021)
[52] Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: Gazelle: A low latency framework for secure neural network inference. In: 27th USENIX Security Symposium (USENIX Security 18). pp. 1651–1669 (2018)
[53] Katz, J., Lindell, Y.: Introduction to modern cryptography. CRC press (2014)
[54] Katz, J., Ranellucci, S., Rosulek, M., Wang, X.: Optimizing authenticated garbling for faster secure two-party computation. In: Annual International Cryptology Conference. pp. 365–391. Springer (2018)
[55] Keller, M., Orsini, E., Scholl, P.: Actively secure ot extension with optimal overhead. In: Annual Cryptology Conference. pp. 724–741. Springer (2015)
[56] Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making spdz great again. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 158–189. Springer (2018)
[57] Kolesnikov, V., Malozemoff, A.J.: Public verifiability in the covert model (almost) for free. In: Iwata, T., Cheon, J.H. (eds.) Advances in Cryptol-

ogy – ASIACRYPT 2015. pp. 210–235. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)

[58] Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free xor gates and applications. In: International Colloquium on Automata, Languages, and Programming. pp. 486–498. Springer (2008)

[59] Konečnỳ, J., McMahan, H.B., Yu, F.X., Richtárik, P., Suresh, A.T., Bacon, D.: Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1610.05492 (2016)

[60] Koti, N., Kukkala, V.B., Patra, A., Raj Gopal, B.: Pentagod: Stepping beyond traditional god with five parties. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 1843–1856 (2022)

[61] Küsters, R., Liedtke, J., Müller, J., Rausch, D., Vogt, A.: Ordinos: a verifiable tally-hiding e-voting system. In: 2020 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 216–235. IEEE (2020)

[62] Küsters, R., Truderung, T., Vogt, A.: Accountability: definition and relationship to verifiability. In: Proceedings of the 17th ACM conference on Computer and communications security. pp. 526–535 (2010)

[63] Lehmkuhl, R., Mishra, P., Srinivasan, A., Popa, R.A.: Muse: Secure inference resilient to malicious clients. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 2201–2218 (2021)

[64] Li, P., Li, J., Huang, Z., Li, T., Gao, C.Z., Yiu, S.M., Chen, K.: Multi-key privacy-preserving deep learning in cloud computing. Future Generation Computer Systems **74**, 76–85 (2017)

[65] Lindell, Y.: Fast cut-and-choose-based protocols for malicious and covert adversaries. Journal of Cryptology **29**(2), 456–490 (2016)

[66] Lindell, Y.: How to simulate it–a tutorial on the simulation proof technique. Tutorials on the Foundations of Cryptography pp. 277–346 (2017)

[67] Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. Journal of Cryptology **28**(2), 312–350 (2015)

[68] Liu, J., He, X., Sun, R., Du, X., Guizani, M.: Privacy-preserving data sharing scheme with fl via mpc in financial permissioned blockchain. In: ICC 2021-IEEE International Conference on Communications. pp. 1–6. IEEE (2021)

[69] Liu, X., Deng, R.H., Yang, Y., Tran, H.N., Zhong, S.: Hybrid privacy-preserving clinical decision support system in fog–cloud computing. Future Generation Computer Systems **78**, 825–837 (2018)

[70] Lu, Y., Zhang, B., Zhou, H.S., Liu, W., Zhang, L., Ren, K.: Correlated randomness teleportation via semi-trusted hardware—enabling silent multi-party computation. In: European Symposium on Research in Computer Security. pp. 699–720. Springer (2021)

[71] Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., Popa, R.A.: Delphi: A cryptographic inference service for neural networks. In: 29th USENIX Security Symposium (USENIX Security 20). pp. 2505–2522 (2020)

[72] Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. In: 2017 IEEE symposium on security and privacy (SP). pp. 19–38. IEEE (2017)

[73] Nie, L., Yao, S., Liu, J.: Secure multiparty computation with identifiable abort and fairness. In: 2023 7th International Conference on Cryptography, Security and Privacy (CSP). pp. 99–106. IEEE (2023)

[74] Nordholt, P.S., Toft, T.: Confidential benchmarking based on multiparty computation. In: Financial Cryptography and Data Security: 20th International Conference, FC 2016, Christ Church, Barbados, February 22–26, 2016, Revised Selected Papers. vol. 9603, p. 169. Springer (2017)

[75] Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: Cryptflow2: Practical 2-party secure inference. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 325–342 (2020)

[76] Rivinius, M., Reisert, P., Rausch, D., Küsters, R.: Publicly accountable robust multi-party computation. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 2430–2449. IEEE (2022)

[77] Roy, L.: Softspokenot: Quieter ot extension from small-field silent vole in the minicrypt model. Springer-Verlag (2022)

[78] Scholl, P., Simkin, M., Siniscalchi, L.: Multiparty computation with covert security and public verifiability. Cryptology ePrint Archive (2021)

[79] So, J., Güler, B., Avestimehr, A.S.: Codedprivateml: A fast and privacy-preserving framework for distributed machine learning. IEEE Journal on Selected Areas in Information Theory **2**(1), 441–451 (2021)

[80] Wang, X., Malozemoff, A.J., Katz, J.: Emp-toolkit: Efficient multiparty computation toolkit (2022), `https://github.com/emp-toolkit/`

[81] Wang, X., Ranellucci, S., Katz, J.: Authenticated garbling and efficient maliciously secure two-party computation. In: Proceedings of the 2017 ACM SIGSAC conference on computer and communications security. pp. 21–37 (2017)

[82] Yang, K., Wang, X., Zhang, J.: More efficient mpc from improved triple generation and authenticated garbling. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1627–1646 (2020)

[83] Yao, A.C.C.: How to generate and exchange secrets. In: 27th annual symposium on foundations of computer science (Sfcs 1986). pp. 162–167. IEEE (1986)

[84] Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 220–250. Springer (2015)

[85] Zhou, J., Feng, Y., Wang, Z., Guo, D.: Using secure multi-party computation to protect privacy on a permissioned blockchain. Sensors **21**(4), 1540 (2021)

A Protocol $\Pi^{\mathsf{OTE}}_{\mathsf{BA,S}}$

See Fig. 7.

B Security in $\Pi^{\mathsf{OTE}}_{\mathsf{LA}}$

B.1 Malicious $\mathbf{P_A}$

The original paper [48] already provides security against a malicious $\mathbf{P_A}$. Since SIG is unforgeable under CMA, receiving two additional messages Δ and $\sigma^{\mathsf{B,OT}}$ does not provide a malicious $\mathbf{P_A}$ with more information or new ability. Besides, in the honest majority setting, a malicious $\mathbf{P_A}$ cannot cause $\mathbf{P_{Ar}}$ to output an incorrect result, since this only depends on the messages and $\sigma^{\mathsf{B,OT}}$ sent by $\mathbf{P_B}$. Thus we do not further prove security but directly derive security against a malicious $\mathbf{P_A}$ from the original paper.

B.2 Malicious $\mathbf{P_{Ar}}$

In the CRS model, remark that the simulator $\mathcal{S}$ can forge $\mathbf{P_B}$'s signature by replacing the signing key pair of $\mathbf{P_B}$ with its own key pair. While $\mathcal{S}$ does not hold the choice bit $\mathbf{r}$ of $\mathbf{P_B}$, we have to prove that $\{\mathbf{v}^i\}$ generated $\mathcal{S}$ is indistinguishable with those generated by an honest $\mathbf{P_B}$ in the real protocol execution. Note that the original $\mathbf{u}^i$ is distributed uniformly at random to $\mathcal{A}$. Since Δ is chosen uniformly at random as well, so the randomly chosen $\mathbf{v}^i$ by $\mathcal{S}$ in the ideal world is indistinguishable with the computed $\mathbf{v}^i \leftarrow \mathbf{u}^i \oplus \Delta$ by $\mathbf{P_B}$ in the real world. The indistinguishability of both ideal world and real protocol execution is thus proved.

Private inputs: $\mathbf{P_A}$ has input $\{(x_j^0, x_j^1)\}_{j\in[m]}$ and key pair $\{\mathsf{pk_A}, \mathsf{sk_A}\}$ for the signature scheme. $\mathbf{P_B}$ has input $\mathbf{r} = (r_1, ..., r_m)$ and key pair $\{\mathsf{pk_B}, \mathsf{sk_B}\}$ for the signature scheme. $\mathbf{P_{Ar}}$ has a key pair $\{\mathsf{pk_{Ar}}, \mathsf{sk_{Ar}}\}$ for the signature scheme.
Public inputs: All three parties know the public key pk_i of each other and a session ID sid.
CRS: $(\mathsf{pk_A}, \mathsf{pk_B}, \mathsf{pk_{Ar}})$.

Protocol:

1. $\mathbf{P_A}$ initializes a random vector $\mathbf{s} \in \{0,1\}^l$ and $\mathbf{P_B}$ chooses random $\{k_0, k_1\}^l$ each of size κ.
2. Parties proceed with a $\mathcal{F}_{\mathsf{BA,R}}^{\mathsf{l\times OT}}$ ($\mathcal{F}_{\mathsf{BA,R}}^{\mathsf{m\times OT}}$ with $\mathbf{m} = \mathbf{l}$), where $\mathbf{P_A}$ acts as the receiver with input $\mathbf{s}$ and $\mathbf{P_B}$ acts as the sender with input $\{k_0, k_1\}^l$.
3. $\mathbf{P_B}$ samples random w choice bits and extends $\mathbf{r}$ to $\mathbf{r'} = (r_1, ..., r_{m+w})$. Assume $m|w$, let $m' = m + w$. Let $T = [\mathbf{t}^1|...|\mathbf{t}^l]$ denote the $m' \times l$ matrix where ith column is $\mathbf{t}^i$. Let $\mathbf{t}_j$ denote the jth row of T. $\mathbf{P_B}$ uses the extended choice bits $\mathbf{r'}$ to compute $\mathbf{t}^i \leftarrow \mathsf{G}(k_0^i) \in \mathbb{Z}_2^{m'}$ and $\mathbf{u}^i \leftarrow \mathbf{t}^i \oplus \mathsf{G}(k_1^i) \oplus \mathbf{r'} \in \mathbb{Z}_2^{m'}$.
4. $\mathbf{P_B}$ sends $\{\mathbf{u}^i\}^l$.

- **Consistency check:**
  1. Let $o = m/w$, we divide the m' OTs into $o+1$ blocks of w bits. Denote $\hat{\mathbf{r}} = (\hat{\mathbf{r}}_1, ..., \hat{\mathbf{r}}_{o+1}) \in \mathbb{Z}_{2^w}^{o+1}$, $\hat{\mathbf{t}}^i = (\hat{\mathbf{t}}_1^i, ..., \hat{\mathbf{t}}_{o+1}^i) \in \mathbb{Z}_{2^w}^{o+1}$, $\hat{\mathbf{q}}^i = (\hat{\mathbf{q}}_1^i, ..., \hat{\mathbf{q}}_{o+1}^i) \in \mathbb{Z}_{2^w}^{o+1}$.
  2. $\mathbf{P_A}$ samples $\mathcal{Y} = (\mathcal{Y}_1, ...\mathcal{Y}_o) \in \mathbb{Z}_{2^w}^o$, then sends $\mathcal{Y}$ to $\mathbf{P_B}$.
  3. $\mathbf{P_B}$ computes and sends $(x, \{t^i\})$, where:

$$x = \sum_{j=1}^{o} \hat{\mathbf{r}}_j \cdot \mathcal{Y}_j \oplus \hat{\mathbf{r}}_{o+1} \quad \text{and} \quad t^i = \sum_{j=1}^{o} \hat{\mathbf{t}}^i \cdot \mathcal{Y}_j \oplus \hat{\mathbf{t}}_{o+1}^i \quad \text{for} \quad i = 1, ..., \kappa$$

  4. $\mathbf{P_A}$ computes:

$$q^i = \sum_{j=1}^{o} \hat{\mathbf{q}}^i \cdot \mathcal{Y}_j \oplus \hat{\mathbf{q}}_{o+1}^i \quad \text{for} \quad i = 1, ..., \kappa$$

  then checks whether $q^i = t^i \oplus s_i \cdot x$ for all $i = 1, ..., \kappa$, and aborts with output $\perp$ if not.

5. For $i \in [l]$, $\mathbf{P_A}$ defines $\mathbf{q}^i = (s_i \cdot \mathbf{u^i}) \oplus \mathsf{G}(k_{s_i}^i)$. Let $Q = [q^1|...|q^l]$ denote the $m \times l$ bit matrix with $\mathbf{q^i}$ representing its ith column. Let $\mathbf{q}_j$ denote the jth row of the matrix Q. Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$.
6. $\mathbf{P_A}$ computes and sends $\{y_j^0, y_j^1\}_{j\in[m]}$, where:

$$y_j^0 = x_j^0 \oplus \mathsf{H}(j, \mathbf{q}_j) \quad \text{and} \quad y_j^1 = x_j^1 \oplus \mathsf{H}(j, \mathbf{q}_j \oplus \mathbf{s})$$

** $\mathbf{P_A}$ computes a corresponding signature $\sigma^{\mathsf{A,OT}} \leftarrow \mathsf{SIG.Sign}(\mathsf{sk_A}, \{\mathbf{u}^i\}||\{y_j^0, y_j^1\}||\mathsf{sid})$, then sends to $\mathbf{P_B}$, which checks whether $\sigma^{\mathsf{A,OT}}$ is valid, and aborts with output $\perp$ if not.

7. $\mathbf{P_B}$ outputs $(x_1^{r_1}, ..., x_m^{r_m})$ with $x_j^{r_j} = y_j^{r_j} \oplus \mathsf{H}(j, \mathbf{t}_i)$.

* **Arbitrate:**
  1. $\mathbf{P_B}$ sends an arbitrate query $(\{\mathbf{u}^i\}^l, \{y_j^0, y_j^1\}, \sigma^{\mathsf{A,OT}}, \mathsf{sid})$ to $\mathbf{P_{Ar}}$, which checks whether $\sigma^{\mathsf{A,OT}}$ is valid, and aborts with output $\perp$ if not.
  2. $\mathbf{P_B}$ sends an arbitrate query $((\mathbf{arbitrate}, \mathbf{P_A}), \mathbf{P_B}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{BA,R}}^{\mathsf{l\times OT}}$. If $\mathbf{P_{Ar}}$ receives $(\mathbf{cheated}, \mathbf{P_A}, \mathsf{sid})$, then the arbitration ends here. If $\mathbf{P_{Ar}}$ receives $(\mathbf{honest}, \mathbf{P_A}, \mathsf{sid})$ and $(\mathbf{s}, \{k_{s_i}^i\})$, the arbitration proceeds.
  3. $\mathbf{P_{Ar}}$ computes $\mathbf{q}^i = (s_i \cdot \mathbf{u^i}) \oplus \mathsf{G}(k_{s_i}^i)$ and derives $\mathbf{q}_j$ as $\mathbf{P_A}$. It then computes $x_j^0 = y_j^0 \oplus \mathsf{H}(j, q_j)$ and $x_j^1 = y_j^1 \oplus \mathsf{H}(j, q_j \oplus \mathbf{s})$.

Fig. 7. A secure OT Extension protocol $\mathit{\Pi}_{\mathsf{BA,S}}^{\mathsf{OTE}}$ that UC-realizes $\mathcal{F}_{\mathsf{BA,S}}^{\mathsf{m\times OT}}$.