From Input Private to Universally Composable Secure Multiparty Computation Primitives

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

March 17, 2014

Abstract

Secure multiparty computation systems are commonly built form a small set of primitive components. Composability of security notions has a central role in the analysis of such systems, since 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 multiparty 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 desired outputs. We must additionally require composability meaning that these protocols should remain secure independently of the context in which it is executed. There have been different formalisations of composable security proofs: 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. Overview of RSIM framework is given in Sec. 2.

In this paper, we define privacy for the RSIM framework and show that such definition is composable. In addition, we define a restricted version of composition where some composed systems can only have one-way communication. Such a structure is natural for arithmetic and boolean circuits where the data dependency between the operations is acyclic. Finally, we show that a restricted composition of a private and a secure protocol, where all outputs of a protocol come from the secure protocol, is secure in the passive model. The exact set of conditions that are needed to achieve security in the active model is out of our scope.

There are different means of achieving secure computation, but we focus on computing on secret shared data [7, 8, 9]. Trivially privacy means that all of the secret inputs of the parties remain private and only information revealed about them is the computation output. For shared data this implies that also the shares of honest parties remain private. We give the precise security definitions in Sec. 3.

The initial idea for our composition of private and secure protocols as well as the main example result from the Sharemind secure multiparty computation framework [7, 10]. The most up to

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

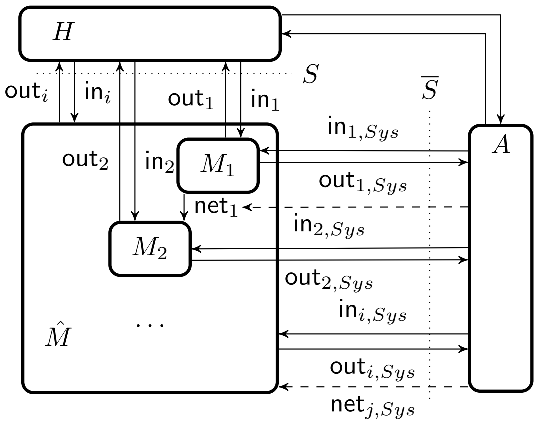

Figure 1: Canonical interface between adversary, honest user and system

date version of this approach can be found in [11]. However, current treatment is more general and more rigorous than the original exposition. This technique has also been used for obtaining other efficient secure multiparty computation protocols that are introduced in Sec. 3.3. However, previously the security of each protocol obtained in this manner had to be proven separately. Our work establishes a framework for analogous protocol compositions.

2 Reactive Simulatability

2.1 Interaction model for asynchronous reactive systems

This section gives only a high-level overview of the RSIM framework and exposes only the details that are necessary to understand our main results. Further details and detailed discussions be found in the original papers [5, 6]. 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 the user and system can interact multiple times and asynchronous means that all communication channels are asynchronous and an adversary can control their timing.

Each party in the 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 $\hat{M}$ or machines.

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 output ports. A shorthand p^c denotes a port that is complementary to p. 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 comprising 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. 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 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 some outputs to 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 C 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, but 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 $S \subseteq free(\hat{M})$. Denote $\overline{S} = free(\hat{M}) \backslash S$. Ports in S are for communicating with the environment H and ports in $\overline{S}$ are for communicating with the adversary A. We emphasise that H cannot have ports matching with S and 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. However, in this work we only consider adaptive corruption which is modelled by systems with exactly one structure and machines that accept corruption requests during their work. In the following we simplify the original theory for the cases where $Sys = (\hat{M}, S)$.

Definition 1 (Composition)**.** Structures $(\hat{M}_1, S_1)$, $\ldots$, $(\hat{M}_n, S_n)$ are *composable* if they have compatible port layouts: $ports(\hat{M}_i) \cap forb(\hat{M}_j, S_j) = \emptyset$ and $S_i \cap free([\hat{M}_j])^c = S_j^c \cap free([\hat{M}_i])$ for all $j \neq j$. Their *composition* is defined as $(\hat{M}, S) = (\hat{M}_1, S_1)|| \ldots ||(\hat{M}_n, S_n)$ with $\hat{M} = \hat{M}_1 \cup \ldots \cup \hat{M}_n$ and $S = (S_1 \cup \ldots \cup S_n) \cap free([\hat{M}])$.

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 to 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 $(s_M, I_M), (s'_M, O_M)$ as pairs of the initial state and input and the resulting state and output to the run. A *view* of a set of parties $\hat{M} \subseteq C$ is a subset of a run that corresponds to the steps relating to machines in $\hat{M}$ as is denoted by $view(\hat{M})$. Note that the view does not contain port names and therefore we are allowed to rename ports and buffers when composing systems.

Definition 2 (Simulatability)**.** Let systems Sys_1 and Sys_2 be given. We say that Sys_1 is perfectly as secure as Sys_2 ($Sys_1 \geq_{sec}^{perf} Sys_2$) 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 $view_{conf_1}(H) = view_{conf_2}(H)$.

We can give analogous definitions for statistical security ($Sys_1 \geq_{sec}^{small} Sys_2$) if all polynomial size views of H are statistically indistinguishable and computational security ($Sys_1 \geq_{sec}^{poly} Sys_2$) if we require computational indistinguishability of the views and polynomial configurations. Moreover, *as secure as* relation is transitive. We can also define simulatability for restricted classes of machines.

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 union of their ports and transition functions, and the cartesian product of their sets of states. The *simulator* Sim depends only on $\hat{M}$ and S.

Definition 3 (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 ($Sys_1 \geq_{sec}^{perf,\mathcal{A}} Sys_2$) 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 $view_{conf_1}(H) = view_{conf_2}(H)$.

The main result of RSIM is the security of composition of secure systems. The restatement of this theorem is the following.

Theorem 1 (Secure two-system composition)**.** *Assume that we have systems Sys_1, Sys_1' and Sys_2 and such that $Sys_1 \geq Sys_1'$. For structure $(\hat{M}_3, S) = (\hat{M}_1, S_1)||(\hat{M}_2, S_2)$ and $(M_1', S_1) = f_1(\hat{M}_1, S_1)$ the composition $(\hat{M}_1', S_1)||(\hat{M}_2, S_2) = (\hat{M}_4, S)$ exists and satisfies $ports(\hat{M}_1') \cap S_2^c = ports(\hat{M}_1) \cap S_2^c$. Then we have $Sys_3 \geq_{sec} Sys_4$.*

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 privacy notion to this formalisation. The composition theorem also holds for the restricted classes of simulatability.

2.3 Security proofs for passive security

Proving that a system is secure requires defining a simulator or several simulators that unify 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. For example, if something is leaked in the protocol then it should be leaked by the ideal functionality.

We assume that in case of a passive adversary, the adversary can behave as an observer, but can not modify the behaviour of corrupted parties. In the context of Fig. 1, we assume that 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 user and the adversary has to be consistent. Commonly this means that the simulator has to ensure that the view of the corrupted parties is such that it computes the same output value as defined by the corresponding ideal functionality.

3 Secure multiparty computation

We consider secure-multiparty computation based on secret shared data. Thus, all secret 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 $\mathcal{CP}_i$.

3.1 Arithmetic black box

In general the parties compute some functionality f so that each party $\mathcal{CP}_i$ has input x_i and output y_i as $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$. The function f can be expressed as an arithmetic circuit and we require secure protocols for the arithmetic primitives in order to compute f securely. Hence, in practice we need some controller application that can start and direct the computations as well as the arithmetic black box (ABB) for the computations.

The ABB was first specified in [12] with the idea that it is a computer where the majority of the parties has to agree on the functionality that is computed. The ABB defines an ideal functionality $\mathcal{F}_{\text{ABB}}$. All parties can input their private inputs for some label ℓ_x and afterwards parties can cause the computations to be performed by specifying the operand labels and the operations. $\mathcal{F}_{\text{ABB}}$ performs computations if all or majority of the parties give the same command. The $\mathcal{F}_{\text{ABB}}$ only outputs published values. Thus $\mathcal{F}_{\text{ABB}}$ is a monolithic ideal functionality, similar to the

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

```
Data Shared value $[\![u]\!]$.
ResultShared value $[\![w]\!]$ such that $w = u$, all shares $w_i$ are uniformly distributed and $u_i$ and $w_j$
are independent for $i, j = 1, 2, 3$.
  $\mathcal{CP}_1$ 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 $[\![w]\!]$.
```

celebrated 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.

3.2 Lightweight functionalities

We are interested in having lower-level functionalities for secure multiparty computation, where the ideal functionality can also use shares as inputs and outputs. The ideal functionality of a secure multiparty computation primitive $[\![y]\!] = \otimes([\![x_1]\!], \ldots, [\![x_n]\!])$ takes as inputs the shares of $x_1, \ldots, x_n$ and returns uniformly randomly chosen shares of y (if some of $x_1, \ldots, x_n$ and y are public, then plain values are input or output instead).

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. On the other hand, private ideal functionalities do not release the outputs to the adversary. As usually, a protocol is secure if it is as secure as the corresponding ideal functionality.

The corresponding real functionality defines a machine $M_i^{\otimes}$ 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 view to the adversary. For this purpose, each machine $M_i^{\otimes}$ 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.

Each system has this one structure, as commonly used for adaptive corruption in RSIM. We can use this setup to model the static corruption so that each structure has to receive a corruption request or a notification that no-one is corrupted before it starts. All machines notify the adversary about their outputs when they are computed as $(output, \ell_x)$ where ℓ_x is the label of the output. We assume that the state s recorded in the view of the machine contains all computed outputs as $(output, x, \ell_x)$ and possibly other parts of the previous protocol run. When receiving the corruption request the corresponding machine sends its current state s to A. In the following, each time it is clocked it sends its corresponding view $(s_i, I), (s_o, O)$ to the adversary. The adversary can learn all the previous outputs because they are contained in s.

Note that according to this definition, the addition protocols in most ABBs 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. This does not produce uniformly random shares.

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 $\mathsf{Reshare}()$ protocol in Algorithm 1 that for a shared input value $[\![v]\!]$ outputs a uniformly random sharing $[\![w]\!]$ of $w = v$.

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 $[\![w]\!] = [\![u]\!][\![v]\!]$.
  $[\![u']\!] \leftarrow \mathsf{Reshare}([\![u]\!])$
  $[\![v']\!] \leftarrow \mathsf{Reshare}([\![v]\!])$
  $\mathcal{CP}_1$ 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 $[\![w]\!] \leftarrow \mathsf{Reshare}([\![w']\!])$.
```

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 secure according to Def. 2.

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 [17]. Their implementation of the optimised construction proved to be twice as fast as the straightforward composition of secure protocols. Very similar idea is also used for obtaining efficient inner product for Shamir secret sharing in [18].

4 Input privacy

Analogously to the simulatabilty definition in RSIM we need to define privacy as used in secure multiparty computation. For private protocols we also have to consider correctness of the protocol corresponding to the ideal functionality.

4.1 Privacy definition

Consider an honest user H that is a composition of two machines $H = H' \cup H_\perp$ where only H' is allowed to communicate with A 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 a possible two-way communication. A *privacy configuration* is a configuration $conf_p = (\hat{M}, S, H' \cup H_\perp, A)$.

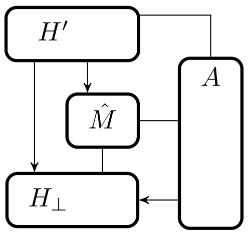

Figure 2: Privacy configuration with two distinct parts for honest user H

Definition 4 (Input privacy)**.** Let $Sys_1 = (\hat{M}_1, S)$ and $Sys_2 = (\hat{M}_2, S)$ be given. We say that $Sys_1 \geq^{perf}_{priv} Sys_2$ (perfectly at least as input private as) 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

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

We can give analogous definitions for statistical and computational simulation of the communication where the output of A_1 and A_2 has to be indistinguishable. In simple words we require that the adversary has the same view in different protocol runs. In addition, we can give this definition for restricted classes of adversaries analogously to Def. 3. 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 Theorem 6 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.

Clearly, for all systems Sys_1 and Sys_0 the relation $Sys_1 \geq^f_{sec} Sys_0$ implies $Sys_1 \geq^f_{priv} Sys_0$. Hence, if a system is as secure as another system, then it is also as private as the other system. In full privacy proof we have to specify an ideal functionality that corresponds to our protocol and prove that our protocol is as private as the ideal functionality.

In addition, a trivial composition result about independent composition holds. The composition of two systems Sys_0 and Sys_1 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

A protocol is said to be private if it is as private as a corresponding ideal functionality. The ideal functionality for the private systems is defined analogously to common ideal functionalities except that it does not give outputs to the adversary. Let us define when a structure $Sys = (\hat{M}, S)$ can be considered to be an *ideal functionality.* Our lightweight ideal functionalities from Sec. 3.2 will satisfy this definition.

Definition 5 (Ideal functionality)**.** A structure $Sys = (\hat{M}, S)$ 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 it 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 real world is private because what the corrupted party sees does not depend on the inputs of the other parties.

As an ideal functionality $Sys = (\{\mathcal{I}\}, S)$ 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 the *secure ideal functionality* described in Sec. 3.2 is specified analogously, except that in point 5 of Def. 5 it also sends all outputs of party $\mathcal{CP}_i$ out from $\mathsf{out}_I!$.

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 ideal functionality for the composed circuits from the ideal functionalities of the composed systems.

5.1 Ordered composition

For a more straightforward representation of an arithmetic circuit as a composition of protocols we define ordered composition that restricts 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 of two systems)**.** The ordered composition of two structures $Sys_1 = (M_1, S_1)$ and $Sys_2 = (M_2, S_2)$ is defined if

(i) they are composable meaning $ports(M_1) \cap forb(M_2, S_2) = \emptyset$, $ports(M_2) \cap forb(M_1, S_1) = \emptyset$ and $S_1^c \cap free([M_2]) = S_2 \cap free([M_1])^c$, and

(ii) the data flow is restricted to be in one direction only as $\forall p, q \in S_1 \cap free([M_2])^c : dir(p) = dir(q)$.

We say that the order is $Sys_1 \to Sys_2$ if $\forall p \in S_1 \cap free([M_2])^c : dir(p) =!$ meaning that the data flow is from M_1 to M_2.

We say that they are *fully ordered*, if $S_1 \cap free([M_2])^c = \{p : p \in S_1 \wedge dir(p) =!\}$ all the outputs of M_1 go to M_2.

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. 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 circuit is acyclic. We can topologically sort the circuit and define an ordering of the system based on that.

5.2 Composition of ideal functionalities

We note 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 7 (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 $Sys_1 \to Sys_2$ is definable. Let $\mathsf{in}_{Ij}?$ and $\mathsf{out}_{Ij}!$ be the ports that $\mathcal{I}_j$ uses for communicating with the adversary. Let $S_1 = S_{1\mathsf{i}} \cup S_{1\mathsf{o}}$ [resp. $S_2 = S_{2\mathsf{i}} \cup S_{2\mathsf{o}}$] be the partition on S_1 [resp. 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

- $S = S_{1\mathsf{i}} \cup (S_{2\mathsf{i}} \backslash S_{1\mathsf{o}}^c) \cup S_{2\mathsf{o}} \cup (S_{1\mathsf{o}} \backslash S_{2\mathsf{i}}^c)$;
- the partition of the ports in S among n parties follows from the partitioning of S_1 and S_2: $S^i = (S_1^i \cup S_2^i) \cap S$;
- 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;
- 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;
- 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.

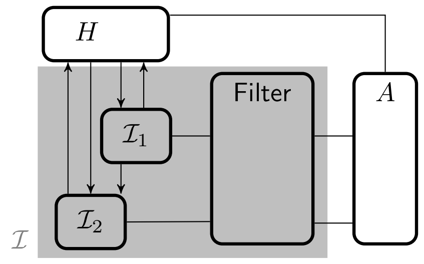

Figure 3: Composition for the ideal functionality

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 "normal" 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 multiparty 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.*

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 Predictable output

Arithmetic circuits are deterministic meaning that their inputs uniquely determine the outputs. In case of secret sharing this means that the output value is determined, but there is randomness in the outputs for individual parties because of secret sharing. We can define this property in terms of predictable outputs using the fact that the outputs of ideal systems are only dependent on the input value and not on the sharing.

To define predictability for a real system Sys_1, the corresponding ideal system $\mathcal{I}_1$, and a secure system $\mathcal{I}_2$ executed after Sys_1 or $\mathcal{I}_1$, we define the following machines and configurations in Fig. 4. Note that the compositions of Sys_1 or $\mathcal{I}_1$ with $\mathcal{I}_2$ in $conf_1$ and $conf_2$ are ordered. 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 user H (corresponding to the inputs from different computing parties $\mathcal{CP}_i$).

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 $\mathcal{I}_*$ (which will be bound with an existential quantifier in the following Def. 8).

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.

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.

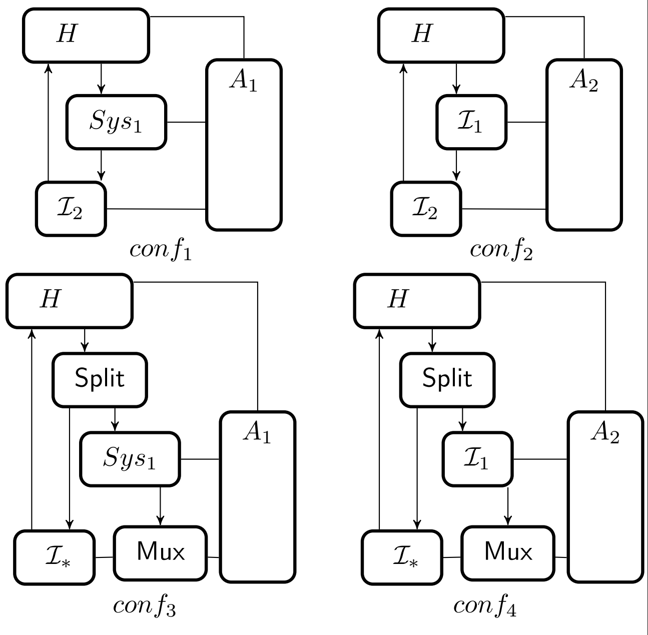

Figure 4: Configurations for predictable output

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 $(output, \ell_x, x, i)$ 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 8. We say that the compositions $Sys_1 \to \mathcal{I}_2$ and $\mathcal{I}_1 \to \mathcal{I}_2$ have *jointly predictable outcome* if there exists a machine $\mathcal{I}_*$ (the "output predictor") such that for the following configurations on Fig. 4 we have $view_{conf_1}(H) = view_{conf_3}(H)$ and $view_{conf_2}(H) = view_{conf_4}(H)$.

Note that this condition is trivially satisfied for our correct protocols and we can obtain suitable $\mathcal{I}_*$ by composition of $\mathcal{I}_1$ and $\mathcal{I}_2$ (with some extra functionality for scheduling) according to Def. 7. Also note that jointly predictable outcome ensures that Sys_1 "computes the same thing" as $\mathcal{I}_1$.

5.4 Composition theorems

Theorem 2 (Black-box privacy composition, informal)**.** *The ordered composition of black-box private protocols is private.*

Theorem 3 (Secure composition, informal)**.** *The fully ordered composition of black-box private and black-box simulatable protocols with jointly predictable outcome is black-box simulatable.*

The formal versions of these theorems are stated and proven in Sec. 7.

6 Simulators

For proving theorems 2 and 3, we have to construct simulators for composed systems. These constructions are more complex and invasive than in the proof of theorem 1, and in this section, we set up some definitions for combining the simulators we have from the premises of the theorems.

6.1 Privacy simulator

Recall that a black-box *privacy simulator* for ideal structure $Id = (\hat{M}_2, S)$ and real structure $RS = (\hat{M}_1, S)$ 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. The simulator satisfies $view_{conf_1}(H') = view_{conf_2}(H')$ for any $H = H' \cup H_\perp$ and A, such that $conf_1 = (\hat{M}_1, S, H, A)$ and $conf_2 = (\hat{M}_2, S, H, \mathsf{Sim}^{Id,RS} \cup A)$ are privacy configurations. 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 is in complete control of scheduling in RS. In particular, the simulator writes into at most one $\mathsf{out}_{i,Sys}$ during each invocation.

6.2 Extended simulator

An extended simulator additionally computes the outputs of corrupted parties. When composing the structures, these outputs are needed to be given as inputs to the simulator(s) of the next stage(s) of the composition. In the next definition, let Sink be a machine with ports $\mathsf{output}_i?$, $\mathsf{foutput}_i?$ for $i \in [n] = \{1, \ldots, n\}$. Let Sink' be a machine with ports $\mathsf{output}_i?$, $\mathsf{output}_i!$ for $i \in [n]$. Both machines have trivial behaviour, as there cannot be any channels from these machines to other ones.

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 $i \in [n]$. 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 $\mathcal{O}(\tau) = (m_1, \ldots, m_n)$, where 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). Note that out_i is the channel that takes the outputs of the system to H or the next system in composition. Let $\mathcal{O}'(\tau) = (m_1, \ldots, m_n)$, where m_i is $\perp$, if there was no corruption request on $\mathsf{in}_{i,Sys}$. If there was such a request, then 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 $\mathcal{O}_{conf}$ [resp. $\mathcal{O}'_{conf}$] be the distribution of $\mathcal{O}(\tau)$ [resp. $\mathcal{O}'(\tau)$] over all possible runs of $conf$.

Definition 9 (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 $\{\mathsf{output}_i!, \mathsf{foutput}_i! \mid i \in [n]\}$;
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 on $\mathsf{foutput}_i$ and clocks that channel;
4. never outputs on $\mathsf{foutput}_i$, except in the case described before;
5. is a simulator: $view_{conf_1}(H') = view_{conf_2}(H')$ for any $H = H' \cup H_\perp$ and A, such that $conf_1 = (\hat{M}, S, H, A \cup \mathsf{Sink}')$ and $conf_2 = (\mathcal{I}, S, H, \mathsf{ExtSim}^{Id,RS} \cup A \cup \mathsf{Sink})$ are privacy configurations;
6. 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 2 (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 giving it control over the clocking of 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 we 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 works as follows:

1. on input $(output, \ell_x)$ from $\mathsf{extr}_i?$ it saves ℓ_x as a label from i to M and forwards the message out on $\mathsf{output}_i!$;
2. on input $(output, x, \ell_x)$ from $\mathsf{extr}_i?$ it works as follows:
   (a) if the label ℓ_x has be stored in M for party i then this input is forwarded on $\mathsf{foutput}_i!$, ℓ_x is removed from M for i and the channel $\mathsf{foutput}_i$ is clocked,
   (b) 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 $i \in [n]$), see Fig. 5. Clearly, it satisfies the structural properties 1, 2 and 4 of Def. 9. It also satisfies 3rd property due to the manner 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 6th 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. $\square$

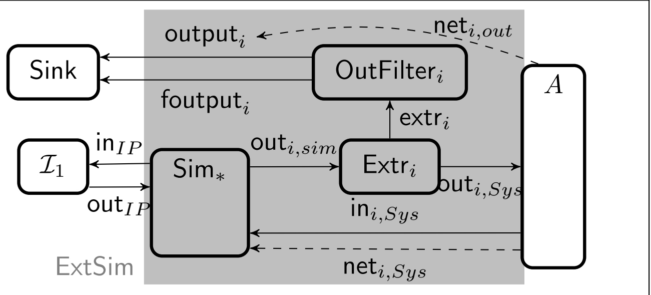

Figure 5: Extended simulator construction with the ideal world functionality, adversary and connection to the rest of the simulator

7 Security of composed protocols

In this section we prove that ordered composition of private and secure protocols is secure. For this we at first show that we can limit the class of adversaries (Thm. 4) and then use this new class to show the security of the composed protocol (Thm. 5). From these results we know that our composition is secure with respect to general adversaries.

7.1 Restricted adversary model

Analysis of ordered compositions can be simplified by restricting 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, i.e., do not send or receive any messages between corruption calls. Then it is easy to see that these restrictions are not limiting.

Theorem 4. *Let two systems Sys_1 and Sys_2 be composed in the ordered composition $Sys_1 \to Sys_2$. Then the composed system is perfectly 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. For that we have to show that the original system is as secure as the composed system with the construction. The construction is shown Fig. 6 and introduces two small machines that work as follows.

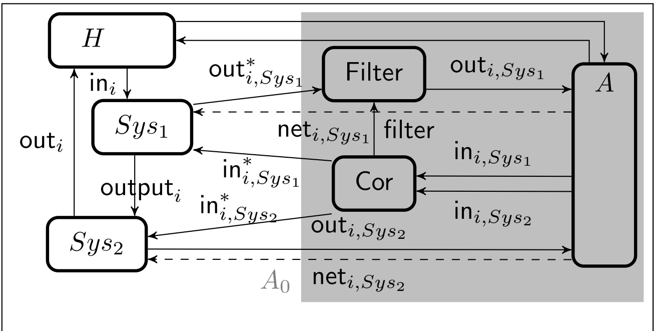

Figure 6: Construction for the restricted adversary

A simple stateless machine Cor assures that a party is always corrupted in Sys_1 before than 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 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 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 $\mathcal{I}_c$ 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 filter? is for updating the list $\mathcal{I}_c$. If an input $(corrupt, i)$ is written to filter? then i is added to $\mathcal{I}_c$ 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 clocking signals so that the list $\mathcal{I}_c$ 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 trivial 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 perfectly 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 how to construct a restricted adversary for $A \in \mathcal{A}_0$. The corresponding construction is analogous to the proof of Theorem 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 consider the more intuitive adversaries of class $\mathcal{A}_1$ in the following composition theorems. Note that this is suitable for cases of static or adaptive corruption, but not for modelling mobile corruption.

7.2 Secure composition

In this section we propose the main theorem that proves 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 5. *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 $Sys_1 \to Sys_2 \geq_{sec}^{model,\mathcal{A}_1} \mathcal{I}$ in 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. Due to the black-box simulatability $Sys_2 \geq_{sec}^{model} \mathcal{I}_2$ and Thm. 1 we have $view_{conf_0}(H) \approx view_{conf_1}(H)$ (see Fig. 8) for some simulator Sim_2.

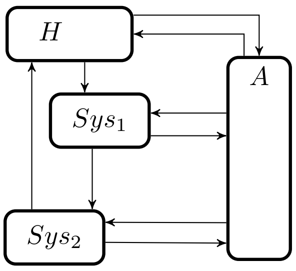

Figure 7: Composition of two real systems, $conf_0$

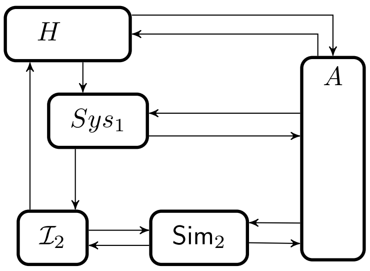

Figure 8: Composition of a real and an ideal system, $conf_1$

Next step of the proof uses the predictable outcome (Def. 8) of $\mathsf{Sys}_1 \to \mathcal{I}_2$. We consider Sim_2 as a part of the adversary and obtain $view_{conf_1}(H) = view_{conf_2}(H)$ (see Fig. 9). Recall that we have joint predictability, hence the same $\mathcal{I}_*$ also demonstrates the predictable outcome of $\mathcal{I}_1 \to \mathcal{I}_2$.

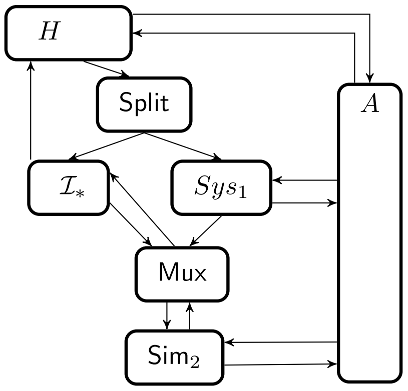

Figure 9: Splitting the composition to two independent parts, $conf_2$

As next we 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 Figure 10. For the machine H trivially $view_{conf_2}(H) = view_{conf_3}(H)$. The scheduling does not change.

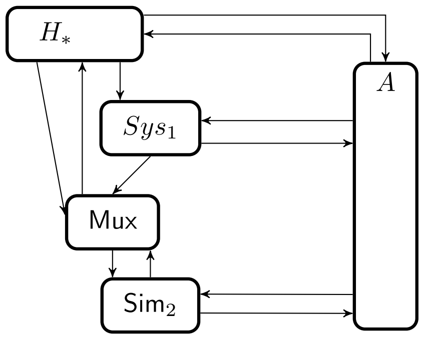

Figure 10: Regrouping the machines, $conf_3$

Next we use the assumption about adversary in class $\mathcal{A}_1$. Namely, we assume that in both systems 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 that does not receive 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 general case this step can still be done by introducing the Extr 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 $H_{*\perp}$ the 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, $view_{conf_3}(H_*) = view_{conf_4}(H_*)$.

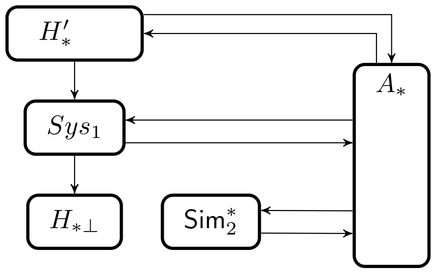

Figure 11: Splitting the Mux by its two roles, $conf_4$

Finally we finish with another cosmetic change to join A_* and Sim_2 to A_{**} as shown on Fig. 12. Trivially, $view_{conf_4}(H_*) = view_{conf_5}(H_*)$.

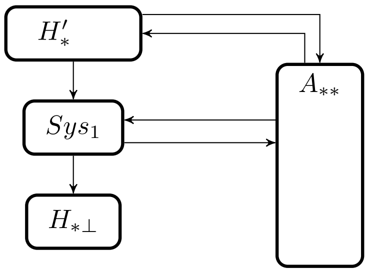

Figure 12: Redefining the adversary, $conf_5$

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.

So far we analysed the composition $Sys_1 \to Sys_2$, as a second step we have to analyse the ideal functionality $\mathcal{I}$ in the same setting. 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. We claim that a combination of machines ExtSim_1 (the existence of which is shown in Lemma 2), Sim_2 and $\mathsf{Mux_t}$ defined below serves as a suitable simulator that proves the theorem. The configuration $conf'_1$ with these machines is depicted in Fig. 13.

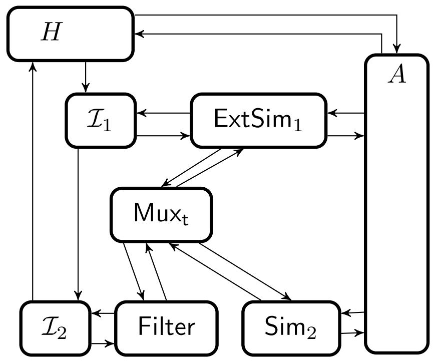

Figure 13: Sys_1 replaced by the ideal system, $conf'_1$

The machine $\mathsf{Mux_t}$ 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 $\mathcal{I}_*$. The scheduling is fixed by the description of individual machines.

We consider A, ExtSim, $\mathsf{Mux_t}$, Filter and Sim_2 as an adversary in Def. 8 to obtain the composition $conf'_2$ analogous to $conf_2$. 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. 8 and $conf'_1$.

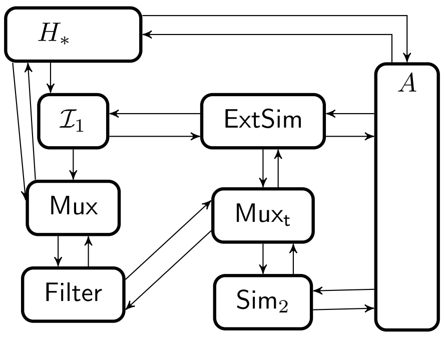

Figure 14: Introduced intermediate machines, $conf'_3$

This results in a quite complicated composition of small machines. At first, note that we can discard Filter which only effects the input values that move out from Mux, but by definition $\mathsf{Mux_t}$ does not do anything with inputs from Filter. The machine Mux has two roles, it can either forward the inputs of the second composed party based on the outputs of $\mathcal{I}_2$ or the outputs of the composition it gets from H_*. For the same reasons as we discarded 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 already.

As the final step we decompose ExtSim to its parts Sim_1, Extr and OutFilter. 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 $view_{conf'_0}(H) \approx view_{conf'_5}(H)$ 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

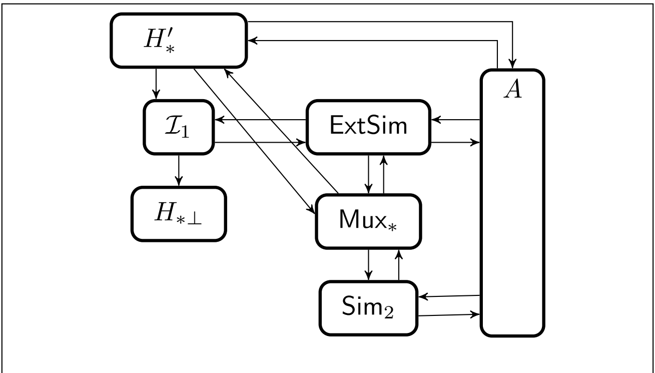

Figure 15: Simplification of $conf'_3$, $conf''_3$

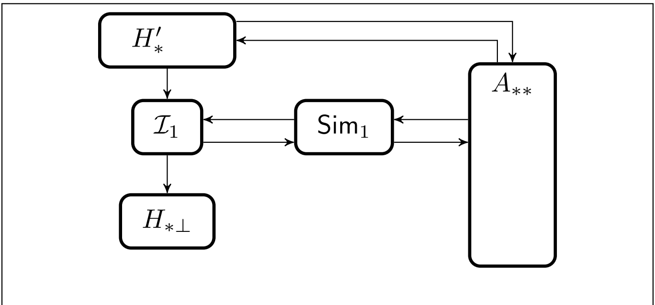

Figure 16: Final configuration, $conf'_5$

trivial as the machines Split and $\mathcal{I}_*$ are the same due to joint predictability (Def. 8). The same 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 $conf'_5$. We can use the final state $conf_5$ as a privacy configuration and we know that by definition Sys_1 is as private as $\mathcal{I}_1$ and that for each adversary A we can use the same simulator Sim$_1$. Hence, $view_{conf_5}(H) \approx view_{conf'_5}(H)$. □

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 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. Direct result from Thm. 4 and 5. □

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 6. *Let $Sys_1 \geq_{priv}^{model} \mathcal{I}_1$ and $Sys_2 \geq_{priv}^{model} \mathcal{I}_2$ in 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 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. For simplicity of exposition we assume that all inputs are inputs to Sys_1, extending this to independent inputs for Sys_2 is trivial modification. We can assume that all inputs that Sys_1 does not use are also its outputs. Therefore, we can consider full ordered composition for simplicity and the step to just ordered composition is trivial.

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 $conf_0$. We can do the step to 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 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 an analogous situation to Fig. 8 with the difference that output of $\mathcal{I}_2$ goes to $H_\perp$.

On the other hand we want to consider a configuration $conf'_0$ with $\mathcal{I}$, H and A, and some simulator. 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 passed through. As the simulator, we propose the combination of machines Sim_1, Extr, OutFilter, Mux and Sim_2, let $conf_s$ be the resulting configuration (see Fig. 17).

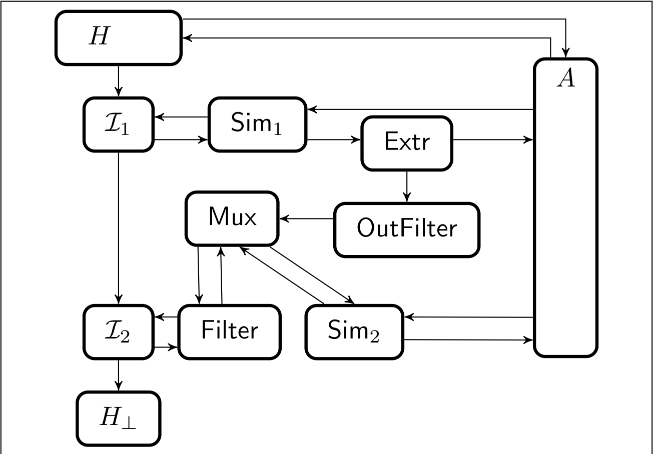

Figure 17: Simulator construction for the $\mathcal{I}$ and $Sys_1 \rightarrow Sys_2$, $conf_s$

Note that the collection Sim_1, Extr and OutFilter is actually ExtSim_1. In addition, Mux works as described in Sec. 5.3 except of the special scheduling.

We consider fully ordered composition, therefore all inputs of $\mathcal{I}_2$ are stopped by the Filter and the private ideal functionality $\mathcal{I}_2$ does not give outputs, therefore 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, 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 Sys_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. □

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

Proof. Direct result from Thm. 4 and 6. □

7.4 Applicability of the composition theorems

In general, we can define the some composed private protocol using the composability of the privacy notion based on Thm. 6 and especially Cor. 3. In addition, we know that all secure systems are also private and we can include secure systems in this part of the protocol. In order to achieve universally composable secure protocols we need to finish the composed private protocol by a secure protocol.

Based on the composability of security (Thm. 1) we can define the finishing part of the protocol as a composition of secure protocols. The main restriction of the fully ordered composition in Thm. 5 and 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.

In the context of secure multiparty computation it is most reasonable to consider private protocols for arithmetic operations. The 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 privacy requirement is sufficient for most secure computation primitives in the passive security model in order to obtain universally composable secure multiparty computation protocols. Private protocols are often more efficient, while their composition is no more complex than composing secure protocols. We can therefore 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 secure finalising step, which can be very simple.

We believe our ideas are applicable also in the case of active adversaries against secure multiparty 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. It is likely that we have pushed most of these difficulties into the definition of predictability, reducing them to standard arguments about the correctness of protocol designs for particular arithmetic operations.

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*, ser. FOCS ’01. Washington, DC, USA: IEEE Computer Society, 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*, ser. CCS ’00. New York, NY, USA: ACM, 2000, pp. 245–254.

[3] U. Maurer and R. Renner, “Abstract cryptography,” in *The Second Symposium in Innovations in Computer Science, ICS 2011*, B. Chazelle, Ed. Tsinghua University Press, Jan. 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*, ser. SP ’01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 184–.

[6] M. Backes, B. Pfitzmann, and M. Waidner, “The reactive simulatability (rsim) framework for asynchronous systems,” *Inf. Comput.*, 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*, ser. Lecture Notes in Computer Science, S. Jajodia and J. Lopez, Eds., vol. 5283. Springer Berlin / Heidelberg, 2008, pp. 192–206.

[8] M. Geisler, “Cryptographic protocols: Theory and implementation,” Ph.D. dissertation, Aarhus University, February 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.* USENIX Association, 2010, pp. 223–240.

[10] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson, "High-performance secure multi-party computation for data mining applications," *Int. J. Inf. Sec.*, 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, http://hdl.handle.net/10062/29041.

[12] I. Damgård and J. B. Nielsen, "Universally composable efficient multiparty computation from threshold homomorphic encryption," in *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, ser. Lecture Notes in Computer Science, vol. 2729. Springer, 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*, S. Jajodia, V. Atluri, and T. Jaeger, Eds. ACM, 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.* IEEE Computer Society, 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] 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, http://eprint.iacr.org/.

[18] O. Catrina and S. Hoogh, "Secure multiparty linear programming using fixed-point arithmetic," in *Computer Security ESORICS 2010*, ser. Lecture Notes in Computer Science, D. Gritzalis, B. Preneel, and M. Theoharidou, Eds. Springer Berlin Heidelberg, 2010, vol. 6345, pp. 134–150.