Delayed-Input Multi-Party Computation

Michele Ciampi¹[®], Jure Sternad^{2†}[®], and Yu Xia¹[®]

¹ The University of Edinburgh, UK; {michele.ciampi, yu.xia}@ed.ac.uk
² Aarhus University, Denmark; jsternad@cs.au.dk

Abstract. In this work, we consider the setting where the process of securely evaluating a multiparty functionality is divided into two phases: offline (or preprocessing) and online. The offline phase is independent of the parties' inputs, whereas the online phase does require the knowledge of the inputs. We consider the problem of minimizing the round of communication required in the online phase and propose a round preserving compiler that can turn a big class of multi-party computation (MPC) protocols into protocols in which only the last two rounds are input-dependent. Our compiler can be applied to a big class of MPC protocols, and in particular to all existing round-optimal MPC protocols. All our results assume no setup and are proven in the dishonest majority setting with black-box simulation. As part of our contribution, we propose a new definition we call Multi-Party Computation with Adaptive-Input Selection, which allows the distinguisher to craft the inputs the honest parties should use during the online phase, adaptively on the offline phase. This new definition is needed to argue that not only are the messages of the offline phase input-independent but also that security holds even in the stronger (and realistic) adversarial setting where the inputs may depend on some of the offline-phase protocol messages. We argue that this is the definition that any protocol should satisfy to be securely used while preprocessing part of the rounds. We are the first to study this definition in a setting where there is no setup, and the majority of the parties can be corrupted. Prior definitions have been presented in the Universal Composable framework, which is unfortunately not well suited for our setting (i.e., no setup and dishonest majority). As a corollary, we obtain the first four-round (which is optimal) MPC protocol, where the first two rounds can be preprocessed, and its security holds against adaptive-input selection.

Keywords: Secure multi-party computation · delayed-input · round-optimal · preprocessing

[†]The main part of the work was done when this author was a student at the University of Edinburgh.

Table of Contents

De	elayed-Input Multi-Party Computation	1
	Michele Ciampi [®] , Jure Sternad ³ [®] , and Yu Xia [®]	
1	Introduction	1
	Our Contributions	3
	1.1 Technical Overview	3
	On The Need For A New Definition	4
	Overview Of Multi-Party Computation With Adaptive-Input Selection.	5
	Supported MPC Protocols.	6
2	Preliminaries	6
	2.1 Symmetric Encryption Scheme	7
	2.2 Message Authentication Code Scheme	7
	2.3 Garbling Scheme	8
	2.4 Additive secret sharing scheme	9
3	Secure Multiparty Computation (MPC)	9
	Real world	9
	Ideal world.	10
	Security Definitions	10
	3.1 Additional notation	11
4	Secure Computation with Adaptive-Input Selection	11
	Real world	12
	Ideal world.	12
5	Our Compiler	13
	5.1 The Security Proofs	14

1 Introduction

Secure multiparty computation (MPC) [Yao86, GMW87] provides a solution to the problem of performing computation on private data by allowing a group of parties to jointly evaluate any function over their inputs in such a manner that no one learns anything beyond the output of the function. Since its conception, MPC has been extensively studied from different angles, in particular concerning: necessary assumptions [GMW87, Kil88, IPS08], and round complexity [GMW87, BMR90, KOS03, KO04, PW10, Pas04, Goy11, GMPP16, ACJ17, BHP17, COSV17a, COSV17b]. Even in a setting where all but one party are corrupted, both of these topics are by now pretty well understood. In the plain model, where no setup is available, parties need at least four rounds of communication⁴ to realize the coin-tossing functionality with black-box simulation [GMPP16, KO04]. A sequence of recent works showed that four rounds are also sufficient, relying on a variety of standard cryptographic assumptions [ACJ17, BHP17, BGJ⁺18, HHPV18]. In particular, in $[CCG^+20]$, the authors show a protocol that relies on the almost minimal assumption of malicious secure four-round Oblivious Transfer (OT), and recently, in [IKSS23] the authors show a protocol that relies on the underlying OT protocol in a black-box way (in this case the authors require the security of the OT protocol to hold against sub-exponential time adversaries). Another prolific line of research that aims to understand the concrete efficiency of MPC rather than focusing strictly on the round complexity is related to the area of secure computation with *preprocessing* [BMR90, Bea95, Bea96, HOSS18, DILO22, HSS17,

³The main part of the work was done when this author was a student at the University of Edinburgh.

⁴Unless otherwise specified, the communication model we consider is simultaneous (i.e., all the parties can speak in the same round). In the multi-party setting, messages are sent over broadcast channels (one round corresponds to one message sent over broadcast).

KRRW18, NNOB12, NST17, BGI19, BGIN22]. In this, the MPC protocols are seen as composed of two phases: offline (or preprocessing) and online. In the offline phase, the parties do not know their inputs but exchange messages to generate some correlated randomness. In the online phase, the parties get access to their inputs and conclude the execution of the protocol by properly combining the inputs and the correlated randomness obtained during the offline phase. The idea behind this paradigm is to delegate most of the expensive operations to the offline phase (which can be run overnight or when there are no other critical activities that need computational resources). In this way, whenever the parties get access to their inputs, they can engage in the online phase and evaluate the output quickly.

In this work, we investigate whether it is possible to have the best of both worlds: a protocol that has optimal round complexity but where the majority of the messages can be exchanged during the offline phase and only a few messages (the last two) are exchanged during the online phase. We note that any generic ℓ -round MPC protocol must use the inputs of the parties at least in the $(\ell - 1)$ -th round⁵. Building on top of this fact, we consider the following natural question.

Is it possible to construct a round-optimal protocol for securely evaluating generic multiparty functionalities where all but the last two rounds of the protocol are preprocessed in the offline phase?

A natural first step to answer this question is to look at existing round-optimal protocols and check in what round the inputs become needed. To the best of our knowledge, in the protocols proposed in [HHPV18, IKSS23], the parties *do need* their inputs already in the first two rounds. In [CCG⁺20, BGJ⁺18] instead, only the last two rounds are input-dependent. Hence, it seems that the protocols of [CCG⁺20, BGJ⁺18] already give a positive answer to the above question.

Unfortunately, this is not the case. These protocols are proven secure with respect to the standalone security definition of MPC. At a very high level, this notion guarantees that no adversary can distinguish real-world executions, where honest parties simply use their inputs and follow the protocol as prescribed, from an ideal-world execution, where honest parties are replaced with a simulator, which only has access to the output of the computation, and has no additional information on the honest parties' inputs. In this definition, it is crucial that the adversary engages exactly in one execution of the protocol, and moreover, the inputs the honest parties use are fixed at the beginning of the experiment (in both the real-world and the ideal-world experiment). This, in particular, means that the distinguisher is not supposed to decide the inputs that the honest parties should use adaptively on, for example, the first two rounds of the protocol. The consequence of this is that even if a protocol is input-independent in the first two rounds, its security holds only if the honest parties' inputs are fixed before the protocol starts. This clearly defeats the entire purpose of the offline/online paradigm. It is indeed easy to design a protocol that is secure under the standard definition of MPC, but it becomes insecure if the distinguisher can decide the inputs adaptively on some of the protocol messages.

To see this more explicitly, consider a protocol Π , secure under the standard MPC security definition, and let us assume that the first two rounds of the protocol are input-independent. We now construct a new protocol Π' , that works exactly as Π , but with the following modification. In the first round, each party samples and sends a random string. Upon receiving the input, each party compares the input with the random string, and if they are equal, then the party sends the input in the clear; otherwise, they simply follow the specification of Π . This protocol is secure under the standard MPC definition, due to the security offered by Π , and because the probability that the input of the party is equal to the random string is negligible. On the other hand, the protocol Π' is clearly not secure if the distinguisher is allowed to pick the input of the honest parties adaptively on the first round of the protocol.

This is, of course, an ad-hoc example, but it should be clear that one should not trust to execute a protocol in the offline/online setting, in which the inputs of the online phase are decided after the offline phase is concluded. Hence, it is natural to introduce a new definition that provides security in the very natural adversarial setting we are considering.

⁵Assume by contradiction that this is not the case, then an adversary, upon receiving the last round of the protocol, can always locally compute the residual function with respect to multiple inputs.

Our Contributions. Building on top of the above observation, we first devise a new security definition, which captures the ability of the distinguisher to pick the input adaptively on the first $\ell - R$ rounds of the protocol (assuming the protocol has ℓ rounds in total, with only the last R rounds requiring inputs from the parties, where $1 \leq R \leq \ell - 1$). We call this new security definition *MPC with Adaptive-Input Selection*. We provide our notion in four different flavours, extending the standard notions of security with *selective*, *selective identifiable*, *unanimous*, and *identifiable* aborts to our adaptive-input paradigm. We recall that the notion of security with unanimous abort guarantees that if an honest party gets the output, then all the honest parties get the output. The notion of identifiable abort is stronger as it guarantees that if honest parties do not get the output, then all the parties agree on the identity of a corrupted party. Another notion is selective identifiable abort, which means that every honest party can either get the output or identify a corrupted party (without having agreement among all honest parties). We also consider the less common selective variants of these notions, where no unanimity is guaranteed (i.e., some honest parties may abort and some others may not).

As our main contribution, we show how to turn a big class of MPC protocols that may require the inputs already to compute the first round into a new protocol that needs the inputs *only* in the last two rounds.

We require the input protocol to admit a simulator that can extract the inputs of the parties before the ℓ -th round and that after the extraction phase, it behaves in a straight-line (i.e., non-rewinding) manner. To the best of our knowledge, this is a property held by most MPC protocols that allow the evaluation of generic functionality. This, in particular, holds true for the protocols proposed in [HHPV18, IKSS23, CCG⁺20, BGJ⁺18].

Moreover, the security of the output protocols satisfies our new definition of security with adaptive-input selection. Notably, our compiler is round-preserving. Hence, it can be applied to all the existing round-optimal protocols. Informally, we prove the following theorem.

Theorem (informal). Assuming the existence of an ℓ -round protocol Π that is secure with selective/selectiveidentifiable/identifiable/unanimous abort, there exists an ℓ -round protocol Π' where the first $\ell - 2$ rounds are input independent. Moreover, Π' is secure with adaptive-input selection with selective/selective-identifiable/ identifiable/unanimous abort.

All the results presented in this paper are with respect to black-box simulation, dishonest majority, static corruption, and in the plain model. Hence, we will not specify this in the remainder of the paper.

1.1 Technical Overview

We design a compiler that takes as input an ℓ -round MPC protocol Π and returns a new ℓ -round protocol $\hat{\Pi}$ where the first $\ell - 2$ rounds are independent of parties' inputs. Our compiler follows the paradigm proposed in [BMR90], but it will require to significantly depart from [BMR90] due to our requirement on the round complexity (we elaborate more on this later). Let f be the function that the parties wish to compute, and for simplicity, let us assume that each party has a one-bit input. The parties will execute Π (the non-delayed input protocol) to compute an *n*-bit-input functionality g.

The functionality g computes a garbled circuit for the function f (the function the parties want to actually compute, and we denote the corresponding circuit as C), performs an *n*-out-of-*n* secret sharing of each input label⁶ and delivers the shares to the *n* parties as part of the output, together with the garbled circuit. More precisely, the shares are returned in a permuted order to hide which shares are related to the label for the bit b (with $b \in \{0, 1\}$). The permutation is decided by the permutation bit d^i that each party P_i uses as a part of the input to g.

Upon receiving the output of g, each party P_i broadcasts $z^i \leftarrow d^i \oplus x^i$ (where x^i corresponds to the input of P_i), and the share of the label related to its input. The bit z^i indicates which one, among the two shares related to the labels of the party P_i , each party P_j should broadcast in the next round. Once the selected

⁶In the literature, the labels are sometimes called keys. We recall that each wire of a garbled circuit has a label/key corresponding to the input 0 and one corresponding to the input 1. In this case, the garbled circuit contains n wires, one per each party.

Party P_i	$\left(d^{i},((k_{0}^{i\rightarrow1},k_{1}^{i\rightarrow1}),\ldots,(k_{0}^{i\rightarrown},k_{1}^{i\rightarrown})))\right)$
- $a \leftarrow \{0, 1\}$ - For $k \in [n], b \in \{0, 1\}$:	Function g :
$k_{b}^{i \to k} \stackrel{\scriptscriptstyle \oplus}{\leftarrow} \mathtt{Gen}_{enc}(1^{\lambda})$	$\mathbf{F} = \{\mathbf{K}_b^k\}_{k \in [n], b \in \{0, 1\}}$
$(\mathcal{G}, \{c_b^{i o k}\}_{i,k \in [n], b \in \{0,1\}})$	- Secret sharing: For $k \in [n], b \in \{0, 1\}$: $(K_b^{1 \to k}, \dots, K_b^{n \to k}) \stackrel{\$}{\leftarrow} \operatorname{share}(K_b^k)$ - Encryption: For $i \in [n], k \in [n], b \in \{0, 1\}$:
	$\mathbf{c}_{d^k \oplus b}^{i \to k} \stackrel{\$}{\leftarrow} \mathbf{enc}(\mathbf{k}_{d^k \oplus b}^{i \to k}, K_b^{i \to k})$

Fig. 1: Gen_{enc} denotes the key generation algorithm of a symmetric key encryption scheme, and enc denotes its encryption algorithm. The garbling of the function f is done through the garble algorithm that takes as input the Boolean circuit C representing f and outputs the garbled circuit \mathcal{G} along with the set of labels \mathbf{K} . A label K_b^k is secret shared by running an *n*-out-of-n additive secret sharing algorithm share that takes as input the label and outputs n secret shares $K_b^{1\to k}, \ldots, K_b^{n\to k}$.

shares have been broadcasted, each party can reconstruct exactly one label per input wire and evaluate the garbled circuit.

Unfortunately, there is a subtlety with the above approach. Indeed, the parties need to wait to receive the shares returned from the evaluation of g by the protocol Π (which requires ℓ rounds) and then exchange two additional rounds, one for sending the permutation bits and the other for sending the share-tag pairs. To make the compiler round-preserving, we need to depart significantly from prior BMR-like approaches considered in the literature so far and apply the following changes. The function g does not return the shares in the clear, but it returns an encryption of the shares. In more detail, for each $i \in [n]$, P_i receives encryptions of all the shares related to the input labels for 0 and 1 of all the parties. In addition, the party P_i receives a set of keys that allow it to decrypt all the ciphertexts containing the *i*-th share corresponding to its and all the other parties' labels.

After the execution of g, each party possesses the encrypted shares of all the garbled circuit labels. To allow each party to decrypt the correct shares and to finally evaluate the garbled circuit, we let the parties broadcast $\overline{z}^i \leftarrow x^i \oplus d^i$. Similarly to what happened in the original protocol, in the last round, the parties send the keys corresponding to the permutation bits received in the previous round. This will enable the decryption of the labels and the evaluation of the garbled circuit.

At first glance, it looks like this modification did not help reduce the round complexity to ℓ . However, we observe that the encryption keys do not need to be decided by the functionality g and can instead be part of the input of g. Hence, the keys have been known to the parties since the beginning of the protocol, and upon receiving the permutation bit in the $(\ell - 1)$ -th round, the parties can send the keys right away without needing to wait to receive the output of g. We refer the reader to Figure 1 for a pictorial overview of the function g and how the input to the function is computed by each party and refer to Figure 2 for a pictorial overview of the entire protocol.

We have overlooked yet another problem in the above discussion, which is related to the fact that parties may send incorrect keys, which may still reconstruct to labels that yield the garbled circuit evaluation to an incorrect (but different from \perp) output.⁷ We solve this problem by letting g authenticate the labels. More details on this are provided in the technical section of the paper.

On The Need For A New Definition. As mentioned, the concept of MPC with pre-processing has been extensively studied. So, using the security definitions proposed in prior works would have been natural.

⁷Note that in the standard definition of the garbled circuit, nothing prevents an adversary from generating tampered labels that yield to an incorrect output (but different from \perp). This does not contradict the security of the garbled circuit, as the output obtained via this process may leak exactly the same (or less) information compared to the correct output.



Fig. 2: Gen_{enc} denotes the key generation algorithm of a symmetric key encryption scheme, and dec denotes its decryption algorithm. Each label K_{xk}^k is reconstructed by running the reconstruction algorithm reconstruct of the *n*-out-of-*n* secret sharing scheme. eval represents the evaluation algorithm of the garbling scheme that takes as input the garbled circuit \mathcal{G} along with *n* garbled labels $K_{x1}^1, \ldots, K_{xn}^n$ and outputs the evaluation of the circuit *y*. The protocol Π evaluates the function *g* on inputs d^i and $\{k_b^{i\to k}\}_{k\in[n],b\in\{0,1\}}$ for each party P_i . We denote the message generated from the party P_i in the *k*-th round of the protocol Π by $\Pi.msg_{k,i}$. For simplicity, only the outgoing messages from the *i*-th party are shown. The protocol is symmetric, so all incoming messages are of the same form.

To the best of our knowledge, most of these works are proven secure in the Universal Composable (UC) setting [Can01]. At a high level, in this setting, there is an adversary and an environment. The role of the environment (among others) is to decide when a protocol starts and what input the parties (both the honest and the corrupted parties) should use. As such, the UC framework captures very well the setting where the inputs of the honest parties are decided adaptively on the messages exchanged during the offline phase or on the messages of other protocols that are running concurrently. However, the UC framework and most of its relaxed versions [CCL15] do not allow the simulator to rewind the environment. In particular, the simulator must be able to simulate in a straight-line fashion. Unfortunately, when no setup is available, and the majority of the parties are corrupted, it is impossible to construct such a straight-line simulator [CKL03]. This is the reason why we could not adopt any of the existing definitions from the UC area, and we needed to modify the standard, standalone MPC security definition to capture the stronger adversarial setting we consider, which has been so far overlooked.

Overview Of Multi-Party Computation With Adaptive-Input Selection. In our new MPC definition, the realworld experiment is nearly the same as the one in the standard definition. The only difference is that after $\ell - R$ rounds, the adversary can decide the input of honest parties (denoted as x') in the experiment. The ideal-world experiment is, instead, quite different. Instead of giving the simulator oracle access to the adversary \mathcal{A} , we provide oracle access to $\mathcal{P}(\mathcal{A})$, and call \mathcal{P} a proxy machine.

 \mathcal{P} acts as a proxy between the adversary and the simulator with respect to all the protocol messages and never provides any information about the honest-parties inputs decided by the adversary. In more detail, we know that at some point, the adversary needs to decide what the inputs of honest parties x' are, and we do not want the simulator to know what x' is (as this would trivialize the definition). Whenever the adversary outputs x', \mathcal{P} will send x' to the ideal functionality f without letting \mathcal{S} know. We also note that S might rewind, and A might change x' during rewinding. Therefore, we require S to signal that it has finished rewinding by sending a confirmation message ok to \mathcal{P} , and only at that point \mathcal{P} will then send x' to the ideal functionality.

We show the ideal-world experiment in the Figure 3. The straight lines represent the flow of protocol messages between S and A, while \mathcal{P} acts as an intermediary, simply relaying these messages between the two entities. When S sends ok to \mathcal{P} (indicated by dashed lines), \mathcal{P} forwards x' to f (marked with dotted lines). Upon receiving x', f waits for S to obtain the inputs of corrupted parties, and then the ideal-world experiment continues in the same way as the ideal-world experiment of the standard MPC definition.



Fig. 3: The randomness that \mathcal{A} should use and all messages from the simulator \mathcal{S} to \mathcal{P} are forwarded to \mathcal{A} . Similarly, all protocol messages of \mathcal{A} are forwarded by \mathcal{P} to \mathcal{S} (denoted by straight lines), except the messages in which \mathcal{A} attempts to send the input for honest parties x' (denoted by dotted lines). \mathcal{P} blocks these messages unless it receives an ok command from \mathcal{S} . If ok is received, \mathcal{P} forwards x' to the ideal functionality, which then waits for the inputs from the corrupted parties (likely extracted by \mathcal{S}) and returns the output y to \mathcal{S} .

Supported MPC Protocols. We mentioned that our compiler supports a big class of protocols. More formally, these protocols are all those that admit a simulator that 1) never generates the last-round message of the protocol before extracting the input of the corrupted parties and 2) after the simulator has queried the ideal functionality, it completes the simulation in a straight line (i.e., it does not perform any additional rewinding). We refer to these types of MPC protocol as a *Special-Extractable* protocol. We find it useful to assume that the non-delayed input protocol satisfies this property because the simulator we design will need to send simulated encryption keys in the last round of the protocol. Without knowing the corrupted parties' inputs, it is not clear how to generate these keys properly. This is why we require the first property. We require the second property to make sure that once the input is extracted, the simulator does not, for example, fully rewind the entire session. This would make our simulation more involved.

2 Preliminaries

We denote the security parameter with $\lambda \in \mathbb{N}$. We use " \leftarrow " as the assigning operator (e.g. if we assign the value of b to a, we write $a \leftarrow b$). For the randomized assignment, we use " \Leftarrow " where the randomness is not explicit. A randomized algorithm A is running in probabilistic polynomial time (PPT) if there exists a polynomial poly(\cdot) such that for every input x the running time of A(x) is bounded by poly(|x|). A function $f(\cdot)$ is negligible in λ , or just negligible, if for every positive polynomial poly(\cdot) and all sufficiently large λ it holds that $f(\lambda) < \frac{1}{\text{poly}(\lambda)}$ [HL10]. In this paper, we use $\epsilon(\cdot)$ to denote a negligible function.

A probability ensemble $X = \{X(a,\lambda)\}_{a \in \{0,1\}^*, \lambda \in \mathbb{N}}$ is an infinite sequence of random variables indexed by a and $\lambda \in \mathbb{N}$. Two probability ensembles $X = \{X(a,\lambda)\}_{a \in \{0,1\}^*, \lambda \in \mathbb{N}}$ and $Y = \{Y(a,\lambda)\}_{a \in \{0,1\}^*, \lambda \in \mathbb{N}}$ are said to be *computationally indistinguishable*, denoted by $X \stackrel{c}{=} Y$, if for every non-uniform polynomial-time algorithm \mathcal{D} there exists a negligible function $\epsilon(\cdot)$ such that for every $a \in \{0,1\}^*$ and every $\lambda \in \mathbb{N}$,

$$\left| \Pr[\mathcal{D}(X(a,\lambda)) = 1] - \Pr[\mathcal{D}(Y(a,\lambda)) = 1] \right| \le \epsilon(\lambda) [\text{HL10}].$$

2.1 Symmetric Encryption Scheme

Definition 1. [GB08] A symmetric encryption scheme for the message space M is a tuple of three algorithms $SE = (Gen_{enc}, enc, dec)$:

- $-k \stackrel{\$}{\leftarrow} \operatorname{Gen}_{\operatorname{enc}}(1^{\lambda})$: Takes as input a unary representation of the security parameter λ , and outputs a key k.
- $-c \stackrel{\$}{\leftarrow} \operatorname{enc}(k,m)$: Takes as input the symmetric key k and a message $m \in M$ to encrypt, and outputs a ciphertext c.
- $-m \leftarrow \operatorname{dec}(k,c)$: Takes as input the symmetric key k and a ciphertext c, and outputs a message or \perp if decryption fails.

A scheme SE is correct if, for all $\lambda \in \mathbb{N}$, $k \stackrel{\$}{\leftarrow} \text{Gen}_{enc}(1^{\lambda})$, $m \in M$, we have

$$\Pr[\operatorname{dec}(k, \operatorname{enc}(k, m)) = m] = 1.$$

Definition 2. [KL14] Let $\Pi = (\text{Gen}_{enc}, enc, dec)$ be a symmetric encryption scheme with message space M. Let \mathcal{A} be an adversary, which is formally just a (stateful) algorithm. We define an experiment $\text{PrivK}_{\mathcal{A},\Pi}^{eav}$ in the Figure 2.1 as follows:

Figure 2.1: The adversarial indistinguishability experiment $\operatorname{PrivK}_{\mathcal{A},\Pi}^{eav}$.

- 1. The adversary A outputs a pair of messages $m_0, m_1 \in M$.
- 2. A key k is generated using Gen_{enc} , and a uniform bit $b \in \{0,1\}$ is chosen. Ciphertext $c \xleftarrow{} enc(k, m_b)$ is computed and given to \mathcal{A} . We refer to c as the challenge ciphertext.
- 3. \mathcal{A} outputs a bit b'.
- 4. The output of the experiment is defined to be 1 if b' = b, and 0 otherwise. We write $\text{PrivK}_{\mathcal{A},\Pi}^{\text{eav}} = 1$ if the experiment's output is 1, and say that \mathcal{A} succeeds.

Definition 3. [KL14] Symmetric encryption scheme $\Pi = (\text{Gen}_{enc}, enc, dec)$ with message space M is computationally indistinguishable if for every PPT A it holds that

$$\Pr[\operatorname{PrivK}_{\mathcal{A},\Pi}^{\operatorname{eav}}=1]=\frac{1}{2}+\epsilon(\lambda)$$

2.2 Message Authentication Code Scheme

Definition 4. [KL14] A message authentication code scheme consists of three PPT algorithms $MAC = (Gen_{mac}, Mac, Vrfy)$ such that:

- 1. The key generation algorithm $k \stackrel{\$}{\leftarrow} \text{Gen}_{mac}(1^{\lambda})$ takes as input a unary representation of the security parameter λ and outputs a key k with $|k| \geq \lambda$.
- 2. The tag generation algorithm Mac takes as input a key k and a message $m \in \{0, 1\}^*$, and outputs a tag t. Since this algorithm may be randomized, we write this as $t \stackrel{\$}{\leftarrow} \operatorname{Mac}_k(m)$.
- 3. The deterministic verification algorithm Vrfy takes as input a key k, a message m, and a tag t. It outputs a bit b, with b = 1 meaning valid and b = 0 meaning invalid. We write this as $b \leftarrow Vrfy_k(m, t)$.

It is required that for every λ , every key k output by $\text{Gen}_{max}(1^{\lambda})$, and every $m \in \{0,1\}^*$, it holds that $\operatorname{Vrfy}_k(m, \operatorname{Mac}_k(m)) = 1.$

Definition 5. [KL14] $\Pi = (\text{Gen}_{mac}, \text{Mac}, \text{Vrfy})$ is an computationally secure one-time MAC if for all PPT adversaries \mathcal{A} it holds that:

$$\Pr[\mathsf{Mac} - \mathsf{forge}_{\mathcal{A},\Pi}^{1-\mathsf{time}} = 1] \le \epsilon(n)$$

in the Mac - forge^{1-time}_{\mathcal{A},Π} experiment, as defined in Figure 2.2:

Figure 2.2: Mac - forge $\frac{1-\text{time}}{4}$

- 1. A key k is generated by running Gen_{max} .
- 2. The adversary \mathcal{A} outputs a message m', and m' is given to the challenger. Challenger returns a $tag t' \stackrel{\$}{\leftarrow} \operatorname{Mac}_k(m').$
- 3. \mathcal{A} outputs (m, t).
- 4. The output of the experiment is defined to be 1 if and only if (1) $\operatorname{Vrfy}_{k}(m,t) = 1$ and (2) $m \neq m'$.

$\mathbf{2.3}$ Garbling Scheme

Definition 6. [*DRSY23*] A garbling scheme GC is a pair of algorithms (garble, eval) defined as follows.

- $-(\mathcal{G},\mathbf{K}) \stackrel{\$}{\leftarrow} \operatorname{garble}(1^{\lambda},\mathbb{C})$: The algorithm garble takes as input a unary representation of the security parameter λ and a Boolean circuit $C: \{0,1\}^{\mathsf{L}} \to \{0,1\}^{\mathsf{m}}$, and outputs a garbled circuit \mathcal{G} and L pairs of garbled labels $\mathbf{K} = (K_0^1, K_1^1, \dots, K_0^L, K_1^L)$. For simplicity, we assume that for every $i \in [L]$ and $b \in \{0, 1\}$, it holds that $K_b^i \in \{0, 1\}^{\lambda}$.
- $-y \leftarrow eval(\mathcal{G}, \check{K^1}, \ldots, \check{K^L})$: The algorithm eval takes as input the garbled circuit \mathcal{G} and L garbled labels $K^1, \ldots, K^{\mathsf{L}}$, and outputs a value $y \in \{0, 1\}^m$.

We require the following properties of a garbling scheme:

- Perfect Correctness. We say GC satisfies *perfect correctness* if for any Boolean circuit $C: \{0, 1\}^{L} \rightarrow 0$ $\{0,1\}^m$ and $x = (x_1, \ldots, x_L)$ it holds that

$$\Pr[\texttt{eval}(\mathcal{G}, \mathbf{K}[x]) = \mathtt{C}(x)] = 1,$$

where $(\mathcal{G}, \mathbf{K}) \stackrel{\$}{\leftarrow} \texttt{garble}(1^{\lambda}, \mathbb{C})$ with $\mathbf{K} = (K_0^1, K_1^1, \dots, K_0^{\mathsf{L}}, K_1^{\mathsf{L}})$ and $\mathbf{K}[x] = (K_{x_1}^1, \dots, K_{x_{\mathsf{L}}}^{\mathsf{L}})$.

Privacy. We say that GC satisfies *privacy* if there exists a simulator simGC such that for every PPT adversary \mathcal{A} , it holds that

$$\Pr[\texttt{GC}^{\texttt{priv}} = 1] \le \frac{1}{2} + \epsilon(\lambda),$$

where GC^{priv} experiment is defined in Figure 2.3:

Figure 2.3: The garbling scheme privacy experiment GC^{priv}.

- 1. \mathcal{A} constructs a circuit $C: \{0,1\}^{\mathsf{L}} \to \{0,1\}^{m}$, outputs an input $x = (x_1, \ldots, x_{\mathsf{L}}) \in \{0,1\}^{\mathsf{L}}$ and sends them to the challenger.
- 2. Challenger chooses a random bit $b \in \{0, 1\}$ and does the following:
 - (a) if b = 0: $(\mathcal{G}, (K_0^1, K_1^1, \dots, K_0^L, K_1^L)) \stackrel{\$}{\leftarrow} garble(1^\lambda, \mathbb{C})$. Set $K^i = K_{r_i}^i$ for $i \in [L]$.

(b) if b = 1: $(\mathcal{G}, K_{sim}^1, \dots, K_{sim}^L) \stackrel{\$}{\leftarrow} sim GC(1^{\lambda}, C, C(x))$. Set $K^i = K_{sim}^i$ for $i \in [L]$. 3. Challenger sends $\mathcal{G}, K^1, \dots, K^L$ to \mathcal{A} .

- 4. \mathcal{A} outputs a bit b'.
- 5. \mathcal{A} wins if b = b', and we say that the experiment evaluates to 1 in this case.

2.4 Additive secret sharing scheme

In our work, we will use n-out-of-n secret sharing scheme, which is also referred to as additive secret sharing scheme. We give the definition here:

Definition 7 (n-out-of-n secret sharing scheme). A n-out-of-n secret sharing scheme is a pair of algorithms (share, reconstruct), such that:

- $-(s_1,\ldots,s_n) \stackrel{\$}{\leftarrow} \operatorname{share}(m)$ is a randomized algorithm that on input a secret m, outputs a set of n shares (s_1,\ldots,s_n) .
- $-m \leftarrow \text{reconstruct}(s_1, \ldots, s_n)$ is a deterministic algorithm that on input n shares (s_1, \ldots, s_n) , outputs the secret m.

The scheme needs to satisfy the correctness property if : $\forall m$, we have:

 $\Pr_{\texttt{share}(m) \to (s_1, \dots, s_n)}[\texttt{reconstruct}(s_1, \dots, s_n) = m] = 1$

In addition, the scheme satisfies perfect security property if: $\forall m, m', \forall S \subseteq \{1, \ldots, n\}$, where |S| < n, the following distributions are indistinguishable:

$$\{(s_i : i \in S) \mid (s_1, \dots, s_n) \leftarrow \mathtt{share}(m)\} \\ \{(s'_i : i \in S) \mid (s'_1, \dots, s'_n) \leftarrow \mathtt{share}(m')\}$$

3 Secure Multiparty Computation (MPC)

We follow the real-world/ideal-world simulation paradigm and we adopt the security model of Cohen, Garay and Zikas [CGZ20].

Real world. An *n*-party protocol $\Pi = (P_1, \ldots, P_n)$ is an *n*-tuple of probabilistic polynomial-time (PPT) interactive Turing machines (ITMs), where each party P_i is initialized with input $x_i \in \{0, 1\}^*$ and random coins $r_i \in \{0, 1\}^*$. We let \mathcal{A} denote a special PPT ITM that represents the adversary and that is initialized with input that contains the identities of the corrupt parties, their respective private inputs and an auxiliary input.

The protocol is executed in rounds (i.e., the protocol is synchronous). Each round consists of a send phase and a receive phase, where parties can respectively send and receive messages. The communication in each round happens via a broadcast channel. During the execution of the protocol, the corrupt parties receive arbitrary instructions from the adversary \mathcal{A} , while the honest parties faithfully follow the instructions of the protocol. We consider the adversary \mathcal{A} to be rushing, i.e., during every round, the adversary can see the messages the honest parties sent before producing messages for corrupt parties.

At the end of the protocol execution, the honest parties produce their outputs, and the adversary returns an arbitrary function of the corrupt parties' view. The view of a party during the execution consists of its input, random coins and the messages it sees during the execution.

Definition 8 (Real-world execution). Let $\Pi = (P_1, \ldots, P_n)$ be an n-party protocol and let $\mathcal{J} \subseteq [n]$, of size at most t, denote the set of indices of the parties corrupted by \mathcal{A} . The joint execution of Π under $(\mathcal{A}, \mathcal{J})$ in the real world, on input vector $\mathbf{x} = (x_1, \ldots, x_n)$, auxiliary input aux and the unary representation of the security parameter 1^{λ} , denoted $\operatorname{REAL}_{\Pi,\mathcal{J},\mathcal{A}(\operatorname{aux})}(\mathbf{x}, 1^{\lambda})$, is defined as the output vector of P_1, \ldots, P_n and $\mathcal{A}(\operatorname{aux})$ resulting from the protocol interaction. *Ideal world.* We describe ideal-world executions with selective abort (sa-abort), selective identifiable abort (si-abort) (recently introduced in [DRSY23]), unanimous abort (un-abort), identifiable abort (id-abort).

Definition 9 (Ideal-world Computation). Consider type \in {sa-abort, un-abort, si-abort, id-abort}. Let $f: (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party function and let $\mathcal{J} \subseteq [n]$, of size at most t, be the set of indices of the corrupt parties. Then, the joint ideal execution of f under $(\mathcal{S}, \mathcal{J})$ on input vector $\mathbf{x} = (x_1, \ldots, x_n)$, auxiliary input aux to \mathcal{S} and the unary representation of the security parameter 1^{λ} , denoted $\text{IDEAL}_{f,\mathcal{J},\mathcal{S}(\mathsf{aux})}^{\text{type}}(\mathbf{x}, 1^{\lambda})$, is defined as the output vector of P_1, \ldots, P_n and \mathcal{S} resulting from the following ideal process.

- 1. Parties send inputs to trusted party: An honest party P_i sends its input x_i to the trusted party. The simulator S may send to the trusted party arbitrary inputs for the corrupt parties. Let x'_i be the value actually sent as the input of party P_i .
- 2. Trusted party speaks to simulator: The trusted party computes $(y_1, \ldots, y_n) = f(x'_1, \ldots, x'_n)$. If there are no corrupt parties proceed to step 3. Otherwise, the trusted party sends $\{y_i\}_{i \in \mathcal{J}}$ to \mathcal{S} .
- 3. Simulator S responds to the trusted party:
 - (a) If type = sa-abort: The simulator S can select a set of parties that will not get the output as $\mathcal{K} \subseteq [n] \setminus \mathcal{J}$. (Note that \mathcal{K} can be empty, allowing all parties to obtain the output.) It sends (abort, \mathcal{K}) to the trusted party.
 - (b) If type = un-abort: The simulator can send abort to the trusted party. If it does, we take $\mathcal{K} = [n] \setminus \mathcal{J}$.
 - (c) If type = si-abort: The simulator S can select a set of parties that will not get the output as $\mathcal{K} \subseteq [n] \setminus \mathcal{J}$. (Note that \mathcal{K} can be empty, allowing all parties to obtain the output.) For each party j in \mathcal{K} , the adversary selects a corrupt party $i_j^* \in \mathcal{J}$ who will be blamed by party j. It sends (abort, \mathcal{K} , $\{j, i_j^*\}_{j \in \mathcal{K}}$) to the trusted party.
 - (d) If type = id-abort: If it chooses to abort, the simulator S can select a corrupt party $i^* \in \mathcal{J}$ who will be blamed, and send (abort, i^*) to the trusted party. If it does, we take $\mathcal{K} = [n] \setminus \mathcal{J}$.

Trusted party answers parties:

- (a) If the trusted party got abort from the simulator S,
 - *i.* It sets the abort message abortmsg, as follows:
 - *if* type \in {sa-abort, un-abort}, *we let* abortmsg = \perp .
 - *if* type = si-abort, we let $abortmsg = \{abortmsg_i\}_{j \in \mathcal{K}} = \{(\bot, i_i^*)\}_{j \in \mathcal{K}}$.
 - if type = id-abort, we let abortmsg = (\perp, i^*) .
 - ii. The trusted party sends y_j to every party P_j , $j \in [n] \setminus \mathcal{K}$. If type = si-abort, the trusted party then sends abortmsg_j to each party P_j , $j \in \mathcal{K}$; Otherwise, the trusted party sends abortmsg to every party P_j , $j \in \mathcal{K}$.
- (b) Otherwise, it sends y to every P_j , $j \in [n]$.
- 4. Outputs: Honest parties always output the message received from the trusted party, while the corrupt parties output nothing. The simulator S outputs an arbitrary function of the initial inputs $\{x_i\}_{i \in \mathcal{J}}$, the messages received by the corrupt parties from the trusted party and its auxiliary input.

Security Definitions. Now that we have described the real-world and the ideal-world experiment, we are ready to state the formal definition.

Definition 10. Consider type \in {sa-abort, un-abort, si-abort, id-abort}. Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party function. A protocol Π t-securely computes the function f with type security if for every PPT real-world adversary \mathcal{A} with auxiliary input aux, there exists a PPT simulator \mathcal{S} such that for every $\mathcal{J} \subseteq [n]$ of size at most t, for all $\mathbf{x} \in (\{0,1\}^*)^n$, for all $\lambda \in \mathbb{N}$, it holds that

$$\mathtt{REAL}_{\Pi,\mathcal{J},\mathcal{A}(\mathtt{aux})}(\boldsymbol{x},1^{\lambda}) \stackrel{c}{\equiv} \mathtt{IDEAL}_{f,\mathcal{J},\mathcal{S}(\mathtt{aux})}^{\mathsf{type}}(\boldsymbol{x},1^{\lambda})$$

3.1 Additional notation

Let Π be an ℓ -round MPC protocol. Rather than viewing a protocol Π as an *n*-tuple of ITMs, it is convenient to view each ITM as a sequence of multiple algorithms: $\texttt{frst-msg}_i$, to compute P_i 's first-round messages to its peers; $\texttt{nxt-msg}_i^k$, to compute P_i 's *k*-th round messages for $(2 \le k \le \ell)$; and \texttt{output}_i , to compute P_i 's output. Thus, a protocol Π can be defined as $\{(\texttt{frst-msg}_i, \texttt{nxt-msg}_i^k, \texttt{output}_i)\}_{i \in [n], k \in [\ell] \setminus \{1\}}$.

The syntax of the algorithms is as follows:

- frst-msg_i $(x^i; r_i) \rightarrow msg_{i,1}$ produces the first-round messages of party P_i to all parties.
- $\operatorname{nxt-msg}_{i}^{k}(x^{i}, {\{\operatorname{msg}_{j,l}\}_{j \in [n], l \in \{1, \dots, k-1\}}; r_{i})} \to \operatorname{msg}_{i,k}$ produces the k-th round messages of party P_{i} to all parties.
- $\text{ output}_{i}(x^{i}, \mathsf{msg}_{1,1}, \dots, \mathsf{msg}_{n,1}, \dots, \mathsf{msg}_{1,\ell}, \dots, \mathsf{msg}_{n,\ell}; r_{i}) \to y_{i} \text{ produces the output returned to party } P_{i}.$

We note that, unless needed, to not overburden the notation, we do not pass the random coin r as an explicit input of the cryptographic algorithms.

We have repeatedly mentioned that our compiler works for a big class of multi-party computation protocols (We denote this class of protocol as *Special-Extractable* MPC protocol, as per Definition 11). This property requires the MPC simulator to be able to extract the inputs of the party before generating the last round of the protocol, and after the extraction, it behaves in a straight-line manner. All existing four-round protocols satisfy this property, and it is a property enjoyed by most of the MPC protocols.

Definition 11 (Special-Extractable MPC). A secure multi-party protocol is Special-Extractable, if the simulator S (which exists by Definition 10) works in the following way. S queries the ideal-functionality (i.e., it returns the input of the corrupted parties) without ever sending the last protocol message to the adversary. Moreover, upon receiving the answer from the ideal functionality, S sends only one message to the adversary (on behalf of all the honest parties) and stops.

4 Secure Computation with Adaptive-Input Selection

We now introduce the property of *delayed-inputness* for MPC protocols. To make it more general, for any ℓ -round MPC protocol, the inputs of parties are needed only in the last R rounds, where $1 \le R \le \ell - 1$.

Definition 12 (Delayed-input multi-party computation). Let $\Pi = \{(\texttt{frst-msg}_i, \texttt{nxt-msg}_i^k, \texttt{output}_i)\}_{i \in [n], k \in [l] \setminus \{1\}}$ be an ℓ -round MPC protocol (satisfying Definition 10). Then, Π is delayed-input if the next-message functions $\{(\texttt{frst-msg}_i, \texttt{nxt-msg}_i^k)\}_{i \in [n], k \in [\ell-R] \setminus \{1\}}$ take as their first input a default value (e.g., 0^{λ}) in the place of the actual inputs of the parties. We note that $1 \leq R \leq \ell - 1$.

The above definition just describes a syntactic property that the next-message function algorithms may satisfy. For delayed-input MPC protocols, we consider a security definition that allows the adversary to craft the inputs the honest parties should use in the real-world experiment adaptively on the first $\ell - R$ rounds. To capture this, we modify the standard definition of MPC (that we recall in Definition 10) as follows.

- The real-world experiment proceeds exactly like the real-world experiment of the standard MPC definition, with the difference that the adversary, upon generating the $(\ell - R)$ -th round, also returns a special message (input, $\boldsymbol{x} = \{x_i\}_{i \in \mathcal{I}}$), where \mathcal{I} represents the indices of the honest parties. From that point on, the honest party P_i (for each $i \in \mathcal{I}$) will use the input x_i to evaluate the next message functions and complete the execution of the protocol.
- In the simulated experiment, we give the simulator black-box access to the adversary, but we prevent the simulator from getting the message (input, $\mathbf{x} = \{x_i\}_{i \in \mathcal{I}}$) (clearly, a simulator having access to these messages would trivialize the security definition).

To formalize this aspect, we give the simulator oracle access to a *proxy-machine* $\mathcal{P}(\cdot)$ and modify the ideal-world experiment as follows. The proxy machine has black-box access to \mathcal{A} (denoted with $\mathcal{P}(\mathcal{A})$) and acts as a proxy between the adversary and the calling party (the simulator in our case) with respect to all

the messages of the protocol Π . However, upon receiving the message (input, $\boldsymbol{x} = \{x_i\}_{i \in \mathcal{I}}$) from \mathcal{A} , $\mathcal{P}(\mathcal{A})$ instructs the ideal functionality to use the inputs $\{x_i\}_{i \in \mathcal{I}}$ for the honest parties. The ideal functionality then waits for the simulator to receive the input of the corrupted parties, and upon receiving them, it returns the output to the simulator (that we recall, is the ideal-world adversary). From that point on, the ideal-world experiment behaves exactly like the ideal-world experiment of the standard MPC definition. There is still another minor modification we need to do. The proxy delivers the inputs of the honest parties to the ideal functionality only after receiving an ok message from the simulator. This is needed as the simulator may want to rewind the adversary (rewinding the proxy machine), and this could cause \mathcal{A} to send multiple input commands to the proxy machine. This, in turn, could make the proxy machine communicate to the ideal functionality with some honest parties' inputs, which are not consistent with the inputs that appear in the final simulated thread. To avoid this, we let the simulator decide when is the right moment for $\mathcal{P}(\cdot)$ to send the inputs to the ideal functionality. Looking ahead, this ok command will be sent only after the simulator has decided that no more rewinds will be performed in the first $\ell - R + 1$ rounds.

We now propose a more formal definition of a real-world and ideal-world and finally state the formal definition of adaptive-input selection secure MPC.

Real world. Let $\{P_1, \ldots, P_n\}$ be a set of n parties. Let $\mathcal{I} = [n] \setminus \mathcal{J}$ be the set of honest parties' indexes. In the real world, the adversary \mathcal{A} engages in the execution of an ℓ -round delayed-input MPC protocol. The execution proceeds accordingly to the real world of the standard MPC definition (Definition 8), with the difference that in the $(\ell - R)$ -th round, the adversary sends an additional message (input, $\mathbf{x} = \{x_i\}_{i \in \mathcal{I}}$). For this point on, for each $i \in \mathcal{I}$, the party P_i uses the input x_i to complete the execution of the protocol against \mathcal{A} . We denote with $\text{REAL}_{\Pi,\mathcal{J},\mathcal{A}(aux)}^{\mathsf{DI}}(1^{\lambda})$ the output vector of P_1, \ldots, P_n and $\mathcal{A}(aux)$ resulting from the protocol interaction.

Ideal world. The ideal world also follows the one described in the standard MPC definition (Definition 9), with the following difference. The ideal world is parametrized by a proxy machine $\mathcal{P}(\cdot)$. The proxy machine is initialized with an empty variable HInputs, takes as input a PPT algorithm \mathscr{A} , and interacts with an additional algorithm called *caller* as follows.

- Upon receiving any message from the caller, forward this to \mathscr{A} .
- Upon receiving any message from \mathscr{A} , different from the command (input, \cdot), forward this message to the caller.
- Upon receiving the message (input, x) from \mathscr{A} , set Hlnputs $\leftarrow x$.
- Upon receiving the message ok from the caller, parse HInputs as $\{x_i\}_{i \in \mathcal{I}}$, and send x_i to the ideal functionality on the behalf of the honest party P_i , for each $i \in \mathcal{I}$.

In summary, the honest parties' inputs are decided by $\mathcal{P}(\mathscr{A})$ (in particular, they are decided by \mathscr{A}). The inputs of the corrupted parties instead are decided by the ideal-world adversary \mathcal{S} . When the ideal functionality receives the inputs from the honest and the corrupted parties, it acts exactly like the ideal-world functionality described in Definition 10. We denote with $\text{IDEAL}_{f,\mathcal{J},\mathcal{P}(\mathscr{A}),\mathcal{S}(\text{aux})}^{\text{D},\text{type}}(1^{\lambda})$, the output vector of P_1, \ldots, P_n and \mathcal{S} resulting from the following ideal process.

We are now ready to state the formal definition of adaptive-input selection secure MPC.

Definition 13 (MPC with Adaptive-Input Selection). Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party function. A delayed-input protocol Π t-securely computes the function f with type security if for every PPT real-world adversary \mathcal{A} with auxiliary input aux, there exists a PPT simulator \mathcal{S} that has black-box access to $\mathcal{P}(\mathcal{A})$, such that for every $\mathcal{J} \subseteq [n]$ of size at most t, for all $\lambda \in \mathbb{N}$, it holds that

$$\mathtt{REAL}_{\Pi,\mathcal{J},\mathcal{A}(\mathsf{aux})}^{\mathsf{DI}}(1^{\lambda}) \stackrel{c}{\equiv} \mathtt{IDEAL}_{f,\mathcal{J},\mathcal{P}(\mathcal{A}),\mathcal{S}(\mathsf{aux})}^{\mathsf{DI},\mathsf{type}}(1^{\lambda}).$$

We stress that the simulator does not directly query \mathcal{A} , but it has black-box access to $\mathcal{P}(\mathcal{A})$. This makes the definition non-trivial, preventing the simulator from obtaining the input that \mathcal{A} would send via the command (input, \mathbf{x}).

5 Our Compiler

In this section, we show how our compiler that turns an ℓ -round MPC protocol into an adaptive-input selection secure MPC protocol works. For simplicity, we describe our compiler for the case where $\ell = 4$ (which represents the minimal number of rounds) and R = 2, but extending the protocol to any ℓ and R = 2is trivial. We also assume that each party has only one-bit input and argue that it is trivial to extend to inputs of arbitrary size by letting one party control multiple sub-parties of the one-input protocol. Before describing our compiler, we emphasize that our scheme works for a generic g. As such, we require the input of parties in the second last round (R = 2). We note that this is optimal, as for non-trivial functionality a protocol that requires the input only to compute the last round would trivially be susceptible to residual attacks. It is still an interesting open question describing a compiler that depending on the computed function, can adaptively decide whether R = 1 or R = 2, but we do not consider this *adaptive setting* in our paper.

Our construction makes use of the following tools:

- A four-round special-extractable malicious type secure MPC (Definition 10 and Definition 11) protocol $\Pi = \{(\texttt{frst-msg}_i, \texttt{nxt-msg}_i^k, \texttt{output}_i)\}_{i \in [n], k \in \{2,3,4\}}$ against t < n corruptions, which realizes the functionality g, formally specified in the Figure 5.1 with type $\in \{\texttt{sa-abort}, \texttt{si-abort}, \texttt{un-abort}, \texttt{id-abort}\}$. The j-th round message of the protocol Π sent by party P_i is denoted as $\mathsf{msg}_{j,i}$. The additional message introduced in the compiled protocol is denoted as $\widehat{\mathsf{msg}}_{j,i}$.
- A garbling scheme GC = (garble, eval) (Definition 6).
- A symmetric encryption scheme SE = (Gen_{enc}, enc, dec) (Definition 1), that is computationally indistinguishable as per Definition 3.
- A message authentication code scheme MAC = (Gen_{mac}, Mac, Vrfy) (Definition 4) that is a one-time computationally-secure as per Definition 5.
- An additive secret sharing scheme SS = (share, reconstruct) that satisfies Definition 7. Note that in Figure 5.1 and Figure 5.2, we directly use XOR operation $\bigoplus_{i \in [n]}$ to achieve share and reconstruct.

We propose the formal description of the protocol in Figure 5.2 and refer the reader to the introductory section for its informal description.

Figure 5.1: The functionality g

Input: Each party P_i for $i \in [n]$ provides the following input: $- d^i \in \{0, 1\}$ $- k_{enc}^{i \to k, b} \in \{0, 1\}^{\lambda} \text{ for } k \in [n], b \in \{0, 1\}$ Output: Each party P_i for $i \in [n]$ gets the output $\mathcal{G}, \{\mathbf{c}^{i \to k}\}_{i \in [n], k \in [n]}, \{k_{mac}^{i \to k}\}_{i \in [n], k \in [n]}.$ 1: $(\mathcal{G}, \mathbf{K}) \stackrel{\$}{\leftarrow} garble(1^{\lambda}, C)$, where $\mathbf{K} = \{K_b^i\}_{i \in [n], b \in \{0, 1\}}$ and C is the Boolean circuit for the function f2: Sample uniformly random $K_b^{i \to k}$ for $i \in [n], k \in [n], b \in \{0, 1\}$ such that $\bigoplus_{i \in [n]} K_b^{i \to k} = K_b^k$ 3: for $i \in [n], k \in [n], b \in \{0, 1\}$ do
4: Generate $k_{mac}^{i \to k, d^k \oplus b} \stackrel{\$}{\leftarrow} \text{Gen}_{mac}(1^{\lambda})$ 5: $\gamma_{d^k \oplus b}^{i \to k} \leftarrow \text{Mac}_{k_{mac}^{i \to k, d^k \oplus b}}(K_b^{i \to k})$ 6: $c_{d^k \oplus b}^{i \to k} \leftarrow \text{enc}(k_{enc}^{i \to k, d^k \oplus b}, (\gamma_{d^k \oplus b}^{i \to k})|K_b^{i \to k}))$ 7: end for
8: for $i \in [n], k \in [n]$ do
9: $k_{mac}^{i \to k} \leftarrow (\mathbf{c}_0^{i \to k}, \mathbf{c}_1^{i \to k})$ 11: end for Figure 5.2: The protocol \hat{H} that computes f through four rounds of communication, where the first two rounds are independent of the parties' inputs

Private input: Each party P_i for $i \in [n]$ has private input $x^i \in \{0, 1\}$. **Output:** $y = f(x^1, \ldots, x^n)$ or \bot .

First round: Each party P_i does the following:

1. Sample uniformly random $d^i \stackrel{\$}{\leftarrow} \{0,1\}$ and $\{\mathsf{k}_{\mathsf{enc}}^{i\to k,b}\}_{k\in[n]} \stackrel{\$}{\leftarrow} \mathsf{Gen}_{\mathsf{enc}}(1^\lambda)$ for $b \in \{0,1\}$.

2. Set $x_g^i \leftarrow (d^i, \{k_{enc}^{i \to k, b}\}_{k \in [n], b \in \{0, 1\}})$ and compute $\mathsf{msg}_{1,i} \leftarrow \mathtt{frst-msg}_i(x_g^i)$. 3. Send $\mathsf{msg}_{1,i}$ over broadcast channel.

Second Round: Each party P_i computes $\mathsf{msg}_{2,i} \leftarrow \mathsf{nxt-msg}_i^2(x_q^i, \tau_1)$ and sends it over broadcast channel, where τ_1 is the transcript of the protocol Π up to the first round.

Third Round: Each party P_i does the following:

- 1. Let τ_2 denote the transcript of the protocol Π up to the second round.
- 2. Compute $\mathsf{msg}_{3,i} \leftarrow \mathsf{nxt}\mathsf{-msg}_i^3(x_g^i, \tau_2)$.
- 3. Compute $\widehat{\mathsf{msg}}_{3,i} \leftarrow (\phi^i \leftarrow d^i \oplus x^i)$.
- 4. Send $(\mathsf{msg}_{3,i}, \widehat{\mathsf{msg}}_{3,i})$.

Fourth Round: Each party P_i does the following:

- 1. Upon receiving $\widehat{\mathsf{msg}}_{3,k}$ for $k \in [n] \setminus \{i\}$, compute $\widehat{\mathsf{msg}}_{4,i} \leftarrow (\{\mathsf{k}_{\mathsf{enc}}^{i \to k, \phi^k}\}_{k \in [n]})$. 2. Let τ_3 denote the transcript of the protocol Π up to the third round.
- 3. Compute $\mathsf{msg}_{4,i} \leftarrow \mathsf{nxt}\mathsf{-msg}_i^4(x_q^i, \tau_3)$.
- 4. Send $(\mathsf{msg}_{4,i}, \widehat{\mathsf{msg}}_{4,i})$.

Output Computation: Each party P_i does the following:

- 1. Let τ_4 denote the transcript of the protocol Π up to the fourth round.
- 2. Compute the output of Π as $(\mathcal{G}, \{c^{i \to k}\}_{i \in [n], k \in [n]}, \{k_{\mathsf{mac}}^{i \to k}\}_{i \in [n], k \in [n]}) \leftarrow \mathsf{output}_i(x_g^i, \tau_4).$
- 3. Obtain the shares of the labels, along with the tags by computing $(\gamma_{\phi k}^{i \to k} || K_{\tau k}^{i \to k}) \leftarrow$ $\operatorname{dec}(\mathsf{k}_{\operatorname{enc}}^{i \to k, \phi^k}, \mathsf{c}_{\phi^k}^{i \to k}) \text{ for } i \in [n], k \in [n].$

4. Verify the correctness of the shares of the labels by checking that $\operatorname{Vrfy}_{k^{i\to k},\phi^{k}}(\gamma_{\phi^{k}}^{i\to k},K_{x^{k}}^{i\to k}) = 1$

- for $i \in [n], k \in [n]$. For each invalid pair $(\gamma_{\phi^j}^{k \to j}, K_{x^j}^{k \to j})$ output (abort, k).
- 5. Reconstruct the labels as $K_{x^k}^k = \bigoplus_{i \in [n]} K_{x^k}^{i \to k}$ for $k \in [n]$.
- 6. Output $y \leftarrow \text{eval}(\mathcal{G}, K^1_{x^1}, \dots, K^n_{x^n})$.

The Security Proofs 5.1

We first provide an informal description of the proof and later discuss the proof in detail. The simulator $\mathcal S$ of the compiled protocol Π internally executes the simulator of input protocol Π , which we denote by S_{Π} . To execute this simulator, we need to create a valid adversary for Π (i.e., an adversary that receives and sends only messages related to Π). We simulate such an adversary by means of an *augmented machine* \mathcal{M} . \mathcal{M} internally runs the proxy machine $\mathcal{P}(\mathcal{A})$, and it filters out only the messages related to Π , forwarding these to \mathcal{S}_{Π} . Upon receiving a message from \mathcal{S}_{Π} , \mathcal{M} enriches this message with all the other information that honest parties send in $\hat{\Pi}$ (this messages will be properly simulated as we will soon discuss), and forward everything to $\mathcal{P}(\mathcal{A})$. In summary, $\mathcal{P}(\mathcal{A})$ still receives messages of a format that is consistent with \hat{H} , while S_{II} believes that it is interacting with a valid adversary of Π (please see Figure 5.3 to get a better idea of how this interaction works).

Given that \mathcal{M} is a valid adversary for \mathcal{S}_{Π} , at some point \mathcal{S}_{Π} extracts the input that the adversary is using to execute Π . Let us denote this input by x_g^j , where j denotes the input of the corrupted party (for simplicity, here we assume that there is only one corrupted party). Note that x_q^j encodes the random bit d^{j} the adversary used to evaluate g. Our main simulator S can now infer the input of the adversary (the input for the functionality f) by computing $x^j \leftarrow \phi^j \oplus d^j$ (recall that ϕ^j is sent in the third round by the adversary). Having obtained x^j , S can query the ideal functionality f and simulate the garbled circuit (and the ciphertexts) accordingly. This simulated garbled circuit (and ciphertexts) will be used as input to S_{Π} (S is simulating the behaviour of g, which corresponds to the ideal functionality of S_{Π}).



Theorem 1. Assume Π is a four-round special-extractable malicious type secure multi-party computation protocol that adheres to Definition 10 and Definition 11, which realizes the functionality g, formally specified in the Figure 5.1 (type \in {sa-abort, si-abort, un-abort, id-abort}). Let GC be a garbling scheme in line with Definition 6. Let SE be a symmetric encryption scheme (Definition 1) that is computationally indistinguishable as per Definition 3. Let MAC be a message authentication code scheme (Definition 4) that is a one-time computationally-secure as per Definition 5. Let SS be an additive secret sharing scheme satisfying Definition 7. Then the protocol $\hat{\Pi}$ (in Figure 5.2) is malicious type delayed-input and adaptive-input selection secure.

Proof. The core of our proof revolves around demonstrating that for every non-uniform PPT adversary \mathcal{A} in the real world, a corresponding non-uniform probabilistic expected polynomial-time simulator \mathcal{S} that has black-box access to $\mathcal{P}(\mathcal{A})$ exists in the ideal world:

$$\operatorname{REAL}_{\hat{\Pi},\mathcal{J},\mathcal{A}(\operatorname{aux})}^{\operatorname{DI}}(1^{\lambda}) \stackrel{c}{\equiv} \operatorname{IDEAL}_{f,\mathcal{J},\mathcal{P}(\mathcal{A}),\mathcal{S}(\operatorname{aux})}^{\operatorname{DI},\operatorname{type}}(1^{\lambda}).$$

As just mentioned, when the protocol Π is employed alongside $\hat{\Pi}$, there is a necessity for an intermediary or bridge between these protocols. We refer to this intermediary as the *augmented machine* \mathcal{M}^8 . This machine can simulate the majority of the simulator messages autonomously. For the few messages it cannot simulate, they are directly inputted into \mathcal{M} by the simulator \mathcal{S} . A more profound reason for introducing the augmented machine \mathcal{M} is to address potential rewinds that the simulator of Π might undertake. It is crucial to emphasize that whenever the inner protocol initiates a rewind of the augmented machine, these rewind messages will be forwarded to the proxy machine $\mathcal{P}(\mathcal{A})$, and correspondingly rewinds the adversary \mathcal{A} and continues its execution. \mathcal{M} operates two interfaces:

- The left interface: It acts as a proxy between $\mathcal{P}(\mathcal{A})$ and its external interface the ideal-world simulator \mathcal{S}_{Π} (which exists by the security assumption of the protocol Π) w.r.t. the messages of Π ; or in the case of the hybrid experiment \mathcal{H}_0 , its external interface is the execution of the protocol Π (simulated by \mathcal{S}).
- The right interface: It interacts with the proxy machine $\mathcal{P}(\mathcal{A})$. $\mathcal{P}(\mathcal{A})$ will forward the messages to and from adversary \mathcal{A} (the real-world adversary of the protocol $\hat{\Pi}$).

It is important to note that Π is secure under standard MPC definition, but $\mathcal{P}(\mathcal{A})$ needs to receive the command ok to decide the actual input of honest parties. We also need to describe how the simulator \mathcal{S} generates this message. We specify the generation of ok in \mathcal{S} , and we let \mathcal{M} forward this message to $\mathcal{P}(\mathcal{A})$ in the extraction phase.

While \mathcal{M} interacts with the real-world adversary \mathcal{A} for $\hat{\Pi}$ through $\mathcal{P}(\mathcal{A})$, it also acts as the adversary of the protocol Π for \mathcal{S}_{Π} . The details of \mathcal{M} can be found in Figure 5.4, and the simulator \mathcal{S} is elaborated in Figure 5.5.

Figure 5.4: The augmented machine \mathcal{M}

First round: Upon receiving the message $\mathsf{msg}_{1,i}$ (for $i \in \mathcal{I}$) in the left interface, forward $\mathsf{msg}_{1,i}$ to the right interface.

Upon receiving the message $\mathsf{msg}_{1,j}$ (for $j \in \mathcal{J}$) as the reply from the right interface, send the message $\mathsf{msg}_{1,j}$ to the left interface.

Second round: Upon receiving the message $\mathsf{msg}_{2,i}$ (for $i \in \mathcal{I}$) in the left interface, forward $\mathsf{msg}_{2,i}$ to the right interface.

Upon receiving the message $\mathsf{msg}_{2,j}$ (for $j \in \mathcal{J}$) as the reply from the right interface, send the message $\mathsf{msg}_{2,j}$ to the left interface.

Third round: Upon receiving the message $\mathsf{msg}_{3,i}$ (for $i \in \mathcal{I}$) in the left interface, compute uniformly random $\phi^i \stackrel{\$}{\leftarrow} \{0,1\}$, set $\widehat{\mathsf{msg}}_{3,i} \leftarrow (\phi^i)$ as in the protocol $\hat{\mathcal{H}}$, and output the messages $(\mathsf{msg}_{3,i}, \widehat{\mathsf{msg}}_{3,i})$

⁸This is a standard technique used when proving the security of a protocol executed alongside another protocol. More examples of such techniques can be found in [CRSW22, GJO^+13 , COSW23].

to the right interface.

Upon receiving the messages $\mathsf{msg}_{3,j}$ and $\widehat{\mathsf{msg}}_{3,j}$ (for $j \in \mathcal{J}$) from the right interface, send the message $msg_{3,i}$ in the left interface.

Fourth round: Upon receiving the message $\mathsf{msg}_{4,i}$ (for $i \in \mathcal{I}$) in the left interface, compute $\mathsf{k}_{\mathsf{enc}}^{i \to k, b} \stackrel{\$}{\leftarrow} \mathsf{Gen}_{\mathsf{enc}}(1^{\lambda}) \text{ for } i \in \mathcal{I}, b \in \{0, 1\}, \text{ set } \widehat{\mathsf{msg}}_{4, i} \leftarrow (\{\mathsf{k}_{\mathsf{enc}}^{i \to k, \phi^k}\}_{k \in [n]}) \text{ for } i \in \mathcal{I} \text{ as in the protocol } \hat{\varPi}, and output the messages } (\mathsf{msg}_{4, i}, \widehat{\mathsf{msg}}_{4, i}) \text{ to the right interface.}$

Upon receiving the messages $\mathsf{msg}_{4,j}$ and $\widehat{\mathsf{msg}}_{4,j}$ (for $j \in \mathcal{J}$) from the right interface, send the message $msg_{4,i}$ in the left interface.

Extraction: Upon receiving the message ok in the left interface, forward it to the right interface. **Randomness passing:** Upon receiving the randomness R of adversary $\mathcal{P}(\mathcal{A})$ in the left interface, forward it to the right interface.

Figure 5.5: The Simulator \mathcal{S}

S firstly runs S_{Π} against the adversary \mathcal{M} . According to Definition 13, this interaction is through the proxy machine $\mathcal{P}(\mathcal{A})$, where $\mathcal{P}(\mathcal{A})$ interacts with ideal functionality f and adversary \mathcal{A} of protocol $\hat{\Pi}$. During this interaction, S_{Π} would be able to extract inputs from the corrupted parties. We assume the input from corrupted parties is $\{x_q^j\}_{j\in\mathcal{J}} = \{(d^j, \mathsf{k}_{\mathsf{enc}}^{j\to k, b})\}_{k\in[n], j\in\mathcal{J}, b\in\{0,1\}}$. \mathcal{S}_{Π} will extract the input and send it to \mathcal{S} . Then \mathcal{S} proceeds to perform the following steps:

- Upon receiving $\{x_g^j\}_{j\in\mathcal{J}}$ issued by \mathcal{S}_{Π} , send ok to \mathcal{M} , and do the following:

- 1. Retrieve ϕ^j from $\widehat{\mathsf{msg}}_{3,j}$ and set $x^j \leftarrow d^j \oplus \phi^j$ for $j \in \mathcal{J}$.
- 2. Query the ideal functionality f using x^j for $j \in \mathcal{J}$, and set $y \leftarrow f(x^1, \ldots, x^n)$.
- 3. Generate $(\mathcal{G}, \mathbf{K}) \leftarrow \mathtt{simGC}(1^{\lambda}, \mathbb{C}, y)$, where $\mathbf{K} = \{K_{\mathtt{sim}}^1, \ldots, K_{\mathtt{sim}}^n\}$. 4. Sample uniformly random $K_{\mathtt{sim}}^{i \to k}$ for $i \in [n], k \in [n]$ such that $\bigoplus_{i \in [n]} K_{\mathtt{sim}}^{i \to k} = K_{\mathtt{sim}}^k$.
- 5. Sample $\mathsf{k}_{\mathsf{mac}}^{i \to k, \phi^k}, \mathsf{k}_{\mathsf{mac}}^{i \to k, 1 \phi^k} \stackrel{\$}{\leftarrow} \mathtt{Gen}_{\mathsf{mac}}(1^{\lambda})$ for $i \in [n], k \in [n]$.
- 6. For each $i \in \mathcal{I}, k \in [n]$:

(a)
$$\gamma_{\phi^k}^{i \to k} \leftarrow \operatorname{Mac}_{\mathbf{k}_{\min}^{i \to k, \phi^k}}(K_{\min}^{i \to k})$$

(b)
$$c_{\phi^k}^{i \to k} \leftarrow \operatorname{enc}(\mathsf{k}_{enc}^{i \to k, \phi^k}, (\gamma_{\phi^k}^{i \to k} || K_{sim}^{i \to k})).$$

(c)
$$\mathsf{c}_{1 \ \phi^{k}}^{i \rightarrow k} \leftarrow \mathsf{enc}(\mathsf{k}_{\mathsf{enc}}^{i \rightarrow k, 1-\phi^{k}}, 0)$$

(d)
$$\mathbf{k}^{i \to k} \leftarrow (\mathbf{k}^{i \to k, 0}, \mathbf{k}^{i \to k, 1})$$
.

(e)
$$c^{i \to k} \leftarrow (c^{i \to k}, c^{i \to k})$$

7. For each
$$i \in \mathcal{J}, k \in [n]$$
:

- (a) Sample uniformly random $K_{1-\phi^k}^{j\to k} \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda}$. (b) $(\gamma_{\phi^k}^{j\to k}, \gamma_{1-\phi^k}^{j\to k}) \leftarrow (\operatorname{Mac}_{k_{\max}^{j\to k}, \phi^k}(K_{\sin}^{j\to k}), \operatorname{Mac}_{k_{\max}^{j\to k}, 1-\phi^k}(K_{1-\phi^k}^{j\to k}))$. (c) $(c_{\phi^k}^{j\to k}, c_{1-\phi^k}^{j\to k}) \leftarrow (\operatorname{enc}(k_{\operatorname{enc}}^{j\to k}, (\gamma_{\phi^k}^{j\to k} || K_{\sin}^{j\to k})), \operatorname{enc}(k_{\operatorname{enc}}^{j\to k, 1-\phi^k}, (\gamma_{1-\phi^k}^{j\to k} || K_{1-\phi^k}^{j\to k})))$.

(d)
$$k^{j \to k} \leftarrow (k^{j \to k, 0}, k^{j \to k, 1}).$$

(a)
$$\mathsf{r}_{\mathsf{mac}}$$
 ($\mathsf{r}_{\mathsf{mac}}$, $\mathsf{r}_{\mathsf{mac}}$).
(e) $\mathsf{c}^{j \to k} \leftarrow (\mathsf{c}_{0}^{j \to k}, \mathsf{c}_{1}^{j \to k})$.

$$(e) \ e^{i} \leftarrow (e_{0}, e_{1}).$$

8. Send $\mathcal{G}, \{c^{i \to k}\}_{i \in [n], k \in [n]}, \{k_{\max}^{i \to k}\}_{i \in [n], k \in [n]}$ to \mathcal{S}_{Π} .

- Continue running S_{Π} against the adversary \mathcal{M} , and outputs what S_{Π} outputs.

We now informally describe the following hybrid experiments and refer the reader to the Figure 5.7 for the formal specification.

 \mathcal{H}_0 : Hybrid \mathcal{H}_0 is identical to the real world.

- \mathcal{H}_1 : Hybrid \mathcal{H}_1 is the same as hybrid \mathcal{H}_0 but the messages $\{\mathsf{msg}_{i,k}\}_{k\in[4]}$ are computed by the simulator \mathcal{S}_{Π} . The transition between \mathcal{H}_0 and \mathcal{H}_1 is justified by the malicious type security of Π - if an adversary could distinguish between the two hybrid experiments, it could break the security of the protocol Π .
- \mathcal{H}_2 : Hybrid \mathcal{H}_2 is the same as hybrid \mathcal{H}_1 , but the ciphertexts $c_{1-\phi^j}^{i\to j}$ that P_j does not decrypt at the output computation phase (since it does not have the necessary keys) are encryption of 0. The transition between \mathcal{H}_1 and \mathcal{H}_2 is justified by the indistinguishability of the encryption scheme SE.
- \mathcal{H}_3 : Hybrid \mathcal{H}_3 is the same as hybrid \mathcal{H}_2 , but the ciphertexts $c_{1-\phi^i}^{j\to i}$ that P_j can decrypt but are not used at the output computation phase (since they do not correspond to P_i 's input) are encryption of the uniformly random sampled bit strings that have the same length as the real garbled labels. The transition between \mathcal{H}_2 and \mathcal{H}_3 is justified by the perfect security of the additive secret sharing scheme SS since the adversary can control at most n-1 parties, it can not collect enough shares to reconstruct the labels.
- \mathcal{H}_4 : Hybrid \mathcal{H}_4 is identical to the ideal world. In contrast to \mathcal{H}_3 , the labels $\{K_{x^k}^k\}_{k\in[n]}$ are generated by \mathcal{S} through the simGC function. The transition between \mathcal{H}_3 and \mathcal{H}_4 is justified by the privacy of the garbling scheme GC.

In hybrids $\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$, we use the augmented machine $\mathcal{M}_0(\cdot)$, and in hybrid \mathcal{H}_4 , we use the augmented machine $\mathcal{M}_4(\cdot)$. The above augmented machines have the same left interface as \mathcal{M} in Figure 5.4. The right interface is against different adversaries according to different hybrid experiments. The description of the augmented machines is in Figure 5.6.

Figure 5.6: The augmented machines $\mathcal{M}_0(\mathcal{A}), \mathcal{M}_4(\mathcal{A})$

First round: Upon receiving the message $\mathsf{msg}_{1,i}$ (for $i \in \mathcal{I}$) in the left interface, forward $\mathsf{msg}_{1,i}$ to $\mathcal{P}(\mathcal{A})$.

Upon receiving the message $\mathsf{msg}_{1,j}$ (for $j \in \mathcal{J}$) as the reply from $\mathcal{P}(\mathcal{A})$, send the message $\mathsf{msg}_{1,j}$ to the left interface.

Second round: Upon receiving the message $\mathsf{msg}_{2,i}$ (for $i \in \mathcal{I}$) in the left interface, forward $\mathsf{msg}_{2,i}$ to $\mathcal{P}(\mathcal{A})$.

Upon receiving the message $\mathsf{msg}_{2,j}$ (for $j \in \mathcal{J}$) as the reply from $\mathcal{P}(\mathcal{A})$, send the message $\mathsf{msg}_{2,j}$ to the left interface.

Third round: Upon receiving the message $\mathsf{msg}_{3,i}$ (for $i \in \mathcal{I}$) in the left interface, for $i \in \mathcal{I}$:

- Sample uniformly random $d^i \stackrel{\$}{\leftarrow} \{0,1\}$ and set $\phi^i \leftarrow d^i \oplus x^i$.

- Sample uniformly random $\phi^i \stackrel{\$}{\leftarrow} \{0, 1\}$.

Then set $\widehat{\mathfrak{msg}}_{3,i} \leftarrow (\phi^i)$ as in the protocol $\hat{\Pi}$, and output the messages $(\mathfrak{msg}_{3,i}, \widehat{\mathfrak{msg}}_{3,i})$ to $\mathcal{P}(\mathcal{A})$. Upon receiving the messages $\mathfrak{msg}_{3,j}$ and $\widehat{\mathfrak{msg}}_{3,j}$ (for $j \in \mathcal{J}$) from $\mathcal{P}(\mathcal{A})$, send the message $\mathfrak{msg}_{3,j}$ in the left interface. **Fourth round:** Upon receiving the message $\mathfrak{msg}_{4,i}$ (for $i \in \mathcal{I}$) in the left interface, compute $k_{\mathsf{enc}}^{i \to k, b} \stackrel{\$}{\leftarrow} \mathsf{Gen}_{\mathsf{enc}}(1^{\lambda})$ for $i \in \mathcal{I}, b \in \{0, 1\}$, set $\widehat{\mathfrak{msg}}_{4,i} \leftarrow (\{k_{\mathsf{enc}}^{i \to k, \phi^k}\}_{k \in [n]})$ for $i \in \mathcal{I}$ as in the protocol $\hat{\Pi}$, and output the messages $(\mathfrak{msg}_{4,i}, \widehat{\mathfrak{msg}}_{4,i})$ to $\mathcal{P}(\mathcal{A})$. Upon receiving the messages $\mathfrak{msg}_{4,j}$ and $\widehat{\mathfrak{msg}}_{4,j}$ (for $j \in \mathcal{J}$) from $\mathcal{P}(\mathcal{A})$, send the message $\mathfrak{msg}_{4,j}$ in the left interface. **Extraction:** Upon receiving the message ok in the left interface, forward it to $\mathcal{P}(\mathcal{A})$. **Randomness passing:** Upon receiving the randomness R of adversary $\mathcal{P}(\mathcal{A})$ in the left interface,

forward it to $\mathcal{P}(\mathcal{A})$.

We formally describe the hybrid experiments in Figure 5.7. Before delving into the details, we provide brief instructions on how to interpret the figure. The primary guideline is to prioritize the most restrictive

option for each step when multiple choices are available for a specific hybrid. For instance, in \mathcal{H}_2 , we select the second option from the three available for step 2(c), as it is more constrained compared to the others. Additionally, we cannot choose the third option since it falls within a rectangle box, which exceeds the scope of \mathcal{H}_2 . Consequently, \mathcal{H}_2 incorporates steps 1, 2.(a), 2.(b), 2.(d), 2.(e).i, 2.(e).ii from the rounded box, and 2.(c), 2.(e).iii from the dashed box. Thus, the points of divergence between \mathcal{H}_2 and \mathcal{H}_1 are steps 2.(c) and 2.(e).iii. Similarly, we can discern the discrepancies between each hybrid experiment using this approach.



$$\begin{array}{|c|c|c|c|c|} \hline & \text{iii. } \mathbf{c}_{1-\phi^{k}}^{i\rightarrow k} \leftarrow \operatorname{enc}(\mathbf{k}_{enc}^{i\rightarrow k,1-\phi^{k}},0). \\ \hline & \text{iii. } \mathbf{c}_{1-\phi^{k}}^{j\rightarrow k} \leftarrow \operatorname{enc}(\mathbf{k}_{enc}^{i\rightarrow k,1-\phi^{k}},0). \\ \hline & \text{i. } (\gamma_{\phi^{k}}^{j\rightarrow k},\gamma_{1-\phi^{k}}^{j\rightarrow k}) \leftarrow (\operatorname{Mac}_{\mathbf{k}_{mac}^{j\rightarrow k,\phi^{k}}}(K_{x^{k}}^{j\rightarrow k}),\operatorname{Mac}_{\mathbf{k}_{mac}^{j\rightarrow k,1-\phi^{k}}}(K_{1-x^{k}}^{j\rightarrow k})). \\ \hline & \text{ii. } (\mathbf{c}_{\phi^{k}}^{j\rightarrow k},\mathbf{c}_{1-\phi^{k}}^{j\rightarrow k}) \leftarrow (\operatorname{enc}(\mathbf{k}_{enc}^{j\rightarrow k,\phi^{k}},(\gamma_{\phi^{k}}^{j\rightarrow k}||K_{\phi^{k}}^{j\rightarrow k})),\operatorname{enc}(\mathbf{k}_{enc}^{j\rightarrow k,1-\phi^{k}},(\gamma_{1-\phi^{k}}^{j\rightarrow k}||K_{1-\phi^{k}}^{j\rightarrow k}))). \\ \hline & \text{i. } (\gamma_{\phi^{k}}^{j\rightarrow k},\gamma_{1-\phi^{k}}^{j\rightarrow k}) \leftarrow (\operatorname{enc}(\mathbf{k}_{enc}^{j\rightarrow k,\phi^{k}},(\gamma_{\phi^{k}}^{j\rightarrow k}||K_{1-\phi^{k}}^{j\rightarrow k})),\operatorname{enc}(\mathbf{k}_{enc}^{j\rightarrow k,1-\phi^{k}},(\gamma_{1-\phi^{k}}^{j\rightarrow k}||K_{1-\phi^{k}}^{j\rightarrow k}))). \\ \hline & \text{i. } (\mathbf{c}_{\phi^{k}}^{j\rightarrow k},\mathbf{c}_{1-\phi^{k}}^{j\rightarrow k}) \leftarrow (\operatorname{enc}(\mathbf{k}_{enc}^{j\rightarrow k,\phi^{k}},(\gamma_{\phi^{k}}^{j\rightarrow k}||K_{sim}^{j\rightarrow k})),\operatorname{enc}(\mathbf{k}_{enc}^{j\rightarrow k,1-\phi^{k}},(\gamma_{1-\phi^{k}}^{j\rightarrow k}||K_{1-\phi^{k}}^{j\rightarrow k})))). \\ \hline & \text{i. } (\mathbf{c}_{\phi^{k}}^{j\rightarrow k},\mathbf{c}_{1-\phi^{k}}^{j\rightarrow k}) \leftarrow (\operatorname{enc}(\mathbf{k}_{enc}^{j\rightarrow k},(\gamma_{\phi^{k}}^{j\rightarrow k}||K_{sim}^{j\rightarrow k})),\operatorname{enc}(\mathbf{k}_{enc}^{j\rightarrow k,1-\phi^{k}},(\gamma_{1-\phi^{k}}^{j\rightarrow k}||K_{1-\phi^{k}}^{j\rightarrow k})))). \\ \hline & \text{i. } (\mathbf{c}_{\phi^{k}}^{j\rightarrow k},\mathbf{c}_{1-\phi^{k}}^{j\rightarrow k}) \leftarrow (\operatorname{enc}(\mathbf{k}_{enc}^{j\rightarrow k},(\gamma_{\phi^{k}}^{j\rightarrow k}||K_{sim}^{j\rightarrow k})),\operatorname{enc}(\mathbf{k}_{enc}^{j\rightarrow k},(\gamma_{1-\phi^{k}}^{j\rightarrow k}||K_{1-\phi^{k}}^{j\rightarrow k})))). \\ \hline & \text{i. } (\mathbf{c}_{\phi^{k}}^{j\rightarrow k},\mathbf{c}_{1-\phi^{k}}^{j\rightarrow k}), \mathbf{c}_{enc}^{i\rightarrow k} \leftarrow (\mathbf{c}_{0}^{i\rightarrow k},\mathbf{c}_{1}^{i\rightarrow k}) \text{ for } i\in[n], k\in[n]. \\ \hline & \text{i. } (\mathbf{h}) \text{ Send } \mathcal{G}, \{\mathbf{c}^{i\rightarrow k}\}_{i\in[n],k\in[n]}, \{\mathbf{k}_{mac}^{i\rightarrow k}\}_{i\in[n],k\in[n]} \text{ to } \mathcal{S}_{\Pi}. \\ \hline & \text{i. } Continue running \mathcal{S}_{\Pi}. \\ \hline & \text{i. } Continue running \mathcal{S}_{\Pi}. \\ \hline & \text{i. } (\mathbf{h}) \text{ for }$$

Lemma 1 (Transition from \mathcal{H}_0 to \mathcal{H}_1). Let Π be a malicious type secure multi-party computation protocol satisfying Definition 10, against t < n, which realizes the functionality g, formally specified in the Figure 5.1 (type $\in \{\text{sa-abort}, \text{si-abort}, \text{un-abort}, \text{id-abort}\}$). Then the output distribution of the hybrid experiments \mathcal{H}_0 and \mathcal{H}_1 are computationally indistinguishable.

Proof. Since we assume that Π is malicious type secure, we know that for every non-uniform PPT adversary \mathcal{A} for the real world, with auxiliary input aux, there exists a non-uniform expected PPT simulator \mathcal{S} , it holds that

$$\operatorname{REAL}_{\Pi,\mathcal{J},\mathcal{A}(\operatorname{aux})}(x,1^{\lambda}) \stackrel{c}{\equiv} \operatorname{IDEAL}_{f,\mathcal{J},\mathcal{S}(\operatorname{aux})}^{\operatorname{type}}(x,1^{\lambda}).$$

Let us assume that such an adversary \mathcal{A}_0 exists that can distinguish between hybrid experiments \mathcal{H}_0 and \mathcal{H}_1 with non-negligible probability. Then, we can construct the following adversary $\mathcal{A}'_0 = \mathcal{M}_0(\mathcal{A}_0)$ that, by the assumption, breaks the security of the protocol Π . According to the description of $\mathcal{M}_0(\mathcal{A}_0)$, formally described in Figure 5.6, the adversary $\mathcal{A}'_0 = \mathcal{M}_0(\mathcal{A}_0)$ is a valid adversary for Π .

We define the following reduction with a challenger with the black-box access to \mathcal{A}'_0 . This challenger interacts with \mathcal{A}'_0 either using the messages of the simulator \mathcal{S}_{Π} - meaning that the output of \mathcal{A}'_0 corresponds to the output \mathcal{H}_1 , or using the messages of the protocol Π generated accordingly to the honest procedure meaning that the output of \mathcal{A}'_0 corresponds to the output \mathcal{H}_0 .

However, if there would be such an adversary \mathcal{A}_0 that would be able to distinguish between the two hybrid experiments, then \mathcal{A}'_0 would be able to break the security of the protocol Π , which contradicts the assumption that Π is a malicious type secure multi-party computation protocol.

Lemma 2 (Transition from \mathcal{H}_1 to \mathcal{H}_2). Let SE be a symmetric encryption scheme (Definition 1), that is computationally indistinguishable as per Definition 3. Then the output distribution of the hybrid experiments \mathcal{H}_1 and \mathcal{H}_2 are computationally indistinguishable.

Proof. For each $t \in [n]$ we construct a hybrid experiment \mathcal{H}_1^t in the following way:

- For each t' < t, ciphertexts $\{c_{1-\phi^{t'}}^{i \to t'}\}_{i \in \mathcal{I}}$ are computed as in the hybrid \mathcal{H}_2 .
- For each t' > t, ciphertexts $\{c_{1-\phi^{t'}}^{i \to t'}\}_{i \in \mathcal{I}}$ are computed as in the hybrid \mathcal{H}_1 .
- For t' = t, ciphertexts $\{c_{1-\phi^{t'}}^{i\to t'}\}_{i\in\mathcal{I}}$ are computed either as in the hybrid \mathcal{H}_1 or as in the hybrid \mathcal{H}_2 , depending on the bit chosen by the challenger of the PrivK^{eav} experiment.

This means that \mathcal{H}_1^0 is equivalent to \mathcal{H}_1 and \mathcal{H}_1^n is equivalent to \mathcal{H}_2 . Now, for each $1 \leq t \leq n$ we show that \mathcal{H}_1^{t-1} is computationally indistinguishable from \mathcal{H}_1^t and thereby prove that \mathcal{H}_1 is computationally indistinguishable from \mathcal{H}_2 .

Because of the nature of our protocol, we will use a modified $\mathtt{PrivK}_{\mathcal{A},SE}^{eav}$ experiment where the adversary is tasked with supplying n distinct pairs of messages, denoted as $((m_0^1, m_1^1), \dots, (m_0^n, m_1^n))$. Then the challenger samples $k^i \stackrel{\$}{\leftarrow} \operatorname{Gen}_{\operatorname{enc}}(1^{\lambda})$ for $i \in [n]$, selects a random bit b and outputs $(c^1, \ldots, c^n) =$ $(\operatorname{enc}(k^1, m_b^1), \ldots, \operatorname{enc}(k^n, m_b^n))$ (here, we rely that the computationally indistinguishable encryption scheme is sequentially composable). Let us assume that there exists such an PPT adversary \mathcal{A}_1^t who can distinguish between the hybrid experiments \mathcal{H}_1^{t-1} and \mathcal{H}_1^t . We show an adversary $\mathcal{A}_1'^t$ for the modified $\operatorname{Priv}_{\mathcal{A}_1'^t, SE}^{eav}$ experiment which works as follows:

- Simulate the following experiment for the adversary $\mathcal{A}_{1}^{\prime,t}$:
 - Run \mathcal{S}_{Π} against the adversary $\mathcal{M}_0(\mathcal{A}_1^t)$.
 - Upon receiving a set of queries $\{x_q^j\}_{j\in\mathcal{J}} = \{(d^j, \mathsf{k}_{\mathsf{enc}}^{j\to k,b})\}_{k\in[n], j\in\mathcal{J}, b\in\{0,1\}}$ issued by \mathcal{S}_{Π} , send ok to \mathcal{M}_0 and do the following:
 - 1. Retrieve ϕ^j from $\widehat{\mathsf{msg}}_{3,j}$ and set $x^j \leftarrow d^j \oplus \phi^j$ (for $j \in \mathcal{J}$).

 - 2. Generate $(\mathcal{G}, \mathbf{K}) \stackrel{\$}{\leftarrow} \operatorname{garble}(1^{\lambda}, \mathbb{C})$ where $\mathbf{K} = \{K_b^1, \dots, K_b^n\}$ and $b \in \{0, 1\}$. 3. Sample uniformly random $K_b^{i \to k}$ for $i \in [n], k \in [n], b \in \{0, 1\}$ such that $\bigoplus_{i \in [n]} K_b^{i \to k} = K_b^k$.
 - 4. Sample $\mathsf{k}_{\mathsf{mac}}^{i \to k, \phi^k}, \mathsf{k}_{\mathsf{mac}}^{i \to k, 1 \phi^k} \overset{\$}{\leftarrow} \mathtt{Gen}_{\mathsf{mac}}(1^{\lambda}) \text{ for } i \in [n], k \in [n].$ 5. For each $i \in \mathcal{I}, k \in [t-1]$: (a) $\gamma_{\phi^k}^{i \to k} \leftarrow \mathtt{Mac}_{\mathsf{k}_{\mathsf{mac}}^{i \to k, \phi^k}}(K_{x^k}^{i \to k}).$

(b)
$$\mathbf{c}_{i,k}^{i \to k} \leftarrow \mathbf{enc}(\mathbf{k}_{apc}^{i \to k,\phi^{k}}, (\gamma_{i,k}^{i \to k} || K_{i,k}^{i \to k})).$$

- $\begin{array}{l} (\mathrm{b}) \ \mathbf{c}_{\phi^{k}}^{i \to k} \leftarrow \mathrm{enc}(\mathbf{k}_{\mathrm{enc}}^{i \to k, \phi^{*}}, (\gamma_{\phi^{k}}^{i \to k} || K_{x^{k}}^{i \to k})). \\ (\mathrm{c}) \ \mathbf{c}_{1-\phi^{k}}^{i \to k} \leftarrow \mathrm{enc}(\mathbf{k}_{\mathrm{enc}}^{i \to k, 1-\phi^{k}}, 0). \\ \end{array} \\ \begin{array}{l} 6. \ \mathrm{For \ each} \ i \in \mathcal{I}, k \in [n] \setminus [t]: \\ (\mathrm{a}) \ (\gamma_{\phi^{k}}^{i \to k}, \gamma_{1-\phi^{k}}^{i \to k}) \leftarrow (\mathrm{Mac}_{\mathbf{k}_{\mathrm{mac}}^{i \to k, \phi^{k}}}(K_{x^{k}}^{i \to k}), \mathrm{Mac}_{\mathbf{k}_{\mathrm{mac}}^{i \to k, 1-\phi^{k}}}(K_{1-x^{k}}^{i \to k})). \\ (\mathrm{b}) \ \mathbf{c}_{\phi^{k}}^{i \to k} \leftarrow \mathrm{enc}(\mathbf{k}_{\mathrm{enc}}^{i \to k, \phi^{k}}, (\gamma_{\phi^{k}}^{i \to k} || K_{x^{k}}^{i \to k})). \end{array} \end{array}$

(c)
$$\mathbf{c}_{1-\phi^k}^{i \to k} \leftarrow \operatorname{enc}(\mathbf{k}_{\operatorname{enc}}^{i \to k, 1-\phi^k}, (\gamma_{1-\phi^k}^{i \to k} || K_{1-x^k}^{i \to k})).$$

- 7. For each $i \in \mathcal{I}$:
 - (a) $(\gamma_{\phi^t}^{i \to t}, \gamma_{1-\phi^t}^{i \to t}) \leftarrow (\operatorname{Mac}_{\mathsf{k}_{\mathsf{mac}}^{i \to t}, \phi^t}(K_{x^t}^{i \to t}), \operatorname{Mac}_{\mathsf{k}_{\mathsf{mac}}^{i \to t}, 1-\phi^t}(K_{1-x^t}^{i \to t})).$ (b) Set $m_0^i = \{(\gamma_{1-\phi^t}^{i \to t} || K_{1-x^t}^{i \to t})\}_{i \in [n]}$ and $m_1^i = 0.$
 - (c) Send (m_0^i, m_1^i) to the challenger of the PrivK^{eav}_{A',SE} experiment and receive challenge ciphertext $\begin{array}{l} c^{i} \text{ as a reply.} \\ (\mathrm{d}) \ \mathbf{c}_{\phi^{t}}^{i \to t} \leftarrow \mathbf{enc}(\mathbf{k}_{\mathsf{enc}}^{i \to t,\phi^{t}},(\gamma_{\phi^{t}}^{i \to t}||K_{x^{t}}^{i \to t})). \end{array}$

(e)
$$\mathbf{c}_{1-\phi^t}^{i \to t} \leftarrow c^i$$

- (e) $\mathbf{c}_{1-\phi^{i}}^{j \to i} \leftarrow c^{*}$. 8. For each $j \in \mathcal{J}, k \in [n]$: (a) $(\gamma_{\phi^{k}}^{j \to k}, \gamma_{1-\phi^{k}}^{j \to k}) \leftarrow (\operatorname{Mac}_{\mathsf{k}_{mac}^{j \to k}, \phi^{k}}(K_{\phi^{k}}^{j \to k}), \operatorname{Mac}_{\mathsf{k}_{mac}^{j \to k}, 1-\phi^{k}}(K_{1-\phi^{k}}^{j \to k}))$. (b) $(\mathbf{c}_{\phi^{k}}^{j \to k}, \mathbf{c}_{1-\phi^{k}}^{j \to k}) \leftarrow (\operatorname{enc}(\mathsf{k}_{enc}^{j \to k}, \phi^{k}, (\gamma_{\phi^{k}}^{j \to k} || K_{\phi^{k}}^{j \to k})), \operatorname{enc}(\mathsf{k}_{enc}^{j \to k}, 1-\phi^{k}, (\gamma_{1-\phi^{k}}^{j \to k} || K_{1-\phi^{k}}^{j \to k})))$. 9. For $i \in [n], k \in [n]$: (a) $\mathsf{k}_{mac}^{i \to k} \leftarrow (\mathsf{k}_{mac}^{i \to k}, 0, \mathsf{k}_{mac}^{i \to k})$. (b) $\mathbf{c}^{i \to k} \leftarrow (\mathbf{c}_{0}^{i \to k}, \mathbf{c}_{1}^{i \to k})$. 10. Send $\mathcal{G}, \{\mathbf{c}^{i \to k}\}_{i \in [n], k \in [n]}, \{\mathsf{k}_{mac}^{i \to k}\}_{i \in [n], k \in [n]} \text{ to } \mathcal{S}_{\Pi}$.
- Continue running S_{II} against the adversary $M_0(\mathcal{A}_1^t)$ and output whatever $M_0(\mathcal{A}_1^t)$ outputs.

Notice that random bit b of the $\operatorname{PrivK}_{\mathcal{A}_1^{\prime,t}, \operatorname{SE}}^{\operatorname{eav}}$ which the challenger samples, decides which of the \mathcal{H}_1^{t-1} or \mathcal{H}_1^t the adversary $\mathcal{A}_1^{\prime,t}$ simulates from the adversary $M_0(\mathcal{A}_1^t)$. That is, if $b = 0, \mathcal{A}_1^{\prime,t}$ simulates the experiment identical to the hybrid \mathcal{H}_1^{t-1} ; if b = 1, then the simulated experiment is identical to the hybrid \mathcal{H}_1^t . We can see that if \mathcal{A}_1^t can distinguish between the hybrid experiments \mathcal{H}_1^{t-1} and \mathcal{H}_1^t , then $\mathcal{A}_1'^t$ can break the security of the encryption scheme SE. However, since SE is a computationally indistinguishable encryption scheme, such a PPT adversary cannot exist.

We note that in order for the reduction to remain polynomial time, we need to bound the running time of the simulator S_{Π} . Since the simulator S_{Π} is expected to run in polynomial time, we limit its execution to l steps, where l represents the expected polynomial running time of S_{II} . Therefore, with non-negligible probability, the simulator will stop.

Lemma 3 (Transition from \mathcal{H}_2 to \mathcal{H}_3). Let SS be an additive secret sharing scheme satisfying the Definition 7. Then the output distribution of the hybrid experiments \mathcal{H}_2 and \mathcal{H}_3 are computationally indistinquishable.

Proof. We note that the difference between \mathcal{H}_2 and \mathcal{H}_3 is that, the ciphertext $c_{\phi^i}^{j \to i}$ for $i \in \mathcal{I}, j \in \mathcal{J}$ is an encryption of the uniformly random λ -bit long string, and these bit strings have no relation with the garbled labels $\mathbf{K}_{1-x^i}^i$. The corrupted party P_j has the ability to open both $\mathbf{c}_{\phi^i}^{j \to i}$ and $\mathbf{c}_{1-\phi^i}^{j \to i}$. However, because the adversary can control at most n-1 parties, it can not collect enough shares to reconstruct the original labels. Therefore, due to the perfect security of additive secret sharing scheme, \mathcal{H}_2 and \mathcal{H}_3 are computationally indistinguishable.

Lemma 4 (Transition from \mathcal{H}_3 to \mathcal{H}_4). Let the GC be a garbling scheme satisfying the Definition 6. Then the output distribution of the hybrid experiments \mathcal{H}_3 and \mathcal{H}_4 are computationally indistinguishable.

Proof. Notice that the only difference between \mathcal{H}_3 and \mathcal{H}_4 are the values encrypted in ciphertexts $c_{\phi^i}^i$ for $i \in \mathcal{I}$ which are replaced by the labels that the simGC outputs in the \mathcal{H}_4 .

Let us assume that there exists such an adversary \mathcal{A} who can distinguish between the hybrid experiments \mathcal{H}_3 and \mathcal{H}_4 . We show an adversary \mathcal{A}' for the $\mathsf{GC}_{\mathcal{A}'}^{\mathsf{priv}}$ which works as follows:

– Simulate the following experiment for the adversary \mathcal{A}' :

- Run \mathcal{S}_{Π} against the adversary $\mathcal{M}_4(\mathcal{A})$.
- Upon receiving a set of queries $\{x_a^j\}_{j\in\mathcal{J}} = \{(d^j, k_{enc}^{j\to k,b})\}_{k\in[n], j\in\mathcal{J}, b\in\{0,1\}}$ issued by \mathcal{S}_{Π} , send ok to \mathcal{M}_4 and do the following:
 - 1. Retrieve ϕ^j from $\widehat{\mathsf{msg}}_{3,j}$ and set $x^j \leftarrow d^j \oplus \phi^j$ (for $j \in \mathcal{J}$).
 - 2. Construct the circuit \tilde{C} of the function f and send it along with the input $x = (x^1, \ldots, x^n)$ to the challenger.

 - Receive G, (K¹_{sim},...,Kⁿ_{sim}) as a reply from the challenger.
 Sample uniformly random K^{i→k}_{sim} for i ∈ [n], k ∈ [n] such that ⊕_{i∈[n]} K^{i→k}_{sim} = K^k_{sim}.
 - 5. Sample $\mathsf{k}_{\mathsf{mac}}^{i \to k, \phi^k}, \mathsf{k}_{\mathsf{mac}}^{i \to k, 1 \phi^k} \overset{\$}{\leftarrow} \mathsf{Gen}_{\mathsf{mac}}(1^{\lambda})$ for $i \in [n], k \in [n]$. 6. For each $i \in \mathcal{I}, k \in [n]$:

(a)
$$\gamma_{\phi^k}^{i \to k} \leftarrow \operatorname{Mac}_{\mathbf{k}^{i \to k, \phi^k}}(K_{\operatorname{sim}}^{i \to k}).$$

- (b) $c_{\phi^k}^{i \to k} \leftarrow \operatorname{enc}(k_{\operatorname{enc}}^{i \to k,\phi^k}, (\gamma_{\phi^k}^{i \to k} || K_{\operatorname{sim}}^{i \to k})).$ (c) $c_{1-\phi^k}^{i \to k} \leftarrow \operatorname{enc}(k_{\operatorname{enc}}^{i \to k,1-\phi^k}, 0).$

(d)
$$k^{i \to k} \leftarrow (k^{i \to k, 0} \ k^{i \to k, 1})$$

- $\begin{array}{ll} (\mathrm{d}) \ \mathsf{k}_{\mathsf{mac}}^{i \to k} \leftarrow (\mathsf{k}_{\mathsf{mac}}^{i \to k, \mathrm{U}}, \mathsf{k}_{\mathsf{mac}}^{i \to k, \mathrm{I}}). \\ (\mathrm{e}) \ \mathsf{c}^{i \to k} \leftarrow (\mathsf{c}_{0}^{i \to k}, \mathsf{c}_{1}^{i \to k}). \end{array}$
- 7. For each $j \in \mathcal{J}, k \in [n]$:
 - (a) Sample uniformly random $K_{1-\phi^k}^{j\to k} \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda}$.

(b)
$$(\gamma_{j \to k}^{j \to k}, \gamma_{1 \to j \to k}^{j \to k}) \leftarrow (\operatorname{Mac}_{j \to k \to k}(K_{\mathsf{cir}}^{j \to k}), \operatorname{Mac}_{j \to k \to j \to k}(K_{1 \to j \to k}^{j \to k})).$$

 $\begin{array}{l} \text{(b)} \quad (\gamma_{\phi^k}^{j \to k}, \gamma_{1-\phi^k}^{j \to k}) \leftarrow (\texttt{Mac}_{\mathsf{k}_{\mathsf{mac}}^{j \to k, \phi^k}}(K_{\mathsf{sim}}^{j \to \kappa}), \texttt{Mac}_{\mathsf{k}_{\mathsf{mac}}^{j \to k, 1-\phi^k}}(K_{1-\phi^k}^{j \to \kappa})). \\ \text{(c)} \quad (\mathsf{c}_{\phi^k}^{j \to k}, \mathsf{c}_{1-\phi^k}^{j \to k}) \leftarrow (\mathsf{enc}(\mathsf{k}_{\mathsf{enc}}^{j \to k}, (\gamma_{\phi^k}^{j \to k} || K_{\mathsf{sim}}^{j \to k})), \mathsf{enc}(\mathsf{k}_{\mathsf{enc}}^{j \to k, 1-\phi^k}, (\gamma_{1-\phi^k}^{j \to k} || K_{1-\phi^k}^{j \to k}))). \end{array}$

(d)
$$k_{mac}^{j \to k} \leftarrow (k_{mac}^{j \to k,0}, k_{mac}^{j \to k,1}).$$

(a)
$$k_{\text{mac}} \leftarrow (k_{\text{mac}}, k_{\text{mac}})$$

(e) $c^{j \to k} \leftarrow (c_0^{j \to k}, c_1^{j \to k})$. - Send $\mathcal{G}, \{c^{i \to k}\}_{i \in [n], k \in [n]}, \{k_{mac}^{i \to k}\}_{i \in [n], k \in [n]}$ to \mathcal{S}_{Π} . - Continue running \mathcal{S}_{Π} against the adversary $\mathcal{M}_4(\mathcal{A})$ and output whatever $\mathcal{M}_4(\mathcal{A})$ outputs.

Notice that the random bit b of the $GC_{\mathcal{A}'}^{priv}$ experiment, which the challenger samples, decides which of the \mathcal{H}_3 or \mathcal{H}_4 the adversary \mathcal{A}' simulates for the adversary $\mathcal{M}_4(\mathcal{A})$. That is, if $b = 0, \mathcal{A}'$ simulates the experiment identical to the hybrid \mathcal{H}_3 ; if b = 1, then the simulated experiment is identical to the hybrid \mathcal{H}_4 . We can see that if \mathcal{A} can distinguish between the hybrid experiments \mathcal{H}_3 and \mathcal{H}_4 , then \mathcal{A}' can break the privacy of the garbling scheme GC. However, since we assume that GC satisfies the Definition 6, we come toward a contradiction.

Again, in order for the reduction to remain polynomial time, we need to bound the running time of the simulator S_{Π} . Since the simulator S_{Π} is expected to run in polynomial time, we limit its execution to p steps, where p represents the expected polynomial running time of S_{II} . Therefore, with non-negligible probability, the simulator will stop.

Before completing this proof, we must show that no party can cheat and cause the other party to compute a wrong output. In the protocol, this is solved by each party verifying the labels and seeing whether they correspond to the key and permutation bit that the other party sent. We prove this in the following claim by showing that the probability that a different output is computed in the real world and ideal world is negligible.

Lemma 5. Let MAC be a message authentication code scheme (Definition 4) that is a one-time computationallysecure as per Definition 5. Then the probability that S queries the ideal functionality f and obtains the output y and that the real-world execution of the protocol $\hat{\Pi}$ outputs $f(\cdot,\ldots,\cdot) \neq y$ is negligible in the security parameter.

Proof. Let us assume that the lemma does not hold. By the construction of our compiler, the honest parties obtain incorrect outputs means that the adversary sends incorrect shares of garbled labels s.t. when the honest parties reconstruct and evaluate the garble circuits, they obtain some values that are not computed from the input of the adversary. Note that in our compiler, the shares of labels is computed by decrypting the cipher texts from the functionality q. In the functionality q, every possible share of labels are authenticated. By the security of underlying MPC protocol Π , g is correctly realized.

Then if there exists an adversary \mathcal{A} that can make the honest parties obtain incorrect output of function f. We show an adversary \mathcal{A}_{MAC} for the $Mac - forge_{\mathcal{A},\Pi}^{1-time}$ experiment which works as in the following. Notice that although the simulation is exactly as the one in the \mathcal{H}_4 , in order to obtain the ciphertexts that the honest parties would receive at the end of the computation, we need to simulate the honest parties as well. This means that in the simulation, the adversary \mathcal{A}_{MAC} is controlling the whole set of parties $\{P_i\}_{i \in \mathcal{I}}$.

- Sample uniformly random $d^i \stackrel{\$}{\leftarrow} \{0,1\}$ for $i \in \mathcal{I}$ and $k_{enc}^{i \to k,b}$ for $b \in \{0,1\}, i \in \mathcal{I}, k \in [n]$. Upon receiving $\{x_g^j\}_{j \in \mathcal{J}}$ issued by \mathcal{S}_{Π} , send ok to \mathcal{M}_4 and do the following:

 - 1. Retrieve ϕ^j from $\widehat{\mathsf{msg}}_{3,j}$ and compute $x^j = d^j \oplus \phi^j$ (for $j \in \mathcal{J}$).
 - 2. Query the ideal functionality f using x^j for $j \in \mathcal{J}$, and get $y \leftarrow f(x^1, \ldots, x^n)$.

 - 3. Generate $(\mathcal{G}, \mathbf{K}) \stackrel{\$}{\leftarrow} \operatorname{sim}\operatorname{GC}(1^{\lambda}, \mathbb{C}, y)$, where $\mathbf{K} = \{K^{1}_{\operatorname{sim}}, \dots, K^{n}_{\operatorname{sim}}\}$. 4. Sample uniformly random $K^{i \to k}_{\operatorname{sim}}$ for $i \in [n], k \in [n]$ such that $\bigoplus_{i \in [n]} K^{i \to k}_{\operatorname{sim}} = K^{k}_{\operatorname{sim}}$.
 - 5. Send K^{j'→i'}_{sim} to the challenger of the Mac forge^{1-time}_{A,Π} experiment and receive γ^{j'→i'} as a reply.
 6. Sample uniformly random K^{i→k}_{sim} for i ∈ [n], k ∈ [n] such that ⊕_{i∈[n]} K^{i→k}_{sim} = K^k_{sim}.

 - 7. For each $i \in \mathcal{I}, k \in [n]$: (a) $\gamma_{\phi^k}^{i \to k} \leftarrow \operatorname{Mac}_{\mathsf{k}_{\mathsf{mac}}^{i \to k, \phi^k}}(K_{\mathtt{sim}}^{i \to k})$.
 - (b) $\mathbf{c}_{\phi^k}^{i \to k} \leftarrow \mathbf{enc}(\mathbf{k}_{\mathtt{enc}}^{i \to k, \phi^k}, (\gamma_{\phi^k}^{i \to k} || K_{\mathtt{sim}}^{i \to k}))$

 - (c) $\mathbf{c}_{1-\phi^{k}}^{i\to k} \leftarrow \operatorname{enc}(\mathbf{k}_{\operatorname{enc}}^{i\to k,1-\phi^{k}}, 0).$ (d) $\mathbf{k}_{\operatorname{mac}}^{i\to k} \leftarrow (\mathbf{k}_{\operatorname{mac}}^{i\to k,0}, \mathbf{k}_{\operatorname{mac}}^{i\to k,1}).$ (e) $\mathbf{c}^{i\to k} \leftarrow (\mathbf{c}_{0}^{i\to k}, \mathbf{c}_{1}^{i\to k}).$
 - 8. For each $j \in \mathcal{J}, k \in [n]$:
 - (a) Sample $K_{1-\phi^k}^{j \to k} \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda}$. $\begin{array}{l} \text{(b)} \quad (\gamma_{\phi^k}^{j \to k}, \gamma_{1-\phi^k}^{j \to k}) \leftarrow (\operatorname{Mac}_{\mathsf{k}_{\mathsf{mac}}^{j \to k, \phi^k}}(K_{\mathsf{sim}}^{j \to k}), \operatorname{Mac}_{\mathsf{k}_{\mathsf{mac}}^{j \to k, 1-\phi^k}}(K_{1-\phi^k}^{j \to k})). \\ \text{(c)} \quad (\mathsf{c}_{\phi^k}^{j \to k}, \mathsf{c}_{1-\phi^k}^{j \to k}) \leftarrow (\operatorname{enc}(\mathsf{k}_{\mathsf{enc}}^{j \to k, \phi^k}, (\gamma_{\phi^k}^{j \to k} || K_{\mathsf{sim}}^{j \to k})), \operatorname{enc}(\mathsf{k}_{\mathsf{enc}}^{j \to k, 1-\phi^k}, (\gamma_{1-\phi^k}^{j \to k} || K_{1-\phi^k}^{j \to k}))). \\ \text{(d)} \quad \mathsf{k}_{\mathsf{mac}}^{j \to k} \leftarrow (\mathsf{k}_{\mathsf{mac}}^{j \to k, 0}, \mathsf{k}_{\mathsf{mac}}^{j \to k, 1}). \end{array}$

- (e) $\mathbf{c}^{j \to k} \leftarrow (\mathbf{c}_0^{j \to k}, \mathbf{c}_1^{j \to k}).$ 9. Send $\mathcal{G}, \{\mathbf{c}^{i \to k}\}_{i \in [n], k \in [n]}, \{\mathbf{k}_{\mathsf{mac}}^{i \to k}\}_{i \in [n], k \in [n]}$ to \mathcal{S}_{Π} .
- Continue running S_{Π} against the adversary $\mathcal{M}_4(\mathcal{A})$.
- Upon receiving the message $\widehat{\mathsf{msg}}_{4,j'}$ from $P_{j'}$, retrieve $\mathsf{k}_{\mathsf{enc}}^{j' \to i', \phi^{i'}}$ and decrypt $(K^* || \gamma^*) \leftarrow \mathsf{dec}(\mathsf{k}_{\mathsf{enc}}^{j' \to i', \phi^{i'}})$ $c_{\phi^{i'}}^{j' \to i'}).$ - Output (K^*, γ^*)

We can see that the pair (K^*, γ^*) that the adversary \mathcal{A}_{MAC} computes is exactly the same as the one that the honest party $P_{i'}$ would decrypt. Since we assume that the honest party would not abort, that means that (K^*,γ^*) is a valid pair of the share of the label and the tag - meaning that the adversary \mathcal{A}_{MAC} would win in the Mac - $forge_{\mathcal{A},\Pi}^{1-time}$ experiment with non-negligible probability.

As before, in order for the reduction to remain polynomial time, we need to bound the running time of the simulator S_{Π} . Since the simulator S_{Π} is expected to run in polynomial time, we limit its execution to t steps, where t represents the expected polynomial running time of S_{II} . Therefore, with non-negligible probability, the simulator will stop.

The final part of the proof is to show why the protocol $\hat{\Pi}$ preserves the same type security as Π . We note that it is straightforward to see why a corrupt party cannot deviate from the protocol in a way that would break the type security of the input protocol (do more "damage").

More formally, the final step is to argue that $\hat{\Pi}$ held the same type security as Π . In $\hat{\Pi}$, the adversary (assume it controls P_i) can deviate from protocol description by using several approaches:

- 1. Deviate from the protocol Π . However, by security of Π , this type of deviation behavior will be detected, and type security is guaranteed in this case.
- 2. Send inconsistent $\widehat{\mathsf{msg}}_{3,j}$ or $\widehat{\mathsf{msg}}_{4,j}$ to different parties. We note that \hat{H} has broadcast channel access for all four rounds, P_j can not send inconsistent messages to different honest parties.
- 3. Do not send any third-round or fourth-round messages. Due to the broadcast channel access, all the honest parties can see it and identify P_i as corrupted. In other words, if the adversary only have this adversarial behavior, we can achieve id-abort security in this case. However, because of the type security of Π , the best security we can achieve is type security.
- 4. Send inconsistent third round from fourth round, and it contradicts to Lemma 5. We emphasize that change both $\widehat{\mathsf{msg}}_{3,j}$ and $\widehat{\mathsf{msg}}_{4,j}$ that use $1 - \phi^j$ is a valid behavior, and it is the same as that the adversary use the input $1 - x^j$.
- 5. Send invalid third-round or fourth-round messages. In this circumstance, by the correctness of MAC, the honest party can detect this misbehavior and identify P_i correctly. Also, because every party does the same thing in the output computation phase if one honest party can identify it correctly, all the honest parties must be able to do it, which means we achieve id-abort security if the adversary only has this adversarial behavior. However, because of the type security of Π , the best security we can achieve is type security.

Acknowledgement. We thank the reviewer of ACNS for suggesting making our new definitions more general, and overall improving the quality of our paper. We also acknowledge that this research is partially supported by the Input Output Research Hub (IORH) of the University of Edinburgh.

References

ACJ17. Prabhanjan Ananth, Arka Rai Choudhuri, and Abhishek Jain. A new approach to round-optimal secure multiparty computation. In Jonathan Katz and Hovav Shacham, editors, Advances in Cryptology CRYPTO 2017, Part I, volume 10401 of Lecture Notes in Computer Science, pages 468-499, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland.

- Bea95. Donald Beaver. Precomputing oblivious transfer. In Don Coppersmith, editor, Advances in Cryptology
 CRYPTO'95, volume 963 of Lecture Notes in Computer Science, pages 97–109, Santa Barbara, CA, USA, August 27–31, 1995. Springer Berlin Heidelberg, Germany.
- Bea96. Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In 28th Annual ACM Symposium on Theory of Computing, pages 479–488, Philadephia, PA, USA, May 22–24, 1996. ACM Press.
- BGI19. Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. In Dennis Hofheinz and Alon Rosen, editors, TCC 2019: 17th Theory of Cryptography Conference, Part I, volume 11891 of Lecture Notes in Computer Science, pages 341–371, Nuremberg, Germany, December 1–5, 2019. Springer, Cham, Switzerland.
- BGIN22. Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Secure multiparty computation with sublinear preprocessing. In Orr Dunkelman and Stefan Dziembowski, editors, Advances in Cryptology – EURO-CRYPT 2022, Part I, volume 13275 of Lecture Notes in Computer Science, pages 427–457, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.
- BGJ⁺18. Saikrishna Badrinarayanan, Vipul Goyal, Abhishek Jain, Yael Tauman Kalai, Dakshita Khurana, and Amit Sahai. Promise zero knowledge and its applications to round optimal MPC. In Hovav Shacham and Alexandra Boldyreva, editors, Advances in Cryptology – CRYPTO 2018, Part II, volume 10992 of Lecture Notes in Computer Science, pages 459–487, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.
- BHP17. Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In Yael Kalai and Leonid Reyzin, editors, TCC 2017: 15th Theory of Cryptography Conference, Part I, volume 10677 of Lecture Notes in Computer Science, pages 645–677, Baltimore, MD, USA, November 12–15, 2017. Springer, Cham, Switzerland.
- BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In 22nd Annual ACM Symposium on Theory of Computing, pages 503–513, Baltimore, MD, USA, May 14–16, 1990. ACM Press.
- Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In 42nd Annual Symposium on Foundations of Computer Science, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.
- CCG⁺20. Arka Rai Choudhuri, Michele Ciampi, Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Round optimal secure multiparty computation from minimal assumptions. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part II*, volume 12551 of *Lecture Notes in Computer Science*, pages 291–319, Durham, NC, USA, November 16–19, 2020. Springer, Cham, Switzerland.
- CCL15. Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, Advances in Cryptology – CRYPTO 2015, Part II, volume 9216 of Lecture Notes in Computer Science, pages 3–22, Santa Barbara, CA, USA, August 16–20, 2015. Springer Berlin Heidelberg, Germany.
- CGZ20. Ran Cohen, Juan A. Garay, and Vassilis Zikas. Broadcast-optimal two-round MPC. In Anne Canteaut and Yuval Ishai, editors, Advances in Cryptology – EUROCRYPT 2020, Part II, volume 12106 of Lecture Notes in Computer Science, pages 828–858, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.
- CKL03. Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable twoparty computation without set-up assumptions. In Eli Biham, editor, Advances in Cryptology – EU-ROCRYPT 2003, volume 2656 of Lecture Notes in Computer Science, pages 68–86, Warsaw, Poland, May 4–8, 2003. Springer Berlin Heidelberg, Germany.
- COSV17a. Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Delayed-input non-malleable zero knowledge and multi-party coin tossing in four rounds. In Yael Kalai and Leonid Reyzin, editors, TCC 2017: 15th Theory of Cryptography Conference, Part I, volume 10677 of Lecture Notes in Computer Science, pages 711–742, Baltimore, MD, USA, November 12–15, 2017. Springer, Cham, Switzerland.
- COSV17b. Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Round-optimal secure two-party computation from trapdoor permutations. In Yael Kalai and Leonid Reyzin, editors, TCC 2017: 15th Theory of Cryptography Conference, Part I, volume 10677 of Lecture Notes in Computer Science, pages 678–710, Baltimore, MD, USA, November 12–15, 2017. Springer, Cham, Switzerland.
- COSW23. Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Hendrik Waldner. Round-optimal black-box multiparty computation from polynomial-time assumptions. Cryptology ePrint Archive, Report 2023/1742, 2023.
- CRSW22. Michele Ciampi, Divya Ravi, Luisa Siniscalchi, and Hendrik Waldner. Round-optimal multi-party computation with identifiable abort. In Orr Dunkelman and Stefan Dziembowski, editors, Advances in Cryp-

tology – EUROCRYPT 2022, Part I, volume 13275 of Lecture Notes in Computer Science, pages 335–364, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.

- DILO22. Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Authenticated garbling from simple correlations. In Yevgeniy Dodis and Thomas Shrimpton, editors, Advances in Cryptology CRYPTO 2022, Part IV, volume 13510 of Lecture Notes in Computer Science, pages 57–87, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.
- DRSY23. Ivan Damgård, Divya Ravi, Luisa Siniscalchi, and Sophia Yakoubov. Minimizing setup in broadcastoptimal two round MPC. In Carmit Hazay and Martijn Stam, editors, Advances in Cryptology – EU-ROCRYPT 2023, Part II, volume 14005 of Lecture Notes in Computer Science, pages 129–158, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.
- GB08. S. Goldwasser and M. Bellare. Lecture notes on cryptography. Summer Course "Cryptography and Computer Security" at MIT. 2008.
- GJO⁺13. Vipul Goyal, Abhishek Jain, Rafail Ostrovsky, Silas Richelson, and Ivan Visconti. Concurrent zero knowledge in the bounded player model. In Amit Sahai, editor, *TCC 2013: 10th Theory of Cryptography Conference*, volume 7785 of *Lecture Notes in Computer Science*, pages 60–79, Tokyo, Japan, March 3–6, 2013. Springer Berlin Heidelberg, Germany.
- GMPP16. Sanjam Garg, Pratyay Mukherjee, Omkant Pandey, and Antigoni Polychroniadou. The exact round complexity of secure computation. In Marc Fischlin and Jean-Sébastien Coron, editors, Advances in Cryptology – EUROCRYPT 2016, Part II, volume 9666 of Lecture Notes in Computer Science, pages 448–476, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany.
- GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, 19th Annual ACM Symposium on Theory of Computing, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.
- Goy11. Vipul Goyal. Constant round non-malleable protocols using one way functions. In Lance Fortnow and Salil P. Vadhan, editors, 43rd Annual ACM Symposium on Theory of Computing, pages 695–704, San Jose, CA, USA, June 6–8, 2011. ACM Press.
- HHPV18. Shai Halevi, Carmit Hazay, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Round-optimal secure multi-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, Advances in Cryptology – CRYPTO 2018, Part II, volume 10992 of Lecture Notes in Computer Science, pages 488–520, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.
- HL10. Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols*. Springer Berlin Heidelberg, 2010.
- HOSS18. Carmit Hazay, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Concretely efficient largescale MPC with active security (or, TinyKeys for TinyOT). In Thomas Peyrin and Steven Galbraith, editors, Advances in Cryptology – ASIACRYPT 2018, Part III, volume 11274 of Lecture Notes in Computer Science, pages 86–117, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Cham, Switzerland.
- HSS17. Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In Tsuyoshi Takagi and Thomas Peyrin, editors, Advances in Cryptology – ASIACRYPT 2017, Part I, volume 10624 of Lecture Notes in Computer Science, pages 598–628, Hong Kong, China, December 3–7, 2017. Springer, Cham, Switzerland.
- IKSS23. Yuval Ishai, Dakshita Khurana, Amit Sahai, and Akshayaram Srinivasan. Round-optimal black-box MPC in the plain model. In Helena Handschuh and Anna Lysyanskaya, editors, Advances in Cryptology - CRYPTO 2023, Part I, volume 14081 of Lecture Notes in Computer Science, pages 393–426, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.
- IPS08. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer efficiently. In David Wagner, editor, Advances in Cryptology CRYPTO 2008, volume 5157 of Lecture Notes in Computer Science, pages 572–591, Santa Barbara, CA, USA, August 17–21, 2008. Springer Berlin Heidelberg, Germany.
- Kil88. Joe Kilian. Founding cryptography on oblivious transfer. In 20th Annual ACM Symposium on Theory of Computing, pages 20–31, Chicago, IL, USA, May 2–4, 1988. ACM Press.
- KL14. J. Katz and Y. Lindell. Introduction to Modern Cryptography, Second Edition. Chapman & Hall/CRC Cryptography and Network Security Series. Taylor & Francis, 2014.
- KO04. Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings, volume 3152 of Lecture Notes in Computer Science, pages 335–354. Springer, 2004.

- KOS03. Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Round efficiency of multi-party computation with a dishonest majority. In Eli Biham, editor, Advances in Cryptology – EUROCRYPT 2003, volume 2656 of Lecture Notes in Computer Science, pages 578–595, Warsaw, Poland, May 4–8, 2003. Springer Berlin Heidelberg, Germany.
- KRRW18. Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In Hovav Shacham and Alexandra Boldyreva, editors, Advances in Cryptology – CRYPTO 2018, Part III, volume 10993 of Lecture Notes in Computer Science, pages 365–391, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.
- NNOB12. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, Advances in Cryptology – CRYPTO 2012, volume 7417 of Lecture Notes in Computer Science, pages 681–700, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany.
- NST17. Jesper Buus Nielsen, Thomas Schneider, and Roberto Trifiletti. Constant round maliciously secure 2PC with function-independent preprocessing using LEGO. In ISOC Network and Distributed System Security Symposium – NDSS 2017, San Diego, CA, USA, February 26 – March 1, 2017. The Internet Society.
- Pas04. Rafael Pass. Bounded-concurrent secure multi-party computation with a dishonest majority. In László Babai, editor, 36th Annual ACM Symposium on Theory of Computing, pages 232–241, Chicago, IL, USA, June 13–16, 2004. ACM Press.
- PW10. Rafael Pass and Hoeteck Wee. Constant-round non-malleable commitments from sub-exponential oneway functions. In Henri Gilbert, editor, Advances in Cryptology – EUROCRYPT 2010, volume 6110 of Lecture Notes in Computer Science, pages 638–655, French Riviera, May 30 – June 3, 2010. Springer Berlin Heidelberg, Germany.
- Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In 27th Annual Symposium on Foundations of Computer Science, pages 162–167, Toronto, Ontario, Canada, October 27– 29, 1986. IEEE Computer Society Press.