

# The Exact Round Complexity of Secure Computation<sup>\*</sup>

Sanjam Garg<sup>1</sup>, Pratyay Mukherjee,<sup>1</sup> Omkant Pandey<sup>2</sup>, Antigoni Polychroniadou<sup>3</sup>

<sup>1</sup> University of California, Berkeley  
{sanjamg, pratyay85}@berkeley.edu

<sup>2</sup> Stony Brook University  
omkant@cs.stonybrook.edu

<sup>3</sup> Aarhus University, Denmark  
{antigoni}@cs.au.dk

**Abstract.** We revisit the *exact* round complexity of secure computation in the multi-party and two-party settings. For the special case of two-parties *without* a simultaneous message exchange channel, this question has been extensively studied and resolved. In particular, Katz and Ostrovsky (CRYPTO '04) proved that five rounds are necessary and sufficient for securely realizing every two-party functionality where both parties receive the output. However, the exact round complexity of general multi-party computation, as well as two-party computation *with* a simultaneous message exchange channel, is not very well understood.

These questions are intimately connected to the round complexity of non-malleable commitments. Indeed, the *exact* relationship between the round complexities of non-malleable commitments and secure multi-party computation has also not been explored.

In this work, we revisit these questions and obtain several new results. First, we establish the following main results. Suppose that there exists a  $k$ -round non-malleable commitment scheme, and let  $k' = \max(4, k + 1)$ ; then,

- **(Two-party setting with simultaneous message transmission):** there exists a  $k'$ -round protocol for securely realizing *every* two-party functionality;
- **(Multi-party setting):** there exists a  $k'$ -round protocol for securely realizing the *multi-party coin-flipping* functionality.

As a corollary of the above results, by instantiating them with existing non-malleable commitment protocols (from the literature), we establish that **four** rounds are both necessary and sufficient for both the results above. Furthermore, we establish that, for *every multi-party functionality* five rounds are sufficient. We actually obtain a variety of results offering trade-offs between rounds and the cryptographic assumptions used, depending upon the particular instantiations of underlying protocols.

## 1 Introduction

The round complexity of secure computation is a fundamental question in the area of secure computation [Yao82, Yao86, GMW87]. In the past few years, we have seen tremendous progress on this question, culminating into constant round protocols for securely computing any multi-party functionality [BMR90, KOS03, Pas04, DI05, DI06, PPV08, Wee10, Goy11, LP11a, GLOV12]. These works essentially settle the question of *asymptotic* round complexity of this problem.

---

<sup>\*</sup> Research supported in part from a DARPA/ARL SAFEWARE award, AFOSR Award FA9550-15-1-0274, and NSF CRII Award 1464397. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government. Also, Antigoni Polychroniadou received funding from CTIC under the grant 61061130540 and from CFEM supported by the Danish Strategic Research Council. This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #CNS-1523467.

The *exact* round complexity of secure computation, however, is still not very well understood <sup>1</sup>. For the special case of two-party computation, Katz and Ostrovsky [KO04] proved that 5 rounds are necessary and sufficient. In particular, they proved that two-party coin-flipping cannot be achieved in 4 rounds, and presented a 5-round protocol for computing every functionality. To the best of our knowledge, the exact round complexity of multi-party computation has never been addressed before.

The standard model for multi-party computation assumes that parties are connected via authenticated point-to-point channels as well as *simultaneous message exchange* channels where everyone can send messages at the same time. Therefore, in each round, all parties can simultaneously exchange messages.

This is in sharp contrast to the “standard” model for two-party computation where, usually, a simultaneous message exchange framework is not considered. Due to this difference in the communication model, the negative result of Katz-Ostrovsky [KO04] for 4 rounds, does not apply to the multi-party setting. In particular, a 4 round multi-party coin-flipping protocol might still exist!

In other words, the results of Katz-Ostrovsky only hold for the special case of two parties *without* a simultaneous message exchange channel. The setting of two-party computation *with a simultaneous message exchange channel* has not been addressed before. Therefore, in this work we address the following two questions:

*What is the exact round complexity of secure multi-party computation?*

*In the presence of a simultaneous message exchange channel, what is the exact round complexity of secure two-party computation?*

These questions are intimately connected to the round complexity of *non-malleable commitments* [DDN91]. Indeed, new results for non-malleable commitments have almost immediately translated to new results for secure computation. For example, the round complexity of coin-flipping was improved by Barak [Bar02], and of every multi-party functionality by Katz, Ostrovsky, and Smith [KOS03] based on techniques from non-malleable commitments. Pass [Pas04] constructed the first *constant* round bounded-concurrent protocol based on polynomial-time assumptions, since the constant round protocol of [KOS03] relied on sub-exponential hardness assumptions. Likewise, black-box constructions for constant-round non-malleable commitments resulted in constant-round black-box constructions for secure computation [Wee10,Goy11]. However, all of these results only focus on *asymptotic* improvements and do not try to resolve the exact round complexity, thereby leaving the following fundamental question unresolved:

*What is the relationship between the exact round complexities of non-malleable commitments and secure computation?*

This question is at the heart of understanding the exact round complexity of secure computation in both multi-party, and two-party with simultaneous message transmission.

## 1.1 Our Contributions

In this work we try to resolve the questions mentioned above. We start by focusing on the simpler case of two-party computation with a simultaneous message exchange channel, since it is a direct special case of the multi-party setting. We then translate our results to the multi-party setting.

**Lower bounds for coin-flipping.** We start by focusing on the following question.

*How many simultaneous message exchange rounds are necessary for secure two-party computation?*

We show that four *simultaneous message exchange rounds* are necessary. More specifically, we show that:

---

<sup>1</sup> Our rough estimate for the exact round complexity of aforementioned multi-party results in the computational setting is 20-30 rounds depending upon the underlying components and assumptions.

**Theorem (Informal):** *Let  $\kappa$  be the security parameter. Even in the simultaneous message model, there does not exist a three-round protocol for the two-party coin-flipping functionality for  $\omega(\log \kappa)$  coins which can be proven secure via black-box simulation.*

In fact, as a corollary all of the rounds must be “strictly simultaneous message transmissions”, that is, both parties must *simultaneously* send messages in each of the 4 rounds. This is because in the simultaneous message exchange setting, the security is proven against the so called “rushing adversaries” who, in each round, can decide their message after seeing the messages of all honest parties in that round. Consequently, if only one party sends a message for example in the fourth round, this message can be “absorbed” within the third message of this party<sup>2</sup>, resulting in a three round protocol.

**Results in the two-party setting with a *simultaneous message exchange channel*.** Next, we consider the task of constructing a protocol for coin-flipping (or any general functionality) in four simultaneous message exchange rounds and obtain a positive result. In fact, we obtain our results by directly exploring the *exact relationship* between the round complexities of non-malleable commitments and secure computation. Specifically, we first prove the following result:

**Theorem (Informal):** *If there exists a  $k$ -round protocol for (parallel & 3-robust) non-malleable commitment,<sup>3</sup> then there exists a  $k'$ -round protocol for securely computing every two-party functionality with black-box simulation in the presence of a malicious adversary in the simultaneous message model, where  $k' = \max(4, k + 1)$ .*

Instantiating this protocol with non-malleable commitments from [PPV08], we get a **four round** protocol for every two-party functionality in the presence of a simultaneous message exchange channel, albeit under a non-standard assumption (adaptive one-way function). However, a recent result by Goyal et al. [GPR15] constructs a non-malleable commitment protocol in *three* rounds from one-way function, although their protocol is not robust and does not immediately extend to a parallel setting. Instantiating our protocol with an extension of [GPR15] presented in [COSV16] based on one-way permutations secure w.r.t. sub-exponential time adversaries yields a *four round* two-party protocol.

**Results in the multi-party setting.** Next, we focus on the case of the multi-party *coin flipping* functionality. We show that a simpler version of our two-party protocol gives a result for multi-party *coin-flipping*:

**Theorem (Informal):** *If there exists a  $k$ -round protocol for (parallel, input-delayed & 3-robust) non-malleable commitments, then there exists a  $k'$ -round protocol for securely computing the multi-party coin-flipping functionality with black-box simulation in the presence of a malicious adversary for polynomially many coins where  $k' = \max(4, k + 1)$ .*

Combining this result with the two-round multi-party protocol of Mukherjee and Wichs [MW15] (based on the LWE [Reg05]), we obtain a  $k' + 2$  round protocol for computing *every multi-party* functionality. Instantiating these protocols with non-malleable commitments from [PPV08], or an extension of [COSV16] to the parallel setting, we obtain a **four round** protocol for *coin-flipping* and a **six round** protocol for *every* functionality.

Finally, we show that the coin-flipping protocol for the multi-party setting can be extended to compute what we call the “coin-flipping with committed inputs” functionality. Using this protocol with the two-round

<sup>2</sup> Note that, such absorption is only possible when it maintains the mutual dependency among the messages, in particular does not affect the next-message functions.

<sup>3</sup> Parallel simply means that the man-in-the-middle receives  $\kappa$  non-malleable commitments in *parallel* from the left interaction and makes  $\kappa$  commitments on the right. Almost all known non-malleable commitment protocols satisfy this property. We describe the notion of a robust non-malleable commitment in Section 2.1; traditional definitions of non-malleability consider a setting where a man-in-the middle adversary is participating in two (or more) executions of the *same* protocol. However, 3-robust non-malleable commitments w.r.t. arbitrary protocols considers a class of adversaries that can participate in a left interaction of any 3-round *arbitrary* protocol; a formal definitions can be found in Section 2.1.

protocol of [GGHR14a] based on indistinguishability obfuscation [GGH<sup>+</sup>13], we obtain a **five round** MPC protocol.

## 1.2 Related Work

The round complexity of secure computation has a rich and long history. We only mention the results that are most relevant to this work in the computational setting. Note that, unconditionally secure protocols such as [BGW88, CCD88] are inherently non-constant round. More specifically, the impossibility result of [DNPR16] implies that a fundamental new approach must be found in order to construct protocols, that are efficient in the circuit size of the evaluated function, with reduced communication complexity that beat the complexities of BGW, CCD, GMW etc.

For the computational setting and the special case of two party computation, the semi-honest secure protocol of Yao [Yao82, Yao86, LP11b] consists of only three rounds (see Section 2). For malicious security<sup>4</sup>, a constant round protocol based on GMW was presented by Lindell [Lin01]. Ishai, Prabhakaran, and Sahai [IPS08] presented a different approach which also results in a constant round protocol.

The problem of exact round complexity of two party computation was studied in the beautiful work of Katz and Ostrovsky [KO04] who provided a 5 round protocol for computing any two-party functionality. They also ruled out the possibility of a four round protocol for coin-flipping, thus completely resolving the case of two party (albeit without simultaneous message exchange, as discussed earlier). Recently Ostrovsky, Richelson and Scafuro [ORS15] constructed a different 5-round protocol for the general two-party computation by only relying on *black-box* usage of the underlying trapdoor one-way permutation.

As discussed earlier, the standard setting for two-party computation does not consider simultaneous message exchange channels, and hence the negative results for the two-party setting do not apply to the multi-party setting where simultaneous message exchange channels are standard. To the best of our knowledge, prior to our work, the case of the two-party setting in the presence of a simultaneous message exchange channel was not explored in the context of the exact round complexity of secure computation.

For the multi-party setting, the exact round complexity has remained open for a long time. The work of [BMR90] gave the first constant-round non black-box protocol for honest majority (improved by the black-box protocols of [DI05, DI06]). Katz, Ostrovsky, and Smith [KOS03], adapted techniques from [DDN91, Bar02, BMR90, CLOS02] to construct the first asymptotically round-optimal protocols for any multi-party functionality for the dishonest majority case based on subexponential hardness assumptions ([KOS03] achieve logarithmic round-complexity based on polynomial-time assumptions). In particular, apart from enhanced trapdoor permutations their protocol relies on the assumption of sub-exponentially secure dense cryptosystems as well as sub-exponentially secure collision resistant hash functions. Pass [Pas04] constructed a constant-round bounded-concurrent protocol based on standard polynomial-time assumptions, i.e. enhanced trapdoor permutations and collision resistant hash functions. The constant-round protocols of [KOS03, Pas04] relied on non-black-box use of the adversary’s algorithm [Bar01]. However, constant-round protocols making black-box use of the adversary were constructed by [PPV08, LP11a, Goy11], and making black-box use of one-way functions by Wee in  $\omega(1)$  rounds [Wee10] and by Goyal in constant rounds [Goy11]. Furthermore, based on the non-malleable commitment scheme of [Goy11], [GLOV12] construct a constant-round multi-party coin-tossing protocol. Lin, Pass, and Venkatasubramanian [LPV09] presented a unified approach to construct UC-secure protocols from non-malleable commitments. However, as mentioned earlier, none of the aforementioned works focused on the *exact* round complexity of secure computation based on the round-complexity of non-malleable commitments. For a detailed survey of round complexity of secure computation in the preprocessing model or in the CRS model we refer to [Pol17].

## 1.3 An Overview of Our Approach

We now provide an overview of our approach. As discussed earlier, we first focus on the two-party setting with a simultaneous message exchange channel.

---

<sup>4</sup> From here on, unless specified otherwise, we are always in the malicious setting by default.

The starting point of our construction is the Katz-Ostrovsky (KO) protocol [KO04] which is a *four round* protocol for *one-sided* functionalities, i.e., in that only one party gets the output. Recall that, this protocol does not assume the presence of a simultaneous message exchange channel. At the cost of an extra round, the KO two-party protocol can be converted to a *complete* (i.e. both-sided) protocol where both parties get their corresponding outputs via a standard trick [Gol03] as follows: parties compute a modified functionality in which the first party  $P_1$  learns its output as well as the output of the second party  $P_2$  in an “encrypted and authenticated”<sup>5</sup> form. It then sends the encrypted value to  $P_2$  who can decrypt and verify its output.

A natural first attempt is to adapt this simple and elegant approach to the setting of simultaneous message exchange channel, so that the “encrypted/authenticated output” can somehow be communicated to  $P_2$  simultaneously at the same time when  $P_2$  sends its last message, thereby removing the additional round.

It is not hard to see that any such approach would not work. Indeed, in the presence of malicious adversaries while dealing with a simultaneous message exchange channel, the protocol must be proven secure against “rushing adversaries” who can send their messages after looking at the messages sent by the other party. This implies that, if  $P_1$  could indeed send the “encrypted/authenticated output” message simultaneously with last message from  $P_2$ , it could have sent it earlier as well. Now, applying this argument repeatedly, one can conclude that any protocol which does not use the simultaneous message exchange channel necessarily in all of the four rounds, is bound to fail (see Section 3). In particular, any such protocol can be transformed, by simple rescheduling, into a 3-round protocol contradicting our lower bound<sup>6</sup>.

This means that we must think of an approach which must use the simultaneous message exchange channel in each round. In light of this, a natural second attempt is to run two executions of a 4-round protocol (in which only one party learns the output) in “opposite” directions. This would allow both parties to learn the output. Unfortunately, such approaches do not work in general since there is no guarantee that an adversarial party would use the same input in both protocol executions. Furthermore, another problem with this approach is that of “non-malleability” where a cheating party can make its input dependent on the honest party’s input: for example, it can simply “replay” back the messages it receives. A natural approach to prevent such attacks is to deploy non-malleable commitments, as we discuss below.

**Simultaneous executions + non-malleable commitments.** Following the approach discussed above we observe that:

1. A natural direction is to use two simultaneous executions of the KO protocol (or any other similar 4-round protocol) over the simultaneous message exchange channel in opposite directions. Since we have only 4 rounds, a different protocol (such as some form of 2-round semi-honest protocol based on Yao) is not a choice.
2. We must use non-malleable commitments to prevent replay/mauling attacks.

We remark that, the fact that non-malleable commitments come up as a natural tool is not a coincidence. As noted earlier, the multi-party case is well known to be *inherently connected* to non-malleable commitments. Even though our current focus is solely on the two-party case, this setting is essentially (a special case of) the multi-party setting due to the use of the simultaneous message exchange channel. Prior to our work, non-malleable commitments have been used extensively to design multi-party protocols [Goy11,LP11b,LPTV10,GLOV12]. However, all of these works result in rather poor round complexity because of their focus on asymptotic, as opposed to exact, number of rounds.

To obtain our protocol, we put the above two ideas together, modifying several components of KO<sup>7</sup> to use non-malleable commitments. These components are then put together in a way such that, even though there

<sup>5</sup> In particular, the encryption prevents  $P_1$  to know  $P_2$ ’s output ensuring output privacy whereas the authentication does not allow  $P_1$  to send  $P_2$  a wrong output.

<sup>6</sup> Recall that we show that (see Thoerem 2 for a formal statement) 4 rounds are necessary even with simultaneous message exchange channels.

<sup>7</sup> The KO protocol uses a clever combination of garble circuits, semi-honest oblivious transfer, coin-tossing, and WIPOK to ensure that the protocol is executed with a fixed input (allowing at the same time simulation extractability of the input), and relies on the zero-knowledge property of a modified Feige-Shamir proof to achieve output simulation.

are essentially two simultaneous executions of the protocol in opposite directions, messages of one protocol cannot be maliciously used to affect the other messages. In the following, we highlight the main ideas of our construction:

1. The first change we make is to the proof systems used by KO. Recall that KO uses the Fiege-Shamir (FS) protocol as a mechanism to “force the output” in the simulation. Our first crucial modification is to consider a variant of the FS protocol in which the verifier gives two non-malleable commitments (**nmcom**) to two strings  $\sigma_1, \sigma_2$  and gives a *witness indistinguishable proof-of-knowledge* (WIPOK) that it knows one of them. These are essentially the simulation trapdoors, but implemented through **nmcom** instead of a one-way function. This change is actually crucial, and as such, brings in an effect similar to “simulation sound” zero-knowledge.
2. The oblivious transfer protocol based on trapdoor permutations and coin-tossing now performs coin-tossing with the help of **nmcom** instead of simple commitments. This is a crucial change since this allows us to slowly get rid of the honest party’s input in the simulation and still argue that the distribution of the adversary’s input does not change as a result of this.  
We note that there are many parallel executions on **nmcom** that take place at this stage, and therefore, we require that **nmcom** should be non-malleable under many parallel executions. This is indeed true for most **nmcom**.
3. Finally, we introduce a mechanism to ensure that the two parties use the exact same input in both executions. Roughly speaking, this is done by requiring the parties to prove consistency of messages “across” protocols.
4. To keep the number of rounds to  $k + 1$  (or 4 if  $k < 3$ ), many of the messages discussed above are “absorbed” with other rounds by running in parallel.

**Multi-party setting.** The above protocol does not directly extend to the multi-party settings. Nevertheless, for the special case of *coin flipping*, we show that a (simplified) version of the above protocol works for the multi-party case. This is because the coin-tossing functionality does not really require any computation, and therefore, we can get rid of components such as oblivious transfer. In fact, this can be extended “slightly more” to also realize the “coin-flipping with committed inputs” since committing the input does not depend on inputs of other parties.

Next, to obtain our result for general functionalities, we simply invoke known results: using [MW15] with coin-flipping gives us a six round protocol, and using [GGHR14b] gives a five round result.

## 2 Preliminaries

**Notation.** We denote the security parameter by  $\kappa$ . We say that a function  $\mu : \mathbb{N} \rightarrow \mathbb{N}$  is *negligible* if for every positive polynomial  $p(\cdot)$  and all sufficiently large  $\kappa$ ’s it holds that  $\mu(\kappa) < \frac{1}{p(\kappa)}$ . We use the abbreviation PPT to denote probabilistic polynomial-time. We often use  $[n]$  to denote the set  $\{1, \dots, n\}$ . Moreover, we use  $d \leftarrow \mathcal{D}$  to denote the process of sampling  $d$  from the distribution  $\mathcal{D}$  or, if  $\mathcal{D}$  is a set, a uniform choice from it. If  $\mathcal{D}_1$  and  $\mathcal{D}_2$  are two distributions, then we denote that they are statistically close by  $\mathcal{D}_1 \approx_s \mathcal{D}_2$ ; we denote that they are computationally indistinguishable by  $\mathcal{D}_1 \approx_c \mathcal{D}_2$ ; and we denote that they are identical by  $\mathcal{D}_1 \equiv \mathcal{D}_2$ . Let  $V$  be a random variable corresponding to the distribution  $\mathcal{D}$ . Sometimes we abuse notation by using  $V$  to denote the corresponding distribution  $\mathcal{D}$ .

We assume familiarity with several standard cryptographic primitives. For notational purposes, we recall here the basic working definitions for some of them. We skip the well-known formal definitions for secure two-party and multi-party computations (see Appendix A for a formal description). It will be sufficient to have notation for the two-party setting. We denote a two party functionality by  $F : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^* \times \{0, 1\}^*$  where  $F = (F_1, F_2)$ . For every pair of inputs  $(x, y)$ , the output-pair is a random variable  $(F_1(x, y), F_2(x, y))$  ranging over pairs of strings. The first party (with input  $x$ ) should obtain  $F_1(x, y)$  and the second party (with input  $y$ ) should obtain  $F_2(x, y)$ . Without loss of generality, we assume that  $F$  is deterministic. The security

is defined through the ideal/real world paradigm where for adversary  $\mathcal{A}$  participating in the real world protocol, there exists an ideal world simulator  $\mathcal{S}$  such that for every  $(x, y)$ , the output of  $\mathcal{S}$  is indistinguishable from that of  $\mathcal{A}$ . See Appendix A for an extended discussion.

We now recall the definitions for non-malleable commitments as well as some components from the work of Katz-Ostrovsky [KO04].

## 2.1 Tag Based Mon-Malleable Commitments

Let  $\text{nmcom} = \langle C, R \rangle$  be a  $k$ -round commitment protocol where  $C$  and  $R$  represent (randomized) committer and receiver algorithms, respectively. Denote the messages exchanged by  $(\text{nm}_1, \dots, \text{nm}_k)$  where  $\text{nm}_i$  denotes the message in the  $i$ -th round.

For some string  $u \in \{0, 1\}^\kappa$ , tag  $\text{id} \in \{0, 1\}^t$ , non-uniform PPT algorithm  $M$  with “advice” string  $z \in \{0, 1\}^*$ , and security parameter  $\kappa$ , consider the following experiment:  $M$  on input  $(1^\kappa, z)$ , interacts with  $C$  who commits to  $u$  with tag  $\text{id}$ ; simultaneously,  $M$  interacts with  $R(1^\kappa, \tilde{\text{id}})$  attempting to commit to a related value  $\tilde{u}$ , again using identity  $\tilde{\text{id}}$  of its choice ( $M$ ’s interaction with  $C$  is called the left interaction, and its interaction with  $R$  is called the right interaction);  $M$  controls the scheduling of messages; the output of the experiment is denoted by a random variable  $\text{nmc}_{\langle C, R \rangle}^M(u, z)$  that describes the view of  $M$  in both interactions and the value  $\tilde{u}$  which  $M$  commits to  $R$  in the right execution unless  $\tilde{\text{id}} = \text{id}$  in which case  $\tilde{u} = \perp$ , i.e., a commitment where the adversary copies the identity of the left interaction is considered invalid.

**Definition 1 (Tag based non-malleable commitments)** *A commitment scheme  $\text{nmcom} = \langle C, R \rangle$  is said to be non-malleable with respect to commitments if for every non-uniform PPT algorithm  $M$  (man-in-the-middle), for every pair of strings  $(u^0, u^1) \in \{0, 1\}^\kappa \times \{0, 1\}^\kappa$ , every tag-string  $\text{id} \in \{0, 1\}^t$ , every  $\kappa \in \mathbb{N}$ , every (advice) string  $z \in \{0, 1\}^*$ , the following two distributions are computationally indistinguishable:*

$$\text{nmc}_{\langle C, R \rangle}^M(u^0, z) \stackrel{c}{\approx} \text{nmc}_{\langle C, R \rangle}^M(u^1, z)$$

**Parallel Non-Malleable Commitments.** We consider a strengthening of  $\text{nmcom}$  in which  $M$  can receive commitments to  $m$  strings on the “left”, say  $(u_1, \dots, u_m)$ , with tags  $(\text{id}_1, \dots, \text{id}_m)$  and makes  $m$  commitments on the “right” with tags  $(\tilde{\text{id}}_1, \dots, \tilde{\text{id}}_m)$ . We assume that  $m$  is a fixed, possibly a-priori bounded, polynomial in the security parameter  $\kappa$ . In the following let  $i \in [m], b \in \{0, 1\}$ : We say that a  $\text{nmcom}$  is an  $m$ -bounded parallel non-malleable commitment if for every pair of sequences  $\{u_i^b\}$  the random variables  $\text{nmc}_{\langle C, R \rangle}^M(\{u_i^0\}, z)$  and  $\text{nmc}_{\langle C, R \rangle}^M(\{u_i^1\}, z)$  are computationally indistinguishable where  $\text{nmc}_{\langle C, R \rangle}^M(\{u_i^b\}, z)$  describes the view of  $M$  and the values  $\{\tilde{u}_i^b\}$  committed by  $M$  in the  $m$  sessions on the right with tags  $\{\tilde{\text{id}}_i\}$  while receiving parallel commitments to  $\{u_i^b\}$  on left with tags  $\{\text{id}_i\}$ .

**First message binding property.** It will be convenient in the notation to assume that the first message  $\text{nm}_1$  of the non-malleable commitment scheme  $\text{nmcom}$  statistically determines the message being committed. This can be relaxed to only require that the message is fixed before the last round if  $k \geq 3$  (in fact, in some components of our protocol, non-malleable commitments that *delay* their input message until the last round can also be used).

**Non-Malleable Commitment Robust w.r.t.  $k$ -round Protocols.** Lin and Pass [LP09a] introduced the notion of non-malleability w.r.t. arbitrary  $k$ -round protocols. Traditional definitions of non-malleability consider a setting where a man-in-the middle adversary is participating in two (or more) executions of the *same* protocol. However, non-malleability w.r.t. arbitrary protocols considers a class of adversaries that can participate in a left interaction of any arbitrary protocol.

Consider a one-many man-in-the-middle adversary  $M$  that participates in one left interaction – communicating with a machine  $B$  – and one right interaction – acting as a committer using the commitment scheme  $\langle C, R \rangle$ . As in the standard definition of non-malleability,  $M$  can adaptively choose the identity in

the right interaction. We denote by  $\text{nmcom}_{(C,R)}^{B,M}(y,z)$  the random variable consisting of the view of  $M(z)$  in a man-in-the-middle execution when communicating with  $B(y)$  on the left and an honest receiver on the right, combined with the value  $M$  commits to on the right. Intuitively, we say that  $\langle C, R \rangle$  is non-malleable w.r.t.  $B$  if  $\text{nmcom}_{(C,R)}^{B,M}(y^1, z)$  and  $\text{nmcom}_{(C,R)}^{B,M}(y^2, z)$  are indistinguishable, whenever interactions with  $B(y^1)$  and  $B(y^2)$  cannot be distinguished.

**Definition 2 (Robust non-malleable commitments)** *Let  $B$  a probabilistic polynomial-time machine. A commitment scheme  $\text{nmcom} = \langle C, R \rangle$  is said to be non-malleable w.r.t.  $B$ , if for every two sequences  $\{y_\kappa^1\}_{\kappa \in \mathbb{N}}$  and  $\{y_\kappa^2\}_{\kappa \in \mathbb{N}}$  where  $y_\kappa^1, y_\kappa^2 \in \{0, 1\}^\kappa$ , every  $\kappa \in \mathbb{N}$ , every (advice) string  $z \in \{0, 1\}^*$ , the following two distributions are computationally indistinguishable for all non-uniform PPT algorithms  $\tilde{M}$ :*

$$\text{view}_{\tilde{M}}[\langle B(y_\kappa^1), \tilde{M}(z) \rangle(1^\kappa)] \stackrel{c}{\approx} \text{view}_{\tilde{M}}[\langle B(y_\kappa^2), \tilde{M}(z) \rangle(1^\kappa)].$$

where  $\text{view}_{\tilde{M}}[\langle B(y), M(z) \rangle(1^\kappa)]$  denotes the view of  $\tilde{M}$  in interaction with  $B$  on common input  $1^\kappa$ , and private inputs  $z$  and  $y$  respectively, then it also holds that for every non-uniform PPT man-in-the-middle adversary  $M$  the following two distributions are computationally indistinguishable:

$$\text{nmcom}_{(C,R)}^{B,M}(y_\kappa^1, z) \stackrel{c}{\approx} \text{nmcom}_{(C,R)}^{B,M}(y_\kappa^2, z)$$

We say that  $\text{nmcom}$  is non-malleable w.r.t.  $k$ -round protocols if  $\text{nmcom}$  is non-malleable w.r.t. any machine  $B$  that interacts with the man-in-the-middle adversary in  $k$  rounds. Any commitment scheme that is "extractable" and has more than  $k$  "rewinding slots" is directly one-many non-malleable w.r.t.  $k$ -round protocols [LP09a]. In this work, we focus on non-malleability w.r.t 3-round protocols and in particular 3-round WI proof of knowledge protocols; we call such non malleable commitments *3-robust*.

## 2.2 Components of our Protocol

In this section, we recall some components from the KO protocol [KO04]. These are mostly standard and recalled here for a better exposition. The only (minor but crucial) change needed in our protocol is to the FLS proof system [FLS99, FS90, Fei90] where a (3-robust and parallel) *non-malleable* commitment protocol is used by the verifier. For concreteness, we discuss these modifications below.

**Modified Feige-Shamir proof systems.** We use two proof systems:  $\Pi_{\text{WIPOK}}$  and  $\Pi_{\text{FS}}$ . Protocol  $\Pi_{\text{WIPOK}}$  is the 3-round, public-coin, witness-indistinguishable proof-of-knowledge based on the work of Feige, Lapidot, Shamir [FLS99] for proving graph Hamiltonicity. This proof system proves statements of the form  $\text{st}_1 \wedge \text{st}_2$  where  $\text{st}_1$  is fixed at the first round of the protocol, but  $\text{st}_2$  is determined only in the last round of the protocol.<sup>8</sup> For concreteness, this proof system is given in Appendix B.

Protocol  $\Pi_{\text{FS}}$  is the 4-round zero-knowledge argument-of-knowledge protocol of Feige and Shamir [FS90], which allows the prover to prove statement  $\text{thm}$ , with the modification that the protocol from verifier's side is implemented using  $\text{nmcom}$ . More specifically,

- Recall that the Feige-Shamir protocol consists of two executions of  $\Pi_{\text{WIPOK}}$  in reverse directions. In the first execution, the verifier selects a one-way function  $f$  and sets  $x_1 = f(w_1)$ ,  $x_2 = f(w_2)$  and proves the knowledge of a witness for  $x_1 \vee x_2$ . In the second execution, prover proves the knowledge of a witness to the statement  $\text{thm} \vee (x_1 \vee x_2)$  where  $\text{thm}$  is the statement to be proven. The rounds of these systems can be somewhat parallelized to obtain a 4-round protocol.
- Our modified system, simply replaces the function  $f$  and  $x_1, x_2$  with two executions of  $\text{nmcom}$ . For convenience, suppose that  $\text{nmcom}$  has only 3 rounds. Then, our protocol creates the first message of two independent executions of  $\text{nmcom}$  to strings  $\sigma_1, \sigma_2$ , denoted by  $\text{nm}_1^{\sigma_1}, \text{nm}_1^{\sigma_2}$  respectively, and sets

<sup>8</sup> Typically,  $\text{st}_1$  is a empty statement and not usually mentioned; but KO [KO04] uses a specific, non-empty, statement and so does this work.



$x_1 = \text{nm}_1^{\sigma_1}, x_2 = \text{nm}_1^{\sigma_2}$ . The second and third messages of  $\text{nmcom}$  are sent with the second and third messages of the original FS protocol.

If  $\text{nmcom}$  has more than 3 rounds, simply complete the first  $k - 3$  rounds of the two executions before the 4 messages of the proof system above are exchanged.

- As before, although  $\Pi_{\text{FS}}$  proves statement  $\text{thm}$ , as noted in [KO04], it actually proves statements of the form  $\text{thm} \wedge \text{thm}'$  where  $\text{thm}$  can be fixed in the second round, and  $\text{thm}'$  in the fourth round. Usually  $\text{thm}$  is empty and not mentioned. Indeed, this is compatible with the second  $\Pi_{\text{WIPOK}}$  which proves statement of the form  $\text{st}_1 \wedge \text{st}_2$ , just set  $\text{st}_1 = \text{thm}, \text{st}_2 = \text{thm}'$ .

For completeness, we describe the full  $\Pi_{\text{FS}}$  protocol in Appendix B.

## Components of Katz-Ostrovsky Protocol

The remainder of this section is largely taken from [KO04] where we provide basic notations and ideas for semi-honest secure two-party computation based on Yao’s garbled circuits and semi-honest oblivious transfer (based on trapdoor one-way permutations). Readers familiar with [KO04] can skip this part without loss in readability.

**Semi-Honest Secure Two-party Computation.** We view Yao’s garbled circuit scheme [Yao82,LP09b] as a tuple of PPT algorithms ( $\text{GenGC}, \text{EvalGC}$ ), where  $\text{GenGC}$  is the “generation procedure” which generates a garbled circuit for a circuit  $C$  along with “labels,” and  $\text{EvalGC}$  is the “evaluation procedure” which evaluates the circuit on the “correct” labels. Each individual wire  $i$  of the circuit is assigned two labels, namely  $Z_{i,0}, Z_{i,1}$ . More specifically, the two algorithms have the following format (here  $i \in [\kappa], b \in \{0, 1\}$ ):

- $(\{Z_{i,b}\}, \text{GC}_y) \leftarrow \text{GenGC}(1^\kappa, F, y)$ :  $\text{GenGC}$  takes as input a security parameter  $\kappa$ , a circuit  $F$  and a string  $y \in \{0, 1\}^\kappa$ . It outputs a *garbled circuit*  $\text{GC}_y$  along with the set of all *input-wire labels*  $\{Z_{i,b}\}$ . The garbled circuit *may be viewed as representing the function*  $F(\cdot, y)$ .
- $v = \text{EvalGC}(\text{GC}_y, \{Z_{i,x_i}\})$ : Given a *garbled circuit*  $\text{GC}_y$  and a set of *input-wire labels*  $\{Z_{i,x_i}\}$  where  $x \in \{0, 1\}^\kappa$ ,  $\text{EvalGC}$  outputs either an *invalid symbol*  $\perp$ , or a *value*  $v = F(x, y)$ .

The following properties are required:

**Correctness.**  $\Pr[F(x, y) = \text{EvalGC}(\text{GC}_y, \{Z_{i,x_i}\})] = 1$  for all  $F, x, y$ , taken over the correct generation of  $\text{GC}_y, \{Z_{i,b}\}$  by  $\text{GenGC}$ .

**Security.** There exists a PPT simulator  $\text{SimGC}$  such that for any  $(F, x)$  and uniformly random labels  $\{Z_{i,b}\}$ , we have that:

$$(\text{GC}_y, \{Z_{i,x_i}\}) \stackrel{c}{\approx} \text{SimGC}(1^\kappa, F, v)$$

where  $(\{Z_{i,b}\}, \text{GC}_y) \leftarrow \text{GenGC}(1^\kappa, F, y)$  and  $v = F(x, y)$ .

In the semi-honest setting, two parties can compute a function  $F$  of their inputs, in which only *one* party, say  $P_1$ , learns the output, as follows. Let  $x, y$  be the inputs of  $P_1, P_2$ , respectively. First,  $P_2$  computes  $(\{Z_{i,b}\}, \text{GC}_y) \leftarrow \text{GenGC}(1^\kappa, F, y)$  and sends  $\text{GC}_y$  to  $P_1$ . Then, the two parties engage in  $\kappa$  parallel instances of OT. In particular, in the  $i$ -th instance,  $P_1$  inputs  $x_i$ ,  $P_2$  inputs  $(Z_{i,0}, Z_{i,1})$  to the OT protocol, and  $P_1$  learns the “output”  $Z_{i,x_i}$ . Then,  $P_1$  computes  $v = \text{EvalGC}(\text{GC}_y, \{Z_{i,x_i}\})$  and outputs  $v = F(x, y)$ .

A 3-round, semi-honest, OT protocol can be constructed from enhanced trapdoor permutations (TDP). For notational purposes, define TDP as follows:

**Definition 3 (Trapdoor permutations)** Let  $\mathcal{F}$  be a triple of PPT algorithms  $(\text{Gen}, \text{Eval}, \text{Invert})$  such that if  $\text{Gen}(1^\kappa)$  outputs a pair  $(f, \text{td})$ , then  $\text{Eval}(f, \cdot)$  is a permutation over  $\{0, 1\}^\kappa$  and  $\text{Invert}(f, \text{td}, \cdot)$  is its inverse.  $\mathcal{F}$  is a trapdoor permutation such that for all PPT adversaries  $A$ :

$$\Pr[(f, \text{td}) \leftarrow \text{Gen}(1^\kappa); y \leftarrow \{0, 1\}^\kappa; x \leftarrow A(f, y) : \text{Eval}(f, x) = y] \leq \mu(\kappa).$$

For convenience, we drop  $(f, \text{td})$  from the notation, and write  $f(\cdot), f^{-1}(\cdot)$  to denote algorithms  $\text{Eval}(f, \cdot), \text{Invert}(f, \text{td}, \cdot)$  respectively, when  $f, \text{td}$  are clear from the context. We assume that  $\mathcal{F}$  satisfies (a weak variant of) “certifiability”: namely, given some  $f$  it is possible to decide in polynomial time whether  $\text{Eval}(f, \cdot)$  is a permutation over  $\{0, 1\}^\kappa$ .

Let  $\mathsf{H}$  be the *hardcore bit function for  $\kappa$  bits* for the family  $\mathcal{F}$ ;  $\kappa$  hardcore bits are obtained from a single-bit hardcore function  $h$  and  $f \in \mathcal{F}$  as follows:  $\mathsf{H}(z) = h(z) \| h(f(z)) \| \dots \| h(f^{\kappa-1}(z))$ . Informally,  $\mathsf{H}(z)$  looks pseudorandom given  $f^\kappa(z)$ .

The semi-honest OT protocol based on TDP is constructed as follows. Let  $P_2$  hold two strings  $Z_0, Z_1 \in \{0, 1\}^\kappa$  and  $P_1$  hold a bit  $b$ . In the first round,  $P_2$  chooses trapdoor permutation  $(f, f^{-1}) \leftarrow \text{Gen}(1^\kappa)$  and sends  $f$  to  $P_1$ . Then  $P_1$  chooses two random string  $z'_0, z'_1 \leftarrow \{0, 1\}^\kappa$ , computes  $z_b = f^\kappa(z'_b)$  and  $z_{1-b} = z'_{1-b}$  and sends  $(z_0, z_1)$  to  $P_2$ . In the last round  $P_2$  computes  $W_a = Z_a \oplus \mathsf{H}(f^{-\kappa}(z_a))$  where  $a \in \{0, 1\}$ ,  $\mathsf{H}$  is the hardcore bit function and sends  $(W_0, W_1)$  to  $P_1$ . Finally,  $P_2$  can recover  $Z_b$  by computing  $Z_b = W_b \oplus \mathsf{H}(z_b)$ .

Putting it altogether, we obtain the following 3-round, semi-honest secure two-party protocol for the single-output functionality  $F$  (here only  $P_1$  receives the output):

**Protocol  $\Pi_{\text{SH}}$ .**  $P_1$  holds input  $x \in \{0, 1\}^\kappa$  and  $P_2$  holds inputs  $y \in \{0, 1\}^\kappa$ . Let  $\mathcal{F}$  be a family of trapdoor permutations and let  $\mathsf{H}$  be a hardcore bit function. For all  $i \in [\kappa]$  and  $b \in \{0, 1\}$  the following steps are executed:

**Round-1** :  $P_2$  computes  $(\{Z_{i,b}\}, \text{GC}_y) \leftarrow \text{GenGC}(1^\kappa, F, y)$  and chooses trapdoor permutation  $(f_{i,b}, f_{i,b}^{-1}) \leftarrow \text{Gen}(1^\kappa)$  and sends  $(\text{GC}_y, \{f_{i,b}\})$  to  $P_2$ .

**Round-2** :  $P_1$  chooses random strings  $\{z'_{i,b}\}$ , computes  $z_{i,b} = f^\kappa(z'_{i,b})$  and  $z_{i,1-b} = z'_{i,1-b}$  and sends  $\{z_{i,b}\}$  to  $P_2$ .

**Round-3** :  $P_2$  computes  $W_{i,b} = Z_{i,b} \oplus \mathsf{H}(f_{i,b}^{-\kappa}(z_{i,b}))$  and sends  $\{W_{i,b}\}$  to  $P_2$ .

**Output** :  $P_1$  recovers the labels  $Z_{i,x_i} = W_{i,x_i} \oplus \mathsf{H}(z_{i,x_i})$  and computes  $v = \text{EvalGC}(\text{GC}_y, \{Z_{i,x_i}\})$  where  $v = F(x, y)$

**Equivocal Commitment scheme Eqcom.** We assume familiarity with equivocal commitments, and use the following equivocal commitment scheme Eqcom based on any (standard) non-interactive, perfectly binding, commitment scheme com: to commit to a bit  $x$ , the sender chooses coins  $\zeta, \xi$  and computes  $\text{Eqcom}(x; \zeta, \xi) \stackrel{\text{def}}{=} \text{com}(x; \zeta) \| \text{com}(x; \xi)$ . It sends  $C_x = \text{Eqcom}(x; \zeta, \xi)$  to the receiver along with a zero-knowledge proof that  $C_x$  was constructed correctly (i.e., that there exist  $x, \zeta, \xi$  such that  $C_x = \text{Eqcom}(x; \zeta, \xi)$ ).

To decommit, the sender chooses a bit  $b$  at random and reveals  $(x, \zeta)$  if  $b = 0$  ( $(x, \xi)$  if  $b = 1$ ), denoted by  $\text{open}_{C_x}$ . Note that a simulator can “equivocate” the commitment by setting  $C = \text{com}(x; \zeta) \| \text{com}(\bar{x}; \xi)$  for a random bit  $x$ , simulating the zero-knowledge proof and then revealing  $\zeta$  or  $\xi$  depending on  $x$  and the bit to be revealed. This extends to strings by committing bitwise.

**Sketch of the Two-Party KO Protocol.** The main component of the two-party KO protocol is Yao’s 3-round protocol  $\Pi_{\text{SH}}$ , described above, secure against semi-honest adversaries. In order to achieve security against a malicious adversary their protocol proceeds as follows. Both parties commit to their inputs; run (modified) coin-tossing protocols to guarantee that each party obtains random coins which are committed to the other party (note that coin flipping for the side of the garbler  $P_2$  is not needed since a malicious garbler  $P_2$  gains nothing by using non-uniform coins. To force  $P_1$  to use random coins the authors use a 3-round sub-protocol which is based on the work of [BL02]); and run the  $\Pi_{\text{SH}}$  protocol together with ZK arguments to avoid adversarial inconsistencies in each round. Then, simulation extractability is guaranteed by the use of WI proof of knowledge and output simulation by the Feige-Shamir ZK argument of knowledge.

However, since even a ZK argument for the first round of the protocol alone will already require 4 rounds, the authors use specific proof systems to achieve in total a 4-round protocol. In particular, the KO protocol uses a specific WI proof of knowledge system with the property that the statement to be proven need not be known until the last round of the protocol, yet soundness, completeness, and witness-indistinguishability still hold. Also, this proof system has the property that the first message from the prover is computed

independently of the statement being proved. Note that their 4-round ZK argument of knowledge enjoys the same properties. Furthermore, their protocol uses an equivocal commitment scheme to commit to the garble circuit for the following reason. Party  $P_1$  may send his round-two message before the proof of correctness for round one given by  $P_2$  is complete. Therefore, the protocol has to be constructed in a way that the proof of correctness for round one completes in round three and that party  $P_2$  reveals the garbled circuit in the third round. But since the proof of security requires  $P_2$  to commit to a garble circuit at the end of the first round,  $P_2$  does so using an equivocal commitment scheme.

### 3 The Exact Round Complexity of Coin Tossing

In this section we first show that it is impossible to construct two-party (simulatable) coin-flipping for a super-logarithmic number of coins in 3 simultaneous message exchange rounds. We first recall the definition of a simulatable coin flipping protocol using the real/ideal paradigm from [KOS03].

**Definition 1** ([KOS03]). *An  $n$ -party protocol  $\Pi$  is a simulatable coin-flipping protocol if it is an  $(n - 1)$ -secure protocol realizing the coin-flipping functionality. That is, for every PPT adversary  $\mathcal{A}$  corrupting at most  $n - 1$  parties there exists an expected PPT simulator  $\mathcal{S}$  such that the (output of the) following experiments are indistinguishable. Here we parse the result of running protocol  $\Pi$  with adversary  $\mathcal{A}$  (denoted*

|   |  |
|---|--|
| REAL( $1^\kappa, 1^\lambda$ )   | IDEAL( $1^\kappa, 1^\lambda$ )   |
| $c, \text{view}_{\mathcal{A}} \leftarrow \text{REAL}_{\Pi, \mathcal{A}}(1^\kappa, 1^\lambda)$ | $c' \leftarrow \{0, 1\}^\lambda$   |
| Output $(c, \text{view}_{\mathcal{A}})$   | $\tilde{c}, \text{view}_{\mathcal{S}} \leftarrow \mathcal{S}^{\mathcal{A}}(c', 1^\kappa, 1^\lambda)$ |
|   | If $\tilde{c} = \{c', \perp\}$ then Output $(\tilde{c}, \text{view}_{\mathcal{S}})$                  |
|   | Else output <b>fail</b>  |

by  $\text{REAL}_{\Pi, \mathcal{A}}(1^\kappa, 1^\lambda)$ ) as a pair  $(c, \text{view}_{\mathcal{A}})$  where  $c \in \{0, 1\}^\lambda \cup \{\perp\}$  is the outcome and  $\text{view}_{\mathcal{A}}$  is the view of the adversary  $\mathcal{A}$ .

We restrict ourselves to the case of two parties ( $n = 2$ ), which can be extended to any  $n > 2$ . Below we denote messages in protocol  $\Pi$  which are sent by party  $P_i$  to party  $P_j$  in the  $\rho$ -th round by  $\mathbf{m}_{i,j}^{\Pi[\rho]}$ .

As mentioned earlier, Katz and Ostrovsky [KO04] showed that simulatable coin-flipping protocol is impossible in 4 rounds without simultaneous message exchange. Since we will use the result for our proofs in this section, we state their result below without giving their proof.

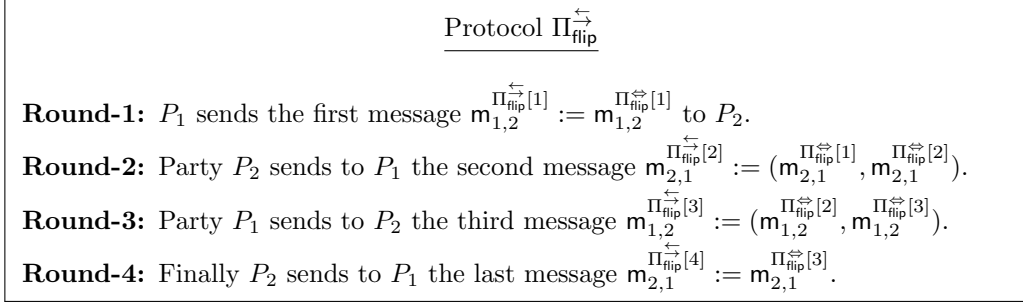
**Lemma 1.** [KO04, Theorem 1] *Let  $p(\kappa) = \omega(\log \kappa)$ , where  $\kappa$  is the security parameter. Then there does not exist a 4-round protocol without simultaneous message transmission for tossing  $p(\kappa)$  coins which can be proven secure via black-box simulation.*

In the following, we state our impossibility result for coin-flipping in 3 rounds of simultaneous message exchange.

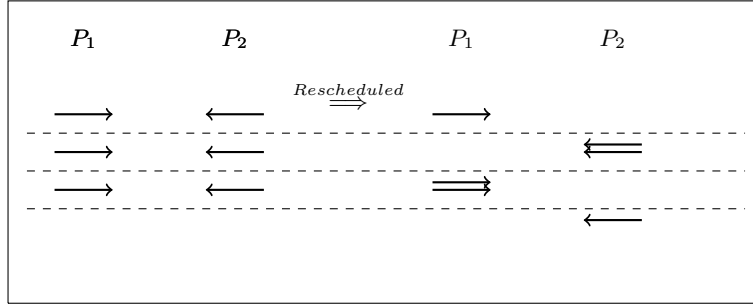
**Lemma 2.** *Let  $p(\kappa) = \omega(\log \kappa)$ , where  $\kappa$  is the security parameter. Then there does not exist a 3-round protocol with simultaneous message transmission for tossing  $p(\kappa)$  coins which can be proven secure via black-box simulation.*

**Proof:** We prove the above statement by showing that a 3-round simultaneous message exchange protocol can be “rescheduled” to a 4-round *non-simultaneous* protocol which contradicts the impossibility of [KO04]. Here by rescheduling we mean rearrangement of the messages without violating mutual dependencies among them, in particular without altering the next-message functions.

For the sake of contradiction, assume that there exists a protocol  $\Pi_{\text{flip}}^{\leftrightarrow}$  which realizes simulatable coin-flipping in 3 simultaneous message exchange rounds, then we can reschedule it in order to construct a protocol  $\Pi_{\text{flip}}^{\leftarrow}$  which realizes simulatable coin-flipping in 4 rounds<sup>9</sup> without simultaneous message exchange as follows:



We provide a pictorial presentation of the above rescheduling in Fig. 1 for better illustration.



**Fig. 1.** A 3-round simultaneous protocol rescheduled to a 4-round non-simultaneous protocol.

Now, without loss of generality assume that  $P_1$  is corrupted. Then we need to build an expected PPT simulator  $\mathcal{S}_{P_1}$  (or simply  $\mathcal{S}$ ) meeting the adequate requirements (according to Def. 1). First note that, since by assumption the protocol  $\Pi_{\text{flip}}^{\leftrightarrow}$  is secure (i.e. achieves Def. 1) the following holds: for any corrupt  $P_1^{\leftrightarrow}$  executing the simultaneous message exchange protocol  $\Pi_{\text{flip}}^{\leftrightarrow}$  there exists an expected PPT simulator  $\mathcal{S}^{\leftrightarrow}$  (let us call it the “inner” simulator and  $\mathcal{S}$  the “outer” simulator) in the ideal world. So,  $\mathcal{S}$  can be constructed using  $\mathcal{S}^{\leftrightarrow}$  for a corrupted party  $P_1^{\leftrightarrow}$  which can be *emulated* by  $\mathcal{S}$  based on  $P_1$ . Finally,  $\mathcal{S}$  just outputs whatever  $\mathcal{S}^{\leftrightarrow}$  returns.  $\mathcal{S}$  emulates the interaction between  $\mathcal{S}^{\leftrightarrow}$  and  $P_1^{\leftrightarrow}$  as follows:

1. On receiving a value  $c' \in \{0, 1\}^\lambda$  from the ideal functionality,  $\mathcal{S}$  runs the inner simulator  $\mathcal{S}^{\leftrightarrow}(c', 1^\kappa, 1^\lambda)$  to get the first message  $\mathbf{m}_{2,1}^{\leftrightarrow[\text{1}]}$ . Notice that in protocol  $\Pi_{\text{flip}}^{\leftrightarrow}$  the first message from (honest) party  $P_2^{\leftrightarrow}$  does not depend on the first message of the corrupted party  $P_1^{\leftrightarrow}$ . So, the inner simulator must be able to produce the first message even before seeing the first message of party  $P_1$  (or the emulated party  $P_1^{\leftrightarrow}$ )<sup>10</sup>. Then it runs  $P_1$  to receive the first message  $\mathbf{m}_{1,2}^{\leftrightarrow[\text{1}]}$ .
2. Then  $\mathcal{S}$  forwards  $\mathbf{m}_{1,2}^{\leftrightarrow[\text{1}]}$  to the inner simulator which then returns the second simulated message  $\mathbf{m}_{2,1}^{\leftrightarrow[\text{2}]}$ .

Now  $\mathcal{S}$  can construct the simulated message  $\mathbf{m}_{2,1}^{\leftarrow[\text{2}]}$  by combining  $\mathbf{m}_{2,1}^{\leftrightarrow[\text{2}]}$  and  $\mathbf{m}_{2,1}^{\leftrightarrow[\text{1}]}$  received earlier (see above) which  $\mathcal{S}$  then forwards to  $P_1$ .

<sup>9</sup> The superscript  $\leftrightarrow$  stands for the simultaneous message exchange setting and  $\leftarrow$  for the setting without simultaneous message exchange

<sup>10</sup> In particular, for so-called “rushing” adversaries, who can wait until receiving the first message and then send its own, the inner simulator must simulate the first message to get the first message from the adversary.

3. In the next step,  $\mathcal{S}$  gets back messages  $\mathbf{m}_{1,2}^{\overleftarrow{\Pi}_{\text{flip}}[3]} = (\mathbf{m}_{1,2}^{\overleftarrow{\Pi}_{\text{flip}}[2]}, \mathbf{m}_{1,2}^{\overleftarrow{\Pi}_{\text{flip}}[3]})$  from  $P_1$ . It then forwards the second message  $\mathbf{m}_{1,2}^{\overleftarrow{\Pi}_{\text{flip}}[2]}$  to  $\mathcal{S}^{\leftrightarrow}$ , which then returns the third simulated message  $\mathbf{m}_{2,1}^{\overleftarrow{\Pi}_{\text{flip}}[3]}$ . Finally it forwards the third message  $\mathbf{m}_{1,2}^{\overleftarrow{\Pi}_{\text{flip}}[3]}$  to  $\mathcal{S}^{\leftrightarrow}$ .
4.  $\mathcal{S}$  outputs whatever transcript  $\mathcal{S}^{\leftrightarrow}$  outputs in the end.
5. Note that, whenever the inner simulator  $\mathcal{S}^{\leftrightarrow}$  asks to rewind the emulated  $P_1^{\leftrightarrow}$ ,  $\mathcal{S}$  rewinds  $P_1$ .

It is not hard to see that the simulator  $\mathcal{S}$  emulates correctly the party  $P_1^{\leftrightarrow}$  and hence by the security of  $\Pi_{\text{flip}}^{\leftrightarrow}$ , the inner simulator  $\mathcal{S}^{\leftrightarrow}$  returns an indistinguishable (with the real world) view. The key-point is that the re-scheduling of the messages from protocol  $\Pi_{\text{flip}}^{\leftrightarrow}$  does not affect the dependency (hence the corresponding next message functions) and hence the correctness and security remains intact in  $\Pi_{\text{flip}}^{\overleftarrow{\cdot}}$ .

We stress that the proof for the case where  $P_2$  is corrupted is straightforward given the above. However, in that case, since  $P_2$ 's first message depends on the first message of honest  $P_1$ , it is mandatory for the inner simulator  $\mathcal{S}^{\leftrightarrow}$  to output the first message before seeing anything even in order to run the corrupted  $P_2$  which is not necessary in the above case. As we stated earlier this is possible as the inner simulator  $\mathcal{S}^{\leftrightarrow}$  should be able to handle rushing adversaries.

Hence we prove that if the underlying protocol  $\Pi_{\text{flip}}^{\leftrightarrow}$  securely realizes simulatable coin-flipping in 3 simultaneous rounds then  $\Pi_{\text{flip}}^{\overleftarrow{\cdot}}$  securely realizes coin-flipping in 4 non-simultaneous rounds which contradicts the KO lower bound (Lemma 1). This concludes the proof.  $\square$

Going a step further we show that any four-round simultaneous message exchange protocol realizing simulatable coin-flipping must satisfy a necessary property, that is each round must be a strictly simultaneous message exchange round, in other words, both parties must send some “non-redundant” message in each round. By “non-redundant” we mean that the next message from the other party must depend on the current message. Below we show the above, otherwise the messages can be again subject to a “rescheduling” mechanism similar to the one in Lemma 2, to yield a four-round non-simultaneous protocol; thus contradicting Lemma 1. More specifically,

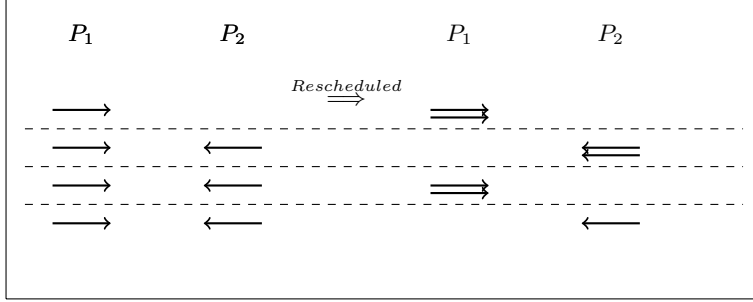
**Lemma 3.** *Let  $p(\kappa) = \omega(\log \kappa)$ , where  $\kappa$  is the security parameter. Then there does not exist a 4-round protocol with at least one unidirectional round (i.e. a round without simultaneous message exchange) for tossing  $p(\kappa)$  coins which can be proven secure via black-box simulation.*

**Proof:** [Proof (Sketch)] We provide a sketch for any protocol with exactly one unidirectional round where only one party, say  $P_1$  sends a message to  $P_2$ . Clearly, there can be four such cases where  $P_2$ 's message is omitted in one of the four rounds. In Fig. 2 we show the case where  $P_2$  does not send the message in the first round, and any such protocol can be re-scheduled (similar to the proof of Lemma 2) to a non-simultaneous 4-round protocol without altering any possible message dependency. This observation can be formalized in a straightforward manner following the proof of Lemma 2 and hence we omit the details. Therefore, again combining with the impossibility from Lemma 1 by [KO04] such simultaneous protocol can not realize simulatable coin-flipping. The other cases can be easily observed by similar rescheduling trick and therefore we omit the details for those cases.  $\square$

## 4 Two-Party Computation in the Simultaneous Message Exchange Model

In this section, we present our two party protocol for computing any functionality in the presence of a static, malicious and rushing adversary. As discussed earlier, we are in the simultaneous message exchange channel setting where both parties can simultaneously exchange messages in each round. The structure of this protocol will provide a basis for our later protocols as well.

An overview of the protocol appears in the introduction (Sec. 1). In a high level, the protocol consists of two simultaneous executions of a one-sided (single-output) protocol to guarantee that both parties learn



**Fig. 2.** Rescheduling when  $P_2$  does not send the first message.

the output. The overall skeleton of the one-sided protocol resembles the KO protocol [KO04] which uses a clever combination of OT, coin-tossing, and  $\Pi_{\text{WIPOK}}$  to ensure that the protocol is executed with a fixed input (allowing at the same time simulation extractability of the input), and relies on the zero-knowledge property of  $\Pi_{\text{FS}}$  to “force the output”. A sketch of the KO protocol is given in Section 2.2. In order to ensure “independence of inputs” our protocol relies heavily on non-malleable commitments. To this end, we change the one-sided protocol to further incorporate non-malleable commitments so that similar guarantees can be obtained even in the presence of the “opposite side” protocol, and we further rely on zero-knowledge proofs to ensure that parties use the same input in both executions.

## 4.1 Our Protocol

To formally define our protocol, let:

- $(\text{GenGC}, \text{EvalGC})$  be the garbled-circuit mechanism with simulator  $\text{SimGC}$ ;  $\mathcal{F} = (\text{Gen}, \text{Eval}, \text{Invert})$  be a family of TDPs with domain  $\{0, 1\}^\kappa$ ;  $\text{H}$  be the hardcore bit function for  $\kappa$  bits;  $\text{com}$  be a perfectly binding non-interactive commitment scheme;  $\text{Eqcom}$  be the equivocal scheme based on  $\text{com}$ , as described in Section 2;
- $\text{nmcom}$  be a tag based, *parallel*<sup>11</sup> and *3-robust* non-malleable commitment scheme for strings, supporting tags/identities of length  $\kappa$ ;
- $\Pi_{\text{WIPOK}}$  be the witness-indistinguishable proof-of-knowledge for NP as described in Section 2;
- $\Pi_{\text{FS}}$  be the proof system for NP, based on  $\text{nmcom}$  and  $\Pi_{\text{WIPOK}}$ , as described in Section 2;
- **Simplifying assumption:** *for notational convenience only*, we assume for now that  $\text{nmcom}$  consists of exactly three rounds, denoted by  $(\text{nm}_1, \text{nm}_2, \text{nm}_3)$ . This assumption is removed later (see Remark 1).

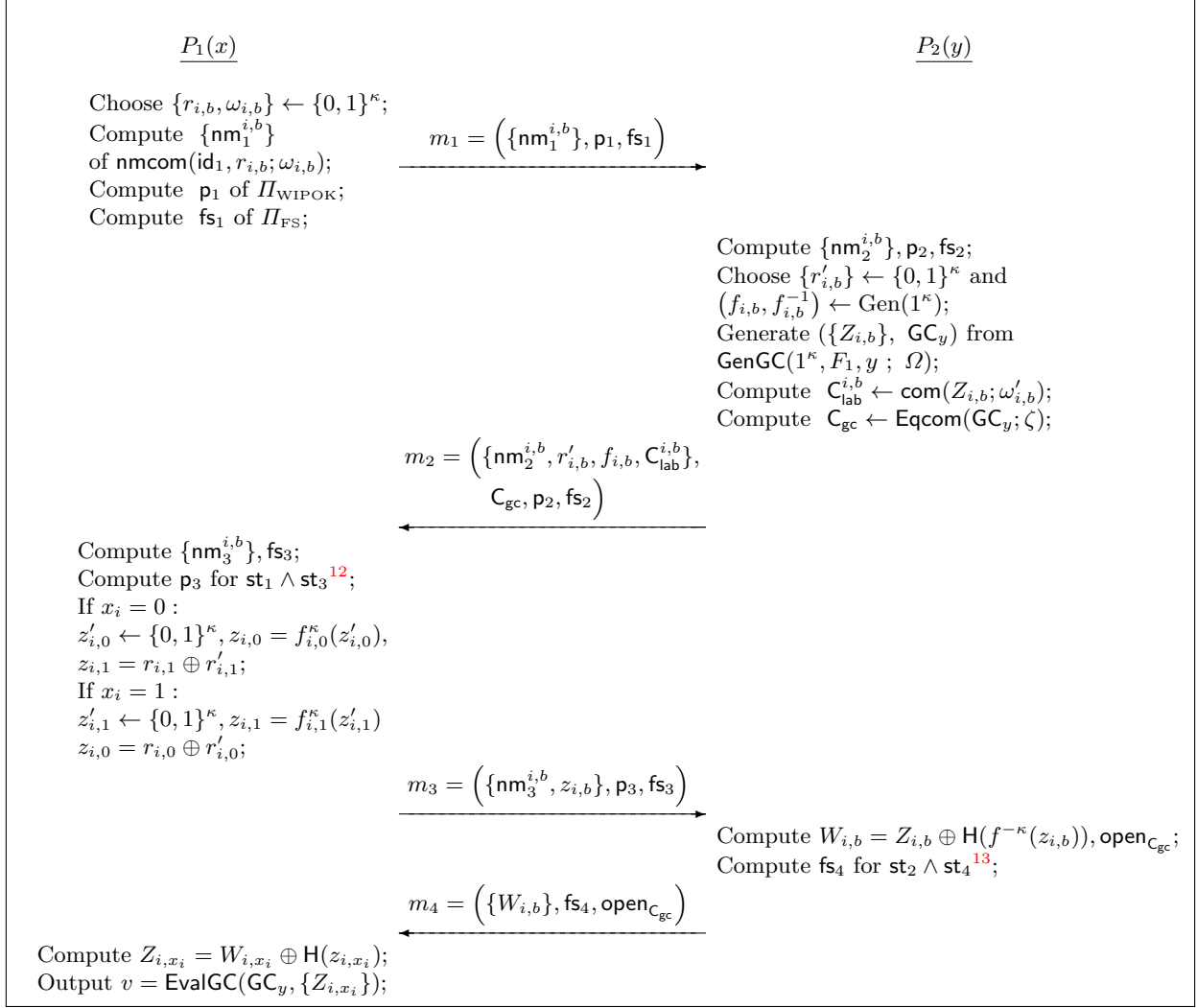
We also assume that the first round,  $\text{nm}_1$ , is from the committer and statistically determines the message to be committed. We use the notation  $\text{nm}_1 = \text{nmcom}_1(\text{id}, r; \omega)$  to denote the committer’s first message when executing  $\text{nmcom}$  with identity  $\text{id}$  to commit to string  $r$  with randomness  $\omega$ .

We are now ready to describe our protocol. A high level sketch of the left execution of our protocol where  $P_1$  receives the output is given in Figure 3.

<sup>11</sup> We actually need security against an a-priori bounded number of polynomial executions. Almost all known protocols for  $\text{nmcom}$  have this additional property.

<sup>12</sup> Informally,  $\text{st}_1$  represents that  $P_1$  “knows” one of the decommitment values of the first round for every  $i$  and  $\text{st}_3$  says that  $P_1$  correctly constructed  $\{z_{i,b}\}$ .

<sup>13</sup> Informally,  $\text{st}_2$  is the statement that  $P_2$  performed his first step correctly and  $\text{st}_4$  is the statement that  $P_2$  performed *both* oblivious transfers correctly.



**Fig. 3.** High-level description of the left execution of  $\Pi_{2PC}$ .

**Protocol  $\Pi_{2PC}$ .** We denote the two parties by  $P_1$  and  $P_2$ ;  $P_1$  holds input  $x \in \{0, 1\}^\kappa$  and  $P_2$  holds input  $y \in \{0, 1\}^\kappa$ . Furthermore, the identities of  $P_1, P_2$  are  $id_1, id_2$  respectively where  $id_1 \neq id_2$ . Let  $F := (F_1, F_2) : \{0, 1\}^\kappa \times \{0, 1\}^\kappa \rightarrow \{0, 1\}^\kappa \times \{0, 1\}^\kappa$  be the functions to be computed.

The protocol consists of four (strictly) simultaneous message exchange rounds, i.e., both parties send messages in each round. The protocol essentially consists of two simultaneous executions of a protocol in which only one party learns the output. In the first protocol,  $P_1$  learns the output and the messages of this protocol are denoted by  $(m_1, m_2, m_3, m_4)$  where  $(m_1, m_3)$  are sent by  $P_1$  and  $(m_2, m_4)$  are sent by  $P_2$ . Likewise, in the second protocol  $P_2$  learns the output and the messages of this protocol are denoted by  $(\tilde{m}_1, \tilde{m}_2, \tilde{m}_3, \tilde{m}_4)$  where  $(\tilde{m}_1, \tilde{m}_3)$  are sent by  $P_2$  and  $(\tilde{m}_2, \tilde{m}_4)$  are sent by  $P_1$ . Therefore, messages  $(m_j, \tilde{m}_j)$  are exchanged simultaneously in the  $j$ -th round,  $j \in \{1, \dots, 4\}$  (see figure 4).

We now describe how these messages are constructed in each round below. In the following  $i$  always ranges from 1 to  $\kappa$  and  $b$  from 0 to 1.

**Round 1.** In this round  $P_1$  sends a message  $m_1$  and  $P_2$  sends a symmetrically constructed message  $\tilde{m}_1$ . We first describe how  $P_1$  constructs  $m_1$ .

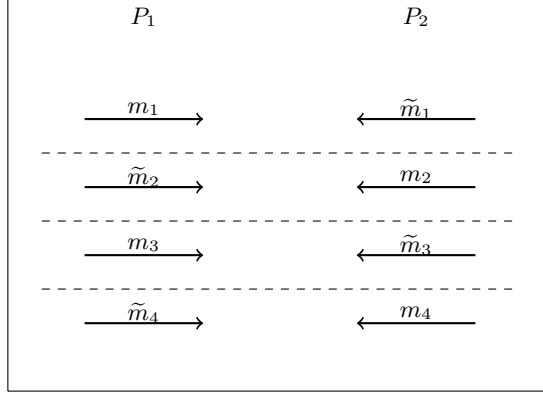


Fig. 4. 2-PC in the simultaneous message exchange model.

Actions of  $P_1$ :

1.  $P_1$  starts by committing to  $2\kappa$  random strings  $\{(r_{1,0}, r_{1,1}), \dots, (r_{\kappa,0}, r_{\kappa,1})\}$  using  $2\kappa$  parallel and independent executions of  $\text{nmcom}$  with identity  $\text{id}_1$ . I.e., it uniformly chooses strings  $r_{i,b}$ , randomness  $\omega_{i,b}$ , and generates  $\text{nm}_1^{i,b}$  which is the first message corresponding to the execution of  $\text{nmcom}(\text{id}_1, r_{i,b}; \omega_{i,b})$ .
2.  $P_1$  prepares the first message  $\mathbf{p}_1$  of  $\Pi_{\text{WIPOK}}$ , as well as the first message  $\mathbf{fs}_1$  of  $\Pi_{\text{FS}}$ . For later reference, define  $\mathbf{st}_1$  to be the following:  $\exists \{(r_i, \omega_i)\}_{i \in [\kappa]}$  s.t.:

$$\forall i : (\text{nm}_1^{i,0} = \text{nmcom}_1(\text{id}_1, r_i; \omega_i) \vee \text{nm}_1^{i,1} = \text{nmcom}_1(\text{id}_1, r_i; \omega_i))$$

Informally,  $\mathbf{st}_1$  represents that  $P_1$  “knows” one of the decommitment values for every  $i$ .

3. Message  $m_1$  is defined to be the tuple  $(\{\text{nm}_1^{i,b}\}, \mathbf{p}_1, \mathbf{fs}_1)$ .

Actions of  $P_2$ :

Performs the same actions as  $P_1$  to sample the values  $\{(\tilde{r}_{i,b}, \tilde{\omega}_{i,b})\}$  and constructs  $\tilde{m}_1 := (\{\tilde{\text{nm}}_1^{i,b}\}, \tilde{\mathbf{p}}_1, \tilde{\mathbf{fs}}_1)$  where all  $\tilde{\text{nm}}_1^{i,b}$  are generated with  $\text{id}_2$ . Define the statement  $\tilde{\mathbf{st}}_1$  analogously for these values.

**Round 2.** In this round  $P_2$  sends a message  $m_2$  and  $P_1$  sends a symmetrically constructed message  $\tilde{m}_2$ . We first describe how  $P_2$  constructs  $m_2$ .

Actions of  $P_2$ :

1.  $P_2$  generates the second messages  $\{\text{nm}_2^{i,b}\}$  corresponding to all executions of  $\text{nmcom}$  initiated by  $P_1$  (with  $\text{id}_1$ ).
2.  $P_2$  prepares the second message  $\mathbf{p}_2$  of the  $\Pi_{\text{WIPOK}}$  protocol initiated by  $P_1$ .
3.  $P_2$  samples random strings  $\{r'_{i,b}\}$  and  $(f_{i,b}, f_{i,b}^{-1}) \leftarrow \text{Gen}(1^\kappa)$  for the oblivious transfer executions.
4.  $P_2$  obtains the garbled labels and the circuit for  $F_1$ :  $(\{Z_{i,b}\}, \text{GC}_y) = \text{GenGC}(1^\kappa, F_1, y; \Omega)$ .
5.  $P_2$  generates standard commitments to the labels, and an equivocal commitment to the garbled circuit: i.e.,  $C_{\text{lab}}^{i,b} \leftarrow \text{com}(Z_{i,b}; \omega'_{i,b})$  and  $C_{\text{gc}} \leftarrow \text{Eqcom}(\text{GC}_y; \zeta)$ .
6.  $P_2$  prepares the second message  $\mathbf{fs}_2$  of the  $\Pi_{\text{FS}}$  protocol initiated by  $P_1$ . For later reference, define  $\mathbf{st}_2$  to be the following:  $\exists (y, \Omega, \text{GC}_y, \{Z_{i,b}, \omega'_{i,b}\}, \zeta)$  s.t.:
  - (a)  $(\{Z_{i,b}\}, \text{GC}_y) = \text{GenGC}(1^\kappa, F_1, y; \Omega)$
  - (b)  $\forall (i, b) : C_{\text{lab}}^{i,b} = \text{com}(Z_{i,b}; \omega'_{i,b})$
  - (c)  $C_{\text{gc}} = \text{Eqcom}(\text{GC}_y; \zeta)$
(Informally,  $\mathbf{st}_2$  is the statement that  $P_2$  performed this step correctly.)



7. Define message  $m_2 := \left( \{nm_2^{i,b}, r'_{i,b}, f_{i,b}, C_{\text{lab}}^{i,b}\}, C_{\text{gc}}, p_2, fs_2 \right)$ .

Actions of  $P_1$ :

Performs the same actions as  $P_2$  in the previous step to construct the message  $\tilde{m}_2 := \left( \{\tilde{nm}_2^{i,b}, \tilde{r}'_{i,b}, \tilde{f}_{i,b}, \tilde{C}_{\text{lab}}^{i,b}\}, \tilde{C}_{\text{gc}}, \tilde{p}_2, \tilde{fs}_2 \right)$  w.r.t. identity  $\text{id}_2$ , function  $F_2$ , and input  $x$ . Define the (remaining) values  $\tilde{f}'_{i,b}{}^{-1}, \tilde{Z}_{i,b}, \tilde{\omega}'_{i,b}, \tilde{GC}_x, \tilde{\Omega}, \tilde{\zeta}$  and statement  $\tilde{st}_2$  analogously.

**Round 3.** In this round  $P_1$  sends a message  $m_3$  and  $P_2$  sends a symmetrically constructed message  $\tilde{m}_3$ . We first describe how  $P_1$  constructs  $m_3$ .

Actions of  $P_1$ :

1.  $P_1$  prepares the third message  $\{nm_3^{i,b}\}$  of  $\text{nmcom}$  (with  $\text{id}_1$ ).
2. If any of  $\{f_{i,b}\}$  are invalid,  $P_1$  aborts. Otherwise, it invokes  $\kappa$  parallel executions of oblivious transfer to obtain the input-wire labels corresponding to its input  $x$ . More specifically,  $P_1$  proceeds as follows:
  - If  $x_i = 0$ , sample  $z'_{i,0} \leftarrow \{0, 1\}^\kappa$ , set  $z_{i,0} = f_{i,0}^\kappa(z'_{i,0})$ , and  $z_{i,1} = r_{i,1} \oplus r'_{i,1}$ .
  - If  $x_i = 1$ , sample  $z'_{i,1} \leftarrow \{0, 1\}^\kappa$ , set  $z_{i,1} = f_{i,1}^\kappa(z'_{i,1})$ , and  $z_{i,0} = r_{i,0} \oplus r'_{i,0}$ .
3. Define  $st_3$  to be the following:  $\exists \{(r_i, \omega_i)\}_{i \in [\kappa]}$  s.t.  $\forall i$ :
  - (a)  $(nm_1^{i,0} = \text{nmcom}_1(\text{id}_1, r_i; \omega_i) \wedge z_{i,0} = r_i \oplus r'_{i,0})$ , **or**
  - (b)  $(nm_1^{i,1} = \text{nmcom}_1(\text{id}_1, r_i; \omega_i) \wedge z_{i,1} = r_i \oplus r'_{i,1})$
 Informally,  $st_3$  says that  $P_1$  correctly constructed  $\{z_{i,b}\}$ .
4.  $P_1$  prepares the final message  $p_3$  of  $\Pi_{\text{WIPOK}}$  proving the statement:  $st_1 \wedge st_3$ .<sup>14</sup>  $P_1$  also prepares the third message  $fs_3$  of  $\Pi_{\text{FS}}$ .
5. Define  $m_3 := \left( \{nm_3^{i,b}, z_{i,b}\}, p_3, fs_3 \right)$  to  $P_2$ .

Actions of  $P_2$ :

Performs the same actions as  $P_1$  in the previous step to construct the message  $\tilde{m}_3 := \left( \{\tilde{nm}_3^{i,b}, \tilde{z}_{i,b}\}, \tilde{p}_3, \tilde{fs}_3 \right)$  w.r.t. identity  $\text{id}_2$  and input  $y$ . The (remaining) values  $\{\tilde{z}_{i,b}, \tilde{z}'_{i,b}\}$  and statement  $\tilde{st}_3$  are defined analogously.

**Round 4.** In this round  $P_2$  sends a message  $m_4$  and  $P_1$  sends a symmetrically constructed message  $\tilde{m}_4$ . We first describe how  $P_2$  constructs  $m_4$ .

Actions of  $P_2$ :

1. If  $p_3, fs_3$  are not accepting,  $P_2$  aborts. Otherwise,  $P_2$  completes the execution of the oblivious transfers for every  $(i, b)$ . I.e., it computes  $W_{i,b} = Z_{i,b} \oplus H(f^{-\kappa}(z_{i,b}))$ . Moreover,  $P_2$  decommits  $C_{\text{gc}}$  as  $GC_y$ , denoted by  $\text{open}_{C_{\text{gc}}}$  to  $A_1$ .
2. Define  $st_4$  to be the following:  $\exists (y, \Omega, GC_y, \{Z_{i,b}\}, \omega'_{i,b}, z'_{i,b}, \tilde{z}'_i)_{i \in [\kappa], b \in \{0,1\}}$  s.t.
  - (a)  $\forall (i, b): \left( C_{\text{lab}}^{i,b} = \text{com}(Z_{i,b}; \omega'_{i,b}) \right) \wedge \left( f_{i,b}^\kappa(z'_{i,b}) = z_{i,b} \right) \wedge \left( W_{i,b} = Z_{i,b} \oplus H((z'_{i,b})) \right)$
  - (b)  $\left( (\{Z_{i,b}\}, GC_y) = \text{GenGC}(1^\kappa, F_1, y; \Omega) \right) \wedge \left( C_{\text{gc}} = \text{Eqcom}(GC_y; \zeta) \right)$
  - (c)  $\forall i: \tilde{z}_{i,y_i} = \tilde{f}'_{i,y_i}(\tilde{z}'_i)$
 Informally, this means that  $P_2$  performed *both* oblivious transfers correctly.
3.  $P_2$  prepares the final message  $fs_4$  of  $\Pi_{\text{FS}}$  proving the statement  $st_2 \wedge st_4$ .<sup>15</sup>
4. Define  $m_4 := \left( \{W_{i,b}\}, fs_4, \text{open}_{C_{\text{gc}}} \right)$ .

Actions of  $P_1$ :

Performs the same actions as  $P_2$  in the previous step to construct the message  $\tilde{m}_4 := \left( \{\tilde{W}_{i,b}\}, \tilde{fs}_4, \tilde{\text{open}}_{C_{\text{gc}}} \right)$  and analogously defined statement  $\tilde{st}_4$ .

**Output computation.**

<sup>14</sup> Honest  $P_1$  knows multiple witnesses for  $st_1$ . For concreteness, we have to use one of them randomly in the proof.

<sup>15</sup> Recall that  $\Pi_{\text{FS}}$  is a modified version of FS protocol: it uses two executions of  $\text{nmcom}$  to construct its first message, namely, the first message consists of  $(nm_1^1, nm_1^2)$  corresponding to two executions of  $\text{nmcom}$  committing to strings  $\sigma_1, \sigma_2$  (see Section 2).

**$P_1$ 's output:** If any of  $(\mathbf{fs}_4, \text{GC}_y, \text{open}_{\text{GC}_y})$  or the openings of  $\{W_{i,b}\}$  are invalid,  $P_1$  aborts. Otherwise,  $P_1$  recovers the garbled labels  $\{Z_i := Z_{i,x_i}\}$  from the completion of the oblivious transfer, and computes  $F_1(x, y) = \text{EvalGC}(\text{GC}_y, \{Z_i\})$ .

**$P_2$ 's output:** If any of  $(\tilde{\mathbf{fs}}_4, \text{GC}_x, \widetilde{\text{open}}_{\text{GC}_x})$  or the openings of  $\{\tilde{W}_{i,b}\}$  are invalid,  $P_2$  aborts. Otherwise,  $P_2$  recovers the garbled labels  $\{\tilde{Z}_i := \tilde{Z}_{i,y_i}\}$  from the completion of the oblivious transfer, and computes  $F_2(x, y) = \text{EvalGC}(\text{GC}_x, \{\tilde{Z}_i\})$ .

*Remark 1.* If  $\text{nmcom}$  has  $k > 3$  rounds, the first  $k - 3$  rounds can be performed before the 4 rounds of  $\Pi_{2\text{PC}}$  start; this results in a protocol with  $k + 1$  rounds. If  $k < 3$ , then the protocol has only 4 rounds. Also, for large  $k$ , it suffices if the first  $k - 2$  rounds of  $\text{nmcom}$  statistically determine the message to be committed; the notation is adjusted to simply use the transcript up to  $k - 2$  rounds to define the statements for the proof systems.

Finally, the construction is described for a deterministic  $F$ . Known transformations (see [Gol04, Section 7.3]) yield a protocol for randomized functionalities, without increasing the rounds.

## 4.2 Proof of Security

We prove the security of our protocol according to the ideal/real paradigm. We design a sequence of hybrids where we start with the real world execution and gradually modify it until the input of the honest party is not needed. The resulting final hybrid represents the simulator for the ideal world.

**Theorem 1** *Assuming the existence of a trapdoor permutation family and a  $k$ -round parallel and 3-robust non-malleable commitment scheme, protocol  $\Pi_{2\text{PC}}$  securely computes every two-party functionality  $F = (F_1, F_2)$  with black-box simulation in the presence of a malicious adversary. The round complexity of  $\Pi_{2\text{PC}}$  is  $k' = \max(4, k + 1)$ .*

**Proof:** Due to the symmetric nature of our protocol, it is *sufficient* to prove security against the malicious behavior of any party, say  $P_1$ . We show that for every adversary  $\mathcal{A}$  who participates as  $P_1$  in the “real” world execution of  $\Pi_{2\text{PC}}$ , there exists an “ideal” world adversary (simulator)  $\mathcal{S}$  such that for all inputs  $x, y$  of equal length and security parameter  $\kappa \in \mathbb{N}$ :

$$\{\text{IDEAL}_{F, \mathcal{S}}(\kappa, x, y)\}_{\kappa, x, y} \stackrel{c}{\approx} \{\text{REAL}_{\Pi, \mathcal{A}}(\kappa, x, y)\}_{\kappa, x, y}$$

We prove this claim by considering hybrid experiments  $H_0, H_1, \dots$  as described below. We denote by  $\text{Hybrid}_{F, \mathcal{S}^i}^i(\kappa, \cdot)$  the random variable that corresponds to the simulator’s output in hybrid execution  $H_i$ . For simplicity of exposition, we abbreviate it to  $\text{Hybrid}_i$ . We start with  $H_0$  which has access to both inputs  $x$  and  $y$ , and gradually get rid of the honest party’s input  $y$  to reach the final hybrid.

$H_0$ : Identical to the real execution. More specifically,  $H_0$  starts the execution of  $\mathcal{A}$  providing it fresh randomness and input  $x$ , and interacts with it honestly by performing all actions of  $P_2$  with uniform randomness and input  $y$ . The output consists of  $\mathcal{A}$ ’s view.

By construction,  $H_0$  and the output of  $\mathcal{A}$  in the real execution are identically distributed.

$H_1$ : Identical to  $H_0$  except that this hybrid also performs extraction of  $\mathcal{A}$ ’s implicit input  $x^*$  from  $\Pi_{\text{WIPOK}}$ ; in addition, it also extracts the “simulation trapdoor”  $\sigma$  from the first three rounds  $(\mathbf{fs}_1, \mathbf{fs}_2, \mathbf{fs}_3)$  of  $\Pi_{\text{FS}}$ .<sup>16</sup> More specifically,  $H_1$  proceeds as follows:

1. It completes the first three broadcast rounds exactly as in  $H_0$ , and waits until  $\mathcal{A}$  either aborts or successfully completes the third round.

<sup>16</sup> Recall that  $(\mathbf{fs}_1, \mathbf{fs}_2, \mathbf{fs}_3)$  contains two non-malleable commitments (to values  $\sigma_1, \sigma_2$ ) along with proof-of-knowledge of one of the committed values (see appendix B.2) using  $\Pi_{\text{WIPOK}}$ ; this execution of  $\Pi_{\text{WIPOK}}$  runs in parallel and therefore, it is possible to extract from it at the same time as  $x^*$ .

2. At this point,  $H_1$  proceeds to extract the witness corresponding to each proof-of-knowledge completed in the first three rounds.

Specifically,  $H_1$  defines a cheating prover  $P^*$  which acts identically to  $H_0$ , simulating all messages for  $\mathcal{A}$ , except those corresponding to (each execution of)  $\Pi_{\text{WIPOK}}$  which are forwarded outside. It then applies the extractor of  $\Pi_{\text{WIPOK}}$  to obtain the “witnesses” which consists of the following: values  $\{(r_i, \omega_i)\}_{i \in [\kappa]}$  which is the witness for  $\text{st}_1 \wedge \text{st}_3$ , and a value  $(\sigma, \omega_\sigma)$  which is the simulation trapdoor for  $\Pi_{\text{FS}}$ .

If extraction fails,  $H_1$  outputs fail. Otherwise, let  $b_i \in \{0, 1\}$  be such that  $\text{nmcom}_1^{i, b_i} = \text{nmcom}_1(\text{id}_1, r_i; \omega_i)$ .  $H_1$  defines a string  $x^* = (x_1^*, \dots, x_\kappa^*)$  as follows:

$$\text{If } z_{i, b_i} = r_i \oplus r'_{i, b_i} \text{ then } x_i^* = 1 - b_i; \text{ otherwise } x_i^* = b_i$$

3.  $H_1$  completes the final round and prepares the output exactly as  $H_0$ .

**Claim 1**  $H_1$  is expected polynomial time, and  $\text{Hybrid}_0 \approx_s \text{Hybrid}_1$ .

*Proof sketch:* This is a (completely) standard proof which we sketch here. Let  $p$  be the probability with which  $\mathcal{A}$  completes  $\Pi_{\text{WIPOK}}$  in the third round, and let  $\text{trans}$  be the transcript. The extractor for  $\Pi_{\text{WIPOK}}$  takes expected time  $\text{poly}(\kappa)/p$  and succeeds with probability  $1 - \mu(\kappa)$ . It follows that the expected running time of  $H_1$  is  $\text{poly}(\kappa) + p \cdot \frac{\text{poly}(\kappa)}{p} = \text{poly}(\kappa)$ , and its output is statistically close to that of  $H_0$ .<sup>17</sup>  $\diamond$

$H_2$ : Identical to  $H_1$  except that this hybrid uses the simulation trapdoor  $(\sigma, \omega_\sigma)$  as the witness to compute  $\text{fs}_4$  in the last round. (Recall that  $\text{fs}_4$  is the last round of an execution of  $\Pi_{\text{WIPOK}}$ .)

It is easy to see that  $H_2$  and  $H_1$  are computationally indistinguishable due the WI property of  $\Pi_{\text{WIPOK}}$ .

$H_3$ : In this hybrid, we change the witness used in  $\Pi_{\text{WIPOK}}$  by the honest party  $P_2$  (for proving the statement  $\tilde{\text{st}}_1 \wedge \tilde{\text{st}}_3$ ). In particular, instead of using the witness corresponding to its input  $y$ ,  $P_2$  uses a random witness. (Note that at this point,  $P_2$  does not really need to know any pre-images, so it can set all  $\tilde{z}$ -values as XOR of the corresponding random strings instead of computing some of them using  $f$ ).

**Claim 2**  $\text{Hybrid}_2 \approx_c \text{Hybrid}_3$ .

*Proof.* We primarily rely on the witness indistinguishability property of  $\Pi_{\text{WIPOK}}$  to prove this claim. However, to use it correctly, we also need to crucially use the fact that  $\text{nmcom}$  is *robust* w.r.t. all 3-round protocols (and hence  $\Pi_{\text{WIPOK}}$ ). The main reason for this is as follows: the WIPOK protocol runs in parallel with  $P_2$ 's protocol from which the simulation trapdoor  $\sigma$  was extracted and used in the previous hybrid. This trapdoor will still be needed to complete the view, the “rewindings” for getting  $\sigma$  will violate the WI property of the WIPOK protocol. This will make it impossible to argue that changing the witness in this hybrid keeps the view indistinguishable.

We get around this by realizing that since  $\text{nmcom}$  is 3-robust, we can participate in an external execution of WIPOK and then stop the internal execution without rewinding. We will know  $\sigma$  at this point, but we will have a partially generated view. Since we modified  $\Pi_{\text{FS}}$  to use  $\text{nmcom}$ , this view contains a non-malleable commitment to  $\sigma$ . Now, if the full views were distinguishable, we can violate the 3-robust non-malleability of  $\text{nmcom}$  as follows: we give the partial view and the value committed in the  $\text{nmcom}$  (which is  $\sigma$ ) to a distinguisher; the distinguisher now has  $\sigma$  (simulation trapdoor), which it can use to convert the partial view into full view just like  $H_2$  does. So, if the full views were indistinguishable, by 3-robustness, the external protocol must not have been WI. The details follow.

Assume, for contradiction, that there exists a distinguisher  $D$  that distinguishes  $H_2$  and  $H_3$ . We show how this violates the robustness of  $\text{nmcom}$ . Consider the machine  $B(b^\kappa)$ , which upon receiving a statement  $x$ , and two witnesses for  $x$ ,  $w_0$  and  $w_1$ , provides a  $\Pi_{\text{WIPOK}}$  proof of the statement  $x$  using witness  $w_b$ . Since  $\Pi_{\text{WIPOK}}$  is a 3-round protocol where the second message is sent from the verifier,  $(x, w_0, w_1)$  can be sent with the second message from the  $\Pi_{\text{WIPOK}}$  verifier. It follows directly from the WI property of

<sup>17</sup> See “witness extended emulation” in [Lin01] for full exposition.

the  $\Pi_{\text{WIPOK}}$  proof that no (non-uniform) PPT adversary can distinguish interactions with  $B(0^\kappa)$  and  $B(1^\kappa)$ .

Next, we construct an adversary  $\tilde{M}$  such that the view and values that  $\tilde{M}$  committed to while interacting with  $B(0^\kappa)$  and  $B(1^\kappa)$  can be distinguished by a distinguisher  $\tilde{D}$  that appropriately incorporates  $D$ .

**Description of  $\tilde{M}$ :** Machine  $\tilde{M}$  proceeds in the following two phases.

- **Main execution phase:**  $\tilde{M}$  incorporates a man-in-the-middle adversary  $\mathcal{A}$  internally, and proceeds exactly as  $H_2$  by sampling all messages internally except for the messages of  $\Pi_{\text{WIPOK}}$ . In particular,  $\tilde{M}$  honestly emulates the left committer and right receivers for  $\mathcal{A}$  with the following exceptions:
  1. To emulate the proof of the left interaction it externally sends  $B$  the statement  $x$  and the witnesses  $w_0, w_1$  where  $w_1$  is the “fake witness”. It next forwards the  $\Pi_{\text{WIPOK}}$  proof from  $B$  to  $\mathcal{A}$ .
  2. In the right interactions, it externally forwards messages from  $\mathcal{A}$  to an honest receiver of  $\text{nmcom}$ .
- **Output phase:** After the first three rounds of the protocol are finished,  $\tilde{M}$  halts by outputting its view, denoted by  $\mathcal{V}_{\tilde{M}}$ . In particular,  $\tilde{M}$  does not continue further like  $H_2$ , it does not extract any values, and does not complete the fourth round. (In fact,  $\tilde{M}$  cannot complete the fourth round, since it does not have the witness.) Formally, for  $i \in [\kappa], b \in \{0, 1\}$ ,  $\tilde{M}$  outputs  $\mathcal{V}_{\tilde{M}}$  that includes the view of  $\tilde{M}$  from the *main execution phase* and the values  $\{\tilde{u}_{i,b}\}$  committed by  $\tilde{M}$  in the  $\kappa$  sessions on the right with tags  $\{\tilde{\text{id}}_i\}$  while receiving parallel commitments to  $\{u_{i,b}\}$  on left with tags  $\{\text{id}_i\}$ .

**Description of  $\tilde{D}$ :** Distinguisher  $\tilde{D}$  proceeds in the following two phases.

- **Main execution phase:**  $\tilde{D}$  incorporates both  $\tilde{M}$  and  $D$  internally, it receives as input the view  $\mathcal{V}_{\tilde{M}}$  and proceeds as follows:
  1.  $\tilde{D}$  parses  $\mathcal{V}_{\tilde{M}}$  to obtain the view  $\mathcal{V}_{\mathcal{A}}$  of  $\mathcal{A}$ , the committed values  $\{\tilde{u}_{i,b}\}$  and consequently a string  $\sigma$  corresponding to the “trapdoor witness.”<sup>18</sup>
  2.  $\tilde{D}$  starts  $\tilde{M}$  and feeds him the view  $\mathcal{V}_{\mathcal{A}}$  together with the committed values and continues the execution just like  $H_2$ . It, however, does not rewind  $\mathcal{A}$  (internal to  $\tilde{M}$ ), instead it uses  $\sigma$  and values in  $\mathcal{V}_{\mathcal{A}}$  to complete the last round of the protocol.
- **Output phase:** When  $\mathcal{A}$  halts,  $\tilde{D}$  feeds the view  $\mathcal{V}_{\mathcal{A}}$  to  $D$  and outputs whatever  $D$  outputs.

Since  $\tilde{M}$ , in interaction with  $B(0^\kappa)$  (resp.,  $B(1^\kappa)$ ), perfectly emulates the view of  $\mathcal{A}$  in the hybrid experiment  $H_2$  (resp.,  $H_3$ ), the extracted view  $\mathcal{V}_{\mathcal{A}}$  and the committed values are identically distributed to the view of  $H_2$  (resp.,  $H_3$ ). It follows that  $\tilde{D}$  distinguishes the view and the values committed to by  $\tilde{M}$  using  $\text{nmcom}$ , which contradicts the robustness property of  $\text{nmcom}$ . The claim follows.  $\diamond$

$H_4$ : In this hybrid, we get rid of  $P_2$ 's input  $y$  that is implicitly present in values  $\{\tilde{z}_{i,b}\}$  and  $\{\tilde{r}_{i,b}\}$  in  $\text{nmcom}$  (but keep it everywhere else for the time being). Formally,  $H_4$  is identical to  $H_3$  except that in round 3 it sets  $\tilde{z}_{i,b} = \tilde{r}_{i,b} \oplus \tilde{r}'_{i,b}$  for all  $(i, b)$ .

**Claim 3**  $\text{Hybrid}_3 \approx_c \text{Hybrid}_4$ .

*Proof.* We rely on the non-malleability of  $\text{nmcom}$  to prove this claim. Assume, for contradiction, that there exists a distinguisher  $D$  that distinguishes  $H_3$  and  $H_4$ . We show how this violates the non-malleability property of  $\text{nmcom}$ .

The high level idea is as follows: first we define two string sequences  $\{u_{i,b}^0\}$  and  $\{u_{i,b}^1\}$  and a man-in-the-middle  $M$  which incorporates  $\mathcal{A}$  and receives non-malleable commitments to one of these sequences in parallel. Then we define a distinguisher  $D_{\text{nm}}$  which incorporates both  $M$  and  $D$ , takes as input the value committed by  $M$  and its view, and can distinguish which sequence was committed to  $M$ . This violates non-malleability of  $\text{nmcom}$ .

Formally, we define a man-in-middle adversary  $M$  who receives  $2\kappa$   $\text{nmcom}$  commitments on the left and makes  $2\kappa$   $\text{nmcom}$  commitments on the right.

**Description of  $M$ :** Machine  $M$  proceeds in the following two phases.

<sup>18</sup> Note that, by construction, such a value is guaranteed in both sequences and w.l.o.g. can be the value in the first  $\text{nmcom}$ .

- **Main execution phase:**  $M$  incorporates  $\mathcal{A}$  internally, and proceeds exactly as  $H_3$  by sampling all messages internally except for the messages of  $\text{nmcom}$  corresponding to  $P_2$ . These messages are received from an outside committer as follows.

1.  $M$  samples uniformly random values  $\{\tilde{z}_{i,b}\}$  and  $\{\tilde{r}'_{i,b}\}$  and defines  $\{u_{i,b}^0\}$  and  $\{u_{i,b}^1\}$  as:

$$u_{i,\bar{y}_i}^0 = \tilde{z}_{i,\bar{y}_i} \oplus \tilde{r}'_{i,\bar{y}_i}, \quad u_{i,y_i}^0 \leftarrow \{0,1\}^\kappa, \quad u_{i,b}^1 = \tilde{z}_{i,b} \oplus \tilde{r}'_{i,b} \quad \forall(i,b)$$

2.  $M$  forwards  $\{u_{i,b}^0\}$  and  $\{u_{i,b}^1\}$  to the outside committer who commits to one of these sequences in parallel.  $M$  forwards these messages to  $\mathcal{A}$ , and forwards the message given by  $\mathcal{A}$  corresponding to  $\text{nmcom}$  to the outside receiver.

- **Output phase:** After the first three rounds of the protocol are finished,  $\tilde{M}$  halts by outputting its view, denoted by  $\mathcal{V}_M$ . In particular,  $M$  does not continue further like  $H_3$ , it does not extract any values, and does not complete the fourth round. (Note that  $M$  cannot complete the fourth round, since it does not have the witness.) Formally, for  $i \in [\kappa], b \in \{0,1\}$ ,  $M$  outputs  $\mathcal{V}_M$  that includes the view of  $M$  from the *main execution phase* and the values  $\{\tilde{u}_{i,b}\}$  committed by  $M$  in the  $\kappa$  sessions on the right with tags  $\{\tilde{\text{id}}_i\}$  while receiving parallel commitments to  $\{u_{i,b}\}$  on left with tags  $\{\text{id}_i\}$ .

Let  $\{\tilde{u}_{i,b}^0\}$  (resp.,  $\{\tilde{u}_{i,b}^1\}$ ) be the sequence of values committed by  $M$  with  $\{\tilde{\text{id}}_i\}$  when it receives a commitment to  $\{u_{i,b}^0\}$  (resp.,  $\{u_{i,b}^1\}$ ) with  $\{\text{id}_i\}$ .

**Description of  $D_{\text{nm}}$ :** Distinguisher  $D_{\text{nm}}$  proceeds in the following two phases.

- **Main execution phase:**  $D_{\text{nm}}$  incorporates both  $M$  and  $D$  internally, it receives as input the view  $\mathcal{V}_M$  and proceeds as follows:

1.  $D_{\text{nm}}$  parses  $\mathcal{V}_M$  to obtain the view of  $\mathcal{V}_A$ , the committed values  $\{\tilde{u}_{i,b}\}$  and consequently a string  $\sigma$  corresponding to the “trapdoor witness.”
2.  $D_{\text{nm}}$  starts  $M$  and feeds him the view  $\mathcal{V}_A$  together with the committed values and continues the execution just like  $H_3$ . It, however, does not rewind  $\mathcal{A}$  (internal to  $M$ ), instead it uses  $\sigma$  and values in  $\mathcal{V}_A$  to complete the last round of the protocol.

- **Output phase:** When  $\mathcal{A}$  halts,  $D_{\text{nm}}$  feeds the view  $\mathcal{V}_A$  to  $D$  and outputs whatever  $D$  outputs.

It is straightforward to verify that if  $M$  receives commitments corresponding to  $\{u_{i,b}^0\}$  (resp.,  $\{u_{i,b}^1\}$ ) then the output of  $D_{\text{nm}}$  is identical to that of  $H_3$  (resp.,  $H_4$ ). The claim follows.  $\diamond$

$H_5$ : Identical to  $H_4$  except that  $H_5$  changes the “inputs of the oblivious transfer” from  $(Z_{i,0}, Z_{i,1})$  to  $(Z_{i,x_i^*}, Z_{i,x_i^*})$ . Formally, in the last round,  $H_5$  sets  $W_{i,b} = Z_{i,x_i^*} \oplus \text{H}((z'_{i,b}))$  for every  $(i,b)$ , but does everything else as  $H_4$ .

$H_4$  and  $H_5$  are computationally indistinguishable due to the (indistinguishable) security of oblivious transfer w.r.t. a malicious receiver. This part is identical to the proof in [KO04], and relies on the fact that one of the two strings for oblivious transfer are obtained by “coin tossing;” and therefore its inverse is hidden, which implies that the hardcore bits look pseudorandom.

$H_6$ : Identical to  $H_5$  except that now we simulate the garbled circuit and its labels for values  $x^*$  and  $F_1(x^*, y)$ .

Formally,  $H_6$  starts by proceeding exactly as  $H_5$  up to round 3 except that instead of committing to correct garbled circuit and labels in round 2, it simply commits to random values. After completing round 3,  $H_6$  extracts  $x^*$  exactly as in  $H_5$ . If extraction succeeds, it sends  $x^*$  to the trusted party, receives back  $v_1 = F_1(x^*, y)$ , and computes  $(\{Z_{i,b}\}, \text{GC}_*) \leftarrow \text{SimGC}(1^\kappa, F_1, x^*, v_1)$ . It uses labels  $\{Z_{i,x_i^*}\}$  to define the values  $\{W_{i,b}\}$  as in  $H_5$ , and equivocates  $\text{C}_{\text{gc}}$  to obtain openings corresponding to the simulated circuit  $\text{GC}^*$ . It then computes  $\text{fs}_4$  as before (by using the trapdoor witness  $(\sigma, \omega_\sigma)$ ), and constructs  $m_4 := (\{W_{i,b}\}, \text{fs}_4, \text{GC}^*, \zeta)$ . It feeds  $m_4$  to  $\mathcal{A}$  and finally outputs  $\mathcal{A}$ 's view and halts.

We claim that  $H_5$  and  $H_6$  are computationally indistinguishable. First observe that the joint distribution of values  $(\{\text{C}_{\text{lab}}^{i,b}\}, \text{C}_{\text{gc}})$  and  $\text{GC}_y$  (along with real openings) in  $H_5$  is indistinguishable from the joint distribution of the values  $(\{\text{C}_{\text{lab}}^{i,b}\}, \text{C}_{\text{gc}})$  and  $\text{GC}^*$  (along with equivocal openings) in  $H_6$ . The two hybrids are identical except for sampling of these values, and can be simulated perfectly given these values from outside. The claim follows.<sup>19</sup>

<sup>19</sup> Let us note that changing the commitment in second round (from correct garbled labels/circuit to random strings) is performed from the beginning—i.e., in the “main thread” of simulation—therefore the running time stays expected polynomial time as in claim 5.

Observe that  $H_6$  is now independent of the input  $y$ . Our simulator  $\mathcal{S}$  is  $H_6$ . This completes the proof.  $\square$

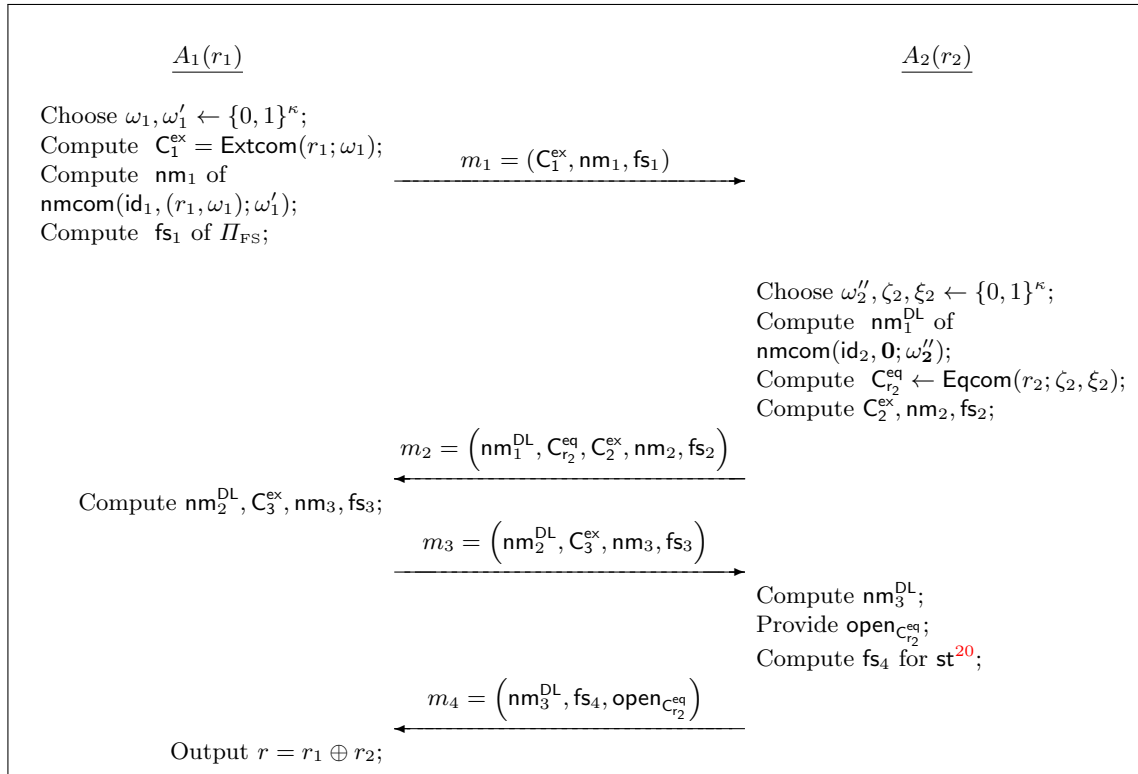
*Remark 2.* Note that robustness is sufficient but it is not necessary since what is actually required is a non-malleable commitment robust against a specific class of protocols, i.e., 3-round WIPOK protocols. The commitment scheme of [PPV08] is proven to be robust only against a 3-round WIPOK based on adaptive PRGs. In particular, the authors of [PPV08] prove the following lemma.

**Lemma 41** ([PPV08]) *There exists a 3-round WIPOK protocol  $\Pi_{\text{WIPOK}}$  for NP-statements and a 2-round non-malleable commitment scheme that is robust w.r.t. to  $\Pi_{\text{WIPOK}}$  assuming the existence of adaptive PRGs.*

## 5 Multi-Party Coin Flipping Protocol

In this section, we show a protocol for the multi-party coin-flipping functionality. Since we need neither OT nor garbled circuits for coin-flipping, this protocol is simpler than the two-party protocol.

At a high level, the protocol simply consists of each party “committing” to a random string  $r$ , which is opened in the last round along with a simulatable proof of correct opening given to *all* parties independently. The output consists of the  $\oplus$  of all strings. This actually does not work directly as stated, but with a few more components, such as equivocal commitment as well as extractable commitment to  $r$  for the proof to go through. A high level description of the protocol between two parties ( $A_1, A_2$ ) is given in Figure 5.



**Fig. 5.** High-level description of the left execution of  $\Pi_{\text{CF}}$ .

<sup>20</sup> Informally,  $\text{st}$  says that either the commitments  $C_{r_2}^{\text{eq}}$ ,  $\widetilde{\text{nm}}_1$  and  $\widetilde{C}_1^{\text{ex}}$  commit to the same value  $r_2$  or that  $\text{nm}_1^{\text{DL}}$  is a commitment to a trapdoor  $\sigma_1$  or  $\sigma_2$  for the proof provided by  $A_2$ .

**Protocol  $\Pi_{\text{MCF}}$ .** Let  $\mathcal{P} = \{P_1, \dots, P_n\}$  be the set of parties. Furthermore, denote by  $(\text{id}_1, \dots, \text{id}_n)$  the unique identities of parties  $\{P_1, \dots, P_n\}$ , respectively. Let  $\text{Extcom} = (\text{C}_1^{\text{ex}}, \text{C}_2^{\text{ex}}, \text{C}_3^{\text{ex}})$  be a three-round extractable commitment scheme [DDN91, PRS02, Ros04]. In a nutshell, in order to commit to a random string  $r$  the committer picks  $\{r_i^0\}_{i \in \kappa}, \{r_i^1\}_{i \in \kappa}$  such that  $\forall i, r_i^0 \oplus r_i^1 = r$ . Next, the receiver sends a random  $\kappa$ -bit string  $e = (e_1, \dots, e_\kappa)$  and then the Committer decommits to  $r_1^{e_1}, \dots, r_\kappa^{e_\kappa}$ .

Let us denote by  $F_{\text{MCF}} : \{0, 1\}^{\kappa \cdot n} \rightarrow \{0, 1\}^\kappa$  the function  $F_{\text{MCF}}(r_1, \dots, r_n) = \bigoplus_{i \in [n]} r_i$ . The protocol starts with each party  $P_i$  choosing a random string  $r_i$ . It consists of four rounds, i.e., all parties send messages in each round and the messages of all executions are seen by every party.

The protocol essentially consists of  $n$  simultaneous executions of a two-party coin-flipping protocol  $\Pi_{\text{CF}} = \langle A_1, A_2 \rangle$  between parties  $(P_i, P_j)$  where  $P_i$  acts as  $A_1$  with input  $r_i$  and  $P_j$  acts as  $A_2$  with input  $r_j$  (both are symmetric). This protocol is described below.

*Protocol  $\Pi_{\text{CF}} = \langle A_1, A_2 \rangle$ :* Let the input of  $A_1$  be  $r_1$ , and the input of  $A_2$  be  $r_2$ . Furthermore, let  $F_{2\text{CF}} = r_1 \oplus r_2$  be the evaluated function. The protocol has a structure similar to our previous protocol. The set of messages enabling  $A_1$  to learn the output are denoted by  $(m_1, m_2, m_3, m_4)$  where  $(m_1, m_3)$  are sent by  $A_1$  and  $(m_2, m_4)$  are sent by  $A_2$ . Likewise, the set of messages enabling  $A_2$  to learn the output are denoted by  $(\tilde{m}_1, \tilde{m}_2, \tilde{m}_3, \tilde{m}_4)$  where  $(\tilde{m}_1, \tilde{m}_3)$  are sent by  $A_2$  and  $(\tilde{m}_2, \tilde{m}_4)$  are sent by  $A_1$ . Therefore, messages  $(m_\ell, \tilde{m}_\ell)$  are simultaneously exchanged in the  $\ell$ -th round,  $\ell \in \{1, \dots, 4\}$ . We now describe how these messages are constructed in each round.

**Round 1.** We first describe how  $A_1$  constructs  $m_1$ .

1.  $A_1$  commits to  $r_1$  using  $\text{Extcom}$ . I.e., it uniformly chooses  $\omega_1$  and generates  $\text{C}_1^{\text{ex}}$  which is the first message corresponding to the execution of  $\text{Extcom}(r_1; \omega_1)$ .
2.  $A_1$  commits to the decommitment information of  $\text{C}_1^{\text{ex}}$  using  $\text{nmcom}$  with identity  $\text{id}_1$ . I.e., it uniformly chooses  $\omega'_1$  and generates  $\text{nm}_1$  which is the first message corresponding to the execution of  $\text{nmcom}(\text{id}_1, (r_1, \omega_1); \omega'_1)$ .
3.  $A_1$  prepares the first message  $\text{fs}_1$  of  $\Pi_{\text{FS}}$ .
4. Message  $m_1$  is defined to be  $(\text{C}_1^{\text{ex}}, \text{nm}_1, \text{fs}_1)$ .

Likewise,  $A_2$  performs the same actions as  $A_1$  to sample the values  $(r_2, \omega_2)$  and constructs  $\tilde{m}_1 := (\tilde{\text{C}}_1^{\text{ex}}, \tilde{\text{nm}}_1, \tilde{\text{fs}}_1)$ .

**Round 2.** In this round  $A_2$  sends message  $m_2$  and  $A_1$  sends  $\tilde{m}_2$ . We first describe how  $A_2$  constructs  $\tilde{m}_2$ .

1.  $A_2$  commits to a  $\kappa$ -bit zero-string  $\mathbf{0}$  using an input-delayed  $\text{nmcom}$  (where the committed value can be chosen in the last round of the  $\text{nmcom}$  protocol) with identity  $\text{id}_2$ . I.e., it uniformly chooses  $\omega''_2$  and generates  $\text{nm}_1^{\text{DL}}$  which is the first message corresponding to the execution of  $\text{nmcom}(\text{id}_2, \mathbf{0}; \omega''_2)$ . In an input-delayed non-malleable.
2.  $A_2$  prepares the second message  $\text{C}_2^{\text{ex}}$  of the  $\text{Extcom}$  protocol initiated by  $A_1$ .
3.  $A_2$  prepares the second message  $\text{nm}_2$  of the  $\text{nmcom}$  protocol initiated by  $A_1$ .
4.  $A_2$  generates an equivocal commitment to his input: i.e.,  $\text{C}_2^{\text{eq}} \leftarrow \text{Eqcom}(r_2; \zeta_2, \xi_2)$  where  $\text{Eqcom}(r_2; \zeta_2, \xi_2) \stackrel{\text{def}}{=} \text{com}(r_2; \zeta_2) \parallel \text{com}(r_2; \xi_2)$ .
5.  $A_2$  prepares the second message  $\text{fs}_2$  of the  $\Pi_{\text{FS}}$  protocol initiated by  $A_1$ .
6. Define message  $m_2 := (\text{nm}_1^{\text{DL}}, \text{C}_2^{\text{eq}}, \text{C}_2^{\text{ex}}, \text{nm}_2, \text{fs}_2)$ .

Likewise,  $A_1$  performs the same actions as  $A_2$  in the previous step to construct the message  $\tilde{m}_2 := (\tilde{\text{nm}}_1^{\text{DL}}, \tilde{\text{C}}_2^{\text{eq}}, \tilde{\text{C}}_2^{\text{ex}}, \tilde{\text{nm}}_2, \tilde{\text{fs}}_2)$  w.r.t. identity  $\text{id}_1$ .

**Round 3.** In this round  $A_1$  sends message  $m_3$  and  $A_2$  sends  $\tilde{m}_3$ .  $A_1$  prepares  $m_3$  as follows:

1.  $A_1$  generates the second messages  $\text{nm}_2^{\text{DL}}$  corresponding to the delayed-input  $\text{nmcom}$  execution initiated by  $A_2$  (with  $\text{id}_2$ ).
2.  $A_1$  prepares the final message  $\text{C}_3^{\text{ex}}$  of  $\text{Extcom}$  and  $\text{nm}_3$  of the  $\text{nmcom}$ .  $A_1$  also prepares the third message  $\text{fs}_3$  of  $\Pi_{\text{FS}}$ .
3. Define  $m_3 := (\text{nm}_2^{\text{DL}}, \text{C}_3^{\text{ex}}, \text{nm}_3, \text{fs}_3)$  to  $A_2$ .

Likewise,  $A_2$  performs the same actions as  $A_1$  in the previous step to construct the message  $\tilde{m}_3 := (\widetilde{\text{nm}}_2^{\text{DL}}, \widetilde{\text{C}}_3^{\text{ex}}, \widetilde{\text{nm}}_3, \widetilde{\text{fs}}_3)$  w.r.t. identity  $\text{id}_1$ .

**Round 4.** In this round  $A_2$  sends message  $m_4$  and  $A_1$  sends  $\tilde{m}_4$ .  $A_2$  constructs  $m_4$  as follows:

1. If  $\text{fs}_3$  is not accepting,  $A_2$  aborts. Otherwise,  $A_2$  decommits  $\text{C}_{r_2}^{\text{eq}}$  as  $r_2$ , denoted by  $\text{open}_{\text{C}_{r_2}^{\text{eq}}}$  to  $A_1$ .
2.  $A_2$  prepares the final message  $\text{nm}_3^{\text{DL}}$  of delayed-input  $\text{nmcom}$  with respect to his identity  $\text{id}_2$ .
3.  $A_2$  prepares the final message  $\text{fs}_4$  of  $\Pi_{\text{FS}}$  proving the statement  $\text{st}$  defined as follows:  
 $\exists r_2, \tilde{\omega}_2, \tilde{\omega}'_2, \omega''_2, \zeta_2, \xi_2, \sigma_1, \sigma_2$ <sup>21</sup> s.t.:

$$\left( \widetilde{\text{C}}_1^{\text{ex}} = \text{Extcom}(r_2; \tilde{\omega}_2) \wedge \widetilde{\text{nm}}_1 = \text{nmcom}_1(\text{id}_2, (r_2, \tilde{\omega}_2); \tilde{\omega}'_2) \wedge \text{C}_{r_2}^{\text{eq}} = \text{com}(r_2; \zeta_2) \parallel \text{com}(r_2; \xi_2) \right) \vee$$

$$(\text{nm}_1^{\text{DL}} = \text{nmcom}_1(\text{id}_2, \sigma_1; \omega''_2) \vee \text{nm}_1^{\text{DL}} = \text{nmcom}_1(\text{id}_2, \sigma_2; \omega''_2))$$

Informally,  $\text{st}$  says that either the commitments  $\text{C}_{r_2}^{\text{eq}}$  and  $\widetilde{\text{C}}_1^{\text{ex}}$  commit to the same value  $r_2$  and  $\widetilde{\text{nm}}_1$  commits to the valid decommitment of  $\widetilde{\text{C}}_1^{\text{ex}}$  or that  $\text{nm}_1^{\text{DL}}$  is a commitment to  $\sigma_1$  or  $\sigma_2$ .

4. Define  $m_4 := (\text{nm}_3^{\text{DL}}, \text{fs}_4, \text{open}_{\text{C}_{r_2}^{\text{eq}}})$ .

Likewise,  $A_1$  performs the same actions as  $A_2$  in the previous step to send the message  $\tilde{m}_4 := (\widetilde{\text{nm}}_3^{\text{DL}}, \text{fs}_4, \widetilde{\text{open}}_{\text{C}_{r_2}^{\text{eq}}})$ .

**Output computation of  $\Pi_{\text{MCF}}$ .** After the completion of the 4th round each party checks that all pairs of parties used the same inputs  $(r_1, r_2, \dots, r_n)$  in all executions and that they were all successful. If so, each party outputs  $r = F_{\text{MCF}}(r_1, r_2, \dots, r_n) = r_1 \oplus \dots \oplus r_n$ .

## 5.1 Proof of Security

The proof of security is very similar to the proof for the two-party case. We directly present the hybrids, and discuss the intuition when necessary.

**Theorem 2** *Assuming the existence of a trapdoor permutation family and  $k$ -round parallel, 3-robust and input-delayed non-malleable commitments, then the multi-party protocol  $\Pi_{\text{MCF}}$  securely computing the multi-party coin-flipping functionality with black-box simulation in the presence of a malicious adversary for polynomially many coins. The round complexity of  $\Pi_{\text{MCF}}$  is  $k' = \max(4, k + 1)$ .*

**Proof:** Let  $\mathcal{P} = \{P_1, \dots, P_n\}$  be the set of parties participating in the execution of  $\Pi_{\text{MCF}}$ . Also let  $\mathcal{P}^* \subseteq \mathcal{P}$  be the set of parties corrupted by the adversary  $\mathcal{A}$ . The simulator  $\mathcal{S}$  only generates messages on behalf of parties  $\mathcal{P} \setminus \mathcal{P}^*$ . In particular, we show that for every adversary  $\mathcal{A}$  there exists an “ideal” world adversary  $\mathcal{S}$  such that:

$$\{\text{IDEAL}_{F_{\text{MCF}}, \mathcal{S}}(\kappa, \cdot)\}_{\kappa} \stackrel{c}{\approx} \{\text{REAL}_{\Pi, \mathcal{A}}(\kappa, \cdot)\}_{\kappa}$$

We prove this claim by considering hybrid experiments  $H_0, H_1, \dots$  as described below. In the sequel, without loss of generality we will assume that party  $P_1$  (party  $A_1$  in the description of  $\Pi_{\text{CF}}$ ) is the only honest party since our protocol is secure against  $n - 1$  corruptions. We denote by  $\mathbf{Hybrid}_{F, \mathcal{S}}^i(\kappa, \cdot)$  the random variable that corresponds to the simulator’s output in hybrid execution  $H_i$  that includes the view of the adversary  $\mathcal{A}$  and the values committed to by  $\mathcal{A}$ . For simplicity of exposition, we abbreviate it to  $\text{Hybrid}_i$ .

$H_0$ : Identical to the real execution. More specifically,  $H_0$  starts the execution of  $\mathcal{A}$  providing it fresh randomness, and interacts with it honestly by performing all actions of  $P_1$  with uniform randomness  $r_1$ . The output consists of  $\mathcal{A}$ ’s view. By construction,  $H_0$  and the output of  $\mathcal{A}$  in the real execution are identically distributed.

Note that we are going to maintain the following invariant under each hybrid: the  $\text{nm}^{\text{DL}}$  messages generated by  $\mathcal{A}$  on behalf of the parties in  $\mathcal{P}^*$  cannot correspond to commitments to the simulation trapdoors of the  $\Pi_{\text{FS}}$  scheme.

<sup>21</sup> The message  $\text{fs}_1$  contains the commitments  $\text{nm}_1^{\sigma_1}, \text{nm}_1^{\sigma_2}$  to the trapdoor  $\sigma$  for the proof provided by  $A_2$  (see Section 2.2).



**Claim 4** *The probability that the adversary  $\mathcal{A}$  commits to the simulation trapdoors  $\{\tilde{\sigma}_i\}_{i \in [\mathcal{P}^*]}$  of the proof system  $\Pi_{\text{FS}}$  (between prover  $P_1$  and verifiers  $\mathcal{P}^*$ ) in the delayed-input non-malleable commitment initiated in the second round is negligible.*

*Proof.* Assume for contradiction that the adversary  $\mathcal{A}$  commits to the simulation trapdoors in the delayed-input non-malleable commitment  $\text{nm}^{\text{DL}}$  with non-negligible probability. That said, there are at least two string values, denoted by  $\tilde{\sigma}_i^1$  and  $\tilde{\sigma}_i^2$ , that the adversary commits to in  $\text{nm}^{\text{DL}}$  with non-negligible probability. We rely on the non-malleability of  $\text{nmcom}$  to prove this claim and show how this violates the non-malleability property of  $\text{nmcom}$ .

The high level idea is as follows: we define a man-in-the-middle  $M$  which incorporates  $\mathcal{A}$  and receives a non-malleable commitment to one of the values  $\tilde{\sigma}_i^1$  and  $\tilde{\sigma}_i^2$ . Then we define a distinguisher  $D_{\text{nm}}$  which incorporates  $M$ , takes as input the value committed by  $M$  and its view, and can distinguish which value was committed to  $M$ . This violates non-malleability of  $\text{nmcom}$ .

Formally, we define a man-in-middle adversary  $M$  who receives a commitment on the left and makes a commitment on the right.

**Description of  $M$ :** Machine  $M$  proceeds in the following two phases.

- **Main execution phase:**  $M$  incorporates  $\mathcal{A}$  internally, and proceeds as follows:
  1.  $M$  proceeds exactly as  $H_0$  by sampling all messages internally except for the messages of  $\text{nmcom}$  corresponding to  $P_i^*$  for the messages  $\text{nm}^{\text{DL}}$ . In particular, messages  $\text{nm}^{\text{DL}}$  are received from an outside committer as follows.
  2.  $M$  forwards  $\tilde{\sigma}_i^1$  and  $\tilde{\sigma}_i^2$  to the outside committer who commits to one of these two values.  $M$  forwards these messages to  $\mathcal{A}$ , and forwards the message given by  $\mathcal{A}$  corresponding to  $\text{nmcom}$  to the outside receiver.
- **Output phase:**  $M$  halts by outputting its view, denoted by  $\mathcal{V}_M$ . Formally,  $M$  outputs  $\mathcal{V}_M$  that includes the view of  $M$  from the *main execution phase* and the value  $\tilde{u}$  committed by  $M$  on the right with tag  $\tilde{\text{id}}_k$  while receiving parallel commitments to  $u$  on left with tag  $\text{id}_k$ .

Let  $\tilde{u}^0$  (resp.,  $\tilde{u}^1$ ) be the values committed by  $M$  with  $\tilde{\text{id}}_k$  when it receives a commitment to  $u^0$  (resp.,  $u^1$ ) with  $\text{id}_k$ .

**Description of  $D_{\text{nm}}$ :** Distinguisher  $D_{\text{nm}}$  proceeds in the following two phases.

- **Main execution phase:**  $D_{\text{nm}}$  incorporates both  $M$  internally, it receives as input the view  $\mathcal{V}_M$  and proceeds as follows:
  1.  $D_{\text{nm}}$  parses  $\mathcal{V}_M$  to obtain the view of  $\mathcal{V}_A$ , the committed values  $\{(r_i^*, w_i^*)\}_{i \in [\mathcal{P}^*]}$  and the “simulation trapdoors”  $\{\tilde{\sigma}_i\}_{i \in [\mathcal{P}^*]}$  of each  $\Pi_{\text{FS}}$ .
  2.  $D_{\text{nm}}$  starts  $M$  and feeds him the view  $\mathcal{V}_A$  together with the committed values and continues the execution just like  $H_0$  till  $\mathcal{A}$  halts.

It is straightforward to verify that if  $M$  receives commitments corresponding to  $\tilde{\sigma}_i^1$  (resp.,  $\tilde{\sigma}_i^2$ ) then the output of  $D_{\text{nm}}$  is distinguishable. The claim follows.  $\diamond$

$H_1$ : Identical to  $H_0$  except that this hybrid also performs extraction of  $\mathcal{A}$ ’s implicit inputs  $\{r_i^*\}_{i \in [\mathcal{P}^*]}$  from the parallel executions of  $\text{Extcom}$ ; in addition, it also extracts the “simulation trapdoors”  $\{\tilde{\sigma}_i\}_{i \in [\mathcal{P}^*]}$  from the first three rounds of each  $\Pi_{\text{FS}}$ . More specifically,  $H_1$  proceeds as follows:

1. It completes the first three simultaneous message exchange rounds exactly as in  $H_0$ , and waits until  $\mathcal{A}$  either aborts or successfully completes the third round.
2. At this point,  $H_1$  proceeds to extract the witness corresponding to each proof-of-knowledge completed in the first three rounds while the inputs  $\{r_i\}_{\mathcal{P}^*}$  are as in  $H_0$ .

Specifically,  $H_1$  defines a cheating prover  $P^*$  for each corrupted party  $P_i$  which acts identically to  $H_0$ , simulating all messages for  $\mathcal{A}$ , except those corresponding to (each execution of)  $\text{Extcom}$ . It then proceeds to a rewinding phase. In this phase, it repeatedly rewinds the adversary and samples new messages for the  $\Pi_{\text{FS}}$  and  $\text{nmcom}$  protocols using fresh randomness. Specifically,  $\mathcal{S}^1$  continues rewinding  $\mathcal{A}$  until it successfully extracts the committed values  $r_i^*$  and a value  $(\tilde{\sigma}_i, \omega_{\tilde{\sigma}_i})$  which is the simulation trapdoor for  $\Pi_{\text{FS}}$ .

If extraction fails,  $H_1$  outputs fail.

3.  $H_1$  now completes the final round and prepares the output exactly as  $H_0$ .

**Claim 5**  $\text{Hybrid}_0 \approx_s \text{Hybrid}_1$ .

Hybrid  $H_0$  and  $H_1$  are statistically close and extraction happens in expected polynomial time (see Claim 1). Furthermore, the  $\text{nm}^{\text{DL}}$  messages generated by  $\mathcal{A}$  cannot correspond to commitments to the simulation trapdoors of  $\Pi_{\text{FS}}$  since  $H_0$  and  $H_1$  are statistically close.

$H_2$ : Identical to  $H_1$  except that  $\mathcal{S}_2$  commits to the simulation trapdoor  $(\tilde{\sigma}_1, \omega_{\tilde{\sigma}_1})$  in  $\text{nm}^{\text{DL}}$  created in the second round on behalf of the honest party  $P_1$ . Note that due to the scheduling of  $\Pi_{\text{FS}}$  and  $\text{Extcom}$  rewinding does not cause an issue since all their values are already fixed in the first round. Hybrids  $H_2$  and  $H_1$  are indistinguishable due to the non-malleability property of the  $\text{nmcom}$  scheme.

**Claim 6**  $\text{Hybrid}_1 \approx_c \text{Hybrid}_2$ .

*Proof.* We rely on the non-malleability of  $\text{nmcom}$  to prove this claim. Assume, for contradiction, that there exists a distinguisher  $D$  that distinguishes  $H_1$  and  $H_2$ . We show how this violates the non-malleability property of  $\text{nmcom}$ .

The high level idea is as follows: first we define  $u^0$  and  $u^1$  and a man-in-the-middle  $M$  which incorporates  $\mathcal{A}$  and receives a non-malleable commitment to one of these two values. Then we define a distinguisher  $D_{\text{nm}}$  which incorporates both  $M$  and  $D$ , takes as input the value committed by  $M$  and its view, and can distinguish which value was committed to  $M$ . This violates non-malleability of  $\text{nmcom}$ .

Formally, we define a man-in-middle adversary  $M$  who receives a commitment on the left and makes a commitment on the right.

**Description of  $M$ :** Machine  $M$  proceeds in the following two phases.

– **Main execution phase:**  $M$  incorporates  $\mathcal{A}$  internally, and proceeds as follows:

1.  $M$  proceeds exactly as  $H_1$  by sampling all messages internally except for the messages of  $\text{nmcom}$  corresponding to  $\mathcal{P}^*$  for the messages  $\text{nm}^{\text{DL}}$ . In particular,  $M$  completes the rewinding as in  $H_1$ , extracts the trapdoors  $\{\tilde{\sigma}_i\}_{i \in [\mathcal{P}^*]}$  and then to change the committed value it suffices to change the fourth round of the protocol because the non malleable commitment is a 3-round input-delayed protocol that completes in the fourth round. Messages  $\text{nm}^{\text{DL}}$  are received from an outside committer as follows.
2.  $M$  samples a uniformly random value  $u^0$  and forwards  $u^0$  and  $u^1$  where wlog  $u^1 = \tilde{\sigma}_i$  to the outside committer who commits to one of these two values.  $M$  forwards these messages to  $\mathcal{A}$ , and forwards the message given by  $\mathcal{A}$  corresponding to  $\text{nmcom}$  to the outside receiver.
3.  $M$  also completes the fourth round using the simulation trapdoors  $\{\tilde{\sigma}_i\}_{i \in [\mathcal{P}^*]}$ .

– **Output phase:**  $M$  halts by outputting its view, denoted by  $\mathcal{V}_M$ . Formally,  $M$  outputs  $\mathcal{V}_M$  that includes the view of  $M$  from the *main execution phase* and the value  $\tilde{u}$  committed by  $M$  on the right with tag  $\tilde{\text{id}}_k$  while receiving parallel commitments to  $u$  on left with tag  $\text{id}_k$ .

Let  $\tilde{u}^0$  (resp.,  $\tilde{u}^1$ ) be the values committed by  $M$  with  $\text{id}_k$  when it receives a commitment to  $u^0$  (resp.,  $u^1$ ) with  $\text{id}_k$ .

**Description of  $D_{\text{nm}}$ :** Distinguisher  $D_{\text{nm}}$  proceeds in the following two phases.

– **Main execution phase:**  $D_{\text{nm}}$  incorporates both  $M$  and  $D$  internally, it receives as input the view  $\mathcal{V}_M$  and proceeds as follows:

1.  $D_{\text{nm}}$  parses  $\mathcal{V}_M$  to obtain the view of  $\mathcal{V}_A$ , the committed values  $\{r_i^*\}_{i \in [\mathcal{P}^*]}$  and the “simulation trapdoors”  $\{\tilde{\sigma}_i\}_{i \in [\mathcal{P}^*]}$  of each  $\Pi_{\text{FS}}$ .
2.  $D_{\text{nm}}$  starts  $M$  and feeds him the view  $\mathcal{V}_A$  together with the committed values and continues the execution just like  $H_1$ . It, however, does not rewind  $\mathcal{A}$  (internal to  $M$ ), instead it uses  $\tilde{\sigma}_i$  and values in  $\mathcal{V}_A$  that include the values  $\{r_i^*\}_{i \in [\mathcal{P}^*]}$  to complete the last round of the protocol. Since the commitment is input-delayed the trapdoor  $\tilde{\sigma}_i$  is committed in the last round.

– **Output phase:** When  $\mathcal{A}$  halts,  $D_{\text{nm}}$  feeds the view  $\mathcal{V}_A$  to  $D$  and outputs whatever  $D$  outputs.

It is straightforward to verify that if  $M$  receives commitments corresponding to  $u^0$  (resp.,  $u^1$ ) then the output of  $D_{\text{nm}}$  is identical to that of  $H_1$  (resp.,  $H_2$ ). The claim follows.  $\diamond$

$H_3$ : This hybrid is identical to  $H_2$  except that for all  $P_i \in \mathcal{P}^*$  this hybrid uses each simulation trapdoor  $(\tilde{\sigma}_i, \omega_{\tilde{\sigma}_i})$  as the witness to compute  $\text{fs}_4$  in the last round. (Recall that  $\text{fs}_4$  is the last round of an execution of  $\Pi_{\text{WIPOK}}$ .) Furthermore, the adversary can never succeeds in committing to the simulation trapdoor of the  $\Pi_{\text{FS}}$  scheme.

**Claim 7**  $\text{Hybrid}_2 \approx_c \text{Hybrid}_3$ .

The proof follows due to the 3-robustness property w.r.t. the 3-round  $\Pi_{\text{WIPOK}}$  protocol of  $\Pi_{\text{FS}}$  (see Claim 2).

$H_4$ : This hybrid is identical to  $H_3$  except that we change the equivocal commitment in the second round to its equivocation mode. More specifically,  $H_4$  starts by proceeding exactly as  $H_3$  except that instead of committing to the correct randomness in round 2, it simply commits to it in an equivocal way. That said, the simulator generates  $C_{r_1}^{\text{eq}} = \text{com}(r_1; \zeta_1) || \text{com}(\bar{r}_1; \xi_1)$ .

**Claim 8**  $\text{Hybrid}_3 \approx_c \text{Hybrid}_4$ .

Indistinguishability follows due to the 3-robustness property (1-robustness is sufficient) w.r.t. the equivocal commitment in a similar fashion as in Claim 2 where instead of forwarding the WIPOK messages on the left, the messages of the equivocal commitment are forwarded to the adversary.

$H_5$ : In this hybrid, we change the committed value to a dummy one used in Extcom on behalf of the honest party.

**Claim 9**  $\text{Hybrid}_4 \approx_c \text{Hybrid}_5$ .

The outputs of  $H_4$  and  $H_5$  are indistinguishable due to the 3-robustness property w.r.t. the extractable commitment as in Claim 2 where instead of forwarding the WIPOK messages on the left, the messages of Extcom are forwarded to the adversary.

$H_6$ : In this hybrid, we get rid of  $P_1$ 's input  $r_1$  used in nm initiated in the first round. Formally,  $H_6$  is identical to  $H_5$  except that in the first round  $P_1$  commits to the zero string.

**Claim 10**  $\text{Hybrid}_5 \approx_c \text{Hybrid}_6$ .

Indistinguishability follows due to the non-malleability property of nmcom as in Claim 6 except that both values  $u^0, u^1$  are chosen uniformly at random.

$H_7$ : Identical to  $H_6$  except that we change the opening of the equivocal commitment in the fourth round. More specifically,  $H_7$  starts by proceeding exactly as  $H_6$  up to round 3. After completing round 3,  $H_7$  extracts the simulation trapdoors and values  $\{r_i^*\}_{i \in [\mathcal{P}^]}$  by rewinding. If extraction succeeds, it receives the output  $r^*$  from the trusted party. Then, it computes  $r_1 = r^* \oplus_{i \in [\mathcal{P}^]} r_i^*$ . Next, it equivocates  $C_{r_1}^{\text{eq}}$  to obtain openings corresponding to the simulated output. It then computes each  $\text{fs}_4$  message as before (by using the trapdoor witness  $(\tilde{\sigma}_i, \omega_{\tilde{\sigma}_i})$ ) and constructs  $m_4 := (\text{nm}_3^{\text{DL}}, \text{fs}_4, \text{open}_{C_{r_1}^{\text{eq}}})$ . It feeds  $m_4$  to  $\mathcal{A}$  and finally outputs  $\mathcal{A}$ 's view and halts. Note that verification of st still succeeds due to the fact that  $\mathcal{S}_4$  has committed of behalf of  $P_1$  in the  $\text{nm}^{\text{DL}}$  message to the simulation trapdoor.

Note that the adversary  $\mathcal{A}$  either aborts in the fourth round of the protocol or it opens the equivocal commitments to the extracted values  $\{r_i^*\}_{i \in [\mathcal{P}^]}$  except with negligible probability. Therefore, it remains to prove that  $\mathcal{A}$  can only open the equivocal commitments to  $\{r_i^*\}_{i \in [\mathcal{P}^]}$ . If  $\mathcal{A}$  does not open to  $\{r_i^*\}_{i \in [\mathcal{P}^]}$ , then it must either use a fake witness (trapdoor witness) or it must prove a false statement. Since we have shown that  $\mathcal{A}$  cannot commit to the trapdoors in  $\text{nm}^{\text{DL}}$ , it has to be that  $\mathcal{A}$  proves a false statement. In the following, we show that  $\mathcal{A}$  cannot prove a false statement.

**Claim 11** *The adversary does not prove a false statement in the fourth round.*

Assume for contradiction that the adversary does prove a false statement in the fourth round. We define a cheating prover that incorporates  $\mathcal{A}$  internally executing the simulation as in  $H_6$  but it forwards the  $\Pi_{\text{WIPOK}}$  messages in the second, third and fourth rounds of the  $\Pi_{\text{FS}}$  proof system outside. This means that  $H_7$  rewinds, extracts the trapdoors, fakes the  $\Pi_{\text{FS}}$  proof in the fourth round and only after performing the rewinding it forwards the  $\Pi_{\text{WIPOK}}$  messages of  $\Pi_{\text{FS}}$  outside to the adversary. Next, it follows that it can distinguish the committed values. This violates the soundness of  $\Pi_{\text{WIPOK}}$  inside the  $\Pi_{\text{FS}}$  proof system.

Our simulator  $\mathcal{S}$  is  $H_7$ . This completes the proof. □

## 5.2 Coin Flipping with Committed Inputs

We now discuss an extension of the coin-flipping functionality which will be useful in the next section. The extension considers a functionality which, in addition to providing a random string to the parties, also “attests” to a commitment to their input.

More specifically, we consider the following setting. Each party  $P_i$  has an input string  $x_i$  and randomness  $\rho_i$ . Let  $\text{com}$  be a non-interactive perfectly-binding commitment scheme. The Coin Flipping with Committed Inputs functionality  $\mathcal{F}_{\text{CF-CI}}$  acts as follows:

1. Each party sends  $(x_i, \rho_i, c_i)$  to the functionality where  $c_i = \text{com}(x_i; \rho_i)$ .
2. Functionality samples a random string  $r$ .
3. Functionality tests that for every  $i$ ,  $c_i = \text{com}(x_i; \rho_i)$ . If the test succeeds, it sets  $y_i = (r, c_i, \text{true})$ ; otherwise,  $y_i = (r, c_i, \text{false})$ .
4. Functionality sends  $(y_1, \dots, y_n)$  to all parties.

We claim that a minor modification of our coin-flipping protocol can actually implement  $\mathcal{F}_{\text{CF-CI}}$ . The modification is as follows:

1. In the protocol  $\Pi_{\text{MCF}}$ , view the input  $r_i$  of every  $P_i$  as two parts:  $r_i = r_i^1 || r_i^2$ .
2. In the second round of the protocol, in addition to sending the equivocal commitment, party  $P_i$  also sends  $c_i = \text{com}(x_i \oplus r_i^2; \rho_i)$  where  $\rho_i$  is a random string.
3. The proof system  $\Pi_{\text{FS}}$  now actually proves a modified statement, which in addition to all conditions as before, also includes the condition that: “there exist  $(x_i, \rho_i)$  such that  $c_i = \text{com}(x_i \oplus r_i^2; \rho_i)$ ” where note that  $r_i^2$  is already part of the opened value  $r_i$ .

We call the new protocol  $\Pi_{\text{CF-CI}}$ . The proof that  $\Pi_{\text{CF-CI}}$  implements  $\mathcal{F}_{\text{CF-CI}}$  is identical to the proof for  $\Pi_{\text{MCF}}$  with minor modifications.

## 5.3 Results for General Multi-Party Functionalities

We now discuss how to obtain protocols for general, as opposed to coin-flipping, functionalities in the multiparty case.

Mukherjee and Wichs [MW15] construct a 2-round protocol for general multiparty functionalities under the Learning With Errors (LWE) assumption in the CRS model. Combining their protocol with  $\Pi_{\text{MCF}}$  (to obtain the CRS), we obtain a protocol for general functionalities with  $k' + 2$  rounds under the LWE assumption.

Likewise, Garg et al. [GGHR14a] also construct a 2-round protocol for the same task in the CRS model, under the assumption that general purpose indistinguishability obfuscation exists. Their protocol actually has a special structure: it can be computed in just one round given access to the  $\mathcal{F}_{\text{CF-CI}}$  functionality that we have defined above. Consequently, using their protocol with protocol  $\Pi_{\text{CF-CI}}$  actually gives a  $k' + 1$  round protocol.

We thus get the following theorem.

**Theorem 3** *Assuming the existence of a trapdoor permutation family and  $k$ -round parallel, 3-robust and input-delayed non-malleable commitment schemes, there exists a protocol for securely computing every multiparty functionality such that: (a) the protocol has  $k' + 1$  rounds assuming general purpose indistinguishability obfuscation, and (b)  $k' + 2$  rounds assuming the LWE assumption where  $k' = \max(4, k + 1)$ .*

As a corollary of the above theorem, an instantiation of the above protocols with the  $\text{nmcom}$  scheme in [PPV08] gives a **five** round protocol (assuming indistinguishability obfuscation), and a **six** round protocol (assuming LWE) for general multiparty functionalities.

We note that we can also use the four round protocol of [COSV17] for  $\text{nmcom}$  which is concurrently (and hence parallel) non-malleable;<sup>22</sup> this will result in one extra round and give 7 rounds under LWE, and 6 under indistinguishability obfuscation.

<sup>22</sup> The 4-round protocol of [GRRV14] is claimed to be only standalone non-malleable (according to its latest ePrint version).

**Acknowledgements.** We thank Carmit Hazay and Muthuramakrishnan Venkitasubramaniam for pointing out a missing hybrid in the previous proof of Theorem 1.

## References

- Bar01. Boaz Barak. How to go beyond the black-box simulation barrier. In *42nd Annual Symposium on Foundations of Computer Science*, pages 106–115. IEEE Computer Society Press, October 2001.
- Bar02. Boaz Barak. Constant-round coin-tossing with a man in the middle or realizing the shared random string model. In *43rd Annual Symposium on Foundations of Computer Science*, pages 345–355. IEEE Computer Society Press, November 2002.
- BGW88. Michael Or Ben, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10. ACM Press, May 1988.
- BL02. Boaz Barak and Yehuda Lindell. Strict polynomial-time in simulation and extraction. In *34th Annual ACM Symposium on Theory of Computing*, pages 484–493. ACM Press, May 2002.
- BL04. Boaz Barak and Yehuda Lindell. Strict polynomial-time in simulation and extraction. *SIAM J. Comput.*, 33(4):738–818, 2004.
- 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. ACM Press, May 1990.
- CCD88. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 11–19. ACM Press, May 1988.
- CLOS02. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing*, pages 494–503. ACM Press, May 2002.
- COSV16. Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Concurrent non-malleable commitments (and more) in 3 rounds. *Lecture Notes in Computer Science*, pages 270–299. Springer, Heidelberg, August 2016.
- COSV17. Michele Ciampi, Rafail Ostrovsky, Luisa Siniscalchi, and Ivan Visconti. Four-Round Concurrent Non-Malleable Commitments from One-Way Functions. In *CRYPTO, 2017*.
- DDN91. Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography (extended abstract). In *23rd Annual ACM Symposium on Theory of Computing*, pages 542–552. ACM Press, May 1991.
- DI05. Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 378–394. Springer, Heidelberg, August 2005.
- DI06. Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 501–520. Springer, Heidelberg, August 2006.
- DNPR16. Ivan Damgård, Jesper Buus Nielsen, Antigoni Polychroniadou, and Michael Raskin. On the communication required for unconditionally secure multiplication. In *Advances in Cryptology – CRYPTO 2016, Part II*, *Lecture Notes in Computer Science*, pages 459–488. Springer, Heidelberg, August 2016.
- Fei90. Uriel Feige. *Alternative Models for Zero Knowledge Interactive Proofs*. PhD thesis, 1990.
- FLS99. Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM J. Comput.*, 29(1):1–28, 1999.
- FS90. Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd Annual ACM Symposium on Theory of Computing*, pages 416–426. ACM Press, May 1990.
- GGH<sup>+</sup>13. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Foundations of Computer Science*, pages 40–49. IEEE Computer Society Press, October 2013.
- GGHR14a. Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, pages 74–94, 2014.
- GGHR14b. Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 74–94. Springer, Heidelberg, February 2014.

- GLOV12. Vipul Goyal, Chen-Kuei Lee, Rafail Ostrovsky, and Ivan Visconti. Constructing non-malleable commitments: A black-box approach. In *53rd Annual Symposium on Foundations of Computer Science*, pages 51–60. IEEE Computer Society Press, October 2012.
- GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.
- Gol03. Oded Goldreich. Draft of a chapter on cryptographic protocols., <http://www.wisdom.weizmann.ac.il/oded/foc-vol2.html>, June 2003.
- Gol04. Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.
- 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. ACM Press, June 2011.
- GPR15. Vipul Goyal, Omkant Pandey, and Silas Richelson. Textbook non-malleable commitments. Manuscript, November 2015.
- GRRV14. Vipul Goyal, Silas Richelson, Alon Rosen, and Margarita Vald. An algebraic approach to non-malleability. In *55th Annual Symposium on Foundations of Computer Science*, pages 41–50. IEEE Computer Society Press, October 2014.
- 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. Springer, Heidelberg, August 2008.
- KO04. Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, pages 335–354, 2004.
- KOS03. Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Round efficiency of multi-party computation with a dishonest majority. In *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, pages 578–595, 2003.
- Lin01. Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 171–189. Springer, Heidelberg, August 2001.
- LP09a. Huijia Lin and Rafael Pass. Non-malleability amplification. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 189–198. ACM Press, May / June 2009.
- LP09b. Yehuda Lindell and Benny Pinkas. A proof of security of Yao’s protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.
- LP11a. Huijia Lin and Rafael Pass. Constant-round non-malleable commitments from any one-way function. In Lance Fortnow and Salil P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 705–714. ACM Press, June 2011.
- LP11b. Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 329–346. Springer, Heidelberg, March 2011.
- LPTV10. Huijia Lin, Rafael Pass, Wei-Lung Dustin Tseng, and Muthuramakrishnan Venkatasubramanian. Concurrent non-malleable zero knowledge proofs. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 429–446. Springer, Heidelberg, August 2010.
- LPV09. Huijia Lin, Rafael Pass, and Muthuramakrishnan Venkatasubramanian. A unified framework for concurrent security: universal composability from stand-alone non-malleability. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 179–188. ACM Press, May / June 2009.
- MW15. Pratyay Mukherjee and Daniel Wichs. Two round MPC from LWE via multi-key FHE. *IACR Cryptology ePrint Archive*, 2015:345, 2015.
- ORS15. Rafail Ostrovsky, Silas Richelson, and Alessandra Scafuro. Round-optimal black-box two-party computation. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 339–358, 2015.
- 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. ACM Press, June 2004.
- Pol17. Antigoni Polychroniadou. *On the Communication and Round Complexity of Secure Computation*. PhD thesis, 2017.

- PPV08. Omkant Pandey, Rafael Pass, and Vinod Vaikuntanathan. Adaptive one-way functions and applications. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 57–74. Springer, Heidelberg, August 2008.
- PRS02. Manoj Prabhakaran, Alon Rosen, and Amit Sahai. Concurrent zero knowledge with logarithmic round-complexity. In *43rd Annual Symposium on Foundations of Computer Science*, pages 366–375. IEEE Computer Society Press, November 2002.
- Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93. ACM Press, May 2005.
- Ros04. Alon Rosen. A note on constant-round zero-knowledge proofs for NP. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 191–202. Springer, Heidelberg, February 2004.
- Wee10. Hoeteck Wee. Black-box, round-efficient secure computation via non-malleability amplification. In *51st Annual Symposium on Foundations of Computer Science*, pages 531–540. IEEE Computer Society Press, October 2010.
- Yao82. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164. IEEE Computer Society Press, November 1982.
- Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

## A Secure Computation Definitions

For completeness, we recall the definition of secure computation based on [Gol04, Chapter 7] here. We only recall the two party case as it is most relevant to our proofs. The description naturally extends to multi-party case as well (details can be found in [Gol04]).

**Two-party computation.** A two-party protocol problem is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a functionality and denote it  $F : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^* \times \{0, 1\}^*$  where  $F = (F_1, F_2)$ . That is, for every pair of inputs  $(x, y)$ , the output-pair is a random variable  $(F_1(x, y), F_2(x, y))$  ranging over pairs of strings. The first party (with input  $x$ ) wishes to obtain  $F_1(x, y)$  and the second party (with input  $y$ ) wishes to obtain  $F_2(x, y)$ .

**Adversarial behavior.** Loosely speaking, the aim of a secure two-party protocol is to protect an honest party against dishonest behavior by the other party. In this paper, we consider malicious adversaries who may arbitrarily deviate from the specified protocol. When considering malicious adversaries, there are certain undesirable actions that cannot be prevented. Specifically, a party may refuse to participate in the protocol, may substitute its local input (and use instead a different input) and may abort the protocol prematurely. One ramification of the adversary’s ability to abort, is that it is impossible to achieve *fairness*. That is, the adversary may obtain its output while the honest party does not. In this work we consider a static corruption model, where one of the parties is adversarial and the other is honest, and this is fixed before the execution begins.

**Communication channel.** In our results we consider a secure simultaneous message exchange channel in which all parties can simultaneously send messages over the channel at the same communication round. Moreover, we assume an asynchronous network<sup>23</sup> where the communication is open (i.e. all the communication between the parties is seen by the adversary) and delivery of messages is not guaranteed. For simplicity, we assume that the delivered messages are authenticated. This can be achieved using standard methods.

<sup>23</sup> The fact that the network is asynchronous means that the messages are not necessarily delivered in the order which they are sent.

**Security of protocols (informal).** The security of a protocol is analyzed by comparing what an adversary can do in the protocol to what it can do in an ideal scenario that is secure by definition. This is formalized by considering an ideal computation involving an incorruptible trusted third party to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output. Loosely speaking, a protocol is secure if any adversary interacting in the real protocol (where no trusted third party exists) can do no more harm than if it was involved in the above-described ideal computation.

**Execution in the ideal model.** As we have mentioned, some malicious behavior cannot be prevented (for example, early aborting). This behavior is therefore incorporated into the ideal model. An ideal execution proceeds as follows:

**Inputs:** Each party obtains an input, denoted  $w$  ( $w = x$  for  $P_1$ , and  $w = y$  for  $P_2$ ).

**Send inputs to trusted party:** An honest party always sends  $w$  to the trusted party. A malicious party may, depending on  $w$ , either abort or send some  $w' \in \{0, 1\}^{|w|}$  to the trusted party.

**Trusted party answers first party:** In case it has obtained an input pair  $(x, y)$ , the trusted party first replies to the first party with  $F_1(x, y)$ . Otherwise (i.e., in case it receives only one valid input), the trusted party replies to both parties with a special symbol  $\perp$ .

**Trusted party answers second party:** In case the first party is malicious it may, depending on its input and the trusted party's answer, decide to stop the trusted party by sending it  $\perp$  after receiving its output. In this case the trusted party sends  $\perp$  to the second party. Otherwise (i.e., if not stopped), the trusted party sends  $F_2(x, y)$  to the second party.

**Outputs:** An honest party always outputs the message it has obtained from the trusted party. A malicious party may output an arbitrary (probabilistic polynomial-time computable) function of its initial input and the message obtained from the trusted party.

Let  $F : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^* \times \{0, 1\}^*$  be a functionality where  $F = (F_1, F_2)$  and let  $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$  be a pair of non-uniform probabilistic expected polynomial-time machines (representing parties in the ideal model). Such a pair is *admissible* if for at least one  $i \in \{1, 2\}$  we have that  $\mathcal{S}_i$  is honest (i.e., follows the honest party instructions in the above-described ideal execution). Then, the *joint execution of  $F$  under  $\mathcal{S}$  in the ideal model* (on input pair  $(x, y)$  and security parameter  $\kappa$ ), denoted  $\text{IDEAL}_{F, \mathcal{S}}(\kappa, x, y)$  is defined as the output pair of  $\mathcal{S}_1$  and  $\mathcal{S}_2$  from the above ideal execution.

**Execution in the real model.** We next consider the real model in which a real (two-party) protocol is executed (and there exists no trusted third party). In this case, a malicious party may follow an arbitrary feasible strategy; that is, any strategy implementable by non-uniform probabilistic polynomial-time machines. In particular, the malicious party may abort the execution at any point in time (and when this happens prematurely, the other party is left with no output). Let  $F$  be as above and let  $\Pi$  be a two-party protocol for computing  $F$ . Furthermore, let  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$  be a pair of non-uniform probabilistic polynomial-time machines (representing parties in the real model). Such a pair is *admissible* if for at least one  $i \in \{1, 2\}$  we have that  $\mathcal{A}_i$  is honest (i.e., follows the strategy specified by  $\Pi$ ). Then, the *joint execution of  $\Pi$  under  $\mathcal{A}$  in the real model*, denoted  $\text{REAL}_{\Pi, \mathcal{A}}(\kappa, x, y)$ , is defined as the output pair of  $\mathcal{A}_1$  and  $\mathcal{A}_2$  resulting from the protocol interaction.

**Security as emulation of a real execution in the ideal model.** Having defined the ideal and real models, we can now define security of protocols. Loosely speaking, the definition asserts that a secure two-party protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that admissible pairs in the ideal model are able to simulate admissible pairs in an execution of a secure real-model protocol.

**Definition 4 (secure two-party computation)** *Let  $F$  and  $\Pi$  be as above. Protocol  $\Pi$  is said to securely compute  $F$  (in the malicious model) if for every pair of admissible non-uniform probabilistic polynomial-time*



machines  $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$  for the real model, there exists a pair of admissible non-uniform probabilistic expected polynomial-time machines  $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$  for the ideal model, such that:

$$\{\text{IDEAL}_{F,\mathcal{S}}(\kappa, x, y)\}_{\kappa \in \mathbb{N}, x, y \text{ s.t. } |x|=|y|} \stackrel{c}{\approx} \{\text{REAL}_{\Pi, \mathcal{A}}(\kappa, x, y)\}_{\kappa \in \mathbb{N}, x, y \text{ s.t. } |x|=|y|}$$

We note that the above definition assumes that the parties know the input lengths (this can be seen from the requirement that  $|x| = |y|$ ). Some restriction on the input lengths is unavoidable, see [Gol04, Section 7.1] for discussion. We also note that we allow the ideal adversary/simulator to run in expected (rather than strict) polynomial-time. This is essential for constant-round protocols [BL04].

## B Proof Systems

We provide a detailed description of the proof systems used in this work.

### B.1 Protocol $\Pi_{\text{WIPOK}}$

This is essentially the Feige-Lapidot-Shamir protocol, slightly reworded in [KO04], mostly for notational convenience. We recall this protocol here. We denote the messages of this protocol by  $(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3)$ .

We will be working with the NP-complete language  $HC$  of graph Hamiltonicity, and thus assume statements to be proven take the form of graphs, while witnesses correspond to Hamilton cycles. If  $\mathbf{thm}$  is a graph, we abuse notation and also let  $\mathbf{thm}$  denote the statement “ $\mathbf{thm} \in HC$ ”. We show how the proof system can be used to prove the following statement:  $\mathbf{thm} \wedge \mathbf{thm}'$ , where  $\mathbf{thm}$  will be included as part of the first message, while  $\mathbf{thm}'$  is only decided in the last round. The proof system  $\Pi_{\text{WIPOK}}$  runs  $\kappa$  parallel executions of the following 3-round protocol:

1. The prover commits to two adjacency matrices for two randomly-chosen cycle graphs  $G, G'$ . The commitment is done bit-by-bit using a perfectly-binding commitment scheme.
2. The verifier responds with a single bit  $b$ , chosen at random.
3. If  $b = 0$ , the prover opens all commitments. If  $b = 1$ , the prover sends two permutations mapping the cycle in  $\mathbf{thm}$  (resp.,  $\mathbf{thm}'$ ) to  $G$  (resp.,  $G'$ ). For each non-edge in  $\mathbf{thm}$  (resp.,  $\mathbf{thm}'$ ), the prover opens the commitment at the corresponding position in  $G$  (resp.,  $G'$ ).

The verifier checks that all commitments were opened correctly. If  $b = 0$ , the verifier additionally checks whether both decommitted graphs are indeed cycle graphs. If  $b = 1$ , the verifier checks whether each non-edge in  $\mathbf{thm}$  (resp.,  $\mathbf{thm}'$ ) corresponds to a non-edge in  $G$  (resp.,  $G'$ ).

Note that the prover does not need to know either  $\mathbf{thm}$  or  $\mathbf{thm}'$  (or the corresponding witnesses) until the beginning of the third round. In the above proof system, we assume that  $\mathbf{thm}$  is fixed as part of the first-round message enabling us to claim stronger properties about the proof system. In particular,  $\Pi_{\text{WIPOK}}$  proof system is complete and sound. More specifically, the probability that an all-powerful prover can cause a verifier to accept when either  $\mathbf{thm}$  or  $\mathbf{thm}'$  are not true is at most  $2^{-\kappa}$ . We stress that this holds even if the prover can adaptively choose  $\mathbf{thm}'$  after viewing the second-round message of the verifier. Moreover,  $\Pi_{\text{WIPOK}}$  is witness indistinguishable and it is a proof of knowledge for  $\mathbf{thm}$ . (More formally, we can achieve a notion similar to that of “witness-extended emulation” [Lin01] for  $\mathbf{thm}$ .) Note also that the first round of the above proof system (as well as the internal state of the prover immediately following this round) is independent of  $\mathbf{thm}$  or the associated witness.

### B.2 Protocol $\Pi_{\text{FS}}$

As noted in Section 2, this is essentially the four round zero-knowledge protocol of Feige-Shamir, except that we use *non-malleable commitments* in the first three round of the protocol. Following the discussion in Section 2, we let  $\text{nmcom}$  be a non-malleable commitment scheme, and make the simplifying assumption

that  $\text{nmcom}$  has just three rounds and the first round is committing. Again, these are purely for notational convenience and can easily be removed (as discussed earlier).

We now simply list all the steps of this protocol following [KO04], but using  $\text{nmcom}$ . The messages of this protocol are denoted by  $(\text{fs}_1, \text{fs}_2, \text{fs}_3, \text{fs}_4)$ . It allows the prover to prove  $\text{thm} \wedge \text{thm}'$  where  $\text{thm}$  is sent as part of the second round yet  $\text{thm}'$  is only sent as part of the last round. (Intuitively, statements  $\text{thm}, \text{thm}'$  will correspond to statements  $\text{st}_1, \text{st}_2$  of  $\Pi_{\text{WIPOK}}$  described above.)

The proof system  $\Pi_{\text{FS}}$  proceeds as follows:

1. The first round is as in the original Feige-Shamir protocol but augmented with an  $\text{nmcom}$  scheme. Explicitly, the verifier  $V$  selects randomly and independently two values  $\sigma_1$  and  $\sigma_2$  and computes the first message of two independent executions of  $\text{nmcom}$  for  $\sigma_1$  and  $\sigma_2$ , with randomness  $\rho_1, \rho_2$  respectively. Let  $\text{nm}_1^{\sigma_1}$  and  $\text{nm}_2^{\sigma_2}$  be these messages, which  $V$  sends to  $P$ .  
Moreover,  $V$  sends the first message  $\text{p}_1$  of a WIPOK proof system.
2. The prover  $P$  chooses a random challenge  $R \in \{0, 1\}^{2\kappa}$  and computes  $\text{C}_R = \text{Eqcom}(R; \zeta)$ . Let  $\text{eqthm}$  denote the statement that  $\text{Eqcom}$  was formed correctly.  
Let  $\widetilde{\text{thm}}$  denote the statement:  $(\text{thm} \wedge \text{eqthm}) \vee (\text{nm}_1^{\sigma_1} = \text{nmcom}_1(\sigma_1; \rho_1)) \vee (\text{nm}_2^{\sigma_2} = \text{nmcom}_1(\sigma_2; \rho_2))$  (this statement is reduced to a single graph  $\widetilde{\text{thm}}$ ). Then,  $P$  sends  $\text{C}_R$  and also the first message  $\tilde{\text{p}}_1$  of a separate WIPOK proof system and message  $\text{p}_2$  of  $V$ 's proof.
3.  $V$  sends the last message  $\text{p}_3$  of his WIPOK proof system and completes the proof for the knowledge of the values in  $\text{nmcom}$  (which is also completed along with the first and second rounds<sup>24</sup>).  $V$  additionally sends a random  $R' \in \{0, 1\}^{2\kappa}$  and message  $\tilde{\text{p}}_2$  of  $P$ 's proof
4.  $P$  decommits to  $R$ . Let  $\text{prg}$  be the statement that  $r = R \oplus R'$  is pseudorandom (i.e.,  $\exists s$  s.t.  $\text{PRG}(s) = r$ , where  $\text{PRG}$  is a pseudorandom function). Let  $\widetilde{\text{thm}'}$  be the statement  $\text{thm}' \vee \text{prg}$  (reduced to a single graph  $\widetilde{\text{thm}'}$ ). The prover send the last message  $\tilde{\text{p}}_3$  of the  $\Pi_{\text{WIPOK}}$  proof system and completes the proof for the statement  $\widetilde{\text{thm}} \wedge \widetilde{\text{thm}'}$ .  
 $V$  checks the decommitment of  $R$ , and verifies the proof.

As claimed in [KO04]  $\Pi_{\text{FS}}$  proof system satisfies the following properties. It is complete and sound (for a poly-time prover) for  $\text{thm}$  and  $\text{thm}'$ . Rounds 2 – 4 constitute a proof of knowledge for  $\widetilde{\text{thm}}$ . If a poly-time prover can cause a verifier to accept with “high” probability, then a witness for  $\widetilde{\text{thm}} \wedge \text{eqthm}$  can be extracted with essentially the same probability. If  $\text{eqthm}$  is true, then with all but negligible probability  $\text{prg}$  will not be true. Soundness of the proof of knowledge sub-protocol then implies that  $\widetilde{\text{thm}'}$  is true. But this means that  $\text{thm}'$  is true.  $\Pi_{\text{FS}}$  is also zero-knowledge (in addition, to simulating for  $\widetilde{\text{thm}}$ , the simulator also uses the equivocal commitment property to decommit to an  $R$  such that  $\text{prg}$  is true.). Furthermore,  $\Pi_{\text{FS}}$  is an argument of knowledge for  $\text{thm}$ .

Note that although we are using  $\text{nmcom}$  we are not making any claim here that uses non-malleability. All claims above simply rely on the hiding of  $\text{nmcom}$ . The non-malleability is used by the two-party protocol which uses  $\Pi_{\text{FS}}$ .

Also note that in order to handle a general  $\text{nmcom}$  of  $k$  rounds, simply execute the first  $k - 3$  rounds before the protocol above begins. The statements are then modified to work with the transcript, rather than the first message of the protocol.

<sup>24</sup> If  $k > 3$  then  $V$  completes its WIPOK after the completion of  $\text{nmcom}$ .