An Equational Approach to Secure Multi-party Computation*

Daniele Micciancio† Stefano Tessaro‡

January 12, 2013

Abstract

We present a novel framework for the description and analysis of secure computation protocols that is at the same time mathematically rigorous and notationally lightweight and concise. The distinguishing feature of the framework is that it allows to specify (and analyze) protocols in a manner that is largely independent of time, greatly simplifying the study of cryptographic protocols. At the notational level, protocols are described by systems of mathematical equations (over domains), and can be studied through simple algebraic manipulations like substitutions and variable elimination. We exemplify our framework by analyzing in detail two classic protocols: a protocol for secure broadcast, and a verifiable secret sharing protocol, the second of which illustrates the ability of our framework to deal with probabilistic systems, still in a purely equational way.

1 Introduction

Secure multiparty computation (MPC) is a cornerstone of theoretical cryptography, and a problem that is attracting increasingly more attention in practice too due to the pervasive use of distributed applications over the Internet and the growing popularity of computation outsourcing. The area has a long history, dating back to the seminal work of Yao [29] in the early 1980s, and a steady flow of papers contributing extensions and improvements that lasts to the present day (starting with the seminal works [12, 6, 3] introducing general protocols, and followed by literally hundreds of papers). But it is fair to say that MPC has yet to deliver its full load of potential benefits both to the applied and theoretical cryptography research communities. In fact, large portions of the research community still see MPC as a highly specialized research area, where only the top experts can read and fully understand the highly technical research papers routinely published in mainstream crypto conferences. Two main obstacles have kept, so far, MPC from becoming a more widespread tool to be used both in theoretical and applied cryptography: the prohibitive computational cost of executing many MPC protocols, and the inherent complexity of the models used to describe the protocols themselves. Much progress has been made in improving the efficiency of the first

*An edited version of this work appears in the proceedings of Innovations in Theoretical Computer Science, ITCS 2013. This is the authors' copy. This material is based on research sponsored by DARPA under agreement number FA8750-11-C-0096. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

†University of California, San Diego. daniele@cs.ucsd.edu.

‡Massachusetts Institute of Technology. tessaro@csail.mit.edu

protocols [29, 12, 6] in a variety of models and with respect to several complexity measures, even leading to concrete implementations (cf. e.g. [19, 4, 17, 7, 24, 11]). However, the underlying models to describe and analyze security properties are still rather complex.

What makes MPC harder to model than traditional cryptographic primitives like encryption, is the inherently distributed nature of the security task being addressed: there are several distinct and mutually distrustful parties trying to perform a joint computation, in such a way that even if some parties deviate from the protocol, the protocol still executes in a robust and secure way.

The difficulty of properly modeling secure distributed computation is well recognized within the cryptographic community, and documented by several definitional papers attempting to improve the current state of the art [22, 9, 10, 23, 2, 18, 15].Unfortunately, the current state of the art is still pretty sore, with definitional/modeling papers easily reaching encyclopedic page counts, setting a very high barrier of entry for most cryptographers to contribute or actively follow the developments in MPC research. Moreover, most MPC papers are written in a semi-formal style reflecting an uncomfortable trade-off between the desire of giving to the subject the rigorous treatment it deserves and the implicit acknowledgment that this is just not feasible using the currently available formalisms (and even more so, within the page constraints of a typical conference or even journal publication.) The goal of this paper is to drastically change this state of affairs, by putting forward a model for the study of MPC protocols that is both concise, rigorous, and still firmly rooted in the intuitive ideas that pervade most past work on secure computation and most cryptographers know and love.

An interesting line of work, with goals similar to ours, is described in a recent paper of Mauer and Renner [20]. The works share the common goal of providing a simpler and mathematically rigorous framework for studying cryptographic protocols, but are fundamentally different. The main difference between [20] and our work is that [20] adopts an axiomatic approach (postulating that abstract cryptographic objects satisfy certain natural and desirable properties), while our line of work is model theoretic, offering a (high level, but) concrete model of computation where cryptographic protocols can be directly described by standard mathematical objects. The two approaches are complementary, as the computational model described in this paper satisfies all the natural and desirable properties postulated in [20]. So, one may use the axiomatic framework of [20] to prove some of the properties of protocols specified in our model. It would be interesting to refine the set of axioms proposed in [20] into a sound and complete axiomatization of the model described in this work, in order to show that the two frameworks are equally powerful.

1.1 The simulation paradigm

Let us recall the well known and established simulation paradigm that underlies essentially all MPC security definitions. Cryptographic protocols are typically described by several component programs $P_1, \ldots, P_n$ executed by n participating parties, interconnected by a communication network N, and usually rendered by a diagram similar to the one in Figure 1 (left): each party receives some input x_i from an external environment, and sends/receives messages s_i, r_i from the network. Based on the external inputs x_i, and the messages s_i, r_i transmitted over the network N, each party produces some output value y_i which is returned to the outside world as the visible output of running the protocol. The computational task that the protocol is trying to implement is described by a single monolithic program F, called the "ideal functionality", which has the same input/output interface as the system consisting of $P_1, \ldots, P_n$ and N, as shown in Figure 1 (right). Conceptually, F is executed by a centralized entity that interacts with the individual parties through their local

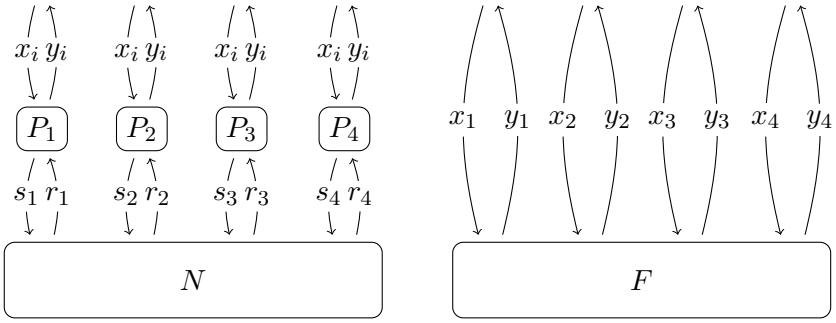

Figure 1: A multiparty protocol $P_1, \ldots, P_4$ with communication network N (left) implementing a functionality F (right).

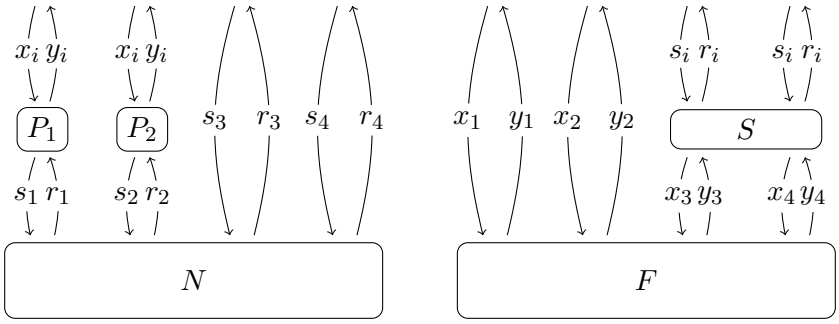

Figure 2: Simulation based security. The protocol $P_1, \ldots, P_4$ is a secure implementation of functionality F if the system (left) exposed to an adversary that corrupts a subset of parties (say P_3 and P_4) is indistinguishable from the one system (right) recreated by a simulator S interacting with F.

input/output interfaces x_i/y_i, and processes the data in a prescribed and trustworthy manner. A protocol $P_1, \ldots, P_n$ correctly implements functionality F in the communication model provided by N if the two systems depicted in Figure 1 (left and right) exhibit the same input/output behavior.

Of course, this is not enough for the protocol to be secure. In a cryptographic context, some parties can get corrupted, in which case an adversary (modeled as part of the external execution environment) gains direct access to the parties' communication channels s_i, r_i and is not bound to follow the instructions of the protocol programs P_i. Figure 2 (left) shows an execution where P_3 and P_4 are corrupted. The simulation paradigm postulates that whatever can be achieved by a concrete adversary attacking the protocol, can also be achieved by an idealized adversary S (called the simulator) attacking the ideal functionality F. In Figure 2 (right), the simulator takes over the role of P_3 and P_4, communicating for them with the ideal functionality, and recreating the attack of a real adversary by emulating an interface that exposes the network communication channels of P_3 and P_4. The protocol $P_1, \ldots, P_n$ securely implements functionality F if the systems described on the left and right of Figure 2 are functionally equivalent: no adversary (environment) connecting

to the external channels $x_1, y_1, x_2, y_2, s_3, r_3, s_4, r_4$ can (efficiently) determine if it is interacting with the system described in Figure 2 (left) or the one in Figure 2 (right). In other words, anything that can be achieved corrupting a set of parties in a real protocol execution, can also be emulated by corrupting the same set of parties in an idealized execution where the protocol functionality F is executed by a trusted party in a perfectly secure manner.

This is a very powerful idea, inspired by the seminal work on zero knowledge proof systems [13], and embodied in many subsequent papers about MPC. But, of course, as much as evocative the diagrams in Figure 2 may be, they fall short of providing a formal definition of security. In fact, a similar picture can be drawn to describe essentially any of the secure multiparty computation models proposed so far at a very abstract level, but the real work is in the definition of what the blocks and communication links connecting them actually represent. Traditionally, building on classical work from computational complexity on interactive proof systems, MPC is formalized by modeling each block by an interactive Turing machine (ITM), a venerable model of sequential computation extended with some "communication tapes" used to model the channels connecting the various blocks. Unfortunately, this only provides an adequate model for the local computation performed by each component block, leaving out the most interesting features that distinguish MPC from simpler cryptographic tasks: computation is distributed and the concurrent execution of all ITMs needs to be carefully orchestrated. In a synchronous communication environment, where local computations proceed in lockstep through a sequence of rounds, and messages are exchanged only between rounds, this is relatively easy. But in asynchronous communication environments like the Internet, dealing with concurrency is a much trickier business. The standard approach to deal with concurrency in asynchronous distributed systems is to use nondeterminism: a system does not describe a single behavior, but a set of possible behaviors corresponding to all possible interleavings and message delivery orders. But nondeterminism is largely incompatible with cryptography, as it allows to break any cryptographic function by nondeterministically guessing the value of a secret key. As a result, cryptographic models of concurrent execution resort to an adversarially and adaptively chosen, but deterministic, message delivery order: whenever a message is scheduled for transmission between two component, it is simply queued and an external scheduling unit (which is also modeled as part of the environment) is notified about the event. While providing a technically sound escape route from the dangers of mixing nondeterministic concurrency with cryptography, this approach has several shortcomings:

- Adding a scheduler further increases the complexity of the system, making simulation based proofs of security even more technical.

- It results in a system that in many respects seems overspecified: as the goal is to design a robust system that exhibits the prescribed behavior in any execution environment, it would seem more desirable to abstract the scheduling away, rather than specifying it in every single detail of a fully sequential ordering of events.

- Finally, the intuitive and appealing idea conveyed by the diagrams in Figure 2 is in a sense lost, as the system is now more accurately described by a collection of isolated components all connected exclusively to the external environment that orchestrates their executions by scheduling the messages.

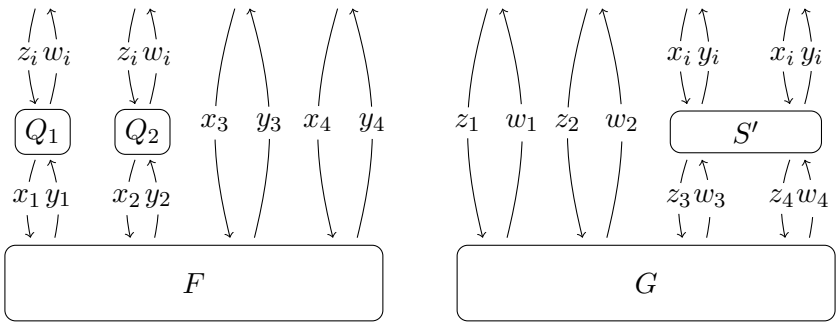

Figure 3: A protocol Q_i implementing G in the F-hybrid model.

1.2 Our work

In this paper we describe a model of distributed computation that retains the simplicity and intuitiveness conveyed by the diagrams in Figures 1 and 2, and still it is both mathematically rigorous and concise. In other words, we seek a model where the components P_i, N, F, S occurring in the description and analysis of a protocol, and the systems obtained interconnecting them, can be given a simple and precise mathematical meaning. The operation of composing systems together should also be well defined, and satisfy a number of useful and intuitive properties, e.g., the result of connecting several blocks together does not depend on the order in which the connections are made. (Just as we expect the meaning of a diagram to be independent of the order in which the diagram was drawn.) Finally, it should provide a solid foundation for equational reasoning, in the sense that equivalent systems can be replaced by equivalent systems in any context.

Within such a framework, the proof that protocols can be composed together should be as simple as the following informal argument. (In fact, given the model formally defined in the rest of the paper, the following is actually a rigorous proof that our definition satisfies a universal composability property.) Say we have a protocol $P_1, \ldots, P_n$ securely implementing ideal functionality F using a communication network N, and also a protocol $Q_1, \ldots, Q_n$ in the F-hybrid model (i.e., an idealized model where parties can interact through functionality F) that securely implements functionality G. The security of the second protocol is illustrated in Figure 3.

Then, the protocol obtained simply by connecting P_i and Q_i together is a secure implementation of G, in the standard communication model N. Moreover, the simulator showing that the composed protocol is secure is easily obtained simply by composing the simulators for the two component protocols. In other words, we want to show that an adversary attacking the real system described in Figure 4 (left) is equivalent to the composition of the simulators attacking the ideal functionality G as described in Figure 4 (right).

This is easily shown by transforming Figure 4 (left) to Figure 4 (right) in two steps, going through the hybrid system described in Figure 5. Specifically, first we use the security of P_i to replace the system described in Figure 2 (left) with the one in Figure 2 (right). This turns the system in Figure 4 (left) into the equivalent one in Figure 5. Next we use the security of Q_i to substitute the system in Figure 3 (left) with the one in Figure 3 (right). This turns Figure 5 into Figure 4 (right).

While the framework proposed in this paper allows to work with complex distributed systems

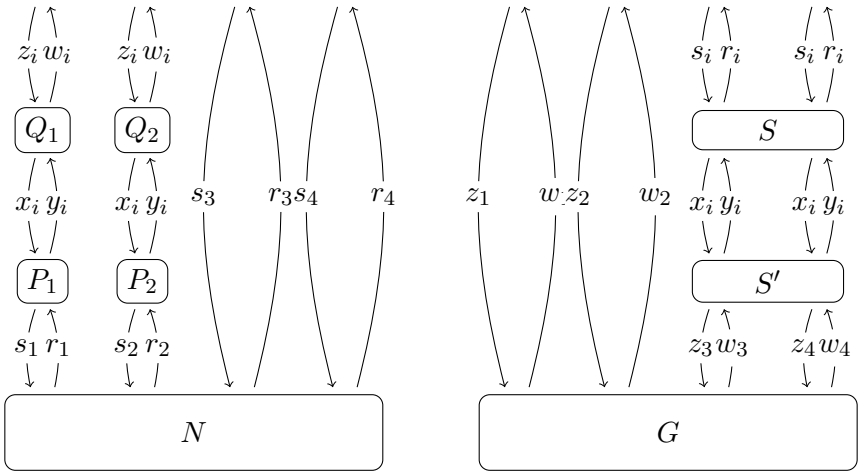

Figure 4: Protocol composition. Security is proved using a hybrid argument.

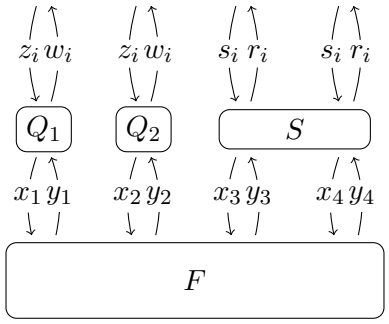

Figure 5: Hybrid system to prove the security of the composed protocol

with the same simplicity as the informal reasoning described in this section, it is quite powerful and flexible. For example, it allows to model not only protocols that are universally composable, but also protocols that retain their security only when used in restricted contexts. For simplicity, in this paper we focus on *perfectly secure* protocols against *unbounded* adversaries, as this already allows us to describe interesting protocols that illustrate the most important feature of our framework: the ability to design and analyze protocols without explicitly resorting to the notion of time and sequential scheduling of messages. Moreover, within the framework of universally composability, it is quite common to design perfectly secure protocols in a hybrid model that offers idealized versions of the cryptographic primitives, and then resorting to computationally secure cryptographic primitives only to realize the hybrid model. So, a good model for the analysis of perfect or statistical security can already be a useful and usable aid for the design of more general computationally secure protocols. Natively extending our framework to statistically or computationally secure protocols is also an attractive possibility. We consider the perfect/statistical/computational security dimension as being mostly orthogonal to the issues dealt with in this paper, and we believe the model described here offers a solid basis for extensions in that direction.

1.3 Techniques

In order to realize our vision, we introduce a computational model in which security proofs can be carried out without explicitly dealing with the notion of time. Formally, we associate to each communication channel connecting two components the set of all possible "channel histories", partially ordered according to their information content or temporal ordering. The simplest example is the set of all finite sequences M^* of messages from some underlying message space, ordered according to the prefix ordering relation. The components of the system are then modeled as functions mapping input histories to output histories. The functions are subject to some natural conditions, e.g., monotonicity: receiving more input values can only results in more output values being produced. Under appropriate technical conditions on the ordered sets associated to the communication channels, and the functions modeling the computations performed by the system components, this results in a well behaved framework, where components can be connected together, even forming loops, and always resulting in a unique and well defined function describing the input/output behavior of the whole system. Previous approaches to model interactive systems, such as Kahn networks [16] and Maurer's random systems [21], can indeed be seen as special cases of our general process model.[1] The resulting model is quite powerful, allowing even to model probabilistic computation as a special case. However, the simplicity of the model has a price: all components of the system must be monotone with respect to the information ordering relation. For example, if a program P on input messages x_1, x_2 outputs $P(x_1, x_2) = (y_1, y_2, y_3)$, then on input x_1, x_2, x_3 it can only output a sequence of messages that extends (y_1, y_2, y_3) with more output. In other words, P cannot "go back in time" and change y_1, y_2, y_3. While this is a very natural and seemingly innocuous restriction, it also means that the program run by P cannot perform operations of the form "if no input message has been received yet, then send y". This is because if an input message is received at a later point, P cannot go back in time and not send y.

[1] For the readers well versed in the subject, we remark that our model can be regarded as a generalization of Kahn networks where the channel behaviors are elements of arbitrary partially ordered sets (or, more precisely, domains) rather than simple sequences of messages. This is a significant generalization that allows to deal with probabilistic computations and intrinsically nondeterministic systems seamlessly, without incurring into the Brock-Ackerman anomaly and similar problems.

It is our thesis that these time dependent operations make cryptographic protocols harder to understand and analyze, and therefore should be avoided whenever possible.

Organization. The rest of the paper is organized as follows. In Section 2 we present our framework for the description and analysis of concurrent processes, and illustrate the definitions using a toy example. Next, we demonstrate the applicability of the framework by carefully describing and analyzing two classic cryptographic protocols: secure broadcast (in Section 3) and verifiable secret sharing (in Section 4). The secure broadcast protocol in Section 3 is essentially the one of Bracha, and only uses deterministic functions. Our modular analysis of the protocol illustrates the use of subprotocols that are not universally composable. The verifiable secret sharing protocol analyzed in Section 4 provides an example of randomized protocol.

2 Distributed Systems, Composition, and Secure Computation

In this section we introduce our mathematical framework for the description and analysis of distributed systems. We start with a high level description of our approach, which will be sufficient to apply our framework and to follow the proofs. We then give more foundational details justifying soundness of our approach. Finally, we provide security definitions for protocols in our framework.

2.1 Processes and systems

Introducing processes: An example and notational conventions. Our framework models (asynchronous and reactive) processes and systems with one or more input and output *channels* as mathematical functions mapping input histories to output histories. Before introducing more formal definitions, let us illustrate this concept with a simple example. Consider a deterministic process with one input and one output channels, which receives as input a sequence of messages, $x[1], \ldots, x[k]$, where each $x[k]$ is (or can be parsed as) an integer. The process receives the messages sequentially, one at a time, and in order to make the process finite one may assume that the process will accept only the first n messages. Upon receiving each input message $x[i]$, the process increments the value and immediately outputs $x[i]+1$. It is not hard to model the process in terms of a function mapping input to output sequences: The input and output of the function modeling the process are the set $\mathbb{Z}^{\leq n}$ of integer sequences of length at most n, and the process is described by the function $\mathrm{F}\colon \mathbb{Z}^{\leq n} \to \mathbb{Z}^{\leq n}$ mapping each input sequence $x \in \mathbb{Z}^k$ (for some $k \leq n$) to the output sequence $y \in \mathbb{Z}^k$ of the same length defined by the equations $y[i] = x[i] + 1$ (for $i = 1, \ldots, k$). There are multiple ways one can possibly describe such a function. We describe the process in equational form as in Figure 6 (left). In the example, the first line assigns names to the function, input and output variables, while the remaining lines are equations that define the value of the output variables in terms of the input variables. Each variable ranges over a specific set ($x, y \in \mathbb{Z}^{\leq n}$), but for simplicity we often leave the specification of this set implicit, as it is usually clear from the context. By convention, all variables that appear in the equations, but not as part of the input or output variables, are considered *local/internal* variables, whose only purpose is to help defining the value of the output variables in terms of the input variables. Free index variables (e.g., i, j) are universally quantified (over appropriate ranges) and used to compactly describe sets of similar equations.

$\mathrm{F}(x) = y$:	$\mathrm{G}(y) = (z, w)$:	$\mathrm{H}(z) = x$:
$y[i] = x[i]+1 \ \ (i = 1, \ldots, \lvert x \rvert)$	$z = y$	$x[1] = 1$
	$w = y$	$x[j+1] = z[j] \qquad (j \leq \min\{\lvert z \rvert, n-1\})$

Figure 6: Some simple processes

Processes as monotone functions. In general, the reason we define a process F as a function mapping sequences to sequences[2] (rather than, say, as a function $f(x) = x + 1$ applied to each incoming message x) is that it allows to describe the most general type of (e.g., stateful, reactive) process, whose output is a function of all messages received as input during the execution of the protocol. (Note that we do not need to model state explicitly.) Also, such functions can describe processes with multiple input and output channels by letting inputs and outputs be *tuples* of message sequences. However, clearly, not any such function mapping input to output sequences can be a valid process. To capture valid functions representing a process, input and output sets are endowed with a partial ordering relation $\leq$, where $x \leq y$ means that y is a possible *future* of x. (In the case of sequences of messages, $\leq$ is the standard prefix partial ordering relation, where $x \leq y$ if $y = x|z$ for some other sequence z, and $x|z$ is the concatenation of the two sequences.) Functions describing processes should be naturally restricted to monotone functions, i.e., functions such that $x \leq y$ implies $\mathrm{F}(x) \leq \mathrm{F}(y)$. In our example, this simply means that if on input a sequence of messages x, $\mathrm{F}(x)$ is produced as output, upon receiving additional messages z, the output sequence can only get longer, i.e., $\mathrm{F}(y) = \mathrm{F}(x|z) = \mathrm{F}(x)|z'$ for some z'. In other words, once the messages $\mathrm{F}(x)$ are sent out, the process cannot change its mind and set the output to a sequence that does not start with $\mathrm{F}(x)$.

Note that so far we only discussed an example of a *deterministic* process. Below, after introducing some further foundational tools, we will see that probabilistic processes are captured in the same way by letting the function output be a *distribution* over sequences, rather than a single sequence of symbols.

Further examples and notational conventions. In the examples, $|x|$ denotes the length of a sequence x, and we use array notation $x[i]$ to index the elements of a sequence. Figure 6 gives two more examples of processes that further illustrate notational conventions. Process G, in Figure 6 (middle), simply duplicates the input y (as usual in $\mathbb{Z}^{\leq n}$) and copies the input messages to two different output channels z and w. When input or output values are tuples, we usually give separate names to each component of the tuple. As before, all variables take values in $\mathbb{Z}^{\leq n}$ and the output values are defined by a set of equations that express the output in terms of the input. Finally, process $\mathrm{H}(z)$ takes as input a sequence $z \in \mathbb{Z}^{\leq n}$, and outputs the message 1 followed by the messages z received as input, possibly truncated to a prefix $z[< n]$ of length at most $n-1$, so that the output sequence x has length at most n.

Process composition. Processes are *composed* in the expected way, connecting some output variables to other input variables. Here we use the convention that variable *names* are used to

[2]Here we use sequences just as a concrete example. Our framework uses more general structures, namely domains.

$$
\begin{array}{|l|l|}
\hline
[\mathrm{G} \mid \mathrm{H}](y) = (w, x): & [\mathrm{G} \mid \mathrm{H}](y) = (w, x): \\
\quad z = y & \quad w = y \\
\quad w = y & \quad x[1] = 1 \\
\quad x[1] = 1 & \quad x[j+1] = y[j] \qquad (j < n) \\
\quad x[j+1] = z[j] \qquad (j < n) & \\
\hline
\end{array}
$$

Figure 7: Process composition

implicitly specify how different processes are meant to be connected together.[3] Composing two processes together yields, in turn, another process, which is obtained simply combining all the equations. We often refer to the resulting process as a *system* to stress its structure as a composition of basic processes. However, it should be noted that both a process and a system are objects of the same mathematical type, namely monotone functions described by systems of equations. For example, the result of composing G and H from Figure 6 yields the process (G | H) shown in Figure 7 (left), with input y and output (w, x), where $w = y$ replicates the input to make it externally visible. We use the convention that by default processes are connected by private channels, not visible outside of the system. This is modeled by turning their common input/output variables into local ones, not part of the input or output of the composed system. Of course, one can always either override this convention by explicitly listing such common input/output variables as part of the output, or bypass it by duplicating the value of a variable as done for example by process G. This is just a syntactical convention, and several other choices are possible, including never hiding variables during process composition and introducing a special projection operator to hide internal variables.

Since processes formally define functions (from input to output variables), and equations are just a syntactic method to specify functions, equations can be simplified without affecting the process. Simplifications are easily performed by substitution and variable elimination. For example, using the first equation $z = y$, one can substitute y for z, turning the last equation in the system into $x[i+1] = y[i]$. At this point, the local variable z is no longer used anywhere, and its defining equation can be removed from the system. The result is shown in Figure 7 (right). We remark that the two systems of equations shown in Figure 7 define the *same* process: they have the same input and output variables, and the equations define precisely the same function.

Feedback loops and recursive equations. Now consider the composition of all three processes F,G,H from Figure 6. Composition can be performed one pair at a time, and in any order, e.g., as [[F | G] | H] or [F | [G | H]]. Given the appropriate mathematical definitions, it can be easily shown that the result is the same, independent from the order of composition. (This is clear at the syntactic level, where process composition is simply defined by combining all the equations together. But associativity of composition can also be proved at the semantic level, where the objects being combined are functions.) So, we write [F | G | H] to denote the result of composing multiple processes together, shown in Figure 8 (left). When studying multi-party computation protocols, one is naturally led to consider collections of processes, e.g., $\mathrm{P}_1, \ldots, \mathrm{P}_n$, corresponding to the individual programs run by each participant. Given a collection $\{\mathrm{P}_i\}_i$ and a subset of indices

[3] We stress that this is just a notational convention, and there are many other syntactical mechanisms that can be used to specify the "wiring" in more or less explicit ways.

[F | G | H]() = w:	[F | G | H]() = w:
$y[i] = x[i] + 1$ $(i \leq n)$	$w[1] = 2$
$z = y$	$w[j+1] = w[j] + 1$ $(j < n)$
$w = y$	
$x[1] = 1$	
$x[j+1] = z[j]$ $(j < n)$	

Figure 8: Example of recursive process

$I \subseteq \{1, \ldots, n\}$, we write P_I to denote the composition of all P_i with $i \in I$. Similarly, we use x_A or $x[A]$ to denote a vector indexed by $i \in A$. As a matter of notation, we also use x^A to denote the $|A|$-dimensional vector indexed by $i \in A$ with all components set equal to x.

The system [F | G | H] has no inputs, and only one output w. More interestingly, the result of composing all three processes yields a recursive system of equations, where y is a function of x, x is a function of z and z is a function of y. Before worrying about solving the recursion, we can simplify the system. A few substitutions and variable eliminations yield the system in Figure 8 (right). The system consists of a single, recursively defined output variable w. The recursive definition of w is easy to solve, yielding $w[i] = i + 1$ for $i \leq n$.

2.2 Foundations: Domain theory and probabilistic processes

So far, equations have been treated in an intuitive and semi-formal way, and in fact obtaining an intuitive and lightweight framework is one of our main objectives. But for the approach to be sound, it is important that the equations and the variable symbols manipulated during the design and analysis of a system be given a precise mathematical meaning. Also, we want to consider a more general model of processes where inputs and outputs are not restricted to simple sequences of messages, but can be more complex objects, including probability distributions. This requires us to introduce some further formal tools.

The standard framework to give a precise meaning to our equations is that of *domain theory*, a well established area of computer science developed decades ago to give a solid foundation to functional programming languages [27, 26, 14, 28, 1]. Offering a full introduction to domain theory is beyond the scope of this paper, but in order to reassure the reader that our framework is sound, we recall the most basic notions and illustrate how they apply to our setting.

Domains and partial orders. *Domains* are a special kind of partially ordered set satisfying certain technical properties. We recall that a partially ordered set (or poset) $(X; \leq)$ is a set X together with a reflexive, transitive and antisymmetric relation $\leq$. We use posets to model the set of possible histories (or behaviors) of communication channels, with the partial order relation corresponding to temporal evolution. For example, a channel that allows the transmission of an arbitrary number of messages from a basic set M (and that preserves the order of transmitted messages) can be modeled by the poset $(M^*; \leq)$ of finite sequences of elements of M together with the prefix partial ordering relation $\leq$. A *chain* $x_1 \leq x_2 \leq \ldots \leq x_n$ represents a sequence of observations at different points in time.[4] In this paper we will extensively use an even simpler

[4]Domain theory usually resorts to the (related, but more general) notion of *directed set*. But not much is lost by restricting the treatment to chains, which are perhaps more intuitive to use in our setting.

poset $M_\perp$, consisting of the base set M extended with a special "bottom" element $\perp$, and the flat partial order where $x \leq y$ if and only if $x = \perp$ or $x = y$. The poset $M_\perp$ is used to model a communication channel that allows the transmission of a single message from M, with the special value $\perp$ representing a state in which no message has been sent yet.

The Scott topology and continuity. Posets can be endowed with a natural topology, called the *Scott topology*, that plays an important role in many definitions. In the case of posets $(X; \leq)$ with no infinite chains, closed sets can be simply defined as sets $C \subseteq X$ that are downward closed, i.e., if $x \in O$ and $y \leq x$, then $y \in C$. Intuitively, a set is closed if it contains all possible "pasts" that lead to a current set of events. Open sets are defined as usual as the complements of closed sets. It is easy to see that the standard (topological[5]) definition of continuous function $f\colon X \to Y$ (according to the Scott topology on posets with no infinite chains) boils down to requiring that f is monotone, i.e., for all $x, y \in X$, if $x \leq y$ in X, then $f(x) \leq f(y)$ in Y. In the case of posets with infinite chains, such as $(M^*; \leq)$, definitions are slightly more complex, and require the definition of limits of infinite chains. For any poset $(X; \leq)$ and subset $A \subseteq X$, $x \in X$ is an upper bound on A if $x \geq a$ for all $a \in A$. The value x is called the least upper bound of A if it is an upper bound on A, and any other upper bound y satisfies $x \leq y$. Informally, if $A = \{a_i \mid i = 1, 2, \ldots\}$ is a chain $a_1 \leq a_2 \leq a_3 \leq \ldots$, and A admits a least upper bound (denoted $\bigvee A$), then we think of $\bigvee A$ as the limit of the monotonically increasing sequence A. (In our setting, where the partial order models temporal evolution, the limit corresponds to the value of the variable describing the entire channel history once the protocol has finished executing.) All Scott domains (and all posets used in this paper) are complete partial orders (or CPO), i.e., posets such that all chains $A \subseteq X$ admit a least upper bound. CPOs have a minimal element $\perp = \bigvee \emptyset$, which satisfies $\perp \leq x$ for all $x \in X$. Closed sets $C \subseteq X$ of arbitrary CPOs X are defined by requiring C to be also closed under limits, i.e., for any chain $Z \subseteq C$ it must be $\bigvee Z \in C$. (Open sets are always defined as the complement of closed sets.) Similarly, continuous functions between CPOs $f\colon X \to Y$ should preserve limits, i.e., any chain $Z \subseteq X$ must satisfy $f(\bigvee Z) = \bigvee f(Z)$.

As an example, we see that $(M^*; \leq)$ is not a CPO. We can define infinite chains A of successively longer strings (e.g., take $x_i = 0^i$ for $M = \{0, 1\}$) such that no limit in M^* exists for this chain. However, note that such a chain always defines an infinite string $x^* \in M^\infty$ which is such that $x^* \leq y$ holds for all $A \leq x$. Therefore, the poset $(M^* \cup M^\infty; \leq)$ is a CPO.[6] This CPO can be used to model processes taking input and output sequences of arbitrary length.

Later on, we often use generalizations of the above limit notion, called the *join* and the *meet*, respectively. For a set $Z \subseteq X$, let $Z^\uparrow = \{z' \in X : \forall z \in Z : z \leq z'\}$ the set of upper bounds on Z. An element $z^* \in Z^\uparrow$ such that $z^* \leq z$ for all $z \in Z^\uparrow$, if it exists, is called the *join* of Z and denoted $\bigvee Z$. The set $Z^\downarrow$ and the *meet* $\bigwedge Z$ are defined symmetrically.

Equational descriptions and fixed points. We can now provide formal justification for our equational approach given above. Note that CPOs can be combined in a variety of ways, using common set operations, while preserving the CPO structure. For example, the cartesian product $A \times B$ of two CPOs is a CPO with the component-wise partial ordering relation. Using cartesian products, one can always describe every valid system of equations (as informally used in the previous

[5]We recall that a function $f\colon X \to Y$ between two topological spaces is continuous if the preimage $f^{-1}(O)$ of any open set $O \subset Y$ is also open.

[6]In fact, usually, one can define M^∞ to be exactly the set of limits of infinite chains from M^*.

paragraphs to define a process or a system) as the definition of a function f of the form

$$f(z) = g(z, x) \quad \text{where} \quad x = h(z, x) \tag{1}$$

for some internal variable x and bivariate continuous[7] functions $h(z, x)$ and $g(z, x)$. An important property of CPOs is that every continuous function $f\colon X \to X$ admits a least fixed point, i.e., a minimal $x \in X$ such that $f(x) = x$, which can be obtained by taking the limit of the chain $\bot \leq f(\bot) \leq \ldots \leq f^n(\bot) \leq \ldots$, and admits an intuitive operational interpretation: starting from the initial value $x = \bot$, one keeps updating the value $x \leftarrow f(x)$ until the computation stabilizes.

Least fixed points are used to define the solution to recursive equations as (1) above as follows, and to show that it is always defined, proving soundness of our approach. For any fixed z, the function $h_z(x) = h(z, x)$ is also continuous, and maps X to itself. So, it admits a least fixed point $x_z = \bigvee_i h_z^i(\bot)$. The function defined by (1) is precisely $f(z) = g(z, x_z)$ where x_z is the least fixed point of $h_z(x) = h(x, z)$. It is a standard exercise to show that the function $f(z)$ so defined is a continuous function of z.

Scott domains are a special class of CPOs satisfying a number of additional properties (technically, they are algebraic bounded complete CPOs), that are useful for the full development of the theory. As most of the concepts used in this paper can be fully described in terms of CPOs, we do not introduce additional definitions, and refer the reader to any introductory textbook on domain theory for a formal treatment of the subject.

Probabilistic processes. So far, our theory does not support yet the definition of processes with probabilistic behavior. Intuitively, we want to define a process as a continuous map from elements of a CPO X to probability *distributions* over some CPO Y. We will now discuss how to define the set $D(Y)$ of such probability distributions, which turns out to be a CPO. Our approach follows [25].

Let $O(X)$ be the open sets of X, and $B(X)$ the Borel algebra of X, i.e., the smallest σ-algebra that contains $O(X)$. We recall that a probability distribution over a set X is a function $p\colon B(X) \to [0, 1]$ that is countably additive and has total mass $p(X) = 1$. The set of probability distributions over a CPO X, denoted $D(X)$, is a CPO according to the partial order relation such that $p \leq q$ if and only if $p(A) \leq q(A)$ for all open sets $A \in O(X)$. This partial order on probability distributions $D(X)$ captures precisely the natural notion of evolution of a probabilistic process: the probability of a closed set can only decrease as the system evolves and probability mass "escapes" from it into the future. A probabilistic process P with input in X and output in Y is described by a continuous functions from X to $D(Y)$ that on input an element $x \in X$ produces an output probability distribution $\mathrm{P}(x) \in D(Y)$ over the set Y.

While these mathematical definitions may seem somehow arbitrary and complicated, we reassure the reader that they correspond precisely to the common notion of probabilistic computation. For example, any function $\mathrm{P}\colon X \to D(Y)$ can be uniquely extended to take as input probability distributions. The resulting function $\hat{\mathrm{P}}\colon D(X) \to D(Y)$, on input a distribution D_X, produces precisely what one could expect: the output probability distribution $D_Y = \hat{\mathrm{P}}(D_X)$ is obtained by first sampling $x \leftarrow D_X$ according to the input distribution, and then sampling the output according to $y \leftarrow \mathrm{P}(x)$. Moreover, the result $\hat{\mathrm{P}}\colon D(X) \to D(Y)$ is continuous according to the standard topology of $D(X)$ and $D(Y)$.

The fact that a distribution $D_X \in D(X)$ and a function $f\colon X \to D(Y)$ can be combined to obtain an output distribution D_Y allows to extend our equational treatment of systems to

[7] Continuity for bivariate functions is defined regarding f as an univariate function with domain $Z \times X$.

probabilistic computations. A probabilistic system is described by a set of equations similar to (1), except that h is a continuous function from $Z \times X$ to $D(X)$, and we write the equation in the form $x \leftarrow h(z, x)$ to emphasize that $h(z, x)$ is a probability distribution to sample from, rather than a single value. For any fixed z, the function $h_z(x) = h(z, x)$ is continuous, and it can be extended to a continuous function $\hat{h}_z \colon D(X) \to D(X)$. The least fixed point of this function is a probability distribution $D_z \in D(X)$, and function f maps the value z to the distribution $g(z, D_z)$.

Formally, the standard mathematical tool to give the equations a precise meaning is the use of monads, where $\leftarrow$ corresponds to the monad "bind" operation. We reassure the reader that this is all standard, well studied in the context of category theory and programming language design, both in theory and practice, e.g., as implemented in mainstream functional programming languages like Haskell. Rigorous mathematical definitions to support the definition of systems of probabilistic equations can be easily given within the framework of domain theory, but no deep knowledge of the theory is necessary to work with the equations, just like knowledge of denotational semantics is not needed to write working computer programs.

2.3 Multi-party computation, security and composability

So far, we have developed a domain-theoretic framework to define processes, their composition, and their asynchronous interaction. We still need to define what it means for such a system to implement a multi-party protocol, and what it means for such a protocol to securely implement some functionality. Throughout this section, we give definitions in the deterministic case for simplicity. The definitions extend naturally to probabilistic processes by letting the output being a probability distribution over (the product of) the output sets.

We model secure multi-party computation along the lines described in the introduction. A secure computation task is modeled by an n-party functionality F that maps n inputs $(x_1, \ldots, x_n)$ to n outputs $(y_1, \ldots, y_n)$ in the deterministic case, or to a distribution on a set of n outputs in the probabilistic case. Each input or output variable is associated to a specific domain X_i/Y_i, and F is a continuous function $F \colon (X_1 \times \cdots \times X_n) \to (Y_1 \times \cdots \times Y_n)$, typically described by a system of domain equations. Each pair X_i/Y_i corresponds to the input and output channels used by user i to access the functionality. We remark that, within our framework, even if F is a (pure) mathematical function, it still models a reactive functionality that can receive inputs and produce outputs asynchronously in multiple rounds.

Sometimes, one knows in advance that F will be used within a certain context. (For example, in the next section we will consider a multicast channel that is always used for broadcast, i.e., in a context where the set of recepient is always set to the entire group of users.) In these settings, for efficiency reasons, it is useful to consider protocols that do not implement the functionality F directly, but only the use of F within the prescribed context. We formalize this usage by introducing the notion of a protocol implementing an *interface* to a functionality. An interface is a collection of continuous functions $I_i \colon X'_i \times Y_i \to X_i \times Y'_i$, where X'_i, Y'_i are the input and output domain of the interface. Combining the interface $I = I_1 \mid \ldots \mid I_n$ with the functionality F, yields a system $(F \mid I)$ with inputs $X'_1, \ldots, X'_n$ and outputs $Y'_1, \ldots, Y'_n$ that offers a limited access to F. The standard definition of (universally composable) security corresponds to setting I to the trivial interface where $X'_i = X_i$, $Y'_i = Y_i$ and each I_i to the identity function offering direct access to F.

Ideal functionalities can be used both to describe protocol problems, and underlying communication models. Let $N \colon S_1 \times \ldots \times S_n \to R_1 \times \ldots \times R_n$ be an arbitrary ideal functionality. One may think of N as modeling a communication network where user i sends $s_i \in S_i$ and receives $r_i \in R_i$,

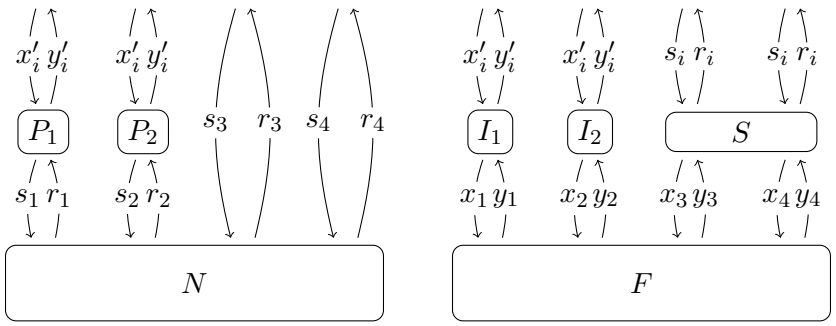

Figure 9: The protocol $(P_1, \ldots, P_4)$ securely implements interface $(I_1, \ldots, I_4)$ to functionality F in the communication model N.

but all definitions apply to arbitrary N.

A protocol implementing an interface I to functionality F in the communication model N is a collection of functions $P_1, \ldots, P_n$ where $P_i \colon X'_i \times R_i \to Y'_i \times S_i$. We consider the execution of protocol P in a setting where an adversary can corrupt a subset of the participants. The set of corrupted players $A \subseteq \{1, \ldots, n\}$ must belong to a given family $\mathcal{A}$ of allowable sets, e.g., all sets of size less than $n/2$ in case security is to be guaranteed only for honest majorities. We can now define security.

Definition 1 *Protocol P securely implements interface I to functionality F in the communication model N if for any allowable set $A \in \mathcal{A}$ and complementary set $H = \{1, \ldots, n\} \setminus A$, there is a simulator $S \colon S_A \times Y_A \to X_A \times R_A$ such that the systems $(P_H \mid N)$ and $(S \mid I_H \mid F)$ are equivalent, i.e., they define the same function.*

$(P_H \mid N)$ is called the real system, and corresponds to an execution of the protocol in which the users in A are corrupted, while those in H are honest and follow the protocol. It is useful to illustrate this system with a diagram. See Figure 9 (left). We see from the diagram that the real system as inputs X'_H, S_A and outputs Y'_H, R_A. In the ideal setting, when the adversary corrupts the users in A, we are left with the system $I_H \mid F$ because corrupted users are not bound to use the intended interface I. This system $I_H \mid F$ has inputs X'_H, X_A and outputs Y'_H, Y_A. In order to turn this system into one with the same inputs and outputs as the real one, we need a simulator of type $S \colon S_A \times Y_A \to X_A \times R_A$. When we compose S with $I_H \mid F$ we get a system $(S \mid I_H \mid F)$ with the same input and output variables as the real system $(P_H \mid N)$. See Figure 9 (right). For the protocol to be secure, the two systems must be equivalent, showing that any attack that can be carried out on the real system by corrupting the set A can be simulated on the ideal system through the simulator.

When composing protocols together, N is not a communication network, but an ideal functionality representing a hybrid model. In this setting, we say that protocol P accesses N through interface $J = (J_1, \ldots, J_n)$ if each party runs a program of the form $P_i = J_i \mid P'_i$. If this is the case, we say that P securely implements interface I to functionality F through interface J to communication model N.

Composition theorems in our framework come essentially for free, and their proof easily follow

from the general properties of systems of equations. For example, we have the following rather general composition theorem.

Theorem 1 *Assume P securely implements interface I to F in the communication model N, and $Q = Q' \mid I$ securely implements G through interface I to F, then the composed protocol $Q' \mid P$ securely implements G in the communication model N.*

The simple proof is similar to the informal argument presented in the introduction, and it is left to the reader as an exercise. The composition theorem is easily extended in several ways, e.g., by considering protocols Q that only implement a given interface J to G, and protocols P that use N through some given interface J'.

3 Secure Broadcast

In this section we provide, as a simple case study, the analysis of a secure broadcast protocol (similar to Bracha's reliable broadcast protocol [8]), implemented on an asynchronous point-to-point network. We proceed in two steps. In the first step, we build a weak broadcast protocol, that provides consistency, but does not guarantee that all parties terminate with an output. In the second step, we use the weak broadcast protocol to build a protocol achieving full security. We present the two steps in reverse order, first showing how to strenghten a weak broadcast protocol, and then implementing the weak broadcast on a point-to-point network.

3.1 Building broadcast from weak broadcast

In this section we build a secure broadcast protocol on top of a weak broadcast channel and a point to point communication network. The broadcast, weak broadcast, and communication network are described in Figure 10 (left). The broadcast functionality (BCast) receives a message x from a dealer, and sends a copy $y_i = x$ to each player. The weak broadcast channel (WCast) allows a dishonest dealer to specify (using a boolean vector $w \in \{\bot, \top\}^n$) which subset of the players will receive the message. Notice that the functionality WCast described in Figure 10 is in fact a multicast channel, that allows the sender to transmit a message to any subset of players of its choice. We call it a weak broadcast, rather than multicast, because we will not use (or implement) this functionality at its full power: the honest dealer in our protocol will always set all $w_i = \top$, and use WCast as a broadcast channel $\text{BCast}(x) = \text{WCast}(x, \top^n)$. The auxiliary inputs w_i are used only to capture the extra power given to a dishonest dealer that, by not following the protocol, may restrict the delivery of the message x to a subset of the players. This will be used in the next section to provide a secure implementation of WCast on top of a point to point communication network.

The broadcast protocol is very simple and it is shown in Figure 10 (right). The dealer simply uses WCast to transmit its input message x to all n players by setting $w[i] = \top$ for all $i \in [n]$. The players have then access to a network functionality Net to exchange messages among themselves. The program run by the players makes use to two threshold functions t_1 and t_2 each taking n inputs, which are assumed to satisfy, for every admissible set of corrupted players $A \subseteq \{1, \ldots, n\}$ and complementary set $H = \{1, \ldots, n\} \setminus A$, input vector u, and value x, the following properties: $t_1(u[A], (x)^H) = t_2(u[A], (x)^H) = x$ (i.e., if all honest players agree on x, then t_1, t_2 output x irrespective of the other values), and $t_1((\bot)^A, u[H]) \geq t_2((\top)^A, u[H])$ (i.e., for any set of values

BCast$(x) = (y_1, \ldots, y_n)$:
$y_i = x$ $(i = 1, \ldots, n)$

WCast$(x', w) = (y'_1, \ldots, y'_n)$:
$y'_i = x' \wedge w[i]$ $(i = 1, \ldots, n)$

Net$(s_1, \ldots, s_n) = (r_1, \ldots, r_n)$:
$r_i[j] = s_j[i]$ $(i, j = 1, \ldots, n)$

Dealer$(x) = (x', w)$:
$w[i] = \top$ $(i = 1, \ldots, n)$
$x' = x$

Player[i]$(y'_i, r_i) = (y_i, s_i)$: $(i = 1, \ldots, n)$
$s_i[j] = y'_i \vee t_1(r_i[1], \ldots, r_i[n])$ $(j = 1, \ldots, n)$
$y_i = t_2(r_i[1], \ldots, r_i[n])$

Figure 10: The Broadcast protocol

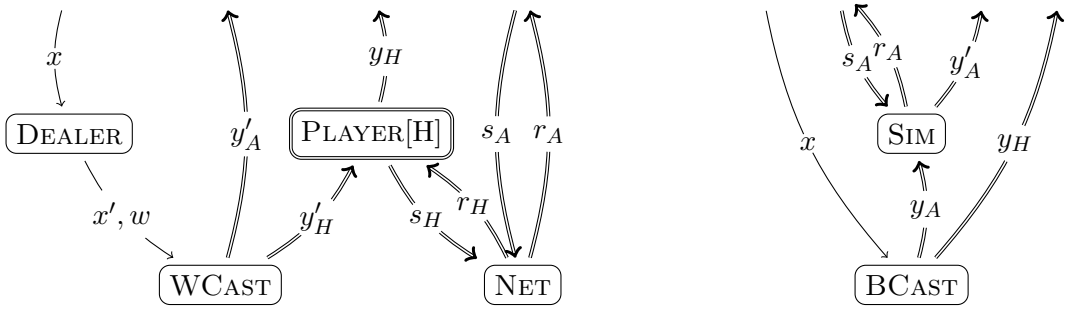

Figure 11: Security of broadcast protocol when the dealer is honest

provided by the honset players, t_1 is always bigger than t_2 regardless of the other values). It is easy to see that the threshold functions $t_i(u) = \bigvee_{|S|=k_i} \bigwedge_{j \in S} u_j$ satisfy these properties provided $|A| < k_1 \leq k_2 - |A| \leq n - 2|A|$, which in paricular requires $n \geq |A| + 1$.

In the security analysis, we distinguish two cases, depending on whether the dealer is corrupt or not.

Honest dealer. First we consider the simple case where the adversary corrupts a set of players $A \subset \{1, \ldots, n\}$, and the dealer behaves honestly. Let $H = \{1, \ldots, n\} \setminus A$ be the set of honest players. An execution of the protocol when players in A are corrupted is described by the system (Dealer | Player[H] | WCast | Net) with input (x, s_A) and output (y_H, y'_A, r_A) depicted in Figure 11 (left). Note that in this (and the following) figures, double arrows and boxes denote parallel processes and channels. Combining the defining equations of Dealer, Player[h] for $h \in H$, WCast and Net, and introducing the auxiliary variables $u_h = x \vee t_1(s_A[h], s_H[h])$ for all $h \in H$, we get that for any $i, j \in [n]$, and $h \in H$ the following holds:

$$\begin{aligned} r_j[i] &= s_i[j] \\ y'_i &= x' \wedge w[i] = x \wedge \top = x \\ s_h[i] &= y'_h \vee t_1(r_h[1], \ldots, r_h[n]) = x \vee t_1(s_A[h], s_H[h]) = u_h \\ y_h &= t_2(r_h[1], \ldots, r_h[n]) = t_2(s_A[h], s_H[h]) = t_2(s_A[h], u_H) \\ u_h &= x \vee t_1(s_A[h], s_H[h]) = x \vee t_1(s_A(h), u_H)\ . \end{aligned}$$

SIM$(y_A, s_A) = (y'_A, r_A)$:	SIM'$(x', w, y_A, s_A) = (x, r_A, y'_A)$:
$r_A[a] = s_a[A]$ $(a \in A)$	$u_h = (x' \wedge w[h]) \vee t_1(s_A[h], u_H)$ $(h \in H)$
$r_A[h] = y_A$ $(h \in H)$	$x = t_2(s_A[h], u_H)$ $(h = \min H)$
$y'_A = y_A$	$y'_a = x' \wedge w[a]$ $(a \in A)$
	$r_a[A] = s_A[a]$ $(a \in A)$
	$r_a[H] = u_H$ $(a \in A)$

Figure 12: Simulators for the broadcast protocol when the dealer is honest (left) or dishonest (right)

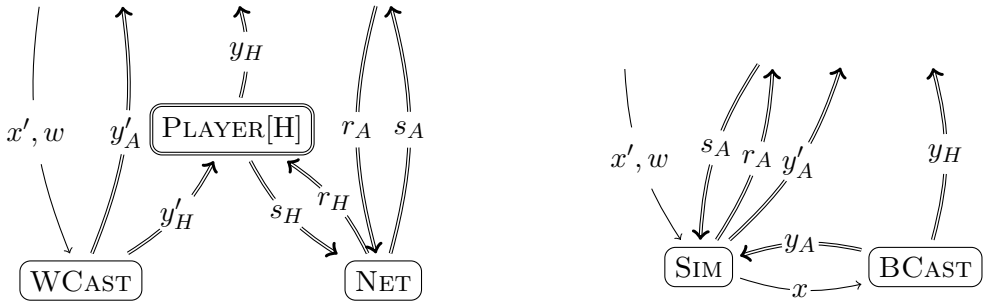

Figure 13: Security of broadcast protocol when the dealer is corrupted.

The last equation $u_h = x \vee t_1(s_A(h), u_H)$ provides a recursive definition of u_H, which can be easily solved by an iterative least fix point computation: starting from $u_H^{(0)} = \perp^H$, we get $u_H^{(1)} = (x \vee t_1(s_A(h), u_H^{(0)}))^H = x^H$, and then again $u_H^{(2)} = (x \vee t_1(s_A(h), u_H^{(1)}))^H = x^H$. Therefore the least fix point is $u_H = x^H$. Substituting $u_H = x^H$ in the previous equations, and using the properties of t_2, we see that the system of equations defined by (DEALER | PLAYER[H] | WCAST | NET) is equivalent to

$$\begin{array}{lcll} r_a & = & (s_A[a], x^H) & (a \in A) \\ y'_a & = & x & (a \in A) \\ y_h & = & t_2(s_A[h], x^H) = x & (h \in H) \end{array} \tag{2}$$

We now show that an equivalent system can be obtained by combining the ideal functionality BCAST with a simulator SIM as in Figure 11 (right). The simulator takes (y_A, s_A) as input, and must output (y'_A, r_A) such that (SIM| BCAST) is equivalent to the system (DEALER| PLAYER[H] | WCAST| NET) specified by the last set of equations. The simulator is given in Figure 12 (left). It is immediate to verify that combining the equations of the simulator SIM with the equations $y_i = x$ of the ideal broadcast functionality, and eliminating local variables, yields a system of equations identical to (2).

Dishonest dealer. We now consider the case where both the dealer and a subset of players A are corrupted. As before, let $H = \{1, \ldots, n\} \setminus A$ be the set of honest players. The system corresponding to a real execution of the protocol when DEALER and PLAYER[A] are corrupted is (PLAYER[H] | WCAST| NET), mapping (x', w, s_A) to (y_H, r_A, y'_A). (See Figure 13 (left).) Using the defining equations of PLAYER[H], WCAST and NET, and introducing auxiliary variables $u_h =$

$y'_h \vee t_1(r_h[1], \ldots, r_h[n])$ for $h \in H$, we get the following set of equations:

$$\begin{array}{rcll} y_h & = & t_2(r_h[A], r_h[H]) = t_2(s_A[h], u_H) & (h \in H) \\ y'_a & = & x' \wedge w[a] & (a \in A) \\ r_a[A] & = & s_A[a] & (a \in A) \\ r_a[H] & = & u_H & \\ u_h & = & (x' \wedge w[h]) \vee t_1(s_A[h], u_H) & (h \in H) \end{array} \tag{3}$$

This time the simulator Sim' takes input (x', w, y_A, s_A) and outputs (x, r_A, y'_A). (See Figure 13 (right).) With these inputs and outputs, the simulator can directly set all variables except y_h just as in the real system (3). The simulator can also compute the value y_h, but it cannot set y_h directly because this variable is defined by the ideal functionality as $y_h = x$. We will prove that all variables y_h defined by (3) take the same value. It follows that the simulator can set $x = y_h$ for any $h \in H$, and the system (BCast,Sim') will be equivalent to (3) (and therefore to (Player[H] | WCast| Net)). The code of the simulator is given in Figure 12 (right), where $x = y_h$ is arbitrarily selected using the smallest index $h = \min H$. (Any other choice of h would have been fine.)

It remains to prove that all y_h take the same value. By antisymmetry, it is enough to show that $y_i \leq y_j$ for all $i, j \in H$. These easily follows from the assumptions on t_1, t_2. In fact, by monotonicity, we have

$$y_i = t_2(s_A[i], u_H) \leq t_2(\top^A, u_H) \leq t_1(\bot^A, u_H) \leq t_1(s_A[j], u_H) \leq u_j.$$

It immeddiately follows that $y_j = t_2(s_A[j], u_H) \geq t_2(s_A[j], (y_i)^H) = y_i$.

3.2 Weak broadcast

In this section we show how to implement the weak broadcast functionality WCast given in Figure 10 to be used within the BCast protocol discussed in the previous section, and analyze is security. We recall that WCast is a multicast functionality connecting a dealer to n other parties, which allows the dealer to send a message x' to a subset of the parties specified by a vector $w \in \{\bot, \top\}^n$. We stress that we do not need a secure implementation of WCast in its full generality, as our higher level broadcast protocol (BCast) uses WCast is a rather restricted way: it always set $w = (\top)^n$ and transmits x' to all parties. Accordingly, we give a protocol that securely implements this interface to WCast. Formally, the dealer's interface Int takes only x' as external input, and passes it along with $w = \top^n$ to the ideal functionality. The other parties have unrestricted access to the ideal functionality, and their interface is the identity function (or empty system of equations).

We implement interface Int to WCast on top of a point-to-point communication network similar to the Net functionality described in Figure 10, with the only difference that here also the dealer can send messages. The protocol is very simple: the dealer transmit the input x' to all parties, and the parties retransmit the message to each other. Each party sets its output using a threshold function of the messages received by the other parties. The equations corresponding to the network Net', interface Int, and protocol programs Dealer, Player[1],...,Player[n] are given in Figure 14. For reference, we have also repeated the definition of BCast from Figure 10. The function t is assumed to satisfy the following properties: $t(u[A], (x)^H) = x$ (i.e., if all honest parties agree on x, then the output is x), and, moreover, for all vectors u, u' with $u[H] = u'[H]$, we have $t(u) = t(u')$ or $t(u) = \bot$. It is easy to see that the threshold function $t(u) = \bigvee_{|S|=k} \bigwedge_{j \in S} u_j$

WCast$(x', w) = (y'_1, \ldots, y'_n)$: $y'_i = x' \wedge w[i]$ $(i = 1, \ldots, n)$ Int: $w = \top^n$ Net'$(s'_0, \ldots, s'_n) = (r'_1, \ldots, r'_n)$: $r'_i[j] = s'_j[i]$ $(i = 1, \ldots, n; j = 0, \ldots, n)$	Dealer$(x') = (s'_0)$: $s'_0[i] = x'$ $(i = 1, \ldots, n)$ Player[i]$(r'_i) = (y'_i, s'_i)$: $(i = 1, \ldots, n)$ $s'_i[j] = r'_i[0]$ $(j = 1, \ldots, n)$ $y'_i = t(r'_i[1], \ldots, r'_i[n])$

Figure 14: Weak broadcast protocol.

satisfies both properties for $k \geq \frac{n+|A|+1}{2}$. Namely, take any two vectors u, u' with $u[H] = u'[H]$, assume that there exist sets S and S' such that $u_j = x$ for all $j \in S$ and $u'_j = y$ all $j \in S'$. Then, since $|S \cap S' \cap H| \geq 2k - n - |A| > 0$, and hence $x = y$.

As usual, we consider two cases in the proof of security, depending on whether the dealer is corrupted or not.

Dishonest dealer. It is convenient to consider the case when the dealer is dishonest first, as some of the derived equations will be useful in the honest dealer case too. Beside the dealer, the players in $A \subseteq \{1, \ldots, n\}$ are corrupted, and we let $H = \{1, \ldots, n\} \setminus A$ be the set of honest players. We consider the real-world system (Player$[H]$ | Net') consisting of the honest partecipants and the network Net'. This is a system with input (s'_0, s'_A) and output (y'_H, r'_A) described by the defining equations of Player$[h]$ for $h \in H$ and Net' given in Figure 14. We use these equations to express each output variable of the system in terms of the input variables. For y'_h $(h \in H)$ we have

$$\begin{aligned} y'_h &= t(r'_h[1], \ldots, r'_h[n]) \\ &= t(s'_1[h], \ldots, s'_n[h]) \\ &= t(s'_A[h], s'_H[h]) \\ &= t(s'_A[h], r'_H[0]) = t(s'_A[h], s'_0[H]). \end{aligned}$$

For the other output variables $r'_A[i]$ we distinguish two cases, depending on whether $i \in A$. For $a \in A$, we immediately get $r'_A[a] = s'_a[A]$. For $h \in H$, we have $r'_A[h] = s'_h[A] = (r'_h[0])^A = (s'_0[h])^A$. The resulting system is given by the following equations

$$r'_A[a] = s'_a[A] \qquad (a \in A) \tag{4}$$

$$r'_A[h] = (s'_0[h])^A \qquad (h \in H) \tag{5}$$

$$y'_h = t(s'_A[h], s'_0[H]) \qquad (h \in H) \tag{6}$$

We now turn to the simulator. Recall that the simulator should turn the system defined by WCast into one equivalent to the real world system. To this end, the simulator should take s'_0, s'_A and y'_A as input (from the external environment and ideal functionality respectively), and output x', w (to the ideal functionality) and r'_A (to the external environment). Notice that the simulator has all the inputs necessary to compute the values defined by the real system, and in fact can set r'_A using just those equations. The only difficulty is that the simulator cannot set y'_h directly, but has only indirect control over its value through the ideal functionality and the variables x', w. From the

properties of function t, we know that all $y'_h = t(s'_A[h], s'_0[H])$ take either the same value or $\perp$. So, the simulator can set x' to this common value, and use w to force some y'_h to $\perp$ as appropriate. The simulator SIM' is given in Figure 15 (right). It is easy to verify that (SIM' | WCAST) is equivalent to the real system.

Honest dealer. In this case, we first consider the real-world system (DEALER| PLAYER$[H]$ | NET') consisting of the dealer, the honest partecipants $H \subseteq \{1, \ldots, n\}$, and the network NET'. The corrupted parties are given by the set $A = \{1, \ldots, n\} \setminus H$. This is a system with input (x', s'_A) and output (y'_H, r'_A) described by the defining equations of DEALER, PLAYER$[h]$ for $h \in H$, and NET' given in Figure 14. Notice that this is a superset of the equations for the real-world system (PLAYER$[H]$ | NET') considered in the dishonest dealer case. So, equations (4), (5) and (6) are still valid. Adding the equations from DEALER and using the properties of t we get that $y'_h = t(s'_A[h], s'_0[H]) = t(s'_A[h], (x)^H) = x$. Similarly, for $h \in H$, we have $r'_A[h] = (s'_0[h])^A = (x)^A$. Finally, we know from (4) that $r'_A[a] = s'_a[A]$. Combining the equations together, we get the following real system:

$$\begin{array}{rcll} y'_h & = & x' & (h \in H) \\ r'_A[a] & = & s'_a[A] & (a \in A) \\ r'_A[h] & = & (x')^A & (h \in H) \end{array}$$

We now move to the simulator. Recall that the simulator should turn the systen defined by WCAST and INT into one equivalent to the real world system. To this end, the simulator should take y'_A and s'_A as input (from the ideal functionality and external environment respectively), and output r'_A. Notice that $y'_h = x'$ in the real system is defined just as in the equations for the ideal functionality WCAST when combined with the (honest) dealer interface INT. (In fact, $y'_a = x'$ also for $a \in A$.) The other variables r'_A can be easily set by the simulator as shown in Figure 15 (left). It is immediate to check that (SIM | INT | WCAST) is equivalent to the real world system.

4 Verifiable Secret Sharing

Let $\mathbb{F}_t[X]$ be the set of all polynomials of degree at most t over a finite field $\mathbb{F}$ such that[8] $\{0, 1, \ldots, n\} \subseteq \mathbb{F}$. We consider the n-party *verifiable secret sharing (VSS)* functionality that takes as input from a *dealer* a degree-t polynomial $\mathsf{p} \in \mathbb{F}_t[X]$ and, for all $i \in [n]$, outputs the evaluation

[8]This assumption is not really necessary; we could replace $\{0, 1, \ldots, n\}$ with $\{0, x_1, \ldots, x_n\}$ for any n distinct field elements $x_1, \ldots, x_n$.

SIM$(y'_A, s'_A) = (r'_A)$:
- $r'_A[a] = s'_a[A]$ $(a \in A)$
- $r'_A[h] = y'_A$ $(h \in H)$

SIM'$(y'_A, s'_0, s'_A) = (x', w, r'_A)$:
- $r'_A[a] = s'_a[A]$ $(a \in A)$
- $r'_A[h] = (s'_0[h])^A$ $(h \in H)$
- $x' = \bigvee_{h \in H} t(s'_A[h].s'_0[H])$
- $w[h] = (t(s'_A[h], s'_0[H]) > \perp)$ $(h \in H)$

Figure 15: Real world systems and simulators for the weak broadcast protocol. Honest dealer case (left) and dishonest dealer case (right)

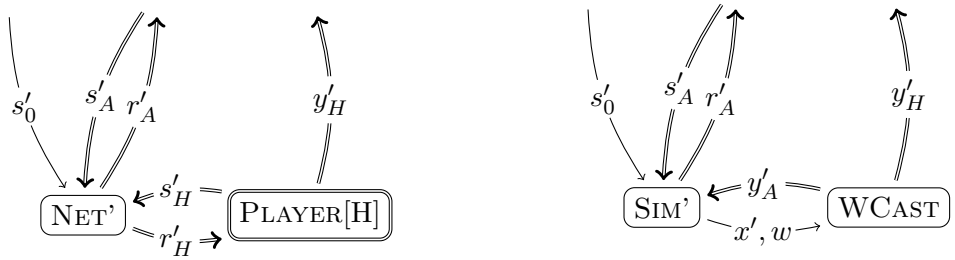

Figure 16: Security of the weak multicast protocol, when the dealer is dishonest. Real world execution on the left. Simulated attack in the ideal world on the right.

$\mathsf{p}(i)$ to the i-th party. The formal definition of VSS: $\mathbb{F}_t[X]_\perp \mapsto \mathbb{F}_\perp^n$ is given in Figure 18 (left), where by convention $\perp(x) = \perp$ for all x.

We devise a protocol implementing the VSS functionality on top of a point-to-point network functionality NET defined as in the previous section that allows the n parties to exchange elements from $\mathbb{F}$, and two other auxiliary functionalities. The protocol is based on the one by [5]. Even though its complexity is exponential in n, we have chosen to present this protocol due to its simplicity. The first auxiliary functionality (GRAPH) grants all parties access to the adjacency matrix of an n-vertex directed graph (with loops), where each party $i \in [n]$ can add outgoing edges to vertex i, but not to any other vertex $j \neq i$. Formally, GRAPH: $\{\perp, \top\}^n \times \cdots \times \{\perp, \top\}^n \to \{\perp, \top\}^{n \times n}$ is given in Figure 18 (center). Setting $G[i, j] = \top$ is interpreted as including an edge from i to j in the graph. GRAPH can be immediately implemented using n copies of a broadcast functionality, where a different party acts as the sender in each copy. We also assume the availability of an additional unidirectional network functionality NET': $(\mathbb{F}_t[X]^2_\perp)^n \to (\mathbb{F}_t[X]^2_\perp)^n$ that allows the VSS dealer to send to each party a pair of polynomials of degree at most t. See Figure 18 (right).

The VSS protocol. We turn to the actual protocol securely implementing the VSS functionality. We first define some auxiliary functions. For any subset $C \subseteq [n]$, let $\mathsf{clique}_C : \{\perp, \top\}^{n \times n} \to \{\perp, \top\}$ be the function $\mathsf{clique}_C(G) = \bigwedge_{i,j \in C} G[i, j]$. This function is clearly monotone, and tests if C is a clique in G. For any set A, we equip the set $A_\perp$ with a monotone equality-test function

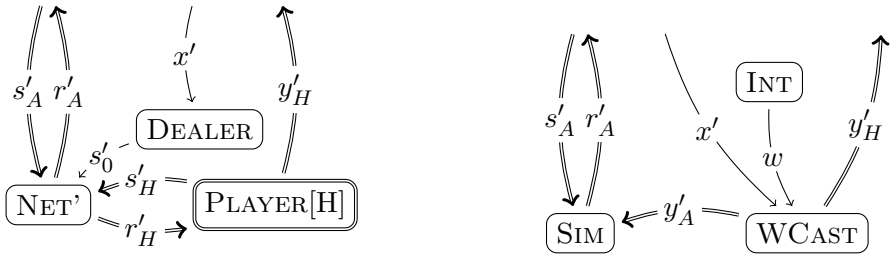

Figure 17: Security of the weak multicast protocol, when the dealer is honest. Real world execution on the left. Simulated attack in the ideal world on the right.

VSS(p) = $(p_1, \ldots, p_n)$ $p_i = \mathsf{p}(i)$ $(i = 1, \ldots, n)$	GRAPH$(G_1, ..., G_n) = G$: $G[i,j] = G_i[j]$ $(i, j = 1, \ldots, n)$.	NET'$(s') = (r'_1, \ldots, r'_n)$: $r'_i = s'[i]$ $(i = 1, \ldots, n)$.

Figure 18: The VSS functionality, and two auxiliary functions used to realize it.

$\mathsf{eq} : A_\perp \times A_\perp \to \{\perp, \top\}$ where $\mathsf{eq}(x, y) \equiv (x = y \neq \perp)$. Monotonicity follows from the fact that all the pairs (x, x) such that $\mathsf{eq}(x, y) = \top$ are maximal elements in $A_\perp \times A_\perp$.

For any $S \subseteq [n]$ of size $|S| \geq t + 1$, and $r \in \mathbb{F}_b^n$, let $\mathsf{interpolate}_S(r) \in \mathbb{F}_t[X]_\perp$ be the (unique) polynomial $\mathsf{h} \in \mathbb{F}_t[X]$ such that $\mathsf{h}(S) = r[S]$ if such polynomial exists, and $\mathsf{interpolate}_S(r) = \perp$ otherwise. For $C \subseteq [n]$, define also a monotone function $\mathsf{interpolate}_{C,t} : {\mathbb{F}_\perp}^n \to \mathbb{F}_t[Y]_\perp^\top$ where $\mathsf{interpolate}_{C,t}(r) = \bigvee\{\mathsf{interpolate}_S(r) \colon S \subseteq C, |S| = |C| - t\}$. Notice that $\mathsf{interpolate}_{C,t}(r) = \perp$ if no interpolating polynomial exists, while $\mathsf{interpolate}_{C,t}(r) = \top$ if there are multiple solutions. Note that if $n \geq 4t + 1$ and $|C| \geq n - t$, then $\top$ never occurs: Indeed, let $S, S' \subseteq C$ be such that $|S| = |S|' = |C| - t$, and such that both $\mathsf{interpolate}_S(r)$ and $\mathsf{interpolate}_{S'}(r)$ differ from $\perp$. Since $|S \cap S'| \geq |C| - 2t \geq n - 3t \geq t + 1$, we must have $\mathsf{interpolate}_S(r) = \mathsf{interpolate}_{S'}(r)$ by the fact that two degree t polynomials agreeing at $t + 1$ points are necessarily equal. For future reference, this is summarized by the following lemma.

Lemma 1 *Let n, t be such that $n \geq 4t + 1$, and let $C \subseteq [n]$ be such that $|C| \geq n - t$. Then $\mathsf{interpolate}_{C,t}(r) \neq \top$ for all $r \in \mathbb{F}_\perp^n$.*

In the following, denote as $\mathbb{F}_t[X, Y]$ the set of polynomials $\mathsf{f} = \mathsf{f}(X, Y)$ in $\mathbb{F}[X, Y]$ with degree at most t in each variable X and Y. For any $\mathsf{p} \in \mathbb{F}_t[X]_\perp$, let $\mathcal{P}_t(\mathsf{p})$ the (uniform distribution over the) set of bivariate polynomials $\mathsf{f} = \mathsf{f}(X, Y) \in \mathbb{F}_t[X, Y]_\perp$ of degree at most t in X and Y such that $\mathsf{f}(\cdot, 0) = \mathsf{p}$. (By convention, if $\mathsf{p} = \perp$, then $\mathcal{P}_t(\mathsf{p}) = \{\perp\}$.)

The protocol consists of a dealer DEALER which, on input a polynomial p, first chooses a random bivariate polynomial f in $\mathcal{P}_t(\mathsf{p})$. (This is the only random choice of the entire protocol.) For all $i \in [n]$, it sends the two polynomials $\mathsf{g}_i = \mathsf{f}(\cdot, i)$ and $\mathsf{h}_i = \mathsf{f}(i, \cdot)$ to player i, with the usual convention that if $\mathsf{f} = \perp$, then $f(\cdot, i) = f(i, \cdot) = \perp$. The players then determine whether the polynomials they received are consistent. This is achieved by having each honest party i send $\mathsf{g}_i(j)$ to player j, who, in turn, checks consistency with $\mathsf{h}_j(i)$. (Note that if the polynomials are correct, then $\mathsf{g}_i(j) = \mathsf{h}_j(i) = \mathsf{f}(j, i)$.) If the consistency check is successful, player j raises the entry $G[j, i]$ to $\top$. Each honest party i then waits for a clique $C \subseteq [n]$ of size (at least) $n - t$ to appear in the directed graph defined by G, and computes the polynomial $\mathsf{o}_i \in \mathbb{F}_t[X]_\perp^\top$ obtained interpolating the values $\mathsf{g}_j(i)$ received from other parties. (Here $\top$ represents an error condition meaning that multiple interpolating polynomials were found, and should not really occur in actual executions, as we will show.) As soon as such a polynomial is found, the honest party terminates with output $p_i = \mathsf{o}_i(0)$. A formal specification is given in Figure 19.

In the following, we turn to proving security of the protocol. The analysis consists of two cases.

Honest dealer security. We start by analyzing the security of the above protocol in the case where the dealer is honest. For all $A \subseteq [n]$ where $|A| = t$ and $n \geq 4t + 1$, define $H = [n] \setminus A$. When the players in the set A are corrupted (and thus the players in H are honest), an execution of the VSS protocol with honest dealer is given by the system (DEALER | PLAYER[H] | NET' | NET | GRAPH) with inputs p, s_A, G_A and outputs $r_D[A], r_A, G, p_H$ given in Figure 20.

DEALER(p) = s':	PLAYER[i](r'_i, G, r_i) = (p_i, s_i, G_i): ($i = 1, \ldots, n$)
$\mathsf{f} \leftarrow \mathcal{P}_t(\mathsf{p})$	$(\mathsf{g}_i, \mathsf{h}_i) = r'_i$
$s'[i] = (\mathsf{f}(\cdot, i), \mathsf{f}(i, \cdot))$ ($i = 1, \ldots, n$)	$s_i[j] = \mathsf{g}_i(j)$ ($j = 1, \ldots, n$)
	$G_i[j] = \mathsf{eq}(r_i[j], \mathsf{h}_i(j))$ ($j = 1, \ldots, n$)
	$\mathsf{o}_i = \bigvee_{\substack{C \subseteq [n] \\ \lvert C \rvert \geq n-t}} \left[\mathsf{clique}_C(G) \wedge \mathsf{interpolate}_{C,t}(r_i)\right]$
	$p_i = \mathsf{o}_i(0)$.

Figure 19: The VSS protocol.

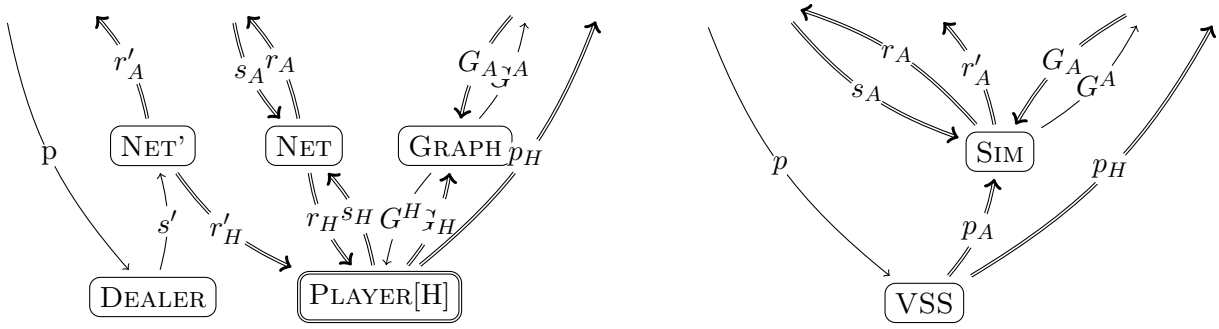

Figure 20: Security of the VSS protocol when the dealer is honest.

We proceed by combining all the equations together, and simplifying the result, until we obtain a system of equations that can be easily simulated. We use the above definition of the system to obtain the following equations describing (DEALER | PLAYER[H] | NET' | NET | GRAPH): For any $i, j \in [n]$, and any $h \in H, a \in A$, we have

$$\begin{aligned} \mathsf{f} &\xleftarrow{\$} \mathcal{P}_t(\mathsf{p}) & G[h,j] &= \mathsf{eq}(r_h[j], \mathsf{f}(h,j)) \\ r'_i &= (\mathsf{f}(\cdot, i), \mathsf{f}(i, \cdot)) & G[a,j] &= G_a[j] \\ r_i[h] &= \mathsf{f}(i,h) & \mathsf{o}_h &= \bigvee_{C \subseteq [n], \lvert C \rvert \geq n-t} \left[\mathsf{clique}_C(G) \wedge \mathsf{interpolate}_{C,t}(r_h)\right] \\ r_i[a] &= s_a[i] & p_h &= \mathsf{o}_h(0)\ . \end{aligned}$$

For convenience, some simplifications have already been made: First g_i and h_i have been replaced by $\mathsf{f}(\cdot, i)$ and $\mathsf{f}(i, \cdot)$, respectively. Second, we used the facts that $r'_i = s'[i]$ and $r_i[h] = s_h[i] = \mathsf{f}(i,h)$ for all $h \in H$ and all $i \in [n]$ by the definitions of the network functionalities NET' and NET. Finally, we have set values for $G[\cdot,\cdot]$ according to the protocol specification (for honest players) and the inputs G_a of players $a \in A$.

In order to further simplify the system, we claim that $p_h = \mathsf{p}(h)$ for $h \in H$. If $\mathsf{p} = \perp$, then this is easy to see because $\mathsf{f} = \perp$ and $G[h,j] = \mathsf{eq}(r_h[j], \perp) = \perp$. Therefore, we necessarily have $\mathsf{clique}_C(G) = \perp$ for all $C \subseteq [n]$ with $|C| \geq n - t$, since $|C \cap H| \geq n - 2t > 0$. So, we only need to prove the claim for $\mathsf{p} \neq \perp$. Notice that the equations $G[h,j] = \mathsf{eq}(r_h[j], \mathsf{f}(h,j))$, depending on

whether $j = h' \in H$ or $j = a \in A$, can be replaced by the set of equations

$$\begin{aligned} G[h,h'] &= \mathsf{eq}(r_h[h'], \mathsf{f}(h,h')) = \mathsf{eq}(\mathsf{f}(h,h'), \mathsf{f}(h,h')) = \top \\ G[h,a] &= \mathsf{eq}(r_h[a], \mathsf{f}(h,a)) = \mathsf{eq}(s_a[h], \mathsf{f}(h,a)) \ . \end{aligned}$$

This in particular implies that $C = H$ is a clique of size at least $n - t$ in the graph defined by G, i.e., we have $\mathsf{clique}_H(G) = \top$ by the above. Also, since $r_h[h'] = \mathsf{f}(h,h')$, we necessarily have

$$\mathsf{o}_h \geq \mathsf{clique}_H(G) \wedge \mathsf{interpolate}_{H,t}(r_h) = \top \wedge \mathsf{f}(h,\cdot) = \mathsf{f}(h,\cdot)$$

by Lemma 1. Now, let $S \subseteq C$ be any sets such that $|C| \geq n - t$ and $|S| = |C| - t \geq n - 2t$. Since $\mathsf{o}_h(h') = \mathsf{f}(h,h')$ for all $h' \in H$ and $|S \cap H| \geq n - 3t \geq t + 1$, we have $\mathsf{interpolate}_S(r_h) \geq \mathsf{f}(h,\cdot)$, and, by Lemma 1, $\mathsf{interpolate}_S(r_h) = \mathsf{f}(h,\cdot)$. This proves that $\mathsf{o}_h = \mathsf{interpolate}_{C,t}(r_h) = \mathsf{f}(h,\cdot)$, and $p_h = \mathsf{o}_h(0) = \mathsf{f}(h,0) = \mathsf{p}(h)$.

Summarizing, the real system is described by the following set of equations:

$$\begin{aligned} \mathsf{f} &\xleftarrow{\$} \mathcal{P}_t(\mathsf{p}) & \qquad G[h,h'] &= (\mathsf{p} > \bot) \\ r'_a &= (\mathsf{f}(\cdot,a), \mathsf{f}(a,\cdot)) & G[h,a] &= \mathsf{eq}(s_a[h], \mathsf{f}(h,a)) \\ r_a[a'] &= s_{a'}[a] & G[a,j] &= G_a[j] \\ r_a[h] &= f(a,h) & p_h &= \mathsf{p}(h) \ . \end{aligned}$$

Notice that this is exactly how p_h is defined by the VSS functionality. So, in order to prove security, it is enough to give a simulator Sim that on input p_A, s_A, G_A, outputs G, r_A and r'_A as defined in the above system of equations. See Figure 20 (right).

The problem faced by the simulator is that it cannot test $\mathsf{p} > \bot$ and generate f as in the equations because it does not know the value of p, rather it only has partial information $p_A = \mathsf{p}(A)$. The first condition $\mathsf{p} > \bot$ is easy to check because it is equivalent to $p_a = \mathsf{p}(a) > \bot$ for any $a \in A$. In order to complete the simulation, we observe that the equations only depend on the $2t$ polynomials $f(\cdot, A)$ and $f(A, \cdot)$. The next lemma shows that, given $\mathsf{p}(A)$, the polynomials $\mathsf{f}(\cdot, A)$ and $\mathsf{f}(A, \cdot)$ are statistically independent from p, and their distribution can be easily sampled.

Lemma 2 *Let $\mathsf{p} \in \mathbb{F}_t[X]$, let $\mathsf{f} \xleftarrow{\$} \mathcal{P}_t(\mathsf{p})$, and for all $a \in A$, let $\mathsf{g}_a = \mathsf{f}(\cdot, a)$ and $\mathsf{h}_a = \mathsf{f}(a, \cdot)$. The conditional distribution of $(\mathsf{g}_a, \mathsf{h}_a)_{a \in A}$ given $\mathsf{p}(A)$ is statistically independent of p, and it can be generated by the following algorithm Samp(p_A): first pick random polynomials $\mathsf{h}_a \in \mathbb{F}_t[Y]$ independently and uniformly at random subject to the constraint $\mathsf{h}_a(0) = p_a$. Then, pick $\mathsf{g}_a \in \mathbb{F}_t[X]$, also independently and uniformly at random, subject to the constraint $\mathsf{g}_a(A) = \mathsf{h}_A(a)$.*

Using the algorithm from the lemma, we obtain the following simulator Sim:

Sim(p_A, s_A, G_A) = (G^A, r_A, r'_A):		$G[h,h'] = \bigvee_{a \in A}(p_a > \bot)$	$(h, h' \in H)$
$(\mathsf{g}_A, \mathsf{h}_A) \leftarrow \mathrm{Samp}(p_A)$		$G[h,a] = \mathsf{eq}(s_a[h], \mathsf{g}_a(h))$	$(h \in H, a \in A)$
$r'_A = (\mathsf{g}_A, \mathsf{h}_A)$		$G[a,j] = G_a[j]$	$(a \in A, j \in [n])$
$r_a[h] = \mathsf{h}_a(h)$	$(h \in H,\ a \in A)$		
$r_a[a'] = s_{a'}[a]$	$(a, a' \in A)$		

As usual, if $\mathsf{p} = \bot$, then $p_A = \bot^A$ and by convention $\mathrm{Samp}(p_A) = \{\bot, \bot\}$.

Dishonest dealer security. We now look at the case where the dealer is not honest. As above, for all $A \subseteq [n]$ where $|A| = t$ and $n \geq 4t+1$, define $H = [n] \setminus A$. When the players in the set A are corrupted (and thus the players in H are honest), an execution of the VSS protocol with dishonest dealer is given by the system (PLAYER[H] | NET' | NET | GRAPH) with inputs s', r_A, G_A, and outputs r'_A, s_A, p_H and G^A. As above, we start with an equational description of the system, and will simplify it below into a form where the construction of a corresponding simulator becomes obvious. For all $i, j \in [n]$, $h, h' \in H$, and $a \in A$, we have

$$\begin{aligned}
(\mathsf{g}_h, \mathsf{h}_h) &= s'[h] & G[h,a] &= \mathsf{eq}(s_a[h], \mathsf{h}_h(a)) \\
r'_a &= s'[a] & G[a,j] &= G_a[j] \\
r_i[h] &= \mathsf{g}_h(i) & \mathsf{o}_h &= \bigvee_{C \subseteq [n], |C| \geq n-t} \left[\mathsf{clique}_C(G) \wedge \mathsf{interpolate}_{C,t}(r_h)\right] \\
r_i[a] &= s_a[i] & & \\
G[h,h'] &= \mathsf{eq}(\mathsf{g}_{h'}(h), \mathsf{h}_h(h')) & p_h &= \mathsf{o}_h(0)\ .
\end{aligned}$$

Notice that we have already undertaken several easy simplification steps, defining variables which are part of the output as a function of the system inputs and of auxiliary variables g_H, h_H, o_H, and r_H. Specifically, to obtain the above equations starting from the original system specification, we have used $r_i[j] = s_j[i]$, where $s_h[i] = \mathsf{g}_h(i)$, together with $r'_i = s'[i]$ and the definition of $G[h,i]$, distinguishing between the cases $i = a \in A$ and $i = h' \in H$.

Recall that in order to prove security, we need to give a simulator SIM with input s', G_A, s_A, p_A and output r'_A, r_A, G^A and p such that (VSS | SIM) is equivalent to the above system. (See Figure 21.) Notice that in the system describing a real execution, all variables except p_h (and intermediate value o_h) are defined as functions of the inputs given to the simulator. So, SIM can set all these variables just as in the system describing the real execution. The only difference between a real execution and a simulation is that the simulator is not allowed to set p_h directly. Rather, it should specify a polynomial $\mathsf{p} \in \mathbb{F}_t[X]_\perp$, which implicitly defines $p_h = \mathsf{p}(h)$ through the equations of the ideal VSS functionality. In other words, in order to complete the description of the simulator we need to show that SIM can determine such a polynomial p based on its inputs s', G_A, s_A, p_A such that $\mathsf{p}(h)$ equals p_h as defined by the above system of equations.

Before defining p, we recall the following lemma whose simple proof is standard and omitted:

Lemma 3 *Let S be such that $|S| \geq t+1$ and let $\{\mathsf{g}_h, \mathsf{h}_h\}_{h \in S}$ be a set of $2 \cdot |S|$ polynomials of degree t. Then, $\mathsf{g}_h(h') = \mathsf{h}_{h'}(h)$ for all $h, h' \in S$ holds if and only if there exists a unique polynomial $\mathsf{f} \in \mathbb{F}_t[X,Y]$ such that $\mathsf{f}(\cdot, h) = \mathsf{g}_h$ and $\mathsf{f}(h, \cdot) = \mathsf{h}_h$ for all $h \in S$.*

For $T \subseteq H$, $|T| \geq t+1$, define $\mathsf{interpolate2}_T(s')$ to be the (unique) polynomial $\mathsf{f} \in \mathbb{F}_t[X,Y]$ such that $\mathsf{f}(\cdot, h) = \mathsf{g}_h$ and $\mathsf{f}(h, \cdot) = \mathsf{h}_h$ for all $h \in T$ (if it exists), and $\perp$ otherwise or if $s'[h] = \perp$ for some $h \in T$. Also, given $C \subseteq [n]$, define

$$\mathsf{interpolate2}_C(s') = \bigvee \{\mathsf{interpolate2}_S(s') \ :\ S \subseteq C, |S| \geq |C| - t\}\ .$$

Note that since $|C| \geq n - t$ and $n \geq 4t+1$, $\mathsf{interpolate2}_C(s') \neq \top$. Indeed, for any two $S, S' \subseteq C$ such that both $\mathsf{interpolate2}_S(s')$ and $\mathsf{interpolate2}_{S'}(s')$ differ from $\perp$, we have $|S \cap S'| \geq t+1$ and hence $\mathsf{interpolate2}_S(s') = \mathsf{interpolate2}_{S \cap S'}(s') = \mathsf{interpolate2}_{S'}(s')$ by Lemma 3. We finally define the polynomial $\mathsf{p} = \widetilde{f}(\cdot, 0)$, where

$$\widetilde{f} = \bigvee_{C \subseteq [n], |C| \geq n-t} \mathsf{clique}_C(G) \wedge \mathsf{interpolate2}_C(s')\ . \tag{7}$$

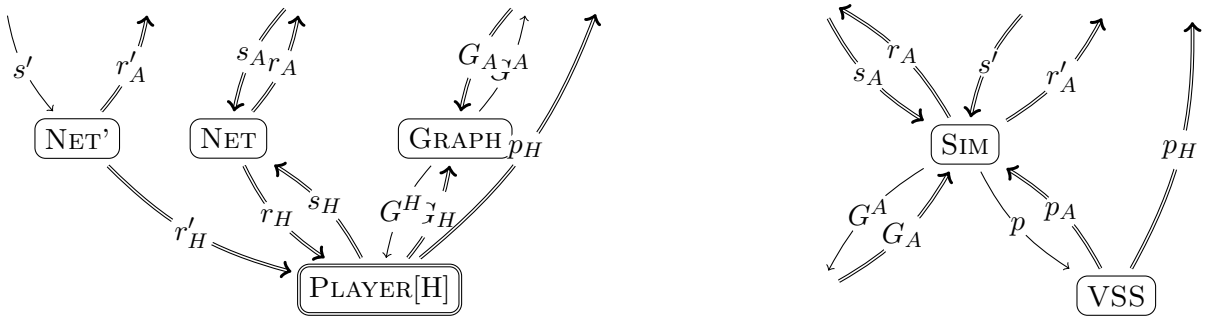

Figure 21: Security of the VSS protocol when the dealer is dishonest.

We first prove that $\mathsf{p} < \top$: To this end, assume that $\mathsf{p} \neq \bot$. Then, $\widetilde{f} \neq \bot$, and there must exist $C \subseteq [n]$ such that $\mathsf{clique}_C(G) = \top$. Let $S = C \cap H$. Note that for all $h, h' \in S$, since $G[h, h'] = \top$, it must be that $\mathsf{h}_h(h') = \mathsf{g}_{h'}(h)$. Therefore, since $|S| \geq n - 2t > 2t + 1$, by Lemma 3, there exists a unique polynomial f_C such that $\mathsf{f}(\cdot, h) = \mathsf{g}_h$ and $\mathsf{f}(h, \cdot) = \mathsf{h}_h$ for all $h \in S$, and by the above

$$\mathsf{f}_C = \mathsf{interpolate2}_C(s') = \mathsf{interpolate2}_{C \cap H}(s') \ .$$

Now assume that there exist two such cliques C and C', with $S = C \cap H$ and $S' = C' \cap H$. Then, since

$$\left|S \cap S'\right| = |S| + \left|S'\right| - \left|S \cup S'\right| \geq 2(|C| - |A|) - |H| \geq n - 3\,|A| \geq t + 1 \ , \tag{8}$$

by Lemma 3, we necessarily have $\mathsf{f}_C = \mathsf{f}_{C'} = \widetilde{\mathsf{f}}$.

Finally, it is easy to see that $\mathsf{p}(h) = p_h$ for all $h \in H$. Namely, if there exists $C \subseteq [n]$ with $\mathsf{clique}_C(G) = \top$, then $r_h[h'] = \mathsf{g}_{h'}(h) = \widetilde{\mathsf{f}}(h, h')$ for all $h' \in C \cap H$, and therefore $\mathsf{o}_h = \mathsf{interpolate}_C(r_h) = \mathsf{interpolate}_{C \cap H}(r_h) = \mathsf{f}(h, \cdot)$, and thus $p_h = \mathsf{o}_h(0) = \widetilde{\mathsf{f}}(h, 0) = \mathsf{p}(h)$.

We therefore conclude that the real system is equivalent to (SIM| VSS) where SIM is the simulator defined by the following equations:

$$\begin{array}{l}
\textsc{Sim}(s', G_A, s_A, p_A) = (r'_A, r_A, G^A, \mathsf{p}): \\
\quad r'_a = s'[a] \qquad\qquad (a \in A) \\
\quad r_a[h] = \mathsf{g}_h(a) \qquad\qquad (h \in H, a \in A) \\
\quad r_a[a'] = s_{a'}[a] \qquad\qquad (a, a' \in A) \\
\quad G[h, h'] = \mathsf{eq}(\mathsf{g}_{h'}(h), \mathsf{h}_h(h')) \qquad (h, h' \in H)
\end{array}$$

$$\begin{array}{l}
G[h, a] = \mathsf{eq}(s_a[h], \mathsf{h}_h(a)) \qquad (a \in A, h \in H) \\
G[a, j] = G_a[j] \qquad\qquad (a \in A, j \in [n]) \\
\widetilde{\mathsf{f}} = \bigvee_{\substack{C \subseteq [n] \\ |C| \geq n - t}} \mathsf{clique}_C(G) \wedge \mathsf{interpolate2}_C(s') \\
\mathsf{p} = \widetilde{\mathsf{f}}(\cdot, 0)
\end{array}$$

5 Conclusions

Recognizing the inherent hardness of delivering security proofs for complex cryptographic protocols that are both precise and intuitive within existing security frameworks, we have presented a new framework to study the security of multi-party computations based on equational descriptions of interactive processes. Our framework allows a simple and intuitive, yet completely formal, description of interactive processes via sets of equations, and its foundations rely on tools from programming-language theory and domain theory. Beyond its simplicity, our framework completely

avoids explicit addressing of non-determinism within cryptographic security proofs, making security proofs a matter of simple equational manipulations over precise mathematical structures. As a case study, we have presented simple security analyses of (variants of) two classical asynchronous protocols within our framework, Bracha's broadcast protocol [8] and the Ben-Or, Canetti, Goldreich VSS protocol [5].

We are convinced that our work will open up the avenue to several directions for future work. First off, while the results in this paper are presented for the special case of perfect security, a natural next step is to extend the framework to statistical and even computational security. Moreover, while the expressiveness of our framework (i.e., the monotonicity restrictions on protocols) remains to be thoroughly investigated, most distributed protocols we examined so far, seemed to admit a representation within our framework, possibly after minor modifications which often resulted in a simpler protocol description. For this reason, our thesis is that this holds true for *all* protocols of interest, and that non-monotonicity, as a source of unnecessary complexity and proof mistakes, should be avoided whenever possible.

References

[1] S. Abramsky and A. Jung. *Handbook of Logic in Computer Science*, volume III, chapter Domain theory, pages 1–168. Oxford University Press, 1994.

[2] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A general composition theorem for secure reactive systems. In *TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 336–354, 2004.

[3] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority. In *CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 589–590, 1989.

[4] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In *ACM Conference on Computer and Communications Security*, pages 257–266, 2008.

[5] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *STOC*, pages 52–61, 1993.

[6] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC '88*, pages 1–10, 1988.

[7] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In *Financial Cryptography (FC 2009)*, volume 5628 of *Lecture Notes in Computer Science*, pages 325–343, 2009.

[8] Gabriel Bracha. An asynchronous [(n-1)/3]-resilient consensus protocol. In *PODC '84*, pages 154–162, 1984.

[9] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[10] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS '01*, pages 136–145, 2001.

[11] Ivan Damgård, Marcel Keller, E. Larraia, C. Miles, and Nigel P. Smart. Implementing aes via an actively/covertly secure dishonest-majority mpc protocol. *IACR Cryptology ePrint Archive*, 2012.

[12] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC '87*, pages 218–229, 1987.

[13] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989.

[14] Carl A. Gunter. *Semantics of programming languages - structures and techniques*. Foundations of computing. MIT Press, 1993.

[15] Dennis Hofheinz and Victor Shoup. Gnuc: A new universal composability framework. *IACR Cryptology ePrint Archive*, 2011.

[16] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information processing*, pages 471–475, Stockholm, Sweden, Aug 1974. North Holland, Amsterdam.

[17] Vladimir Kolesnikov and Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. In *Financial Cryptography (FS 2008)*, volume 5143 of *Lecture Notes in Computer Science*, pages 83–97, 2008.

[18] Ralf Küsters. Simulation-based security with inexhaustible interactive turing machines. In *19th IEEE Computer Security Foundations Workshop, (CSFW-19 2006)*, pages 309–320, 2006.

[19] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.

[20] Ueli Maurer and Renato Renner. Abstract cryptography. In *Innovations in Computer Science - ICS 2010*, pages 1–21, 2011.

[21] Ueli M. Maurer. Indistinguishability of random systems. In *EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 110–132, 2002.

[22] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In *CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404, 1991.

[23] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–, 2001.

[24] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267, 2009.

[25] N. Saheb-Djahromi. Cpo's of measures for nondeterminism. *Theor. Comput. Sci.*, 12:19–37, 1980.

[26] D. A. Schmidt. *Denotational Semantics*. Allyn and Bacon, 1986.

[27] J. E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1977.

[28] Glynn Winskel. *The formal semantics of programming languages - an introduction*. Foundation of computing series. MIT Press, 1993.

[29] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS '82*, pages 160–164, 1982.