

# Non-Determinism in Multi-Party Computation (Abstract)

Michael Backes\*

Birgit Pfitzmann<sup>†</sup>

Michael Waidner<sup>‡</sup>

## Abstract

Outside security, non-determinism is an important tool for specifying systems without fixing unnecessary details. In security, however, normal refinement of non-deterministic specifications is usually not applicable, in particular because it may invalidate secrecy properties. Especially simulatability-based security notions seem to require detailed deterministic or probabilistic specifications. We show how one can nevertheless use the reactive simulatability (RSIM) framework to address non-determinism. In particular we survey its *generic distributed scheduling* for treating the non-determinism of asynchronous execution, discuss the experiences we made with this, and how it encompasses other recent scheduling approaches. We also show how property-based specifications can play the role of highest-level non-determinism in the RSIM context, and how functional non-determinism of machines can be captured by the system-from-structure derivations as well as by call-outs to the adversary or more general resolvers.

## 1 Introduction

In normal design processes, non-determinism is an important tool for initially specifying systems without fixing unnecessary details. Outside security, there are many well-accepted notions of refinement of non-deterministic specifications, e.g., for program verification and distributed systems. In security, however, normal refinement is usually not applicable, in particular because it may invalidate secrecy properties. In cryptography, this is particularly visible in simulatability-based approaches at system specification and refinement, and it may even seem inherent that such specifications cannot be non-deterministic: In cryptographic simulatability definitions, ultimately the views of certain parties are compared in the sense of computational indistinguishability [9]. For this, the views must be families of probability distributions. Hence at this point in the definition, all non-determinism in the specification or implementation of the protocol must have been resolved deterministically or probabilistically. As an example, we sketch the general RSIM definition from [7] in Figure 1.

However, we will show that this is no fundamental problem for using non-determinism in designing cryptographic multi-party protocols, because there are ample opportunities to resolve initial non-determinism within the overall formula in which the view comparison occurs.

## 2 Scheduling – Non-Determinism by Asynchronous Execution

A particular question that has recently found renewed interest is how the inherent non-determinism in the execution order of asynchronous systems can be resolved in cryptographic multi-party computation.

---

\*Saarland University, Saarbrücken, Germany, backes@cs.uni-sb.de

<sup>†</sup>IBM Zurich Research Lab, Switzerland, bpf@zurich.ibm.com

<sup>‡</sup>IBM Software Group, Somers, USA, wmi@us.ibm.com

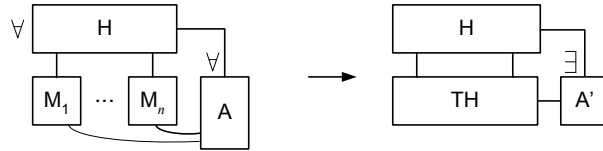


Figure 1: Overview of general reactive simulatability (RSIM). An implementation (often called real system) is on the left, the corresponding specification (ideal system) on the right. Here the views of  $H$  must be indistinguishable.

## 2.1 Typical Scheduling Patterns

In the distributed systems community, this resolution is usually done by a separate, arbitrary full-information scheduler, i.e., a component that at each step knows the entire system state and can base its next scheduling decision on that. For typical computational cryptographic systems, this gives the scheduler too much power. For instance, the scheduler can see internal secrets of honest parties and encode them in scheduling decisions that the adversary can learn [5]. In cryptography, the most typical scheduling pattern is therefore that the adversary schedules everything. However, in some cases even this scheduling is too strong.

- When *liveness*, *availability*, or *fairness* properties of a protocol are considered, some fairness of the underlying scheduling must usually be assumed, because certain messages have to reach their recipients. Cryptographic versions of such properties and corresponding schedulers were introduced in [3].
- *Covert channel prevention* is needed when the absence of information flow between certain parties is considered. Here an adversary should not be able to encode the information whose flow is otherwise prevented into scheduling information. Cryptographic versions of such definitions were introduced in [1].
- *Subprogram-like machine combination*. Proofs of distributed systems often use splitting and recombination of machines with properties such as associativity. Process algebras like  $\pi$ -calculus (first used cryptographically in [5]) have many such properties predefined, and also for the probabilistic IO automata (PIOA) model in the RSIM framework such properties were shown. If every machine recombination would make different channels external and thus open them to adversarial scheduling, it would significantly hinder such modular proofs. Hence it is useful to allow immediate local scheduling of certain channels [5, 8].
- *Adversary-scheduled secure channels*. Secure channels are sometimes needed in initial protocol phases such as the exchange of symmetric master keys. It seems realistic that even if an adversary cannot read and modify messages on such channels, it may be able to influence the channel speed. This is adversarial scheduling, but for channels where the adversary is neither the sender nor the recipient.

## 2.2 Generic Distributed Scheduling

The generic distributed scheduling from the RSIM framework [8] allows all the cases described above, alone or in combination, as well as many other scheduling mechanisms that one might come up with. All this is done with very little overhead compared with standard machine and scheduling definitions. The following two principles are used:

- Schedulers are normal machines.
- For each channel, one can designate which machine schedules it.

Thus the only addition to a concrete specification or a system definition, compared with a system model with fixed scheduling, is that for each channel, not only a sender and a recipient are designated, but also the scheduler. Clearly, all the cases from Section 2.1 can easily be defined as patterns in this model. For specifications or systems that use one of these patterns the scheduling can largely be given by reference to the pattern. (Clearly if, e.g., some channels are scheduled locally and others by the adversary, then one still has to designate which channels are which.) The fact that schedulers are normal machines also makes it easy to define one or many schedulers (e.g., a scheduler hierarchy or local schedulers), to provide each scheduler with arbitrary information, and to define arbitrarily how much an adversary learns from a scheduler (typically nothing beyond what it learns from other sources).

### 2.3 Discussion and Comparison of Scheduling Models

As shown above, special schedulers are usually needed in cryptography if adversarial scheduling is too strong. I.e., given the “normal” machines, a limited set of possible schedulers is defined, e.g., all fair ones that schedule certain channels. The overall set of behaviours of such a system is a subset of the behaviors that can occur with adversarial scheduling, because everything a separate scheduler and an adversary can do could easily be done by a combined adversary too. Thus every security *property* that can be proved for adversarial scheduling also holds for the restricted scheduling, but not vice versa.

Concerning *simulatability* definitions, specific scheduling patterns fall under the existing RSIM definitions and theorems (in particular composition) as long as all involved schedulers can be classified in the quantifier orders as either normal machines, adversaries, or honest users.<sup>1</sup> For all patterns above this is true. Hence generic distributed scheduling has been very useful for treating all these cases with only one set of definitions and theorems. If other quantifier orders are desired (quantifier orders are separate from the basic models in the RSIM framework and many variations have already been compared, starting with [6]), similar theorems need to be reproved. These proofs can follow the same graphical meta-structure as used for the RSIM proofs. For instance, we believe that a composition theorem for the quantifier order  $\forall A \exists A' \forall H \forall S \exists S'$ , where  $S$  and  $S'$  are the schedulers in the implementation and specification, (suggested by Robert Segala) can be proved without any serious change to the proof for general RSIM in [8].<sup>2</sup>

It is even possible with generic distributed scheduling to define full-information schedulers or super-polynomial schedulers for certain system parts (the former simply by letting the machines in this system part send their entire new state to their scheduler in each step), while keeping the adversary polynomial-time and with realistic information. Then more behaviors than with adversarial scheduling are possible. However, we do not believe that there are many cryptographically interesting uses of this: If the machines that are scheduled with full information contain secrets, in most cases the scheduling can leak these secrets to the adversary. If they do not contain secrets, the full-information scheduler cannot do much more than a normal adversary.

All other scheduling models proposed in the literature can, at least on this informal level, be easily mapped into generic distributed scheduling. Let us show this for a particular model [4] that was recently

---

<sup>1</sup>The RSIM framework, in contrast to some related frameworks, allows quantification also over normal machines by considering systems consisting of many possible actual structures; Derivations of such a system from one “intended” structure are an additional definition layer that is currently mostly used for trust models, but can also be used for adding schedulers.

<sup>2</sup>Recall that for security *properties* the structure with separate  $S$  and  $A$  is *weaker* than the standard structure. However, if weaker structures on both sides are compared, there is no trivial relation to standard definitions.

built without any look at related literature (as the introduction to an earlier public version shows and several authors admitted); it can thus count as independent confirmation of the generality of the generic distributed scheduling from RSIM. Like the RSIM framework, it uses PIOAs. However, it schedules message output instead of message arrival. Hence the models are not easy to map formally, but neither the task PIOA authors we spoke to nor we currently think that this makes a significant difference. The “tasks” in that model correspond to the channels in the RSIM framework and in the cryptographic  $\pi$ -calculus from [5]: Like channels, these tasks group messages that the scheduler can schedule without knowing the exact message content. There is one specific scheduler (called “task schedule”), separate from the adversary. It does not get information from the running system and is deterministic. In the RSIM framework such a scheduler would be represented as a deterministic machine without normal in- and output channels. Note that this is a very weak scheduling model for security *properties*. Another variant with full-information schedulers for certain system parts, corresponding to the case we described above, is sketched.

### 3 Functional Non-Determinism in Simulatability Definitions

In design processes, functional non-determinism is rather more important than asynchrony. Typically, non-deterministic machines are used to leave certain choices open, which can be fixed by later refinement. If the design process for a system that contains cryptography is performed entirely with refinement steps that are proven correct with a simulatability definition, such a use of non-determinism may at first glance seem impossible because currently the basic machines in all such frameworks are probabilistic (which includes deterministic, but not non-deterministic).

Nevertheless, functional non-determinism can be handled in two ways. First, in the RSIM framework, one could easily allow non-determinism in the intended structures from which systems (sets of actual structures) are derived. Standard refinement notions could be used for this derivation. As the view comparison as in Figure 1 is only performed on the actual structures, it is not affected.

Secondly, one can call out potential non-deterministic choices and leave them to the adversary. In [7] (and the corresponding longer reports) this falls under the method for identifying “tolerable imperfections”. It was, e.g., used to leave open in a specification how many rounds a synchronous protocol takes, or to allow that the length of a message leaks partially or entirely. The same technique was explicitly or implicitly used by many subsequent protocol specifications used with simulatability definitions. One could generalize these techniques similar to the scheduling, i.e., generally introduce a class of components, say “resolvers”, that take choices left open by others. In the basic RSIM machine and execution model this simply makes no difference, but one could again come up with different quantifier orders between adversaries and other resolvers.

### 4 Non-Determinism by Property-based Specifications

The first specifications in a real-life design process are typically non-deterministic in a more fundamental way than non-deterministic machines: They consist of individual requirements. Ideally those are solicited from human stakeholders and then formalized in some logic, e.g., temporal logic for typical functional requirements on distributed systems. In security, additional requirements are secrecy requirements for certain data or message types. Such properties can also be formalized at an abstract level (one might call this “ideal properties”) and given a cryptographic semantics. It can be shown that the refinement of such properties by abstract system specifications (“ideal systems”) and later simulatability-based refinement to cryptographic systems are compatible. This was first formalized for integrity properties in [7]; secrecy properties exist in

more variants, e.g., [2].

## 5 Conclusion

Non-determinism is an important aspect of general system design processes. We have shown how, in spite of the probabilistic nature of typical cryptographic definitions, non-determinism can play its role also for systems containing cryptographic protocols. In particular the following four mechanisms can be used (roughly ordered from high-level to low-level specifications): cryptographic semantics for property specifications, system definitions via derivations from non-deterministic structures, call-outs of non-deterministic choices to the adversary, and generic distributed scheduling for the general low-level resolution of non-deterministic choices.

**Acknowledgments.** We thank Ling Cheung, Joshua Guttman, Nancy Lynch, John Mitchell, Roberto Segala, and Matthias Schunter for interesting discussions. This work is partially supported by the European Commission through the IST Programme under Contract IST-2002-507932-NOE ECRYPT. In this abstract we could not give a complete literature overview; we refer to the cited papers for more prior and related work.

## References

- [1] M. Backes and B. Pfitzmann. Computational probabilistic non-interference. In *Proc. 7th ESORICS*, volume 2502 of *LNCS*, pages 1–23. Springer, 2002.
- [2] M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Transactions on Dependable and Secure Computing*, 2(2):109–123, 2005.
- [3] M. Backes, B. Pfitzmann, M. Steiner, and M. Waidner. Polynomial fairness and liveness. In *Proc. 15th IEEE CSFW*, pages 160–174, 2002.
- [4] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Task-structured probabilistic I/O automata, 2006. To appear as Technical Report, RU Nijmegen, 2006. Preliminary version of April 2006 at Ling Cheung’s homepage.
- [5] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proc. 5th ACM CCS*, pages 112–121, 1998.
- [6] B. Pfitzmann, M. Schunter, and M. Waidner. Secure reactive systems. IBM Research Report RZ 3206, May 2000. [http://www.semper.org/sirene/publ/PfSW1\\_00ReactSimulIBM.ps.gz](http://www.semper.org/sirene/publ/PfSW1_00ReactSimulIBM.ps.gz).
- [7] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM CCS*, pages 245–254, 2000.
- [8] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symp. on Security & Privacy*, pages 184–200, 2001. More model details in IACR ePrint Report 2004/082 with M. Backes.
- [9] A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd FOCS*, pages 80–91, 1982.