On the Power of an Honest Majority in Three-Party Computation Without Broadcast

Bar Alon[1*], Ran Cohen[2†], Eran Omri[1*], and Tom Suad[1*]

[1] Ariel University. Ariel Cyber Innovation Center (ACIC).
alonbar08@gmail.com, omrier@ariel.ac.il, tomsuad7@gmail.com
[2] Northeastern University. rancohen@ccs.neu.edu

Abstract. Fully secure multiparty computation (MPC) allows a set of parties to compute some function of their inputs, while guaranteeing correctness, privacy, fairness, and output delivery. Understanding the necessary and sufficient assumptions that allow for fully secure MPC is an important goal. Cleve (STOC'86) showed that full security cannot be obtained in general without an honest majority. Conversely, by Rabin and Ben-Or (FOCS'89), assuming a broadcast channel and an honest majority enables a fully secure computation of any function.

Our goal is to characterize the set of functionalities that can be computed with full security, assuming an honest majority, but no broadcast. This question was fully answered by Cohen et al. (TCC'16) – for the restricted class of *symmetric* functionalities (where all parties receive the same output). Instructively, their results crucially rely on *agreement* and do not carry over to general *asymmetric* functionalities. In this work, we focus on the case of three-party asymmetric functionalities, providing a variety of necessary and sufficient conditions to enable fully secure computation.

An interesting use-case of our results is *server-aided* computation, where an untrusted server helps two parties to carry out their computation. We show that without a broadcast assumption, the resource of an external non-colluding server provides no additional power. Namely, a functionality can be computed with the help of the server if and only if it can be computed without it. For fair coin tossing, we further show that the optimal bias for three-party (server-aided) r-round protocol remains $\Theta(1/r)$ (as in the two-party setting).

Keywords: broadcast; point-to-point communication; multiparty computation; coin flipping; impossibility result; honest majority.

*Research supported by ISF grant 152/17, and by the Ariel Cyber Innovation Center in conjunction with the Israel National Cyber directorate in the Prime Minister's Office.

†Research supported by NSF grant 1646671.

1 Introduction

In the setting of secure multiparty computation [39, 28, 9, 16, 38], a set of mutually distrustful parties wish to compute a function f of their private inputs. The computation should preserve a number of security properties even facing a subset of colluding cheating parties, such as: *correctness* (cheating parties can only affect the output by choosing their inputs), *privacy* (nothing but the specified output is learned), *fairness* (all parties receive an output or none do), and even *guaranteed output delivery* (meaning that all honestly behaving parties always learn an output). Informally speaking, a protocol π computes a functionality f with *full security* if it provides all of the above security properties.[3]

In the late 1980's, it was shown that every function can be computed with full security in the presence of malicious adversaries corrupting a strict minority of the parties, assuming the existence of a *broadcast communication channel* (such a channel allows any party to reliably send its message to all other parties, guaranteeing that all parties receive the same message) and pairwise private channels (that can be established over broadcast using standard cryptographic techniques) [9, 38]. On the other hand, a well-known lower bound by Cleve [17] shows that if an honest majority is not assumed, then *fairness* cannot be guaranteed in general (even assuming a broadcast channel). More specifically, Cleve's result showed that given a two-party, r-round coin-tossing protocol, there exists an (efficient) adversarial strategy that can bias the output bit by $\Omega(1/r)$.

Conversely, a second well-known lower bound from the 1980's shows that in the plain model (i.e., without setup/proof-of-work assumptions), no protocol for computing broadcast can tolerate corruptions of one third of the parties [37, 34, 22].

This leads us to the main question studied in this paper:

What is the power of the honest-majority assumption
in a model where parties cannot broadcast?

Namely, we set to characterize the set of n-party functionalities that can be computed with full security over point-to-point channels in the plain model (i.e., without broadcast), in the face of malicious adversaries, corrupting up to t parties, where $n/3 \leq t < n/2$.

Cohen et al. [19] answered the above question for *symmetric functionalities*, where all parties obtain the same common output from the computation. They showed that, in the plain model over point-to-point channels, a function f can be computed with full security if and only if f is $(n-2t)$-*dominated*, i.e., there exists a value y^* such that any $n-2t$ of the inputs can determine the output of f to be y^* (for example, Boolean OR is 1-dominated since any input can be set to 1, forcing the output to be 1). They further showed that there is no n-party, $\lceil n/3 \rceil$-secure, δ-bias coin-tossing protocol, for any $\delta < 1/2$.

[3] The notion of full security is formally captured via the real vs. ideal paradigm, where the protocol is said to be secure if it emulates some ideal setting, in which the capabilities of the adversary are very limited.

The results in [19] leave open the setting of *asymmetric functionalities*, where each party computes a different function over the same inputs. Such functionalities include symmetric computations as a special case, but they are more general since the output that each party receives may be considered private and some parties may not even receive any output. Specifically, the lower bound from [19] does not translate into the asymmetric setting, as it crucially relies on a *consistency* requirement on the protocol, ensuring that all honest parties output the same value.

Asymmetric computations are very natural in the context of MPC in general, however, the following two use-cases are of particular interest:

Server-aided computation: Augmenting a two-party computation with a (potentially untrusted) server that provides no input and obtains no output has proven to be a very useful paradigm in overcoming lower bounds, even when the server may collude with one of the parties as in the case of optimistic fairness [6, 14]. In the broadcast model, considering a *non-colluding* server is a real game changer, as it enables two parties to compute *any* function with full security. In our setting, where broadcast is not available, we explore to what extent a non-colluding server can boost the security of two-party computation. For the specific task of coin tossing we ask: "can two parties use a non-colluding third party to help them toss a coin?"

Computation with solitary output: Halevi et al. [32] studied computations in which only a single party obtains the output, e.g., a server that learns a function of the inputs of two clients. The focus of [32] was on the broadcast model with a dishonest majority, and they showed a variety of feasibility and infeasibility results. In this work we consider a model without broadcast but with an honest majority, which reopens the feasibility question. Fitzi et al. [25] showed that if the three-party solitary-output functionality *convergecast*[4] can be securely computed facing a single corruption, then so can the broadcast functionality; thus, proving the impossibility of securely computing convergecast in our setting. In this work, we extend the exploration of the set of securely computable solitary-output functionalities.

1.1 Split-Brain Simulatability

In this paper, we focus on general asymmetric three-party functionalities, where party A with input x, party B with input y, and party C with input z, compute a functionality $f = (f_1, f_2, f_3)$. The output of A is $f_1(x, y, z)$, the output of B is $f_2(x, y, z)$, and the output of C is $f_3(x, y, z)$. We will also consider the special cases of *two-output functionalities* where only A and B receive output (meaning that f_3 is degenerate), and of *solitary-output functionalities* where only A receives output (meaning that f_2 and f_3 are degenerate).

[4] Convergecast [25] is a three-party functionality where two of the parties start with a non-Boolean input, and the receiver learns exactly one of the input values. The receiver does not learn anything about the other input, and none of the senders learns the receiver's choice as well as the input of the other sender.

Our main technical contribution is adapting the so called *split-brain argument*, which was previously used in the context of Byzantine agreement [21, 12, 11] (where privacy is not required, but agreement must be guaranteed) to the setting of MPC. Indeed, aiming at full security, we are able to broaden the collection of infeasible functionalities. In particular, our results apply to the setting where the parties do not necessarily agree on a common output.

In Section 1.3 we provide a more detailed overview of the split-brain attack, however, the core idea can be explained as follows. Let $f = (f_1, f_2, f_3)$ be an asymmetric (possibly randomized) three-party functionality and let π be a secure protocol computing f over point-to-point channels, tolerating a single corruption. For the sake of simplicity of the presentation, in the remaining of this introduction we only consider perfect security and functionalities with finite domain and range. A formal treatment for general functionalities and computational security is given in Section 3.1. Consider the following two scenarios:

- A corrupted (split-brain) party C playing two independent interactions: in the *first* interaction, C interacts with A on input z_1 acting as if it never received any incoming messages from B, and in the *second* interaction, C interacts with B on input z_2, acting as if it never received any incoming messages from A.
- A corrupted party A internally emulating a first interaction of the above split-brain C: A interacts with B, ignoring all incoming messages from the honest C; instead, A emulates in its head the above (first-interaction) C on input z_1 (This part of the attack relies on the no-trusted-setup assumption, and on the fact that emulating C requires no interaction with B).

Clearly, the view of party B is identically distributed in both of these scenarios; hence, its output must be identically distributed as well. Note that by symmetry, an attacker B can be defined analogously to the above A, causing the output of an honest A to distribute as when interacting with the split-brain C.

By the assumed security of the protocol π, each of the three attacks described above can be simulated in an ideal world where a trusted party computes f for the parties. Since the only power the simulator has in the ideal world is to choose the input for the corrupted party, we can capture the properties that the functionality f must satisfy to enable the existence of such simulators via the following definition.

Definition 1 (CSB-simulatability, informal). *A three-party functionality $f = (f_1, f_2, f_3)$ is* C-split-brain (CSB) simulatable *if for every quadruple (x, y, z_1, z_2), there exist a distribution P_{x,z_1} over the inputs of A, a distribution Q_{y,z_2} over the inputs of B, and a distribution R_{z_1,z_2} over the inputs of C, such that*

$$f_1(x, y^*, z_1) \equiv f_1(x, y, z^*) \quad \text{and} \quad f_2(x^*, y, z_2) \equiv f_2(x, y, z^*),$$

where $x^* \leftarrow P_{x,z_1}$, $y^* \leftarrow Q_{y,z_2}$, and $z^* \leftarrow R_{z_1,z_2}$.

1.2 Our Results

Using the notion of split-brain simulatability, mentioned in Section 1.1, we present several necessary conditions for an asymmetric three-party functionality to be securely computable without broadcast while tolerating a single corruption. We also present a sufficient condition for two-output functionalities (including solitary output functionalities as a special case); the latter result captures and generalizes previously known feasibility results in this setting, including 1-dominated functionalities [18, 19] and fair two-party functionalities [5]. Examples illustrating the implications of these theorems for different functionalities is provided in Table 1.

Impossibility results. Our first impossibility result asserts that CSB simulatability is a necessary condition for securely computing a three-party functionality in our setting.

Theorem 1 (necessity of split-brain simulatability, informal). *A three-party functionality that can be securely computed over point-to-point channels, tolerating a single corruption, must be CSB simulatable.*

We can define A-split-brain and B-split-brain simulatability analogously, thus providing additional necessary conditions for secure computation. For a formal statement and proof we refer the reader to Section 3.1.

To illustrate the usefulness of the theorem, consider the two-output functionality where $f_1 = f_2$ are defined as $f_1(x, y, z) = (x \wedge y) \oplus z$. Note that since f_3 is degenerate, this functionality is not symmetric and therefore the lower bound from [19] does not rule it out. We next show that f is *not* CSB simulatable, and hence, cannot be securely computed. Clearly, input 0 for A and C will fix the output to be 0, whereas input 0 for B and input 1 for C will fix the output to be 1. The CSB simulatability of f would require that there exists distributions for sampling x^* and y^* such that

$$0 \quad \equiv \quad f_1(0, y^*, 0) \quad \equiv \quad f_1(x^*, 0, 1) \quad \equiv \quad 1.$$

This leads to a contradiction.

Our second result, is in the *server-aided* model, where only A and B provide input and receive output. We show that in this model, a functionality can be computed with the help of C if and only if it can be computed without C. In the theorem below we denote by λ the empty string.

Theorem 2 (server-aided computation is as strong as two-party computation, informal). *Let f be a three-party functionality where C has no input and no output. Then, f can be securely computed over point-to-point channels tolerating a single corruption if and only if the induced two-party functionality $g(x, y) = f(x, y, \lambda)$ can be computed with full security.*

An immediate corollary from Theorem 2 is that a non-colluding third party cannot help the two parties to toss a fair coin. In fact, if C cannot attack the

protocol (i.e., C cannot bias the output coin), then the attack that is guaranteed by Cleve [17] (on the implied two-party protocol) can be directly translated to an attack on the three-party protocol, corrupting either A or B. Stated differently, either A or B can bias the output by $\Omega(1/r)$, where r is the number of rounds in the protocol. However, the above argument does not deal with protocols that allow a corrupt C to slightly bias the output. For example, one might try to construct a protocol where every party (including C) can bias the output by at most $1/r^2$. We strengthen the result for coin tossing, showing that this is in fact impossible.

Theorem 3 (implication to coin tossing, informal). *Consider a three-party, two-output, r-round coin-tossing protocol. Then, there exists an adversary corrupting a single party that can bias the output by $\Omega(1/r)$.*

As a result, letting A and B run the protocol of Moran et al. [36] constitutes an optimally fair (up to a constant) coin-tossing protocol.

We note that using a standard player-partitioning argument, the impossibility result extends to n-party r-round coin-tossing protocols, where *two* parties receive the output. Specifically, there exists an adversary corrupting $\lceil n/3 \rceil$ parties that can bias the output by $\Omega(1/r)$. Further, using [19, Lem. 4.10] we rule out any non-trivial n-party coin-tossing where *three* parties receive the output.

Another immediate corollary from Theorem 2 is that two-output functionalities that imply coin-tossing are not securely computable, even if C has an input. For example, the XOR function $(x, y, z) \mapsto x \oplus y \oplus z$ is not computable facing one corruption. For a formal treatment of server-aided computation, see Section 3.2.

Our third impossibility result presents two functionalities that are not captured by Theorems 2 and 3. Interestingly, unlike the previous results, here we make use of the *privacy* requirement on the protocol for obtaining the proof. We refer the reader to Section 1.3 below for an intuitive explanation, and Section 3.3 for a formal proof.

Theorem 4 (Informal). *Let f be a solitary-output three-party functionality where $f_1(x, y, z) = (x \wedge y) \oplus z$ (equivalently, $f_1(x, y, z) = (x \oplus y) \wedge z$). Then, f cannot be securely computed over point-to-point channels tolerating a single corruption.*

Feasibility results. We proceed to state our sufficient condition. We present a class of two-output functionalities f that can be computed with full security. Interestingly, our result shows that if f is CSB simulatable, then under a simple condition that a related two-party functionality needs to satisfy, the problem is reduced to the two-party case. In the related two-party functionality, the first party holds x and z_1, while the second party holds y and z_2, and is defined as $f'((x, z_1), (y, z_2)) = f(x, y, z^*)$, where $z^* \leftarrow R_{z_1, z_2}$ is sampled as in the requirement of CSB simulatability.

Roughly, we require that there exist two distributions, for z_1 and z_2 respectively, such that the input z_1 can be sampled in a way that fixes the distribution

of the output of f' to be independent of z_2, and similarly, that the input z_2 can be sampled in a way that fixes the distribution of the output of f' to be independent of z_1. Specifically, we prove the following.

Theorem 5 (Informal). *Let $f = (f_1, f_2)$ be a CSB simulatable three-party, two-output functionality. Define the two-party functionality f' as $f'((x, z_1), (y, z_2)) = f(x, y, z^*)$, where $z^* \leftarrow R_{z_1,z_2}$.*

Assume that there exists a randomized two-party functionality $g = (g_1, g_2)$ and two distributions R_1 and R_2 over C's inputs such that for every $x, y, z \in \{0,1\}^*$ it holds that $g(x, y) \equiv f'((x, z_1), (y, z)) \equiv f'((x, z), (y, z_2))$, where $z_1 \leftarrow R_1$ and $z_2 \leftarrow R_2$.

If g can be securely computed with full security then f can be securely computed with full security over point-to-point channels tolerating a single corruption.

The idea behind the protocol is as follows. First, by the honest-majority assumption, the parties can compute f with guaranteed output delivery *assuming a broadcast channel* [38]. By [18] it follows that they can compute f with *fairness* without using broadcast. If the parties receive an output, they can terminate; otherwise, A and B compute g using their inputs, *ignoring* C *in the process* (even if it is honest).

Intuitively, the existence of R_1 and R_2 allows the simulators of a corrupt A or a corrupt B, to "force" a computation of g in the ideal world of f; that is, the output will be independent of the input of C. To see this, consider a corrupt A and let $z_1 \leftarrow R_1$. Then, by the CSB simulatability assumption, sending $x^* \leftarrow P_{x,z_1}$ to the trusted party results in the output being

$$f(x^*, y, z) \equiv f(x, y, z^*) \equiv f'((x, z_1), (y, z)) \equiv g(x, y),$$

where $z^* \leftarrow R_{z_1,z}$.

In Section 1.3 below we give a more detailed overview of the proof. In Section 4 we present the formal statement and proof of the theorem.

We briefly describe a few classes of functions that are captured by Theorem 5. First, observe that the class of functionalities satisfying the above conditions contains the class of 1-dominated functionalities [19]. To see this, notice that x^*, y^*, and z^* can be sampled in a way that always fixes the output of f to be some value w^*. Then, any choice of R_1 and R_2 will do. Furthermore, observe that the resulting two-party functionality g will always be the constant function, with the output being w^*.

Another class of functions captured by the theorem is the class of fair two-party functionalities. For such functionalities the distributions R_{z_1,z_2}, R_1, and R_2 can be degenerate, as z^*, z_1, and z_2 play no role in the computation of f and f'. Additionally, taking $x^* = x$ and $y^* = y$ with probability 1 will satisfy the CSB simulatability constraint.

Next, we show that the class of functionalities satisfying the conditions of Theorem 5 includes functionalities that are *not* 1-dominated. Consider as an

example the solitary XOR function $f(x, y, z) = x \oplus y \oplus z$. Note that for solitary-output functionalities the two-party functionality g can always be securely computed assuming oblivious transfer [33]. Furthermore, f is CSB simulatable since we can sample y^* and z^* uniformly at random. In addition, taking R_1 and R_2 to output a uniform random bit as well will satisfy the conditions of Theorem 5.

Finally, there are even two-output functionalities that are not 1-dominated, yet are still captured by Theorem 5. For example, consider the following three-party variant of the GHKL function [29], denoted 3P-GHKL: let $f = (f_1, f_2)$, where $f_1, f_2 : \{0, 1, 2\} \times \{0, 1\} \times \{0, 1\} \mapsto \{0, 1\}$. The functionality is defined by the following two matrices

$$M_0 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \qquad M_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$$

where $f_1(x, y, z) = f_2(x, y, z) = M_z(x, y)$. That is, A's input determines a row, B's input determines a column, and C's input determines the matrix. For the above functionality, sampling y^* and z^* uniformly at random, and taking $x^* = x$ if $x = 2$ and a uniform bit otherwise, will always generate an output that is equal to 1 if $x = 2$ and a uniform bit otherwise. See Section 4.1 for more details.

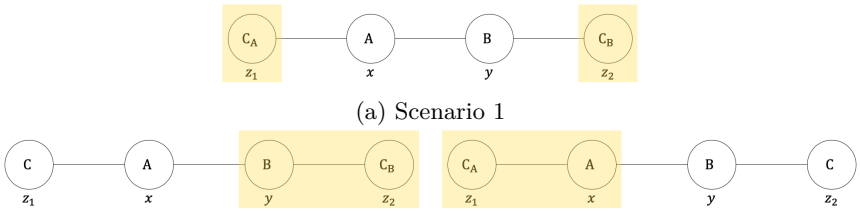
functionality	**A outputs**	**A, B output**	**A, B, C output**
$x \wedge y \wedge z$	✓ [18]	✓ [18]	✓ [18]
$(x \wedge y) \oplus z$ $(x \oplus y) \wedge z$	✗ Thm 4	✗ Thm 1 (also 4)	✗ [19] (also Thm 1, 4)
$x \oplus y \oplus z$ $x \wedge (y \oplus z)$ $x \oplus (y \wedge z)$	✓ Thm 5	✗ Thm 2	✗ [19] (also Thm 2)
3P-GHKL	✓ Thm 5	✓ Thm 5	✗ [19]
δ-bias coin tossing	✓ $\delta = 0$ (trivial)	✓ $\delta = \Theta(1/r)$ [36] ✗ $\delta = o(1/r)$, Thm 3	✗ $\delta < 1/2$ [19]

Table 1: Summarizing the feasibility of interesting three-party functionalities tolerating one corruption. The second column considers the solitary case where only A receives the output, the third the two-output case where both A and B receive the same output, and the last column the case where all parties receive the same output.

1.3 Our Techniques

We now turn to describe our techniques, starting with our impossibility results. The core argument in all of our proofs, is the use of an adaptation of the split-brain argument [21, 12, 11] to the MPC setting.

The C*-split-brain argument.* In the following, let f be a three-party functionality and let π be a protocol computing f with full security over point-to-point channels, tolerating a single corrupted party. Consider the following three attack-scenarios with inputs x, y, z_1, z_2 depicted in Figure 1.



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

Fig. 1: Three adversaries of the C-split-brain attack. The shaded yellow areas in each scenario correspond to the (virtual) parties the adversary controls.

Scenario 1: Parties A and B both play honestly on inputs x and y respectively. The adversary corrupts C and applies the *split-brain attack*, that is, it emulates in its head two virtual copies of C, denoted $\mathsf{C_A}$ and $\mathsf{C_B}$. $\mathsf{C_A}$ interacts with A as an honest C would on input z_1 as if it never received messages from B, and $\mathsf{C_B}$ interacts with B as an honest C would on input z_2 as if it never received messages from A.

By the assumed security of π, there exists a simulator for the corrupted C. This simulator defines a distribution over the input it sends to the trusted party. Thus, the outputs of A and B in this case must equal to $f_1(x, y, z^*)$ and $f_2(x, y, z^*)$, respectively, where z^* is sampled according to some distribution that depends only on z_1 and z_2.

Scenario 2: Party A plays honestly on input x and party C plays honestly on input z_1. The adversary corrupts party B, ignoring all incoming messages from C and not sending it any messages. Instead, the adversary emulates in its head the virtual party $\mathsf{C_B}$ as in Scenario 1, that plays honestly on input z_2 as if it never received any message from A. Additionally, the adversary instructs B to play honestly in this setting.

As in Scenario 1, by the assumed security of π, the output of A in this case must equal $f_1(x, y^*, z_1)$, where y^* is sampled according to some distribution that depends only on y and z_2.

Scenario 3: This is analogous to Scenario 2. Party B plays honestly on input y and party C plays honestly on input z_2. The adversary corrupts party A, ignoring all incoming messages from C and not sending it any messages. Similarly to Scenario 2, the adversary emulates in its head the virtual $\mathsf{C_A}$, that plays honestly on input z_1 as if it never received any message from B, and instructs A to plays honestly in this setting.

As in the previous two scenarios, the output of B must equal $f_2(x^*, y, z_2)$, where x^* is sampled according to some distribution that depends only on x and z_1.

Observe that the view of the honest A in Scenario 1 is identically distributed as its view in Scenario 2; hence the same holds with respect to its output, i.e., $f_1(x, y^*, z_1) \equiv f_1(x, y, z^*)$. Similarly, the view of the honest B in Scenario 1 is identically distributed as its view Scenario 3; hence, $f_2(x^*, y, z_2) \equiv f_2(x, y, z^*)$. This proves the necessity of C-split-brain simulatability (i.e., Theorem 1).

The four-party protocol. A nice way to formalize the above argument is by constructing a four-party protocol π' from the three-party protocol π, where two different parties play the role of C (see Figure 2). In more detail, define the four-party protocol π', with parties A', B', C'_{A}, and C'_{B}, as follows. Party A' follows the code of A, party B' follows the code of B, and parties C'_{A} and C'_{B} follow the code of C.

The parties are connected on a path where (1) C'_{A} is the leftmost node, and is connected only to A', (2) C'_{B} is the rightmost node, and is connected only to B', and (3) A' and B' are also connected to each other. The second communication line of party C'_{A} is "disconnected" in the sense that C'_{A} is sending the messages as instructed by the protocol, but the messages arrive at a "sink" that does not send any messages back. Stated differently, the view of C'_{A} corresponds to the view of an honest C in π that never received any message from B. Similarly, the second communication line of party C'_{B} is "disconnected," and its view corresponds to the view of an honest C in π that never receives any message from A.

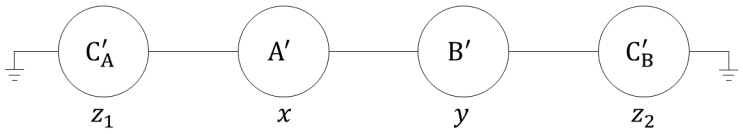


Fig. 2: The induced four-party protocol

Server-aided computation. We now make use of the four-party protocol to sketch the proof of Theorem 2. Recall that here we consider the server-aided model, where C (the server) has no input and obtains no output. Observe that under the assumption that π securely computes f, it follows that the four-party protocol π' correctly computes the two-output four-party functionality $f'(x, y, \lambda, \lambda) := f(x, y, \lambda)$, where λ is the empty string, as otherwise C could emulate C'_{A} and C'_{B} in its head and force A and B to output an incorrect value.

Next, consider the following *two-party* protocol $\widehat{\pi}$ where each of two pairs $\{\mathsf{A}', \mathsf{C}'_{\mathsf{A}}\}$ and $\{\mathsf{B}', \mathsf{C}'_{\mathsf{B}}\}$, is emulated by a single entity, $\widehat{\mathsf{A}}$ and $\widehat{\mathsf{B}}$ respectively, as depicted in Figure 3. Observe that the protocol computes the two-party functionality $g(x, y) := f'(x, y, \lambda, \lambda) = f(x, y, \lambda)$. Furthermore, it computes g securely, since any adversary for the two-party protocol directly translates to an adversary for the three-party protocol corrupting either A or B. Moreover, since C does not have an input, the simulators of those adversaries in π can be directly translated to simulators for the adversaries in $\widehat{\pi}$. Thus, f can be computed with the help of C if and only if it can be computed without C.

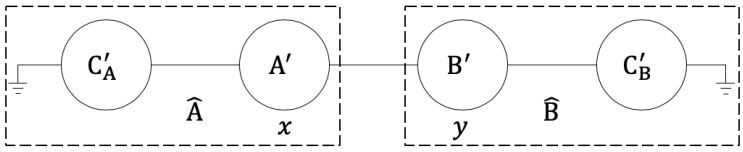


Fig. 3: The induced two-party protocol

The proof of Theorem 3, i.e., that the optimal bias for the server-aided coin-tossing protocol is $\Theta(1/r)$, extends the above analysis. We show that for any r-round server-aided coin-tossing protocol π there exists a constant c and an adversary that can bias the output by at least $1/cr$. Roughly, assuming that party C cannot bias the output of π by more than $1/cr$, the output of A' and B' in the four-party protocol is a common bit that is $(1/cr)$-close to being uniform. Therefore, the same holds with respect to the outputs of $\widehat{\mathsf{A}}$ and $\widehat{\mathsf{B}}$ in the two-party protocol. Now, we can apply the result of Agrawal and Prabhakaran [1], which generalizes Cleve's [17] result to a general two-party sampling functionality. Their result provides an adversary for the two-party functionality that can bias by $1/dr$ for some constant d. Finally, we can emulate their adversary in the three party protocol as depicted in Scenarios 2 and 3. For sufficiently large c (specifically, $c > 2d$), the bias resulting from emulating the adversaries for the two-party protocol will be at least $1/cr$.

Impossibility based on privacy. We next sketch the proof of Theorem 4. That is, the solitary-output functionalities $f_1(x, y, z) = (x \wedge y) \oplus z$ and $f_1(x, y, z) = (x \oplus y) \wedge z$ cannot be computed in our setting. We prove it only for the case where the output of A is defined to be $f_1(x, y, z) = (x \wedge y) \oplus z$. The other case is proved using a similar analysis. The proof starts from the C-split-brain argument used in the proof of Theorem 1. Assume for the sake of contradiction that π computes f with perfect security. First, let us consider the following two scenarios.

- C is corrupted as in Scenario 1, i.e., it applies the split-brain attack with inputs z_1 and z_2. By the security of π, the output of A in this case is $(x \wedge y) \oplus z^*$, for some z^* that is sampled according some distribution that depends on z_1 and z_2.
- B is corrupted as in Scenario 2, i.e., it imagines that it interacts with C_{B} with input z_2 that does not receive any message from A. In this case, the output of A is $(x \wedge y^*) \oplus z_1$, where z_1 is the input of the real C, and y^* is sampled according some distribution that depends on y and z_2.

By Theorem 1 these two distributions are identically distributed for all $x \in \{0, 1\}$. Notice that setting $x = 0$ yields $z^* = z_1$ and that setting $x = 1$ yields $y^* \oplus z_1 = y \oplus z^*$, hence $y^* = y$. Therefore, the output of A in both scenarios is $(x \wedge y) \oplus z_1$.

Finally, consider an execution of π over random inputs y for B and z for C, where A is corrupted as in Scenario 2, and it emulates C'_{A} on input z_1. Since its

view is exactly the same as in the other two scenarios, it can compute $(x \wedge y) \oplus z_1$. However, in this scenario A can choose $x = 1$ and $z_1 = 0$ and thus learn y. In the ideal world, however, A cannot guess y with probability better than $1/2$, as the output it sees is $(x \wedge y) \oplus z$ for random y and z. Hence, we have a contradiction to the security of π.

A protocol for computing certain two-output functionalities. Finally, We describe the idea behind the proof of Theorem 5. First, by the honest-majority assumption, the protocol of Rabin and Ben-Or [38] computes f assuming a *broadcast channel*; by [18] it follows that f can be computed with *fairness* over a point-to-point network. We now describe the protocol. The parties start by computing f with fairness. If they receive outputs, then they can terminate, and output what they received. If the protocol aborts, then A and B compute g with their original inputs using a protocol that guarantees output delivery (such a protocol exists by assumption), and output whatever outcome is computed. Clearly, a corrupt C cannot attack the protocol. Indeed, it does not gain any information in the fair computation of f; hence, if it aborts in this phase then the output of A on input x and B on input y will be $g(x, y) = f'((x, z_1), (y, z)) = f(x, y, z^*)$, where $z_1 \leftarrow R_1$ and $z^* \leftarrow R_{z_1,z}$.

We next consider a corrupt A (the case of a corrupt B is analogous). The idea is to take the distribution over the inputs used by the *two-party* simulator, and translate it into an appropriate distribution for the *three-party* simulator. That is, regardless of the input of C, the output of the honest party will be distributed exactly the same as in the ideal world for the two-party computation. To see how this can be done, consider a sample $z_1 \leftarrow R_1$ and let x' be the input sent by the two-party simulator to its trusted party. The three-party simulator will send to the trusted party the sample $x^* \leftarrow P_{x',z_1}$. Then, by the CSB simulatability constraint, it follows that the output will be $f(x^*, y, z) \equiv f(x', y, z^*)$, where $z^* \leftarrow R_{z_1,z}$. However, by requirement from the two-party functionality g, it follows that $g(x', y) \equiv f(x', y, z^*)$; hence, the output in the three-party ideal-world of f, is identically distributed as in the two-party ideal-world of g.

1.4 Additional Related Work

The split-brain argument has been used in the context of Byzantine agreement (BA) to rule out three-party protocols tolerating one corruption in various settings: over asynchronous networks [12] and partially synchronous networks [21], even with trusted setup assumptions such a public-key infrastructure (PKI), as well as over synchronous networks with weak forms of PKI [11]. The argument was mainly used by considering three parties $(\mathsf{A}, \mathsf{B}, \mathsf{C})$ where party A starts with 0, party B starts with 1, and party C plays towards A with 0 and towards B with 1. By the *validity* property of BA it is shown that A must output 0 and B must output 1, which contradicts the *agreement* property. Our usage of the split-brain argument is different as it considers (1) *asymmetric* computations where parties do not agree on the output (and so we do not rely on violating agree-

ment) and (2) *privacy-aware* computations that do not reveal anything beyond the prescribed output (as opposed to BA which is a privacy-free computation).

For the case of *symmetric* functionalities Cohen et al. [19] showed that $f(x, y, z)$ can be securely computed with guaranteed output delivery over point-to-point channels if and only if f is 1-dominated. Their lower bound followed the classical Hexagon argument of Fischer et al. [22] that was used for various consensus problems. Starting with a secure protocol π for f tolerating one corruption, they constructed a sufficiently large ring system where all nodes are guaranteed to output the same value (by reducing to the agreement property of f). Since the ring is sufficiently large (larger than the number of rounds in π), information from one side could not reach the other side. Combining these two properties yields an attacker that can fix some output value on one side of the ring, and force all nodes to output this value — in particular, when attacking π, the two honest parties participate in the ring (without knowing it) and so their output is fixed by the attacker. We note that this argument completely breaks when considering *asymmetric* functionalities, since it no longer holds that the nodes on the ring output the same value.

Cohen and Lindell [18] showed that any functionality that can be computed with guaranteed output delivery in the broadcast model can also be computed with *fairness* over point-to-point channels (using detectable broadcast protocols [23, 24]); as a special case, any functionality can be computed with fairness assuming an honest majority. Indeed, our lower bounds do not hold when the parties are allowed to abort upon detecting cheats, and rely on robustness of the protocol.

Recently, Garay et al. [26] showed how to compute every function in the honest-majority setting without broadcast or PKI, by restricting the power of the adversary in a proof-of-work fashion. This result falls outside our model as we consider the standard model without posing any restrictions on the resources of the adversary.

The possibility of obtaining fully secure protocols for non-trivial functions in the two-party setting (i.e., with no honest majority) was first investigated by Gordon et al. [29]. The showed that, surprisingly, such protocols do exist, even for functionalities with an embedded XOR. The feasibility and infeasibility results of [29] were substantially generalized in the works [4, 35]. The set of Boolean functionalities that are computable with full security was characterized in [5].

The breakthrough result of Moran et al. [36], who gave an optimally fair two-party coin-tossing protocol, paved the way to a long line of research on optimally fair coin-tossing. Positive results for the multiparty setting (with no honest majority) where given [7, 30, 2, 13, 20] alongside some new lower-bounds [8, 31].

Organization

In Section 2 we present the required preliminaries and formally define the model we consider. Then, in Section 3 we present our impossibility results. Finally, in Section 4 we prove our positive results.

2 Preliminaries

2.1 Notations

We use calligraphic letters to denote sets, uppercase for random variables and distributions, lowercase for values, and we use bold characters to denote vectors. For $n \in \mathbb{N}$, let $[n] = \{1, 2 \dots n\}$. For a set $\mathcal{S}$ we write $s \leftarrow \mathcal{S}$ to indicate that s is selected uniformly at random from $\mathcal{S}$. Given a random variable (or a distribution) X, we write $x \leftarrow X$ to indicate that x is selected according to X. A PPT is probabilistic polynomial time, and a PPTM is a PPT (interactive) Turing machine. We let λ be the empty string.

A function $\mu \colon \mathbb{N} \to [0,1]$ is called negligible, if for every positive polynomial $p(\cdot)$ and all sufficiently large n, it holds that $\mu(n) < 1/p(n)$. For a randomized function (or an algorithm) f we write $f(x)$ to denote the random variable induced by the function on input x, and write $f(x;r)$ to denote the value when the randomness of f is fixed to r. For a 2-ary function f and an input x, we denote by $f(x,\cdot)$ the function $f_x(y) \coloneqq f(x,y)$. Similarly, for an input y we let $f(\cdot,y)$ be the function $f_y(x) \coloneqq f(x,y)$. We extend the notations for n-ary functions in a straightforward way.

A *distribution ensemble* $X = \{X_{a,n}\}_{a \in \mathcal{D}_n, n \in \mathbb{N}}$ is an infinite sequence of random variables indexed by $a \in \mathcal{D}_n$ and $n \in \mathbb{N}$, where $\mathcal{D}_n$ is a domain that might depend on n. The statistical distance between two finite distributions is defined as follows.

Definition 2. *The statistical distance between two finite random variables X and Y is*

$$\mathrm{SD}(X,Y) = \frac{1}{2} \sum_a \left| \Pr[X = a] - \Pr[Y = a] \right|.$$

For a function $\varepsilon : \mathbb{N} \mapsto [0,1]$, the two ensembles $X = \{X_{a,n}\}_{a \in \mathcal{D}_n, n \in \mathbb{N}}$ and $Y = \{Y_{a,n}\}_{a \in \mathcal{D}_n, n \in \mathbb{N}}$ are said to be ε-close, if for all large enough n and $a \in \mathcal{D}_n$, it holds that

$$\mathrm{SD}(X_{a,n}, Y_{a,n}) \leq \varepsilon(n),$$

and are said to be ε-far otherwise. X and Y are said to be statistically close, *denoted $X \overset{S}{\equiv} Y$, if they are ε-close for some negligible function ε. If X and Y are 0-close then they are said to be equivalent, denoted $X \equiv Y$.*

Computational indistinguishability is defined as follows.

Definition 3. *Let $X = \{X_{a,n}\}_{a \in \mathcal{D}_n, n \in \mathbb{N}}$ and $Y = \{Y_{a,n}\}_{a \in \mathcal{D}_n, n \in \mathbb{N}}$ be two ensembles. We say that X and Y are computationally indistinguishable, denoted $X \overset{C}{\equiv} Y$, if for every non-uniform PPT distinguisher D, there exists a negligible function $\mu(\cdot)$, such that for all n and $a \in \mathcal{D}_n$, it holds that*

$$\left| \Pr[\mathsf{D}(X_{a,n}) = 1] - \Pr[\mathsf{D}(Y_{a,n}) = 1] \right| \leq \mu(n).$$

2.2 The Model of Computation

We provide the basic definitions for secure multiparty computation according to the real/ideal paradigm, for further details see [27]. Intuitively, a protocol is considered secure if whatever an adversary can do in the real execution of protocol, can be done also in an ideal computation, in which an uncorrupted trusted party assists the computation. For concreteness, we present the model and the security definition for three-party computation with an adversary corrupting a single party, as this is the main focus of this work. We refer to [27] for the general definition.

The Real Model

A three-party protocol π is defined by a set of three PPT interactive Turing machines A, B, and C. Each Turing machine (party) holds at the beginning of the execution the common security parameter 1^κ, a private input, and random coins. The *adversary* Adv is a *non-uniform* PPT interactive Turing machine, receiving an auxiliary information $\mathsf{aux} \in \{0,1\}^*$, describing the behavior of a corrupted party $\mathsf{P} \in \{\mathsf{A},\mathsf{B},\mathsf{C}\}$. It starts the execution with input that contains the identity of the corrupted party, its input, and an additional auxiliary input aux.

The parties execute the protocol over a synchronous network. That is, the execution proceeds in rounds: each round consists of a *send phase* (where parties send their messages for this round) followed by a *receive phase* (where they receive messages from other parties). The adversary is assumed to be *rushing*, which means that it can see the messages the honest parties send in a round before determining the messages that the corrupted parties send in that round.

We consider a fully connected point-to-point network, where every pair of parties is connected by a communication line. We will consider the *secure-channels* model, where the communication lines are assumed to be ideally private (and thus the adversary cannot read or modify messages sent between two honest parties). We assume the parties *do not* have access to a broadcast channel, and no preprocessing phase (such as a public-key infrastructure that can be used to construct a broadcast protocol) is available. We note that our upper bounds (protocols) can also be stated in the *authenticated-channels* model, where the communication lines are assumed to be ideally authenticated but not private (and thus the adversary cannot modify messages sent between two honest parties but can read them) via standard techniques, assuming public-key encryption. On the other hand, stating our lower bounds assuming secure channels will provide stronger results.

Throughout the execution of the protocol, all the honest parties follow the instructions of the prescribed protocol, whereas the corrupted party receive its instructions from the adversary. The adversary is considered to be *malicious*, meaning that it can instruct the corrupted party to deviate from the protocol in any arbitrary way. Additionally, the adversary has full-access to the view of the corrupted party, which consists of its input, its random coins, and the messages

it sees throughout this execution. At the conclusion of the execution, the honest parties output their prescribed output from the protocol, the corrupted party outputs nothing, and the adversary outputs a function of its view (containing the views of the corrupted party). In some of our proofs we consider *semi-honest* adversaries that always instruct the corrupted parties to honestly execute the protocol, but may try to learn more information than they should.

We denote by $\text{REAL}_{\pi,\mathsf{Adv}(\mathsf{aux})}(\kappa,(x,y,z))$ the joint output of the adversary Adv (that may corrupt one of the parties) and of the honest parties in a random execution of π on security parameter $\kappa \in \mathbb{N}$, inputs $x,y,z \in \{0,1\}^*$, and an auxiliary input $\mathsf{aux} \in \{0,1\}^*$.

The Ideal Model

We consider an ideal computation with *guaranteed output delivery* (also referred to as *full security*), where a trusted party performs the computation on behalf of the parties, and the ideal-world adversary *cannot* abort the computation. An ideal computation of a three-party functionality $f = (f_1, f_2, f_3)$, with $f_1, f_2, f_3 : (\{0,1\}^*)^3 \to \{0,1\}^*$, on inputs $x,y,z \in \{0,1\}^*$ and security parameter κ, with an ideal-world adversary Adv running with an auxiliary input aux and corrupting a single party P proceeds as follows:

Parties send inputs to the trusted party: Each honest party sends its input to the trusted party. The adversary Adv sends a value v from its domain as the input for the corrupted party. Let (x',y',z') denote the inputs received by the trusted party.

The trusted party performs computation: The trusted party selects a random string r, computes $(w_1,w_2,w_3) = f(x',y',z';r)$, and sends w_1 to A, sends w_2 to B, and sends w_3 to C.

Outputs: Each honest party outputs whatever output it received from the trusted party and the corrupted party outputs nothing. The adversary Adv outputs some function of its view (i.e., the input and output of the corrupted party).

We denote by $\text{IDEAL}_{f,\mathsf{Adv}(\mathsf{aux})}(\kappa,(x,y,z))$ the joint output of the adversary Adv (that may corrupt one of the parties) and the honest parties in a random execution of the ideal-world computation of f on security parameter $\kappa \in \mathbb{N}$, inputs $x,y,z \in \{0,1\}^*$, and an auxiliary input $\mathsf{aux} \in \{0,1\}^*$.

Ideal computation with fairness. Although all our results are stated with respect to guaranteed output delivery, in our proofs in Section 4 we will consider a weaker security variant, where the adversary may cause the computation to prematurely abort, but only *before* it learns any new information from the protocol. Formally, an ideal computation with *fairness* is defined as above, with the difference that during the Parties send inputs to the trusted party step, the adversary can send a special abort symbol. In this case, the trusted party sets the output $w_1 = w_2 = w_3 = \perp$ instead of computing the function.

The Security Definition

Having defined the real and ideal models, we can now define security of protocols according to the real/ideal paradigm.

Definition 4 (security). *Let f be a three-party functionality and let π be a three-party protocol. We say that π computes f with* 1-security*, if for every non-uniform* PPT *adversary* Adv*, controlling at most one party in the real world, there exists a non-uniform* PPT *adversary* Sim*, controlling the same party (if there is any) in the ideal world such that*

$$\left\{\text{IDEAL}_{f,\mathsf{Sim}(\mathsf{aux})}\left(\kappa,(x,y,z)\right)\right\}_{\kappa\in\mathbb{N},x,y,z,\mathsf{aux}\in\{0,1\}^*} \stackrel{C}{\equiv} \left\{\text{REAL}_{\pi,\mathsf{Adv}(\mathsf{aux})}\left(\kappa,(x,y,z)\right)\right\}_{\kappa\in\mathbb{N},x,y,z,\mathsf{aux}\in\{0,1\}^*}.$$

We define statistical and perfect 1-security similarly, replacing computational indistinguishability with statistical distance and equivalence, respectively.

The Hybrid Model

The *hybrid model* is a model that extends the real model with a trusted party that provides ideal computation for specific functionalities. The parties communicate with this trusted party in the same way as in the ideal models described above.

Let f be a functionality. Then, an execution of a protocol π computing a functionality g in the f-hybrid model involves the parties sending normal messages to each other (as in the real model) and in addition, having access to a trusted party computing f. It is essential that the invocations of f are done sequentially, meaning that before an invocation of f begins, the preceding invocation of f must finish. In particular, there is at most a single call to f per round, and no other messages are sent during any round in which f is called.

Let $\mathsf{type} \in \{\mathsf{g.o.d.}, \mathsf{fair}\}$. Let Adv be a non-uniform PPT machine with auxiliary input aux controlling a single party $\mathsf{P} \in \{\mathsf{A},\mathsf{B},\mathsf{C}\}$. We denote by $\text{HYBRID}^{f,\mathsf{type}}_{\pi,\mathsf{Adv}(\mathsf{aux})}(\kappa,(x,y,z))$ the random variable consisting of the output of the adversary and the output of the honest parties, following an execution of π with ideal calls to a trusted party computing f according to the ideal model "type", on input vector (x,y,z), auxiliary input aux given to Adv, and security parameter κ. We call this the (f,type)-hybrid model. Similarly to Definition 4, we say that π computes g with 1-security in the (f,type)-hybrid model if for any adversary Adv there exists a simulator Sim such that $\text{HYBRID}^{f,\mathsf{type}}_{\pi,\mathsf{Adv}(\mathsf{aux})}(\kappa,(x,y,z))$ and $\text{IDEAL}_{g,\mathsf{Sim}(\mathsf{aux})}(\kappa,(x,y,z))$ are computationally indistinguishable.

The sequential composition theorem of Canetti [15] states the following. Let ρ be a protocol that securely computes f in the ideal model "type". Then, if a protocol π computes g in the (f,type)-hybrid model, then the protocol π^ρ, that is obtained from π by replacing all ideal calls to the trusted party computing f with the protocol ρ, securely computes g in the real model.

Theorem 6 ([15]). *Let f be a three-party functionality, let $\mathsf{type}_1,\mathsf{type}_2 \in \{\mathsf{g.o.d.},\mathsf{fair}\}$, let ρ be a protocol that 1-securely computes f with type_1, and let π be a protocol that 1-securely computes g with type_2 in the (f,type_1)-hybrid model. Then, protocol π^ρ 1-securely computes g with type_2 in the real model.*

3 Impossibility Results

In the following section, we present our impossibility results. The main ingredient used in the proofs of these results, is the analysis of a *four-party* protocol that is derived from the three-party protocol assumed to exist. In Section 3.1 we present the four-party protocol alongside some of its useful properties. Most notably, we show that if a functionality f can be computed with 1-security, then f must satisfy a requirement that we refer to as *split-brain simulatability*. Then, in Section 3.2 we show our second impossibility result, where we characterize the class of securely computable functionalities where one of the parties has no input. Finally, in Section 3.3 we present a class of functionalities where the impossibility of computing them follows from privacy.

3.1 The Four-Party Protocol

We start by presenting our first impossibility result that provides necessary conditions for secure computation with respect to the outputs of each *pair* of parties. Therefore, without loss of generality we will state and prove the results with respect to the outputs of A and B.

Fix a three-party protocol $\pi = (\mathsf{A}, \mathsf{B}, \mathsf{C})$ that is defined over secure point-to-point channels in the plain model (without a broadcast channel or trusted setup assumptions). Consider the split-brain attacker controlling C that interacts with A on input z_1 as if never receiving messages from B and interacts with B on input z_2 as if never receiving messages from A. The impact of this attacker can be emulated towards B by a corrupt A and towards A by a corrupt B. A nice way to formalize this argument is by considering a four-party protocol, where two different parties play the role of C. The first interacts only with A, and the second interacts only with B. The four-party protocol is illustrated in Figure 2 in the Introduction.

Definition 5 (the four-party protocol). *Given a three-party protocol $\pi = (\mathsf{A}, \mathsf{B}, \mathsf{C})$ we denote by $\pi' = (\mathsf{A}', \mathsf{B}', \mathsf{C}'_{\mathsf{A}}, \mathsf{C}'_{\mathsf{B}})$ the following four-party protocol. Party A' is set with the code of A, Party B' with the code of B, and parties C'_{A} and C'_{B} with the code of C.*

The communication network of π' is a path. Party A' is connected to C'_{A} and to B', and party B' is connected to A' and to C'_{B}. In addition to its edge to A', party C'_{A} has a second edge that leads to a sink that only receives messages and does not send any message (this corresponds to the channel to B in the code of C'_{A}). Similarly, in addition to its edge to B', party C'_{B} has a second edge that leads to a sink.

We now formalize the above intuition, showing that an honest execution in π' can be emulated in π by any corrupted party. In fact, we can strengthen the above observation. Any adversary in π' corrupting A' and C'_{A}, can be emulated by an adversary in π corrupting A. Similarly, we can emulate any adversary corrupting B' and C'_{B} by an adversary in π corrupting B, and any adversary corrupting C'_{A} and C'_{B} by an adversary in π corrupting C.

Lemma 1 (mapping attackers for π' to attackers for π). *Let $\pi = (\mathsf{A}, \mathsf{B}, \mathsf{C})$ and $\pi' = (\mathsf{A}', \mathsf{B}', \mathsf{C}'_{\mathsf{A}}, \mathsf{C}'_{\mathsf{B}})$ be as in Definition 5. Then*

1. *For every non-uniform* PPT *adversary Adv'_1 corrupting $\{\mathsf{A}', \mathsf{C}'_{\mathsf{A}}\}$ in π', there exists a non-uniform* PPT *adversary Adv_1 corrupting A in π, receiving the input z_1 for C'_{A} as auxiliary information, that perfectly emulates Adv'_1, namely*

$$\left\{\mathrm{REAL}_{\pi,\mathsf{Adv}_1(z_1,\mathsf{aux})}(\kappa,(x,y,z_2))\right\}_{\kappa,x,y,z_1,z_2,\mathsf{aux}} \equiv \left\{\mathrm{REAL}_{\pi',\mathsf{Adv}'_1(\mathsf{aux})}(\kappa,(x,y,z_1,z_2))\right\}_{\kappa,x,y,z_1,z_2,\mathsf{aux}}.$$

2. *For every non-uniform* PPT *adversary Adv'_2 corrupting $\{\mathsf{B}', \mathsf{C}'_{\mathsf{B}}\}$ in π', there exists a non-uniform* PPT *adversary Adv_2 corrupting B in π, receiving the input z_2 for C'_{B} as auxiliary information, that perfectly emulates Adv'_2, namely*

$$\left\{\mathrm{REAL}_{\pi,\mathsf{Adv}_2(z_2,\mathsf{aux})}(\kappa,(x,y,z_1))\right\}_{\kappa,x,y,z_1,z_2,\mathsf{aux}} \equiv \left\{\mathrm{REAL}_{\pi',\mathsf{Adv}'_2(\mathsf{aux})}(\kappa,(x,y,z_1,z_2))\right\}_{\kappa,x,y,z_1,z_2,\mathsf{aux}}.$$

3. *For every non-uniform* PPT *adversary Adv'_3 corrupting $\{\mathsf{C}'_{\mathsf{A}}, \mathsf{C}'_{\mathsf{B}}\}$ in π', there exists a non-uniform* PPT *adversary Adv_3 corrupting C in π, receiving the input z_2 for C'_{B} as auxiliary information,[5] that perfectly emulates Adv'_3, namely*

$$\left\{\mathrm{REAL}_{\pi,\mathsf{Adv}_3(z_2,\mathsf{aux})}(\kappa,(x,y,z_1))\right\}_{\kappa,x,y,z_1,z_2,\mathsf{aux}} \equiv \left\{\mathrm{REAL}_{\pi',\mathsf{Adv}'_3(\mathsf{aux})}(\kappa,(x,y,z_1,z_2))\right\}_{\kappa,x,y,z_1,z_2,\mathsf{aux}}.$$

Proof. We first prove Item 1. The proof of Item 2 is done using an analogous argument and is therefore omitted. Fix an adversary Adv'_1 corrupting $\{\mathsf{A}', \mathsf{C}'_{\mathsf{A}}\}$. Consider the following adversary Adv_1 for π that corrupts A. First, it initializes Adv'_1 with input x for A', input z_1 for C'_{A} and auxiliary information aux. In each round, it ignores the messages sent by C, passes the messages it received from B to Adv'_1 (recall that Adv'_1 internally runs A' and C'_{A}), and replies to B as Adv'_1 replied. Finally, Adv_1 outputs whatever Adv'_1 outputs.

By the definition of Adv_1, in each round, the message it receives from B is identically distributed as the message received from B' in π'. Since it ignores the messages sent from the real C and answers as Adv'_1 does, it follows that the messages it will send to B are identically distributed as well. Furthermore, since Adv_1 does not send any message to C, the view of C will be identically distributed as well to that of C'_{B} in π'. In particular, the joint outputs of B and C in π are identically distributed as the joint outputs of B' and C'_{B} in π', conditioned on the messages received from Adv_1 and Adv'_1 respectively, hence

$$\left\{\mathrm{REAL}_{\pi,\mathsf{Adv}_1(z_1,\mathsf{aux})}(\kappa,(x,y,z_2))\right\}_{\kappa,x,y,z_1,z_2,\mathsf{aux}} \equiv \left\{\mathrm{REAL}_{\pi',\mathsf{Adv}'_1(\mathsf{aux})}(\kappa,(x,y,z_1,z_2))\right\}_{\kappa,x,y,z_1,z_2,\mathsf{aux}}.$$

We next prove Item 3. Fix an adversary Adv'_3 corrupting $\{\mathsf{C}'_{\mathsf{A}}, \mathsf{C}'_{\mathsf{B}}\}$. The adversary Adv_3 corrupts C, initializes Adv'_3 with inputs z_1 and z_2 for C'_{A} and C'_{B} respectively, and auxiliary information aux. Then, in each round it passes the messages received from A and B to Adv'_3 and answers accordingly. Finally, it outputs whatever Adv'_3 outputs. Clearly, the transcript of π when interacting with Adv_3 is identically distributed as the transcript of π'. The claim follows.

[5] The choice of giving Adv_3 the input z_2 as auxiliary is arbitrary.

As a corollary, it follows that if π securely computes some functionality f, then any adversary for π' corrupting $\{\mathsf{A}', \mathsf{C}'_{\mathsf{A}}\}$, or $\{\mathsf{B}', \mathsf{C}'_{\mathsf{B}}\}$, or $\{\mathsf{C}'_{\mathsf{A}}, \mathsf{C}'_{\mathsf{B}}\}$ can be simulated in the ideal-world of f.

Corollary 1. *Let $\pi = (\mathsf{A}, \mathsf{B}, \mathsf{C})$ be a three-party protocol that computes a functionality $f : (\{0,1\}^*)^3 \mapsto (\{0,1\}^*)^3$ with 1-security and let $\pi' = (\mathsf{A}', \mathsf{B}', \mathsf{C}'_{\mathsf{A}}, \mathsf{C}'_{\mathsf{B}})$ be as in Definition 5. Then*

1. *For every adversary Adv'_1 for π' corrupting $\{\mathsf{A}', \mathsf{C}'_{\mathsf{A}}\}$ there exists a simulator Sim_1 in the ideal world of f corrupting A, such that*

$$\left\{\mathrm{IDEAL}_{f,\mathsf{Sim}_1(z_1,\mathsf{aux})}(\kappa,(x,y,z_2))\right\}_{\kappa,x,y,z_1,z_2,\mathsf{aux}} \stackrel{C}{\equiv} \left\{\mathrm{REAL}_{\pi',\mathsf{Adv}'_1(\mathsf{aux})}(\kappa,(x,y,z_1,z_2))\right\}_{\kappa,x,y,z_1,z_2,\mathsf{aux}}.$$

2. *For every adversary Adv'_2 for π' corrupting $\{\mathsf{B}', \mathsf{C}'_{\mathsf{B}}\}$ there exists a simulator Sim_2 in the ideal world of f corrupting B, such that*

$$\left\{\mathrm{IDEAL}_{f,\mathsf{Sim}_2(z_2,\mathsf{aux})}(\kappa,(x,y,z_1))\right\}_{\kappa,x,y,z_1,z_2,\mathsf{aux}} \stackrel{C}{\equiv} \left\{\mathrm{REAL}_{\pi',\mathsf{Adv}'_2(\mathsf{aux})}(\kappa,(x,y,z_1,z_2))\right\}_{\kappa,x,y,z_1,z_2,\mathsf{aux}}.$$

3. *For every adversary Adv'_3 for π' corrupting $\{\mathsf{C}'_{\mathsf{A}}, \mathsf{C}'_{\mathsf{B}}\}$ there exists a simulator Sim_3 in the ideal world of f corrupting C, such that*

$$\left\{\mathrm{IDEAL}_{f,\mathsf{Sim}_3(z_2,\mathsf{aux})}(\kappa,(x,y,z_1))\right\}_{\kappa,x,y,z_1,z_2,\mathsf{aux}} \stackrel{C}{\equiv} \left\{\mathrm{REAL}_{\pi',\mathsf{Adv}'_3(\mathsf{aux})}(\kappa,(x,y,z_1,z_2))\right\}_{\kappa,x,y,z_1,z_2,\mathsf{aux}}.$$

One important use-case of Corollary 1 is when the three adversaries for π' are *semi-honest*. This is due to the fact that the views of the honest parties are identically distributed in all three cases, hence the same holds with respect to their outputs. Next, we consider the distributions over the outputs of A and B in the ideal world of f with respect to each such simulators. Recall that these simulators are for the *malicious* setting, hence they can send arbitrary inputs to the trusted party. Thus, the distributions over the outputs depend on the distribution over the input sent by each simulator to the trusted party. Notice that when considering semi-honest adversaries for π' that have no auxiliary input, these distributions would depend only on the security parameter and the inputs given to the semi-honest adversary. For example, in the case where $\{\mathsf{A}', \mathsf{C}'_{\mathsf{A}}\}$ are corrupted, the simulator samples an input x^* according to some distribution P that depends only on the security parameter κ, the input x, and the input z_1, given to the semi-honest adversary corrupting A' and C'_{A}. We next give a notation for semi-honest adversaries, their corresponding simulators, and the distributions used by the simulators to sample an input value.

Definition 6. *Let $\pi = (\mathsf{A}, \mathsf{B}, \mathsf{C})$ be a three-party protocol that computes a three-party functionality $f : (\{0,1\}^*)^3 \mapsto (\{0,1\}^*)^3$ with 1-security. We let $\mathsf{Adv}^{\mathsf{sh}}_1$ be the semi-honest adversary for π', corrupting $\{\mathsf{A}', \mathsf{C}'_{\mathsf{A}}\}$. Similarly, we let $\mathsf{Adv}^{\mathsf{sh}}_2$ and $\mathsf{Adv}^{\mathsf{sh}}_3$ be the semi-honest adversaries corrupting $\{\mathsf{B}', \mathsf{C}'_{\mathsf{B}}\}$ and $\{\mathsf{C}'_{\mathsf{A}}, \mathsf{C}'_{\mathsf{B}}\}$, respectively. Let Sim_1, Sim_2, and Sim_3 be the three simulators for the* malicious *ideal world of f, that simulate $\mathsf{Adv}^{\mathsf{sh}}_1$, $\mathsf{Adv}^{\mathsf{sh}}_2$, and $\mathsf{Adv}^{\mathsf{sh}}_3$, respectively, guaranteed to exist by Corollary 1.*

We define the distribution $P^{\mathsf{sh}}_{\kappa,x,z_1}$ *to be the distribution over the inputs sent by* Sim_1 *to the trusted party, given the inputs* x, z_1 *and security parameter* κ. *Similarly, we define the distributions* $Q^{\mathsf{sh}}_{\kappa,y,z_2}$ *and* $R^{\mathsf{sh}}_{\kappa,z_1,z_2}$ *to be the distributions over the inputs sent by* Sim_2 *and* Sim_3, *respectively, to the trusted party. Additionally, we let* $\mathcal{P}^{\mathsf{sh}} = \{P^{\mathsf{sh}}_{\kappa,x,z_1}\}_{\kappa\in\mathbb{N},x,z_1\in\{0,1\}^*}$, $\mathcal{Q}^{\mathsf{sh}} = \{Q^{\mathsf{sh}}_{\kappa,y,z_2}\}_{\kappa\in\mathbb{N},y,z_2\in\{0,1\}^*}$, *and* $\mathcal{R}^{\mathsf{sh}} = \{R^{\mathsf{sh}}_{\kappa,z_1,z_2}\}_{\kappa\in\mathbb{N},z_1,z_2\in\{0,1\}^*}$ *be the corresponding distribution ensembles.*

Now, as the simulators simulate the semi-honest adversaries for π', it follows that all the outputs of the honest parties in $\{\mathsf{A},\mathsf{B}\}$ in the executions of the ideal-world computations are the same. This results in a necessary condition that the functionality f must satisfy for it to be securely computable. We call this condition C-*split-brain simulatability*. Similarly, we can define A-split-brain and B-split-brain simulatability to get additional necessary conditions. We next formally define the class of functionalities that are C-split-brain simulatable. We then show that it is indeed a necessary condition.

Definition 7 (C-split-brain simulatability). *Let* $f = (f_1, f_2, f_3)$ *be a three-party functionality and let* $\mathcal{P} = \{P_{\kappa,x,z_1}\}_{\kappa\in\mathbb{N},x,z_1\in\{0,1\}^*}$, $\mathcal{Q} = \{Q_{\kappa,y,z_2}\}_{\kappa\in\mathbb{N},y,z_2\in\{0,1\}^*}$, *and* $\mathcal{R} = \{R_{\kappa,z_1,z_2}\}_{\kappa\in\mathbb{N},z_1,z_2\in\{0,1\}^*}$ *be three ensembles of efficiently samplable distributions over* $\{0,1\}^*$. *We say that* f *is* computationally $(\mathcal{P},\mathcal{Q},\mathcal{R})$-$\mathsf{C}$-split-brain (CSB) simulatable *if*

$$\left\{f_1(x,y^*,z_1)\right\}_{\kappa\in\mathbb{N},x,y,z_1,z_2\in\{0,1\}^*} \overset{C}{\equiv} \left\{f_1(x,y,z^*)\right\}_{\kappa\in\mathbb{N},x,y,z_1,z_2\in\{0,1\}^*}, \text{ and}$$
$$\left\{f_2(x^*,y,z_2)\right\}_{\kappa\in\mathbb{N},x,y,z_1,z_2\in\{0,1\}^*} \overset{C}{\equiv} \left\{f_2(x,y,z^*)\right\}_{\kappa\in\mathbb{N},x,y,z_1,z_2\in\{0,1\}^*},$$

where $x^* \leftarrow P_{\kappa,x,z_1}$, $y^* \leftarrow Q_{\kappa,y,z_2}$, *and* $z^* \leftarrow R_{\kappa,z_1,z_2}$. *We say that* f *is* computationally CSB simulatable, *if there exist three ensembles* $\mathcal{P}$, $\mathcal{Q}$, *and* $\mathcal{R}$ *such that* f *is* $(\mathcal{P},\mathcal{Q},\mathcal{R})$-*CSB simulatable.*

We define *statistically* and *perfectly* CSB simulatable functionalities in a similar way, replacing computational indistinguishability with statistical closeness and equivalence, respectively. In Section 3.1.1 we give several simple examples and properties of CSB simulatable functionalities.

We next prove the main result of this section, asserting that if a functionality is computable with 1-security, then it must be CSB simulatable. We stress that CSB simulatability is *not* a sufficient condition for secure computation. Indeed, the coin-tossing functionality is CSB simulatable, however, as we show in Section 3.2 below, it cannot be computed securely.

Theorem 7 (CSB simulatability – a necessary condition). *If a three-party functionality* $f = (f_1, f_2, f_3)$ *is computable with 1-security over secure point-to-point channels, then* f *is computationally CSB simulatable.*

Proof. Let $\pi = (\mathsf{A},\mathsf{B},\mathsf{C})$ be a protocol that securely realizes f tolerating one malicious corruption. Consider the four-party protocol $\pi' = (\mathsf{A}',\mathsf{B}',\mathsf{C}'_{\mathsf{A}},\mathsf{C}'_{\mathsf{B}})$ from Definition 5, and let $\mathsf{Adv}^{\mathsf{sh}}_1$, $\mathsf{Adv}^{\mathsf{sh}}_2$, and $\mathsf{Adv}^{\mathsf{sh}}_3$, be three *semi-honest* adversaries

corrupting $\{\mathsf{A}', \mathsf{C}'_{\mathsf{A}}\}$, $\{\mathsf{B}', \mathsf{C}'_{\mathsf{B}}\}$, and $\{\mathsf{C}'_{\mathsf{A}}, \mathsf{C}'_{\mathsf{B}}\}$, respectively, with no auxiliary information. We will show that f is $(\mathcal{P}^{\mathsf{sh}}, \mathcal{Q}^{\mathsf{sh}}, \mathcal{R}^{\mathsf{sh}})$-$\mathsf{C}$-split-brain simulatable, where $\mathcal{P}^{\mathsf{sh}}$, $\mathcal{Q}^{\mathsf{sh}}$, and $\mathcal{R}^{\mathsf{sh}}$ are as defined in Definition 6.

We next analyze the output of the honest party B', once when interacting with $\mathsf{Adv}_1^{\mathsf{sh}}$ and once when interacting with $\mathsf{Adv}_3^{\mathsf{sh}}$. The claim would then follow since the view of B', and in particular its output, is identically distributed in both cases. The analysis of the output of an honest A' is similar and therefore is omitted. Let us first focus on the output of B' when interacting with $\mathsf{Adv}_1^{\mathsf{sh}}$. By Corollary 1 there exists a simulator Sim_1 for $\mathsf{Adv}_1^{\mathsf{sh}}$ in the ideal world of f, that is,

$$\left\{\mathrm{IDEAL}_{f,\mathsf{Sim}_1(z_1)}\left(\kappa,(x,y,z_2)\right)\right\}_{\kappa,x,y,z_1,z_2} \overset{\mathrm{C}}{\equiv} \left\{\mathrm{REAL}_{\pi',\mathsf{Adv}_1^{\mathsf{sh}}}\left(\kappa,(x,y,z_1,z_2)\right)\right\}_{\kappa,x,y,z_1,z_2}.$$

In particular, the equivalence holds with respect to the output of the honest party B on the left-hand side, and the output of B' on the right-hand side. Recall that $P^{\mathsf{sh}}_{\kappa,x,z_1}$ is the probability distribution over the inputs sent by Sim_1 to the trusted party. Thus, letting $x^* \leftarrow P^{\mathsf{sh}}_{\kappa,x,z_1}$, it follows that the output of B in the ideal world is identically distributed to $f_2(x^*, y, z_2)$; hence, the output of B' in the four-party protocol is computationally indistinguishable from $f_2(x^*, y, z_2)$. By a similar argument, the output of B' when interacting with $\mathsf{Adv}_3^{\mathsf{sh}}$ is computationally indistinguishable from $f_2(x, y, z^*)$, where $z^* \leftarrow R^{\mathsf{sh}}_{\kappa,z_1,z_2}$, and the claim follows.

3.1.1 Properties of Split-Brain Simulatable Functionalities

Having defined C-split-brain simulatability, we proceed to provide several examples and properties of CSB simulatable functionalities. The proof and further generalizations of these properties appear in the full version [3]. To simplify the presentation, we consider deterministic two-output functionalities $f = (f_1, f_2)$ over a finite domain with $f_1 = f_2$. We denote it as a single functionality $f : \mathcal{X} \times \mathcal{Y} \times \mathcal{Z} \mapsto \mathcal{W}$. Furthermore, we only discuss *perfect* CSB simulatability, and where the corresponding ensembles are independent of κ.

In the Introduction (Section 1.2) we showed that the two-output three-party functionality $f : \{0,1\}^3 \mapsto \{0,1\}$ defined by $f(x,y,z) = (x \wedge y) \oplus z$ is not CSB simulatable. We next state a generalization of this example.

Proposition 1. *Let $f : \mathcal{X} \times \mathcal{Y} \times \mathcal{Z} \mapsto \mathcal{W}$ be a perfectly CSB simulatable two-output three-party functionality. Assume there exist inputs $x \in \mathcal{X}$, $y \in \mathcal{Y}$, and $z_1, z_2 \in \mathcal{Z}$, and two outputs $w_1, w_2 \in \mathcal{W}$, such that $f(x, \cdot, z_1) = w_1$ and $f(\cdot, y, z_2) = w_2$. Then, $w_1 = w_2$.*

We next show that for C-split-brain simulatable functionalities, if a pair of parties P and C, where $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$, can fix the output to be some w, then P can do it by itself. In particular, if C can fix the output to be w, then A and B can do it as well, which implies that f must be 1-dominated.

Proposition 2. *Let $f : \mathcal{X} \times \mathcal{Y} \times \mathcal{Z} \mapsto \mathcal{W}$ be a perfectly $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-CSB simulatable two-output three-party functionality. Assume there exist $x \in \mathcal{X}$, $z \in \mathcal{Z}$, and $w \in \mathcal{W}$ such that $f(x, \cdot, z) = w$. Then, there exists an input $x^* \in \mathcal{X}$ such that $f(x^*, \cdot, \cdot) = w$. Similarly, assuming there exist $y \in \mathcal{Y}$ and $z \in \mathcal{Z}$ such that $f(\cdot, y, z) = w$, then there exists an input $y^* \in \mathcal{Y}$ such that $f(\cdot, y^*, \cdot) = w$.*

3.2 Server-Aided Two-Party Computation

In this section, we consider the server-aided model where two parties – A and B – use the help of an additional *untrusted* yet *non-colluding* server C that has no input in order to securely compute a functionality. The main result of this section is showing that the additional server does not provide any advantage in the secure point-to-point channels model. The proof of Theorem 8 can be found in the full version.

Theorem 8. *Let $f = (f_1, f_2, f_3)$ with $f_1, f_2, f_3 : \{0,1\} \times \{0,1\} \times \emptyset \mapsto \{0,1\}$ be a three-party functionality computable with 1-security over secure point-to-point channels. Then, the two-party functionality*

$$g(x, y) := \Big(\big(f_1(x, y, \lambda), f_3(x, y, \lambda) \big), \big(f_2(x, y, \lambda), f_3(x, y, \lambda) \big) \Big)$$

is computable with 1-security.

As a corollary of Theorem 8, a *two-output* server-aided functionality can be computed with C if and only if it can be computed without it. Formally, we have the following.

Corollary 2. *Let $f = (f_1, f_2)$ with $f_1, f_2 : \{0,1\} \times \{0,1\} \times \emptyset \mapsto \{0,1\}$ be a two-output three-party functionality and let $g(x, y) = (f_1(x, y, \lambda), f_2(x, y, \lambda))$ be its induced two-party variant. Then, f can be computed with 1-security if and only if g can be computed with 1-security.*

Observe that Theorem 8 only provides a necessary condition for secure computation, while Corollary 2 asserts that for two-output functionalities, the necessary condition is also sufficient. In other words, even when the induced two-party functionality g can be securely computed, if f_3 is non-degenerate, then f might not be computable with 1-security. Indeed, consider the functionality $(x, \lambda, \lambda) \mapsto (x, x, \lambda)$. Clearly, it is computable in our setting, however, the functionality where C also receive x is the broadcast functionality, hence it cannot be computed securely.

The proof of Theorem 8 is done by constructing a two-protocol for computing g. We next present the construction. The proof of security is deferred to the full version of the paper [3].

Definition 8 (the two-party protocol). *Fix a protocol $\pi = (\mathsf{A}, \mathsf{B}, \mathsf{C})$ and let $\pi' = (\mathsf{A}', \mathsf{B}', \mathsf{C}'_{\mathsf{A}}, \mathsf{C}'_{\mathsf{B}})$ be the related four-party protocol from Definition 5. We define the two-party protocol $\widehat{\pi} = (\widehat{\mathsf{A}}, \widehat{\mathsf{B}})$ as follows. On input $x \in \{0,1\}^*$,*

party $\widehat{\mathsf{A}}$ *will simulate* A' *holding* x *and* C'_{A} *in its head. Similarly, on input* $y \in \{0,1\}^*$*, party* $\widehat{\mathsf{B}}$ *will simulate* B' *holding* y *and* C'_{B}*. The messages exchanged between* $\widehat{\mathsf{A}}$ *and* $\widehat{\mathsf{B}}$ *will be the same as the messages exchanged between* A' *and* B' *in* π'*, according to their simulated random coins, inputs, and the communication transcript so far. Finally,* $\widehat{\mathsf{A}}$ *will output whatever* A' *and* C'_{A} *output, and similarly,* $\widehat{\mathsf{B}}$ *will output whatever* B' *and* C'_{B} *output.*

Similarly to the four-party protocol, we can emulate any malicious adversary attacking $\widehat{\pi}$ using the appropriate adversary for the three-party protocol π.

Lemma 2 (mapping attackers for $\widehat{\pi}$ to attackers for π). *For any non-uniform* PPT *adversary* $\widehat{\mathsf{Adv}_1}$ *corrupting* $\widehat{\mathsf{A}}$ *in* $\widehat{\pi}$*, there exists a non-uniform* PPT *adversary* Adv_1 *corrupting* A *in* π *that perfectly emulates* $\widehat{\mathsf{Adv}_1}$*. That is, the following two random variables are identically distributed:*

$$\left\{\mathrm{REAL}_{\pi,\mathsf{Adv}_1(\mathsf{aux})}\left(\kappa,(x,y)\right)\right\}_{\kappa,x,y,\mathsf{aux}} \equiv \left\{\mathrm{REAL}_{\widehat{\pi},\widehat{\mathsf{Adv}_1}(\mathsf{aux})}\left(\kappa,(x,y)\right)\right\}_{\kappa,x,y,\mathsf{aux}}. \quad (1)$$

Similarly, for any non-uniform PPT *adversary* $\widehat{\mathsf{Adv}_2}$ *corrupting* $\widehat{\mathsf{B}}$ *in* $\widehat{\pi}$*, there exists a non-uniform* PPT *adversary* Adv_2 *corrupting* B *in* π*, that perfectly emulates* $\widehat{\mathsf{Adv}_1}$*, namely*

$$\left\{\mathrm{REAL}_{\pi,\mathsf{Adv}_2(\mathsf{aux})}\left(\kappa,(x,y)\right)\right\}_{\kappa,x,y,\mathsf{aux}} \equiv \left\{\mathrm{REAL}_{\widehat{\pi},\widehat{\mathsf{Adv}_2}(\mathsf{aux})}\left(\kappa,(x,y)\right)\right\}_{\kappa,x,y,\mathsf{aux}}. \quad (2)$$

3.2.1 Coin-Tossing Protocols Coin-tossing protocols [10] allow a set of parties to agree on uniform common random bit, such that even if some of the parties are malicious, the honest parties output a common bit close to being uniform. The seminal result of Cleve [17] states that unless an honest majority is assumed, for any r-round protocol computing coin-tossing there is always an adversary that can bias the outcome by at least $\Omega\left(1/r\right)$ (even assuming a broadcast channel). In the honest-majority case without broadcast, the result of [19] rules out non-trivial n-party coin-tossing protocols (with any bias $\delta < 1/2$) that tolerate $\lceil n/3 \rceil$ corruptions, as long as *all* parties receive the output.

In this section, we consider three-party two-output coin tossing that tolerates one corruption; the results of [17, 19] do not apply in this case since we assume an honest majority but not all parties learn the output. A direct implication of Corollary 2 together with [17] shows that this variation of coin tossing cannot be computed with negligible bias. Looking further into the proof, it follows that if we are given a protocol that is secure against a malicious C (i.e., C cannot bias the output coin), then Cleve's attackers (on the implied two-party protocol) can be directly translated to attackers for the three-party protocol, corrupting either A or B. Thus, either A or B can bias the output by $\Omega\left(1/r\right)$, where r is the number of rounds in the protocol. However, the above argument does not deal with protocols that allow a corrupt C to slightly bias the output. For example, one might try to construct a protocol where no party (including C) can bias the output by more than $1/r^2$.

We next prove a stronger result, showing that this is impossible. In particular, letting A and B execute the protocol of Moran, Naor, and Segev [36] results with an optimally fair protocol (up to a constant factor). We first formalize the security notion of the task at hand.

Definition 9 (coin-tossing protocol). *Let $\gamma, \delta : \mathbb{N} \mapsto \mathbb{N}$ be such that $\gamma(r) \leq \delta(r)$ for all $r \in \mathbb{N}$, and let $t, \ell, n \in \mathbb{N}$ be such that $t, \ell \leq n$. A polynomial-time n-party protocol $\pi = (\mathsf{P}_1, \ldots, \mathsf{P}_n)$ is a $(\gamma, \delta, t, \ell)$-bias coin-tossing protocol, if the following holds.*

1. *In an honest execution, parties $\mathsf{P}_1, \ldots, \mathsf{P}_\ell$ output a common bit that is γ-close to a uniform random bit. The rest of the parties do not output any value.*
2. *For any PPT adversary corrupting at most t parties, the honest parties in $\{\mathsf{P}_1, \ldots, \mathsf{P}_\ell\}$ output a common bit that is δ-close to a uniform bit.*

We next state that in the three-party case, for every protocol there exists an adversary corrupting a single party that can bias the output by $\Omega(1/r)$. The proof of Lemma 3 can be found in the full version.

Lemma 3. *There exists $c \in \mathbb{R}^+$, for which there is no r-round three-party $(0, 1/cr, 1, 2)$-bias coin-tossing protocol.*

Using a standard player-partitioning argument, we can extend the impossibility of coin-tossing to the n-party case, assuming at least *two* parties receive the output. Specifically, we show that there exists an adversary biasing the output by $\Omega(1/r)$ if at least *two* parties receive the output, and there exists an adversary biasing the output by $1/2 - 2^{-\kappa}$ if at least *three* parties receive the output.

Corollary 3. *Fix $n \in \mathbb{N}$. Then, there is no r-round n-party $(0, 1/cr, \lceil n/3 \rceil, 2)$-bias coin-tossing protocol, where $c \in \mathbb{R}^+$ is as in Lemma 3. Moreover, there is no n-party $(0, 1/2 - 2^{-\kappa}, \lceil n/3 \rceil, 3)$-bias coin-tossing protocol.*

3.3 Impossibility Based on Privacy

In this section we present examples of functionalities that are C-split-brain simulatable yet are not securely computable tolerating one corruption.

Claim 9 *Let $f = (f_1, f_2, f_3)$ with $f_1, f_2, f_3 : \{0,1\}^3 \mapsto \{0,1\}$ be a functionality where A's output is defined as $f_1(x, y, z) = (x \wedge y) \oplus z$. Then f cannot be computed with 1-security.*

The proof of Claim 9 can be found in the full version. Using a similar argument, if we take $f_1(x, y, z) = (x \oplus y) \wedge z$ then f cannot be computed with 1-security.

Claim 10 *Let $f = (f_1, f_2, f_3)$ with $f_1, f_2, f_3 : \{0,1\}^3 \mapsto \{0,1\}$ be a functionality where A's output is defined as $f_1(x, y, z) = (x \oplus y) \wedge z$. Then f cannot be computed with 1-security.*

4 A Class of Securely Two-Output Computable Functionalities

In this section, we present a class of two-output functionalities that can be securely computed over point-to-point channels tolerating a single corruption. This class generalizes previously known feasibility results in this setting, namely, 1-dominated functionalities [18, 19] and fair two-party functionalities [5].

Our result shows that if f is a two-output $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-CSB simulatable functionality, then under a simple condition that a related two-party functionality needs to satisfy, the problem is reduced to the two-party case. Given a two-output three-party functionality f we define the following related two-party functionality. Roughly, in addition to x and y, each of the two parties holds a possible input for C, denoted z_1 and z_2. The real input of C is then chosen according to some predetermined distribution that depends on z_1 and z_2, and output will be whatever is computed by f.

Definition 10. *Let $f : (\{0,1\}^*)^3 \mapsto (\{0,1\}^*)^2$ be a two-output three-party functionality and let $\mathcal{R} = \{R_{\kappa,z_1,z_2}\}_{z_1,z_2\in\{0,1\}^*}$ be an ensemble of efficiently samplable distributions over $\{0,1\}^*$. Define the two-party functionality $h_{\mathcal{R}} : (\{0,1\}^* \times \{0,1\}^*)^2 \mapsto (\{0,1\}^*)^2$ as $h_{\mathcal{R}}((x,z_1),(y,z_2)) = f(x,y,z^*)$, where $z^* \leftarrow R_{\kappa,z_1,z_2}$.*

We now present a sufficient condition for f to be computable with 1-security. Roughly, we require that there exist two distributions, for z_1 and z_2 respectively, such that the input z_1 can be sampled in a way that fixes the distribution of the output of $h_{\mathcal{R}}$ to be independent of z_2, and furthermore, the same holds with respect to z_2. Specifically, we prove the following.

Theorem 11. *Let $f : (\{0,1\}^*)^3 \mapsto (\{0,1\}^*)^2$ be a computationally $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-CSB simulatable two-output three-party functionality, and let $h_{\mathcal{R}} : (\{0,1\}^* \times \{0,1\}^*)^2 \mapsto (\{0,1\}^*)^2$ be the two-party functionality from Definition 10.*

Assume there exist a (randomized) two-party functionality $g : (\{0,1\}^*)^2 \mapsto (\{0,1\}^*)^2$ and two ensembles of efficiently samplable distributions $\mathcal{R}_1 = \{R_{1,\kappa}\}_{\kappa\in\mathbb{N}}$ and $\mathcal{R}_2 = \{R_{2,\kappa}\}_{\kappa\in\mathbb{N}}$ over $\{0,1\}^*$, such that for every $x, y, z \in \{0,1\}^*$, and sufficiently large $\kappa \in \mathbb{N}$, it holds that

$$g(x,y) \quad \equiv \quad h_{\mathcal{R}}((x,z_1),(y,z)) \quad \equiv \quad h_{\mathcal{R}}((x,z),(y,z_2)),$$

where $z_1 \leftarrow R_{1,\kappa}$ and $z_2 \leftarrow R_{2,\kappa}$. Then, f can be computed with 1-security over secure point-to-point channels in the $(g, \mathsf{g.o.d.})$-hybrid model.

Stated differently, if the two-party functionality g can be computed with 1-security then the three-party f can be computed with 1-security as well.

In Section 4.1 below, we give another example of a non-solitary functionality where both A and B receive the same output, that can be securely computed.

We proceed by providing a general construction alongside the necessary and sufficient conditions for security to hold. The proof of Theorem 11 is deferred to the full version.

4.1 The Protocol

We proceed to describe a simple generic protocol $\pi_{\mathcal{R}^*}$ for computing an arbitrary two-output three-party functionality f. The protocol is parametrized by an ensemble of efficiently samplable distributions $\mathcal{R}^* = \{R^*_\kappa\}_{\kappa\in\mathbb{N}}$. Lemma 4 below describes the properties that f and $\mathcal{R}^*$ must satisfy in order for the protocol to be 1-secure.

To illustrate the main ideas behind the protocol and its proof of security, we first give a simple example. Consider the two-output functionality $f = (f_1, f_2)$, where $f_1, f_2 : \{0,1,2\} \times \{0,1\} \times \{0,1\} \mapsto \{0,1\}^2$ given by the following two matrices

$$M_0^{\mathsf{A}} = M_0^{\mathsf{B}} = \begin{pmatrix} 0\,1 \\ 1\,0 \\ 1\,1 \end{pmatrix} \qquad M_1^{\mathsf{A}} = M_1^{\mathsf{B}} = \begin{pmatrix} 1\,0 \\ 0\,1 \\ 1\,1 \end{pmatrix}$$

Stated differently, we let $f_1 \equiv f_2$ and let $M_z^{\mathsf{A}}(x,y) = f_1(x,y,z)$, where A's input determines a row, B's input determines a column, and C's input determines the matrix. The protocol follows similar lines to that of [19]. First, the parties compute f with fairness (this can be done over point-to-point channels by the honest-majority assumption [18]). If the computation fails, then A and B compute the following symmetric randomized two-party functionality

$$\begin{pmatrix} 1/2 & 1/2 \\ 1/2 & 1/2 \\ 1 & 1 \end{pmatrix}$$

That is, if A's input is 2 the output is 1; otherwise, the output is a uniform bit. Note that this two-party functionality can be computed with guaranteed output delivery [5]. Clearly, as B and C have no affect over the distribution of the output of the two-party functionality, any adversary corrupting either party can be simulated by sending a uniformly random bit to the trusted party. The output of the honest parties will be 1 if A inputs 2 and a uniform bit otherwise, regardless of the other honest party's input – the same distribution is induced by the simulator for the two-party functionality. To simulate a corrupt A in the three-party protocol, the simulator will send input 2 with the same probability p that the two-party simulator sends input 2 to the trusted party; otherwise, the simulator will send a uniform random bit. Observe that regardless of the input of C, the output of B will be 1 with probability $\frac{1}{2}(1+p)$. Similarly to the previous cases, this is the same distribution as the one induced by the simulator for the two-party functionality.

We now generalize the above ideas. We next present the protocol in the $\{(f, \mathsf{fair}), (f_{\mathcal{R}^*}, \mathsf{g.o.d.})\}$-hybrid model.

Protocol 12 ($\pi_{\mathcal{R}^*}$)

Private input: party A *holds* $x \in \{0,1\}^*$*, party* B *holds* $y \in \{0,1\}^*$*, and party* C *holds* $z \in \{0,1\}^*$*.*

Common input: the parties hold the security parameter 1^κ.

1. *Each party invokes* (f, fair) *with its input. Let* w_1 *be the output of* A *and* w_2 *the output of* B.
2. *If* $w_1, w_2 \neq \perp$ *then* A *outputs* w_1 *and* B *outputs* w_2.
3. *Otherwise,* A *and* B *invoke* $(f_{\mathcal{R}^*}, \mathsf{g.o.d.})$ *on their inputs* x *and* y*, respectively, and output the result.*

We next intuitively explain the properties that f and $\mathcal{R}^*$ must satisfy for $\pi_{\mathcal{R}^*}$ to be secure. Since there are no messages exchanged between the parties, constructing a simulator amounts to defining an appropriate distribution over the inputs of the corrupted parties. In particular, simulating a corrupt C can be easily simulated by either sending the input it used when calling (f, fair), or sampling according to R^*_κ. Next, consider a corrupted A or B. Similarly to the above example, we first take the distribution given by the simulator for the two-party functionality $f_{\mathcal{R}^*}$, and construct a distribution for the three-party functionality f, so that the outputs of the honest party in both ideal-worlds are identically distributed, regardless of C's inputs. That is, A and B can each sample an input for the three-party functionality f in such a way that the distribution over the output is the same as if C sampled its input according to R^*_κ.

Lemma 4. *Let* $f : (\{0,1\}^*)^3 \mapsto (\{0,1\}^*)^2$ *be a two-output three-party functionality, and let* $\mathcal{R}^*$ *be an ensemble of efficiently samplable distributions over* $\{0,1\}^*$. *Assume that the following holds.*

1. *There exists an ensemble of efficiently samplable distributions* $\mathcal{P}^* = \{P^*_{\kappa,x}\}_{\kappa\in\mathbb{N},x\in\{0,1\}^*}$ *over* $\{0,1\}^*$ *such that*

$$\left\{ f_{\mathcal{R}^*,2}\left(x,y\right) \right\}_{\kappa,x,y,z} \stackrel{C}{\equiv} \left\{ f_2\left(x^*,y,z\right) \right\}_{\kappa,x,y,z},$$

   where $x^* \leftarrow P^*_{\kappa,x}$.
2. *There exists an ensemble of efficiently samplable distributions* $\mathcal{Q}^* = \{Q^*_{\kappa,y}\}_{\kappa\in\mathbb{N},y\in\{0,1\}^*}$ *over* $\{0,1\}^*$ *such that*

$$\left\{ f_{\mathcal{R}^*,1}\left(x,y\right) \right\}_{\kappa,x,y,z} \stackrel{C}{\equiv} \left\{ f_1\left(x,y^*,z\right) \right\}_{\kappa,x,y,z},$$

   where $y^* \leftarrow Q^*_{\kappa,y}$.

Then, $\pi_{\mathcal{R}^*}$ *computes* f *with 1-security in the* $\{(f, \mathsf{fair}), (f_{\mathcal{R}^*}, \mathsf{g.o.d.})\}$*-hybrid model.*

The proof of Lemma 4 can be found in the full version.

Bibliography

[1] S. Agrawal and M. Prabhakaran. On fair exchange, fair coins and fair sampling. In *CRYPTO'13, part I*, pages 259–276, 2013.

[2] B. Alon and E. Omri. Almost-optimally fair multiparty coin-tossing with nearly three-quarters malicious. In *TCC'16-B, part I*, pages 307–335, 2016.

[3] B. Alon, R. Cohen, E. Omri, and T. Suad. On the power of an honest majority in three-party computation without broadcast. Cryptology ePrint Archive, Report 2020/1170, 2020. https://eprint.iacr.org/2020/1170.

[4] G. Asharov. Towards characterizing complete fairness in secure two-party computation. In *TCC'14*, pages 291–316, 2014.

[5] G. Asharov, A. Beimel, N. Makriyannis, and E. Omri. Complete characterization of fairness in secure two-party computation of Boolean functions. In *TCC'15, part I*, pages 199–228, 2015.

[6] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures (extended abstract). In *EUROCRYPT'98*, pages 591–606, 1998.

[7] A. Beimel, E. Omri, and I. Orlov. Protocols for multiparty coin toss with a dishonest majority. *J. Cryptology*, 28(3):551–600, 2015.

[8] A. Beimel, I. Haitner, N. Makriyannis, and E. Omri. Tighter bounds on multiparty coin flipping via augmented weak martingales and differentially private sampling. In *FOCS*, pages 838–849, 2018.

[9] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[10] M. Blum. Coin flipping by telephone. In *CRYPTO'81*, pages 11–15, 1981.

[11] M. Borderding. Levels of authentication in distributed agreement. In *10th International Workshop on Distributed Algorithms WDAG*, pages 40–55, 1996.

[12] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985.

[13] N. Buchbinder, I. Haitner, N. Levi, and E. Tsfadia. Fair coin flipping: Tighter analysis and the many-party case. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2580–2600, 2017.

[14] C. Cachin and J. Camenisch. Optimistic fair secure computation. In *CRYPTO'00*, pages 93–111, 2000.

[15] R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[16] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC*, pages 11–19, 1988.

[17] R. Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369, 1986.

[18] R. Cohen and Y. Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. *J. Cryptology*, 30(4):1157–1186, 2017.

[19] R. Cohen, I. Haitner, E. Omri, and L. Rotem. Characterization of secure multiparty computation without broadcast. *J. Cryptology*, 31(2):587–609, 2018.

[20] R. Cohen, I. Haitner, E. Omri, and L. Rotem. From fairness to full security in multiparty computation. In *Proceedings of the 11th Conference on Security and Cryptography for Networks (SCN)*, pages 216–234, 2018.

[21] C. Dwork, N. A. Lynch, and L. J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[22] M. J. Fischer, N. A. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.

[23] M. Fitzi, N. Gisin, U. M. Maurer, and O. von Rotz. Unconditional byzantine agreement and multi-party computation secure against dishonest minorities from scratch. In *EUROCRYPT'02*, pages 482–501, 2002.

[24] M. Fitzi, D. Gottesman, M. Hirt, T. Holenstein, and A. D. Smith. Detectable byzantine agreement secure against faulty majorities. In *PODC*, pages 118–126, 2002.

[25] M. Fitzi, J. A. Garay, U. M. Maurer, and R. Ostrovsky. Minimal complete primitives for secure multi-party computation. *J. Cryptology*, 18(1):37–61, 2005.

[26] J. A. Garay, A. Kiayias, R. M. Ostrovsky, G. Panagiotakos, and V. Zikas. Resource-restricted cryptography: Revisiting MPC bounds in the proof-of-work era. In *EUROCRYPT'20, part II*, pages 129–158, 2020.

[27] O. Goldreich. *Foundations of Cryptography – VOLUME 2: Basic Applications*. Cambridge University Press, 2004.

[28] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[29] S. D. Gordon, C. Hazay, J. Katz, and Y. Lindell. Complete fairness in secure two-party computation. In *STOC*, pages 413–422, 2008.

[30] I. Haitner and E. Tsfadia. An almost-optimally fair three-party coin-flipping protocol. *SICOMP*, 46(2):479–542, 2017.

[31] I. Haitner, N. Makriyannis, and E. Omri. On the complexity of fair coin flipping. In *TCC'18, part I*, pages 539–562, 2018.

[32] S. Halevi, Y. Ishai, E. Kushilevitz, N. Makriyannis, and T. Rabin. On fully secure MPC with solitary output. In *TCC'19, part I*, pages 312–340, 2019.

[33] J. Kilian. Founding cryptography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[34] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3): 382–401, 1982.

[35] N. Makriyannis. On the classification of finite Boolean functions up to fairness. In *Proceedings of the 9th Conference on Security and Cryptography for Networks (SCN)*, pages 135–154, 2014.

[36] T. Moran, M. Naor, and G. Segev. An optimally fair coin toss. *J. Cryptology*, 29 (3):491–513, 2016.

[37] M. C. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[38] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *FOCS*, pages 73–85, 1989.

[39] A. C. Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982.