Parallel Reducibility for Information-Theoreticaly Secure Computation

Yevgeniy Dodis* Silvio Micali†

Abstract

Secure Function Evaluation (SFE) protocols are very hard to design, and *reducibility* has been recognized as a highly desirable property of SFE protocols. Informally speaking, reducibility (a.k.a. modular composition) is the automatic ability to break up the design of a complex SFE protocols into several simpler, individually secure components. Despite much effort, only the most basic type of reducibility, *sequential reducibility* (where only a single sub-protocol can be run at a time), has been considered and proven to hold for a specific class of SFE protocols. Unfortunately, sequential reducibility does not allow one to save on the number of rounds (often the most expensive resource in a distributed setting), and achieving more general notions is not easy (indeed, certain SFE notions provably enjoy sequential reducibility, but fail to enjoy more general ones).

In this paper, for information-theoretic SFE protocols, we

- Formalize the notion of *parallel reducibility*, where sub-protocols can be run at the same time;
- Clarify that there are two *distinct* forms of parallel reducibility:
  - ⋆ *Concurrent reducibility*, which applies when the order of the sub-protocol calls is not important (and reduces the round complexity dramatically as compared to sequential reducibility); and
  - ⋆ *Synchronous reducibility*, which applies when the sub-protocols must be executed simultaneously (and allows modular design in settings where sequential reducibility does not even apply).
- Show that a large class of SFE protocols (i.e., those satisfying the definitions of [22]) provably enjoy (both forms of) parallel reducibility.

* Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139. E-mail: `yevgen@theory.lcs.mit.edu`.

† Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139. E-mail: `silvio@theory.lcs.mit.edu`.

1 Introduction

The objective of this paper is to understand, define, and prove the implementability of the notion of parallel reducibility for information-theoretically secure multi-party computation. Let us start by discussing the relevant concepts.

SFE Protocols. A secure function evaluation (SFE) is a communication protocol enabling a network of players (say, having a specified threshold of honest players) to compute a (probabilistic) function in a way that is as correct and as private as if an uncorruptable third party had carried out the computation on the players' behalf. SFE protocols were introduced by Goldreich, Micali and Wigderson [20] in a *computational* setting (where the parties are computationally bounded, but can observe all communication), and by Ben-Or, Goldwasser and Wigderson [5] and Chaum, Crèpeau and Damgård [13] in an *information-theoretic* setting (where the security is unconditional, and is achieved by means of *private channels*[1]). We focus on the latter setting.

SFE Definitions. Together with better SFE protocols, increasingly precise definitions for information-theoretic SFE have been proposed; in particular, those of Beaver [2], Goldwasser and Levin [16], Canetti [7], and Micali and Rogaway [22]. At a high-level, these definitions express that whatever an adversary can do in the *real model* (i.e., in the running of the actual protocol, where no trusted party exists) equals what an adversary can do in the *ideal model* (i.e., when players give their inputs to the trusted third party, who then computes the function for them). This more or less means that the most harm the adversary can do in the real model is to change the inputs of the faulty players (but not based on the inputs of the honest players!), and then run the protocol honestly.

All these prior definitions are adequate, in the sense that they (1) reasonably capture the desired intuition of SFE, and (2) provide for the existence of SFE protocols (in particular, the protocol of [5] satisfies all of them). Were properties (1) and (2) all one cared about, then the most "liberal" definition of SFE might be preferable, because it would allow a greater number of reasonable protocols to be called secure. However, if one cared about satisfying *additional* properties, such as reducibility (i.e., as discussed below, the ability of designing SFE protocols in a modular fashion), then *more stringent* notions of SFE would be needed.

Reducibility and Sequential Reducibility. Assume that we have designed a SFE protocol, F, for a function f in a so called *semi-ideal* model, where one can use a trusted party to evaluate some other functions $g^1, \ldots, g^k$. Assume also that we have designed a SFE protocol, G_i, for each function g^i. Then, the reducibility property says that, by substituting the ideal calls to the g^i's in F with the corresponding SFE protocols G_i's, we are *guaranteed* to obtain a SFE protocol for f in the *real* model.

Clearly, reducibility is quite a fundamental and desirable property to have, because it allows one to break the task of designing a secure protocol for a complex function into the task of designing secure protocols for simpler functions. Reducibility, however, is not trivial to satisfy. After considerable effort, only the the most basic notion of reducibility, *sequential reducibility*, has been proved to hold for some SFE notions: those of [7] and [22]. Informally, sequential reducibility guarantees that substituting the ideal calls to the g^i's in F with the corresponding G_i's yields a SFE protocol for f in the real model *only if* a single G_i is executed (in its entirety!) at a time.[2] Therefore, sequential reducibility is not general enough to handle protocols like the expected $O(1)$-round Byzantine agreement protocol of [15] (which relies on the concurrent execution of n^2 specific SFE protocols) whose security, up to now, must be proven "from scratch".

[1]This means that every pair of players has a dedicated channel for communication, which the adversary can listen to only by corrupting one of the players.

[2]This is true even if, within F, one could "ideally evaluate" all or many of the g^i's "in parallel."

1.1 Our Results

In this paper, we put forward the notion of *parallel reducibility* and prove which SFE protocols satisfy it. We actually distinguish two forms of parallel reducibility:

- *Concurrent reducibility.*

  This type of reducibility applies when, in the semi-ideal model, the $g^1, \ldots, g^k$ can be executed in any order. The goal of concurrent reducibility is *improving the round-complexity* of modularly designed SFE protocols.

- *Synchronous reducibility.*

  This type of reducibility applies when, in the semi-ideal model, the $g^1, \ldots, g^k$ must be executed "simultaneously." The goal of synchronous reducibility is *enlarging the class of modularly designable SFE protocols* (while being round-efficient as well).

1.1.1 Concurrent Reducibility

There are many ways to execute several programs $G_1, \ldots, G_k$ at a time. Each such way is called an *interleaving*. The $k!$ sequential executions of $G_1, \ldots, G_k$ are examples of interleavings. But they are very special and "very few," because interleavings may occur at a round-level. For instance, we could execute the G_i's one round at a time in a round-robin manner, or we could execute in single round r the r-th round (if any) of all the G_i's. Saying that programs $G_1, \ldots, G_k$ are concurrently executable means that some specified goal is achieved *for all of their interleavings*.

Assume now that a function f is securely evaluated by a semi-ideal protocol F which, in a set of contiguous instructions, only makes ideal calls to functions $g^1, \ldots, g^k$, and let G_i be a SFE protocol for g^i (in the real model). Then, a fundamental question arise:

Will substituting each g^i with G_i yield a (real-model) SFE
protocol for f in which the G_i's are concurrently executable?

Of course, if F calls g^2 on inputs that include an output of g^1, we cannot hope that the G_i's are concurrently executable. Thus, to make sense of the question, all the inputs to g^i's should be determined before any of them is ideally evaluated. Moreover, even if all g^i's are evaluated on completely unrelated and "independent" inputs, F may be secure only for some orders of the g^i's, but not for others, which is illustrated by the following example.

Example 1: Let f be the coin-flipping function (takes no inputs and outputs a joint random bit), g^1 be a coin-flipping function as well, and g^2 be the majority function on n bits. Let F be the following semi-ideal protocol. Each player P_j locally flips a random bit b_j. Then the players "concurrently" use ideal calls to g^1 and $g^2(b_1, \ldots, b_n)$, getting answers r and c respectively. The common output of F is $r \oplus c$. We claim that F is secure if we first call g^2 (the majority) and then g^1 (the coin-flip), but insecure if we do it the other way around. Indeed, irrespective of which c we get in the first ordering, since r is random (and independent of c), then so is $r \oplus c$. On the other hand, assume we first learn the random bit r and assume faulty players want the bias the resulting coin-flip to 0. Then faulty players pretend that their (supposedly random) inputs b_j for the majority are all equal to r. This is very likely to bias the outcome c of majority to r as well (provided there are enough faulty players), making the coin-flip equal to $r \oplus r = 0$ with high probability.

Clearly, in the case of the above example, we cannot hope to execute the G_i's concurrently: one of the possible interleavings is the one that sequentially executes the G_i's in the order that is insecure even in the semi-ideal model. Thus, the example illustrates that the following condition is *necessary* for the concurrent execution of the G_i's.

Condition 1: F is secure in the semi-ideal model for any order of the g^i's.

Is the above necessary condition also sufficient? Of course, the answer also depends on the type of SFE notion we are using. But, if the answer were YES, then we would get the "strongest possible form of concurrent reducibility." Let us then be optimistic and put forward the following informal definition.

Definition 1: We say that a SFE notion satisfies *concurrent reducibility* if, whenever the protocols $F, G_1, \ldots, G_k$ satisfy this SFE notion, Condition 1 is (both necessary and) sufficient.

Our optimism is justified in view of the following

Theorem 1: The SFE notion of Micali and Rogaway [22] satisfies concurrent reducibility.

We note that we have been unable to prove an analogous theorem for all other more liberal notions of SFE, and we conjecture that no such theorem exist. In support of our conjecture, we shall point out in Section 4.3 which stricter properties of the definition of [22] seem to be essential in establishing Theorem 1.

The importance of establishing (as in Theorem 1) the existence of SFE notions satisfying concurrent reducibility arises from the efficiency gains of concurrent reducibility, as expressed by the following immediate Corollary of Definition 1.

Corollary 1: Assume $F, g^1, \ldots, g^k$ satisfy Condition 1, G_i is a protocol for g^i taking R_i rounds, and $F, G_1, \ldots, G_k$ are SFE protocols according to a SFE notion satisfying concurrent reducibility. Then, there is a (real model) SFE implementation of F executing all the G_i's in $\max(R_1, \ldots, R_k)$ rounds.

This number of rounds is the smallest one can hope for, and should be contrasted with $R_1 + \cdots + R_k$, the number of rounds required by sequential reducibility.

1.1.2 Synchronous Reducibility

The need to execute several protocols in parallel does not necessarily arise from efficiency considerations or from the fact that it is nice not to worry about the order of the execution. A special type of parallel execution, *synchronous execution*, is needed for correctness itself.

Example 2: Let f be the coin-flipping function that returns a random bit to the first two players, P_1 and P_2, of a possibly larger network. That is, $f(\lambda, \lambda, \lambda, \ldots, \lambda) = (x, x, \lambda, \ldots, \lambda)$, where x is a random bit (and λ is the empty string). Consider now the following coin-flipping protocol F. P_1 randomly and secretly selects a bit x_1, P_2 randomly and secretly selects a bit x_2, and then P_1 and P_2 "exchange" their selected bits and both output $x = x_1 \oplus x_2$.

Clearly, F is a secure function evaluation of f only if the exchange of x_1 and x_2 is "simultaneous", that is, P_1 learns x_2 only after it declares x_1 and vice versa. This requirement can be modeled as the parallel composition of two sending protocols: $g^1(x_1, \lambda, \lambda, \ldots, \lambda) = (x_1, x_1, \lambda, \ldots, \lambda)$ and $g^2(\lambda, x_2, \lambda, \ldots, \lambda) = (x_2, x_2, \lambda, \ldots, \lambda)$. That is, we can envisage a semi-ideal protocol in which players P_1 and P_2 locally flip coins x_1 and x_2, then *simultaneously* evaluate g^1 and g^2, and finally exclusive OR their outputs of g^1 and g^2. However, *no sequential order* of the ideal calls to g^1 and g^2 would result in a secure coin-flipping protocol, so the need for a special type of parallel composition is motivated by security rather than efficiency considerations.

The ability to evaluate several functions *synchronously* is very natural to define in the ideal model: the players simultaneously give all their inputs to the trusted party, who then gives them all the outputs (i.e., no output is given before all inputs are presented). We can also naturally define the corresponding semi-ideal model, where the players can ideally and simultaneously (i.e., within a single round) evaluate several functions. Assume now that we have a semi-ideal protocol F for some function f which simultaneously evaluates functions $g^1, \ldots, g^k$, and let G_i be a secure protocol for g^i. Given an interleaving I of the G_i's, we let F^I denote the (real-model) protocol where we substitute the *single* ideal call to $g^1, \ldots, g^k$ with k real executions of the protocols G_i interleaved according to I. As apparent from Example 2, we cannot hope that *every* interleaving I will be "good," that is, will yield a SFE protocol F^I for f. (For instance, in the semi-ideal coin-flipping protocol F of Example 2, no matter how we design SFE protocols G_1 and G_2 for g^1 and g^2, any sequential interleaving of G_1 and G_2 yields an insecure protocol.) Actually, the guaranteed existence of even a *single* good interleaving cannot be taken for granted, therefore:

Can we be guaranteed that there is always an interleaving I of $G_1, \ldots, G_k$ such that F^I is a SFE protocol for f?

Of course, the answer to the above question should depend on the notion of SFE we are using. This leads us to the following informal definition.

Definition 2: We say that a SFE notion satisfies *synchronous reducibility* if, whenever the protocols $F, G_1, \ldots, G_k$ satisfy this SFE notion, there exists an interleaving I such that F^I is a SFE protocol under this notion.

Example 2 not only shows that there are bad interleavings, but also that a "liberal" enough definition of SFE will not satisfy synchronous reducibility. Indeed, according to the SFE notions of [7, 2, 16], the protocol G_1 consisting of player P_1 sending x_1 to player P_2 is a secure protocol for g^1. Similarly, the protocol G_2 consisting of player P_2 sending x_2 to player P_1 is a secure protocol for g^2. However, there is *no interleaving* of G_1 and G_2 that will result in a secure coin-flip. This is because the last player to send its bit (which includes the case when the players exchange their bits in one round, due to the "rushing" ability of the adversary; see Section 2) is completely controlling the outcome. Thus, this example shows that the SFE notions of [7, 2, 16] *do not support synchronous reducibility*. However, we show[3]

Theorem 2: The SFE notion of Micali and Rogaway [22] satisfies synchronous reducibility.

Theorem 2 actually has a quite constructive nature. Namely, the nature of the definition in [22] not only guarantees that "good" interleavings I always exist, but also that there are many of them, that they are easy to find, and that some of them produce efficient protocols. We summarize the last property in the following corollary.

Corollary 2: With respect to the Micali-Rogaway definition of SFE, let F be an ideal protocol for f that simultaneously calls the functions $g^1, \ldots, g^k$, and let G_i be an R_i-round SFE protocol for g_i. Then there exists (an easy to find) interleaving I of the G_i's, consisting of $\leq 2\max(R_1, \ldots, R_k)$ rounds, such that F^I is secure.

In other words, independent on the number of sub-protocols, we can synchronously interleave them using at most twice as many rounds as the longest of them takes.[4] Let us remark that, unlike Corollary 1 (that simply follows from the definition of concurrent reducibility), Corollary 2 crucially depends on the very notion of [22], as is discussed more in Section 4.3.

[3] As is illustrated in Section 4.3, the above "natural" protocols G_1 and G_2 are indeed insecure according to the definition of [22].

[4] We note that the factor of 2 is typically too pessimistic. As it will be clear from the precise statement of synchronous reducibility in Section 3, natural protocols G_i (like the ones designed using a general paradigm of [5]) can be synchronously interleaved in $\max(R_1, \ldots, R_k)$ rounds.

1.1.3 In Sum

We have clarified the notion of parallel reducibility, distilled two important flavors of it, and showed that there exist SFE notions (e.g., the one of [22]) as well as general SFE protocols (e.g., the one of [5]) that satisfy (both forms of) parallel reducibility.

Theorems 1 and 2 (and their corollaries) do not necessarily imply that the definition of [22] is "preferable" to other others. If the protocol one is designing is simple enough or is unlikely to be composed in parallel with other protocols, other definitions are equally adequate (and may actually be simpler to use). It is, however, crucial to understand which SFE notions yield parallel reducibility if we want to simplify the complex task of designing secure computation protocols.

2 The Micali-Rogaway Definition of SFE

Consider a probabilistic function $f(\mathbf{x}, r) = (f_1(\mathbf{x}, r), \ldots, f_n(\mathbf{x}, r))$ (where $\mathbf{x} = (x_1, \ldots, x_n)$). We wish to define a protocol F for computing f that is *secure* against any *adversary* A that is allowed to *corrupt* in a dynamic fashion up to t (out of n) players.[5]

2.1 Protocols and Adversaries

Protocol: An n-party *protocol* F is a tuple $(\hat{F}, LR, CR, \mathcal{I}, \mathcal{O}, f)$ where

- $\hat{F}$ is a collection of n interactive probabilistic Turing machines that interact in synchronous rounds.
- LR — the last round of F (a fixed integer, for simplicity).
- CR — the *committal round* (a fixed integer, for simplicity).
- $\mathcal{I}$ — the *effective-input function*, a computable function from strings to strings.
- $\mathcal{O}$ — the *effective-output function*, a computable function from strings to strings.
- f — a (probabilistic) function being allegedly computed.

Adversary: An *adversary* A is a probabilistic algorithm.

Executing F and A: Adversary A interacts with protocol F as a traditional adaptive adversary in the rushing model. Roughly, this is explained below.

The execution of F with an adversary A proceeds as follows. Initially, each player j has an input x_j (for f) and an auxiliary input a_j, while A has an auxiliary input α. (Auxiliary inputs represent any a-priori information known to the corresponding party like the history of previous protocol executions. An honest player j should ignore a_j, but a_j might be useful later to the adversary.) At any point during the execution of F, A is allowed to corrupt some player j (as long as A corrupts no more than t players overall). By doing so, A learns the entire *view of* j (i.e., x_j, a_j, j's random tape, and all the messages sent and received by j) up to this point input. From now on, A can completely control the behavior of j and thus make j deviate from F in any malicious way. At the beginning of each round, A first learns all the messages sent from currently good players to the corrupted ones.[6] Then A can adaptively corrupt several players, and only then does he send the messages from bad players to good ones. Without loss of generality, A never sends a message from a bad player to another bad player.

[5] More generally, one can have an adversary that can corrupt only certain "allowable" subsets of players. The collection of these allowable subsets is usually called the *adversary structure*. For simplicity purposes only, we consider *threshold* adversary structures, i.e. the ones containing all subsets of cardinality t or less. We call any such adversary t*-restricted*.

[6] We can even let the adversary schedule the delivery of good-to-bad messages and let him adaptively corrupt a new player in the middle of this process. For simplicity, we stick to our version.

At the end of F, the *view of* A, denoted $\mathit{View}(A, F)$ consists of α, A's random coins and the views of all the corrupted players. The *traffic of a player* j *up to round* R consists of all the messages received and sent by j up to round R. Such traffic is denoted $\mathit{traffic}_j(R)$ (or by $\mathit{traffic}_j(R, F[A])$ whenever we wish to stress the protocol and the adversary executing with it).

Effective Inputs and Outputs of a Real Execution: In an execution of F with A, the *effective input* of player j (whether good or bad), denoted $\hat{x}_j^F$, is determined at the *committal round* CR by evaluating the effective-input function $\mathcal{I}$ on j's traffic at round CR: $\hat{x}_j^F = \mathcal{I}(\mathit{traffic}_j(CR, F[A]))$. The *effective output* of player j, denoted $\hat{y}_j^F$, is determined from j's traffic at the last round LR via the effective output function $\mathcal{O}$: $\hat{y}_j^F = \mathcal{O}(\mathit{traffic}_j(LR, F[A]))$. Note that, for now, the effective inputs $\hat{\mathbf{x}}^F$ and outputs $\hat{\mathbf{y}}^F$ are unrelated to computing f.

History of a Real Execution: We let the *history of a real execution*, denoted $\mathit{History}(A, F)$, to be $\langle \mathit{View}(A, F),\ \hat{\mathbf{x}}^F,\ \hat{\mathbf{y}}^F \rangle$. Intuitively, the history contains all the relevant information of what happened when A attacked the protocol F: the view of A, i.e. what he "learned", and the effective inputs and outputs of all the players.

2.2 Simulators and Adversaries

Simulator: A *simulator* is a probabilistic, oracle-calling, algorithm S.

Executing S with A: Let A be an adversary for a protocol F for function f. In an execution of S with A, there are no real players and there is no real network. Instead, S interacts with A in a round-by-round fashion, playing the role of all currently good players in an execution of A with the real network, i.e.: (1) (makes up and) sends to A a view of a player j immediately after A corrupts j, (2) sends to A the messages of currently good players to currently bad players[7] and (3) receives the messages sent by A (on behalf of the corrupted players) to currently good players. In performing these tasks, S makes use of the following *oracle* $O(\mathbf{x}, \mathbf{a})$[8]:

- *Before* CR. When a player j is corrupted by A before the committal round, O immediately sends S the input values x_j and a_j. In particular, S uses these values in making up the view of j.
- *At* CR. At the end of the committal round CR, S sends O the value $\hat{x}_j^S = \mathcal{I}(\mathit{traffic}_j(CR))$ for each corrupted player j.[9] In response, O randomly selects a string r, sets $\hat{x}_j^S = x_j$ for all currently good players j, computes $\hat{\mathbf{y}}^S = f(\hat{\mathbf{x}}^S, r)$, and for each corrupted player j sends $\hat{y}_j^S$ back to S.
- *After* CR. When a player j is corrupted by A after the committal round, O immediately sends S the input values x_j and a_j, as well as the computed value $\hat{y}_j^S$. In particular, S uses these values in making up the view of j.

We denote by $\mathit{View}(A, S)$ the view of A when interacting with S (using O).

Effective Inputs and Outputs of a Simulated Execution: Consider an execution of S (using oracle $O(\mathbf{x}, \mathbf{a})$) with adversary A. Then, the effective inputs of this execution consist of the above defined values $\mathbf{x}^S$. Namely, if a player j is corrupted before the committal round CR, then its effective input is $\hat{x}_j^S = \mathcal{I}(\mathit{traffic}_j(CR, S[A]))$; otherwise ($j$ is never corrupted, or is corrupted after the committal round) its effective input is $\hat{x}_j^S = x_j$. The effective outputs are the values $\mathbf{y}^S$ defined above. Namely, $\hat{\mathbf{y}}^S = f(\hat{\mathbf{x}}^S, r)$.

[7]Notice that S does not (and cannot) produce the messages from good players to good players.

[8]Such oracle is meant to represent the trusted party in an ideal evaluation of f. Given this oracle, S's goal is making A believe that it is executing F in a real network in which the players have inputs $\mathbf{x}$ and auxiliary inputs $\mathbf{a}$.

[9]Here $\mathit{traffic}_j(R) = \mathit{traffic}_j(R, S[A])$ of a corrupted player j denotes what A "thinks" the traffic of j after round R is.

History of a Simulated Execution: We let the *history of a simulated execution*, denoted $History(A, S)$, to be $\langle View(A, S),\ \hat{\mathbf{x}}^S,\ \hat{\mathbf{y}}^S \rangle$. Intuitively, the history contains all the relevant information of what happened when A was communicating with S (and O): the view of A, i.e. what he "learned", and the effective inputs and outputs of all the players.

2.3 Secure Computation

Definition 3: An n-party protocol F is a SFE protocol resilient against t-restricted adversaries that computes a probabilistic n-input/n-output function $f(\mathbf{x}, r)$, if there exists a simulator S such that for any input $\mathbf{x} = (x_1, \ldots, x_n)$, auxiliary input $\mathbf{a} = (a_1, \ldots, a_n)$, and any t-restricted adversary A with some auxiliary input α, the histories of the real and the simulated executions are identically distributed:

$$History(A, F) \equiv History(A, S) \tag{1}$$

Equivalently, $\langle View(A, F),\ \hat{\mathbf{x}}^F,\ \hat{\mathbf{y}}^F \rangle \equiv \langle View(A, S),\ \hat{\mathbf{x}}^S,\ \hat{\mathbf{y}}^S \rangle$.

Simulators and Oracles vs. Ideal Adversaries. A standard benchmark in determining if a SFE notion is "reasonable" is the fact that for every real adversary A there exists an "ideal adversary" A' that can produce (in the ideal model with the trusted party) the same view as A got from the real network.[10] We argue that the existence of a simulator S in the Micali-Rogaway definition indeed implies the existence of such an adversary A'. A' simply runs A against the simulator S. If A corrupts a player j before the committal round, A' corrupts j in the ideal model, and gives the values x_j and a_j (that it just learned) to S on behalf of the oracle O. Right after the committal round of F has been simulated by S, A' computes from the traffic of A the effective inputs $\hat{x}_j^S$ of currently corrupted players j, hands them to the trusted party, and returns the outputs of the corrupted players to S on behalf of O. Finally, if A corrupts a player j after the committal round, A' corrupts j in the ideal model, and gives the values x_j, a_j and the output of j (that it just learned) to S on behalf of the oracle O. At the end, A' simply outputs the resulting view of A in the simulation.[11]

We notice, however, that the "equivalent" ideal adversary A' implied by the definition of [22] is much more special than the possible ideal adversary envisaged by other definitions (e.g., [7]).[12]

3 The Notion of Parallel Reducibility

First, let us define the *semi-ideal* model which generalizes the real model with the ability to ideally evaluate some functions. More precisely, in addition to regular rounds (where each player sends messages to other players), the semi-ideal model allows players to have *ideal rounds*. In such a round, the players can *simultaneously* evaluate several functions $g^1, \ldots, g^k$ using a trusted third party. More specifically, at the beginning of this round each player gives the k-tuple of his inputs to a trusted party. At the end of the round, each player gets back from the trusted party the corresponding k-tuple of outputs. (Note, these k-tuples are parts of players' traffic.)

The Micali-Rogaway definition of security of a protocol F in the semi-ideal model is the same as that of a real model protocol with the following addition:

- The simulator S has to simulate all the ideal rounds as well, since they are part of what the adversary A expects. S has to do this using no special "g-oracle". In other words, given the g-inputs of corrupted

[10] In fact, this requirement is more or less the SFE definition of [7].

[11] The construction of A' intuitively explains the definition of effective inputs $\hat{\mathbf{x}}^S$ and effective outputs $\hat{\mathbf{y}}^S$ of the simulated execution, as they are exactly the inputs/outputs in the run of A' in the ideal model.

[12] For instance, such A' is constrained to run A only once and in a black-box manner.

players in an ideal round, S has to generate the corresponding outputs of corrupted players and give them back to A. Also, when A corrupts a player j, S has to produce on its own the g-inputs/outputs of player j during all the ideal rounds that happened so far (as these are parts of j's traffic, and therefore j's view).

Let F be a SFE protocol for f in the semi-ideal model, and let us fix our attention on any particular ideal round R that evaluates some functions $g^1, \ldots, g^k$. We say that the ideal round R is *order-independent* if for any sequential ordering π of $g^1, \ldots, g^k$, semi-ideal protocol F remains secure if we replace the ideal round R with k ideal rounds evaluating a single g^i at a time in the order given by π (we denote this semi-ideal protocol by F^π).

Let $G_1, \ldots, G_k$ be SFE protocols for $g^1, \ldots, g^k$. We would like to substitute the ideal calls to g^i's with the corresponding protocols G_i's and still get a secure protocol for f. As we informally argued before, there are many ways to substitute (or to *interleave*) the G_i's, which is made precise by the following definition.

Definition 4:

- An *interleaving* of protocols $G_1, \ldots, G_k$ is any schedule I of their execution. Namely, a single round of an interleaving may execute in parallel one round of one or more G_i's with the only restriction that the rounds of each G_i are executed in the same order as they are in G_i.
- A *synchronous interleaving* of protocols $G_1, \ldots, G_k$ with committal rounds $CR_1, \ldots, CR_k$ is any interleaving I such that for any $1 \leq i, \ell \leq k$, round CR_i of G_i strictly precedes round $CR_\ell + 1$ of G_ℓ. We call the place after all the "pre-committal" rounds but before all the "post-committal" rounds the *synchronization point of I*.
- Given an interleaving I of $G_1, \ldots, G_k$, we let F^I be a protocol obtained by substituting the ideal round R with the execution of the protocols $G_1, \ldots, G_k$ in the order specified by I. The committal round of F^I, its effective input and output functions are defined in a straightforward manner from those of F and $G_1, \ldots, G_k$. More specifically, given the traffic of j in F^I, we replace all j's traffic inside G_i (if any) with the *effective inputs and outputs* of j in G_i, and apply the corresponding effective input/output function of F to the resulting traffic. We also remark that when we run G_i, we let the auxiliary input of player j to be its view of the computation so far.

The fundamental question addressed by parallel reducibility is

Assuming $F, G_1, \ldots, G_k$ are SFE protocols, under which conditions is F^I a SFE protocol as well?

We highlight two kinds of sufficient conditions: (1) special properties of the protocol F making F^I secure *irrespective of I* (which will lead us to *concurrent reducibility*), and (2) restrictions on the interleaving I such that mere security of F and $G_1, \ldots, G_k$ is enough (which will lead us to *synchronous reducibility*). The following Main Theorem restates Theorem 1 and 2 of the introduction.

Parallel-Reducibility Theorem: Consider the SFE notion of Micali-Rogaway. Let F be a semi-ideal SFE protocol for f evaluating $g^1, \ldots, g^k$ in an ideal round R; let G_i be a SFE protocol for g^i; and let I be an interleaving of $G_1, \ldots, G_k$. Then F^I is a SFE protocol for f if either of the following conditions holds:

1. **(Concurrent-Reducibility Theorem)** R is an order-independent round of F.
2. **(Synchronous-Reducibility Theorem)** I is a synchronous interleaving.

As we argued in the introduction, if we want F^I to be secure for all I, round R must be order-independent. Thus, Micali-Rogaway definition achieves the strongest form of concurrent reducibility. On the other, hand, we also argued that if we do not put any extra conditions on F and $G_1, \ldots, G_k$ (aside from being SFE protocols), not all interleavings I necessarily result in a SFE protocol. In fact, we showed that under a "too liberal" definition of SFE (which includes all SFE definitions other than Micali-Rogaway), it could be that *no interleaving* I will result in a secure protocol F^I. The stringent definition of Micali-Rogaway (in particular, the existence of a committal round) not only shows that such an interleaving *must* exist, but also allows us to define a rich class of interleavings which guarantee the security of F^I: the only thing we require is that all the "pre-committal" rounds precede all the "post-committal" rounds. In other words, players should first "declare" all their inputs to g^i's, and only then proceed with the "actual computation" of any of the g^i's. The intuition behind this restriction is clear: this is exactly what happens in the semi-ideal model when players simultaneously evaluate $g^1, \ldots, g^k$ in F.

Remark 1: In the parallel-reducibility theorem we do not allow the adversary choose the interleaving I adaptively in the process of the computation. This is only done for simplicity. For example, synchronous reducibility will hold provided the adversary is restricted to select a synchronous interleaving I. And concurrent reducibility holds if the semi-ideal protocol F remains secure if we allow the semi-ideal adversary adaptively order the ideal calls to $g^1, \ldots, g^k$.

4 Proof of the Parallel-Reducibility Theorem

For economy and clarity of presentation, we shall prove both concurrent and synchronous reducibility "as together as possible". Let S be the simulator for F, let π be the order of committal rounds of the G_i's in the interleaving I (if several committal rounds of G_i's happen in one round, order them arbitrarily), and let S_i be the simulator for G_i. We need to construct the simulator S^I for F^I. The proofs for the concurrent and synchronous reducibility are going to be very similar, the main differences being the following:

- *Concurrent Reducibility*. Since R is an order-independent round of F, the protocol F^π is also secure, i.e. has a simulator S^π. We will use S^π instead of S (together with $S_1 \ldots S_k$) in constructing S^I. In particular, S^π will simulate the ideal call to g^i right after the committal round of G_i, which is exactly the order given by π.
- *Synchronous Reducibility*. Here we must use S itself. In particular, at some point S will have to simulate the *simultaneous* ideal call to $g^1, \ldots, g^k$, and expects to see the inputs of the corrupted players. Since the interleaving I is a synchronous interleaving, it has a synchronization point where all the effective inputs of the corrupted players are defined before any of the G_i's went on "with the rest of the computation." It is at this point where we let S simulate the ideal call, because we will be able to provide S with all the (effective) inputs.

To simplify matters, we can assume without loss of generality that each round of I executes one round of a *single* G_i. Indeed, if we can construct a simulator for any such interleaving, we can do it for any interleaving executing in one round a round of several G_i's: arbitrarily split this round into several rounds executing a single G_i and use the simulator for this new interleaving to simulate the original interleaving.[13]

4.1 The Simulator S^I

As we will see in Section 4.2, the actual proof will construct S^I in k stages, that is, will construct k simulators $S^1, \ldots, S^k$, where S^k will be S^I. However, we present the final S^I right away because it provides a good intuition of why the proof "goes through" (but can be skipped otherwise).

For concreteness, we concentrate on the concurrent reducibility case. As one can expect, S^I simply runs S^π and uses $S_1, \ldots, S_k$ to simulate the interleaving of $G_1, \ldots, G_k$.

[13] Here we use the fact that non-corrupted players execute G_i's independently from each other, so the adversary can only benefit by executing a round of single G_i at a time.

- Run S^π up to round R (can do it since F^I and F^π are the same up to round R).
- Tell each S_i to corrupt all the players already corrupted by the adversary (it is irrelevant what we give to S_i as their inputs).
- Assume we execute some round of protocol G_i in the interleaving I. S^I then uses S_i to produce the needed messages from good-to-bad players and gives back to S_i the response of the adversary.
- Right after the committal round CR_i of G_i has been simulated, use the *effective input function of* G_i and the traffic of the adversary in the simulation of G_i to determine the effective input w_j^i of each corrupted player j to g^i.
- We notice that at this stage S^π is *exactly* waiting to simulate the ideal call to g^i for the adversary. So S^I gives S^π the effective inputs w_j^i as the adversary's inputs to g^i, and learns from S^π the output z_j^i of each corrupted player j.
- We notice that after round CR_i has been simulated, the simulator S_i expects to see the outputs of all the corrupted players from the g^i-oracle that does not exist in our simulation. Instead, we give S_i the values z_j^i that we just learned from S^π.
- We keep running the above simulation up to the end of the interleaving I. We note that at this stage S^π has just finished simulating the ideal calls to all the g^i's, and waits to keep the simulation of F^π starting from round $R+1$. And we just let S^π do it intil the end of F^I (we can do it since F^I and F^π are the same again from this stage).
- It remains to describe how S^I handles the corruption requests of the adversary. This will depend on where in F^I the corruption request happens. But in any case S^I tells S^π that the adversary asked to corrupt player j and learns from S^π the view V_j of j in (the simulation of) F^π.
  - ⋆ If the corruption request happens before round R, simply return V_j to the adversary.
  - ⋆ Otherwise, the adversary expects to see (possibly partial) transcript of j inside every G_i, which V_j does not contain. However, V_j still contains the supposed inputs w_j^i of player j to each g^i.
  - ⋆ For each i we now ask the simulator S_i to corrupt player j in order to learn its view inside G_i. To answer this request, S_i needs help from the g^i-oracle (that does not exist in our simulation), which S^I provides as follows.
    - If the corruption happened before the committal round CR_i, S_i only expects to see the input and the auxiliary input of player j to g^i. We give him w_j^i as the actual input and extract from V_j the view of j prior to round R as j's auxiliary input.
    - If the corruption happened after round CR_i,[14] S_i also expects to see the output z_j^i of player j in g^i. However, in this case such an output is also contained in V_j, since right after the (already elapsed) round CR_i, we have simulated the ideal call to g^i in F^π. Thus, z_j^i is part of j's view in F^π, and as such should be included by S^π in V_j.
  - ⋆ We see that in any of the above two cases we can provide S_i with the information it expects. Therefore, we get back the view W_j^i of j in G_i so far.
  - ⋆ S^I now simply combines V_j with $W_j^1, \ldots, W_j^k$ to get the final simulated view of j, and gives it back to the adversary (we will argue later that the security of the G_i's implies that these views "match").

We remark that the simulator for synchronous reducibility is very similar. We essentially need to replace S^π by S and let S simulate the single ideal call to $g^1, \ldots, g^k$ at the synchronization point of I, when the traffic

[14]This includes the case when the corruption happened "after the end" of G_i. We treat this corruption as having the adversary corrupt player j at the very end of the computation of G_i. This kind of "post-executuion" corruption has caused a lot of problems preventing some other SFE notions to satisfy reducibility. In our situation, this case presents no special problems due to the universality of the simulator and the information-theoretic security.

of the adversary will simultaneously give S the (effective) inputs of the corrupted players to all the g^i's.

4.2 Proof Outline

While we have already constructed the simulator S^I, in the proof we will need to use the security of some particular G_i. Therefore, we will need "to move slowly" from the assumed secure protocol F or F^π (evaluating all $g^1, \ldots, g^k$ ideally) to the protocol F^I (whose security we need to establish and which runs k real protocols $G_1, \ldots, G_k$). Roughly, we need to "eliminate" one ideal call (to some g^i) at a time, by "replacing" it with the protocol G_i. Using the security of G_i, we will then argue that this "substitution" still leaves the resulting protocol a SFE protocol for f. To make the above idea more precise, we need some notation.[15]

First, from the interleaving I of $G_1, \ldots, G_k$, we define the "projection interleaving" I^i (for each $i \leq k$). This is the interleaving of the protocols $G_1, \ldots, G_i$ intermixed with the ideal calls to $g^{i+1}, \ldots, g^k$. More precisely, we remove from I the rounds of all G_ℓ for $\ell > i$. For concurrent reducibility, we add the ideal calls to g^ℓ (for every $\ell > i$) right after the place where we previously had the committal round of G_ℓ. We notice that this order of the ideal calls is consistent with the permutation π. In particular, we will identify the "base" interleaving I^0 of $g^1, \ldots, g^k$ with the permutation π. For synchronous reducibility, we add a *single* ideal call to $g^{i+1}, \ldots, g^k$ right at the *synchronization point* of I, and still call the resulting interleaving I^i of $G_1, \ldots, G_i, g^{i+1}, \ldots, g^k$ a synchronous interleaving. Notice that I^{i-1} is also a "projection" of I^i.

Slightly abusing the notation, we now define (in a straighforward way) "intermediate" semi-ideal protocols $F^i = F^{I^i}$, which essentially replace the ideal calls to $g^1, \ldots, g^i$ with $G_1, \ldots, G_i$ (but leave the ideal calls to $g^{i+1}, \ldots, g^k$). We note that $F^k = F^I$ and F^0 is either F^π (the concurrent case) or F (the synchronous case). We know by the assumption of the Theorem that F^0 is secure, and need to show that F^k is secure. Naturally, we show it by induction by showing that the security of F^{i-1} implies that of F^i. Not surprisingly, this will follow from the security of G_i.

To summarize, the only thing we need to establish is the following. Assume F^{i-1} is a SFE protocol for f with the simulator S^{i-1}. We need to construct a simulator S^i for F^i such that for all inputs of the players and for any adversary A^i in F^i, we get $\mathit{History}(A^i, F^i) \equiv \mathit{History}(A^i, S^i)$. We construct S^i from S^{i-1} and the simulator S_i for G_i. Essentially, S^i will run S^{i-1} in F^i and use S_i (together with S^{i-1}'s simulation of the ideal call to g^i) to answer the adversary inside G_i. In the "other direction", given adversary A^i in F^i, we define the adversary A^{i-1} in F^{i-1}. This adversary will run A^i in F^{i-1}, and will also use S_i (together with the ideal call to g^i in F^{i-1}) to interact with A^i inside G_i. Informally, we will say that "$S^i = S^{i-1} + S_i$" and "$A^{i-1} = A^i + S_i$".

The assumed security of F^{i-1} implies that $\mathit{History}(A^{i-1}, F^{i-1}) \equiv \mathit{History}(A^{i-1}, S^{i-1})$. Since A^{i-1} essentially runs A^i, the history of A^{i-1} in F^{i-1} will naturally "contain" (we define it precisely later) the history of A^i run against F^{i-1} and the simulator S_i. We denote this history by $\mathit{History}(A^i, F^{i-1} + S_i)$. Then the above equality of histories, combined with the definition of $S^i = S^{i-1} + S_i$, will immediately imply that $\mathit{History}(A^i, F^{i-1} + S_i) \equiv \mathit{History}(A^i, S^i)$. What will remain to show is that $\mathit{History}(A^i, F^i) \equiv \mathit{History}(A^i, F^{i-1} + S_i)$. We remark that the "environments" F^i and $F^{i-1} + S_i$ are identical except the former runs the actual protocol G_i, while the latter evaluates g^i ideally and uses the simulator S_i to deal with A^i inside G_i. Not surprisingly, the last equality (whose verification is the main technical aspect of the proof) will follow from the security of G_i. Namely, assuming that the last equality is false, we will construct an adversary A_i for G_i such that $\mathit{History}(A_i, G_i) \not\equiv \mathit{History}(A_i, S_i)$, a contradiction. Roughly, A_i will simulate the whole network of players in F^i (both the adversary A^i and the honest players!), except when executing G_i.

This completes a brief outline of the proof. The full proof can be found in the Appendix.

[15]Below, we will try to use superscripts when talking about notions related to computing f, like F^i, S^i, A^i. And we will use subscripts for notions related to computing some g^i, like G_i, S_i, A_i.

4.3 The Definitional Support of Parallel Reducibility

Since at least synchronous reducibility provably does not hold for other SFE definitions, one may wonder what specific features of the definition of [22] are "responsible" for parallel reducibility. While such key features can be properly appreciated only from the full proof of the parallel-reducibility theorem, we can already informally highlight two such features on the basis of the above proof outline.

On-line Simulatability: The simulator S not only is universal (i.e., independent of the adversary A) and not only interacts with A in a black-box manner, but must also interact with A "on-line". In other words, S runs with A *only once*: each time that S sends a piece of information to A, this piece becomes part of A's *final view*. This is in contrast with traditional simulators, which would be allowed to interact with A arbitrarily many times, to "rewind" A in the middle of an execution, and to produce any string they want as A's entire view.

The ability to generate A's final view on-line is probably the most crucial for achieveing any kind of parallel reducibility. For example, an adversary A of the composed protocol might base it actions in sub-protocol G_1 depending on what it sees in sub-protocol G_2 and vice versa. Therefore, the resulting views of A inside G_1 and G_2 are very *inter-dependent*. It thus appears crucial that, in order to simulate these inter-dependent views, the simulator S_i for G_i should be capable of extending A's view inside G_i incrementally "in small pieces" (as it happens with A's view in the real execution) that should never "be taken back". If, instead, one were only guaranteed that he could simulate the *entire* (as opposed to "piece-by-piece") view of A in each G_i separately, there is no reason to expect that these two separate views would be as inter-dependent as A can make them in the real model. As demonstrated in Section 4.1, on the other hand, having on-line "one-pass" simulation makes it very easy to define the needed on-line simulator for A.

Committal Rounds: Intuitively, the committal round corresponds to the "synchronization point" in the ideal function evaluation: when all the players have sent their inputs to the trusted party, but have not received their corresponding outputs yet. Not surprisingly, the notion of the committal round plays such a crucial role in synchronous reducibility. In particular, the very existence of "good" interleavings (i.e., synchronous interleaving, as stated in Theorem 2) is based on the committal rounds. Committal rounds also play a crucial role in Corollary 2. Indeed, the greedy concurrent execution of all the "pre-committal" rounds of any number of sub-protocols $G_1, \ldots, G_k$ (which takes at most $\max(R_1, \ldots, R_k)$ rounds), followed by the greedy concurrent execution of all the "post-committal" rounds of $G_1, \ldots, G_k$ (which also takes at most $\max(R_1, \ldots, R_k)$ rounds), yields a *synchronous interleaving* of $G_1, \ldots, G_k$ with the claimed number of rounds.

The Price of Parallel Reducibility. The definitional support of parallel reducibility "comes at a price": it rules out some reasonable protocols from being called secure. For example, having P_1 simply send x_1 to P_2 is not a secure protocol (in the sense of [22]) for the function $g^1(x_1, \lambda, \lambda, \ldots, \lambda) = (x_1, x_1, \lambda, \ldots, \lambda)$ of Example 2. Indeed, assume adversary A corrupts player P_2 before the protocol starts and does not corrupt anyone else later on. Then A will learn x_1 in the real execution. Therefore, for the simulator S to match the view of A, it must also send x_1 to A in round 1. For doing so, S must learn x_1 from its oracle *before round* 1. Since A does not corrut player 1, this can only happen when S learns the output of corrupted player P_2 (which is indeed x_1) after the committal round. Unfortunately, the committal round *is* round 1 itself, because only then does P_1 manifest its input x_1 via its own message traffic. Thus, S will learn x_1 only *after round* 1, which is too late.

In sum, a reasonable protocol for function g^1 is excluded by the definition of [22] from being secure, but this "price" has a reason: Example 2 proves that such (individually) reasonable protocol is not synchronously reducible.

References

[1] D. Beaver, Foundations of Secure interactive Computing. *Proc. of CRYPTO'91*, pp. 377-391, 1991.

[2] D. Beaver, Secure multi-party protocols and zero-knowledge proof systems tolerating a faulty majority. *Journal of Cryptology*, 4(2), pp. 75–122, 1991.

[3] D. Beaver and S. Goldwasser, Multi-parti computation with faulty majority, *Proc. of the 30th FOCS*, pp. 468–473, 1989.

[4] M. Bellare and R. Canetti and H. Krawczyk, A modular approach to the design and analysis of authentication and key-exchange protocols, *Proc. of the 30th STOC*, 1998.

[5] M. Ben-Or, S. Goldwasser and A. Wigderson, Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation, *Proc. of the 20th STOC*, pp. 1–10, 1998.

[6] M. Ben-Or, R. Canetti and O. Goldreich, Asynchronous Secure Computatuions. *Proc. of the 25th STOC*, pp. 52–61, 1993.

[7] R. Canetti, Security and Composition of Multi-parti Cryptographic Protocols. *Journal of Cryptology*, to appear. On-line version available at *http://philby.ucsd.edu/cryptolib/1998/98-18.html*.

[8] R. Canetti, Studies in Secure Multi-party Computatuion and Application, *Ph.D. Thesis*, Weizmann Institute, Israel, 1995.

[9] R. Canetti, U. Feige, O. Goldreich and M. Naor, Adaptively Secure Computation, *Proc. of the 28th STOC*, pp. 639–648, 1996.

[10] R. Canetti and R. Gennaro, Incoercible multi-party computation, *Proc. of the 37th FOCS*, pp. 504–513, 1996.

[11] R. Canetti, E. Kushilevitz, R. Ostrovsky and A. Rosen, Randomness vs. Fault-Tolerance, *16th PODC*, pp. 35–44, 1997.

[12] R. Canetti and R. Ostrovsky, Secure Computation with honest-Looking Parties: What if nobody is truly honest?, *Proc. of STOC*, pp. 35–44, 1999.

[13] D. Chaum, C. Crépeau and I. Damgård, Multiparty unconditionally secure protocols, *Proc. of the 20th STOC*, pp. 11–19, 1988.

[14] R. Cramer, U. Maurer, and I. Damgård, General secure multiparty computation from any linear secret-sharing scheme, *Proc. EUROCRYPT'00*, to appear, 2000.

[15] P. Feldman and S. Micali, Optimal algorithms for Byzantine agreement, *SIAM J. on Computing*, 26(4):873-933, 1997.

[16] S. Goldwasser and L. Levin, Fair computation of general functions in presence of immoral majority, *Proc. CRYPTO '90*, pp. 75–84, 1990.

[17] S. Goldwasser, S. Micali, and C. Rackoff, The knowledge complexity of interactive proof systems, *SIAM Journal on Computing*, 18(2), pp. 186–208, 1989.

[18] O. Goldreich, Foundations of Cryptography (Fragments of a book), Available at *http://theory.lcs.mit.edu/~oded/foc.html*.

[19] O. Goldreich, Secure Multi-Party Computation, Fisrt draft available at *http://theory.lcs.mit.edu/~oded*.

[20] O. Goldreich, S. Micali and A. Wigderson, How to play any mental game, *Proc. of the 19th STOC*, pp. 218–229, 1987.

[21] K. Kilian, E. Kushilevitz, S. Micali and R. Ostrovsky, Reducibility and Completeness in Private Computatuions, To appear in SIAM J. on Computing, preliminary versions in *Proc. of the 23rd STOC*, 1991 by Kilian and in *Proc. of the 35th FOCS*, 1994 by Kushilevits, Micali and Ostrovsky.

[22] S. Micali and P. Rogaway, Secure computation, *Proc. CRYPTO '91*, pp. 392–404, 1991. Also in Workshop On Multi-Party Secure Computation, Weizman Institute, Israel, 1998.

[23] T. Rabin and M. Ben-Or, Verifyable Secret Sharing and Multi-party Protocols with Honest Majority, *Proc. of 21st STOC*, pp. 75–83, 1989.

[24] A. Yao, Protocols for secure computation, *Proc. of the 23rd FOCS*, pp. 160–164, 1982.

[25] A. Yao, How to generate and exchange secrets, *Proc. of the 27th FOCS*, pp. 162–186, 1986.

Full Proof of the Parallel-Reducibility Theorem

Here we give a full proof of the Paralle-Reducibility Theorem following the outline given in Section 4.2. Recall that the only thing we had to prove was the following. Assume F^{i-1} is a SFE protocol for f with the simulator S^{i-1}. We need to show that F^i is a SFE protocol for f as well. That is, we need to construct a simulator S^i for F^i such that for all inputs of the players and for any adversary A^i in F^i, we get $History(A^i, F^i) \equiv History(A^i, S^i)$. For concreteness, we concentrate on the concurrent reducibility case. With all the previous discussion, the proof for synchronous reducibility can be easily traced as well.

Simulator S^i: We construct S^i from S^{i-1} and the simulator S_i for G_i. Essentially, S^i will run S^{i-1} in F^i and use S_i (together with S^{i-1}'s simulation of the ideal call to g^i) to answer the adversary inside G_i. Informally, "$S^i = S^{i-1} + S_i$".

- Run S^{i-1} up to round R (can do it since F^{i-1} and F^i are the same up to round R).
- Tell S_i to corrupt all the players already corrupted by the adversary (it is irrelevant what we give to S_i as their inputs).
- Unless in the interleaving I^i we execute a round of G_i (which we do not have in I^{i-1}), still use S^{i-1} to answer the adversary (this includes a round of G_ℓ for $\ell < i$, or the ideal call to g^ℓ for $\ell > i$).
- If we execute a round of G_i in I^i, use S_i to answer.
- Right after the committal round CR_i of G_i has been simulated, use the *effective input function of* G_i and the traffic of the adversary in the simulation of G_i to determine the effective input w_j of each corrupted player j to g^i.
- We notice from the definition of the interleaving I^{i-1} as a "projection" of the interleaving I^i, that at this stage S^{i-1} is *exactly* waiting to simulate the ideal call to g^i for the adversary. So S^i gives S^{i-1} the effective inputs w_j as the adversary's inputs to g^i, and learns from S^{i-1} the output z_j of each corrupted player j.
- We notice that after round CR_i has been simulated, the simulator S_i expects to see the outputs of all the corrupted players from the g^i-oracle that does not exist in our simulation. Instead, S^i gives S_i the values z_j that it just learned from S^{i-1}.

- We keep running the above simulation up to the end of the interleaving I^i. At this stage, we simply run S^{i-1} (who just finished the simulation of I^{i-1}) until the end of F^i (we can do it since F^i and F^{i-1} are the same again from this stage).
- It remains to describe how S^i handles the corruption requests of the adversary. This will depend on where in F^i the corruption request happens. But in any case S^i tells S^{i-1} that the adversary asked to corrupt player j and learns from S^{i-1} the view V_j of j in (the simulation of) F^{i-1}.
  - ⋆ If the corruption request happens before round R, simply return V_j to the adversary.
  - ⋆ Otherwise, the adversary expects to see (possibly partial) transcript of j inside G_i, which V_j does not contain. However, V_j still contains the supposed inputs w_j of player j to g^i.
  - ⋆ S^i asks the simulator S_i to corrupt player j in order to learn its view inside G_i. To answer this request, S_i needs help from the g^i-oracle (that does not exist in our simulation), which S^i provides as follows.
    - If the corruption happened before the committal round CR_i of G_i, S_i only expects to see the input and the auxiliary input of player j to g^i. We give him w_j as the actual input and extract from V_j the view of j prior to round R as j's auxiliary input.
    - If the corruption happened after round CR_i (including the case when it happened after "the end" of G_i), S_i also expects to see the output z_j of player j in g^i. However, in this case such an output is also contained in the V_j, since right after the (already elapsed) round CR_i, we have simulated the ideal call to g^i in F^{i-1}. Thus, z_j is part of j's view in F^{i-1}, and as such should be included by S^{i-1} in V_j.
  - ⋆ We see that in any of the above two cases we can provide S_i with the information it expects. Therefore, S^i gets back the view W_j of j in G_i so far.
  - ⋆ S^i now simply combines V_j with W_j to get the final simulated view of j, and gives it back to the adversary (we will argue later that the security of G_i implies that these views "match").

Now assume we are given any adversary A^i for F^i. In order to argue that $\mathit{History}(A^i, F^i) \equiv \mathit{History}(A^i, S^i)$, we need to define a corresponding adversary A^{i-1} in F^{i-1}.

Adversary A^{i-1}: This adversary will run A^i in F^{i-1}, and will also use S_i (together with the ideal call to g^i in F^{i-1}) to interact with A^i inside G_i. Informally, we will say that "$A^{i-1} = A^i + S_i$". Not surprisingly, the description of A^{i-1} is almost word-for-word the description of the simulator S^i, but "turned the other way around".

- Run A^i up to round R in F^{i-1} (can do it since F^{i-1} and F^i are the same up to round R).
- Tell S_i to corrupt all the players already corrupted by the A^i (it is irrelevant what we give to S_i as their inputs).
- Unless in the interleaving I^i we execute a round of G_i (which we do not have in I^{i-1}), still run A^i in F^{i-1} (this includes a round of G_ℓ for $\ell < i$, or the ideal call to g^ℓ for $\ell > i$).
- If we execute a round of G_i in I^i, use S_i to answer to A^i, but do nothing in F^{i-1}.
- Right after the committal round CR_i of G_i has been simulated, use the *effective input function of* G_i and the traffic of A^i in the simulation of G_i to determine the effective input w_j of each corrupted player j to g^i.
- We notice from the definition of the interleaving I^{i-1} as a "projection" of the interleaving I^i, that at this stage the protocol F^{i-1} is just about to execute the ideal call to g^i and waits for A^{i-1} to provide the inputs of the corrupted players. So A^{i-1} provides the effective inputs w_j it just extracted from the trafic of A^i, and learns the output z_j of each corrupted player j.

- We notice that after round CR_i has been simulated, the simulator S_i expects to see the outputs of all the corrupted players from the g^i-oracle. Instead, A^{i-1} gives S_i the values z_j that it just learned from the ideal call to g^i.
- We keep running the above simulation up to the end of the interleaving I^i. At this stage, we simply run A^i in F^{i-1} until the end of the protocol (we can do it since F^i and F^{i-1} are the same again from this stage).
- It remains to describe how A^{i-1} handles the corruption requests of A^i. This will depend on where in (the simulation of) F^i the corruption request happens. But in any case A^{i-1} corrupts the corresponding player j in F^{i-1} and learns the view V_j of j.
  - ⋆ If the corruption request happens before round R, simply return V_j to A^i.
  - ⋆ Otherwise, A^i expects to see (possibly partial) transcript of j inside G_i, which V_j does not contain. However, V_j still contains the supposed inputs w_j of player j to each g^i.
  - ⋆ A^{i-1} asks the simulator S_i to corrupt player j in order to learn its view inside G_i. To answer this request, S_i needs help from the g^i-oracle, which A^{i-1} provides as follows.
    - If the corruption happened before the committal round CR_i, S_i only expects to see the input and the auxiliary input of player j to g^i. We give him w_j as the actual input and extract from V_j the view of j prior to round R as j's auxiliary input.
    - If the corruption happened after round CR_i, S_i also expects to see the output z_j of player j in g^i. However, in this case such an output is also contained in the V_j, since right after the (already elapsed) round CR_i, we have made the ideal call to g^i in F^{i-1}. Thus, z_j is part of j's view V_j in F^{i-1}, and can be provided to S_i as well.
  - ⋆ We see that in any of the above two cases we can provide S_i with the information it expects. Therefore, A^{i-1} gets back the view W_j of j in G_i so far.
  - ⋆ A^{i-1} now simply combines V_j with W_j to get the final simulated view of j, and gives it back to A^i (we will argue later that the security of the G_i's implies that these views "match").

Equality of Distributions: From the security of F^{i-1}, we know that

$$History(A^{i-1}, F^{i-1}) \equiv History(A^{i-1}, S^{i-1}) \tag{2}$$

which is the same as

$$\langle View(A^{i-1}, F^{i-1}),\ \hat{\mathbf{x}}^{F^{i-1}},\ \hat{\mathbf{y}}^{F^{i-1}} \rangle \equiv \langle View(A^{i-1}, S^{i-1}),\ \hat{\mathbf{x}}^{S^{i-1}},\ \hat{\mathbf{y}}^{S^{i-1}} \rangle \tag{3}$$

We notice that the view of A^{i-1} (both against F^{i-1} and S^{i-1}) actually contains the view of the adversary A^i that A^{i-1} was running in the background. We denote these views by $View(A^i, F^{i-1} + S_i)$ and $View(A^i, S^{i-1} + S_i)$, and let

$$History(A^i, F^{i-1} + S_i) \stackrel{\text{def}}{=} \langle View(A^i, F^{i-1} + S_i),\ \hat{\mathbf{x}}^{F^{i-1}},\ \hat{\mathbf{y}}^{F^{i-1}} \rangle \tag{4}$$

$$History(A^i, S^{i-1} + S_i) \stackrel{\text{def}}{=} \langle View(A^i, S^{i-1} + S_i),\ \hat{\mathbf{x}}^{S^{i-1}},\ \hat{\mathbf{y}}^{S^{i-1}} \rangle \tag{5}$$

Thus, Equation (2) (i.e., assumed security of F^{i-1}) implies that

$$History(A^i, F^{i-1} + S_i) \equiv History(A^i, S^{i-1} + S_i) \tag{6}$$

However, from the definition of S^i and the definitions of the effective inputs/outputs of F^i based on those of F^{i-1}, we observe that the latter distribution is *syntactically the same* as $History(A^i, S^i)$! That is,

$$History(A^i, S^{i-1} + S_i) \equiv History(A^i, S^i) \tag{7}$$

Therefore, Equation (6) and Equation (7) imply that what remains to prove is that

$$History(A^i, F^i) \equiv History(A^i, F^{i-1} + S_i) \tag{8}$$

The Last Piece: We finally show Equation (8). We remark that the "environments" F^i and $F^{i-1}+S_i$ are identical except the former runs the actual protocol G_i, while the latter evaluates g^i ideally and uses the simulator S_i to deal with A^i inside G_i. We call the first experiment the "real" experiment and the second – the "simulated" experiment. Assume that Equation (8) it is false for some input configuration $\chi = \langle \mathbf{x}, \mathbf{a}, \alpha \rangle$. Let $H(G_i) = \mathit{History}(A^i, F^i)$ and $H(S_i) = \mathit{History}(A^i, F^{i-1}+S_i)$ on the configuration χ. Thus, $H(G_i) \not\equiv H(S_i)$.

We notice that the overall randomness generating the histories of the real and the simulated experiments is identical except the real experiment uses the coins C of the honest players inside G_i (which do not depend on anything else as players are supposed to use brand new randomness inside a sub-routine), while the simulated experiment uses the randomness of the simulator S_i and the g^i-oracle executing the ideal call in F^{i-1} (which again do not depend on anything else; call them D). Since $H(G_i) \not\equiv H(S_i)$, there exists a particular setting γ of all the other randomness except for C and D (this includes the randomness of A^i, of all the honest players everywhere but in G_i, all the trusted parties for g^ℓ where $\ell > i$) such that $H(G_i \mid \gamma) \not\equiv H(S_i \mid \gamma)$.

We let $\alpha' = \langle \chi, \gamma \rangle$ be the auxiliary string of the adversary A_i for G_i that we will construct. We notice that α' determines the entire (identical) state of the real and simulated experiments up to round R; in particular, set B of players currently corrupted by A^i, and fixed inputs $\mathbf{w}$ and auxiliary inputs $\mathbf{b}$ of all currently honest players to g^i. Since A_i will immediately corrupt players in B and ignore their inputs, their inputs to g^i will not be relevant to get the contradiction, so the initial configuration for G_i where A_i will successfully run can be thought as $\langle \mathbf{w}, \mathbf{b}, \alpha' \rangle$.

Here is the description of A_i for G_i. As we said, it starts from corrupting players in B and ignoring their inputs. Then it simply keeps running A^i against the entire network of honest players in F^i (i.e, simulating *both*, which A_i can do because it has χ and γ) *except* for the run of A^i inside G_i, where A_i actually uses the network available to him. When the running of A^i inside G_i is completed, A_i knows the view of A^i inside G_i, and it also simulated completely in its mind the run of A^i in the interleaving I^i. Now A_i wants to continue running in its mind the interaction of A^i with the honest players for the rest of F^i. For that, it needs to know *the outputs of honest players in G_i*. To "get them", A_i samples uniformly a *consistent* randomness C of honest players inside G_i that would have produced the view that the adversary A^i got inside G_i (note, this step is not polynomial time, but we do not care). Here we use the fact that C are supposed to be brand new random coins independent of everything else, and never used by honest players upon the termination of G_i. Having sampled C, A_i can simply *deterministically* finish the run of A^i (as it knows γ, C and χ). Having done so, A_i stops.

We see that embedded in the view of A_i is the view of A^i that A_i ran in the background. We notice that when A_i interacts with the real network G_i, this view of A^i, and in fact the entire "history" of this run of A^i (its view we got from A_i and the effective inputs and outputs of F^i of all the players, assuming honest players used randomness C inside G_i), is *identically the same* as $\mathit{History}(A^i, F^i \mid \gamma) = H(G_i \mid \gamma)$. Indeed, it does not matter if honest players sampled C from the beginning at random and used it, or that we let honest players sample random C', got the history of A^i, sampled *random* C consistent with this history, and pretend the honest players actually used C.

Now assume that we run A_i against S_i. Up to the completion of the interleaving, the entire "history" of the run of A^i we got from A_i is *syntactically* the same that when we run it against $F^{i-1}+S_i$. However, when we finish the interleaving, a tricky thing happens. In the first case, we interpret the run of A^i against S_i as if honest players executed G_i, and sample random consistent randomness C of honest players. In the second case, we just give players their effective outputs from the trusted party, and generate the actual randomness C_j of player j using S_i, but only if A^i corrupts j later. If we argue that the latter two processes are indeed identical (i.e. it is OK to sample random consistent C when A_i is run against S_i), we would be done obtaining a contradiction. We need to use a somewhat elaborate argument for that, which we semi-informally sketch.

We emphasize again the 3 experiments that we need to compare:

1. Run of A^i against F^i.
2. Run of A^i against F^{i-1}, where we let S_i simulate G_i, then pick a random consistent C and run the "resulting" F^i until completion.
3. Run of A^i against $F^{i-1} + S_i$, where we let S_i simulate G_i, as well as generate the randomness C_j of player j corrupted by A^i after the end of the interleaving.

We know from the security of G_i that the Experiments 1. and 2. are identical "all the way" (if not, we are done getting a contradiction, as they correspond to the runs of A_i against G_i and S_i). We also know that Experiments 2. and 3. are *syntactically* the same up to the end of the interleaving I^i. We assumed that Experiment 1. and 3. are "different" (in their entirety). To still get a contradiction we show by extending the argument "one-round-at-a-time" that Experiments 2. and 3. must be identical "all the way" as well. For that we will use the *universality of the simulator* S_i, i.e. that it "does not know" which adversary it is talking to.

Assume we established up to round $\tilde{R}$ that Experiments 2. and 3. are the same. The starting $\tilde{R}$ is the end of the interleaving, where we know this is the case. We also know from this, that the effective outputs of honest players in G_i are distributed the same in Experiments 2. and 3.

If in round $(\tilde{R}+1)$ the adversary A^i does not corrupt any player, we are done, since honest players do not use their randomness C_j they used inside G_i, only their inputs and outputs, which we know are distributed the same. The only problem is when A^i corrupts a player. In Experiment 2. we return the (consistent) value C_j that we sampled at the end of the interleaving. In Experiment 3. we let the simulator S_i generate this randomness. However, we still argue that these two answers are distributed in the same way (conditioned on what happened before). In particular, assume A^i so far has corrupted players $j_1, \ldots, j_p$ after the end of the interleaving (so that $j_p = j$). A^i could base its decisions to corrupt these players on some powerful information it extracted since the end of the interleaving.

However, the simulator S_i is universal and has to answer in the same way no matter why A^i asked to corrupt these players. In particular, there exists an adversary A_i' that does the same thing as A_i up to the end of the run of A^i inside the interleaving, and then for "no specific reason" asks S_i to corrupt the same players $j_1, \ldots, j_p$. Since S_i cannot distinguish between these two cases (it only sees the requests of whom to corrupt), its responses must be the same as well. But when the adversary asks to corrupt these players "for no reason" right at the end of the interleaving, the security of G_i (against this A_i') implies that the answers $C_{j_1}, \ldots, C_{j_p}$ that S_i gives are distributed *exactly the same* as the true randomness of the actual players conditioned on the view A^i got inside G_i, which are exactly the answers we sampled in Experiment 2.!

This shows that Experiments 2. and 3. are indeed the same, Experiments 1. and 2. are the same, and yet we assumed that Experiments 1. and 3. are different, a contradiction.