Covert Adaptive Adversary Model: A New Adversary Model for Multiparty Computation

Isheeta Nargis and Anwar Hasan

University of Waterloo
Waterloo, Ontario, Canada
isheeta@gmail.com, ahasan@uwaterloo.ca

Abstract. In covert adversary model, the corrupted parties can behave in any possible way like active adversaries, but any party that attempts to cheat is guaranteed to get caught by the honest parties with a minimum fixed probability. That probability is called the deterrence factor of covert adversary model. Security-wise, covert adversary is stronger than passive adversary and weaker than active adversary. It is more realistic than passive adversary model. Protocols for covert adversaries are significantly more efficient than protocols for active adversaries. Covert adversary model is defined only for static corruption. Adaptive adversary model is more realistic than static adversaries. In this article, we define a new adversary model, the covert adaptive adversary model, by generalizing the definition of covert adversary model for the more realistic adaptive corruption. We prove security relations between the new covert adaptive adversary model with existing adversary models like passive adaptive adversary model, active adaptive adversary model and covert static adversary model. We prove the sequential composition theorem for the new adversary model which is necessary to allow modular design of protocols for this new adversary model.
Keywords: Covert Adversary, Covert Adaptive Adversary, Static Adversary, Adaptive Adversary, Passive Adversary, Active Adversary, Simulation-based Security Definition.

1 Introduction

In a secure multiparty computation (MPC) problem, a group of mutually distrusting parties compute a possibly randomized function of their inputs in such a way that the privacy of inputs is maintained and the computed output follows the distribution of the function definition. MPC is a very strong primitive in cryptography since almost all cryptographic problems can be solved, in principle, by a general secure MPC protocol. Yao introduced the concept of MPC in 1982. He described the famous Millionaires' problem where two millionaires want to find out who is richer without revealing the amount of wealth each owns. MPC has subsequently been applied in a variety of areas.

- In the electronic voting problem, some parties want to elect one candidate from a list of candidates such that the result of vote counting is correct and the votes remain private.
- Consider the situation where some hospitals wish to perform research on a certain disease by performing data mining on their patient database, but the hospitals have to maintain the privacy of their patient database.
- Sometimes government security wants to check whether there is any criminal, from a list of criminals, on a particular flight of an airline company. The government does not want to disclose the list of criminals to the airline company, and the airline company does not like to reveal the information of all the passengers on a flight.

There are many other applications of secure MPC such as financial analysis, secure auction, privacy-preserving biometric identification, secure computation on gene sequences, private information retrieval, private set intersection and privacy-preserving machine learning.

The parties do not trust one another. Some parties may try to learn about other parties' inputs. Some parties may try to modify the outputs of other parties. The problems and risks associated with a multiparty

computation problem are modeled by an entity called the ***adversary***. The adversary tries to control parties. A party which is controlled by the adversary is called a ***corrupted party***. A party which is not controlled by the adversary is called an ***honest party***. Depending on the assumption on the computational power of the adversary and the communication channel, there are two types of security models. In ***cryptographic model of security***, it is assumed that the adversary can see all the communication between any pair of parties and the adversary is a probabilistic polynomial-time Turing machine. In ***information-theoretic model of security***, it is assumed that the adversary cannot see the communication between any pair of honest parties and the computational power of the adversary is unlimited. An adversary that corrupts at most t parties is called a ***t-limited adversary*** or a ***threshold adversary***. In that case, the number t is called the ***threshold***.

In order to simplify the analysis, the efficiency of protocols is generally measured in terms of a special parameter called the *security parameter*. All parties and the adversary get the security parameter as an input. The efficiency of MPC protocols are measured by some metrics. One *round* of an MPC protocol is a sequence of steps of the MPC protocol such that each party sends one message to each other party in that sequence. The *round complexity* of an MPC protocol is the number of rounds needed for executing that protocol. The *communication complexity* of an MPC protocol is the total communication (in bits) among the parties during the execution of that MPC protocol. The *computational complexity* of an MPC protocol is the asymptotic computational complexity needed for executing that protocol. Sometimes some other parameters are also used to get an estimate of the computational complexity of an MPC protocol. Many MPC protocols use public key encryption (PKE) scheme as its building blocks. Usually the PKE operations constitute the main bottleneck in the time consumed by an MPC protocol. For this reason, the number of PKE operations performed by each party gives a good measure of the computational complexity of MPC protocols. The number of exponentiation operations performed by each party is another performance metric of MPC protocols since the exponentiation operations take a big amount of time.

In *passive adversary model*, it is assumed that the corrupted parties collaborate together to learn more about the inputs and outputs of the honest parties but the corrupted parties still follow the protocol. In *active adversary model*, the corrupted parties can behave in any possible way, including the violation of the protocol. Active adversary model portrays the real world scenario better than passive adversary model. Active adversary notion is more secure than passive adversary notion. Usually the protocols for passive adversary model are simpler and more efficient than protocols for active adversary model. In *static adversary model*, it is assumed that the adversary selects the parties to corrupt before the protocol starts and the set of corrupted parties remain fixed throughout the execution of the protocol. In *adaptive adversary model*, the adversary can corrupt a party at any time, even after the the execution of the protocol is finished. Adaptive adversary model is more realistic that static adversary model. Adaptive adversary model is a stronger security model than static adversary model. It is possible to design simpler and more efficient protocols for static adversary model than the protocols for adaptive adversary model. Depending on the assumption of erasure, there are two types of adaptive adversary model. In *adaptive adversary model with erasure*, it is assumed that the parties can erase some local data. In *erasure-free adaptive adversary model*, it is assumed that the adversary learns the full history of a party when it corrupts that party. Assuming erasure is unrealistic as complete erasure is sometimes impossible to achieve. Moreover, erasure is a property that cannot be verified by another party. For these reasons, erasure-free adaptive adversary model is more realistic than adaptive adversary model with erasure. The protocols for adaptive adversary model with erasure are simpler than the protocols for erasure-free adaptive adversaries.

Aumann and Lindell [AL10] defined a new type of adversaries called the covert adversaries. In *covert adversary model* [AL10], the corrupted parties can behave in any possible way like active adversaries, but any party that attempts to cheat is guaranteed to get caught by the honest parties with a minimum fixed probability. That probability is called the *deterrence factor* of covert adversary model. This definition is suitable in many application settings including business, financial, political and diplomatic setting where getting caught cheating results in a loss of reputation, embarrassment or negative press. Security-wise, covert adversary is stronger than passive adversary and weaker than active adversary [AL10]. It is more realistic than passive adversary model. Protocols for covert adversaries are significantly more efficient than protocols for active adversaries [AL10].

Aumann and Lindell [AL10] defined covert adversary model only for static corruption. Adaptive adversary model is more realistic and more secure than static adversaries.

In this article, we define a new adversary model, the covert adaptive adversary model, by generalizing the definition of covert adversary model for the more realistic adaptive corruption. Since the original covert adversary model defined by Aumann and Lindell is only defined for static corruption, we call the covert adversary model of Aumann and Lindell the *covert static adverasry model.*

We prove security relations between the new covert adaptive adversary model with existing adversary models like passive adaptive adversary model, active adaptive adversary model and covert static adversary model. We prove the sequential composition theorem for the new adversary model which is necessary to allow modular design of protocols for this new adversary model.

2 Background

2.1 Preliminary Definitions

For a set R, let $r \xleftarrow{\$} R$ denote that r is obtained by sampling uniformly at random from R. For a probabilistic polynomial-time algorithm A, let $Coins(A)$ denote the distribution of the internal randomness of A. The notation $y = A(x, r)$ means that y has been computed by running A on input x and randomness r. The notation $y \leftarrow A(x)$ means that y should be computed by running A on input x and randomness r where $r \xleftarrow{\$} Coins(A)$. For a binary string x, let $|x|$ denote the length of x.

Definition 1. *The **security parameter** is an additional integer valued parameter used to specify the guaranteed "level of security". All the parties and the adversary receive the security parameter as an input. The security parameter is denoted by s. All complexity characteristics (or the efficiency parameters) are measured in terms of the security parameter.*

An MPC problem is defined by a functionality.

Definition 2 ([Gol09]). *Let n denote the number of parties. An n**-party functionality**, denoted $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ is a random process that maps sequences of the form $\overline{x} = \{x_1, \ldots, x_n\}$ into sequences of random variables $f(\overline{x}) = (f_1(\overline{x}), \ldots, f_n(\overline{x}))$. For each $i \in \{1, \ldots, n\}$, the i-th party, P_i, has input x_i and wishes to obtain the i-th element in $f(x_1, \ldots, x_n)$, denoted $f_i(x_1, \ldots, x_n)$. Functions mapping n inputs to n outputs are a special case of functionality. Functionalities are randomized extensions of functions.*

There are two types of functionalities – non-reactive functionalities and reactive functionalities.

Definition 3. *In a **standard functionality** or **non-reactive functionality**, each party has a single input and a single output. The output of each party is a probabilistic function of the inputs of all the parties. This is also called **secure function evaluation** which is a widely used term.*

Definition 4 ([Gol09]). *In a **reactive functionality**, parties perform some computations for multiple iterations. There exists a global state that is updated in each iteration. The global state may not be known by any individual party. It is shared among the parties. Initially, the global state is an empty state. Each iteration proceeds in the following way.*

1. *Each party receives an input for current iteration.*
2. *Parties compute the outputs for current iteration. The outputs of parties in current iteration depend on the inputs of the parties in current iteration and the global state in current iteration.*
3. *Parties update the global state for the next iteration based on the inputs of the parties in current iteration and the global state in current iteration.*

Canetti [Can00] used the term "probabilistic function" for defining an MPC problem. Goldreich [Gol09] used the term "functionality" to define an MPC problem. The term "functionality" is more general than the term "probabilistic function". The term "probabilistic function" covers only the non-reactive functionalities. On the other hand, functionalities can be either non-reactive or reactive. For this reason, we use the term "functionality" is used to define MPC problems.

Interactive Turing machines are an extension of classical Turing machines in the sense that they allow interaction among the machines. For more details on their definition, see [GMR85] and [Gol06], page 191. In an MPC problem, each party is modeled by an Interactive Turing machine.

Parties use a protocol to solve an MPC problem.

Definition 5 ([Can01]). *A **protocol** is an algorithm written for a distributed system. A protocol describes how the parties communicate among themselves to compute a given functionality. An n-party protocol is represented as a system of n interactive Turing machines where each interactive Turing machine represents the program to be run within a different party. Conventionally, the Turing machine representing the i-th party is called P_i.*

The notion of balanced vectors are used in the security definitions of MPC protocols. A vector $\overline{x} = \{x_1, \ldots, x_n\}$ such that $x_i \in \{0,1\}^*$ for $1 \leq i \leq n$, is called ***balanced***, if, for every i, j such that $1 \leq i, j, \leq n$, $|x_i| = |x_j|$.

The notion of distribution ensembles are used in the security definition of MPC protocols.

Definition 6 ([Can00]). *A **distribution ensemble**, $X = \{X(s,a)\}_{s\in\mathbb{N},a\in\{0,1\}^*}$, is an infinite sequence of probability distributions, where a distribution $X(s,a)$ is associated with each value of $s \in \mathbb{N}$ and $a \in \{0,1\}^*$.*

Distribution ensembles are used to represent the outputs of computation where the parameter s denotes the security parameter and the parameter a represents the input of the computation.

The notion of computational indistinguishability is used in the security definition of MPC protocols. Intuitively, two given distribution ensembles are ***computationally indistinguishable*** if no efficient (polynomial-time) machine can differentiate between them.

Definition 7 ([Can00]). *Let $\delta : \mathbb{N} \to [0,1]$. Two distribution ensembles $X = \{X(s,a)\}_{s\in\mathbb{N},a\in\{0,1\}^*}$ and $Y = \{Y(s,a)\}_{s\in\mathbb{N},a\in\{0,1\}^*}$ have **computational distance at most** δ if, for every algorithm D that is probabilistic polynomial-time in its first input, for all sufficiently large s, all $a \in \{0,1\}^*$, and all auxiliary information $w \in \{0,1\}^*$, the following holds:*

$$|Pr[D(1^s, a, w, x) = 1] - Pr[D(1^s, a, w, y) = 1]| < \delta,$$

where x is chosen from distribution $X(s,a)$, y is chosen from distribution $Y(s,a)$, and the probabilities are taken over the choices of x, y, and the random choices of D.

*Two distribution ensembles $X = \{X(s,a)\}_{s\in\mathbb{N},a\in\{0,1\}^*}$ and $Y = \{Y(s,a)\}_{s\in\mathbb{N},a\in\{0,1\}^*}$ are defined to be **computationally indistinguishable**, denoted $X \stackrel{c}{\equiv} Y$, if X and Y have computational distance at most s^{-c} for all $c > 0$.*

2.2 The Simulation Paradigm Based Security Definition

To define security for MPC problemd, it is necessary to think about what security requirements are desirable in the solution. Some important security requirements are as follows.

- **Correctness:** The outputs of the honest parties follow the same distribution as the outputs of the honest parties defined according to the functionality evaluated at the inputs of all parties.
- **Privacy:** The adversary cannot learn more about the inputs of the honest parties than what it can learn from the outputs of the corrupted parties.

– **Independence of Inputs:** No party can select its input based on the inputs of other parties.

Proving the correctness requirement and the privacy requirement separately is not sufficient for proving a multiparty computation protocol secure. The reason is that these two requirements are dependent on one another ([Can00]). The correctness requirement states that the outputs of the honest parties must be the "correct" values evaluated at the inputs of all parties. The privacy requirement states that the adversary cannot learn more about the inputs of the honest parties than what it can learn from the outputs of the honest parties. The correctness requirement depends on the privacy requirement in the sense that the correctness requirement must ensure that the input values that the corrupted parties contribute to the computation are selected without any knowledge of the inputs of the honest parties. The privacy requirement depends on the correctness requirement since the privacy requirement have to make sure that the adversary learns only the "correct" values of the outputs of the corrupted parties.

There has been several attempts to give security definition for multiparty computation problems. One possible approach for defining security of protocols is to list all the requirements that have to be satisfied by a protocol to be secure ([HL10], Section 1.1). There are two problems in this approach. First, some important requirement may be missed in the process of listing. Different applications need different sets of requirements. It would be better if a general definition of the security of protocols for all cryptographic applications can be established. Second, the security definition should be simple enough so that it is trivial to see that all possible adversarial attacks are prevented by the security definition.

The second approach for defining security of protocols is called the ***simulation paradigm*** based security definition. Goldwasser and Micali [GM84] introduced the notion of simulation paradigm for defining security of encryption schemes. Goldwasser et al. [GMR85] used the simulation paradigm for defining security of zero-knowledge proofs. In his seminal work, Canetti [Can00] proposed the security definition of MPC protocols. His security definition extends the simulation paradigm for the more general and more complex setting of the security definition of MPC protocols. His security definition captures the following security requirements – correctness, privacy and independence of inputs. In his definition, the correctness and privacy requirements are combined together. His definition is a generalized definition, meaning that it can be applied for defining security of protocol for any cryptographic application.

One important property of Canetti's simulation paradigm based security definition is that it satisfies "sequential composition" which allows modular design of protocols (For more details on modular design of protocols, see Section 2.3). Simply stated, if a security definition of protocols satisfy sequential composition, then it is not necessary to consider the details inside a subprotocol that is invoked in another "outer" protocol while considering the security of the outer protocol. When considering the security of the outer protocol, the subprotocol can be though of as a black-box that satisfies the input-output property of the problem solved by the subprotocol. The term "sequential" is used to denote that this property is satisfied if there are more than one subprotocols invoked by the "outer" protocol in such a way that one subprotocol is invoked only after the invocation of the previous subprotocol is finished within the outer protocol. In other words, no two subprotocols are invoked in parallel within the outer protocol.

Canetti's security definition is widely accepted as a standard security definition of MPC protocols.

In a ***simulation paradigm based security definition*** of protocols, two conceptual worlds are defined – the ideal world and the real world.

The ***ideal world*** models the situation where a functionality is evaluated in an ideal scenario, as described below. There exists an incorruptible trusted party. Each party sends its private input to the trusted party. The adversary cannot see the communication between an honest party and the trusted party. The trusted party may select some value uniformly at random for using in the evaluation of the functionality. The trusted party computes the functionality on the inputs received from the parties (and using the randomly generated value if necessary). The trusted party then sends each party its output. Each honest party outputs the message it receives from the trusted party. Each corrupted party outputs a special symbol $\perp$, denoting that it is corrupted. The adversary outputs an arbitrary probabilistic polynomial time function of whatever it sees during the computation. A functionality computed in the ideal world in the above manner is called an ***ideal functionality***.

When defining the ideal world, special consideration have to be given about the behavior of the adversary that cannot be prevented in anyway. In the case of an active adversary, one behavior that cannot be prevented is that the corrupted parties may replace their original inputs to different inputs before sending them to the trusted party.

The ***real world*** defines the situation where the parties execute a protocol to compute the functionality. There is no trusted party. Each honest party outputs whatever is prescribed in the protocol. Each corrupted party outputs a special symbol $\perp$. The adversary outputs an arbitrary probabilistic polynomial time function of whatever it sees during the protocol execution.

To be secure, a protocol have to satisfy that the real world, where the protocol computes the functionality, "emulates" the ideal world where there exists a trusted party for computing the functionality. A protocol is called secure if no probabilistic polynomial time adversary can do more harm in the real world than what it can do in the ideal world. It is formalized by saying that for any probabilistic polynomial time adversary that performs some attack in the real world, there exists an ideal world adversary (or a simulator) that can execute the same attack in the ideal world. Since the ideal world is defined in such a way that no adversarial attack is possible, from the security definition it follows that no adversarial attack is possible in the real world.

2.3 Composition

If a security definition allows ***composition***, then the following holds. Let ρ be a secure MPC protocol computing functionality f. If protocol ρ is used as a sub-protocol in another MPC protocol Π, then, when considering the security of protocol Π, it is sufficient to use the definition of functionality f of protocol ρ, as if, there is a trusted party for computing functionality f in the real world execution of protocol Π.

Definition 8 ([Can00]). *Let m be a fixed positive integer known to all parties. The imaginary world, where a multiparty functionality g is computed and there exists a trusted party computing some other multiparty functionalities $f_1, \ldots, f_m$, is called a **hybrid world with ideal access to** $f_1, \ldots, f_m$, denoted by the term $(f_1, \ldots, f_m)$-hybrid world.*

Two types of composition are described below – the sequential composition and the universal composition.

Sequential Composition. Sequential composition theorem is defined as follows.

Definition 9. *Let m be a fixed positive integer known to all parties. Let $\rho_1, \ldots, \rho_m$ be secure MPC protocols computing functionalities $f_1, \ldots, f_m$, respectively. If a security definition satisfies **sequential composition theorem**, then the following holds. If protocols $\rho_1, \ldots, \rho_m$ are executed inside another MPC protocol Π in such a way that protocol ρ_{i+1} is executed after the execution of protocol ρ_i has finished, then it is sufficient to consider each protocol ρ_j replaced by a trusted party computing function f_j inside protocol Π. It is called sequential composition theorem as the inner protocols $\rho_1, \ldots, \rho_m$ are called inside the outer protocol Π in a sequential manner.*

Universally Composable Security. In ***universally composable security definition***, multiple instances of the same functionality can be executed at the same time without any global coordination ([Can01]). The communication is open, unauthenticated and asynchronous. The functionalities are reactive. The universally composable security is a stronger notion of security than the security notion where only sequential composition theorem is satisfied.

2.4 Security Definition of MPC Protocols Secure Against Covert Static Adversaries

Aumann and Lindell [AL10] presented three formulations of the covert static adversary model – the failed simulation formulation, the explicit-cheat formulation and the strong explicit-cheat formulation. Security-wise, the explicit cheat formulation is strictly stronger than the failed simulation formulation, and strong

explicit cheat formulation is strictly stronger than the explicit-cheat formulation [AL10]. All protocols designed for the covert static adversary model are designed for the most secure formulation, that is, the covert static adversary model in the strong explicit cheat formulation. This is the most secure formulation. In most literature, the term "covert adversary model" is used to refer to the strong explicit cheat formulation of the covert static adversary model. The other two formulations of covert static adversary model are not used. For this reason, we only present the strong explicit cheat formulation of the covert static adversary model of [AL10].

In this section, we present the security definition of MPC protocols secure against covert static adversaries following [AL10].

In static adversary model, the adversary receives an *auxiliary input*. The auxiliary input can be thought of as the advice of the non-uniform Turing machine. It represents the *a priori* information that the adversary possesses at the start of computation. The security definition have to ensure that the adversary cannot gain anything more in the real world than what it gains in the ideal world, even when the adversary possesses this *a priori* information.

Let n denote the number of parties. Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ denote an n-party functionality where $f = (f_1, \ldots, f_n)$. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time adversary. Let $I \subset \{1, \ldots, n\}$ denote the set of corrupted parties. Let $\epsilon : \mathbb{N} \to [0,1]$ be a function.

The Ideal World in Covert Static Adversary Model. The execution in the ideal world with deterrence ϵ proceeds in the following way.

Input: Each party P_i obtains an input x_i. It is assumed that the inputs of all parties have the same length s where s is the security parameter. The adversary $\mathcal{A}$ receives an auxiliary input z.

Sending Inputs to the Trusted Party: Each honest party P_i sends its original input x_i to the trusted party. Each corrupted party P_i may send its original input x_i or some other input with the same length as x_i to the trusted party. The decision of the inputs sent by the corrupted parties is taken by $\mathcal{A}$ and may depend on the inputs of the corrupted parties and the auxiliary input z. Let $\overline{w} = (w_1, \ldots, w_n)$ denote the set of inputs received by the trusted party where w_i denotes the input received by the trusted party from P_i.

Abort Options: If a corrupted party P_i sends $w_i = Abort_i$ to the trusted party, then the trusted party sends $Abort_i$ to the honest parties and halts. There are two special types of input in covert static adversary model – $Corrupted_i$ and $Cheat_i$. These inputs are not available in passive adversary model or active adversary model. If a corrupted party P_i sends $w_i = Corrupted_i$ to the trusted party, then the trusted party sends $Corrupted_i$ to the honest parties and halts. If multiple parties send $Abort_i$ (respectively, $Corrupted_i$), then the trusted party relates only to one of them (say, the one with the smallest i). If both $Corrupted_i$ and $Abort_j$ messages are sent, then the trusted party ignores the $Corrupted_i$ message.

Attempted Cheat Option: If a corrupted party P_i sends $w_i = Cheat_i$ to the trusted party, then the trusted party works in the following way.

1. With probability ϵ, the trusted party sends $Corrupted_i$ to $\mathcal{A}$ and the honest parties, and halts.
2. With probability $(1 - \epsilon)$, the trusted party sends $Undetected$ and the set $\{x_j\}_{j \in \{1,\ldots,n\} \setminus I}$ of honest parties' inputs to $\mathcal{A}$. Then, $\mathcal{A}$ sends the set $\{\widehat{y}_j\}_{j \in \{1,\ldots,n\} \setminus I}$ of output values of its choice for the honest parties to the trusted party. Then, for each $j \in \{1, \ldots, n\} \setminus I$, the trusted party sends $\widehat{y}_j$ to P_j. Then, the trusted party halts.

If multiple parties send $Cheat_i$, then the trusted party relates only to one of them (say, the one with the smallest i).

Trusted Party Answers Adversary: The trusted party may select some value uniformly at random for using in the evaluation of the functionality. The trusted party computes $(f_1(\overline{w}), \ldots, f_n(\overline{w}))$. For each $i \in I$, the trusted party sends $f_i(\overline{w})$ to $\mathcal{A}$.

Trusted Party Answers Honest Parties: $\mathcal{A}$ sends either $Abort_i$ for some $i \in I$ or the message $Continue$ to the trusted party. If the trusted party receives $Abort_i$ from $\mathcal{A}$, then it sends $Abort_i$ to the honest parties and halts. If the trusted party receives the message $Continue$ from $\mathcal{A}$, then it sends $f_i(\overline{w})$ to P_i, for each $i \in \{1, \ldots, n\} \setminus I$.

Output: The notion of the view of the adversary is used in the security definition of MPC protocols.

Definition 10. *The **view of the adversary in the ideal world of covert static adversary model** consists of the inputs of the corrupted parties, the auxiliary input of the adversary and the messages the adversary receives from the trusted party.*

Each honest party outputs what it receives from the trusted party. The corrupted parties output nothing. $\mathcal{A}$ outputs an arbitrary (probabilistic polynomial-time computable) function of its view. It can be assumed that the output of $\mathcal{A}$ consists of its entire view.

To model the outcome of a protocol execution or functionality evaluation, the notion of global output is used.

Definition 11 ([Can00]). *The **global output for static adversaries** is defined as the random variable consisting of the output of the adversary and the outputs of the honest parties. Since the output of the adversary can be assumed to be its entire view, the global output for static adversaries is defined as the random variable consisting of the view of the adversary and the outputs of the honest parties.*

The definition of global output in the ideal world of covert static adversary model is given below.

Definition 12. *Let the global output $IDEALCS^{\epsilon}_{f,\mathcal{A},I}(s,\overline{x},z)$ denote the vector consisting of the outputs of the honest parties and the output of $\mathcal{A}$ after parties $P_1,\ldots,P_n$ evaluating functionality f in the presence of a non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the ideal world of covert static adversary model with deterrence ϵ where P_i has input x_i, $\overline{x}=\{x_1,\ldots,x_n\}$, s is the security parameter and z is the auxiliary input of $\mathcal{A}$.*

Aumann and Lindell [AL10] used the notation $IDEAL^{\epsilon}_{f,\mathcal{A}(z),I}(\overline{x},s)$ for denoting the global output in the ideal world of covert static adversary model. We use the notation $IDEALCS^{\epsilon}_{f,\mathcal{A},I}(s,\overline{x},z)$ to denote the global output in the ideal world of the covert static adversary model so that the notation is consistent with the use of notation $IDEAL_{f,\mathcal{S},\mathcal{Z}}(s,\overline{x},z,\overline{r})$ for the global output in the ideal world of adaptive adversary model in Section 2.5.

The Real World in Covert Static Adversary Model. The real world in covert static adversary model is the same as the real world in active static adversary model, described below. In the real world, each party P_i obtains an input $x_i\in\{0,1\}^s$ and a random input $r_i\in\{0,1\}^*$. The adversary $\mathcal{A}$ receives an auxiliary input z.

Parties execute an n-party protocol Π for evaluating functionality f. There is no trusted party. $\mathcal{A}$ sends all messages on behalf of the corrupted parties and may follow an arbitrary polynomial-time strategy. The honest parties follow the instructions of Π. A corrupted party P_i may abort at any point during the execution of Π. If a corrupted party aborts before the execution of Π is finished, then the honest parties cannot compute their outputs. After the execution of Π is finished, each honest party outputs whatever is specified by Π. The corrupted parties output nothing.

Definition 13. *The **view of the adversary in the real world of covert static adversary model** consists of the inputs and random inputs of the corrupted parties, the auxiliary input, and all the messages sent and received by the corrupted parties during the execution of protocol Π.*

$\mathcal{A}$ outputs an arbitrary (probabilistic polynomial-time computable) function of its view. It can be assumed that the output of $\mathcal{A}$ consists of its entire view.

The definition of global output in the real world of covert static adversary model is given below.

Definition 14. *Let the global output $REAL_{\Pi,\mathcal{A},I}(s,\overline{x},z)$ denote the vector consisting of the outputs of the honest parties and the output of $\mathcal{A}$ after parties $P_1,\ldots,P_n$ executing protocol Π in the presence of a non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real world of covert static adversary model where P_i has input x_i, $\overline{x}=\{x_1,\ldots,x_n\}$, s is the security parameter and z is the auxiliary input of $\mathcal{A}$.*

Aumann and Lindell [AL10] used the notation $REAL_{\Pi,\mathcal{A}(z),I}(\overline{x},s)$ for denoting the global output in the real world of covert static adversary model. We use the notation $REAL_{\Pi,\mathcal{A},I}(s,\overline{x},z)$ for making consistent

with the use of notation $REAL_{f,\mathcal{S},\mathcal{Z}}(s,\overline{x},z,\overline{r})$ for the global output in the ideal world of adaptive adversary model in Section 2.5.

Security in the presence of covert static adversaries is defined as follows.

Definition 15 ([AL10]). *Let f be an n-party functionality and let Π be an n-party protocol. Let $\epsilon : \mathbb{N} \to [0,1]$ be a function. Protocol Π is said to* ***securely compute*** *f* ***in the presence of covert static adversaries with deterrence*** *ϵ if, for any non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real world, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ in the ideal world such that for every $I \subseteq \{1,\ldots,n\}$ and every balanced vector $\overline{x}$, it holds that*

$$\left\{IDEAL^{\epsilon}_{f,\mathcal{S},I}(s,\overline{x},z)\right\}_{s\in\mathbb{N},\langle\overline{x},z\rangle\in(\{0,1\}^*)^{n+1}} \stackrel{c}{\equiv} \left\{REAL_{\Pi,\mathcal{A},I}(s,\overline{x},z)\right\}_{s\in\mathbb{N},\langle\overline{x},z\rangle\in(\{0,1\}^*)^{n+1}}.$$

Aumann and Lindell [AL10] defined the hybrid world for covert static adversary model in the following way.

Definition 16 ([AL10]). *An (f,ϵ)-hybrid world in the covert static adversary model is defined as the hybrid world of the active static adversary model with the following modification: the trusted party works as the trusted party defined in the ideal world of covert static adversary model with deterrence ϵ.*

Aumann and Lindell [AL10] defined the global output in the hybrid world for covert static adversary model in the following way.

Definition 17 ([AL10]). *Let $f_1,\ldots,f_{p(s)}$ be probabilistic polynomial-time functionalities and let Π be an n-party protocol that uses ideal calls to a trusted party computing $f_1,\ldots,f_{p(s)}$. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine and let I be the set of corrupted parties. Then, the $f_1,\ldots,f_{p(s)}$-hybrid execution of Π on input $\overline{x}$, auxiliary input z to $\mathcal{A}$ and security parameter s, denoted $HYBRID^{f_1,\ldots,f_{p(s)}}_{\Pi,\mathcal{A},I}(s,\overline{x},z)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the hybrid execution of Π with a trusted party computing $f_1,\ldots,f_{p(s)}$.*

Aumann and Lindell [AL10] used the notation $HYBRID^{f_1,\ldots,f_{p(s)}}_{\Pi,\mathcal{A}(z),I}(\overline{x},s)$ for denoting the global output in the hybrid world of covert static adversary model. We use the notation $HYBRID^{f_1,\ldots,f_{p(s)}}_{\Pi,\mathcal{A},I}(s,\overline{x},z)$ for making consistent with the use of notation $HYBRID^{f_1,\ldots,f_m}_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z)$ for denoting the global output in the hybrid world of adaptive adversary model in Section 2.5.

Let $\Pi^{\rho_1,\ldots,\rho_{p(s)}}$ denote protocol Π (which is originally designed for the $\left((f_1,\epsilon_1),\ldots,\left(f_{p(s)},\epsilon_{p(s)}\right)\right)$-hybrid world) where each ideal evaluation of functionality f_i is replaced by a subroutine call to protocol ρ_i.

Aumann and Lindell [AL10] proved the sequential composition theorem for their security definition against covert static adversaries. It is presented below.

Theorem 1 ([AL10]). *Let $p(s)$ be a polynomial, let $f_1,\ldots,f_{p(s)}$ be multiparty probabilistic polynomial-time functionalities and let $\rho_1,\ldots,\rho_{p(s)}$ be protocols that securely compute $f_1,\ldots,f_{p(s)}$ in the presence of covert static adversaries with deterrence $\epsilon_1,\ldots,\epsilon_{p(s)}$, respectively. Let g be a multiparty functionality and let Π be a secure protocol for computing g in the $\left((f_1,\epsilon_1),\ldots,\left(f_{p(s)},\epsilon_{p(s)}\right)\right)$-hybrid world (using a single call to each f_i) in the presence of covert static adversaries with deterrence ϵ. Then, $\Pi^{\rho_1,\ldots,\rho_{p(s)}}$ securely computes g in the presence of covert static adversaries with deterrence ϵ.*

Aumann and Lindell [AL10] proved that active static security implies covert static security with any valid value of the deterrence factor.

Proposition 1 ([AL10]). *Let Π be a protocol that securely computes some functionality f with abort in the presence of active static adversaries. Then, Π securely computes f in the presence of covert static adversaries with deterrence ϵ for every ϵ such that $0 \leq \epsilon \leq 1$.*

Definition 18 ([AL10]). *A passive adaptive adversary that is allowed to modify its input before the execution of the protocol begins is called an augmented passive static adversary.*

Aumann and Lindell [AL10] proved that covert static security implies passive static security.

Proposition 2 ([AL10]). *Let Π be a protocol that securely computes some functionality f in the presence of covert static adversaries with deterrence ϵ for $\epsilon(s) \geq 1/poly(s)$ where $poly(s)$ denotes polynomial of s. Then, Π securely computes f in the presence of augmented passive static adversaries.*

2.5 Security Definition of MPC Protocols Secure against Adaptive Adversaries

In this section, the security definition of MPC protocols secure in the presence of adaptive adversaries are presented, following [Can00].

In adaptive adversary model, there is an additional entity called the ***environment***, denoted $\mathcal{Z}$, which is not present in static adversary model. It represents the external world in this model. It is a non-uniform probabilistic polynomial time interactive Turing machine. The environment in adaptive adversary model is a generalization of the auxiliary input in static adversary model. It supplies the auxiliary input to the adversary before the start of computation. It adaptively releases information to the adversary during the computation. It also adaptively obtains information from the adversary during the computation and after the computation.

Let n denote the number of parties. Let $f : (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$ denote an n-party functionality where $f = (f_1, \ldots, f_n)$. Let $\mathcal{S}$ denote the adaptive adversary in the ideal world which is a probabilistic polynomial-time interactive Turing machine. Let $\mathcal{A}$ denote the adaptive adversary in the real world which is a probabilistic polynomial-time interactive Turing machine. Let $\mathcal{Z}$ denote the environment which is a probabilistic polynomial-time interactive Turing machine.

The Ideal World in Adaptive Adversary Model. The execution in the ideal world proceeds in the following way.

Input: Each party P_i has an input $x_i \in \{0,1\}^*$ and the security parameter s. The ideal-world adversary $\mathcal{S}$ has a random input r_0 and the security parameter s. The environment $\mathcal{Z}$ has an input z, a random input r_z and the security parameter s.

First Corruption Stage: $\mathcal{S}$ receives some information from $\mathcal{Z}$. This information corresponds to the auxiliary input of $\mathcal{S}$ in static adversary model. Then, $\mathcal{S}$ proceeds in iterations, where in each iteration $\mathcal{S}$ may decide to corrupt some party, based on its random input r_0 and the information gathered so far. If $\mathcal{S}$ corrupts a party P_i, then $\mathcal{S}$ learns the input x_i of P_i, $\mathcal{Z}$ learns the identity of P_i, and $\mathcal{Z}$ sends some extra auxiliary information to $\mathcal{S}$. This information represents P_i's internal data in other protocols run by P_i.

Computation Stage: Each honest party P_i sends its original input x_i to the trusted party. If $\mathcal{S}$ is passive, then each corrupted party P_i sends its original input x_i to the trusted party. If $\mathcal{S}$ is active, then each corrupted party may replace its input before sending it to the trusted party. In that case, $\mathcal{S}$ decides what inputs the corrupted parties send to the trusted party based on the information gathered so far. Let $\overline{w} = (w_1, \ldots, w_n)$ denote the set of inputs received by the trusted party where w_i denotes the input received by the trusted party from P_i. If $\mathcal{S}$ is passive, then $\overline{w} = \{x_1, \ldots, x_n\}$. The trusted party may select some value uniformly at random for using in the evaluation of the functionality. The trusted party computes $(f_1(\overline{w}), \ldots, f_n(\overline{w}))$. For each $i \in \{1, \ldots, n\}$, the trusted party sends $f_i(\overline{w})$ to P_i.

Second Corruption Stage: $\mathcal{S}$ proceeds in another sequence of iterations where in each iteration $\mathcal{S}$ may decide to corrupt some party, based on r_0 and the information gathered so far. If $\mathcal{S}$ corrupts a party P_i, then $\mathcal{S}$ learns the input and output of P_i, $\mathcal{Z}$ learns the identity of P_i, and $\mathcal{Z}$ sends some extra auxiliary information to $\mathcal{S}$, as before.

Output: Each honest party outputs $f_i(\overline{w})$. Each corrupted party outputs a special symbol $\perp$, specifying that it is corrupted.

Definition 19 ([Can00]). *The **view of the adversary in the ideal world of adaptive adversary model** consists of the random input of the adversary, the auxiliary input it receives from the environment and the inputs and outputs of the corrupted parties.*

$\mathcal{S}$ outputs an arbitrary (probabilistic polynomial-time computable) function of its view. Without loss of generality, it can be assumed that the output of $\mathcal{S}$ consists of its entire view. $\mathcal{Z}$ learns the outputs of all parties and the output of $\mathcal{S}$.

Post-Execution Corruption Stage: $\mathcal{Z}$ and $\mathcal{S}$ interact in rounds until $\mathcal{Z}$ halts with some output. Each round proceeds in the following way.

1. $\mathcal{Z}$ sends a "Corrupt P_i" request for some P_i to $\mathcal{S}$.
2. $\mathcal{S}$ sends $\mathcal{Z}$ some auxiliary information based on its view. For this purpose, $\mathcal{S}$ may corrupt P_i following the process described in the second corruption stage. The information sent by $\mathcal{S}$ to $\mathcal{Z}$ represents P_i's internal data during the evaluation of f in the ideal world.

If $\mathcal{S}$ is t-limited, then at most t parties are corrupted throughout, even if $\mathcal{Z}$ requests to corrupt more parties. If $\mathcal{Z}$ requests to $\mathcal{S}$ to corrupt more parties after already t parties have been corrupted, then $\mathcal{S}$ ignores the request of $\mathcal{Z}$.

Security is defined by comparing global output in the ideal world and the real world. The global output for adaptive adversaries is defined as follows.

Definition 20. *The **global output for adaptive adversaries** is defined as the output of the environment. The environment learns the outputs of all parties. The output of the environment can be assumed to be consisting of the outputs of all parties and the output of the adversary. In both worlds, the corrupted parties output a special symbol $\perp$. In both worlds, the output of the adversary can be assumed to be the view of the adversary. For this reason, the global output for adaptive adversaries can be defined as the random variable consisting of the view of the adversary and the outputs of the honest parties.*

Let the global output $IDEAL_{f,\mathcal{S},\mathcal{Z}}(s,\overline{x},z,\overline{r})$ denote the output of environment $\mathcal{Z}$ on input z, random input r_z, and security parameter s after interacting with parties $P_1,\ldots,P_n$, an ideal world adversary $\mathcal{S}$ and a trusted party computing f in the ideal world of the adaptive adversary model, where P_i has input x_i, $\overline{x}=\{x_1,\ldots,x_n\}$, the trusted party uses random input r_f for evaluating f, and $\overline{r}=\{r_z,r_0,r_f\}$. Let $IDEAL_{f,\mathcal{S},\mathcal{Z}}(s,\overline{x},z)$ denote the distribution of $IDEAL_{f,\mathcal{S},\mathcal{Z}}(s,\overline{x},z,\overline{r})$ when $\overline{r}$ is uniformly distributed in its domain. Let $IDEAL_{f,\mathcal{S},\mathcal{Z}}$ denote the distribution ensemble $\{IDEAL_{f,\mathcal{S},\mathcal{Z}}(s,\overline{x},z)\}_{s\in\mathbb{N},\langle\overline{x},z\rangle\in\{0,1\}^*}$.

The Real World in Adaptive Adversary Model. Each party P_i has an input $x_i\in\{0,1\}^*$, a random input $r_i\in\{0,1\}^*$ and the security parameter s. The real-world adversary $\mathcal{A}$ has a random input r_0 and the security parameter s. The environment $\mathcal{Z}$ has an input z, a random input r_z and the security parameter s.

Parties execute an n-party protocol Π for evaluating functionality f. There is no trusted party. $\mathcal{A}$ sends all messages on behalf of the corrupted parties and may follow an arbitrary polynomial-time strategy. The honest parties follow the instructions of Π.

In the real world, the execution proceeds in the following way. First, $\mathcal{A}$ receives the auxiliary input from $\mathcal{Z}$. The computation proceeds in rounds where each round consists of some ***mini-rounds***. In the start of each mini-round, $\mathcal{A}$ corrupts parties one by one in an adaptive way. If $\mathcal{A}$ is a t-limited adversary, then $\mathcal{A}$ corrupts at most t parties. If $\mathcal{A}$ corrupts a party P_i, then $\mathcal{A}$ learns the input x_i of P_i, the random input r_i of P_i and the entire history of messages sent and received by P_i. $\mathcal{Z}$ learns the identity of the newly corrupted party P_i and $\mathcal{Z}$ sends some extra auxiliary information to $\mathcal{A}$. This information represents P_i's internal data in other protocols run by P_i. From this point on, $\mathcal{A}$ learns all the messages sent to P_i. Then, $\mathcal{A}$ activates one honest party P_i which has not been activated in current round. After being activated, P_i receives all the messages sent to it in the previous round and generates its outgoing messages for current round and the next mini-round begins. $\mathcal{A}$ learns the messages sent by P_i to already corrupted parties. Once all the honest parties are activated, $\mathcal{A}$ generates the messages to be sent by the corrupted parties and the next round begins. If $\mathcal{A}$ is passive, then the corrupted parties follow the instructions of Π. If $\mathcal{A}$ is active, then the corrupted parties follow the instructions of $\mathcal{A}$ and may violate the protocol.

At the end of the protocol execution, each honest party outputs whatever is specified by protocol Π. Each corrupted party outputs $\perp$.

Definition 21 ([Can00]). *The **view of the adversary in the real world of adaptive adversary model** consists of the random input of the adversary, the auxiliary input that it received from the environment, the inputs and random inputs of the corrupted parties, and all the messages sent and received by the corrupted parties during the execution of protocol Π.*

$\mathcal{A}$ outputs an arbitrary (probabilistic polynomial-time computable) function of its view. Without loss of generality, it can be assumed that the output of the adversary consists of its entire view. The environment $\mathcal{Z}$ learns all outputs.

After this, a post-execution corruption stage happens which is similar to the post-execution corruption stage in the ideal world of adaptive adversary model. Without loss of generality, it can be assumed that the output of $\mathcal{Z}$ consists of its entire view of its interaction with $\mathcal{S}$ and the parties.

Let the global output $REAL_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z,\overline{r})$ denote the output of environment $\mathcal{Z}$ on input z, random input r_z, and security parameter s after interacting with adversary $\mathcal{A}$ and parties $P_1,\ldots,P_n$ running protocol Π in the real world of the adaptive adversary model, where P_i has input x_i, $\overline{x}=\{x_1,\ldots,x_n\}$, and $\overline{r}=\{r_z,r_0,r_1,\ldots,r_n\}$. Canetti [Can00] used the notation $EXEC_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z,\overline{r})$ for denoting this. We use the notation $REAL_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z,\overline{r})$ for ease of understanding and making consistent with the use of notation $IDEAL_{f,\mathcal{S},\mathcal{Z}}(s,\overline{x},z,\overline{r})$ for the global output in the ideal world of adaptive adversary model. Let $REAL_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z)$ denote the random variable describing $REAL_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z,\overline{r})$ when $\overline{r}$ is uniformly distributed in its domain. Let $REAL_{\Pi,\mathcal{A},\mathcal{Z}}$ denote the distribution ensemble $\{REAL_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z)\}_{s\in\mathbb{N},\langle\overline{x},z\rangle\in\{0,1\}^*}$.

Security of MPC protocols in the presence of adaptive adversaries is defined as follows.

Definition 22 ([Can00]). *Let f be an n-party functionality and let Π be a protocol for n parties. Protocol Π is said to **securely compute** f **in the presence of** t**-limited adaptive adversaries** if, for any probabilistic polynomial-time t-limited adaptive adversary $\mathcal{A}$ in the real world and any probabilistic polynomial-time environment $\mathcal{Z}$, there exists a probabilistic polynomial-time adaptive adversary $\mathcal{S}$ in the ideal world such that*

$$IDEAL_{f,\mathcal{S},\mathcal{Z}} \stackrel{c}{\equiv} REAL_{\Pi,\mathcal{A},\mathcal{Z}}.$$

Sequential Composition Theorem for Canetti's Security Definition for Adaptive Adversaries. Let the global output $HYBRID^{f_1,\ldots,f_m}_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z)$ denote the output of environment $\mathcal{Z}$ on input z, and security parameter s after interacting with adversary $\mathcal{A}$ and parties $P_1,\ldots,P_n$ running protocol Π in the $(f_1,\ldots,f_m)$-hybrid world of the adaptive adversary model, where P_i has input x_i, $\overline{x}=\{x_1,\ldots,x_n\}$, and all the necessary random inputs are selected uniformly at random from the corresponding domains. Canetti [Can00] used the notation $EXEC^{f_1,\ldots,f_m}_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z)$ for denoting this. We use the notation $HYBRID^{f_1,\ldots,f_m}_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z)$ for ease of understanding and making consistent with the use of notation $IDEAL_{f,\mathcal{S},\mathcal{Z}}(s,\overline{x},z)$ for denoting the global output in the ideal world of adaptive adversary model and the use of notation $REAL_{\Pi,\mathcal{S},\mathcal{Z}}(s,\overline{x},z)$ for denoting the global output in the real world of adaptive adversary model. Let $HYBRID^{f_1,\ldots,f_m}_{\Pi,\mathcal{A},\mathcal{Z}}$ denote the distribution ensemble $\left\{HYBRID^{f_1,\ldots,f_m}_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z)\right\}_{s\in\mathbb{N},\langle\overline{x},z\rangle\in\{0,1\}^*}$.

Let $\Pi^{\rho_1,\ldots,\rho_m}$ denote protocol Π (which is originally designed for the $(f_1,\ldots,f_m)$-hybrid world) where each ideal evaluation of functionality f_i is replaced by a subroutine call to protocol ρ_i.

The following theorem of Canetti [Can00] describes the general statement of modular composition for active adaptive adversaries for the cryptographic security model ([Can00], Theorem 15).

Theorem 2 (Adaptive Computational Modular Composition: General Statement ([Can00])). *Let $t<n,m\in\mathbb{N}$. Let $f_1,\ldots,f_m$ be n-party functionalities. Let Π be an n-party protocol in the $(f_1,\ldots,f_m)$-hybrid world where no more than one ideal evaluation call is made at each round. Let $\rho_1,\ldots,\rho_m$ be n-party protocols such that ρ_i securely computes functionality f_i in the presence of t-limited active adaptive adversaries according to Definition 22. Then, for any probabilistic polynomial-time t-limited active adaptive adversary $\mathcal{A}$ in the real world of adaptive adversary model and any probabilistic polynomial-time environment $\mathcal{Z}$, there exists a probabilistic polynomial-time t-limited active adaptive adversary $\mathcal{S}$ in the $(f_1,\ldots,f_m)$-hybrid*

world of adaptive adversary model such that

$$HYBRID_{\Pi,\mathcal{S},\mathcal{Z}}^{f_1,\ldots,f_m} \stackrel{c}{\equiv} REAL_{\Pi^{\rho_1,\ldots,\rho_m},\mathcal{A},\mathcal{Z}}.$$

As mentioned in [Can00], the above theorem does not assume any security properties from protocol Π. It states that the "input-output functionality" of any protocol Π in the hybrid world is successfully "emulated" by $\Pi^{\rho_1,\ldots,\rho_m}$ in the real world. This more general statement is relevant for both non-reactive and reactive functionalities.

Protocols for securely computing a functionality g in the $(f_1,\ldots,f_m)$-hybrid world in the presence of t-limited active adaptive adversaries is defined in the following way ([Can00], Definition 16).

Definition 23 ([Can00]). *Let $f_1,\ldots,f_m,g$ be n-party functionalities. Let Π be an n-party protocol in the $(f_1,\ldots,f_m)$-hybrid world. It is said that Π* ***securely computes*** *g* ***in the*** *$(f_1,\ldots,f_m)$-**hybrid world in the presence of** t-**limited active adaptive adversaries** if, for any probabilistic polynomial-time t-limited active adaptive adversary $\mathcal{A}$ in the $(f_1,\ldots,f_m)$-hybrid world of adaptive adversary model and any probabilistic polynomial-time environment $\mathcal{Z}$, there exists a probabilistic polynomial-time t-limited active adaptive adversary $\mathcal{S}$ in the ideal world of adaptive adversary model such that*

$$IDEAL_{g,\mathcal{S},\mathcal{Z}} \stackrel{c}{\equiv} HYBRID_{\Pi,\mathcal{A},\mathcal{Z}}^{f_1,\ldots,f_m}.$$

The following corollary of [Can00] describes the modular composition for secure function evaluation for active adaptive adversaries for the cryptographic security model ([Can00], Corollary 17).

Corollary 1 (Adaptive Computational Modular Composition: Secure Function Evaluation ([Can00])). *Let $t < n, m \in \mathbb{N}$. Let $f_1,\ldots,f_m,g$ be n-party functionalities. Let Π be an n-party protocol in the $(f_1,\ldots,f_m)$-hybrid world where no more than one ideal evaluation call is made at each round. Let $\rho_1,\ldots,\rho_m$ be n-party protocols such that ρ_i securely computes functionality f_i in the presence of t-limited active adaptive adversaries according to Definition 22. Then, $\Pi^{\rho_1,\ldots,\rho_m}$ securely computes functionality g in the presence of t-limited active adaptive adversaries.*

3 New Covert Adaptive Adversary Model

In this section we present the new covert adaptive adversary model. We present the security definitions of MPC protocols secure in the presence of the new covert adaptive adversary model.

The Ideal World in Covert Adaptive Adversary Model

Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, where $f = (f_1,\ldots,f_n)$. There exists an incorruptible trusted party in the ideal world. Let $P_1,\ldots,P_n$ denote the parties. Party P_i has input $x_i \in \{0,1\}^*$ – no random input is required. Let $\overline{x} = \{x_1,\ldots,x_n\}$. Parties want to evaluate $f(\overline{x}) = \{f_1(\overline{x}),\ldots,f_n(\overline{x})\}$. Let $\mathcal{A}$ denote an adaptive ideal world adversary that is a non-uniform interactive probabilistic polynomial time Turing machine with random input r_0 and security parameter s. Let $\mathcal{Z}$ denote the environment that is a non-uniform interactive probabilistic polynomial time Turing machine with input z, random input r_z and security parameter s. Let $\epsilon : \mathbb{N} \to [0,1]$ be a function.
The execution in the ideal world with ϵ proceeds as follows.

Inputs:

Each party obtains an input. Let x_i denote the input of P_i.

First Corruption Stage:

The adversary $\mathcal{A}$ receives auxiliary information from $\mathcal{Z}$. Then $\mathcal{A}$ proceeds in iterations. In each iteration, $\mathcal{A}$ may decide to corrupt some party, depending on $\mathcal{A}$'s random input and the information collected so far. When a party is corrupted, the adversary $\mathcal{A}$ gets its input, the environment $\mathcal{Z}$ learns the identity of the corrupted party and sends some extra auxiliary information to $\mathcal{A}$. This information represents the internal history of the newly corrupted party in other protocol executions. Let I_1 denote the set of corrupted parties at the end of this stage.

Computation Stage:

Send inputs to the Trusted Party:

Each honest party P_j sends its received input x_j to the trusted party. The corrupted parties, controlled by $\mathcal{A}$, may either send their received input, or send some other input of the same length to the trusted party. This decision is made by $\mathcal{A}$ and may depend on the information gathered so far. Let $\overline{w} = \{w_1, \ldots, w_n\}$ denote the vector of inputs sent to the trusted party.

Abort Options:

If a corrupted party sends $w_i = abort_i$ to the trusted party as its input, then the trusted party sends $abort_i$ to all of the honest parties and halts. If a corrupted party sends $w_i = corrupted_i$ to the trusted party as its input, then the trusted party sends $corrupted_i$ to all of the honest parties and halts. If multiple parties send $abort_i$ (respectively, $corrupted_i$), then the trusted party relates only to one of them (say, the one with the smallest i). If both $corrupted_i$ and $abort_j$ messages are sent, then the trusted party ignores the $corrupted_i$ message.

Attempted Cheat Option:

If a corrupted party sends $w_i = cheat_i$ to the trusted party as its input, then the trusted party works as follows:

1. With probability ϵ, the trusted party sends $corrupted_i$ to the adversary and all of the honest parties.
2. With probability $(1 - \epsilon)$, the trusted party sends *undetected* to the adversary along with the honest parties' inputs $\{x_j\}_{j \in [n] \setminus I_1}$. Following this, the adversary sends the trusted party output values $\{y_j\}_{j \in [n] \setminus I_1}$ of its choice for the honest parties. Then, for every $j \in [n] \setminus I_1$, the trusted party sends y_j to P_j.

The ideal execution then ends here.

If no w_i equals $abort_i, corrupted_i$ or $cheat_i$, then the ideal execution continues below.

Trusted Party Answers Adversary:

The trusted party computes $(f_1(\overline{w}), \ldots, f_n(\overline{w}))$ and sends $f_i(\overline{w})$ to $\mathcal{A}$, for all $i \in I_1$.

Second Corruption Stage:

After learning the outputs of the corrupted parties, $\mathcal{A}$ proceeds in another sequence of iterations. In each iteration, $\mathcal{A}$ may decide to corrupt some party, where the decision of $\mathcal{A}$ depends on the information obtained so far. When a party is corrupted, the adversary $\mathcal{A}$ gets its input and output, the environment $\mathcal{Z}$ learns the identity of the corrupted party and sends some extra auxiliary information to $\mathcal{A}$. This information represents the internal history of the newly corrupted party in other protocol executions. Let I_2 denote the set of corrupted parties at the end of this stage.

Trusted Party Answers Honest Parties:

After the second corruption stage, the adversary sends either $abort_i$ for some $i \in I_2$, or *continue* to the trusted party.

If the trusted party receives $abort_i$ for some $i \in I_2$, then it sends $abort_i$ to all honest parties and halts.

If the trusted party receives *continue*, then it sends $f_j(\overline{w})$ to party P_j, for each $j \in [n] \setminus I_2$.

Outputs:

An honest party always outputs the message it obtained from the trusted party. The corrupted parties output nothing. The adversary $\mathcal{A}$ outputs any arbitrary (probabilistic polynomial-time computable) function of the information gathered during the computation in the ideal world. The environment $\mathcal{Z}$ learns all outputs.

Post-Execution Corruption Stage:

The environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ interacts in rounds. In each round, $\mathcal{Z}$ generates a "Corrupt P_i" request for some P_i to $\mathcal{A}$. After receiving this request, $\mathcal{A}$ sends $\mathcal{Z}$ some arbitrary information. This information represents the internal history of P_i pertaining to the evaluation of f. For this purpose, $\mathcal{A}$ may corrupt more parties as described in the second corruption stage. The interaction continues until $\mathcal{Z}$ halts, with some output. Without loss of generality, the output of $\mathcal{Z}$ can be defined as the entire view of $\mathcal{Z}$ during its interaction with $\mathcal{A}$ and the parties. Let I_3 denote the set of corrupted parties at the end of this stage. The global output is defined to be the output of $\mathcal{Z}$. The output of $\mathcal{Z}$ may include the outputs of all parties and the output of the adversary $\mathcal{A}$.

Definition 24. *Let $IDEALCA^{\epsilon}_{f,\mathcal{A},\mathcal{Z}}(s,\overline{x},z,\overline{r})$ denote the output of the environment $\mathcal{Z}$ after parties $P_1,\ldots,P_n$ performing an evaluation of f with deterrence ϵ in the ideal world of covert adaptive adversary model in the presence of adversary $\mathcal{A}$ where party P_i has input x_i, the adversary $\mathcal{A}$ has random input r_0, the environment $\mathcal{Z}$ has input z and random input r_z, the input vector is $\overline{x}=\{x_1,\ldots,x_n\}$, the vector of random inputs is $\overline{r}=\{r_z,r_0\}$, and s is the security parameter. Let $IDEALCA^{\epsilon}_{f,\mathcal{A},\mathcal{Z}}(s,\overline{x},z)$ denote the random variable describing the distribution of $IDEALCA^{\epsilon}_{f,\mathcal{A},\mathcal{Z}}(s,\overline{x},z,\overline{r})$ where $\overline{r}$ is selected uniformly at random from its domain.*

The Real World in Covert Adaptive Adversary Model

The real world in the covert adaptive adversary model is the same as the real world in the active adaptive adversary model. For completeness, we describe the real world here.

Let $f:(\{0,1\}^*)^n\rightarrow(\{0,1\}^*)^n$ be an n-party functionality, where $f=(f_1,\ldots,f_n)$. Let Π be an n-party protocol that evaluates f. There is no trusted party. The adversary $\mathcal{A}$ sends all messages in place of the corrupted parties and may follow an arbitrary polynomial-time strategy. The honest parties follow the instructions of Π. Each party P_i has input $x_i\in\{0,1\}^*$, random input $r_i\in\{0,1\}^*$ and the security parameter s. Let $\overline{x}=\{x_1,\ldots,x_n\}$. Let $\mathcal{A}$ denote an adaptive real world adversary that is a non-uniform interactive probabilistic polynomial time Turing machine with random input r_0 and security parameter s. Let $\mathcal{Z}$ denote the environment that is a non-uniform interactive probabilistic polynomial time Turing machine with input z, random input r_z and security parameter s.

At first the adversary $\mathcal{A}$ receives some auxiliary information from the environment $\mathcal{Z}$. The computation proceeds in rounds. Each round proceeds in a series of mini-rounds. At the start of each mini-round, the adversary $\mathcal{A}$ may corrupt parties one by one in an adaptive way, depending on the information gathered so far. Then $\mathcal{A}$ selects an honest party P_i that has not been activated in this round, and activates it. When activated, P_i receives the messages sent to it in the previous round and generates its messages for this round – then the mini-round ends. $\mathcal{A}$ learns all messages sent by P_i to the corrupted parties. When all the honest parties have been activated, $\mathcal{A}$ generates the messages to be sent by the corrupted parties that were not activated in this round, and the next round begins.

When a party is corrupted, $\mathcal{A}$ learns the party's input, random input, and the entire history of the messages sent and received by the party. $\mathcal{Z}$ learns the identity of the corrupted party and sends some additional auxiliary information to $\mathcal{A}$. This information represents the party's internal data from other protocol executions. From this point on $\mathcal{A}$ learns all the messages received by the party and the party behaves according to the instruction of $\mathcal{A}$.

At the end of the protocol execution, the honest parties output whatever is specified by the protocol. The corrupted parties output nothing. The adversary $\mathcal{A}$ outputs some arbitrary (probabilistic polynomial time computable) function computed from its view. The environment $\mathcal{Z}$ learns all outputs.

Then a "Post-Execution Corruption Stage" begins. In this stage, $\mathcal{Z}$ and $\mathcal{A}$ interacts in rounds. In each round, $\mathcal{Z}$ generates a "Corrupt P_i" request to $\mathcal{A}$. Then $\mathcal{A}$ sends $\mathcal{Z}$ some arbitrary information. This information represents the internal data of P_i during the execution of protocol Π. The interaction continues until $\mathcal{Z}$ halts, with some output. The output of $\mathcal{Z}$ is defined to be its entire view during its interaction with $\mathcal{A}$. The global output is defined to be the output of $\mathcal{Z}$.

Below we describe the definition of the execution in the real world of the active adaptive adversary model which we obtain by plugging in the active adversary model in the generic definition of the execution in the real world of the adaptive adversary model.

Definition 25. *Let $REAL_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z,\overline{r})$ denote the output of the environment $\mathcal{Z}$ after parties $P_1,\ldots,P_n$ running protocol Π in the real world of active adaptive adversary model in the presence of adversary $\mathcal{A}$ where party P_i has input x_i and random input r_i, the adversary $\mathcal{A}$ has random input r_0, the environment $\mathcal{Z}$ has input z and random input r_z, the input vector is $\overline{x}=\{x_1,\ldots,x_n\}$, the vector of random inputs is $\overline{r}=\{r_z,r_0,r_1,\ldots,r_n\}$, and s is the security parameter. Let $REAL_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z)$ denote the random variable describing the distribution of $REAL_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z,\overline{r})$ where $\overline{r}$ is selected uniformly at random from its domain.*

Security of MPC protocols in the presence of covert adaptive adversaries with deterrence ϵ is defined as follows.

Definition 26. ***(Security in the presence of Covert Adaptive Adversaries***

A protocol Π is said to securely compute f in the presence of covert adaptive adversaries with deterrence ϵ if for any non-uniform probabilistic polynomial-time adaptive adversary $\mathcal{A}$ for the real world and any environment $\mathcal{Z}$, there exists a non-uniform probabilistic polynomial-time adaptive adversary $\mathcal{S}$ for the ideal world such that the following holds for every balanced vector $\overline{x}$

$$\left\{IDEALCA^{\epsilon}_{f,\mathcal{S},\mathcal{Z}}(s,\overline{x},z)\right\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}} \stackrel{c}{\equiv} \left\{REAL_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z)\right\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}}.$$

4 Security Relationship of the New Covert Adaptive Adversary Model with Existing Adversary Models

In this section, we prove security relations of the new covert adaptive adversary model with active adaptive adversary model, passive adaptive adversary model and covert static adversary model.

We prove that active adaptive security implies covert adptive security.

Proposition 3. *Let Π be a protocol that securely computes some functionality f with abort in the presence of active adaptive adversaries. Then, Π securely computes f in the presence of covert adaptive adversaries with deterrence ϵ for every ϵ such that $0 \leq \epsilon \leq 1$.*

Proof. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time adaptive adversary for the real world and let $\mathcal{Z}$ be the environment. Since Π securely computes f with abort in the presence of active adaptive adversaries, by definition, there exists a simulator $\S_{AA}$ such that the following holds for any balanced vector $\overline{x}$

$$\left\{IDEAL_{f,\S_{AA},\mathcal{Z}}(s,\overline{x},z)\right\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}} \stackrel{c}{\equiv} \left\{REAL_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z)\right\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}}.$$

The only difference between an active adaptive adversary and a covert adaptive adversary is that a covert adaptive adversary may send $corrupted_i$ or $cheat_i$ in the input stage for some $i \in I_1$ while an active adaptive adversary never sends such an input.

For adversary $\mathcal{A}$ and environment $\mathcal{Z}$, we construct a simulator $\S_{CA}$ for active adaptive security definition (Definition 22), from the simulator $\S_{AA}$ as follows.

- $\S_{CA}$ starts with its input.
- If $\S_{CA}$ receives $corrupted_i$ for some $i \in I_1$ from $\mathcal{A}$, then $\S_{CA}$ sends $corrupted_i$ to the trusted party and $\mathcal{A}$. Then $\S_{CA}$ halts.
- If $\S_{CA}$ receives $cheat_i$ for some $i \in I_1$ from $\mathcal{A}$, then $\S_{CA}$ sends $cheat_i$ to the trusted party.
- If the trusted party replies with $corrupted_i$, then $\S_{CA}$ sends $corrupted_i$ to $\mathcal{A}$ and halts.
- If the trusted party replies with *undetected* and the set of inputs of the honest parties, then $\S_{CA}$ sends them to $\mathcal{A}$. Then $\S_{CA}$ receives the set of outputs of the adversary's choice for the honest parties from $\mathcal{A}$. $\S_{CA}$ sends this set to the trusted party and halts.
- If no input equals $corrupted_i$ or $cheat_i$, then $\S_{CA}$ stops this execution and invokes $\S_{AA}$ from the start. $\S_{CA}$ follows whatever $\S_{AA}$ does and at the end, outputs whatever $\S_{AA}$ outputs.
- In the case where there is no $corrupted_i$ or $cheat_i$ input, security in the covert adaptive model follows from the security in the active adaptive model. For the remaining cases, $\S_{CA}$ simulates the ideal world according to Definition 22.

Definition 27. *A passive adaptive adversary that is allowed to modify its input before the execution of the protocol begins is called an augmented passive adaptive adversary.*

We prove that covert adaptive security implies passive adptive security.

Proposition 4. *Let Π be a protocol that securely computes some functionality f in the presence of covert adaptive adversaries with deterrence ϵ and for $\epsilon(s) \geq 1/poly(s)$. Then, Π securely computes f in the presence of augmented passive adaptive adversaries.*

Proof. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time adaptive adversary for the real world and let $\mathcal{Z}$ be the environment. Since Π securely computes f with abort in the presence of covert adaptive adversaries with deterrence ϵ, by definition, there exists a simulator $\S_{CA}$ such that the following holds for any balanced vector $\overline{x}$

$$\left\{IDEALCA^{\epsilon}_{f,\S_{CA},\mathcal{Z}}(s,\overline{x},z)\right\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}} \stackrel{c}{\equiv} \left\{REAL_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z)\right\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}}.$$

The difference between a passive adaptive adversary and a covert adaptive adversary are as follows.

1. A covert adaptive adversary may replace the inputs of the corrupted parties. A passive adaptive adversary never replace the inputs of the corrupted parties.
2. A covert adaptive adversary may send $corrupted_i$ or $cheat_i$ in the input stage for some corrupted party $P_i, i \in I_1$. A passive adaptive adversary never sends such an input.

That means, the simulator $\S_{CA}$ for covert adaptive adversary model can be used as the simulator for the passive adaptive adversary model in the passive adaptive security definition (Definition 22). In the ideal world of the passive adaptive adversary model, the adversary never sends the input $corrupted_i$ or $cheat_i$ for any corrupted party P_i, so the events supposed to happen if those inputs are sent for some corrupted party never occur in the ideal world of the passive adaptive adversary model. Then, the indistinguishability of global outputs in the two worlds of the passive adaptive adversary model is maintained followed by the indistinguishability of global outputs in the two worlds of the covert adaptive adversary model. Then, Π securely computes f in the presence of passive adaptive adversaries.

We first define a transformation.

Definition 28. *Let Π be a protocol that securely computes some functionality f in the presence of covert adaptive adversaries with deterrence ϵ. Let $ToActive()$ denote the transformation of Π to a protocol Π' such that if an honest party is supposed to output $corrupted_i$ in Π, then it outputs $abort_i$ in Π'.*

Proposition 5. *Let Π be a protocol and $\mu(s)$ be a negligible function of s. Let $\Pi' = ToActive(\Pi)$. Then Π securely computes some functionality f in the presence of covert adaptive adversaries with deterrence $\epsilon(s) = 1-\mu(s)$ if and only if Π' securely computes f with abort in the presence of active adaptive adversaries.*

Proof. Let Π be a protocol that securely computes some functionality f in the presence of covert adaptive adversaries with deterrence $\epsilon(s) = 1 - \mu(s)$. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time adaptive adversary for the real world and let $\mathcal{Z}$ be the environment.

If $\mathcal{A}$ does not send $cheat_i$, then the ideal world and the real world execution are the same. If $\mathcal{A}$ sends $cheat_i$, then the corrupted party gets caught with probability negligibly close to 1, so Π' outputs $abort_i$ with probability negligibly close to 1.

For active or passive adversary models, Canetti [Can00] proved that adaptive security implies static or nonadaptive security. Using the same idea as Canetti's proof, we prove that covert adaptive security implies covert static security.

Proposition 6. *Let Π be a protocol that securely computes functionality f in the presence of covert adaptive adversaries with deterrence ϵ. Then, Π securely computes f in the presence of covert static adversaries with deterrence ϵ. Furthermore, assuming the presence of homomorphic public key encryption schemes, there exists protocols that are secure in the presence of covert static adversaries, but not secure in the presence of covert adaptive adversaries.*

Proof. Let Π be a protocol that securely computes functionality f in the presence of covert adaptive adversaries with deterrence ϵ. Let $\overline{x}$ be a balanced vector of inputs for f.

Let $\mathcal{A}_{CS}$ be a static probabilistic polynomial time real world adversary. We construct a probabilistic polynomial time simulator or ideal world adversary $\mathcal{S}_{CS}$ for adversary $\mathcal{A}_{CS}$ such that the requirement of security definition of covert static adversary model with deterrence ϵ is satisfied.

Let $\mathcal{A}_{CA}$ be the adaptive probabilistic polynomial time real world adversary, defined as follows. $\mathcal{A}_{CA}$ receives a value $z = (C, \delta)$ from its environment where C is the set of corrupted parties. $\mathcal{A}_{CA}$ corrupts the parties in C and invokes $\mathcal{A}_{CS}$ with auxiliary input δ and setting the set of corrupted parties to C. Then

$$\{REAL_{\Pi,\mathcal{A}_{CA},\mathcal{Z}}(s,\overline{x},z)\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}} = \{REAL_{\Pi,\mathcal{A}_{CS}}(s,\overline{x},z)\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}}\,.$$

Define $\mathcal{Z}$ to be the environment that, on input z, sends z to the adversary at the start of the execution and remains inactive from that time. Since Π securely computes functionality f in the presence of covert adaptive adversaries with deterrence ϵ, it holds that there exists a simulator $\mathcal{S}_{CA}$ such that the following holds

$$\left\{IDEALCA^{\epsilon}_{f,\mathcal{S}_{CA},\mathcal{Z}}(s,\overline{x},z)\right\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}} \stackrel{c}{\equiv} \{REAL_{\Pi,\mathcal{A}_{CA},\mathcal{Z}}(s,\overline{x},z)\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}}\,.$$

Then $\mathcal{S}_{CA}$ eventually corrupts the parties in set C.

The simulator $\mathcal{S}_{CS}$ works as follows. Given a set C of corrupted parties and auxiliary input δ, $\mathcal{S}_{CS}$ invokes $\mathcal{S}_{CA}$. $\mathcal{S}_{CS}$ acts as the environment for $\mathcal{S}_{CA}$ and sends $z = (C, \delta)$ to $\mathcal{S}_{CA}$. Whenever $\mathcal{S}_{CA}$ corrupts a party in C, $\mathcal{S}_{CS}$ sends $\mathcal{S}_{CA}$ the input of that party for functionality f. $\mathcal{S}_{CS}$ outputs whatever $\mathcal{S}_{CA}$ outputs. Then

$$\left\{IDEALCA^{\epsilon}_{f,\mathcal{S}_{CA},\mathcal{Z}}(s,\overline{x},z)\right\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}} = \left\{IDEALCS^{\epsilon}_{f,\mathcal{S}_{CS}}(s,\overline{x},z)\right\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}},$$

implying that

$$\begin{aligned}\left\{IDEALCS^{\epsilon}_{f,\mathcal{S}_{CS}}(s,\overline{x},z)\right\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}} &\stackrel{c}{\equiv} \{REAL_{\Pi,\mathcal{A}_{CA},\mathcal{Z}}(s,\overline{x},z)\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}}\\ &= \{REAL_{\Pi,\mathcal{A}_{CS}}(s,\overline{x},z)\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}}\end{aligned}$$

Therefore Π securely computes f in the presence of covert static adversaries with deterrence ϵ.

Next we prove the furthermore part. Aumann and Lindell [AL10] designed a protocol Π_{OT} for oblivious transfer secure with deterrence $\frac{1}{2}$ in the presence of covert static adversaries. Protocol Π_{OT} is secure in the presence of covert static adversaries assuming that homomorphic public key encryption scheme exists. Here we do not describe the protocol and its security proof due to space. The full protocol Π_{OT} and its security proof are available in [AL10].

We describe why protocol Π_{OT} is not secure in the presence of covert adaptive adversaries. In case of static adversaries, the adversary corrupts the parties before the execution of the protocol starts and this set remains fixed throughout the execution of protocol. In case of adaptive adversaries, the adversary can corrupt the parties before, during and after the execution of the protocol. Here we are considering adaptive adversaries without erasure. That means the adversary learns the full history of a party after corrupting it.

There are 8 steps in protocol Π_{OT}. Let $\mathcal{SIM}_S$ denote the simulator described in the security proof (for covert static adversaries) by Aumann and Lindell for the case where the sender S is corrupted. In order to work in the presence of covert adaptive adversaries, the simulators designed for the case of covert static adversaries need to be modified so that the new simulator works when the adversary can corrupt a party even after the protocol started and after the protocol finishes its execution. We try to modify the simulator $\mathcal{SIM}_S$ for the covert adaptive adversary model.

Consider the adaptive probabilistic polynomial time adversary $\mathcal{A}$ that corrupts the sender S before the execution of Π_{OT} starts and corrupts the receiver R after step 3 of Π_{OT}.

In step 3 of protocol Π_{OT}, the receiver R performs the following actions.

1. R chooses two random bits α, β.

2. R computes

$$c_0^1 = E_{pk_1}(\alpha),\ c_0^2 = E_{pk_1}(1-\alpha),$$
$$c_1^1 = E_{pk_2}(\beta),\ c_1^2 = E_{pk_2}(1-\beta)$$

using random coins r_0^1, r_0^2, r_1^1 and r_1^2, respectively.
3. R sends (c_0^1, c_0^2) and (c_1^1, c_1^2) to S.

Let a *valid pair of ciphertexts* be a pair of ciphertext such that one of them encrypts zero and the other one encrypts one. In step 3(b), R sends two vaild pair of ciphertexts, but the order of these ciphertexts in each pair is selected at random by R.

Up to the end of step 3, the adversary $\mathcal{A}$ corrupted only S. So the simulator $\mathcal{SIM}_S$ has to produce the message sent by honest R in the ideal world. $\mathcal{SIM}_S$ simulates step 3 as follows.

1. $\mathcal{SIM}_S$ chooses two random bits b, α.
2. $\mathcal{SIM}_S$ computes

$$c_b^1 = E_{pk_1}(\alpha),\ c_b^2 = E_{pk_1}(1-\alpha),$$
$$c_{1-b}^1 = E_{pk_2}(1),\ \ c_{1-b}^2 = E_{pk_2}(1).$$

3. $\mathcal{SIM}_S$ sends (c_0^1, c_0^2) and (c_1^1, c_1^2) to $\mathcal{A}$.

In step 3(b), $\mathcal{SIM}_S$ prepares one valid pair of ciphertexts and one invalid pair of ciphertexts. For the invalid pair of ciphertexts, both ciphertexts encrypt one. $\mathcal{SIM}_S$ works in this way so that in a later step (step 6), $\mathcal{SIM}_S$ can learn the input (x_0, x_1) of the corrupted sender S and send (x_0, x_1) to the trusted party. This part is necessary for the security of Π_{OT} against covert static adversaries.

When the adaptive adversary $\mathcal{A}$ corrupts the honest receiver R after step 3, $\mathcal{A}$ gets to know the full history of R. $\mathcal{A}$ is supposed to learn the pair of public key (pk_2, sk_2) since this key pair was generated by the receiver R is step 1. From this secret key sk_2, $\mathcal{A}$ can decrypt the ciphertexts c_{1-b}^1 and c_{1-b}^2 provided by R in step 3. Then $\mathcal{A}$ learns that the pair (c_{1-b}^1, c_{1-b}^2) is an invalid pair ciphertexts as both of these ciphertexts encrypt one. In the real world, the honest receiver R supplies two valid pair of ciphertexts in step 3. In the ideal world, the simulator provides one valid pair and one invalid pair of ciphertexts in step 3. Then the adaptive adversary $\mathcal{A}$ can distinguish the ideal world from the real world in polynomial time.

There is no other way to modify $\mathcal{SIM}_S$ so that the modified simulator works in the presence of a covert adaptive adversary.

Therefore, protocol Π_{OT} is not secure against covert adaptive adversaries.

5 Sequential Composition Theorem for Covert Adaptive Adversaries

We prove sequential composition theorem for covert adaptive adversary model. We prove this theorem in a similar way following the method by which Aumann and Lindell [AL10] proved sequential composition theorem for covert static adversaries.

The Hybrid World.
We first define the hybrid world where the parties run an n-party protocol Π that contains ideal evaluations of some n-party functionalities $f_1, \ldots, f_{p(s)}$. To evaluate these ideal functionalities, parties have access to a trusted party. The trusted party is called in special rounds, determined by protocol Π. In each such round, a functionality f_i (out of $f_1, \ldots, f_{p(s)}$) is specified. The trusted party computing f_i acts as the trusted party in the ideal world of the covert adaptive adversary model with deterrence ϵ_i. First the adversary adaptively corrupts parties, and learns the internal data of corrupted parties. For each newly corrupted party, the adversary receives information from the environment. Then each honest

party P_j sends its input for f_i to the trusted party. The input of an honest party P_j for functionality f_i depends on protocol Π. The adversary sends the inputs of the corrupted parties for f_i to the trusted party. The honest parties all send their inputs for f_i to the trusted party in the same round. The honest parties do not communicate among themselves until they receive their output of f_i from the trusted party. The reason for this is that we are considering *sequential composition* where the ideal evaluation of a functionality f_i is fully finished before the start of the next ideal evaluation of functionality f_{i+1}. The trusted party computes functionality f_i on the received inputs and then sends the respective outputs of f_i to each party. Then the adversary can again adaptively corrupt more parties. After receiving the outputs for f_i from the trusted party, the parties resume the execution of protocol Π. Such a hybrid world is called the $(f_1, \epsilon_1), \ldots, (f_{p(s)}, \epsilon_{p(s)})$-hybrid world.

Definition 29. *Let* $HYBRID_{\Pi,\mathcal{A},\mathcal{Z}}^{(f_1,\epsilon_1),\ldots,(f_{p(s)},\epsilon_{p(s)})}(s, \overline{x}, z, \overline{r})$ *denote the output of the environment* $\mathcal{Z}$ *after parties* $P_1, \ldots, P_n$ *performing an execution of protocol* Π *in the* $(f_1, \epsilon_1), \ldots, (f_{p(s)}, \epsilon_{p(s)})$*-hybrid world in the presence of adversary* $\mathcal{A}$ *where party* P_i *has input* x_i *and random input* r_i*, the adversary* $\mathcal{A}$ *has random input* r_0*, the environment* $\mathcal{Z}$ *has input* z *and random input* r_z*, the input vector is* $\overline{x} = \{x_1, \ldots, x_n\}$*, the vector of random inputs is* $\overline{r} = \{r_z, r_0, r_1, \ldots, r_n\}$*, and* s *is the security parameter. Let* $HYBRID_{\Pi,\mathcal{A},\mathcal{Z}}^{(f_1,\epsilon_1),\ldots,(f_{p(s)},\epsilon_{p(s)})}(s, \overline{x}, z, \overline{r})$ *denote the random variable describing the distribution of* $HYBRID_{\Pi,\mathcal{A},\mathcal{Z}}^{(f_1,\epsilon_1),\ldots,(f_{p(s)},\epsilon_{p(s)})}(s, \overline{x}, z, \overline{r})$ *where* $\overline{r}$ *is selected uniformly at random from its domain.*

Replacing an ideal evaluation with a subroutine call.
Now we describe how to replace an ideal evaluation of a functionality f_i with a protocol ρ_i within the execution of protocol Π. Let ℓ_i denote the round at which protocol ρ_i is invoked within protocol Π. At the start of round ℓ_i, each party P_j saves its internal state relevant to protocol Π in a special tape. Let $\sigma_{j,i}$ denote this state. The call to the trusted party for evaluation of f_i is replaced by the execution of protocol ρ_i. Let $x_{j,i}$ denote the input of P_j for functionality f_i according to protocol Π. P_j sets its input for execution of ρ_i to $x_{j,i}$ and sets its random input for ρ_i to a uniform random element of the appropriate domain. No honest party resumes execution of protocol Π before the execution of ρ_i is finished. All honest parties finish the execution of ρ_i at the same round. When P_j completes execution of protocol ρ_i with output $y_{j,i}$, P_i resumes the execution of Π starting from state $\sigma_{j,i}$ as if $y_{j,i}$ is the value that P_j received as its output for f_i from the trusted party. If P_j gets $corrupted_k$ as its output from the execution of ρ_i, then P_j acts as per the instruction of protocol Π. Let $\Pi^{\rho_1,\ldots,\rho_{p(s)}}$ denote protocol Π (which is initially designed for the $(f_1, \ldots, f_{p(s)}$-hybrid world) where each ideal evaluation call to f_i is replaced by a subroutine call to protocol ρ_i.

We first describe some terms following [Can00].

An *execution* of a functionality f in the ideal world is the process of evaluating f in the ideal world with a given adversary and a given environment on given inputs for the parties, the adversary and the environment, and given random inputs for the adversary and the environment.

An *execution* of a protocol Π in the real world is the process of running Π with a given adversary and a given environment on given inputs and random inputs for the parties, the adversary and the environment.

The *internal state* of an honest party P_i at a given round ℓ consists of the following elements:

1. The contents of all tapes of P_i and the head position and the control state at the end of round ℓ.
2. The message received by P_i at round ℓ.
3. The entire random input of P_i (which includes the part of the random input which has not yet been used).

The *internal history* of an honest party P_i at a given round ℓ is the concatenation of the internal states of of P_i from the beginning of the execution up to round ℓ.

The internal state of the adversary is defined similar to the internal state of a party.

The *global state* at a given round ℓ is defined to be the concatenation of the internal histories of the honest parties, the internal state of the adversary and the state of the environment at round ℓ. This definition ensures that the global state at a given round uniquely determines the rest of the execution of the protocol. The global state is extended to rounds after the execution of the protocol completes, up to the time when the environment halts.

Let $GS_{f,\mathcal{A},Z}(\ell, s, \overline{x}, z, \overline{r})$ denote the global state at round ℓ of an execution of functionality f in the ideal world with adversary $\mathcal{A}$, environment $\mathcal{Z}$, security parameter s, input vector $\overline{x}$ for the parties, input z for the environment, and vector of random inputs $\overline{r}$. Let $GS_{\Pi,\mathcal{A},Z}(\ell, s, \overline{x}, z, \overline{r})$ denote the global state at round ℓ of an execution of protocol Π in the real world with adversary $\mathcal{A}$, environment $\mathcal{Z}$, security parameter s, input vector $\overline{x}$ for the parties, input z for the environment, and vector of random inputs $\overline{r}$.

Lemma 1. *Let f be an n-party functionality. Let Π be an n-party protocol to compute f. Define h to be the reactive functionality such that h acts like the trusted party computing f with deterrence ϵ in the covert adaptive adversary model. Then Π securely computes f with deterrence ϵ in the covert adaptive adversary model if and only if Π securely computes h in the active adaptive adversary model.*

Proof. We first prove the "if" part.

Let Π be a protocol that securely computes f with deterrence ϵ in the covert adaptive adversary model. Let $\mathcal{A}$ be a probabilistic polynomial time adversary. Let $\mathcal{Z}$ be a probabilistic polynomial time environment. Then there exists a probabilistic polynomial time simulator $\mathcal{S}_{CA}$ such that the following holds for every balanced vector $\overline{x}$

$$\left\{IDEALCA^{\epsilon}_{f,\mathcal{S}_{CA},\mathcal{Z}}(s,\overline{x},z)\right\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}} \stackrel{c}{\equiv} \left\{REAL_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z)\right\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}}. \tag{1}$$

For a given adversary $\mathcal{A}$ and environment $\mathcal{Z}$, we construct a simulator $\mathcal{S}_{AA}$ for functionality h in the active adaptive adversary model, from the simulator $\mathcal{S}_{CA}$ as follows.

Let $\overline{r}^f = \{r^f_z, r^f_0, r^f_1\}$ denote the vector of random inputs of the environment, the adversary and the trusted party for functionality f. Here r^f_z is the random input of the environment, r^f_0 is the random input of the adversary, and r^f_1 is the random input of the trusted party. Similarly we define $\overline{r}^h = \{r^h_z, r^h_0, r^h_1\}$ as the vector of random inputs of the environment, the adversary and the trusted party for functionality h.

Let $\overline{r}^{\Pi} = \{r^{\Pi}_z, r^{\Pi}_0, r^{\Pi}_1, \ldots, r^{\Pi}_n\}$ denote the vector of random inputs of the environment, the adversary and the parties for protocol Π. Here r^{Π}_z is the random input of the environment, r^{Π}_0 is the random input of the adversary, and r^{Π}_i is the random input of party P_i.

Let ℓ_{er} denote the round at which the environment $\mathcal{Z}$ halts in the real world. Let ℓ_{ei} denote the round at which the environment $\mathcal{Z}$ halts in the ideal world.

We fix an input vector $\overline{x}$, an environment $\mathcal{Z}$ with input z, and the security parameter s. We fix the vector $\overline{r}^f$. Since h is defined in a way such that it mimics the trusted party computing f, the vector $\overline{r}^f$ works as the vector $\overline{r}^h$ of random inputs for h. We set $\overline{r}^h$ to be equal to $\overline{r}^f$. We fix the vector $\overline{r}^{\Pi}$.

$\mathcal{S}_{AA}$ invokes $\mathcal{S}_{CA}$ on input $\mathcal{A}, \mathcal{Z}, r_0, r_z, s$ where r_0 is the random input of $\mathcal{A}$, r_z is the random input of $\mathcal{Z}$ and s is the security parameter.

The first corruption stage in the ideal worlds in two definitions are identical. So $\mathcal{S}_{AA}$ follows whatever $\mathcal{S}_{CA}$ does.

The set I_1 of corrupted parties (the set of parties that are corrupted before the evaluation of functionality starts in the ideal world or the set of parties that are corrupted before the execution of protocol starts in the real world) are identical in both executions.

In the computation stage, $\mathcal{S}_{AA}$ receives the inputs of the corrupted parties for functionality h from $\mathcal{A}$. Then $\mathcal{S}_{AA}$ sends these inputs to $\mathcal{S}_{CA}$.

If $\mathcal{S}_{CA}$ replies with $abort_i$, then $\mathcal{S}_{AA}$ sends $abort_i$ to $\mathcal{A}$ and halts.

If $\mathcal{S}_{CA}$ replies with $corrupted_i$, then $\mathcal{S}_{AA}$ sends $corrupted_i$ to $\mathcal{A}$ and halts.

If $\mathcal{S}_{CA}$ replies with *undetected* and the inputs of the honest parties, then $\mathcal{S}_{AA}$ sends them to $\mathcal{A}$. Then $\mathcal{S}_{AA}$ receives the output vector of the adversary's choice for the honest parties from $\mathcal{A}$. $\mathcal{S}_{AA}$ sends this vector to $\mathcal{S}_{CA}$ and halts.

If $\mathcal{S}_{CA}$ replies with the outputs of the corrupted parties, then $\mathcal{S}_{AA}$ sends these outputs to $\mathcal{A}$ and halts.

By definition, h is a reactive functionality that acts like the trusted party computing f with deterrence ϵ in the covert adaptive adversary model.

If $\mathcal{A}$ sends $abort_i$ as input of a corrupted party P_i, then h sends $abort_i$ to the honest parties and halts.

If $\mathcal{A}$ sends $corrupted_i$ as input of a corrupted party P_i, then h sends $corrupted_i$ to the honest parties and halts.

If $\mathcal{A}$ sends $cheat_i$ as input of a corrupted party P_i, then h acts as follows.

With probability ϵ, h sends $corrupted_i$ to the honest parties and $\mathcal{A}$, and then halts.

With probability $(1-\epsilon)$, h sends *undetected* and the inputs of the honest parties to $\mathcal{A}$. Then h receives the vector of outputs of the adversary's choice for the honest parties from $\mathcal{A}$. Then h sends these outputs to the corresponding honest parties and halts.

If no input is $abort_i, corrupted_i$ or $cheat_i$, then h computes functionality f and sends each party its output.

The inputs of the corrupted parties are identical in both executions. The inputs of the honest parties are identical in both executions.

That means the global state after the execution of f in the computation stage in the ideal world of the covert adversary model is identically distributed to the global state after the execution of h in the computation stage in the ideal world of the active adaptive adversary model.

For the remaining stages, $\mathcal{S}_{AA}$ follows whatever $\mathcal{S}_{CA}$ does.

Therefore

$$GS_{h,\mathcal{S}_{AA},\mathcal{Z}}(\ell_{ei}, s, \overline{x}, z, \overline{r}^h) = GS_{f,\mathcal{S}_{CA},\mathcal{Z}}(\ell_{ei}, s, \overline{x}, z, \overline{r}^f). \tag{2}$$

From (1), we have

$$GS_{f,\mathcal{S}_{CA},\mathcal{Z}}(\ell_{ei}, s, \overline{x}, z, \overline{r}^f) \stackrel{c}{\equiv} GS_{\Pi,\mathcal{A},\mathcal{Z}}(\ell_{er}, s, \overline{x}, z, \overline{r}^{\Pi}). \tag{3}$$

The real worlds are the same in the active adaptive adversary model and the covert adaptive adversary model.

From (2) and (3), we have

$$GS_{h,\mathcal{S}_{AA},\mathcal{Z}}(\ell_{ei}, s, \overline{x}, z, \overline{r}^h) \stackrel{c}{\equiv} GS_{\Pi,\mathcal{A},\mathcal{Z}}(\ell_{er}, s, \overline{x}, z, \overline{r}^{\Pi}). \tag{4}$$

By letting vectors $\overline{r}^f$ and $\overline{r}^{\Pi}$ to be chosen uniformly at random from their domain, from (4), we can say that the following holds for every balanced vector $\overline{x}$

$$\{IDEAL_{h,\mathcal{S}_{AA},\mathcal{Z}}(s, \overline{x}, z)\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}} \stackrel{c}{\equiv} \{REAL_{\Pi,\mathcal{A},\mathcal{Z}}(s, \overline{x}, z)\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}}.$$

That means Π securely computes h in the active adaptive adversary model.

Next we prove the "only if" part.

Let Π be a protocol that securely computes h in the active adaptive adversary model.

Let $\mathcal{A}$ be a probabilistic polynomial time adversary. Let $\mathcal{Z}$ be a probabilistic polynomial time environment. Then there exists a probabilistic polynomial time simulator $\mathcal{S}_{AA}$ such that the following holds for every

balanced vector $\overline{x}$

$$\{IDEAL_{f,\mathcal{S}_{AA},\mathcal{Z}}(s,\overline{x},z)\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}} \stackrel{c}{\equiv} \{REAL_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z)\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}}. \quad (5)$$

For a given adversary $\mathcal{A}$ and environment $\mathcal{Z}$, we construct a simulator $\mathcal{S}_{CA}$ for computing functionality f with deterrence ϵ in the covert adaptive adversary model, from the simulator $\mathcal{S}_{AA}$ as follows.

We fix an input vector $\overline{x}$, an environment $\mathcal{Z}$ with input z, and the security parameter s. We fix the vector $\overline{r}^h$. Since h is defined in a way such that it mimics the trusted party computing f, the vector $\overline{r}^h$ works as the vector $\overline{r}^f$ of random inputs for f. We set $\overline{r}^f$ to be equal to $\overline{r}^h$. We fix the vector $\overline{r}^{\Pi}$.

$\mathcal{S}_{CA}$ invokes $\mathcal{S}_{AA}$ on input $\mathcal{A},\mathcal{Z},r_0,r_z,s$ where r_0 is the random input of $\mathcal{S}_{AA}$, r_z is the random input of $\mathcal{Z}$ and s is the security parameter.

The first corruption stage in the two definitions are identical. So $\mathcal{S}_{CA}$ follows whatever $\mathcal{S}_{AA}$ does.

In the computation stage, $\mathcal{S}_{CA}$ receives the inputs of the corrupted parties for functionality f from $\mathcal{A}$. Then $\mathcal{S}_{CA}$ sends these inputs to $\mathcal{S}_{AA}$.

For the remaining stages, $\mathcal{S}_{CA}$ follows whatever $\mathcal{S}_{AA}$ does.

By definition, h is a reactive functionality that acts like the trusted party computing f with deterrence ϵ in the covert adaptive adversary model.

Therefore

$$GS_{f,\mathcal{S}_{CA},\mathcal{Z}}(\ell_{ei},s,\overline{x},z,\overline{r}^f) = GS_{h,\mathcal{S}_{AA},\mathcal{Z}}(\ell_{ei},s,\overline{x},z,\overline{r}^h). \quad (6)$$

From (5), we have

$$GS_{h,\mathcal{S}_{AA},\mathcal{Z}}(\ell_{ei},s,\overline{x},z,\overline{r}^h) \stackrel{c}{\equiv} GS_{\Pi,\mathcal{A},\mathcal{Z}}(\ell_{er},s,\overline{x},z,\overline{r}^{\Pi}). \quad (7)$$

From (6) and (7), we have

$$GS_{f,\mathcal{S}_{CA},\mathcal{Z}}(\ell_{ei},s,\overline{x},z,\overline{r}^f) \stackrel{c}{\equiv} GS_{\Pi,\mathcal{A},\mathcal{Z}}(\ell_{er},s,\overline{x},z,\overline{r}^{\Pi}). \quad (8)$$

By letting vectors $\overline{r}^h$ and $\overline{r}^{\Pi}$ to be chosen uniformly at random from their domain, from (8), we can say that the following holds for every balanced vector $\overline{x}$

$$\left\{IDEALSC^{\epsilon}_{f,\mathcal{S}_{CA},\mathcal{Z}}(s,\overline{x},z)\right\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}} \stackrel{c}{\equiv} \{REAL_{\Pi,\mathcal{A},\mathcal{Z}}(s,\overline{x},z)\}_{\overline{x},z\in(\{0,1\}^*)^{n+1};s\in\mathbb{N}}.$$

Therefore Π securely computes f with deterrence ϵ in the covert adaptive adversary model.

Canetti [Can00] proved sequential composition theorem for the active adaptive adversary model. The sequential composition theorem for active adaptive adversaries states that if protocols $\rho_1,\ldots,\rho_{p(s)}$ securely compute functionalities $f_1,\ldots,f_{p(s)}$ respectively in the presence of active adaptive adversaries, and protocol Π securely computes functionality g in the $f_1,\ldots,f_{p(s)}$-hybrid world, then $\Pi^{\rho_1,\ldots,\rho_{p(s)}}$ securely computes g in the presence of active adaptive adversaries (in the real model) [Can00]. Canetti [Can00] proved the sequential composition theorem for standard, non-reactive functionalities. As described in Section 2.5, the proof of Canetti [Can00] can be extended to reactive functionalities.

Theorem 3 (Sequential Composition Theorem for Covert Adaptive Adversary Model.).

Let $p(s)$ be a polynomial. Let $f_1,\ldots,f_{p(s)}$ be n-party probabilistic polynomial-time functionalities. Let $\rho_1,\ldots,\rho_{p(s)}$ be protocols that securely compute functionalities $f_1,\ldots,f_{p(s)}$ in the presence of covert adaptive adversaries with deterrence $\epsilon_1,\ldots,\epsilon_{p(s)}$, respectively. Let g be an n-party functionality. Let Π be a protocol that securely computes g in the $(f_1,\epsilon_1),\ldots,(f_{p(s)},\epsilon_{p(s)})$-hybrid world (using a single call to each f_i in such a way that no more than one ideal evaluation of a functionality is made at each round) in the presence of covert adaptive adversaries with deterrence ϵ. Then $\Pi^{\rho_1,\ldots,\rho_{p(s)}}$ securely computes g in the presence of covert adaptive adversaries with deterrence ϵ.

Proof. For functionality f_i, define h_i to be the reactive functionality such that it acts as the trusted party computing f_i in the presence of covert adaptive adversaries with deterrence ϵ_i. By Lemma 1, protocol ρ_i securely computes f_i in the presence of covert adaptive adversaries with deterrence ϵ_i if and only if ρ_i securely computes the reactive functionality h_i in the presence of active adaptive adversaries. Then we can apply the sequential composition theorem for active adaptive adversary model of [Can00] for reactive functionalities to the functionality Π with a subroutine call to protocol ρ_i, and thereby obtain the sequential composition theorem for covert adaptive adversaries.

References

AL10. Yonatan Aumann and Yehuda Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.

Can00. Ran Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

Can01. Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings, 42nd IEEE Symposium on Foundations of Computer Science — FOCS '01*, pages 136–145, Washington, DC, USA, 2001. IEEE Computer Society. Full version available at http://eprint.iacr.org/2000/067.

GM84. Shari Goldwasser and Silvio Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

GMR85. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The Knowledge Complexity of Interactive Proof-Systems. In *Proceedings, 17th Annual ACM Symposium on Theory of Computing — STOC '85*, pages 291–304, New York, NY, USA, 1985. ACM.

Gol06. Oded Goldreich. *Foundations of Cryptography: Volume 1, Basic Techniques*. Cambridge University Press, New York, NY, USA, 2006.

Gol09. Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

HL10. Carmit Hazay and Yehuda Lindell. *Efficient Secure Two-Party Protocols: Techniques and Constructions*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010.