From Input Private to Universally Composable Secure Multi-party Computation Primitives

Dan Bogdanov [*] dan@cyber.ee
Peeter Laud[*] peeter@cyber.ee
Sven Laur [†] swen@ut.ee
Pille Pullonen[*†] pille.pullonen@cyber.ee

May 29, 2014

Abstract

Secure multi-party computation systems are commonly built from a small set of primitive components. The composability of security notions has a central role in the analysis of such systems, as it allows us to deduce security properties of complex protocols from the properties of its components. We show that the standard notions of universally composable security are overly restrictive in this context and can lead to protocols with sub-optimal performance. As a remedy, we introduce a weaker notion of privacy that is satisfied by simpler protocols and is preserved by composition. After that we fix a passive security model and show how to convert a private protocol into a universally composable protocol. As a result, we obtain modular security proofs without performance penalties.

1 Introduction

Secure multi-party computation is a tool for privacy-preserving collaborative computation. In general, the desired functionality is represented by either a boolean or arithmetic circuit and jointly evaluated. In this context, it is not sufficient to prove that individual protocols for gate operations leak nothing beyond the desired outputs. We additionally require composability, meaning that these protocols should remain secure, independently of the context in which they are executed. There have been different formalisations of composable security: universal composability [1], reactive simulatability [2], abstract cryptography [3] and inexhaustible interactive Turing machines [4]. In this work, we use reactive simulatability (RSIM) for the asynchronous communication [5, 6] due to the precision it offers us in modelling the adversarial network activities.

In this paper, we define privacy for the RSIM framework and show that our definition is composable. There are big differences between privacy and other standard notions of security, such as sequential and universal composability.

First, privacy protects only the inputs of honest parties. Separately taken, privacy is insufficient for guaranteeing the integrity or confidentiality of protocol outputs. A malicious adversary may control the outputs of a private protocol, while passive corruption might be enough for learning them.

Second, privacy guarantees that the adversary does not learn extra information only if the outputs of the protocol are not used in further computations. A private protocol may leak a significant amount of information about the inputs if all parties publish their outputs. Thus, privacy as an intermediate proof goal is useful in very restricted settings.

Nevertheless, the notion of privacy is indispensable in the analysis of secure multi-party computation protocols based on secret sharing [7, 8, 9] that is the focus of this work. In such protocols, no party obtains the desired output. Instead, the desired output y is secret-shared among all participants. This assures that the outputs of the corrupted parties are independent of y and can be

[*]Cybernetica AS
[†]University of Tartu, Institute of Computer Science

simulated without knowing the outcome. Similarly, we can show that the inputs of such protocols never reach the corrupted parties. As a result, privacy provides sufficient security guarantees as long as the parties do not publish the output shares.

Although our current exposition focuses on computing on secret-shared data, the same approach is applicable for all protocols, which use temporary values to pass the state from one sub-protocol to the other. If these values are deleted after the protocol execution, they do not have to be as secure as the outputs of a protocol. Hence, we can compose less secure versions of sub-protocols as long as we finally convert the obtained private protocol into a secure protocol.

To create a clear distinction between protocol outputs and intermediate values, we define a restricted version of composition where two composed systems can only have one-way communication. This notion is a complement to the previous compositions and helps us in clarifying the data dependency graph of the composition.

As the main result, we show that the ordered composition of a private and a secure protocol is secure in the passive model, provided that all outputs of the composed protocol are produced by the secure protocol, i.e., the outputs of the private protocol are indeed temporary. The exact set of conditions that are needed for achieving similar results in the active model is out of our scope in this paper.

The implications of these results are two-fold. First, private versions of protocols can be more efficient than their secure counterparts. Second, proofs of privacy are often more straightforward and easier for the protocol designer.

We stress that the RSIM framework gives complete control over protocol scheduling to the adversary who controls when the messages are delivered. The latter is an appropriate way for modelling standard deployments of SMC frameworks where the underlying communication network is not under our direct control and secrecy and integrity are provided by cryptographic means. To simplify the analysis, we consider the standard adaptive passive security model from RSIM with secure authenticated channels. Such a model is an obvious choice, as we want to model the cases where some of the parties of the computation are corrupted and not to focus on independent network intruders.

The initial idea for our composition of private and secure protocols as well as the main motivation comes from the Sharemind secure multi-party computation framework [7, 10]. The most up-to-date version of this approach can be found in [11]. However, the current treatment in this paper is more general and more rigorous than the original exposition. This technique has also been used for obtaining other efficient secure multi-party computation protocols that are introduced in Sec. 3.3. Previously, the security of each protocol obtained in this manner had to be proven separately. Our work establishes a framework for analogous protocol compositions that simplifies the proofs.

In the following, Sec. 2 gives an overview of the reactive simulatability framework with a focus on the adaptive adversaries and sets the ground for the rest of the paper. Sec. 3 gives further motivation for our work as well as concrete examples of using private sub-protocols in compositions. Sec. 4 gives a formal definition of privacy and specifies the ideal functionalities for private systems. In Sec. 5, we define an *ordered composition* and discuss the ideal functionalities of composed systems. Our main goal is to show that the fully ordered composition of private and secure systems is secure and that privacy is composable. For these purposes, Sec. 6 explores the necessary modifications of the original simulators required for building simulators for the composed systems. Finally, Sec. 7 proves the main composability results by introducing intermediate models of corruption, which simplify the analysis without losing generality.

2 Reactive Simulatability

This section gives a high-level overview of the RSIM framework and exposes only the details that are necessary to understand our main results. Further information and detailed discussions be found in the original papers [5, 6].

2.1 Interaction model for asynchronous reactive systems

In this model, different parties are described by machines that are connected through ports and each connection is actually a buffer that may be under adversarial control. Reactive means that

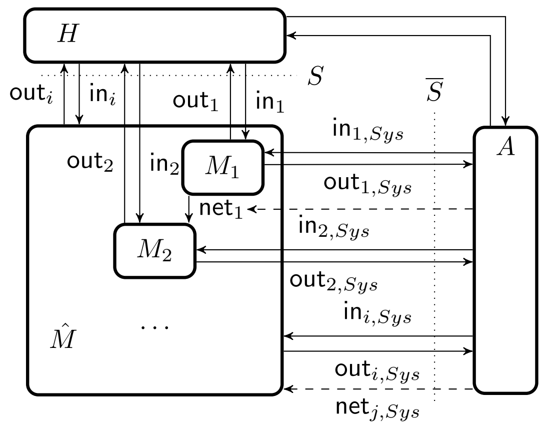

Figure 1: Canonical interface between adversary, honest user and system

the environment and the system can interact multiple times and asynchronous means that the adversary may control delays in the message delivery.

A party in the RSIM framework is modelled as a *machine* specified by tuple $(Name, Ports, States, \sigma, Ini, Fin)$. A string $Name$ is the name of the machine, $Ports$ is a sequence of ports of that machine, $States$ is a set of states, σ is a probabilistic state transition function, Ini and Fin are the sets of initial and final states. We sometimes write $ports(\hat{M})$ to stress that they belong to the set $\hat{M}$ of machines. The set $ports(\hat{M})$ can be further split into the ports that connect machines in the collection $\hat{M}$ and the set of free ports $free(\hat{M})$ that can be accessed from outside.

Data transfer from one machine to another goes through a buffer that connects an output port $p!$ with and input port $p?$. A *buffer* is a dedicated machine that has exactly three ports: clock-in port, buffer in-port and buffer out-port. Each state transition in the system either appends or retrieves an element from a buffer. Ports are named after the buffer they are connected to. A question mark after the port name denotes input ports and the exclamation mark denotes output ports. The shorthand p^c denotes a port that is complementary to p. For a set S of ports, we use S^c in a similar manner. On figures, we use labelled arrows to denote buffers and ports. Buffer clocking channels are denoted as dashed lines where the name corresponds to the name of the buffer.

As an illustrative example, consider the most common setting in the RSIM framework depicted on Fig. 1. In this configuration, a complex system Sys, composed of several subsystems and parties, is under the influence of an adversary A and an environment H, which uses Sys to obtain an output. All machines, except for H, are assumed to have input and output buffers denoted by properly indexed in and out symbols. As the system can be a collection of machines, we use double indexing. In addition, there is a buffer for each network connection j defined for the machines in the system. The double indexing of buffers is necessary for differentiating between analogous buffers in protocol composition. Finally, the adversary A schedules all buffers on figure Fig. 1.

The execution of the machines is controlled by clocking and inputs. A machine is clocked when it receives an input. The corresponding machine uses the state transition function to determine the next state and write data to the output buffers. A machine can clock at most one of the output buffers by sending an input to a clock-in port of the buffer. Upon a clocking input, the buffer releases the desired value and the recipient of the value is clocked. If no machine is scheduled then the control goes to the adversary. The execution ends when the adversary reaches a final state.

A *collection* is a finite set of machines. A collection is closed if the only free port is the *master clock-in port.* Given a closed collection of machines $\hat{M}$ with a master scheduler X, a *run* of this collection is a sequence of steps $(N, c_{\mathsf{in}}, v_{\mathsf{in}}, s, ((c_1, v_1), \ldots, (c_n, v_n)), c_{\mathsf{clk}})$, where N is the name of the machine that made the step after receiving the input v_{in} from an input port $c_{\mathsf{in}}?$, going to state s, writing v_i to output port $c_i!$ and possibly clocking the channel c_{clk}, which may also be $\perp$. For a collection $\hat{M}$, a *completion* $[\hat{M}]$ is $\hat{M}$ together with all of the buffers connected to its ports except the master clock port. The main use of a completion is that often, the collections are defined as sets of simple machines, as these are the main objects to work with. However, occasionally we need to specify that all the buffers are included.

A *structure* $(\hat{M}, S)$ is collection of machines $\hat{M}$ together with a set of ports $S \subseteq free(\hat{M})$. Let $\overline{S} = free(\hat{M}) \backslash S$. Ports in S are for communicating with the environment H and ports in $\overline{S}$ are for communicating with the adversary A. To avoid naming collisions, H cannot have ports belonging to S or ports used to send messages between machines of $\hat{M}$. We denote these *forbidden* ports by $forb(\hat{M}, S)$.

A system Sys is defined as a set of structures. In cryptography, these structures commonly correspond to different sets of statically corrupted parties. Surprisingly enough, adaptive corruption can be modelled by systems with exactly one structure and machines that accept corruption requests during their work. Hence, we simplify the original theory [5, 6] for the cases where $Sys = \{(\hat{M}, S)\}$. Also, note that the model provides a fine-grained way to specify what the adversary can control and influence during the execution of the protocol. By altering the interface $\overline{S}$ we can specify what kind of messages the adversary can delay or read. In our work, the adversary has full control over the protocol and message scheduling, but does not see the messages of uncorrupted parties.

A *configuration of a system* is a tuple $(\hat{M}, S, H, A)$ where H is a simple machine without forbidden ports $forb(\hat{M}, S)$ and the collection $\hat{M}, H, A$ is closed with A being the master scheduler. We use a shorthand $Conf(Sys)$ to denote the set of all configurations.

2.2 Security definitions

Security in the RSIM framework is defined by contrasting two systems Sys_1 and Sys_2 where Sys_2 is the system that is known to be secure and Sys_1 is the system we want to use. Both systems must have the same interface S towards the environment H and there must exist a way to convert a valid adversary A_1 against $(\hat{M}_1, S) \in Sys_1$ to a comparable adversary against $(\hat{M}_2, S) \in Sys_2$.

For each machine M, when it is scheduled, we store the state transition as well as respective input and outputs into the run. A *view* of a set of parties $M_0 \subseteq \hat{M}$ is a subset of a run that corresponds to the steps relating to machines in M_0 as is denoted by $view(M_0)$. Note that the view does not contain port names and therefore we are allowed to rename ports and buffers when composing systems.

Definition 1 (Simulatability)**.** Let systems Sys_1 and Sys_2 be given. We say that Sys_1 is perfectly as secure as Sys_2 (denoted as $Sys_1 \geq_{sec}^{perf} Sys_2$) if, for every configuration $conf_1 = (\hat{M}_1, S, H, A_1) \in Conf(Sys_1)$ where $ports(H) \cap forb(\hat{M}_2, S) = \emptyset$, there exists a configuration $conf_2 = (\hat{M}_2, S, H, A_2) \in Conf(Sys_2)$ such that the environments have coinciding views

$$view_{conf_1}(H) = view_{conf_2}(H) \ .$$

We can give analogous definitions for computational and statistical security by restricting the set of plausible configurations and by varying the requirements on the views. For computational security $Sys_1 \geq_{sec}^{poly} Sys_2$, we require computational indistinguishability of views created by all polynomial configurations $Conf(Sys_1)$. For statistical security $Sys_1 \geq_{sec}^{stat} Sys_2$, we require statistical indistinguishability of views for all configurations $Conf(Sys_1)$. The running-time of A_2 must be polynomial in the running-time of A_1.

Black-box simulatability means that $A_2 = \mathsf{Sim} \cup A_1$ must be the *combination* of a simulator machine Sim and the adversary A_1. Two machines are combined simply by taking the Cartesian product of their sets of states and the union of their ports and transition functions. The *simulator* Sim depends only on $\hat{M}$ and S.

Definition 2 (Simulatability for a class of adversaries)**.** Let systems Sys_1 and Sys_2 be given. We say that Sys_1 is perfectly as secure as Sys_2 for class $\mathcal{A}$ adversaries (denoted as $Sys_1 \geq_{sec}^{perf,\mathcal{A}} Sys_2$) if, for every configuration $conf_1 = (\hat{M}_1, S, H, A_1) \in Conf(Sys_1)$ where $A_1 \in \mathcal{A}$ and $ports(H) \cap forb(\hat{M}_2, S) = \emptyset$, there exists a configuration $conf_2 = (\hat{M}_2, S, H, A_2) \in Conf(Sys_2)$ with $A_2 \in \mathcal{A}$ such that $view_{conf_1}(H) = view_{conf_2}(H)$.

2.3 Security of compositions

The main result of the RSIM framework assures that the security is preserved under the composition. A composition of systems in this setting means that the composed systems are connected using the respective ports.

Definition 3 (Composability and composition)**.** Structures $(\hat{M}_1, S_1), \ldots, (\hat{M}_n, S_n)$ are *composable* if they have compatible port layouts:

$$\forall i \neq j : \ \ ports(\hat{M}_i) \cap forb(\hat{M}_j, S_j) = \emptyset \ ,$$
$$\forall i \neq j : \ \ S_i \cap free([\hat{M}_j])^c = S_j^c \cap free([\hat{M}_i]) \ .$$

Their *composition* $(\hat{M}_1, S_1)|| \ldots ||(\hat{M}_n, S_n)$ is a structure consisting of all machines $\hat{M} = \hat{M}_1 \cup \ldots \cup \hat{M}_n$ that has the interface $(S_1 \cup \ldots \cup S_n) \cap free([\hat{M}])$ for communicating with the environment.

For any systems Sys_1 and Sys_2, let $Sys_1 \circ Sys_2$ denote the system of all valid compositions $(\hat{M}_1, S_1)||(\hat{M}_2, S_2)$ for $(\hat{M}_1, S_1) \in Sys_1$ and $(\hat{M}_2, S_2) \in Sys_2$. Let $(\hat{M}_2, S_2) \in Sys_2$ be composable with Sys_1 if there exists $(\hat{M}_1, S_1) \in Sys_1$ such that $(\hat{M}_1, S_1)||(\hat{M}_2, S_2) \in Sys_1 \circ Sys_2$. Then we can state the following profound result.

Theorem 1 (Secure two-system composition)**.** *Assume that we have three systems Sys_1, Sys'_1, Sys_2 such that $Sys_1 \geq_{sec} Sys'_1$. If for every structure $(\hat{M}_2, S_2) \in Sys_2$ that is composable with Sys_1 and for every structure $(\hat{M}'_1, S_1) \in Sys'_1$ the composition $(\hat{M}'_1, S_1)||(\hat{M}_2, S_2)$ exists and satisfies $ports(\hat{M}'_1) \cap S_2^c = ports(\hat{M}_1) \cap S_2^c$, then $Sys_1 \circ Sys_2 \geq_{sec} Sys'_1 \circ Sys_2$.*

Note that the theorem holds for statistical, perfect and computational security as well as universal and black-box simulatability. We use these definitions and the main theorem to add the privacy notion to this formalisation. The composition theorem also holds for the restricted classes of simulatability.

2.4 Security proofs for passive security

Proving that a system is secure requires defining a simulator or several simulators that unify the adversaries' views on the ideal and real world. An important implication of unified views is that the simulation output has to agree with the output of the ideal party. An ideal system should provide abstract interfaces, but should also specify all weaknesses or imperfections of the system.

In the passive security model, the adversary must behave as an observer and thus can not modify the behaviour of the corrupted parties. In the context of Fig. 1, the only message A can send on $\mathsf{in}_{i,Sys}$ is $(\mathsf{corrupt})$ to corrupt party i for system Sys. Afterwards, each time the machine M_i is clocked, it writes its view to $\mathsf{out}_{i,Sys}$ where the adversary can see this. Otherwise, the corrupted machines follow the protocol as the honest machines and A can not affect their behaviour.

Hence, the security proof for a passive adversary requires a simulator that can simulate the view of the corrupted parties and the network scheduling. In addition, the joint view of the environment and the adversary has to be consistent. For that, the simulator has to ensure that the view of the corrupted parties is such that it leads to the same output value as defined by the corresponding ideal functionality.

3 Secure multi-party computation

In the following, we consider secure multi-party computation based on secret-shared data. In this setting, all inputs are distributed between the computing parties who collaboratively compute the outputs. The inputs and outputs of the computation can either be plain or secret shared values. A plain value is denoted by x and a shared value by $[\![x]\!]$ where x_i denotes the share of party $\mathcal{CP}_i$. W.l.o.g., we can assume that all protocol inputs $x_1, \ldots, x_k$ are in a secret-shared form and that the protocol produces outputs in the secret-shared form $[\![y_1]\!], \ldots, [\![y_\ell]\!]$. If needed, the shares $[\![y_i]\!]$ can be opened to the party who needs the output y_i. The functionality of the protocol is determined by the function $(y_1, \ldots, y_\ell) = f(x_1, \ldots, x_k)$. As any function f can be expressed as an arithmetic circuit, all computations can be reduced to securely evaluating a few arithmetic primitives.

3.1 Arithmetic black box

The arithmetic black box (ABB), first specified by Damgård and Nielsen [12], is a possible representation for the set of protocols for securely evaluating arithmetic primitives. It is an ideal functionality $\mathcal{F}_{\text{ABB}}$, the internal state of which contains a mapping from (public) *handles* to (secret) values. The computing parties can store new values inside $\mathcal{F}_{\text{ABB}}$ under the handles they have chosen. The functionality can do arithmetic with the values stored in it. If the computing parties specify the operation, the handles of the operands, and the handle of the result, then $\mathcal{F}_{\text{ABB}}$ applies the chosen operation to the values associated with the operand handles, and stores the result of the operation under the result handle. Finally, the computing parties can instruct $\mathcal{F}_{\mathsf{ABB}}$ to publish a value stored under a certain handle. The functionality $\mathcal{F}_{\mathsf{ABB}}$ performs these operations only if a majority of parties give the same command to $\mathcal{F}_{\mathsf{ABB}}$.

The availability of the functionality $\mathcal{F}_{\text{ABB}}$ makes the secure computation of a function f represented as an arithmetic circuit conceptually simple. The computing parties first store their inputs in the ABB. They will then interpret the circuit, requesting the ABB to evaluate each gate in turn. Finally, they request the ABB to publish the outputs.

The intermediate values of the computation are stored inside the ABB, accessible to the computing parties only through abstract handles. The only values received from $\mathcal{F}_{\text{ABB}}$ are the published outputs. Thus, $\mathcal{F}_{\text{ABB}}$ is a monolithic ideal functionality, similar to the UC cryptographic library [13], and contrasting with smaller, "lower-level" composable ideal functionalities (e.g. [14, 15, 16]), where the inputs and outputs of operations, represented as bit-strings, are exposed at the interface of the functionality.

The notion of ABB has proven itself as a highly suitable abstraction of SMC, when building applications [17, 18] and tools [19, 20] for SMC. Its monolithic nature makes it less suitable as a specification for *large* SMC libraries. A monolithic $\mathcal{F}_{\text{ABB}}$ would change each time a new operation is added to the library, requiring the security proof (Def. 1) to be redone from scratch, thereby complicating the maintenance of the library. In this paper, we show how to define lower-level abstract privacy-preserving operations, and how to compose them. These allow for more modular abstractions and security proofs. The high-level abstraction in the form of an ABB can be easily composed from the abstract operations.

3.2 Lightweight functionalities

In practice, ABB systems are built using primitive protocols that process input shares $[\![x_1]\!], \ldots, [\![x_k]\!]$ to get output shares $[\![y]\!]$. The corresponding ideal functionality $[\![y]\!] = f([\![x_1]\!], \ldots, [\![x_k]\!])$ reconstructs $x_1, \ldots, x_k$ from the shares and returns uniformly randomly chosen shares of y. The ideal functionality also accepts corruption requests from the adversary and releases the inputs of the corrupted parties to the adversary. In case of secure protocols also the outputs of the corrupted parties are given to the adversary as common for ideal functionalities. As usual, a protocol is secure, if it is as secure as the corresponding ideal functionality.

The corresponding real functionality defines a machine M_i^f for each party $\mathcal{CP}_i$, based on the protocol description. The machines are connected to each other with secure, authenticated channels, the messages on which the adversary cannot see or modify. Still, the adversary can fix the timings of these channels. Additionally, the machines allow passive corruption and the corrupted machines send all of their views to the adversary. Each machine M_i^f can receive a corruption request on the input port $\mathsf{in}_{i,Sys}?$, and will afterwards output the elements of its view to the port $\mathsf{out}_{i,Sys}!$. The channels $\mathsf{in}_{i,Sys}$ and $\mathsf{out}_{i,Sys}$ are clocked by the adversary.

This setup is a standard way for modelling passive adaptive corruption in the RSIM framework. The same setup can be used to model static corruption, if we additionally require that all corruption request must be submitted before execution starts. All corrupted machines notify the adversary about their outputs when they are computed as $(output, \ell_x)$ where ℓ_x is the label of the output and *output* is a keyword. We assume that the state s recorded in the view of the machine contains all computed outputs as $(output, \ell_x, x)$ for each value x and possibly other parts of the previous protocol run. After receiving the corruption request, the corresponding machine sends its current state s to the adversary. Then, the machine sends its view $(v_{\mathsf{in}}, s, ((c_1, v_1), \ldots, (c_n, v_n)), c_{\mathsf{clk}})$ to the adversary each time it is clocked. The adversary can learn all the previous outputs because they are contained in the state s.

We stress that, according to this definition, addition protocols in most ABB-s are insecure. The employed sharings are usually additively homomorphic [12, 7] and the real functionality just consists of each party's machine adding up the shares by itself. As a result, the protocol does not produce freshly generated uniformly random shares.

Algorithm 1 Resharing protocol $[\![w]\!] \leftarrow \mathsf{Reshare}([\![u]\!])$.

```
Data: Shared value $[\![u]\!]$.
Result: Shared value $[\![w]\!]$ such that $w = u$ and freshly generated shares $w_i$ are uniformly distributed.
  $\mathcal{CP}_1$ generates an uniformly distributed $r_{12} \leftarrow \mathbb{Z}_{2^n}$.
  $\mathcal{CP}_2$ generates an uniformly distributed $r_{23} \leftarrow \mathbb{Z}_{2^n}$.
  $\mathcal{CP}_3$ generates an uniformly distributed $r_{31} \leftarrow \mathbb{Z}_{2^n}$.
  All values $r_{ij}$ are sent from $\mathcal{CP}_i$ to $\mathcal{CP}_j$.
  $\mathcal{CP}_1$ computes $w_1 \leftarrow u_1 + r_{12} - r_{31}$.
  $\mathcal{CP}_2$ computes $w_2 \leftarrow u_2 + r_{23} - r_{12}$.
  $\mathcal{CP}_3$ computes $w_3 \leftarrow u_3 + r_{31} - r_{23}$.
  return $[\![w]\!]$.
```

Algorithm 2 Sharemind protocol for secure multiplication

```
Data: $\mathcal{CP}_1$, $\mathcal{CP}_2$, $\mathcal{CP}_3$ hold shared values $[\![u]\!]$ and $[\![v]\!]$.
Result: $\mathcal{CP}_1$, $\mathcal{CP}_2$, $\mathcal{CP}_3$ hold a shared value $[\![w]\!] = [\![uv]\!]$.
  $[\![u']\!] \leftarrow \mathsf{Reshare}([\![u]\!])$
  $[\![v']\!] \leftarrow \mathsf{Reshare}([\![v]\!])$
  $\mathcal{CP}_1$ sends $u'_1$ and $v'_1$ to $\mathcal{CP}_2$.
  $\mathcal{CP}_2$ sends $u'_2$ and $v'_2$ to $\mathcal{CP}_3$.
  $\mathcal{CP}_3$ sends $u'_3$ and $v'_3$ to $\mathcal{CP}_1$.
  $\mathcal{CP}_1$ computes $w'_1 \leftarrow u'_1v'_1 + u'_1v'_3 + u'_3v'_1$.
  $\mathcal{CP}_2$ computes $w'_2 \leftarrow u'_2v'_2 + u'_2v'_1 + u'_1v'_2$.
  $\mathcal{CP}_3$ computes $w'_3 \leftarrow u'_3v'_3 + u'_3v'_2 + u'_2v'_3$.
  return $[\![w]\!] \leftarrow \mathsf{Reshare}([\![w']\!])$.
```

3.3 Efficient composed protocols

Consider a three-party secure computation that is based on additive secret sharing as in Sharemind. We have three participants $\mathcal{CP}_1, \mathcal{CP}_2, \mathcal{CP}_3$ and each secret value v is distributed as shares v_1, v_2, v_3 where $v = v_1 + v_2 + v_3 \bmod 2^n$. Beside arithmetic operations, we have Reshare() protocol in Algorithm 1 that for a shared input value $[\![v]\!]$ outputs a uniformly random sharing $[\![w]\!]$ such that $w = v$.

Sharemind's secure multiplication protocol [11] is given in Algorithm 2. We can consider this protocol as a composition of Reshare() and other computations. The protocol is passively secure according to Def. 1.

We can consider the multiplication protocol without the last line, so that the output is $[\![w']\!]$. However, in this case it is not secure. But composing it with other protocols may give secure protocols. E.g., it's possible to show that in computing $([\![t]\!] \cdot [\![u]\!]) \cdot [\![v]\!]$, only the second multiplication needs Reshare() at the end. Even more strikingly, when computing the inner product of two vectors, it is sufficient to perform a Reshare() at the very end of the computation, not after each multiplication. The performance gain of this approach is illustrated by the performance of Carter-Wegman hash construction in [18]. Their implementation of the optimised construction proved to be twice as fast as the straightforward composition of secure protocols. An analogous idea is also used for obtaining efficient inner products for Shamir secret sharing in [21]. Moreover, there are specific instantiations of Shamir secret sharing schemes, which allow us to perform up to k multiplication through local operations before shares must be re-randomised. As a result, all functions with a multiplicative depth k can be computed locally before the resharing step must be carried out. This significantly reduces the communication overhead.

In the Sharemind ABB, many complex protocols use multiplication as a primitive sub-protocol. Each such a protocol can be made more efficient by replacing a secure multiplication with a private multiplication. Besides multiplication, other primitive protocols such as share conversion, equality check and comparison also benefit from this approach. Among these, share conversion is most like multiplication, that performs some protocol specific computations and then needs the resharing step. Comparison and equality protocols are even further compositions that perform some private protocols for specific computations and then use other sub-protocols and finish with a resharing step.

4 Input privacy

Our definition of privacy has roots in earlier research. Goldreich, in his seminal book *Foundations of Cryptography* [22], defines privacy as indistinquishability of the party's view and the simulation of this view, whereas the simulation is based on the input and output of that party. However, in the context of secret sharing, we can say that the party has no output before the final result is published and the simulation can rely only on the inputs. For private protocols, we also have to consider the correctness of the protocol corresponding to the ideal functionality.

4.1 Privacy definition

Consider an environment that completely ignores the protocol outcomes. Formally, such an environment H is modelled as a composition of two machines, $H = H' \cup H_\perp$, where only H' is allowed to communicate with A and the system and only $H_\perp$ sees the outputs of the protocols. An illustration of this can be found on Fig. 2 where arrows show the direction of communication and lines illustrate possible two-way communication. A *privacy configuration* is a configuration $conf = (\hat{M}, S, H' \cup H_\perp, A)$.

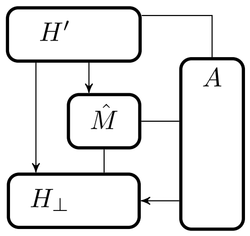

Figure 2: Privacy configuration with two distinct parts for an honest user H

Definition 4 (Input privacy)**.** Let $Sys_1 = (\hat{M}_1, S)$ and $Sys_2 = (\hat{M}_2, S)$ be given. We say that Sys_1 is perfectly at least as input-private as Sys_2 ($Sys_1 \geq_{priv}^{perf} Sys_2$) if, for every privacy configuration $conf_1 = (\hat{M}_1, S, H' \cup H_\perp, A_1) \in Conf(Sys_1)$ where $ports(H) \cap forb(\hat{M}_2, S) = \emptyset$, there exists a privacy configuration $conf_2 = (\hat{M}_2, S, H' \cup H_\perp, A_2) \in Conf(Sys_2)$ with the same $H' \cup H_\perp$, such that the restricted views coincide $view_{conf_1}(H') = view_{conf_2}(H')$.

We can give analogous definitions for computational and statistical privacy by restricting the set of plausible configurations and by varying the requirements on the views. In addition, we can give this definition for restricted classes of adversaries analogously to Def. 2.

We say that a protocol is black-box-private, if A_2 is the combination of a simulator Sim and the adversary A_1, where Sim depends only on $\hat{M}_1$, S and $ports(A_1)$. In the following, we are only interested in the black-box case.

We show later in Thm. 3 that the privacy definition is composable in a sense that a composition of several private systems is as private as the corresponding monolithic ideal functionality. In addition, a trivial composition result about independent composition holds. The composition of two systems Sys_1 and Sys_2 that do not have any connections is private, if both systems are private. Namely, for both of them, we can define a suitable privacy configuration by including the other system in H'.

4.2 Ideal functionality for private protocols

The ideal functionality for a private system is defined analogously to common ideal functionalities, except it does not give outputs to the adversary. This captures the idea, that before the output is published, the SMC protocols have no outputs. A protocol is said to be *private*, if it is as private as the corresponding ideal functionality.

Let us now formally define when a structure can be considered to be an *ideal functionality.*

Definition 5 (Ideal functionality)**.** A structure $\{(\hat{M}, S)\}$ is an *ideal functionality for n parties*, if the following holds.

1. $\hat{M}$ consists of a single machine $\mathcal{I}$.
2. The ports in S have been partitioned to $S^1 \cup \cdots \cup S^n$, where S^i contains the input and output ports for providing the functionality to the i-th party.
3. The machine $\mathcal{I}$ has the ports in S, as well as the ports $\mathsf{in}_I?$ and $\mathsf{out}_I!$ to communicate with the adversary. It clocks the channel out_I. There are no channels from $\mathcal{I}$ to $\mathcal{I}$.
4. $\mathcal{I}$ expects exactly one input at each input port in S (i.e. it will ignore any subsequent inputs). In the course of its work, it will write exactly once to each of the output ports in S. All outputs are produced when all inputs necessary for computing them have been received.
5. On input $(\mathsf{corrupt}, i)$ from $\mathsf{in}_I?$, it will write the inputs it has so far received from the i-th party (from the input ports in S^i) to $\mathsf{out}_I!$ and clock that channel. Subsequently, it forwards any input from the input ports in S^i to $\mathsf{out}_I!$ and clocks that channel.
6. It will not react to any other commands from $\mathsf{in}_I?$ nor write anything else to $\mathsf{out}_I!$.
7. The commands from $\mathsf{in}_I?$ do not affect the input-output behaviour of $\mathcal{I}$, restricted to the ports in S.

This gives a hint about why we call our property *privacy.* Clearly, all that the adversary sees in the ideal world are the inputs of the corrupted party. Thus, if its outputs in the real and the ideal world coincide, then it means that the output of the adversary is not interestingly affected by the messages seen in the real world. Hence, the inputs of honest parties are protected in the real world, since the messages corrupted party sees can not depend on the inputs of the other parties.

Also, note that privacy means that the adversary has the same view in different protocol runs as long as the inputs of the corrupted parties are the same.

As an ideal functionality $(\{\mathcal{I}\}, S)$ is uniquely determined by the machine $\mathcal{I}$, we will, by slight abuse of notation, identify them with each other in the rest of this paper. In addition, note that our lightweight ideal functionalities from Sec. 3.2 will satisfy this definition if we modify them so that they do not send outputs to corrupted parties.

Since the definitions of ideal functionalities differ for private and secure protocols, we cannot conclude directly that all secure protocols are also private. However, if the output is secret-shared and the corresponding sharing is secure against adaptive corruption, then we can perfectly simulate the outputs of the protocol by sharing a random value. Thus, any secure protocol is also a private protocol.

5 Composition

In this section, we define a restricted version of composition that is sufficient for composing arithmetic circuits. In addition, we specify a way to obtain an ideal functionality for the composed circuits from the ideal functionalities of the composed systems. This formalism is necessary to enforce the requirement that only intermediate values in the composed protocol are allowed to be less secure than required by the traditional UC security definitions.

5.1 Ordered composition

We define an ordered composition that restricts the data dependency of the composed protocols. Namely, we assume that the first system can give inputs to the second, but not vice versa. In addition, this definition reduces the complexity of the proofs of the following composition theorems.

Definition 6 (Ordered composition)**.** The ordered composition $Sys_1 \to Sys_2$ of two structures $Sys_1 = \{(M_1, S_1)\}$ and $Sys_2 = \{(M_2, S_2)\}$ is defined if the structures are composable and the data flow is limited to passing from M_1 to M_2.

More formally, the structures have to satisfy the following conditions. First, the port structure must be compatible:

$$\begin{aligned} ports(M_1) \cap forb(M_2, S_2) = \emptyset \ , \\ ports(M_2) \cap forb(M_1, S_1) = \emptyset \ , \end{aligned}$$

and input-output ports must match:

$$S_1 \cap free([M_2])^c = S_2^c \cap free([M_1]) \ .$$

Second, all ports $S_1 \cap free([M_2])^c$ must be output ports and all ports $S_2 \cap free([M_1])^c$ must be input ports to force unidirectional connections from M_1 to M_2.

Definition 7 (Fully ordered composition)**.** The composition $Sys_1 \to Sys_2$ is fully ordered if all the outputs of M_1 go to M_2, i.e., all output ports in S_1 belong to $free([M_2])^c$.

Note that machines in ordered composition can work either sequentially or in parallel and the only thing limited by this definition is the data dependency. Hence, this definition is a complement to traditional composition definitions, that propose additional restrictions. Machines that are not connected directly or through other machines in the system, can trivially be said to be in ordered–but not fully ordered–composition.

Moreover, ordered composition is sufficient for arithmetic circuits, because the circuits are acyclic directed graphs. We can topologically sort such a circuit and define an ordering of the system based on the result.

5.2 Composition of ideal functionalities

We emphasize that the composition of ideal functionalities is no longer an ideal functionality, because it consists of several machines and has a wrong set of ports. We will thus define a separate notion of what it means to compose ideal functionalities. This notion will also be ordered.

Definition 8 (Ideal composition of ideal functionalities)**.** Let $Sys_1 = \{(\{\mathcal{I}_1\}, S_1)\}$ and $Sys_2 = \{(\{\mathcal{I}_2\}, S_2)\}$ be two ideal functionalities for n parties, such that $Sys_1 \to Sys_2$ can be defined. Let $\mathsf{in}_{Ij}?$ and $\mathsf{out}_{Ij}!$ be the ports that $\mathcal{I}_j$ uses for communicating with the adversary. Let $S_1 = S_{1\mathrm{i}} \cup S_{1\mathrm{o}}$ and $S_2 = S_{2\mathrm{i}} \cup S_{2\mathrm{o}}$ be the partitions on S_1 and S_2 to input and output ports. The *ideal composition* of Sys_1 and Sys_2 is the ideal functionality $Sys = \{(\mathcal{I}, S)\}$, where

1. $S = S_{1\mathrm{i}} \cup (S_{2\mathrm{i}} \backslash S_{1\mathrm{o}}^c) \cup S_{2\mathrm{o}} \cup (S_{1\mathrm{o}} \backslash S_{2\mathrm{i}}^c)$;
2. ports in S are divided among n parties according to the partitioning of S_1 and S_2, i.e., $S^i = (S_1^i \cup S_2^i) \cap S$;
3. machine $\mathcal{I}$ has the ports in S, as well as the ports $\mathsf{in}_I?$ and $\mathsf{out}_I!$ to communicate with the adversary;
4. machine $\mathcal{I}$ executes by waiting for input on all input ports in S, then runs $\mathcal{I}_1$ and $\mathcal{I}_2$, and writes the output to output ports in S;
5. on input $(\mathsf{corrupt}, i)$ from in_I, machine $\mathcal{I}$ will behave as required in Def. 5 for the ports in S^i defined for party $\mathcal{CP}_i$ only.

As a shorthand, we denote this resulting machine by $\mathcal{I}_1|\mathcal{I}_2$. We see that the ideal composition of ideal functionalities Sys_1 and Sys_2 behaves as a standard composition, except that the intermediate results of the computation that $\mathcal{I}_1$ sends to $\mathcal{I}_2$ are not sent to the adversary, even if certain parties are corrupted. This corresponds to our intuition of ideal functionalities for secure multi-party composition.

Lemma 1. *The machine $\mathcal{I} = \mathcal{I}_1|\mathcal{I}_2$ can be obtained from $\mathcal{I}_1$ and $\mathcal{I}_2$ with a filter* Filter *where the filter is uniquely determined by the composition.*

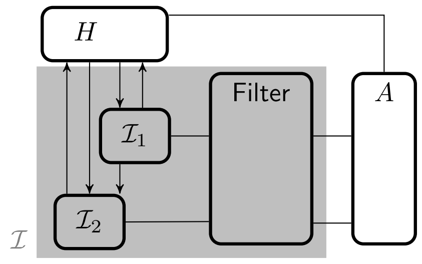

Figure 3: Composition for the ideal functionality

Proof. This is trivial as the output behaviour on out_I and in_I is defined based on the set S of ports. Hence, the Filter has to be such that for both functionalities $\mathcal{I}_1$ and $\mathcal{I}_2$, it forwards all messages from A to corresponding in_{I_j}. However, for all messages from out_I, it only forwards the message corresponding to S_1 or S_2, if the given port is in S. The set S is uniquely fixed by the composition, therefore, Filter is fixed. The Filter always clocks its outputs. This construction is shown on Fig. 3. □

Note that for fully ordered composition, we can assume that the Filter only affects the $\mathcal{I}_2$ part of the composition.

5.3 Ordered composition with predictable outputs

Privacy is a fragile notion of security. A private protocol might leak extra information about inputs when all output shares are sent to a dedicated computing party. For instance, the multiplication protocol depicted in Fig. 2 is private without the resharing step, but it can leak the factors u and v besides the desired output uv, when shares are revealed.

As a result, an ordered composition $Sys_1 \to Sys_2$ can be insecure if Sys_1 is a private protocol and Sys_2 somehow reveals the output shares to party $\mathcal{CP}_i$. As such a protocol Sys_2 is a secure implementation of a functionality $\mathcal{I}_2$ that reveals inputs of all other parties to $\mathcal{CP}_i$, the security of Sys_2 is not sufficient for the security of $Sys_1 \to Sys_2$.

To seal such leakages, we must guarantee that the ideal functionality corresponding to the system $Sys_1 \to Sys_2$ does not reference intermediate values and thus the outputs can be predicted from the inputs. As Sys_2 is assumed to be a secure implementation of $\mathcal{I}_2$, we can formalise the notion in terms of $\mathcal{I}_2$ and consider the system $Sys_1 \to \mathcal{I}_2$.

Also, note that we can always assume that $\mathcal{I}_2$ has no other inputs as the outputs of Sys_1 or $\mathcal{I}_1$, since we can always add extra inputs that are locally forwarded as outputs.

Fig. 4 depicts all configurations that are needed to define the predictability of the outputs. Each line or arrow in Fig. 4 may denote several channels. In particular, Sys_1, $\mathcal{I}_1$ and $\mathcal{I}_2$ have several input and output ports for the environment H corresponding to the different computing parties $\mathcal{CP}_i$.

The configuration $conf_1$ depicts a normal interaction between the system $Sys_1 \to \mathcal{I}_2$ and the adversary A_1. The configuration $conf_3$ depicts an interaction between A_1 and a modified system where the outputs of $\mathcal{I}_2$ are generated by an *output predictor* $\mathcal{I}_*$ that sees only the inputs of Sys_1. Configuration $conf_2$ and $conf_4$ depict the same interaction for the ideal functionality $\mathcal{I}_1$ corresponding to Sys_1.

More precisely, the machine Split in $conf_3$ and $conf_4$ copies each input to both output channels. After that, it clocks the channel leading to $\mathcal{I}_*$. The machine Split has another input port $ctrl_{\mathsf{Split}}?$; the machine $\mathcal{I}_*$ is intended to have the corresponding output and clocking ports $ctrl_{\mathsf{Split}}!$ and $ctrl_{\mathsf{Split}}{}^{\triangleleft}!$. The machine Split ignores the inputs it receives from $ctrl_{\mathsf{Split}}?$. But whenever Split is invoked with an input from this port, it clocks the other output channel, to which it had copied its input. In this manner, Split is able to immediately pass its input to both machines expecting them, with slight help from the machine $\mathcal{I}_*$.

The machine Mux works as follows. It has a set C of the identities of parties that the adversary A has corrupted, and a list E to store the messages it received from Sys_1 or $\mathcal{I}_1$ and did not yet send to the adversary.

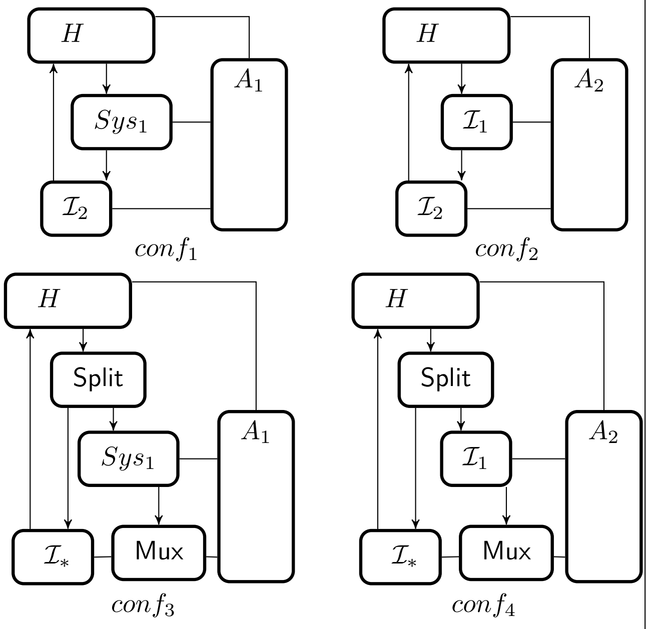

Figure 4: Configurations for predictable output

1. On input $(output, \ell_x, x)$, on behalf of party $\mathcal{CP}_i$ from Sys_1 (or $\mathcal{I}_1$), it sends $(output, \ell_x, i)$ to $\mathcal{I}_*$, and clocks the channel to $\mathcal{I}_*$. Immediately after that, it expects to get a message on a channel from $\mathcal{I}_*$ (this channel is clocked by $\mathcal{I}_*$). Mux ignores the received message. Instead, it sends $m = (input, \ell_x, x, i)$ to A, if $i \in C$. Otherwise, it adds m to E.

2. On input $(corrupt, i)$ from A it adds i to C and sends to A all entries $(input, \ell_x, x, i) \in E$. It forwards the corruption request to $\mathcal{I}_*$.

3. On input $(output, \ell_x, x, i)$ from $\mathcal{I}_*$ it sends an entry $(output, \ell_x, x, i)$ to A and clocks the output.

4. On input $(input, \ell_x, x, i)$ from $\mathcal{I}_*$ it does nothing.

We see that, similarly to Split, the machine Mux expects the help of $\mathcal{I}_*$ in sending a message both to it and to the adversary. The machine $\mathcal{I}_*$ is expected to use these notifications to keep track of the progression of the computation, in order to provide the outputs to H at the same time it would have received them in $conf_1$ and $conf_2$.

Definition 9. We say that the composition $Sys_1 \to \mathcal{I}_2$ has *a predictable outcome* if there exists a predictor machine $\mathcal{I}_*$ such that, for the following configurations on Fig. 4, we have $view_{conf_1}(H) = view_{conf_3}(H)$.

Definition 10. We say that the compositions $Sys_1 \to \mathcal{I}_2$ and $\mathcal{I}_1 \to \mathcal{I}_2$ have *a jointly predictable outcome* if there exists a single output predictor $\mathcal{I}_*$ such that, for the following configurations on Fig. 4, we have $view_{conf_1}(H) = view_{conf_3}(H)$ and $view_{conf_2}(H) = view_{conf_4}(H)$.

Lemma 2. *If Sys_1 is a correct implementation of $\mathcal{I}_1$ then $Sys_1 \to \mathcal{I}_2$ has a predictable outcome iff $Sys_1 \to \mathcal{I}_2$ and $\mathcal{I}_1 \to \mathcal{I}_2$ have a jointly predictable outcome.*

Proof. Let $\mathcal{I}_*$ be a predictor for $Sys_1 \to \mathcal{I}_2$. Then it must be a suitable predictor also for $\mathcal{I}_1 \to \mathcal{I}_2$. Indeed, if A_2 could cause $view_{conf_2}(H) \neq view_{conf_4}(H)$, then we could construct A_1 that runs A_2, but filters out all extra messages sent by Sys_1. Since Sys_1 is a correct implementation of $\mathcal{I}_1$, the resulting configurations would be equivalent to $conf_2$ and $conf_4$. Consequently, $view_{conf_1}(H) \neq view_{conf_3}(H)$. □

This result assures that all output predictability is not an issue for the compositions $Sys_1 \to Sys_2$ that evaluate arithmetic circuits, as long as the protocols are correct. The same holds for randomised functions, as long as the outputs are predictable from the inputs and protocols are correct. Output predictability requires separate checks only if the protocols produce approximate results.

5.4 Composition theorems

Using the definition of privacy and ordered composition, we can state the main theorems of this work. These show that the privacy notion is composable and that our approach of combining private and secure protocols is secure.

Theorem 2 (Secure composition, informal)**.** *The fully ordered composition of black-box private and black-box simulatable protocols with jointly predictable outcome is black-box simulatable.*

Theorem 3 (Black-box privacy composition, informal)**.** *The ordered composition of black-box private protocols is black-box private.*

The formal versions of these theorems are stated and proven in Sec. 7. In general, these proofs show how to construct a simulator for the composed system by using the simulators of the sub-protocols. This simulator has to be a simulator for the composed ideal functionality and the composed real system. For that, we show that the indistinguishability of the view of H can be reduced to the privacy definition. The proofs use output predictability to guarantee that the ideal functionality is properly defined and that intermediate values are not revealed in the protocol.

6 Simulators

For proving theorems 2 and 3, we have to construct the simulators for the composed systems. These constructions are more complex and invasive than in the proof of Thm. 1. Hence, we must set up some definitions for combining the simulators that we have from the premises of the theorems. Especially, we need to extend the stand-alone simulators so that they can be combined with each other in order to form a simulator for the composed system.

6.1 Privacy simulator

A simulator is a mediator between a real world adversary and the matching ideal functionality $\mathcal{I}$. Let $RS = (\hat{M}, S)$ be a real structure and $Id = (\{\mathcal{I}\}, S)$ be the corresponding ideal structure. Then a black-box *privacy simulator* for an ideal structure Id and a real structure RS is a machine $\mathsf{Sim}^{Id,RS}$ that has ports $\mathsf{in}_I^{Id}!$ and $\mathsf{out}_I^{Id}?$ for communicating with Id and the set of ports $\mathcal{AP}^{RS}$ for communicating with the adversary that expects to run in parallel to RS.

The channel in_I^{Id} is clocked by the simulator, while out_I^{Id} is clocked by Id. The set $\mathcal{AP}^{RS}$ contains the ports $\mathsf{out}_{i,Sys}!$ for each machine M_i in the structure RS. The channels $\mathsf{out}_{i,Sys}$ are clocked by the adversary. W.l.o.g. we may assume that, each time the simulator is activated, it only writes to the output ports in $\mathcal{AP}^{RS}$ that belong to a single machine in RS. This is because the adversary completely controls the scheduling of RS. In particular, the simulator writes into at most one $\mathsf{out}_{i,Sys}$ during each invocation.

A simulator provides *perfect privacy* if the condition $view_{conf_1}(H') = view_{conf_2}(H')$ is satisfied for any $H = H' \cup H_\perp$ and A, such that $conf_1 = (\hat{M}, S, H, A)$ and $conf_2 = (\mathcal{I}, S, H, \mathsf{Sim}^{Id,RS} \cup A)$ are privacy configurations. Similarly, one can define simulators that provide statistical and computational privacy. In the following, we consider only simulators that provide perfect privacy and reference them as *privacy simulators*.

6.2 Extended simulator

An extended simulator is a simulator that additionally computes the outputs of corrupted parties. When composing the structures, these outputs have to be given as inputs to the simulator(s) of the next stage(s) of the composition.

Let Sys_1 and Sys_2 be the composed systems. In the real world, there will be channels $\mathsf{pipe}_1, \ldots, \mathsf{pipe}_k$ between Sys_1 and Sys_2 that are clocked by the adversary. However, as discussed previously, the composition of the respective ideal world machines $\mathcal{I}_1$ and $\mathcal{I}_2$ is a single machine $\mathcal{I}$ without the ideal world equivalents of pipe_i. Therefore, these buffers will have to be part of the simulator of the composed system. We call the respective buffers output_i as they carry the outputs of the simulator for Sys_1 to the simulator of Sys_2.

Let n be the number of parties that could be potentially corrupted by the adversary. Then the extended simulator writes the outputs of corrupted parties into dedicated output ports $\mathsf{output}_1!, \ldots, \mathsf{output}_n!$. As we allow adaptive corruption, the adversary may sometimes corrupt the party after the protocol has been executed. Nevertheless, we might need the output of the corresponding protocol to proceed. Hence, the extended simulator has dedicated ports $\mathsf{foutput}_1!, \ldots, \mathsf{foutput}_n!$ for sending such outputs so that the adversary can not control their timing.

Let *conf* be a configuration with n parties, containing the channels $\mathsf{in}_{i,Sys}$, out_i, output_i, and possibly $\mathsf{foutput}_i$ for $i \in \{1, \ldots, n\}$. As the adversary does not have matching ports, we need a dummy machine Sink to formally complete the configurations. Let Sink be a machine with ports $\mathsf{output}_i?$, $\mathsf{foutput}_i?$ for $i \in \{1, \ldots, n\}$ that just consumes inputs. In protocol composition, the machine Sink would be removed and output_i and $\mathsf{foutput}_i$ would be connected to the simulator of the next protocol. This connection is defined using a multiplexer that joins these channels and the inputs that the simulator commonly obtains from the respective ideal functionality.

Besides Sink, we need another dummy machine Sink' with ports $\mathsf{output}_i?$, $\mathsf{output}_i!$ for $i \in \{1, \ldots, n\}$ in the real world. The sole purpose of Sink' is to allow the adversary A to clock the non-existing output_i channels in the real configuration. This is needed to assure closeness under the composition. Although real-world adversaries have no access to output_i channels, the adversaries created in the composability proofs may clock these channels due to channel renaming introduced by a reduction.

Let τ be a *trace* of this configuration, i.e. a list of pairs (channel name, message), recording which messages were sent on which channel in which order during a run. Let $\mathcal{O}(\tau) = (m_1, \ldots, m_n)$ be the list of protocol outputs observed by the adversary. That is, m_i is either the list of messages output on channel out_i if there was a corruption request on $\mathsf{in}_{i,Sys}$ or $\perp$ if there was no such request. Recall that out_i is the channel that takes the outputs of the system to H or the next system in composition. Let $\mathcal{O}'(\tau) = (m_1, \ldots, m_n)$ be the list of outputs generated by the extended simulator. Again, m_i will be $\perp$ if there was no corruption request on $\mathsf{in}_{i,Sys}$. Otherwise, let m_i be the concatenation of lists of messages that appeared on $\mathsf{foutput}_i$ and on output_i. For the configuration *conf*, let $\mathcal{O}_{conf}$ and $\mathcal{O}'_{conf}$ be the distributions of $\mathcal{O}(\tau)$ and $\mathcal{O}'(\tau)$ over all possible runs of *conf*.

Definition 11 (Extended simulator)**.** An *extended simulator* for an n-party ideal structure $Id = (\{\mathcal{I}\}, S)$ and real structure $RS = (\hat{M}, S)$ is a machine $\mathsf{ExtSim}^{Id,RS}$, that

1. has the ports in the set $\mathcal{AP}^{RS} \cup \{\mathsf{in}_I^{Id}!, \mathsf{out}_I^{Id}?\}$ and in the set $\{\mathsf{output}_1!, \mathsf{foutput}_1!, \ldots, \mathsf{output}_n!, \mathsf{foutput}_n!\}$;
2. clocks the channels $\mathsf{foutput}_i$;
3. in case the corruption request on $\mathsf{in}_{i,Sys}?$ comes after $(output, \ell_x)$ has been sent on $\mathsf{out}_{i,Sys}$, forwards this request to Id, and after learning the input of i-th party, immediately makes an output $(output, \ell_x, x)$ on $\mathsf{foutput}_i$ and clocks that channel;
4. never outputs on $\mathsf{foutput}_i$, except for point 3;
5. is a simulator that provides privacy: $view_{conf_1}(H') = view_{conf_2}(H')$ for any $H = H' \cup H_\perp$ and A, and privacy configurations $conf_1 = (\hat{M}, S, H, A \cup \mathsf{Sink}')$ and $conf_2 = (\mathcal{I}, S, H, \mathsf{ExtSim}^{Id,RS} \cup A \cup \mathsf{Sink})$;
6. correctly computes the outputs: $\mathcal{O}_{conf_1} = \mathcal{O}'_{conf_2}$, for the same $H = H' \cup H_\perp$, A, $conf_1$ and $conf_2$ as before.

Lemma 3 (Extended simulators exist)**.** *Let* $\mathsf{Sim}^{Id,RS}$ *be a privacy simulator for ideal structure Id and real structure RS. Then there exists an extended simulator* $\mathsf{ExtSim}^{Id,RS}$.

Proof. An extended simulator can be constructed as follows. Let $\mathsf{Sim}_*^{Id,RS}$ be a machine obtained from $\mathsf{Sim}^{Id,RS}$ by renaming its ports $\mathsf{out}_{i,Sys}!$ to $\mathsf{out}_{i,sim}!$ and allowing it to clock the channels $\mathsf{out}_{i,sim}$. The machine $\mathsf{Sim}_*^{Id,RS}$ clocks the channel $\mathsf{out}_{i,sim}$ at each invocation when it outputs a message in it. As explained before, there is always at most one such i, that $\mathsf{Sim}_*^{Id,RS}$ has written in $\mathsf{out}_{i,sim}!$.

Let Extr_i be a machine with ports $\mathsf{out}_{i,sim}?$, $\mathsf{extr}_i!$ and $\mathsf{out}_{i,Sys}!$, clocking the channel extr_i. The machine Extr_i copies the inputs from $\mathsf{out}_{i,sim}?$ to $\mathsf{out}_{i,Sys}!$. If the input is of the form $(output, \ldots)$, then it copies that input to $\mathsf{extr}_i!$ as well and clocks that channel.

Let $\mathsf{OutFilter}_i$ be a machine with ports $\mathsf{extr}_i?$, $\mathsf{output}_i!$ and $\mathsf{foutput}_i!$, clocking the channel $\mathsf{foutput}_i$. The machine $\mathsf{OutFilter}_i$ has a buffer M for output notifications and processes two kinds of output tuples: in the form $(output, \ell_x)$ for output notifications of the honest parties for a label ℓ_x, and in the form $(output, x, \ell_x)$ for the outputs of corrupted parties where x is the output value. On input $(output, \ell_x)$ from $\mathsf{extr}_i?$ $\mathsf{OutFilter}_i$ saves ℓ_x as a label from i to M and forwards the message out on $\mathsf{output}_i!$. On input $(output, x, \ell_x)$ from $\mathsf{extr}_i?$ $\mathsf{OutFilter}_i$ works as follows:

1. if the label ℓ_x has been stored in M for party i then this input is forwarded on $\mathsf{foutput}_i!$, ℓ_x is removed from M and the channel $\mathsf{foutput}_i$ is clocked;

2. if this label ℓ_x has not been stored then $\mathsf{OutFilter}$ forwards the message on $\mathsf{output}_i!$.

We let $\mathsf{ExtSim}^{Id,RS}$ be the composition of the machines $\mathsf{Sim}_*^{Id,RS}$, Extr_i and $\mathsf{OutFilter}_i$ for all $i \in \{1, \ldots, n\}$, see Fig. 5. Clearly, it satisfies the structural properties 1, 2 and 4 of Def. 11. It also satisfies the third property due to the manner how the real functionality notifies the adversary of computed outputs, and the manner $\mathsf{OutFilter}_i$ handles these notifications. The 5th property is satisfied because the configurations described there behave in exactly the same way as the privacy configurations in Def. 4, except for the occasional invocations of Sink or Sink' which do not affect the view of H'. The sixth property is satisfied because $\mathsf{Sim}^{Id,RS}$ correctly simulates the outputs of corrupted parties, which are then output on channels $\mathsf{foutput}_i$ and output_i. □

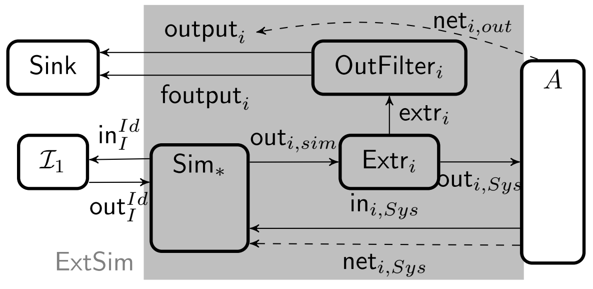

Figure 5: An extended simulator construction with the ideal world functionality, adversary and connection to the rest of the simulator

7 Security of composed protocols

In this section, we prove that the ordered composition of private and secure protocols is secure. For this we first show that we can limit the class of adversaries (Lemma 4) and then use this new class to show the security of the composed protocol (Thm. 2). From these results we know that our composition is secure with respect to general adversaries. Furthermore, in Thm 3 we also show that privacy is composable.

7.1 Restricted adversary model

The analysis of ordered compositions can be simplified by restricting the adversarial behaviour. Let $Sys_1 \to Sys_2$ be an ordered composition with the same set of corruptible parties $\mathcal{CP}_1, \ldots, \mathcal{CP}_n$. Let $\mathcal{A}_0$ be a subclass of adversaries that never corrupt a party $\mathcal{CP}_i$ in Sys_2 before they corrupt the party $\mathcal{CP}_i$ in Sys_1 and let $\mathcal{A}_1 \subseteq \mathcal{A}_0$ be a subclass of adversaries that corrupts each party $\mathcal{CP}_i$ simultaneously in both systems. Then it is easy to see that these restrictions are not limiting.

Lemma 4. *Let two systems Sys_1 and Sys_2 be composed in the ordered composition as $Sys_1 \to Sys_2$. Then the composed system is as secure with respect to the general set of adversaries as it is for adversaries in class $\mathcal{A}_0$.*

Proof. For the proof, we show how to construct a restricted adversary from a general adversary A so that the view of H in the two constructions coincides, therefore satisfying the security requirement. For that, we have to show that the original system is as secure as the composed system with the construction on Fig. 6.

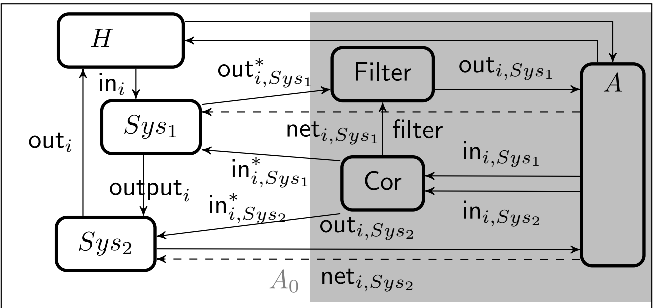

Figure 6: Construction for the restricted adversary

A simple stateless machine Cor assures that a party is always corrupted in Sys_1 before in Sys_2. It has two input ports $\mathsf{in}_{i,Sys_1}?$ and $\mathsf{in}_{i,Sys_2}?$ for receiving corruption requests for the machines in Sys_1 or Sys_2 and output ports $\mathsf{in}_{i,Sys_2}!$ and $\mathsf{in}_{i,Sys_1}!$ for forwarding the corruption requests. The output port $\mathsf{filter!}$ is used for controlling a delay box Filter. The machine Cor works as follows:

- on an input $(corrupt, i)$ from $\mathsf{in}_{i,Sys_1}?$ it forwards the input to the ports $\mathsf{filter!}$ and $\mathsf{in}^*_{i,Sys_1}!$.
- on an input $(corrupt, i)$ from $\mathsf{in}_{i,Sys_2}?$ it forwards the input to the ports $\mathsf{in}^*_{i,Sys_1}!$ and $\mathsf{in}^*_{i,Sys_2}!$.

To prevent A from receiving unrequested information, we have inserted Filter between Sys_1 and A. Filter keeps a set of corrupted parties $\mathcal{C}$ whose input must go through. For other parties, the filter stores the last message or passes the messages $(output, \ell_x)$ that the honest parties are supposed to send to A. The port $\mathsf{filter?}$ is for updating the list $\mathcal{C}$. If an input $(corrupt, i)$ is written to $\mathsf{filter?}$ then i is added to $\mathcal{C}$ and the last message from $\mathcal{CP}_i$ is released for A as the current state of $\mathcal{CP}_i$. The scheduling of Cor and Filter is fixed by the clocking signals so that the list $\mathcal{C}$ is always updated before $(corrupt, i)$ is written to $\mathsf{in}^*_{i,Sys_1}!$.

Sys_1 may have received more corruption requests in this setting than in the original construction, but the Filter reduces the view to only corrupted parties. It is easy to see that the reduced view has the same probability distribution as the simulation output if only this set of parties is corrupted, because it clearly holds in the real world. Hence, the outputs of A in different worlds coincide. The correspondence of this construction and the restriction on class $\mathcal{A}_0$ is trivial. □

Corollary 1. *Let two systems Sys_1 and Sys_2 be composed in the ordered composition $Sys_1 \to Sys_2$. Then the composed system is as secure with respect to the general set of adversaries as it is for adversaries in class $\mathcal{A}_1$.*

Proof. It is sufficient to show the construction for a restricted adversary $A \in \mathcal{A}_1$. The corresponding construction is analogous to the proof of Lemma 4. We must define a machine Cor that corrupts a party simultaneously in Sys_1 and Sys_2 and a filter Filter for deleting unexpected messages. As a result, we get an adversary that behaves identically. □

Based on these results, we define and prove composition theorems for the class $\mathcal{A}_1$, since their behaviour is more easy to understand and analyse. Note that this approach is suitable for cases of static or adaptive corruption, but not for modelling mobile corruption where parties can be corrupted for some time, but can become uncorrupted afterwards.

7.2 Secure composition

In this section, we restate the main theorem and prove the security of the fully ordered composition of private and secure systems. In general, the idea is that a private protocol finished by a secure protocol is secure. Informally, we know that both simulators ensure that the view of A in the protocol is indistinguishable from the real world. In addition, the simulator of the secure protocol ensures that the view of A is consistent with the view of H.

Theorem 2. *Let $Sys_1 \geq_{priv}^{model} \mathcal{I}_1$ in a black-box manner and $Sys_2 \geq_{sec}^{model} \mathcal{I}_2$ in a black-box manner, where model may be perfect, statistical, or computational. Let the ordered compositions $Sys_1 \to \mathcal{I}_2$ and $\mathcal{I}_1 \to \mathcal{I}_2$ have a jointly predictable outcome. Then $Sys_1 \to Sys_2 \geq_{sec}^{model,\mathcal{A}_1} \mathcal{I}$ in a black-box manner, where the composition $Sys_1 \to Sys_2$ is fully ordered and $\mathcal{I}$ is the ideal composition of $\mathcal{I}_1$ and $\mathcal{I}_2$.*

Proof. Fig. 7 illustrates the ordered composition of two systems that is our $conf_0$. Due to the black-box simulatability of $Sys_2 \geq_{sec}^{model} \mathcal{I}_2$ and Thm. 1 we have $view_{conf_0}(H) \approx view_{conf_1}(H)$ for some simulator Sim_2. See Fig. 8.

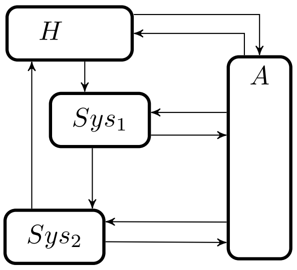

Figure 7: Composition of two real systems, $conf_0$

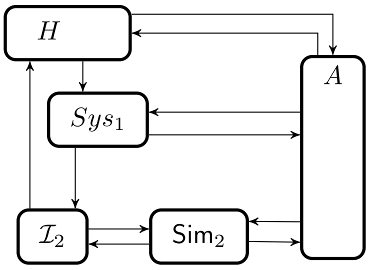

Figure 8: Composition of a real and an ideal system, $conf_1$

As the next step of the proof, we use output predictability (formalised as Def. 9) of $Sys_1 \to \mathcal{I}_2$ to separate systems Sys_1 and Sys_2. We consider Sim_2 as a part of the adversary and obtain $view_{conf_1}(H) = view_{conf_2}(H)$ for the configuration $conf_2$ depicted in Fig. 9. Recall that we have joint predictability, hence, the same $\mathcal{I}_*$ also demonstrates the predictable outcome of $\mathcal{I}_1 \to \mathcal{I}_2$.

Now we can do a cosmetic step that simplifies the exposition by introducing a new machine H_* that is the combination $H_* = H \cup \mathcal{I}_* \cup \mathsf{Split}$. This can be seen on Fig. 10. For the machine H trivially $view_{conf_2}(H) = view_{conf_3}(H)$. The scheduling does not change.

Next, we use the assumption about the adversary in class $\mathcal{A}_1$. Namely, we assume that the adversary has corrupted party $\mathcal{CP}_i$ in either both systems or in none. This implies that it has seen all inputs of Sim_2 as corrupted parties' outputs in Sys_1. This enables us to do one more rewiring that results in Fig. 11 with a new adversary A_* that acts like Mux with the difference that it does not receive the honest parties' outputs from Sys_1. The part $H_\perp$ that was previously used for honest parties' outputs as part of Mux is now used for all outputs of Sys_1. In the case of static corruption, this step could be done trivially because we could divide Mux based on corrupted and not corrupted ports. In the general case, this step can still be done by introducing the Extr

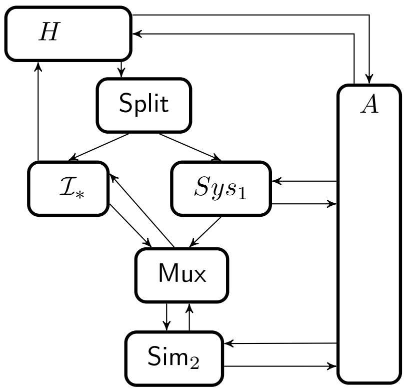

Figure 9: Splitting the composition to two independent parts, $conf_2$

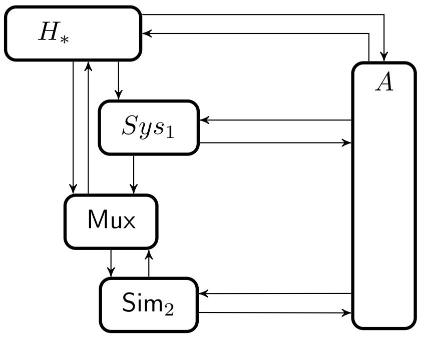

Figure 10: Regrouping the machines, $conf_3$

and OutFilter from the ExtSim construction to the channel from Sys_1 to A. The outputs from OutFilter serve as the inputs of the Mux_* that in is just Mux with different input ports. Note that an analogous setup can be seen on Fig. 15 except we have just Extr and OutFilter and not full ExtSim. We then use the machines Extr, OutFilter and Mux_* together with A to form A_*. The scheduling, in general, remains the same, but each time the adversary A clocks the outputs Sys_1 to $H_{*\perp}$, A_* also clocks the output_i of the OutFilter. The simulator Sim_2 also changes in a non-essential way to Sim_2^*, as some of its inputs and outputs now move over different channels.

Therefore, $view_{conf_3}(H_*) = view_{conf_4}(H_*)$.

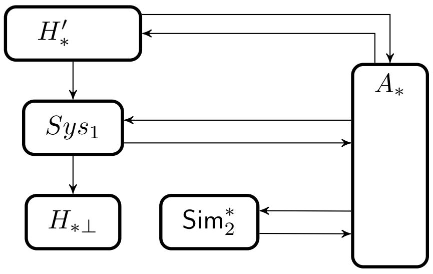

Figure 11: Splitting the Mux by its two roles, $conf_4$

Finally we finish with another cosmetic change to join A_* and Sim_2 to A_{**} as shown on Fig. 12. Trivially, $view_{conf_4}(H_*) = view_{conf_5}(H_*)$. Hence, by tracing back the changes, we have $view_{conf_0}(H) \approx view_{conf_5}(H)$ as H is a sub-machine of H_* and the view of H does not change if the view of H_* remains the same.

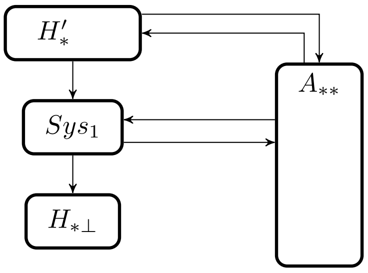

Figure 12: Redefining the adversary, $conf_5$

So far, we just simplified the composition $Sys_1 \to Sys_2$. To complete the proof, we must provide the simulator that works in conjunction with the ideal functionality $\mathcal{I}$ in the configuration corresponding to $conf_0$. Let this be $conf'_0$. Lemma 1 allows us to express $\mathcal{I}$ as a composition of $\mathcal{I}_1$, $\mathcal{I}_2$ and Filter. Then this composition corresponds to the left grey box in Fig. 13. We claim that a combination of machines ExtSim_1 (the existence of which is shown in Lemma 3), Sim_2 and Mux_t defined below serves as a suitable simulator that proves the theorem. The corresponding configuration $conf'_1$ with these machines is depicted on Fig. 13.

The machine Mux_t acts like Mux in Sec. 5.3 in the predictable output definition except that ExtSim is in the role of Sys_1 and Filter is in the role of $\mathcal{I}_*$. In addition, Mux has two types of input channels, output_i and $\mathsf{foutput}_i$ from ExtSim, that are both used as inputs from Sys_1 in Sec. 5.3. The scheduling is fixed by the description of individual machines.

We consider A, ExtSim, Mux_t, Filter and Sim_2 as an adversary in Def. 10 to obtain the composition $conf'_2$ analogous to $conf_2$. For this, we introduce the machines Split, $\mathcal{I}_*$ and Mux according to the predictable output definition illustrated on Fig. 4. Secondly, we also do the simplification step to push $\mathcal{I}_*$ and Split to H to obtain H_* and the resulting configuration $conf'_3$ can be seen on Fig. 14. The scheduling is defined by Def. 10 and $conf'_1$.

This results in a quite complicated composition of small machines. At first, note that we can discard Filter which only affects the input values that move out from Mux, but by definition Mux_t does not do anything with inputs from Filter. The machine Mux has two roles, it can either forward the inputs of the second composed protocol based on the outputs of ExtSim (that represent the

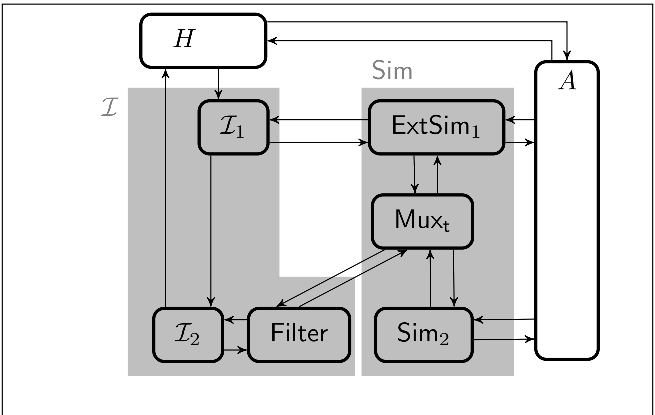

Figure 13: Sys_1 replaced by the ideal system, $conf'_1$

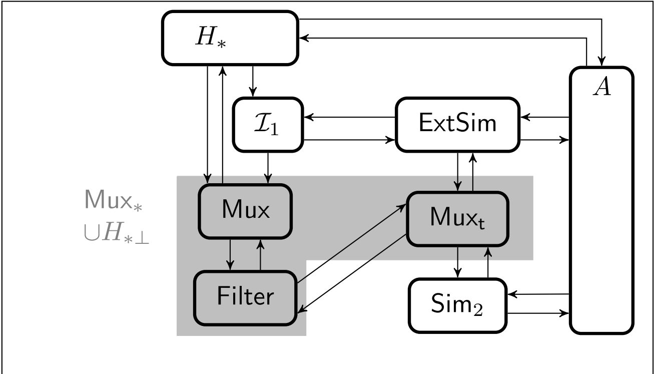

Figure 14: Introduced intermediate machines, $conf'_3$

outputs of the first system) or the outputs of the composition it gets from H_*. For the same reasons that we used to discard Filter, we can also discard the part of Mux that uses the values from $\mathcal{I}_1$ because $\mathsf{Mux_t}$ never uses them. Hence, we join this part of Mux with $\mathsf{Mux_t}$ to obtain Mux_* as on Fig. 15. The part of Mux that does not use inputs from $\mathcal{I}_1$ is represented by $H_{*\perp}$. This is analogous to Fig. 10, except that Mux has been split to two distinct parts.

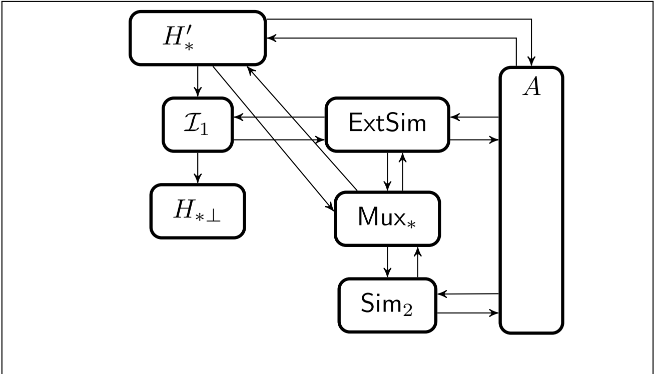

Figure 15: Simplification of $conf'_3$, $conf''_3$

As the final step, we decompose ExtSim to its parts Sim_1, Extr and OutFilter as illustrated on Fig. 5. Then we introduce A_{**} by combining A with Extr, OutFilter, Mux_* and Sim_2 to arrive at configuration $conf'_5$ on Fig. 16. With this we have shown that the view of the adversary in the ideal case is equivalent to $conf'_5$ or more formally $view_{conf'_0}(H) \approx view_{conf'_5}(H)$ as H is a sub-machine of H'_*.

Note that the party H_* in both of the reductions is the same if initial H is the same. This is trivial as the machines Split and $\mathcal{I}_*$ are the same due to joint predictability (Def. 10). The same

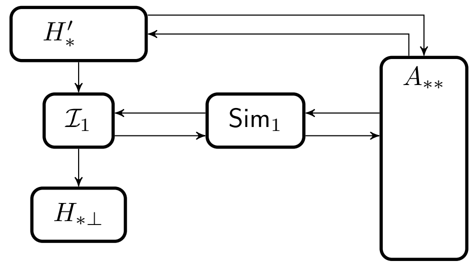

Figure 16: Final configuration, $conf'_5$

holds for A_{**}. Trivially, the machine Sim_2^* included to A is the same. In addition, for $conf_3$ we argued that the step to $conf_4$ can be done by adding Extr and OutFilter which is exactly what we added in $conf'_4$. Finally, the part Mux_* has exactly the same functionality in the two descriptions.

With these two reductions we have shown that the question if $Sys_1 \to Sys_2$ is as secure as $\mathcal{I}$ is reduced to question if $conf_5$ is indistinguishable from $conf'_5$. For that we can view the final state $conf_5$ as a privacy configuration. By definition Sys_1 is as private as $\mathcal{I}_1$ only if for each adversary A_{**} we can use the same simulator Sim_1 such that $view_{conf_5}(H) \approx view_{conf'_5}(H)$. The claim follows. □

Corollary 2. *Let $Sys_1 \geq_{priv}^{model} \mathcal{I}_1$ in black-box manner and $Sys_2 \geq_{sec}^{model} \mathcal{I}_2$ in black-box manner, where model may be perfect, statistical, or computational. Let the ordered compositions $Sys_1 \to \mathcal{I}_2$ and $\mathcal{I}_1 \to \mathcal{I}_2$ have jointly predictable outcome. Then $Sys_1 \to Sys_2 \geq_{sec}^{model} \mathcal{I}$ in a black-box manner, where the composition $Sys_1 \to Sys_2$ is fully ordered and $\mathcal{I}$ is the ideal composition of $\mathcal{I}_1$ and $\mathcal{I}_2$.*

Proof. The proof is a direct result from Lemma 4 and Thm. 2. □

7.3 Composability of privacy

In this section, we show that the composition of black-box-private protocols is black-box-private. We prove the theorem for the composition of two systems and based on this it can be extended for larger compositions. Note that a plain channel that does not use or modify the inputs is always private and based on privacy configuration definition, it is easy to see that a composition of two systems, that do not communicate, is private.

Theorem 3. *Let $Sys_1 \geq_{priv}^{model} \mathcal{I}_1$ and $Sys_2 \geq_{priv}^{model} \mathcal{I}_2$ in a black-box manner, where model may be perfect, statistical, or computational. Then $Sys_1 \to Sys_2 \geq_{priv}^{model,\mathcal{A}_1} \mathcal{I}$ in a black-box manner, where the composition $Sys_1 \to Sys_2$ is ordered and $\mathcal{I}$ is the ideal composition of $\mathcal{I}_1$ and $\mathcal{I}_2$.*

Proof. Note that we can make several simplifying assumptions to the compositions. First, we can assume that all inputs are inputs to Sys_1. If it is not the case, then we can add inputs of Sys_2 as inputs of Sys_1 that are just forwarded as outputs. Similarly, we can assume that all outputs of Sys_1 are used as inputs of Sys_2 by using similar input forwarding. Hence, we can prove the claim only for fully ordered composition.

Consider a configuration analogous to Fig. 7 except with a privacy configuration with $H = H' \cup H_\perp$, such that the output of Sys_2 goes to $H_\perp$ in $conf_0$. Let $conf'_0$ be the corresponding ideal configuration. For the proof, we need a simulator Sim that can be inserted between the ideal functionality $\mathcal{I}$ and the real world adversary. In the following, we build this simulator step by step.

As the first step, we can replace Sys_2 with the corresponding ideal functionality and corresponding simulator Sim_2 by joining H' and Sys_1 to new H'_* and using this as a new privacy configuration. The proof for this substitution is analogous to the proof of the original Thm. 1 about the composability of security [2]. We get a configuration $conf_r$ where $view_{conf_r}(H'_*) \approx view_{conf_0}(H'_*)$. Thus, also $view_{conf_0}(H') \approx view_{conf_r}(H')$ because H' is a submachine of H'_*. We get a situation illustrated on Fig. 17.

Next, consider the ideal configuration $conf'_0$ with $\mathcal{I}$, H and A, and the desired simulator Sim. We use Lemma 1 to replace $\mathcal{I}$ with a combination of $\mathcal{I}_1$, $\mathcal{I}_2$, and Filter. Note that if $A \in \mathcal{A}_1$, then Filter only needs to connect to $\mathcal{I}_2$, because all messages between $\mathcal{I}_1$ and the adversary are simply

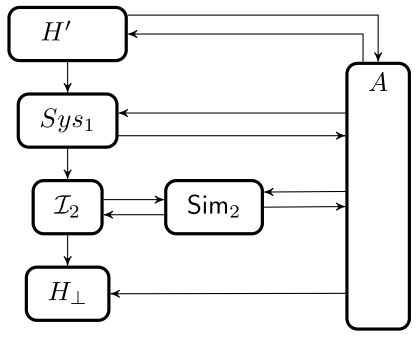

Figure 17: Composition of real and ideal private systems, $conf_r$

passed through. As the simulator, we propose the combination of machines Sim_1, Extr, $\mathsf{OutFilter}$, Mux and Sim_2, let $conf_s$ be the resulting configuration (see Fig. 18).

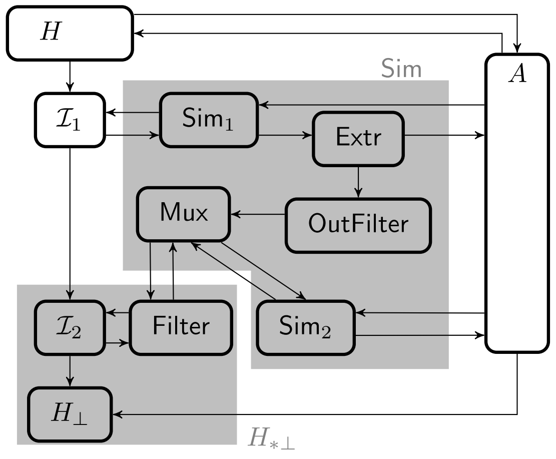

Figure 18: Simulator construction for the $\mathcal{I}$ and $Sys_1 \to Sys_2$, $conf_s$

Note that the collection Sim_1, Extr and $\mathsf{OutFilter}$ is actually ExtSim_1. In addition, Mux works as described in Sec. 5.3 without the special scheduling and two equivalent input channels from $\mathsf{OutFilter}$.

As we consider fully ordered composition, all inputs of $\mathcal{I}_2$ are stopped by the Filter and the private ideal functionality $\mathcal{I}_2$ does not give outputs, accordingly, there is no communication from Filter to Mux. Hence, we can define a new $H_{*\perp}$ that is a collection $\mathcal{I}_2$, Filter and $H_\perp$. If we consider a new adversary A_* that is the combination of A, Extr, $\mathsf{OutFilter}$, Mux and Sim_2 then we have a privacy configuration. We can thus replace $\mathcal{I}_1$ and Sim_1 in $conf_s$ with Sys_1 without changing the view of H.

We claim that the setup of $conf_s$ results in the same view as $conf_r$. By definition, we know that Sim_1 and Sim_2 can produce a view that is indistinguishable from the real protocol run. Therefore, the only thing to argue is that Sim_2 sees equivalent inputs in the two configurations. By definition, in $conf_r$ the machine Sim_2 receives from $\mathcal{I}_2$ exactly the inputs that are the corrupted parties' outputs in Sys_1. However, this is also the case in $conf_s$ because, by definition, Extr can also extract the outputs of the corrupted parties in a real protocol run and, therefore, the corrupted parties' outputs in Sys_1. $\square$

Corollary 3. *Let $Sys_1 \geq^{model}_{priv} \mathcal{I}_1$ and $Sys_2 \geq^{model}_{priv} \mathcal{I}_2$ in a black-box manner, where model may be perfect, statistical, or computational. Then $Sys_1 \to Sys_2 \geq^{model}_{priv} \mathcal{I}$ in a black-box manner, where the composition $Sys_1 \to Sys_2$ is ordered and $\mathcal{I}$ is the ideal composition of $\mathcal{I}_1$ and $\mathcal{I}_2$.*

Proof. Again, a direct result from Lemma 4 and Thm. 3. □

7.4 Applicability of the composition theorems

By using structural induction and Cor. 3 it is straightforward to prove that an ordered composition of two or more private protocols remains private. To achieve universally composable security, we must rerandomise the outputs of the resulting private protocol. The main restriction posed by Cor. 2 is that all outputs of the private system have to be used by the secure system so that the outputs of the composition are the outputs of the secure system.

This result is especially valuable when we consider the secure evaluation of arithmetic circuits, since it allows us to define a solution based on private protocols and use secure resharing protocols to achieve universal composability. The resulting efficiency gain can be significant when the computational procedure requires many intermediate values to compute the outcome. Moreover, the corresponding choice between private and secure operations can be completely automated. Secure protocols are mainly required for publishing or resharing a value which can be used for finishing computations or finishing some stage of computations.

8 Conclusions

We have shown that the privacy requirement is sufficient for most secure computation primitives in the passive security model in order to obtain universally composable secure multi-party computation protocols.

Private protocols are often more efficient, while their composition is no more complex than composing secure protocols. Therefore, we can obtain better performance without compromising the rigour of security arguments. We have also shown that privacy and security are tightly related, and a private protocol can be made secure by introducing a secure finalising step, which can be very simple.

We believe our ideas are applicable also in the case of active adversaries against secure multi-party computation protocols. There will be additional difficulties because the adversarial behaviour is more complex and the validity of the sharing has to be taken into account. Still, we can make some observations about these extended notions.

First, it is easy to see that the formal extension of privacy for the setting with active corruption leads to a notion that prohibits protocol faults that depend on the inputs. Hence, many weaker notions of security, such as consistency and covert security, are insufficient for privacy. At the same time, protocols without consistency checks remain private.

Consequently, it might turn out that a further decomposition of the protocols into an evaluation and a verification phase might be necessary to simplify reasoning about privacy in the active setting.

Second, it is much harder to extend the notion of output predictability, as the active adversary can alter protocol inputs. Thus, the output predictor must have some interaction with the adversary. However, the amount of the adversarial influence must be precisely limited. The current definition of output privacy can be formally extended only for robust multi-party computations where the inputs are uniquely determined regardless of adversarial actions.

Acknowledgements

This research was supported by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS, by the European Social Fund through the the ICT doctoral school, IKTDK, European Union 7th Framework Programme under grant agreement No. 284731, and by Estonian Research Council through grants IUT27-1 and IUT2-1.

References

[1] R. Canetti, "Universally Composable Security: A New Paradigm for Cryptographic Protocols," in *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, 2001, pp. 136–145.

[2] B. Pfitzmann and M. Waidner, "Composition and integrity preservation of secure reactive systems," in *Proceedings of the 7th ACM Conference on Computer and Communications Security*, 2000, pp. 245–254.

[3] U. Maurer and R. Renner, "Abstract Cryptography," in *The Second Symposium in Innovations in Computer Science, ICS 2011*, 2011, pp. 1–21.

[4] R. Küsters and M. Tuengerthal, "The IITM Model: a Simple and Expressive Model for Universal Composability," Cryptology ePrint Archive, Report 2013/025, 2013.

[5] B. Pfitzmann and M. Waidner, "A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, 2001, pp. 184–.

[6] M. Backes, B. Pfitzmann, and M. Waidner, "The reactive simulatability (RSIM) framework for asynchronous systems," *Information and Computation*, vol. 205, no. 12, pp. 1685–1720, 2007.

[7] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A Framework for Fast Privacy-Preserving Computations," in *Proceedings of the 13th European Symposium on Research in Computer Security - ESORICS'08*, vol. 5283, 2008, pp. 192–206.

[8] M. Geisler, "Cryptographic Protocols: Theory and Implementation," Ph.D. dissertation, Aarhus University, 2010.

[9] M. Burkhart, M. Strasser, D. Many, and X. A. Dimitropoulos, "SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics," in *USENIX Security Symposium*, 2010, pp. 223–240.

[10] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson, "High-performance secure multi-party computation for data mining applications," *International Journal of Information Security*, vol. 11, no. 6, pp. 403–418, 2012.

[11] D. Bogdanov, "Sharemind: programmable secure computations with practical applications," Ph.D. dissertation, University of Tartu, 2013.

[12] I. Damgård and J. B. Nielsen, "Universally Composable Efficient Multiparty Computation from Threshold Homomorphic Encryption," in *Advances in Cryptology - CRYPTO 2003*, vol. 2729, 2003, pp. 247–264.

[13] M. Backes, B. Pfitzmann, and M. Waidner, "A composable cryptographic library with nested operations," in *ACM Conference on Computer and Communications Security*, 2003, pp. 220–230.

[14] R. Küsters and M. Tuengerthal, "Universally Composable Symmetric Encryption," in *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, July 8-10, 2009*, 2009, pp. 293–307.

[15] R. Canetti and J. Herzog, "Universally Composable Symbolic Security Analysis," *Journal of Cryptology*, vol. 24, no. 1, pp. 83–147, 2011.

[16] J. Groth, R. Ostrovsky, and A. Sahai, "New Techniques for Noninteractive Zero-Knowledge," *Journal of ACM*, vol. 59, no. 3, pp. 11:1–11:35, 2012.

[17] T. Toft, "Solving linear programs using multiparty computation," in *Financial Cryptography*, vol. 5628, 2009, pp. 90–107.

[18] S. Laur, R. Talviste, and J. Willemson, "From oblivious AES to efficient and secure database join in the multiparty setting," Cryptology ePrint Archive, Report 2013/203, 2013.

[19] D. Bogdanov, P. Laud, and J. Randmets, "Domain-Polymorphic Programming of Privacy-Preserving Applications," Cryptology ePrint Archive, Report 2013/371, 2013.

[20] C. Liu, Y. Huang, E. Shi, M. Hicks, and J. Katz, “Automating Efficient RAM-Model Secure Computation,” in *35th IEEE Symposium on Security and Privacy*, 2014.

[21] O. Catrina and S. Hoogh, “Secure Multiparty Linear Programming Using Fixed-Point Arithmetic,” in *Computer Security ESORICS 2010*, 2010, vol. 6345, pp. 134–150.

[22] O. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press, 2004.