On the Exact Round Complexity of Self-Composable Two-Party Computation

Sanjam Garg[1⋆], Susumu Kiyoshima[2], Omkant Pandey[3]

[1] University of California, Berkeley, USA
`sanjamg@berkeley.edu`
[2] NTT Secure Platform Laboratories, Tokyo, Japan
`kiyoshima.susumu@lab.ntt.co.jp`
[3] Stony Brook University, Stony Brook, USA
`omkant@cs.stonybrook.edu`

Abstract. The round complexity of secure computation has been a fundamental problem in cryptography. Katz and Ostrovsky proved that 5 rounds are both necessary and sufficient for secure computation in the stand alone setting, thus resolving the *exact* round complexity of *standalone* secure computation.

In contrast, round complexity of secure computation in the *concurrent* setting, where several protocols may run simultaneously, is poorly understood. Since standard polynomial time simulation is impossible in the concurrent setting, alternative security notions have been proposed, e.g., *super-polynomial simulation* (SPS). While SPS security can be achieved in constant rounds, the actual constant (> 20) is far from optimal.

In this work, we take the first steps towards studying the exact round complexity of concurrent secure computation. We focus on the two party case and present a new secure computation protocol that achieves SPS security under concurrent self-composition. Our protocol has 5 rounds assuming quasi-polynomially-hard injective one-way functions (or 7 rounds assuming standard polynomially-hard collision-resistant hash functions). We also require other standard assumptions, specifically trapdoor OWPs and lossy TDFs. This matches the rounds for standalone secure computation.

More specifically, our security proof presents a *polynomial time* reduction from SPS security to 3-round public-coin non-malleable commitments with appropriate extractability properties. Such commitments are known based on quasi-polynomially-hard injective OWFs. (The reduction also works with a special 6-round non-malleable commitment to yield the 7-round result under CRHFs.)

⋆ Research supported in part from DARPA/ARL SAFEWARE Award W911NF15C0210, AFOSR Award FA9550-15-1-0274, NSF CRII Award 1464397, AFOSR YIP Award and research grants by the Okawa Foundation and Visa Inc. The views expressed are those of the author and do not reflect the official policy or position of the funding agencies.

1 Introduction

Secure computation protocols are protocols that enable mutually distrustful parties to compute a functionality without compromising the correctness of the outputs and the privacy of their inputs. Secure computation protocols have been studied in both two-party case and multi-party case, and it was shown that secure computation protocols for any functionality can be constructed in both cases in a model with malicious adversaries and a dishonest majority [36, 12].

The security of secure computation protocols is defined by using simulation paradigm. Specifically, to define the security of a protocol π for computing a function f, we consider the *real world*, where the parties compute f by executing π, and the *ideal world*, where the parties compute f by interacting with a trusted third party. Then, we define the security by requiring that for any adversary in the real world there exists a *simulator* in the ideal world such that whatever an adversary can do in the real world can be "simulated" in the ideal world by the simulator.

Round complexity of secure computation. A fundamental question in this area is to understand how many rounds are necessary and sufficient for securely computing general functionalities. Katz and Ostrovsky [18] proved that five rounds are both necessary and sufficient for secure two-party computation in the *standalone* setting where there is only one protocol execution. These results were further extended in [30] and [11] w.r.t. black-box constructions and simultaneous message channels. These results completely settle the round complexity of two-party computation in the *standalone* setting.

The concurrent setting. While standalone security is sufficient for many applications, other situations (such as protocol execution over the Internet) require stronger notions of security. This setting where there may be many protocols executions at the same time, is called the *concurrent* setting. Unfortunately, it is known that stand-alone security does not necessarily imply security in the concurrent setting [8].

Secure computation in the concurrent setting is more challenging to define than the standalone setting. Canetti [4] proposed the notion of *universally composable* (UC) security where protocols maintain their strong simulation based security guarantees even in the presence of other arbitrary protocols. However, achieving UC-security in the plain model turned out to be impossible [4, 5]. Moreover, Lindell [25, 26] proved that even in the special case where only instantiations of the *same* protocol are allowed, standard notion of polynomial-time simulation is impossible to achieve. (This is called "self composition" and corresponds to the setting we are interested in.)

These strong negative results motivated the study of alternative notions for concurrent secure computation, such as *super-polynomial-time simulation* (SPS) security (and the closely related *angel-based* security), input-indistinguishable computation, bounded concurrent composition, and multiple ideal-query model [31, 35, 32, 2, 29, 28, 6, 10, 23, 33, 13, 20, 19, 14].

In this work we focus on SPS security in the two-party setting. In SPS security, the simulator is allowed to run in super-polynomial time; thus, SPS security guarantees that whatever an adversary can do in the real world can also be done in the ideal world *in super-polynomial time*. Although allowing the simulator to run in super-polynomial time weakens the security guarantee, SPS security still guarantees meaningful security for many functionalities. Furthermore, it was shown that under SPS security, concurrent self-composition can be achieved in the plain model. (In what follows, by SPS security we mean SPS-security under concurrent self-composition.)

SPS security has been extensively studied and improved upon in the literature. Prabhakaran and Sahai [35] provided the initial positive result for SPS security. Although, these early results [35, 28] relied on non-standard/subexponential-time assumptions, Canetti, Lin and Pass achieved this (actually, the angel-based) notion under standard polynomial-time assumptions [6] in a polynomial number of rounds. Soon after, Garg et al. [10] presented a *constant round* SPS-secure protocol, thus resolving the *asymptotic* round-complexity of SPS-secure computation (under polynomially-hard assumptions).

Exact round complexity of SPS-secure computation. Although the SPS-secure protocol of [10] has asymptotically constant rounds, its exact round complexity is actually quite large (more than 20). In contrast, the standalone setting only requires five rounds [18]. Is this gap necessary? What is the exact round complexity of SPS-secure protocols for computing general functionalities? To the best of our knowledge, these questions have not been explored before.

1.1 Our Results

In this work, we take the first steps towards studying the exact round complexity of concurrent secure computation. We present a new secure computation protocol whose round complexity matches that of the stand alone setting. More specifically, we present a *five-round* SPS-secure two-party computation protocol. Our protocol guarantees security under concurrent self-composition.

We are interested in basing the security of our protocol on standard, polynomially-hard, assumptions. We do this by providing a *polynomial-time* reduction that reduces the SPS-security of our protocol to that of 3-round public-coin non-malleable commitments with some natural extractability properties. In particular, we want 3-round non-malleable commitments that are extractable *without over-extraction* [19].

One caveat is that such non-malleable commitments, at present, are only known to exist under quasi-polynomially-hard injective OWFs [16].[1] Consequently, we only achieve a result under quasi-polynomially hard injective OWFs. We remark that even under super-polynomially hard assumptions, pre-

[1] The 3-round non-malleable commitment of [16] was claimed to be secure under polynomially-hard injective OWFs; however, the public-coin variant of their scheme is proven secure only under quasi-polynomially-hard injective OWFs (see the latest ePrint version [15]).

vious SPS-secure protocols have quite large round complexity (more than 20) [2, 24, 33, 20, 14].

While existence of quasi-polynomially-hard injective OWFs is considered a standard assumption, it would be interesting to know if we can rely only on *polynomially*-hard assumptions. Towards this goal, we realize that our construction actually works with a special 6-round non-malleable commitment scheme based on (polynomially-hard) CRHFs. This gives us a 7-round SPS-secure protocol for general functionalities where all underlying assumptions are only polynomially-hard.

1.2 Overview of Techniques

Our overall strategy is to apply the techniques of the constant-round SPS-secure protocol of Garg et al. [10] to the five-round secure two-party computation protocol of Katz and Ostrovsky [18]. In this subsection, we first recall the techniques of Garg et al. and explain the difficulty in applying the techniques of Garg et al. to the protocol of Katz and Ostrovsky. After that, we give an overview of our techniques.

SPS protocol of Garg et al. Like other SPS protocols, the concurrently SPS-secure multi-party computation protocol of Garg et al. [10] has "trapdoor secrets" that enable simulation,[2] and the simulator obtains the trapdoor secrets by breaking cryptographic primitives by brute-force in super-polynomial time. The main technical challenge in the proof of security is to design a *polynomial-time* reduction that reduces the security of the protocol to the security of underlying cryptographic primitives. In fact, since the simulator runs in super-polynomial time, a naive approach that having the reduction emulate the simulator internally can only result in super-polynomial-time reductions.

To obtain a polynomial-time reduction in the proof of security, Garg et al. consider a hybrid experiment in which the brute-force extraction of the trapdoor secrets is replaced with polynomial-time rewinding extraction. With such a hybrid experiment, Garg et al. designs a security proof roughly as follows.

1. First, the indistinguishability between the real and the hybrid experiment is reduced to the security of various protocol components. The reductions run in polynomial time since both the real and the hybrid experiment run in polynomial time.
2. Next, the indistinguishability between the hybrid and the ideal experiment is shown without relying on any cryptographic assumptions. No cryptographic assumption is needed to show this indistinguishability since the two experiments differ only in how the trapdoor secrets are extracted and anyway the same trapdoor secrets are extracted in both experiments except with negligible probability.

[2] Concretely, the trapdoor secrets enable the simulator to give "proofs of correct behavior" while executing the protocol incorrectly.

However, since the protocol is executed in the concurrent setting, the use of rewinding extraction causes problems.

The first problem is that rewinding can become recursive in the concurrent setting, which often leads to the necessity of large round complexity of the protocol. Recall that rewinding extraction typically requires the creation of "look-ahead threads," i.e., rewinding the adversary and interacting with it again from an earlier point of the protocol. If the simulator is required to do simulation even on the look-ahead threads, the rewinding can become recursive—if the adversary starts new sessions on look-ahead threads, the simulator need to extract the trapdoor secrets from these newly started sessions, and thus, need to rewind the adversary recursively. A key observation by Garg et al. is that, since the look-ahead threads are created only in the hybrid experiment, the simulator does not need to do "full simulation" on the look-ahead threads. More precisely, Garg et al. observe that in the hybrid experiment, the simulator can use the secret inputs of the honest party to execute newly started sessions honestly on the look-ahead threads, by which the simulator can avoid rewinding the adversary recursively. (The secret inputs of the honest parties are used only on the look-ahead threads, and they are never used on the "main thread.")

The second problem is that the components of the protocol can be rewound in the hybrid experiment due to the rewinding extraction of the trapdoor secrets, which makes it hard to show the indistinguishability between the real and the hybrid experiment. Specifically, since any component in a session can be rewound due to the rewinding extraction of other sessions, and the security of a cryptographic primitive is in general not preserved when it is rewound, it is not clear if the indistinguishability of the real and the hybrid experiment can really be reduced to the security of the components. Garg et al. solved this problem by carefully designing their protocol and a sequence of intermediate hybrid experiments. Specifically:

1. They define the sequence of intermediate hybrids between the real and the hybrid experiment in such a way that the concurrent sessions are switched from honestly executed ones to simulated ones session by session in the order of their $\mathsf{special\ messages}$—namely, the messages such that the look-ahead threads are created from the rounds of these messages.[3] Switching in this order guarantees that in each intermediate hybrid, rewinding occurs only until $\mathsf{special\ message}$ of the session that has just been switched to simulation.
2. Then, they design the protocol in such a way that all the "rewinding-insecure" components (namely, the components whose security is not preserved when they are rewound) start only after $\mathsf{special\ message}$ of the protocol. A key point is that when the protocol is designed in this way, it is guaranteed that in an intermediate hybrid where a session is switched to simulation (and therefore rewinding occurs only until $\mathsf{special\ message}$ of this session), all the rewinding-insecure components in this session are not rewound and therefore their security can be used in the proof of indistinguishability.

[3] In the actual security proof in [10], the sessions are switched to honestly executed ones in a more complex manner since each session has *two* $\mathsf{special\ messages}$.

Applying techniques of Garg et al. to Katz-Ostrovsky two-party protocol. Unfortunately, the techniques of Garg et al. cannot be applied on the round-optimal two-party secure computation protocol of Katz and Ostrovsky (KO) [18] in a straightforward manner.

The main difficulty is that in the KO protocol, the techniques of Garg et al. is not helpful to solve the second problem described above, i.e., the problem that the components of the protocol can be rewound in the hybrid experiment. Recall that Garg et al. solve this problem by designing their protocol in such a way that the rewinding-insecure components start only after special message. In the KO protocol, however, some components are executed in parallel to compress the round complexity and therefore a rewinding-insecure component starts before special message.

To see the difficulty in more details, let us first recall the KO protocol. (In this overview, we concentrate on the setting where only one party obtains the output. In this setting, the KO protocol has only four rounds.) Roughly speaking, the KO protocol is a semi-honest secure two-party computation protocol that is augmented with proofs of correct behavior. Since the protocol has only four rounds, these proofs are executed somewhat in parallel: One party, P_1, gives a proof in Rounds 1–3 and the other party, P_2, gives in Rounds 1–4. Also, these proofs have the proof-of-knowledge property (and thus are rewinding insecure) and the simulator can extract the implicit input of the adversary from them. When extracting the implicit input, the simulator rewinds the adversary in the last two rounds of the proof; hence, when P_1 is corrupted, special message is the message in Round 2 (since look-ahead threads are created from Round 2), and when P_2 is corrupted, special message is the message in Round 3 (since look-ahead threads are created from Round 3). Notice that when P_2 is corrupted, the proof by P_1 in Rounds 1–3 is executed before special message in Round 4.

Then, the difficulty is the following. Let us consider that we design a sequence of intermediate hybrids following the approach of Garg et al. In the intermediate hybrids, all we can guarantee is that when a session is switched to simulation, no rewinding occurs after special message of this session—hence, when P_2 is corrupted, we can only guarantee that no rewinding occurs after Round 4, and thus, cannot guarantee that the proof by P_1 in Rounds 1–3 is not rewound in this session. Then, since the simulation of the KO protocol is indistinguishable only when the proof by P_1 is secure, it seems hard to prove the indistinguishability among the intermediate hybrids unless the proof by P_1 is rewinding secure. Furthermore, since we require that the proof by P_1 has the proof-of-knowledge property, it seems unlikely that the proof by P_1 can be rewinding secure.

Our techniques. To solve the problem that the components of the protocol are rewound in the hybrid experiment, we use the fact that, as observed by Garg et al., in the SPS setting the look-ahead threads can depend on the inputs of the honest parties. Specifically, we use the fact that we do not need to remove the input of P_1 from the proof of correct behavior on the look-ahead threads.

First, we recall the KO protocol in more details. For simplicity, we assume that each party has only 1-bit input. In this case, P_1 gives the proof of correct behavior using a witness indistinguishable proof of knowledge Π_{WIPOK} for a statement of the form $\mathsf{st}_0 \vee \mathsf{st}_1$, where in the honest execution, only one of st_0 and st_1 is true depending on the input of P_1. In simulation, in a session where P_2 is corrupted, the simulator makes both st_0 and st_1 true and simulates the proof of correct behavior using a witness for st_0. (Notice that the proof no longer depends on P_1's input.) In a session where P_1 is corrupted, the simulator extracts the implicit input of the adversary by extracting a witness from Π_{WIPOK} and checking whether the extracted witness is a witness for st_0 or not.

Then, our idea is the following. We replace Π_{WIPOK} with other cryptographic components—witness-indistinguishable one and extractable one—so that we can have both rewinding-secure witness-indistinguishability and extractability. Specifically, the components we use are:

- **a ZAP system** (namely, a two-round public-coin witness indistinguishable proof system). Since ZAP has only two rounds, it is witness indistinguishable even when it is rewound.
- **a three-round honest-committer extractable commitment** (namely, a commitment scheme such that, as long as the committer behaves honestly, the committed value can be extracted by rewinding the committer). The honest-committer extractable scheme that we use, denoted by ExtCom', is a variant of a three-round challenge-response based extractable scheme. To commit to a message m using ExtCom', the committer commits to many 2-out-of-2 secret shares $\{(\alpha_i^0, \alpha_i^1)\}$ of m using a standard non-interactive commitment scheme in the first round, and after receiving challenge $\{e_i\}$ from the receiver, the committer reveals $\{\alpha_i^{e_i}\}$ in the third round *but does not open the corresponding commitments.* An important property of ExtCom' is that the committer's messages in the first and the third round can be simulated independently of each other. In particular, we can simulate a commitment by committing to all-zero strings in the first round and sending random strings in the last round. (Later, we use this property to say that even though ExtCom' is extractable, it also has some rewinding security.)

We then modify the KO protocol in such a way that P_1 gives two ExtCom' commitments in Rounds 1–3, where one is correctly constructed and the other is simulated, and then proves by ZAP in Rounds 2–3 that either a witness for st_0 is committed in the first ExtCom' commitment or a witness for st_1 is committed in the second one. (Recall that only one of st_0 and st_1 is true in the KO protocol depending on the input of P_1) With this modification, we can solve the problem as follows. When P_2 is corrupted, the simulator makes both st_0 and st_1 true (as the KO simulator does), commits to witnesses for st_0 and st_1 in the two ExtCom' commitments, and completes the ZAP proof using a witness for that st_0 is committed in the first ExtCom' commitment. Then, even though ZAP and ExtCom' can be rewound in the hybrid experiments, we can show the indistinguishability using their security for the following reasons.

1. First, the simulator can switch a simulated ExtCom' commitment to a honest one in an indistinguishable way even under rewinding *as long as the commitment does not need to be a honest one on the look-ahead threads.* This is

because the third message of the simulated commitment consists of just random strings; we can design a reduction that obtains a ExtCom' commitment (either a simulated one or a honest one) on the main thread from an external committer while internally simulating the look-ahead threads by simulating the third message of this ExtCom' commitment with random strings.

2. Second, the witness indistinguishability of ZAP holds even when it is rewound. This is because it has only two rounds.

On the other hand, when P_1 is corrupted, the simulator can extract the implicit input from the adversary by extracting the committed values from the two ExtCom' commitments and checking whether a witness for st_0 or st_1 is extracted. Even though ExtCom' is only honest-committer extractable, the simulator can extract the implicit input in this way since the soundness of ZAP guarantees that at least one of the ExtCom' commitments is constructed correctly.

Other technicalities To prove security formally, we need to modify the KO protocol further.

First, we need to add non-malleability to the KO protocol because in the concurrent setting with interchangeable roles, the adversary can participate as the first party in a session while participating as the second party in another session. To add non-malleability, we use a non-malleable commitment in a similar manner as Barak et al. [1], who constructed a concurrent non-malleable zero-knowledge argument using a non-malleable commitment. In particular, we modify the KO protocol in such a way that, instead of giving a proof of correct behavior, a party commits to a witness for the correct behavior using a non-malleable commitment and then proves that it committed to a valid witness. As in the protocol of Barak et al. [1], we assume that the non-malleable commitment is extractable and that some components of our protocol are statistically secure. (Roughly, this is for guaranteeing that the non-malleable commitment is non-malleable w.r.t. not only itself but also the other components of our protocol.)

Second, for technical reasons, we augment the KO protocol with a lossy encryption scheme, i.e., an encryption scheme that has a lossy key generation algorithm such that lossy keys statistically hide the plaintexts. Roughly speaking, this is because unlike the SPS-secure protocol of Garg et al. [10], the KO protocol does not have the property that the same information is extracted by rewinding extraction and by brute-force extraction. (Recall that this property is required when the indistinguishability between the hybrid and the ideal experiment is shown.) Specifically, an adversary can make the rewinding simulator obtain a valid implicit input whereas the brute-force simulator obtain an invalid one. We therefore modify the KO protocol so that for such an adversary, all the messages that depend on the extracted implicit input are encrypted under a lossy key (whereas they are encrypted under a normal key in the honest execution).

2 Preliminaries

In this paper, we denote the security parameter by κ. We assume familiarity with the definitions of basic cryptographic schemes and protocols, such as secret-key/public-key encryption schemes, message authentication codes, commitment schemes, and witness-indistinguishable proof/argument of knowledge. We remind the reader that there exists a non-interactive perfectly binding commitment scheme under the existence of injective one-way functions, and there exists a two-round statistically hiding commitment scheme under the existence of collision-resistance hash functions.

2.1 Components of Katz-Ostrovsky 2-Party Computation

We recall the secure two-party computation protocol of Katz and Ostrovsky [18] and its components. Part of the text is taken from [18, 11].

Semi-honest two-party computation based on Yao's garbled circuits. We first recall that a semi-honest secure two-party computation protocol can be constructed using Yao's garbled circuit scheme [36, 27].

We view Yao's garbled circuit scheme as a tuple of PPT algorithms $(\mathsf{GenGC}, \mathsf{EvalGC})$, where GenGC is the "generation procedure" that generates a garbled circuit for a circuit C along with "labels," and EvalGC is the "evaluation procedure" that evaluates the circuit on the "correct" labels. Each individual wire i of the circuit is assigned two labels, $Z_{i,0}, Z_{i,1}$. More specifically, the two algorithms have the following format (here $i \in [\kappa], b \in \{0,1\}$).

– $(\mathsf{GC}_y, \{Z_{i,b}\}) \leftarrow \mathsf{GenGC}(1^\kappa, F, y)$: GenGC takes as input a security parameter κ, a circuit F, and a string $y \in \{0,1\}^\kappa$. It outputs a *garbled circuit* GC_y along with the set of all *input-wire labels* $\{Z_{i,b}\}$.
– $v = \mathsf{EvalGC}(\mathsf{GC}_y, \{Z_{i,x_i}\})$: Given a garbled circuit GC_y and a set of input-wire labels $\{Z_{i,x_i}\}$, where $x = x_1 x_2 \cdots x_\kappa \in \{0,1\}^\kappa$, EvalGC outputs either an invalid symbol $\perp$ or a value $v = F(x,y)$.

The two algorithms have the following properties.

– **Correctness:** $\Pr[\mathsf{EvalGC}(\mathsf{GC}_y, \{Z_{i,x_i}\}) = F(x,y)] = 1$ for all F, x, y, taken over the correct generation of $\mathsf{GC}_y, \{Z_{i,b}\}$ by GenGC.
– **Security:** There exists a PPT simulator SimGC such that for any F, we have $\{(\mathsf{GC}_y, \{Z_{i,x_i}\})\}_{x,y} \approx_{\mathrm{c}} \{\mathsf{SimGC}(1^\kappa, F, v)\}_{x,y}$, where $(\mathsf{GC}_y, \{Z_{i,b}\}) \leftarrow \mathsf{GenGC}(1^\kappa, F, y)$ and $v = F(x,y)$.

Yao's garbled circuit scheme is based on the existence of one-way functions.

Using Yao's garbled circuit scheme and a semi-honest OT protocol, two parties, P_1 and P_2, can compute a function F of their inputs in the semi-honest setting as follows. Let x, y be the inputs of P_1, P_2 respectively. Consider the setting that only one party, say P_1, learns the output of F. Then, P_2 first computes $(\mathsf{GC}_y, \{Z_{i,b}\}) \leftarrow \mathsf{GenGC}(1^\kappa, F, y)$ and sends GC_y to P_1. The two parties then engage in κ parallel instances of OT, where in the i-th instance, P_1 inputs x_i and P_2 inputs $(Z_{i,0}, Z_{i,1})$ to the OT protocol, and P_1 learns Z_{i,x_i}. Then, P_1 computes $v = \mathsf{EvalGC}(\mathsf{GC}_y, \{Z_{i,x_i}\})$ and outputs $v = F(x,y)$.

We next recall that a three-round semi-honest OT protocol can be constructed from enhanced trapdoor permutations (TDPs).

Definition 1 (Trapdoor permutations). *Let* TDP *be a triple of PPT algorithms* $(\mathsf{TDP.Gen}, \mathsf{TDP.Eval}, \mathsf{TDP.Invert})$ *such that if* $\mathsf{TDP.Gen}(1^\kappa)$ *outputs a pair* (f, td)*, then* $\mathsf{TDP.Eval}(f, \cdot)$ *is a permutation over* $\{0,1\}^\kappa$ *and* $\mathsf{TDP.Invert}(f, \mathsf{td}, \cdot)$ *is its inverse.* TDP *is a trapdoor permutation if for any PPT adversary* A*, there exists a negligible function* μ *such that* $\Pr[(f,\mathsf{td}) \leftarrow \mathsf{TDP.Gen}(1^\kappa); y \leftarrow \{0,1\}^\kappa; x \leftarrow A(f,y) : \mathsf{TDP.Eval}(f,x) = y] \leq \mu(\kappa)$.

For convenience, we drop (f, td) from the notation and write $f(\cdot), f^{-1}(\cdot)$ to denote algorithms $\mathsf{TDP.Eval}(f,\cdot), \mathsf{TDP.Invert}(f,\mathsf{td},\cdot)$ respectively. We assume that TDP satisfies a weak variant of certifiability, namely, given f it is possible to decide in polynomial time whether $\mathsf{TDP.Eval}(f,\cdot)$ is a permutation over $\{0,1\}^\kappa$. Let H be the function that is obtained from a single-bit hardcore function h of $f \in \mathsf{TDP}$ as follows: $\mathsf{H}(z) = h(z)\|h(f(z))\|\ldots\|h(f^{\kappa-1}(z))$. Informally, $\mathsf{H}(z)$ looks pseudorandom given $f^\kappa(z)$.

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

Putting it altogether, we obtain the following three-round semi-honest secure two-party computation protocol for the single-output functionality F:

Protocol Π_{SH}: P_1 holds input $x \in \{0,1\}^\kappa$ and P_2 holds inputs $y \in \{0,1\}^\kappa$. Let TDP be a family of trapdoor permutations and H be its hardcore bit function for κ bits. In the following, i always ranges from 1 to κ and b from 0 to 1.

Round-1 : P_2 computes $(\mathsf{GC}_y, \{Z_{i,b}\}) \leftarrow \mathsf{GenGC}(1^\kappa, F, y)$, chooses $\{(f_{i,b}, f_{i,b}^{-1})\}$ using $\mathsf{TDP.Gen}(1^\kappa)$, and sends $(\mathsf{GC}_y, \{f_{i,b}\})$ to P_2.
Round-2 : P_1 chooses random strings $\{z'_{i,b}\}$, computes $z_{i,x_i} = f^\kappa(z'_{i,x_i})$ and $z_{i,1-x_i} = z'_{i,1-x_i}$, and sends $\{z_{i,b}\}$ to P_2.
Round-3 : P_2 computes $W_{i,b} = Z_{i,b} \oplus \mathsf{H}(f_{i,b}^{-\kappa}(z_{i,b}))$ and sends $\{W_{i,b}\}$ to P_2.
Output : P_1 recovers the labels $Z_{i,x_i} = W_{i,x_i} \oplus \mathsf{H}(z'_{i,x_i})$ and computes $v = \mathsf{EvalGC}(\mathsf{GC}_y, \{Z_{i,x_i}\})$.

Equivocal commitment scheme Eqcom. We next recall the equivocal commitment scheme of [18] that is based on any (standard) non-interactive perfectly binding commitment scheme Com. To commit to a bit $x \in \{0,1\}$, the sender chooses coins ζ_1, ζ_2 and computes $\mathsf{Eqcom}(x; \zeta_1, \zeta_2) \overset{\text{def}}{=} \mathsf{Com}(x;\zeta_1)\|\mathsf{Com}(x;\zeta_2)$. It sends $\mathsf{C}_x = \mathsf{Eqcom}(x;\zeta_1,\zeta_2)$ to the receiver along with a zero-knowledge proof that C_x was constructed correctly (i.e., that there exist x, ζ_1, ζ_2 such that $\mathsf{C}_x = \mathsf{Eqcom}(x;\zeta_1,\zeta_2)$). To decommit, the sender chooses a bit b at random and reveals x, ζ_b, denoted by $\mathsf{open}_{\mathsf{C}_x}$. Note that a simulator can "equivocate" the commitment by setting $C = \mathsf{Com}(x;\zeta_1)\|\mathsf{Com}(1-x;\zeta_2)$ for a random bit

$x \in \{0,1\}$, simulating the zero-knowledge proof, and then revealing ζ_1 or ζ_2 depending on x and the bit to be revealed. This extends to strings by committing bitwise.

Sketch of the Katz-Ostrovsky Two-Party Protocol. The main components of the secure two-party computation protocol of Katz and Ostrovsky [18] are the three-round semi-honest secure two-party computation protocol Π_{SH} and proofs about the correctness of each round. Specifically, the protocol of [18] proceeds as follows. First, both parties commit to their inputs. Then, they run (modified) coin-tossing protocols to guarantee that each party obtains random coins that are committed to the other party. Finally, they run the Π_{SH} protocol together with proofs about the correctness of each round.

Since even a zero-knowledge argument alone requires four rounds, in the protocol of [18] the proof-of-correctness part is executed in parallel with Π_{SH}. To enable such a parallel execution, Katz and Ostrovsky use a zero-knowledge argument system with a "delayed input" property, i.e., a property that the statement to be proven need not be known until the last round. (Specifically, they use a variant of the four-round zero-knowledge proof system by Feige and Samir [9].) Furthermore, for technical reasons, in the protocol of [18] the above equivocal commitment scheme is used to commit to the garbled circuit.

2.2 Component of Our Protocol

Statistical Feige-Shamir zero-knowledge argument Π_{FS}. We use a four-round "delayed-input" statistical zero-knowledge argument Π_{FS} that is based on the four-round zero-knowledge argument system by Feige and Shamir [9]. Recall that the Feige-Shamir zero-knowledge argument for a statement thm consists of the following two (somewhat parallelized) executions of a witness-indistinguishable proof-of-knowledge system: in the first execution (in Rounds 1–3), the verifier proves the knowledge of "simulation trapdoor" σ—namely, selects a one-way function f, sets $x_1 = f(w_1)$ and $x_2 = f(w_2)$, and proves the knowledge of a witness for $\exists w$ s.t. $x_1 = f(w) \vee x_2 = f(w)$; in the second execution (in Rounds 2–4), the prover proves the knowledge of a witness for thm or the simulation trapdoor—i.e., proves the knowledge of a witness for $\mathsf{thm} \vee (\exists w \text{ s.t. } x_1 = f(w) \vee x_2 = f(w))$. We then obtain Π_{FS} by using Blum's three-round witness-indistinguishable proof of knowledge (denoted by Π_{WIPOK}) in the first execution and a four-round statistical witness-indistinguishable version of the "delayed input" witness-indistinguishable argument of Lapidot and Shamir [21] (denoted by Π_{SWIAOK}) in the second execution. It is not hard to see that Π_{FS} has a property that the statement to be proven is not needed until its last round, and it is complete, sound, and zero-knowledge even when the statement is determined in the last round.

Extractable commitment scheme ExtCom'. We use the following commitment scheme ExtCom', which is used in [10]. Let Com be any non-interactive perfectly binding commitment.

Commit Phase: The common input is security parameter 1^κ. The input to the committer is a string $m \in \{0,1\}^{\mathrm{poly}(\kappa)}$.

1. The committer chooses κ independent random pairs $\{\alpha_i^0, \alpha_i^1\}_{i\in[\kappa]}$ such that $\alpha_i^0 \oplus \alpha_i^1 = m$ for every $i \in [\kappa]$. The committer then commits to α_i^b for every $i \in [m]$, $b \in \{0,1\}$ using Com. Let c_i^b be the commitment to α_i^b.
2. The receiver sends uniformly random bits $\{e_i\}_{i\in[\kappa]}$.
3. The committer sends $\alpha_i^{e_i}$ for every $i \in [\kappa]$.
   COMMENT: *The committer just sends $\alpha_i^{e_i}$ and does not decommit $c_i^{e_i}$.*

Open Phase: The committer decommits c_i^b to α_i^b for every $i \in [\kappa], b \in \{0,1\}$.

ExtCom' has extractability in the sense that we can extract the committed value if we can obtain two *correctly constructed* transcripts by rewinding the committer in the last two rounds. We remark that if the commitment is *invalid*, i.e., there is no value to which the commitment can be correctly decommitted, this extracting procedure can output any value. We also remark that in ExtCom', a committer can easily give an invalid commitment by committing to all-zero strings in the first round and sending random strings in the last round. We use such an "fake" execution of ExtCom' in our protocol.

Non-malleable commitment scheme NMCom**.** Let $\langle C, R\rangle$ be a tag-based commitment scheme (i.e., a commitment scheme that takes a κ-bit string—a *tag*—as an additional input). Informally, $\langle C, R\rangle$ is *non-malleable* if for any man-in-the-middle adversary $\mathcal{M}$, who gives a commitment of $\langle C, R\rangle$ in the "right" interaction while receiving a commitment of $\langle C, R\rangle$ in the "left" interaction, the value committed in the right interaction is "independent" of the value committed in the left interaction as long as the tags in the two interactions are different. See, e.g., [22] for a formal definition.

In our main result, we use a non-malleable commitment scheme such that:

1. The scheme is public coin (i.e., the receiver is public coin) and the round complexity is 3.
2. The scheme has the following extractability: an extractor extracts the committed value from a valid commitment and extracts $\perp$ from an invalid one.

Such a non-malleable commitment exist under quasi-polynomially-hard injective OWFs [16, 15]; see Footnote 1. For simplicity, we also assume that the extractor E rewinds the committer in the last two rounds until it obtains two accepting transcripts. That is, we assume that E interacts with the committer in the same way as the honest receiver on the *main thread* while repeatedly interacting with it from the second round with fresh randomness on the *look-ahead threads*, and when the commitment is accepting on the main thread, E extracts the committed values using the accepting commitment on a look-ahead thread.

Lossy encryption scheme. Informally, a *lossy encryption scheme* [3, 17] is a public-key encryption scheme such that, in addition to the standard key generation algorithm, it has a lossy key generating algorithm with the following property: A lossy public key is indistinguishable from a standard public key, and a ciphertext generated under a lossy public key statistically hides the information of the plaintext. More precisely, a lossy public-key encryption scheme is a tuple $(\mathsf{LE.Gen}, \mathsf{LE.Enc}, \mathsf{LE.Dec})$ of PPT algorithms such that:

– $\mathsf{LE.Gen}(1^\kappa, \mathsf{inj})$ outputs *injective keys* $(\mathsf{pk}, \mathsf{sk})$.
– $\mathsf{LE.Gen}(1^\kappa, \mathsf{lossy})$ outputs *lossy keys* $(\mathsf{pk}_{\mathsf{lossy}}, \mathsf{sk}_{\mathsf{lossy}})$.

For a formal security definition, see [3, 17].

It is shown in [3] that a lossy encryption scheme can be constructed from *lossy trapdoor functions* [34], which in turn can be realized based on a variety of assumptions including the DDH assumption and the LWE assumption.

ZAP Π_{ZAP}. ZAPs are two-message public-coin witness-indistinguishable proof systems, and can be based on doubly enhanced trapdoor permutations [7].

3 UC Security and Its SPS Variant

We recall the definition of UC security [4] and its SPS variant [35, 2, 10]. A part of the text below is taken from [10].

3.1 UC Security

We assume familiarity with the UC framework. For full details, see [4].

Recall that in the UC framework, the model for protocol execution consists of the environment $\mathcal{Z}$, the adversary $\mathcal{A}$, and the parties running protocol π. In this paper, we consider static adversaries and assume the existence of authenticated communication channels. Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$ denote a random variable for the output of $\mathcal{Z}$ on security parameter $\kappa \in \mathbb{N}$ and input $z \in \{0,1\}^*$ with a uniformly-chosen random tape. Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z)\}_{\kappa\in\mathbb{N}, z\in\{0,1\}^*}$.

The security of a protocol π is defined using the *ideal protocol*. In the execution of the ideal protocol, all the parties simply hand their inputs to the *ideal functionality* $\mathcal{F}$. The ideal functionality $\mathcal{F}$ carries out the desired task securely and gives outputs to the parties, and the parties forward these outputs to $\mathcal{Z}$. The adversary $\mathcal{S}$ in the execution of the ideal protocol is often called the *simulator*. Let $\pi(\mathcal{F})$ denote the ideal protocol for functionality $\mathcal{F}$.

We say that a protocol π *emulates* protocol ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties running π or it is interacting with $\mathcal{S}$ and parties running ϕ. We say that π *securely realizes* an ideal functionality $\mathcal{F}$ if it emulates the ideal protocol $\Pi(\mathcal{F})$.

3.2 UC Security with Super-polynomial Simulation

We next provide a relaxed notion of UC security where the simulator is given access to super-polynomial computational resources.

Definition 2. *Let π and ϕ be protocols. We say that π* UC-SPS-emulates *ϕ if for any adversary $\mathcal{A}$ there exists a super-polynomial-time adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ that obeys the rules of interaction for UC security, we have* $\mathrm{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$.

Definition 3. *Let $\mathcal{F}$ be an ideal functionality and let π be a protocol. We say that π* UC-SPS-realizes *$\mathcal{F}$ if π UC-SPS-emulates the ideal process $\Pi(\mathcal{F})$.*

The multi-session extension of an ideal functionality. When showing concurrent security of a protocol π under SPS security, we need to construct a simulator in a setting where parties execute π concurrently. (In other words, unlike in UC security, we cannot rely on the *composition theorem* in SPS security.)

To consider the simulator in such a setting, we use a *multi-session extension* of an ideal functionality. Roughly speaking, the multi-session extension $\hat{\mathcal{F}}$ of an ideal functionality $\mathcal{F}$ is a functionally that internally runs multiple copies of $\mathcal{F}$.

4 Our Five-round Secure Two-party Protocol

In this section, we prove our main result.

Theorem 1. *Assume the existence of collision-resilient hash function families, trapdoor permutation families*[4]*, lossy encryption schemes, and quasi-polynomially-hard injective one-way functions. Let $\mathcal{F}$ be any well-formed two-party functionality and $\hat{\mathcal{F}}$ be its multi-session extension. Then, there exists a five-round protocol that UC-SPS realizes $\hat{\mathcal{F}}$.*

The other result, a seven-round protocol under polynomially-hard assumptions, is given in the full version of this paper.

Recall that in the UC framework, there are any number of parties $P_1, P_2, \ldots$, and any two of them (say, P_i and P_j) can compute $\mathcal{F}$ using $\hat{\mathcal{F}}$ in each subsession. To simplify the description of the protocol and the proofs, in what follows we denote the first party of $\mathcal{F}$ by P_1 and the second party of $\mathcal{F}$ by P_2 in every subsession. (Equivalently, we consider a setting where two parties P_1, P_2 compute $\mathcal{F}$ any number of times using $\hat{\mathcal{F}}$, and $\mathcal{A}$ corrupts either P_1 or P_2 in each subsession.)

4.1 Our Protocol Π_{2PC}

Our protocol is based on the two-party computation protocol of Katz and Ostrovsky [18]; their protocol is described in Section 2.1. In our protocol, we use the primitives that are described in Sections 2.1 and 2.2, and additionally, we use a symmetric-key encryption scheme $\mathsf{SKE} = (\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ and a message authentication code MAC.

Our protocol Π_{2PC}. We denote the two parties by P_1 and P_2. P_1 holds input $x \in \{0,1\}^\kappa$ and P_2 holds input $y \in \{0,1\}^\kappa$. The identities of P_1 and P_2 (i.e., their PIDs) are id_1 and id_2 respectively, where $\mathsf{id}_1 \neq \mathsf{id}_2$. Let $\mathcal{F} = (F_1, F_2) : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa \times \{0,1\}^\kappa$ be the functionality to be computed. Let $\mathcal{F}' = (F'_1, F'_2)$ be a functionality such that:

[4] Recall that we assume that the trapdoor permutation families satisfy (a weak form of) "certifiability" and their domain/range is $\{0,1\}^\kappa$.

– $F'_1(x, y') = (F_1(x, y), \mathsf{enc}, \mathsf{mac})$, where $y' = (y, \mathsf{sk}_{\rm ske}, \mathsf{sk}_{\rm mac}, \omega_{\rm enc}) \in \{0,1\}^{4\kappa}$, $\mathsf{enc} = \mathsf{SKE.Enc}_{\mathsf{sk}_{\rm ske}}(F_2(x, y); \omega_{\rm enc})$, and $\mathsf{mac} = \mathsf{MAC}_{\mathsf{sk}_{\rm mac}}(\mathsf{enc})$.
– $F'_2(x, y') = \perp$ for any x and y'.

In the following i always ranges from 1 to κ and b from 0 to 1. We will skip mentioning the SID and SSID to keep the protocol specification simple.

Round 1. P_1 sends a message m_1 that is defined as follows.

1. P_1 commits to 2κ random strings $\{r_{i,b}\}$ using 2κ parallel and independent executions of Com. I.e., it chooses uniformly random strings $r_{i,b}$ and randomness $\omega^{i,b}_{\rm com}$ and then generates $\mathsf{com}_{i,b} = \mathsf{Com}(r_{i,b}; \omega^{i,b}_{\rm com})$.
2. P_1 starts committing to κ strings $\{r_{i,1-x_i} \| \omega^{i,1-x_i}_{\rm com}\}$ using κ parallel and independent executions of ExtCom' and also starts κ "fake" executions of ExtCom'. Concretely, P_1 prepares $\{\mathsf{ext}^{i,b}_1\}$ as follows.
   – For every $i \in [\kappa]$, P_1 prepares $\mathsf{ext}^{i,1-x_i}_1$ by committing to $r_{i,1-x_i} \| \omega^{i,1-x_i}_{\rm com}$ using ExtCom'. I.e., it generates $\mathsf{ext}^{i,1-x_i}_1 \leftarrow \mathsf{ExtCom}'_1(r_{i,1-x_i} \| \omega^{i,1-x_i}_{\rm com})$, which is the first message of $\mathsf{ExtCom}'(r_{i,1-x_i} \| \omega^{i,1-x_i}_{\rm com})$.
   – For every $i \in [\kappa]$, P_1 prepares ext^{i,x_i}_1 by committing to all-zero strings using Com. (Recall that the first round of ExtCom' consists of 2κ executions of Com.)
3. P_1 prepares the first message fs_1 of $\Pi_{\rm FS}$.
4. Message m_1 is the tuple $(\{\mathsf{com}_{i,b}, \mathsf{ext}^{i,b}_1\}, \mathsf{fs}_1)$.

Round 2. P_2 sends a message m_2 that is defined as follows.

1. P_2 samples secret-keys $\mathsf{sk}_{\rm ske}$ and $\mathsf{sk}_{\rm mac}$ for SKE and MAC respectively and chooses randomness $\omega_{\rm enc}$ for $\mathsf{SKE.Enc}$.
2. P_2 prepares a garbled circuit and labels for F'_1 with input $y' = (y, \mathsf{sk}_{\rm ske}, \mathsf{sk}_{\rm mac}, \omega_{\rm enc})$. I.e., it uniformly chooses randomness Ω and generates $\big(\mathsf{GC}, \{Z_{i,b}\}\big) = \mathsf{GenGC}\big(1^\kappa, F'_1, y'; \Omega\big)$.
3. P_2 generates standard commitments to the labels and an equivocal commitment to the garbled circuit. I.e., it uniformly chooses randomness $\{\omega^{i,b}_{\rm lab}\}$ and $\omega_{\rm gc}$ and generates $\mathsf{C}^{i,b}_{\rm lab} = \mathsf{Com}(Z_{i,b}; \omega^{i,b}_{\rm lab})$ and $\mathsf{C}_{\rm gc} = \mathsf{Eqcom}(\mathsf{GC}; \omega_{\rm gc})$. Let $\mathsf{open}_{\mathsf{C}_{\rm gc}}$ be the decommitment information that decommits $\mathsf{C}_{\rm gc}$ to GC.
4. P_2 samples random strings $\{r'_{i,b}\}$ and $\big(f_{i,b}, f^{-1}_{i,b}\big) \leftarrow \mathsf{TDP.Gen}(1^\kappa)$ for the coin tossing and the oblivious transfer executions.
5. P_2 generates the second messages $\{\mathsf{ext}^{i,b}_2\}$ for all the executions of ExtCom' initiated by P_1.
6. P_2 prepares the first message zap_1 of $\Pi_{\rm ZAP}$.
7. P_2 prepares the second message fs_2 of $\Pi_{\rm FS}$ initiated by P_1.
8. P_2 chooses randomness $\omega_{\rm leEnc}$ for $\mathsf{LE.Enc}$.
9. Let $\mathsf{wit}_2 := (y', \Omega, \mathsf{GC}, \omega_{\rm gc}, \mathsf{open}_{\mathsf{C}_{\rm gc}}, \{Z_{i,b}, \omega^{i,b}_{\rm lab}\})$ and $\mathsf{wit}_4 := (\mathsf{wit}_2, \{f^{-1}_{i,b}\}, \omega_{\rm leEnc})$. Then, P_2 starts committing to wit_4 using NMCom with identity id_2. I.e., it generates $\mathsf{nm}_1 \leftarrow \mathsf{NMCom}_1(\mathsf{id}_2, \mathsf{wit}_4)$, which is the first message of $\mathsf{NMCom}(\mathsf{id}_2, \mathsf{wit}_4)$.
   We remark that wit_2 is a witness for the following statement $\mathsf{st}_2 = (F'_1, \mathsf{C}_{\rm gc}, \{\mathsf{C}^{i,b}_{\rm lab}\})$.

$\exists$ $\mathsf{wit}_2 = \big(y', \varOmega, \mathsf{GC}, \omega_{\mathrm{gc}}, \mathsf{open}_{\mathsf{C}_{\mathrm{gc}}}, \{Z_{i,b}, \omega_{\mathrm{lab}}^{i,b}\}\big)$ s.t.
(a) $\big(\mathsf{GC}, \{Z_{i,b}\}\big) = \mathsf{GenGC}\big(1^\kappa, F_1', y'; \varOmega\big)$, and
(b) $\mathsf{C}_{\mathrm{gc}} = \mathsf{Eqcom}(\mathsf{GC}; \omega_{\mathrm{gc}})$ and $\forall(i,b) : \mathsf{C}_{\mathrm{lab}}^{i,b} = \mathsf{Com}(Z_{i,b}; \omega_{\mathrm{lab}}^{i,b})$, and
(c) $\mathsf{open}_{\mathsf{C}_{\mathrm{gc}}}$ is a valid decommitment that opens C_{gc} to GC.

COMMENT: *Informally,* st_2 *is the statement that* P_2 *performed this step correctly, i.e., generated a garbled circuit and labels correctly and then committed to them in* $\mathsf{C}_{\mathrm{gc}}, \{\mathsf{C}_{\mathrm{lab}}^{i,b}\}$.

10. Message m_2 is the tuple $(\{\mathsf{C}_{\mathrm{lab}}^{i,b}, r'_{i,b}, f_{i,b}, \mathsf{ext}_2^{i,b}\}, \mathsf{C}_{\mathrm{gc}}, \mathsf{zap}_1, \mathsf{fs}_2, \mathsf{nm}_1)$.

Round 3. If any of $\{f_{i,b}\}$ is invalid, P_1 aborts. Otherwise, P_1 sends a message m_3 that is defined as follows.

1. P_1 invokes κ parallel executions of oblivious transfer to obtain the input-wire labels corresponding to its input x. More specifically, P_1 does the following for every $i \in [\kappa]$.
   – If $x_i = 0$, sample $z'_{i,0} \leftarrow \{0,1\}^{\kappa/2}$ and then set $z_{i,0} := f_{i,0}^\kappa(\mathsf{PRG}(z'_{i,0}))$ and $z_{i,1} := r_{i,1} \oplus r'_{i,1}$.
   – If $x_i = 1$, sample $z'_{i,1} \leftarrow \{0,1\}^{\kappa/2}$ and then set $z_{i,1} := f_{i,1}^\kappa(\mathsf{PRG}(z'_{i,1}))$ and $z_{i,0} := r_{i,0} \oplus r'_{i,0}$.
2. P_1 prepares $\{\mathsf{ext}_3^{i,b}\}$, where $\{\mathsf{ext}_3^{i,1-x_i}\}$ are the third messages of ExtCom' and $\{\mathsf{ext}_3^{i,x_i}\}$ are random strings.
3. P_1 prepares injective keys $(\mathsf{pk}_{\mathrm{le}}, \mathsf{sk}_{\mathrm{le}})$ of the lossy encryption scheme, i.e., it generates $(\mathsf{pk}_{\mathrm{le}}, \mathsf{sk}_{\mathrm{le}}) \leftarrow \mathsf{LE.Gen}(1^\kappa, \mathsf{inj})$.
4. P_1 prepares the second message zap_2 of $\varPi_{\mathrm{ZAP}}$ proving the following statement $\mathsf{st}_3 = (\{\mathsf{com}_{i,b}, \mathsf{ext}_1^{i,b}, \mathsf{ext}_2^{i,b}, \mathsf{ext}_3^{i,b}, r'_{i,b}, z_{i,b}, f_{i,b}\}, \mathsf{pk}_{\mathrm{le}})$:
   $\exists$ $\mathsf{wit}_3 = (\{b_i, r_i, \omega_{\mathrm{com}}^i, \omega_{\mathrm{ext}}^i, z'_i\}_{i\in[\kappa]}, \mathsf{sk}_{\mathrm{le}}, \omega_{\mathrm{leGen}})$ s.t. $\forall i$:
   (a) $\mathsf{com}_{i,b_i} = \mathsf{Com}(r_i; \omega_{\mathrm{com}}^i)$, and
   (b) ext_1^{i,b_i} and ext_3^{i,b_i} are the first and the third messages of $\mathsf{ExtCom}'(r_i \| \omega_{\mathrm{com}}^i; \omega_{\mathrm{ext}}^i)$ with the second message being ext_2^{i,b_i}, and
   (c) $z_{i,b_i} = r_{i,b_i} \oplus r'_{i,b_i}$, and
   (d) $z_{i,1-b_i} = f_{i,1-b_i}^\kappa(\mathsf{PRG}(z'_i)) \bigvee (\mathsf{pk}_{\mathrm{le}}, \mathsf{sk}_{\mathrm{le}}) = \mathsf{LE.Gen}(1^\kappa, \mathsf{lossy}; \omega_{\mathrm{leGen}})$.

   P_1 uses $(\{1 - x_i, r_{i,1-x_i}, \omega_{\mathrm{com}}^{i,1-x_i}, \omega_{\mathrm{ext}}^{i,1-x_i}, z'_{i,x_i}\}_{i\in[\kappa]}, \bot, \bot)$ as the witness.
   COMMENT: *Informally,* st_3 *is the statement that either* P_1 *performed this step correctly (i.e., one of* $z_{i,0}, z_{i,1}$ *is an image of* $f_{i,1-b_i}^\kappa(\mathsf{PRG}(\cdot))$ *and the other is the outcome of the coin-tossing) or* $\mathsf{pk}_{\mathrm{le}}$ *is a lossy key. Here,* PRG *is used to make sure that* $z_{i,1-b_i} \neq r_{i,1-b_i} \oplus r'_{i,1-b_i}$ *holds when* $\mathsf{pk}_{\mathrm{le}}$ *is an injective key.*
5. P_1 prepares the third message fs_3 of $\varPi_{\mathrm{FS}}$.
6. P_1 prepares the second message nm_2 of NMCom.
7. Message m_3 is the tuple $(\{z_{i,b}, \mathsf{ext}_3^{i,b}\}, \mathsf{pk}_{\mathrm{le}}, \mathsf{zap}_2, \mathsf{fs}_3, \mathsf{nm}_2)$.

Round 4. If zap_2 or fs_3 is not accepting, P_2 aborts. Otherwise, P_2 sends a message m_4 that is defined as follows.

1. P_2 completes the execution of the oblivious transfers by computing $W_{i,b} = Z_{i,b} \oplus \mathsf{H}(f_{i,b}^{-\kappa}(z_{i,b}))$.
2. P_2 encrypts $\{W_{i,b}\}\|\mathsf{GC}\|\mathsf{open}_{\mathsf{C}_{\mathrm{gc}}}$ using the lossy encryption scheme with public key $\mathsf{pk}_{\mathrm{le}}$ and randomness ω_{leEnc} (which was chosen in Round 2), i.e., it computes $\mathsf{CT}_{\mathrm{gc}} = \mathsf{LE.Enc}_{\mathsf{pk}_{\mathrm{le}}}(\{W_{i,b}\}\|\mathsf{GC}\|\mathsf{open}_{\mathsf{C}_{\mathrm{gc}}}; \omega_{\mathrm{leEnc}})$.
3. P_2 prepares the final message nm_3 of NMCom.
4. Let $\mathsf{st}_4 = (\{f_{i,b}, z_{i,b}\}, \mathsf{st}_2, \mathsf{CT}_{\mathrm{gc}})$ be the following statement:
   $\exists \mathsf{wit}_4 = (\mathsf{wit}_2, \{g_{i,b}\}, \omega_{\mathrm{leEnc}})$ s.t.
   (a) $\mathsf{wit}_2 = (y', \Omega, \mathsf{GC}, \omega_{\mathrm{gc}}, \mathsf{open}_{\mathsf{C}_{\mathrm{gc}}}, \{Z_{i,b}, \omega_{\mathrm{lab}}^{i,b}\})$ is a valid witness for st_2, and
   (b) $\forall (i,b)$: $f_{i,b}^{\kappa}(g_{i,b}^{\kappa}(z_{i,b})) = z_{i,b}$, and
   (c) $\mathsf{CT}_{\mathrm{gc}} = \mathsf{LE.Enc}_{\mathsf{pk}_{\mathrm{le}}}(\{W_{i,b}\}\|\mathsf{GC}\|\mathsf{open}_{\mathsf{C}_{\mathrm{gc}}}; \omega_{\mathrm{leEnc}})$, where $W_{i,b} = Z_{i,b} \oplus \mathsf{H}(g_{i,b}^{\kappa}(z_{i,b}))$.

   Then, P_2 prepares the final message fs_4 of Π_{FS} proving the following statement $(\mathsf{nm}_1, \mathsf{nm}_2, \mathsf{nm}_3, \mathsf{st}_4)$.
   $\exists\ \omega_{\mathrm{nm}}$ and wit_4 s.t.
   (a) nm_1 and nm_3 are the first and the third message of $\mathsf{NMCom}(\mathsf{id}_2, \mathsf{wit}_4; \omega_{\mathrm{nm}})$ with the second message being nm_2, and
   (b) wit_4 is a valid witness for st_4.

   I.e., P_2 proves that it committed to a witness for st_4 using NMCom.
   COMMENT: *Informally,* st_4 *is the statement that* P_1 *performed this step and the previous step correctly (in particular, the final messages of the oblivious transfers and the opening of* C_{gc} *were encrypted in* $\mathsf{CT}_{\mathrm{gc}}$*).*
5. Message m_4 is the tuple $(\mathsf{CT}_{\mathrm{gc}}, \mathsf{fs}_4, \mathsf{nm}_3)$.

Round 5. If fs_4 or nm_3 is not accepting, P_1 aborts. Otherwise, P_1 sends a message m_5 that is defined as follows.

1. P_1 recovers $\{W_{i,b}\}\|\mathsf{GC}\|\mathsf{open}_{\mathsf{C}_{\mathrm{gc}}}$ by decrypting $\mathsf{CT}_{\mathrm{gc}}$, i.e., it computes $\{W_{i,b}\}\|\mathsf{GC}\|\mathsf{open}_{\mathsf{C}_{\mathrm{gc}}} = \mathsf{LE.Dec}_{\mathsf{sk}_{\mathrm{le}}}(\mathsf{CT}_{\mathrm{gc}})$. If $(\mathsf{GC}, \mathsf{open}_{\mathsf{C}_{\mathrm{gc}}})$ is not a valid opening of C_{gc}, P_1 aborts. Otherwise, P_1 recovers the garbled labels $\{Z_i := Z_{i,x_i}\}$ from the completion of the oblivious transfer, and then it computes $(F_1(x,y), \mathsf{enc}, \mathsf{mac}) = \mathsf{EvalGC}(\mathsf{GC}, \{Z_i\})$.
2. Message m_5 is the tuple $(\mathsf{enc}, \mathsf{mac})$.

Output computation.

P_1's output: P_1 outputs $F_1(x,y)$, which it obtained in Round 5.
P_2's output: If $\mathsf{MAC}_{\mathsf{sk}_{\mathrm{mac}}}(\mathsf{enc}) \neq \mathsf{mac}$, P_2 outputs $\perp$. Otherwise, it outputs $F_2(x,y) = \mathsf{SKE.Dec}_{\mathsf{sk}_{\mathrm{ske}}}(\mathsf{enc})$.

4.2 Description of Simulator $\mathcal{S}$

The simulator $\mathcal{S}$ internally invokes $\mathcal{A}$ and simulates the real-world execution for $\mathcal{A}$ as follows. To simulate the interaction between $\mathcal{A}$ and $\mathcal{Z}$, $\mathcal{S}$ simply forwards messages between $\mathcal{A}$ and $\mathcal{Z}$. To simulate the interaction between P_1 and P_2, $\mathcal{S}$ does the following in each subsession.

Case 1: P_1 is corrupted. $\mathcal{S}$ simulates P_2's messages as follows.

- In Round 1, $\mathcal{S}$ receives $m_1 = (\{\mathsf{com}_{i,b}, \mathsf{ext}_1^{i,b}\}, \mathsf{fs}_1)$ from $\mathcal{A}$.
- In Round 2, $\mathcal{S}$ prepares $m_2 = (\{\mathsf{C}_{\mathrm{lab}}^{i,b}, r'_{i,b}, f_{i,b}, \mathsf{ext}_2^{i,b}\}, \mathsf{C}_{\mathrm{gc}}, \mathsf{zap}_1, \mathsf{fs}_2, \mathsf{nm}_1)$ in the same way as P_2 does except for the following.
  - $\mathcal{S}$ generates $\{\mathsf{C}_{\mathrm{lab}}^{i,b}\}$ by committing to all-zero strings.
  - $\mathcal{S}$ generates C_{gc} in a way that it can be decommitted to any value by using equivocality.
  - $\mathcal{S}$ generates nm_1 by committing to a all-zero string using NMCom.

  Then, $\mathcal{S}$ sends m_2 to $\mathcal{A}$.
- In Round 3, $\mathcal{S}$ receives $m_3 = (\{z_{i,b}, \mathsf{ext}_3^{i,b}\}, \mathsf{pk}_{\mathrm{le}}, \mathsf{zap}_2, \mathsf{fs}_3, \mathsf{nm}_2)$ from $\mathcal{A}$. If m_3 is accepting, $\mathcal{S}$ does the following.
  1. Extracts the committed values of the ExtCom' commitments $\{(\mathsf{ext}_1^{i,b}, \mathsf{ext}_2^{i,b}, \mathsf{ext}_3^{i,b})\}$ by brute force. The extracted values are denoted by $\{\tilde{r}_{i,b} \| \tilde{\omega}_{\mathrm{com}}^{i,b}\}$. (If a commitment is invalid, its committed value is defined to be $\bot$.) If there is $i \in [\kappa]$ such that for any $b_i^* \in \{0,1\}$, either $(\tilde{r}_{i,b_i^*}, \tilde{\omega}_{\mathrm{com}}^{i,b_i^*})$ is not a valid decommitment of com_{i,b_i^*} or it holds that $z_{i,b_i^*} \neq \tilde{r}_{i,b_i^*} \oplus r'_{i,b_i^*}$, $\mathcal{S}$ aborts the simulation with output Abort_1.
  2. Define $x^* = (x_1^*, \ldots, x_\kappa^*)$ as follows: for each $i \in [\kappa]$, if $(\tilde{r}_{i,0}, \tilde{\omega}_{\mathrm{com}}^{i,0})$ is a valid decommitment of $\mathsf{com}_{i,0}$ and furthermore it holds that $z_{i,0} = \tilde{r}_{i,0} \oplus r'_{i,0}$, define $x_i^* := 1$, and otherwise, define $x_i^* := 0$.
  3. Send x^* to the ideal functionality $\mathcal{F}$ (through $\hat{\mathcal{F}}$) and obtain $v_1 = F_1(x^*, y)$.
  4. Extract the "simulation trapdoor" σ of Π_{FS} by brute force from its first three rounds $(\mathsf{fs}_1, \mathsf{fs}_2, \mathsf{fs}_3)$.
- In Round 4, $\mathcal{S}$ prepares $m_4 = (\mathsf{CT}_{\mathrm{gc}}, \mathsf{fs}_4, \mathsf{nm}_3)$ in the same way as P_2 does except for the following.
  - $\mathcal{S}$ generates $\mathsf{CT}_{\mathrm{gc}}$ as follows. First, $\mathcal{S}$ simulates a garbled circuit and labels by $(\mathsf{GC}^*, \{Z_i^*\}) \leftarrow \mathsf{SimGC}(1^\kappa, F'_1, v'_1)$, where $v'_1 = (v_1, \tilde{\mathsf{enc}}, \tilde{\mathsf{mac}})$, $\tilde{\mathsf{enc}} \leftarrow \mathsf{SKE.Enc}_{\mathsf{sk}_{\mathrm{ske}}}(0^\kappa)$, and $\tilde{\mathsf{mac}} = \mathsf{MAC}_{\mathsf{sk}_{\mathrm{mac}}}(\tilde{\mathsf{enc}})$ for randomly sampled $\mathsf{sk}_{\mathrm{ske}}$ and $\mathsf{sk}_{\mathrm{mac}}$. Second, using the equivocality of Eqcom, $\mathcal{S}$ obtains a decommitment $\mathsf{open}_{\mathsf{C}_{\mathrm{gc}}}^*$ that opens C_{gc} to GC^*. Third, $\mathcal{S}$ generates $\{W_{i,b}\}$ by $W_{i,x_i^*} := Z_i^* \oplus \mathsf{H}(f_{i,x_i^*}^{-\kappa}(z_{i,b}))$ and $W_{i,1-x_i^*} \leftarrow \{0,1\}^\kappa$. Finally, $\mathcal{S}$ generates $\mathsf{CT}_{\mathrm{gc}}$ by $\mathsf{CT}_{\mathrm{gc}} \leftarrow \mathsf{LE.Enc}_{\mathsf{pk}_{\mathrm{le}}}(\{W_{i,b}\} \| \mathsf{GC}^* \| \mathsf{open}_{\mathsf{C}_{\mathrm{gc}}}^*)$.
  - $\mathcal{S}$ generates fs_4 by completing Π_{FS} using the simulation trapdoor σ.

  Then, $\mathcal{S}$ sends m_4 to $\mathcal{A}$.
- In Round 5, $\mathcal{S}$ receives $m_5 = (\mathsf{enc}, \mathsf{mac})$. If m_5 is accepting, $\mathcal{S}$ tells the ideal functionality $\mathcal{F}$ to send the output to P_2.

Case 2: P_2 is corrupted. $\mathcal{S}$ simulates P_1's messages as follows.

- In Round 1, $\mathcal{S}$ generates $m_1 = (\{\mathsf{com}_{i,b}, \mathsf{ext}_1^{i,b}\}, \mathsf{fs}_1)$ in the same way as P_1 except that $\mathcal{S}$ generates $\mathsf{ext}_1^{i,b}$ by committing to $r_{i,b} \| \omega_{\mathrm{com}}^{i,b}$ using ExtCom' correctly for every $i \in [\kappa]$ and $b \in \{0,1\}$. Then, $\mathcal{S}$ sends m_1 to $\mathcal{A}$.
- In Round 2, $\mathcal{S}$ receives $m_2 = (\{\mathsf{C}_{\mathrm{lab}}^{i,b}, r'_{i,b}, f_{i,b}, \mathsf{ext}_2^{i,b}\}, \mathsf{C}_{\mathrm{gc}}, \mathsf{zap}_1, \mathsf{fs}_2, \mathsf{nm}_1)$ from $\mathcal{A}$.
- In Round 3, $\mathcal{S}$ generates $m_3 = (\{z_{i,b}, \mathsf{ext}_3^{i,b}\}, \mathsf{pk}_{\mathrm{le}}, \mathsf{zap}_2, \mathsf{fs}_3, \mathsf{nm}_2)$ in the same way as P_1 except for the following.

  - $\mathcal{S}$ generates $\{z_{i,b}\}$ by $z_{i,b} := r_{i,b} \oplus r'_{i,b}$ for every $i \in [\kappa]$ and $b \in \{0,1\}$.
  - $\mathcal{S}$ generates $\mathsf{ext}_3^{i,b}$ by executing ExtCom' correctly for every $i \in [\kappa]$ and $b \in \{0,1\}$.
  - $\mathcal{S}$ generates $\mathsf{pk}_{\mathrm{le}}$ by $(\mathsf{pk}_{\mathrm{le}}, \mathsf{sk}_{\mathrm{le}}) \leftarrow \mathsf{LE.Gen}(1^\kappa, \mathsf{lossy})$ with randomness ω_{leGen}. (I.e., $\mathcal{S}$ generates a lossy public key rather than an injective one.)
  - When generating zap_2, $\mathcal{S}$ uses $(\{1, r_{i,1}, \omega_{\mathrm{com}}^{i,1}, \omega_{\mathrm{ext}}^{i,1}, \bot\}_{i\in[\kappa]}, \mathsf{sk}_{\mathrm{le}}, \omega_{\mathrm{leGen}})$ as the witness. (I.e., $\mathcal{S}$ proves that $\{(\mathsf{com}_{i,1}, \mathsf{ext}_1^{i,1}, \mathsf{ext}_3^{i,1}, z_{i,1})\}$ are computed correctly and $\mathsf{pk}_{\mathrm{le}}$ is a lossy public key.)

  Then, $\mathcal{S}$ sends m_3 to $\mathcal{A}$.
- In Round 4, $\mathcal{S}$ receives $m_4 = (\mathsf{CT}_{\mathrm{gc}}, \mathsf{fs}_4, \mathsf{nm}_3)$. If m_4 is accepting, $\mathcal{S}$ does the following.
  1. Extract the committed value of the NMCom commitment $(\mathsf{nm}_1, \mathsf{nm}_2, \mathsf{nm}_3)$ by brute force. If the extracted value is not a valid witness for st_4, $\mathcal{S}$ aborts the simulation with output Abort_2. Otherwise, the extracted value is denoted by $\mathsf{wit}_4 = \mathsf{wit}_2 \| \{g_{i,b}\} \| \omega_{\mathrm{leEnc}}$, where $\mathsf{wit}_2 = (y', \mathit{\Omega}, \mathsf{GC}, \omega_{\mathrm{gc}}, \mathsf{open}_{\mathsf{C}_{\mathrm{gc}}}, \{Z_{i,b}, \omega_{\mathrm{lab}}^{i,b}\})$.
  2. Parse y' as $(y, \mathsf{sk}_{\mathrm{ske}}, \mathsf{sk}_{\mathrm{mac}}, \omega_{\mathrm{enc}})$, send y to the ideal functionality $\mathcal{F}$, and receive $v_2 = F_2(x, y)$.
- In Round 5, $\mathcal{S}$ generates $m_5 = (\mathsf{enc}, \mathsf{mac})$ by $\mathsf{enc} := \mathsf{SKE.Enc}_{\mathsf{sk}_{\mathrm{ske}}}(v_2; \omega_{\mathrm{enc}})$ and $\mathsf{mac} = \mathsf{MAC}_{\mathsf{sk}_{\mathrm{mac}}}(\mathsf{enc})$.

4.3 Proof of Indistinguishability

Fix any PPT adversary $\mathcal{A}$, and assume for contradiction that there exists a PPT environment $\mathcal{Z}$ and a PPT distinguisher D such that for infinitely many κ:

$$\varepsilon(\kappa) \overset{\mathrm{def}}{=} \left|\Pr\left[D(\mathrm{EXEC}_{\mathit{\Pi}_{\mathrm{2PC}},\mathcal{A},\mathcal{Z}}(\kappa)) = 1\right] - \Pr\left[D(\mathrm{EXEC}_{\mathit{\Pi}(\mathcal{F}),\mathcal{S},\mathcal{Z}}(\kappa)) = 1\right]\right| \geq \frac{1}{\mathrm{poly}(\kappa)} \quad . \tag{1}$$

We derive a contradiction by a hybrid argument. Let m be an upper bound on the number of subsessions (e.g., an upper bound on the running time of $\mathcal{Z}$). Let $N := (10m\kappa/\varepsilon)^2$, which is a parameter that we use in the hybrid experiments. (Roughly speaking, we use N to determine the number of rewinding during extraction procedures in the hybrid experiments. We define N so that the extraction fails with probability that is much smaller than ε.)

Before defining the hybrid experiments, we define the order of the sessions. The order of the sessions is determined by the order of $\mathsf{special\ messages}$, where the message in Round 2 is $\mathsf{special\ message}$ when P_1 is corrupted, and the message in Round 3 is $\mathsf{special\ message}$ when P_2 is corrupted.

Then, we define the hybrid experiments, $H_{0:17}$, $H_{k:j}$ ($k \in [m], j \in [17]$), and $H_{m+1:1}$, as follows. Hybrid $H_{0:17}$ is identical with the real experiment. Hence, in $H_{0:17}$, several parties (environment $\mathcal{Z}$, adversary $\mathcal{A}$, and two parties P_1, P_2) are invoked and then protocol $\mathit{\Pi}_{\mathrm{2PC}}$ is executed concurrently multiple times among them; we call these executions of $\mathit{\Pi}_{\mathrm{2PC}}$ the *main thread*. Next, for every $k \in [m]$, hybrids $H_{k:1}, \ldots, H_{k:17}$ are defined as follows.

Hybrid $H_{k:1}$ is the same as $H_{k-1:17}$ except that in session k on the main thread, if P_1 is corrupted, then the simulation trapdoor σ and the implicit input x^* are extracted as follows.

1. Just before $\mathsf{special\ message}$ of session k appears on the main thread, $2N$ *look-ahead threads* are created. Namely, from $\mathsf{special\ message}$ of session k (inclusive), the main thread of $H_{k-1:17}$ is executed $2N$ times with fresh randomness by rewinding all the parties including $\mathcal{Z}$ and $\mathcal{A}$.
   If there are at least two look-ahead threads on which Round 3 of session k is accepting, $\{\tilde{r}_{i,b}, \tilde{\omega}^{i,b}_{\mathrm{com}}\}$ are defined as follows.
   (a) For every u, v such that $1 \leq u < v \leq 2N$, if Round 3 of session k is accepting both on the u-th look-ahead thread and on the v-th one, and a valid decommitment of $\mathsf{com}_{i,b}$ is extractable from ExtCom' on these threads, then $\tilde{r}_{i,b}$ and $\tilde{\omega}^{i,b}_{\mathrm{com}}$ are defined to be the extracted decommitment.
   (b) If $\tilde{r}_{i,b}$ and $\tilde{\omega}^{i,b}_{\mathrm{com}}$ are not defined by the above process, then $\tilde{r}_{i,b} = \tilde{\omega}^{i,b}_{\mathrm{com}} = \bot$.
2. Then, the main thread is resumed from $\mathsf{special\ message}$ of session k, and if Round 3 of session k is accepting on the main thread, the following are done. If there are less than two look-ahead threads on which Round 3 of session k is accepting, the experiment is aborted with output Abort_1. Otherwise, the simulation trapdoor σ is extracted based on the information on the look-ahead threads and the main thread; if a valid simulation trapdoor is not extractable, the experiment is aborted with output Abort_1. Next, $x^* = (x^*_1, \ldots, x^*_\kappa)$ is defined as follows: For every $i \in [\kappa]$, let b^*_i be the bit such that $(\tilde{r}_{i,b^*_i}, \tilde{\omega}^{i,b^*_i}_{\mathrm{com}})$ is a valid decommitment of com_{i,b^*_i} and furthermore it holds that $z_{i,b^*_i} = \tilde{r}_{i,b^*_i} \oplus r'_{i,b^*_i}$; if there is no such b^*_i, the experiment is aborted with output Abort_1, and if b^*_i is not uniquely determined, $b^*_i := 0$; then, define $x^*_i := 1 - b^*_i$.

Hybrid $H_{k:2}$ is the same as $H_{k:1}$ except that in session k on the main thread, if P_1 is corrupted, then Π_{FS} in session k is switched to simulation, i.e., fs_4 is generated by using σ as the witness.

Hybrid $H_{k:3}$ is the same as $H_{k:2}$ except that in session k on the main thread, if P_1 is corrupted, then the value committed by NMCom is switched to a all-zero string.

Hybrid $H_{k:4}$ is the same as $H_{k:3}$ except that in session k on the main thread, if P_1 is corrupted, then $\{\mathsf{C}^{i,b}_{\mathrm{lab}}\}$ are generated by committing to all-zero strings by using Com.

Hybrid $H_{k:5}$ is the same as $H_{k:4}$ except that in session k on the main thread, if P_1 is corrupted, then C_{gc} is generated in a way that it can be opened to any value using the equivocality and $\mathsf{open}_{\mathsf{C}_{\mathrm{gc}}}$ is computed by the equivocality.

Hybrid $H_{k:6}$ is the same as $H_{k:5}$ except that in session k on the main thread, if P_1 is corrupted, then $\{W_{i,b}\}$ are generated by $W_{i,x^*_i} := Z_{i,x^*_i} \oplus \mathsf{H}(f^{-\kappa}_{i,x^*_i}(z_{i,x^*_i}))$ and $W_{i,1-x^*_i} \leftarrow \{0,1\}^\kappa$.

Hybrid $H_{k:7}$ is the same as $H_{k:6}$ except that in session k on the main thread, if P_1 is corrupted, then GC and labels are generated by simulation, i.e., as follows.

1. Compute $v_1 = F_1(x^*, y)$ and $v_2 = F_2(x^*, y)$, where y is the input of P_2 in session k.
2. Compute $(\mathsf{GC}^*, \{Z_i^*\}) \leftarrow \mathsf{SimGC}(1^\kappa, F_1', v_1')$, where $v_1' = (v_1, \tilde{\mathsf{enc}}, \tilde{\mathsf{mac}})$, $\tilde{\mathsf{enc}} \leftarrow \mathsf{SKE.Enc}_{\mathsf{sk}_{\mathrm{ske}}}(v_2)$, and $\tilde{\mathsf{mac}} = \mathsf{MAC}_{\mathsf{sk}_{\mathrm{mac}}}(\tilde{\mathsf{enc}})$ for random $\mathsf{sk}_{\mathrm{ske}}$ and $\mathsf{sk}_{\mathrm{mac}}$.
3. Set $\mathsf{GC} := \mathsf{GC}^*$ and $Z_{i,x_i^*} := Z_i^*$. (Labels $\{Z_{i,1-x_i^*}\}$ are not used in $H_{k:6}$.)

Hybrid $H_{k:8}$ is the same as $H_{k:7}$ except that in session k on the main thread, if P_1 is corrupted, then honest P_2's output v_2 is computed as follows.

1. If $\mathsf{MAC}_{\mathsf{sk}_{\mathrm{mac}}}(\mathsf{enc}) \neq \mathsf{mac}$, P_2 outputs $\perp$. Otherwise, it outputs $F_2(x^*, y)$.

Hybrid $H_{k:9}$ is the same as $H_{k:8}$ except that in session k on the main thread, if P_1 is corrupted, then $\tilde{\mathsf{enc}}$ is generated by $\tilde{\mathsf{enc}} \leftarrow \mathsf{SKE.Enc}_{\mathsf{sk}_{\mathrm{ske}}}(0^\kappa)$ during the generation of GC and labels.

Hybrid $H_{k:10}$ is the same as $H_{k:9}$ except that in session k on the main thread, if P_2 is corrupted, then wit_4 is extracted in session k as follows.

1. Just before $\mathsf{special\ message}$ of session k appears on the main thread, N look-ahead threads are created. Namely, from $\mathsf{special\ message}$ of session k (inclusive), the main thread of $H_{k:9}$ is executed N times with fresh randomness by rewinding all the parties including $\mathcal{Z}$ and $\mathcal{A}$.
2. The main thread is resumed from $\mathsf{special\ message}$ of session k. If Round 4 of session is accepting on the main thread, extract wit_4 from NMCom using the information on the look-ahead threads and the main thread; if the extraction fails or wit_4 is not a valid witness for st_4, the experiment is aborted with output Abort_2.

Hybrid $H_{k:11}$ is the same as $H_{k:10}$ except that in session k on the main thread, if P_2 is corrupted, then honest P_1's output v_1 is computed as follows.

1. Parse the extracted wit_4 as $\mathsf{wit}_2 \| \{g_{i,b}\} \| \omega_{\mathrm{leEnc}}$, where $\mathsf{wit}_2 = (y', \varOmega, \mathsf{GC}, \omega_{\mathrm{gc}}, \mathsf{open}_{\mathsf{C}_{\mathrm{gc}}}, \{Z_{i,b}, \omega_{\mathrm{lab}}^{i,b}\})$ and $y' = (y, \mathsf{sk}_{\mathrm{ske}}, \mathsf{sk}_{\mathrm{mac}}, \omega_{\mathrm{enc}})$.
2. Set $v_1 := F_1(x, y)$ if the message m_4 in Round 4 is accepting and set $v_1 := \perp$ otherwise.

Hybrid $H_{k:12}$ is the same as $H_{k:11}$ except that in session k on the main thread, if P_2 is corrupted, then $m_5 = (\mathsf{enc}, \mathsf{mac})$ is generated using the keys $\mathsf{sk}_{\mathrm{ske}}, \mathsf{sk}_{\mathrm{mac}}$ and the randomness ω_{enc} in the extracted wit_4.

Hybrid $H_{k:13}$ is the same as $H_{k:12}$ except that in session k on the main thread, if P_2 is corrupted, then $\mathsf{pk}_{\mathrm{le}}$ is switched to a lossy public key, and $\mathsf{CT}_{\mathrm{gc}}$ is no longer decrypted in Round 5.

Hybrid $H_{k:14}$ is the same as $H_{k:13}$ except that in session k on the main thread, if P_2 is corrupted, then zap_2 is generated by using $(\{1 - x_i, r_{i,1-x_i}, \omega_{\mathrm{com}}^{i,1-x_i}, \omega_{\mathrm{ext}}^{i,1-x_i}, \perp\}_{i\in[\kappa]}, \mathsf{sk}_{\mathrm{le}}, \omega_{\mathrm{leGen}})$ as the witness (i.e, by using a witness for the fact that $\mathsf{pk}_{\mathrm{le}}$ is a lossy public key).

Hybrid $H_{k:15}$ is the same as $H_{k:14}$ except that in session k on the main thread, if P_2 is corrupted, then $z_{i,b}$ is generated by $z_{i,b} := r_{i,b} \oplus r'_{i,b}$ for every $i \in [\kappa]$ and $b \in \{0,1\}$.

Hybrid $H_{k:16}$ is the same as $H_{k:15}$ except that in session k on the main thread, if P_2 is corrupted, then $\mathsf{ext}_1^{i,b}$ and $\mathsf{ext}_3^{i,b}$ are generated by committing to $r_{i,b} \| \omega_{\mathrm{com}}^{i,b}$ correctly using ExtCom' for every $i \in [\kappa]$ and $b \in \{0,1\}$.

Hybrid $H_{k:17}$ is the same as $H_{k:16}$ except that in session k on the main thread, if P_2 is corrupted, then zap_2 is generated by using $(\{1, r_{i,1}, \omega^{i,1}_{\mathrm{com}}, \omega^{i,1}_{\mathrm{ext}}, \bot\}_{i\in[\kappa]}, \mathsf{sk}_{\mathrm{le}}, \omega_{\mathrm{leGen}})$ as the witness (i.e., by using a witness for the fact that $\{(\mathsf{com}_{i,1}, \mathsf{ext}^{i,1}_1, \mathsf{ext}^{i,1}_3, z_{i,1})\}$ are correctly constructed and $\mathsf{pk}_{\mathrm{le}}$ is a lossy public key).

Finally, hybrid $H_{m+1:1}$ is identical with the ideal experiment.

Remark 1. The hybrid experiments $H_{k:1}, \ldots, H_{k:17}$ are designed so that no look-ahead thread is created after $\mathsf{special\ message}$ of session k.

Our goal is to show that the output of the first hybrid $H_{0:17}$ and that of the last hybrid $H_{m+1:1}$ are indistinguishable (more precisely, are distinguishable with advantage at most $\varepsilon/2$.) Toward this goal, we show the indistinguishability among the outputs of the intermediate hybrids. Also, for a technical reason, we show that the following condition holds with high probability in each hybrid: In a session in which P_2 is corrupted, if the session is accepting then the NMCom commitment from P_2 is valid and the committed value wit_4 is a valid witness for st_4. (Notice that if this condition holds, then we can extract the input of P_2 from NMCom.) Formally, for every $k' \in [m]$ let $\textsc{Bad}_{k'}$ be the event that in the k'-th session on the main thread, P_2 is corrupted, Round 4 is accepting, but the committed value wit_4 of NMCom is not a valid witness for st_4, and let $\rho_{k:j:k'}$ be the probability that $\textsc{Bad}_{k'}$ occurs in $H_{k:j}$. We first observe that $\rho_{0:17:k'}$ is negligible for every k' (i.e., $\textsc{Bad}_{k'}$ occurs in the real experiment with negligible probability for every k').

Lemma 1. *For every $k' \in [m]$, $\rho_{0:17:k'} = \mathsf{negl}(\kappa)$.*

Proof: This lemma follows from the soundness of Π_{FS} because P_2 proves in Π_{FS} that a valid witness for st_4 is committed in NMCom. □

Now we are ready to show the indistinguishability among the outputs of the hybrids. Let $\mathsf{H}_{k:j}$ be the random variable representing the output of $H_{k:j}$. We first prove the following lemma.

Lemma 2. *For every $k \in [m]$, the following two inequalities hold.*

1. $$\left|\Pr[D(\mathsf{H}_{k-1:17}) = 1] - \Pr[D(\mathsf{H}_{k:17}) = 1]\right| \le \frac{2\kappa+1}{\sqrt{N}} + \frac{1}{N} + \mathsf{negl}(\kappa)\ . \quad (2)$$
2. $$\forall k' \in [m] : \rho_{k:17:k'} \le \rho_{k-1:17:k'} + \mathsf{negl}(\kappa)\ . \quad (3)$$

Proof: Fix any $k \in [m]$. From Lemma 1, it suffices to show that the above two inequalities hold whenever we have

$$\forall k' \in [m] : \rho_{k-1:17:k'} = \mathsf{negl}(\kappa)\ . \quad (4)$$

In what follows we show claims about the outputs of each neighboring hybrids.

Claim 1. $\left|\Pr[D(\mathsf{H}_{k-1:17}) = 1] - \Pr[D(\mathsf{H}_{k:1}) = 1]\right| \le \frac{2\kappa+1}{\sqrt{N}} + \mathsf{negl}(\kappa)$. *Furthermore, for every $k' \in [m]$, $\rho_{k:1:k'} \le \rho_{k-1:17:k'}$.*

Proof: We first show the indistinguishability of the outputs of the hybrids. The output of $H_{k:1}$ differs from that of $H_{k-1:17}$ only when it outputs Abort_1 in session k, and $H_{k:1}$ outputs Abort_1 in session k only when one of the following events occur.

Event E_1: Round 3 of session k is accepting on less than two look-ahead threads but it is accepting on the main thread.
Event E_2: The extraction of the simulation trapdoor σ fails.
Event $E_{3,i}$ $(i \in [\kappa])$: There is no b_i^* such that $(\tilde{r}_{i,b_i^*}, \tilde{\omega}_{\mathrm{com}}^{i,b_i^*})$ is a valid decommitment of com_{i,b_i^*} and $z_{i,b_i^*} = \tilde{r}_{i,b_i^*} \oplus r'_{i,b_i^*}$.

From Markov's inequality, E_1 occurs with probability at most $1/\sqrt{N}$, and from the extractability of $\mathit{\Pi}_{\mathrm{WIPOK}}$ (inside $\mathit{\Pi}_{\mathrm{FS}}$), E_2 occurs with negligible probability. In what follows, we show that for every $i \in [\kappa]$, $E_{3,i}$ occurs with probability at most $2/\sqrt{N} + \mathsf{negl}(\kappa)$. Let prefix be any prefix of the execution of $H_{k:1}$ up until the creation of the look-ahead threads in the k-th session (exclusive). We show that for every i, under that condition that a prefix of the execution of $H_{k:1}$ is prefix, $E_{3,i}$ occurs with probability at most $2/\sqrt{N} + \mathsf{negl}(\kappa)$. For $b \in \{0, 1\}$, let us say that session k is (i, b)-*good* if its Round 3 is accepting, a valid decommitment $(r_{i,b}, \omega_{\mathrm{com}}^{i,b})$ of $\mathsf{com}_{i,b}$ is correctly committed in $(\mathsf{ext}_1^{i,b}, \mathsf{ext}_2^{i,b}, \mathsf{ext}_3^{i,b})$, and it holds that $z_{i,b} = r_{i,b} \oplus r'_{i,b}$. From the extractability of ExtCom', one of the following events occurs whenever $E_{3,i}$ occurs.

- Session k is $(i, 0)$-good on the main thread, but it is $(i, 0)$-good on less than two look-ahead threads. If session k is $(i, 0)$-good on the main thread with probability at most $1/\sqrt{N}$, this event occurs with probability at most $1/\sqrt{N}$. Furthermore, even if session k is $(i, 0)$-good on the main thread with probability at least $1/\sqrt{N}$, this event occurs with probability at most $1/\sqrt{N}$, since from Markov's inequality, session k is $(i, 0)$-good on less than two look-ahead threads with probability at most $1/\sqrt{N}$.
- Session k is $(i, 1)$-good on the main thread, but it is $(i, 1)$-good on less than two look-ahead threads. From the same argument as above, this event occurs with probability at most $1/\sqrt{N}$.
- On the main thread, Round 3 of session k is accepting but it is neither $(i, 0)$-good nor $(i, 1)$-good. From the soundness of $\mathit{\Pi}_{\mathrm{ZAP}}$, this event occurs with negligible probability.

Hence, for every $i \in [\kappa]$, $E_{3,i}$ occurs with probability at most $2/\sqrt{N} + \mathsf{negl}(\kappa)$. From the union bound, the probability that there exists $i \in [\kappa]$ such that $E_{3,i}$ occurs is at most $2\kappa/\sqrt{N} + \mathsf{negl}(\kappa)$. Since prefix is any prefix, we conclude that even without conditioning that a prefix of the execution of $H_{k:1}$ is prefix, the probability that there exists $i \in [\kappa]$ such that $E_{3,i}$ occurs is at most $2\kappa/\sqrt{N} + \mathsf{negl}(\kappa)$. Hence, the indistinguishability follows.

We next observe that we have $\rho_{k:1:k'} \leq \rho_{k-1:17:k'}$. This is because the main thread of $H_{k:1}$ is identical with that of $H_{k-1:17}$ until the experiment outputs Abort_1 in session k, and when it outputs Abort_1, the experiment is aborted immediately and no further NMCom commitment is created. □

Claim 2. $|\Pr[D(\mathsf{H}_{k:1}) = 1] - \Pr[D(\mathsf{H}_{k:2}) = 1]| \leq \mathsf{negl}(\kappa)$. *Furthermore, for every* $k' \in [m]$, $\rho_{k:2:k'} \leq \rho_{k:1:k'} + \mathsf{negl}(\kappa)$.

Proof: We first show the indistinguishability of the outputs. $H_{k:2}$ differs from $H_{k:1}$ only in that in session k on the main thread, the simulation trapdoor is used in Π_{SWIAOK} (inside Π_{FS}) as the witness. We then observe that, since no look-ahead thread is created after Round 2 of session k on the main thread, Π_{SWIAOK} in session k is not rewound after its second round, and so the indistinguishability follows from the witness indistinguishability of Π_{SWIAOK}.

We next observe that $\rho_{k:2:k'} \leq \rho_{k:1:k'} + \mathsf{negl}(\kappa)$ follows from the statistical witness indistinguishability of Π_{SWIAOK}. Specifically, if $\rho_{k:2:k'}$ differs from $\rho_{k:1:k'}$ with non-negligible amount, we can break the statistical witness indistinguishability of Π_{SWIAOK} by checking whether $\text{BAD}_{k'}$ occurs or not by extracting the committed value of the NMCom commitment by brute force. □

Claim 3. $|\Pr[D(\mathsf{H}_{k:2}) = 1] - \Pr[D(\mathsf{H}_{k:3}) = 1]| \leq \mathsf{negl}(\kappa)$. *Furthermore, for every* $k' \in [m]$, $\rho_{k:3:k'} \leq \rho_{k:2:k'} + \mathsf{negl}(\kappa)$.

Proof: We first show the indistinguishability of the outputs. $H_{k:3}$ differs from $H_{k:2}$ only in the committed value of NMCom in session k on the main thread. We then observe that, since no look-ahead thread is created after Round 2 of session k on the main thread, NMCom in session k on the main thread is not rewound. Hence, the indistinguishability follows from the hiding property of NMCom.

We next observe that $\rho_{k:3:k'} \leq \rho_{k:2:k'} + \mathsf{negl}(\kappa)$ follows from the non-malleability of NMCom. Specifically, if $\rho_{k:3:k'}$ differs from $\rho_{k:2:k'}$ with non-negligible amount, we can break the non-malleability of NMCom by considering an adversary that internally emulates $H_{k:2}$ while obtaining the NMCom commitment of session k from the external committer and forwarding the NMCom commitment of session k' to the external receiver. We remark that since NMCom is public coin, we can emulate $H_{k:2}$ while forwarding the NMCom commitment of session k' to the external receiver (without worrying that it can be rewound). □

Claim 4. $|\Pr[D(\mathsf{H}_{k:3}) = 1] - \Pr[D(\mathsf{H}_{k:4}) = 1]| \leq \mathsf{negl}(\kappa)$. *Furthermore, for every* $k' \in [m]$, $\rho_{k:4:k'} \leq \rho_{k:3:k'} + \mathsf{negl}(\kappa)$.

Proof: We first show the indistinguishability of the outputs. $H_{k:4}$ differs from $H_{k:3}$ only in the committed values of Com in session k. Hence, the indistinguishability follows from the hiding properly of Com.

We next observe that $\rho_{k:4:k'} \leq \rho_{k:3:k'} + \mathsf{negl}(\kappa)$ follows from the hiding property of Com and the extractability of NMCom. Specifically, if $\rho_{k:4:k'}$ differs from $\rho_{k:3:k'}$ with non-negligible amount, we can break the hiding property of Com by considering an adversary that internally emulates $H_{k:3}$ while obtaining $\{\mathsf{C}^{i,b}_{\text{lab}}\}$ of session k from the external committer and extracting the committed value of the NMCom commitment in session k'. We remark that there are two subtleties:

1. The extraction from NMCom requires rewinding, and hence the Com commitment in session k might be rewound during the extraction from NMCom. Nevertheless, we can use the hiding property of Com since Com is non-interactive (which trivially implies that Com is hiding even when it is rewound).

2. The NMCom commitment in session k' might be rewound in $H_{k:3}$ during the creation of the look-ahead threads. Nevertheless, we can use its extractability since NMCom is public coin (which implies that an adversary can internally emulate $H_{k:3}$ while forwarding NMCom to an external receiver).

$\square$

We remark that the statement of Claim 4 also holds w.r.t. $H_{k:j}$ and $H_{k:j+1}$ for $j = 4, \ldots, 8$. The proofs are similar to the proof of Claim 4: the indistinguishability between the outputs of the hybrids is shown by relying on the security of the components (e.g., the equivocality of Eqcom), and the inequality between $\rho_{k:j:k'}$ and $\rho_{k:j+1:k'}$ is shown by additionally using the extractability of NMCom. We therefore have the following claim.

Claim 5. $|\Pr[D(\mathsf{H}_{k:4}) = 1] - \Pr[D(\mathsf{H}_{k:9}) = 1]| \leq \mathsf{negl}(\kappa)$. *Furthermore, for every* $k' \in [m]$, $\rho_{k:9:k'} \leq \rho_{k:4:k'} + \mathsf{negl}(\kappa)$.

A formal argument for this claim is given in the full version of this paper.

Claim 6. $|\Pr[D(\mathsf{H}_{k:9}) = 1] - \Pr[D(\mathsf{H}_{k:10}) = 1]| \leq \frac{1}{N} + \mathsf{negl}(\kappa)$. *Furthermore, for every* $k' \in [m]$, $\rho_{k:10:k'} \leq \rho_{k:9:k'}$.

Proof: We first show the indistinguishability of the outputs. The output of $H_{k:10}$ differs from that of $H_{k:9}$ only when it outputs Abort_2 in session k, and $H_{k:10}$ outputs Abort_2 in session k only when one of the following happens.

1. Round 4 of session k does not complete on the look-ahead threads but it completes on the main thread.
2. Even though Round 3 of session k completes on a look-ahead thread and the main thread, a valid witness wit_4 for st_4 is not extractable.

The former occurs with probability at most $1/N$ from the swapping argument. The latter occurs with negligible probability since we have $\rho_{k:9:k'} = \mathsf{negl}(\kappa)$ from Equation (4). (Notice that when P_2 is corrupted in session k, the main thread of $H_{k:9}$ proceeds identically with that of $H_{k-1:17}$.) Hence, the indistinguishability follows.

We next observe that we have $\rho_{k:10:k'} \leq \rho_{k:9:k'}$. This is because $H_{k:10}$ is identical with $H_{k:9}$ until it outputs Abort_2 in session k, and when it outputs Abort_2, the experiment is immediately aborted. $\square$

We remark that the statement of Claim 4 also holds w.r.t. $H_{k:j}$ and $H_{k:j+1}$ for $j = 10, \ldots, 14$; the proofs are similar to the proof of Claim 4. We therefore have the following claim.

Claim 7. $|\Pr[D(\mathsf{H}_{k:10}) = 1] - \Pr[D(\mathsf{H}_{k:15}) = 1]| \leq \mathsf{negl}(\kappa)$. *Furthermore, for every* $k' \in [m]$, $\rho_{k:15:k'} \leq \rho_{k:10:k'} + \mathsf{negl}(\kappa)$.

A formal argument for this claim is given in the full version of this paper.

Claim 8. $|\Pr[D(\mathsf{H}_{k:15}) = 1] - \Pr[D(\mathsf{H}_{k:16}) = 1]| \leq \mathsf{negl}(\kappa)$. *Furthermore, for every* $k' \in [m]$, $\rho_{k:16:k'} \leq \rho_{k:15:k'} + \mathsf{negl}(\kappa)$.

Proof: We first show the indistinguishability of the outputs. $H_{k:16}$ differs from $H_{k:15}$ only in that ext_1^{i,x_i} and ext_3^{i,x_i} are generated by committing to $r_{i,x_i} \| \omega_{\mathrm{com}}^{i,x_i}$ correctly using ExtCom' (rather than by executing "fake" ExtCom'). Since ext_1^{i,x_i} consists of Com commitments to κ pairs of 2-out-of-2 secret shares and ext_3^{i,x_i} consists of the revealing of the shares that are selected by ext_2^{i,x_i}, we use the hiding property of Com to show the indistinguishability. Assume for contradiction that the output of $H_{k:15}$ and that of $H_{k:16}$ are distinguishable. Then, we consider an adversary $\mathcal{A}_{\mathsf{Com}}$ that internally emulates $H_{k:16}$ honestly except for the following.

– In Round 1 of session k on the main thread, $\mathcal{A}_{\mathsf{Com}}$ obtains $\{\mathsf{ext}_1^{i,x_i}\}$ from the external committer, where each ext_1^{i,x_i} consists of Com commitments whose committed values are either all-zero strings or pairs of 2-out-of-2 secret shares of $r_{i,x_i} \| \omega_{\mathrm{com}}^{i,x_i}$.
– In Round 3 of session k on the main thread, $\mathcal{A}_{\mathsf{Com}}$ computes ext_3^{i,x_i} as in the correct execution of ExtCom' assuming that the values committed in ext_1^{i,x_i} are the pairs of 2-out-of-2 secret shares.

When $\mathcal{A}_{\mathsf{Com}}$ receives Com commitment to the pairs of 2-out-of-2 secret shares, the internally emulated experiment is identical with $H_{k:16}$. When $\mathcal{A}_{\mathsf{Com}}$ receives Com commitments to all-zero strings, the internally emulated experiment is identical with $H_{k:15}$ (since in this case, ext_3^{i,x_i} consists of random strings that are independent of other parts of the experiment). Hence, we derive a contradiction.

Remark 2. ExtCom' in session k might be rewound in $H_{k:15}$ and $H_{k:16}$ since look-ahead threads might be created after Round 1 of session k on the main thread (for simulating other sessions). Nevertheless, $\mathcal{A}_{\mathsf{Com}}$ can emulate $H_{k:16}$ while obtaining ext_1^{i,x_i} from the external committer because (1) the randomness for generating ext_1^{i,x_i} and ext_3^{i,x_i} is not used after Round 1 and (2) ext_3^{i,x_i} on look-ahead thread is a random string (and thus can be simulated trivially).

We next observe that $\rho_{k:16:k'} \leq \rho_{k:15:k'} + \mathsf{negl}(\kappa)$ follows from the indistinguishability of Com. The argument for this statement is similar to the one in the proof of Claim 4. □

Claim 9. $|\Pr[D(\mathsf{H}_{k:16}) = 1] - \Pr[D(\mathsf{H}_{k:17}) = 1]| \leq \mathsf{negl}(\kappa)$. *Furthermore, for every* $k' \in [m]$, $\rho_{k:17:k'} \leq \rho_{k:16:k'} + \mathsf{negl}(\kappa)$.

Proof: We first show the indistinguishability of the outputs. $H_{k:17}$ differs from $H_{k:16}$ only in the witness used in Π_{ZAP}. Hence, the indistinguishability follows form the witness indistinguishability of Π_{ZAP}.

We next observe that $\rho_{k:17:k'} \leq \rho_{k:16:k'} + \mathsf{negl}(\kappa)$ follows from the witness indistinguishability of of Π_{ZAP} and the extractability of NMCom. The argument for this statement is similar to the one in the proof of Claim 4. □

By combining Claims 1 - 9, we conclude that the two inequalities in the statement of Lemma 2 hold for k. This concludes the proof of Lemma 2. □

We next show that the output of $H_{m:17}$ and that of the last hybrid $H_{m+1:1}$ (i.e., the ideal experiment) is indistinguishable.

Lemma 3.

$$|\Pr[D(\mathsf{H}_{m:17})=1]-\Pr[D(\mathsf{H}_{m+1:1})=1]| \le m\left(\frac{2\kappa+1}{\sqrt{N}}+\frac{1}{N}+\mathsf{negl}(\kappa)\right)+\mathsf{negl}(\kappa)\ .$$

Proof: We consider an intermediate hybrid $\hat{H}_{m:17}$ that is the same as $H_{m:17}$ except that the extractions from ExtCom', $\mathit{\Pi}_{\text{WIPOK}}$, and NMCom are performed by brute force rather than by rewinding (hence, no look-ahead thread is created in $\hat{H}_{m:17}$). That is, $\hat{H}_{m:17}$ is the same as $H_{m:17}$ except that in a session in which P_1 is corrupted, the simulation trapdoor σ and the committed values $\{\tilde{r}_{i,b}\|\tilde{\omega}^{i,b}_{\text{com}}\}$ of ExtCom' are extracted by brute force, and in a session in which P_2 is corrupted, the committed value wit_4 of NMCom is extracted by brute force.

First, we observe that the output of $\hat{H}_{m:17}$ and that of $H_{m+1:1}$ are identical, that is,

$$\Pr\left[D(\hat{\mathsf{H}}_{m:17})=1\right]=\Pr[D(\mathsf{H}_{m+1:1})=1]\ . \tag{5}$$

This can be seen by inspection: in $\hat{H}_{m:17}$, all the messages of the honest parties are generated in the same way as in $H_{m+1:1}$ and the outputs of the honest parties are computed in the same way as in $H_{m+1:1}$.

Next, we show the indistinguishability between the output of $\hat{H}_{m:17}$ and that of $H_{m:17}$. We first observe that when $H_{m:17}$ outputs neither Abort_1 nor Abort_2, the messages and outputs of the honest parties are statistically close to those that would be computed with brute-force extractions (i.e., as in $\hat{H}_{m:17}$).

- When $H_{m:17}$ does not output Abort_2, a valid witness wit_4 for st_4 is extracted in every session in which P_2 is corrupted, and the same wit_4 would be also extracted by brute-force extraction. (This is because from Lemmas 1 and 2, the probability that $\textsc{Bad}_{k'}$ occurs in $H_{m:17}$ is negligible for every $k' \in [m]$.)
- When $H_{m:17}$ does not output Abort_1, a valid simulation trapdoor σ is extracted in every session in which P_1 is corrupted, and although a different simulation trapdoor might be extracted as σ by brute-force extraction, the information about σ is statistically hidden because a statistical witness-indistinguishable argument $\mathit{\Pi}_{\text{SWIAOK}}$ is used in $\mathit{\Pi}_{\text{FS}}$.
- When $H_{m:17}$ does not output Abort_1, an implicit input x^* is defined according to the values extracted from ExtCom' in every session in which P_1 is corrupted. If pk_{le} is an injective public key in such a session, the same x^* would be defined by brute-force extraction except with negligible probability. (This is because if pk_{le} is an injective public key, the soundness of $\mathit{\Pi}_{\text{ZAP}}$ guarantees that for every $i \in [\kappa]$, there is a unique $b^*_i \in \{0,1\}$ such that $(\mathsf{ext}_1^{i,b^*_i}, \mathsf{ext}_2^{i,b^*_i}, \mathsf{ext}_3^{i,b^*_i})$ is a correct ExtCom' commitment to a valid decommitment $(r_{i,b^*_i}, \omega^{i,b^*_i}_{\text{com}})$ of com_{i,b^*_i} and $z_{i,b^*_i} = r_{i,b^*_i} \oplus r'_{i,b^*_i}$.) If pk_{le} is a lossy public key in such a session, a different x^* might be defined by brute-force extraction.[5] However, x^* is used only to compute CT_{gc} and the output of honest

[5] This is because from an *invalid* ExtCom' commitment, the brute-force extractor always outputs $\perp$ but the rewinding extractor can output any value (in particular, it can output even a valid decommitment of Com).

P_2, where $\mathsf{CT}_{\mathrm{gc}}$ is generated by $\mathsf{LE.Enc}_{\mathsf{pk}_{\mathrm{le}}}(\cdot)$ (which statistically hides the plaintext when $\mathsf{pk}_{\mathrm{le}}$ is lossy) and the output of P_2 is $\perp$ when mac in Round 5 is rejecting (which is almost always the case when $\mathsf{pk}_{\mathrm{le}}$ is lossy because $\mathsf{sk}_{\mathrm{mac}}$ is statistically hidden in this case). Thus, the information about x^* is statistically hidden in this case.

We next analyze the probability that $H_{m:17}$ outputs Abort_1 or Abort_2. From Lemma 2, we have

$$|\Pr[D(\mathsf{H}_{0:17}) = 1] - \Pr[D(\mathsf{H}_{m:17}) = 1]| \le m\left(\frac{2\kappa+1}{\sqrt{N}} + \frac{1}{N} + \mathsf{negl}(\kappa)\right) .$$

Then, since $H_{0:17}$ (i.e., the real experiment) never output Abort_1 or Abort_2, we have that $H_{m:17}$ outputs Abort_1 or Abort_2 with probability at most

$$m\left(\frac{2\kappa+1}{\sqrt{N}} + \frac{1}{N} + \mathsf{negl}(\kappa)\right) .$$

By combining the above, we obtain

$$\left|\Pr\left[D(\hat{\mathsf{H}}_{m:17}) = 1\right] - \Pr[D(\mathsf{H}_{m:17}) = 1]\right| \le m\left(\frac{2\kappa+1}{\sqrt{N}} + \frac{1}{N} + \mathsf{negl}(\kappa)\right) + \mathsf{negl}(\kappa) . \tag{6}$$

From Equations (5) and (6), we obtain

$$|\Pr[D(\mathsf{H}_{m:17}) = 1] - \Pr[D(\mathsf{H}_{m+1:1}) = 1]| \le m\left(\frac{2\kappa+1}{\sqrt{N}} + \frac{1}{N} + \mathsf{negl}(\kappa)\right) + \mathsf{negl}(\kappa) .$$

□

From Lemmas 2 and 3 and $N = (10m\kappa/\varepsilon)^2$, we have

$$\begin{aligned}
&\left|\Pr[D(\mathrm{EXEC}_{\Pi_{\mathrm{2PC}},\mathcal{A},\mathcal{Z}}(\kappa)) = 1] - \Pr\left[D(\mathrm{EXEC}_{\Pi(\mathcal{F}),\mathcal{S},\mathcal{Z}}(\kappa)) = 1\right]\right| \\
&= |\Pr[D(\mathsf{H}_{0:17}) = 1] - \Pr[D(\mathsf{H}_{m+1:1}) = 1]| \\
&\le 2m\left(\frac{2\kappa+1}{\sqrt{N}} + \frac{1}{N} + \mathsf{negl}(\kappa)\right) + \mathsf{negl}(\kappa) \le \frac{5m\kappa}{\sqrt{N}} = \frac{\varepsilon}{2} .
\end{aligned}$$

This contradicts to Equation (1). This concludes the proof of Theorem 1.

References

1. Barak, B., Prabhakaran, M., Sahai, A.: Concurrent non-malleable zero knowledge. In: 47th FOCS. pp. 345–354. IEEE Computer Society Press (Oct 2006)
2. Barak, B., Sahai, A.: How to play almost any mental game over the net - Concurrent composition via super-polynomial simulation. In: 46th FOCS. pp. 543–552. IEEE Computer Society Press (Oct 2005)
3. Bellare, M., Hofheinz, D., Yilek, S.: Possibility and impossibility results for encryption and commitment secure under selective opening. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 1–35. Springer, Heidelberg (Apr 2009)

4. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001)
5. Canetti, R., Kushilevitz, E., Lindell, Y.: On the limitations of universally composable two-party computation without set-up assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 68–86. Springer, Heidelberg (May 2003)
6. Canetti, R., Lin, H., Pass, R.: Adaptive hardness and composable security in the plain model from standard assumptions. In: 51st FOCS. pp. 541–550. IEEE Computer Society Press (Oct 2010)
7. Dwork, C., Naor, M.: Zaps and their applications. In: 41st FOCS. pp. 283–293. IEEE Computer Society Press (Nov 2000)
8. Feige, U., Shamir, A.: Witness indistinguishable and witness hiding protocols. In: 22nd ACM STOC. pp. 416–426. ACM Press (May 1990)
9. Feige, U., Shamir, A.: Zero knowledge proofs of knowledge in two rounds. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 526–544. Springer, Heidelberg (Aug 1990)
10. Garg, S., Goyal, V., Jain, A., Sahai, A.: Concurrently secure computation in constant rounds. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 99–116. Springer, Heidelberg (Apr 2012)
11. Garg, S., Mukherjee, P., Pandey, O., Polychroniadou, A.: The exact round complexity of secure computation. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 448–476. Springer, Heidelberg (May 2016)
12. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press (May 1987)
13. Goyal, V., Jain, A.: On concurrently secure computation in the multiple ideal query model. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 684–701. Springer, Heidelberg (May 2013)
14. Goyal, V., Lin, H., Pandey, O., Pass, R., Sahai, A.: Round-efficient concurrently composable secure computation via a robust extraction lemma. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part I. LNCS, vol. 9014, pp. 260–289. Springer, Heidelberg (Mar 2015)
15. Goyal, V., Pandey, O., Richelson, S.: Textbook non-malleable commitments. Cryptology ePrint Archive, Report 2015/1178 (2015), http://eprint.iacr.org/2015/1178
16. Goyal, V., Pandey, O., Richelson, S.: Textbook non-malleable commitments. In: Wichs, D., Mansour, Y. (eds.) 48th ACM STOC. pp. 1128–1141. ACM Press (Jun 2016)
17. Hemenway, B., Libert, B., Ostrovsky, R., Vergnaud, D.: Lossy encryption: Constructions from general assumptions and efficient selective opening chosen ciphertext security. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 70–88. Springer, Heidelberg (Dec 2011)
18. Katz, J., Ostrovsky, R.: Round-optimal secure two-party computation. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 335–354. Springer, Heidelberg (Aug 2004)
19. Kiyoshima, S.: Round-efficient black-box construction of composable multi-party computation. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 351–368. Springer, Heidelberg (Aug 2014)
20. Kiyoshima, S., Manabe, Y., Okamoto, T.: Constant-round black-box construction of composable multi-party computation protocol. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 343–367. Springer, Heidelberg (Feb 2014)

21. Lapidot, D., Shamir, A.: Publicly verifiable non-interactive zero-knowledge proofs. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO'90. LNCS, vol. 537, pp. 353–365. Springer, Heidelberg (Aug 1991)
22. Lin, H., Pass, R.: Non-malleability amplification. In: Mitzenmacher, M. (ed.) 41st ACM STOC. pp. 189–198. ACM Press (May / Jun 2009)
23. Lin, H., Pass, R.: Black-box constructions of composable protocols without set-up. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 461–478. Springer, Heidelberg (Aug 2012)
24. Lin, H., Pass, R., Venkitasubramaniam, M.: A unified framework for concurrent security: universal composability from stand-alone non-malleability. In: Mitzenmacher, M. (ed.) 41st ACM STOC. pp. 179–188. ACM Press (May / Jun 2009)
25. Lindell, Y.: Bounded-concurrent secure two-party computation without setup assumptions. In: 35th ACM STOC. pp. 683–692. ACM Press (Jun 2003)
26. Lindell, Y.: Lower bounds for concurrent self composition. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 203–222. Springer, Heidelberg (Feb 2004)
27. Lindell, Y., Pinkas, B.: A proof of security of Yao's protocol for two-party computation. Journal of Cryptology 22(2), 161–188 (Apr 2009)
28. Malkin, T., Moriarty, R., Yakovenko, N.: Generalized environmental security from number theoretic assumptions. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 343–359. Springer, Heidelberg (Mar 2006)
29. Micali, S., Pass, R., Rosen, A.: Input-indistinguishable computation. In: 47th FOCS. pp. 367–378. IEEE Computer Society Press (Oct 2006)
30. Ostrovsky, R., Richelson, S., Scafuro, A.: Round-optimal black-box two-party computation. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 339–358. Springer, Heidelberg (Aug 2015)
31. Pass, R.: Simulation in quasi-polynomial time, and its application to protocol composition. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 160–176. Springer, Heidelberg (May 2003)
32. Pass, R.: Bounded-concurrent secure multi-party computation with a dishonest majority. In: Babai, L. (ed.) 36th ACM STOC. pp. 232–241. ACM Press (Jun 2004)
33. Pass, R., Lin, H., Venkitasubramaniam, M.: A unified framework for UC from only OT. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 699–717. Springer, Heidelberg (Dec 2012)
34. Peikert, C., Waters, B.: Lossy trapdoor functions and their applications. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC. pp. 187–196. ACM Press (May 2008)
35. Prabhakaran, M., Sahai, A.: New notions of security: Achieving universal composability without trusted setup. In: Babai, L. (ed.) 36th ACM STOC. pp. 242–251. ACM Press (Jun 2004)
36. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS. pp. 162–167. IEEE Computer Society Press (Oct 1986)