# The Layered Games Framework

**for Specifications and Analysis of Security Protocols**

**Full version, draft of June 22, 2008**

**Extended abstract version in TCC'08, full version in eprint [31]**

Amir Herzberg and Igal Yoffe

Computer Science Department, Bar Ilan University,
Ramat Gan, 52900, Israel
{herzbea,ioffei}@cs.biu.ac.il

**Abstract.** We establish rigorous foundations to the use of modular, layered design for building complex distributed systems, resilient to failures and attacks. Layering is key to the design of the Internet and other distributed systems. Hence, solid, theoretical foundations are essential, especially when considering adversarial settings, such as for security and cryptographic protocols.
We use games to define specifications for each layer. A protocol realizes a layer (over some lower layer), if it 'wins', with high probability, a specified game, when running over any implementation of the lower layer. This is in contrast to existing frameworks allowing modular design of cryptographic protocols, e.g. Universal Composability [15], where protocols must emulate an *ideal functionality.* Ideal functionalities are a very elegant method for specifications, but we argue that often, game-based specifications are more appropriate. In particular, it may be hard to design the 'correct' ideal functionality, and avoid over-specification ('forcing' the protocol to follow a particular design) and under-specification (e.g., allowing protocols that work reasonably only for worst-case adversary but poorly for realistic adversaries); see details within.
Our definitions include the basic concepts for modular, layered design: *protocols, systems, configurations, executions*, and *models.* We also define three basic relations: *indistinguishability* (between two systems), *satisfaction* (of a model by a system), and *realization* (by protocol, of one model over another model).
We prove several basic properties, including the *layering lemma* and the *indistinguishability lemma.* The layering lemma shows that given protocols $\{\pi_i\}_{i=1}^{u}$, if every protocol $\pi_i$ realizes model $\mathcal{M}_i$ over model $\mathcal{M}_{i-1}$, then the composite protocol $\pi_{1||\ldots||u}$ realizes model $\mathcal{M}_u$ over $\mathcal{M}_0$. This allows specification, design and analysis of each layer independently, and combining the results to ensure properties of the complete system.

## 1 Introduction

The design and analysis of complex distributed systems, such as the Internet and applications using it, is an important and challenging goal. Such systems are designed in modular fashion, typically by decomposing the system into multiple *layers*. Some of the well known layered network architectures include the 'OSI 7-layers reference model' and the 'IETF 5-layers reference model' (also referred to as the Internet or TCP/IP model, and often extended with two optional security sub-layers - TLS and IP-Sec); see e.g. Kurose and Ross [33]. The present work is part of an effort, described in Herzberg and Yoffe [28], to extend such layered networking architectures, to support secure e-commerce applications. Figure 1 shows the five IETF layers, together with two optional security sub-layers, and the four secure e-commerce layers of [28].

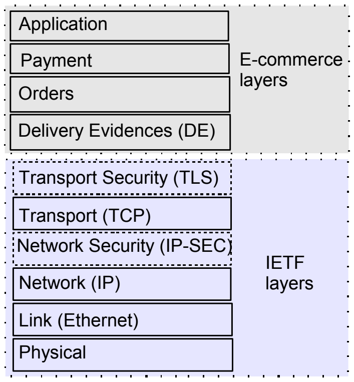


**Fig. 1.** IETF and e-commerce layers; (optional) IETF security sub-layers marked with dotted contour. Layer $i$ expects the (lower) layer $i-1$ to fulfill some (lower) *layer model*. For example, in Herzberg and Yoffe [30] we define the *communication* layer model $\mathcal{M}_{\text{Comm}}$, for the service expected by the Delivery Evidences (DE) layer, operating on top of the IETF layers.

Layered (or modular) architectures allow to specify, design, analyze, implement and test protocols for each layer, independently of protocols for other layers. This is based on the paradigm of *lower layers abstraction*. For each layer $i$, we define a *model* $\mathcal{M}_i$. The model $\mathcal{M}_i$ defines specifications, not just for the protocol $\pi_i$ for layer $i$, but for the entire *system* $\Gamma_i$, comprising of copies of $\pi_i$ running in multiple processors, all communicating via the 'lower layer' system $\Gamma_{i-1}$.

Given protocol $\pi_i$ and (lower layer) system $\Gamma_{i-1}$, let $\Gamma_i \equiv \begin{bmatrix} \pi_i \\ \Gamma_{i-1} \end{bmatrix}$ be the composition of (multiple instances of) $\pi_i$ running over $\Gamma_{i-1}$ (see Figure 5 (b)).

Protocol $\pi_i$ *realizes model* $\mathcal{M}_i$ *over model* $\mathcal{M}_{i-1}$, if $\Gamma_i \equiv \begin{bmatrix} \pi_i \\ \Gamma_{i-1} \end{bmatrix}$ satisfies $\mathcal{M}_i$, provided that $\Gamma_{i-1}$ satisfies $\mathcal{M}_{i-1}$. We write this as: $\mathcal{M}_i \vdash \begin{bmatrix} \pi_i \\ \mathcal{M}_{i-1} \end{bmatrix}$.

Practical, standard specifications for network layers, do not fully define the model of each layer. Precise specifications of layer models is essential, to allow proof that a given protocol realizes one model $\mathcal{M}_i$ over another model $\mathcal{M}_{i-1}$. Precise specifications and analysis (proofs) are especially critical for security and cryptographic protocols.

There are many works defining rigorous security specifications for cryptographic protocols, and proving security of such protocols. These works basically follow one of two approaches: the *game/experiment-based approach*, or the simulation-based approach (referring to simulation of ideal system/scenario by a protocol running in 'real' scenario).

In the simulation-based approach, the specifications consist of a special 'ideal' system $I$, modeling the desired operation of the system if it was implemented by a single machine (rather than by multiple communicating parties). Roughly speaking, protocol $\Pi$ is secure if, for any adversary $A$, there is some other ('simulator') adversary $S$ such that the interaction of $\Pi$ with $A$ is indistinguishable from the interaction of $I$ with $S$. The simulation-based approach was initiated in the seminal works of Yao [39] and Goldreich et al. [23], showing ideal functionalities and their realizations, for computing any function.

In the game/experiment approach, specifications consist of a special *experiment* program, and of a well-defined win/loss game between the experiment, the system and an adversary; a system satisfies the specifications if it wins in a sufficient fraction of the executions. Such experiments (and games) were defined for many cryptographic schemes, and appropriate protocols were proven secure, e.g. encryption [24, 7, 8].

To use protocols in practice, security should be guaranteed when the protocol is composed (run) with arbitrary other protocols, and in particular used as a module by other protocols. However, not all notions of security are preserved under such general compositions. Several notions of security following the simulation-based approach, where shown to be preserved under general compositions, e.g. *universal composability* (UC) by Canetti [15], *reactive simulatability* by Backes, Pfitzmann, and Waidner [4], Pfitzmann and Waidner [37], and *observational equivalence* by Lincoln, Mitchell, Mitchell, and Scedrov [35]. In this work we show, for the first time, a game/experiment security notion, which is also preserved under general compositions.

Since comparable composition results already exist for simulation-based security notions, what is the importance of a composable game/experiment security notion? The basic reason, is that game/experiment definitions seem often more natural[1]. In particular, consider tasks for which the 'ideal' centralized solution is

[1] One may be tempted to define a 'flexible ideal functionality', that is built-in with the experiment, and accept from the adversary an arbitrary program; the ideal functionality would then run the program, continuously validating that its behavior does

hard to define or to realize, e.g. resource allocation problems. In fact, we do not know whether there is some general relation among these two types of security definitions, e.g. whether all game-based security notions can be shown equivalent to corresponding simulation-based notion (for some ideal functionality).

A possibly more practical motivation is that many existing security proofs of cryptographic primitives using experiments and games, can be viewed as special cases of our security notion, and therefore can be used as modules by cryptographic protocols. In fact, some cryptographic primitives have secure implementations for game-based specifications, where the corresponding ideal functionalities are not realizable, see Datta, Derek, Mitchell, Ramanathan, and Scedrov [19], Canetti, Kushilevitz, and Lindell [17], Canetti and Fischlin [14]. Note that Backes et al. [3] define a relaxed notion of *conditional reactive simulatability*, where simulation is required only if the environment fulfills some constraints; this allows them to circumvent such impossibility results, however at price of added complexity. This is very different from the restrictions placed on *lower layers* (modules) in the layered games framework.

Furthermore, there are common scenarios, where a game-based security definitions seem to have advantages. One such advantage is, when comparing performance under adversarial settings, in particular for Denial of Service (DoS) attacks. With simulation-based approach, the protocol is evaluated only for its operation against the strongest adversary. In contrast, an experiment can validate the performance of the protocol, as function of the resources of the adversary. The simulation-based approach is especially problematic in handling realistic network delays and failures. The standard solution is to design the ideal functionality to ask the adversary (simulator) to define such delays and failures (and then simulate the same delays and failures as in the real execution). This is an elegant solution, however, it has drawbacks. In particular, the adversary is exposed to the existence of communication; this is realistic in many scenarios, however in common networking scenarios, the adversary is 'blind' (can inject spoofed traffic but not eavesdrop on communication), and it is unjustified to assume it can know when a message was sent.

To illustrate this advantage of game-based specifications, i.e. the ability to model 'blinded traffic' scenarios (where the mere existence of communication is secret), Algorithm 1 shows the experiment from [27] to define a blinded-traffic asynchronous datagram communication channel. Note that many network security mechanisms assume such channels (often implicitly).

Finally, sometimes it is hard to avoid over-specification with the simulation-based approach. In particular, consider 'external' lower layers, which are not (fully) under the designer's control, such as modeling of physical systems. When designing protocols to operate over such layers, we must make some simplifying assumptions about their behavior; however, any particular 'ideal functionality' may assume behaviors which differs from reality. Namely, while we can define tests which identify unreasonable behaviors, but we may not be able to write an

not violate the tests. However, many tests cannot be validated (efficiently) during execution.

```
On initialization do
    $b \overset{R}{\leftarrow} \perp$ // $b \in \{0, 1, \perp\}$ is the secret, random choice
    $SENT \leftarrow \perp$ // to contain the ‘challenge’ $(s, d, m)$, where $s, d$ are
        processor identifiers, $m$ is a message
On $A2E(\mathsf{S}, s, d, m)$ do
    $\mathsf{S}_s(d, m)$ // adversary instructs $s$ to send $m$ to $d$
On $A2E(\mathsf{select}, s, d, m)$ // adversary selects ‘challenge’
do
    $SENT \leftarrow (s, d, m)$
    $b \overset{R}{\leftarrow} \{0, 1\}$
    if $b = 1$ then
        $\mathsf{S}_s(d, m)$ // adversary has to guess $b$
On $A2E(\mathsf{guess}, g)$ do
    if $b = g \neq \perp$ then
        adversary wins
    else
        adversary losses
    Abort experiment
On $\mathsf{R}_d(s, m)$ do
    if $(s, d, m) \neq SENT$ then
        $E2A(\mathsf{R}, s, d, m)$
```

**Algorithm 1**: Blinded-traffic datagram channel experiment $\mathbf{Exp}_{\mathsf{BTD}}$

'ideal functionality' that has exactly the set of (all) reasonable behaviors. In such cases, a 'negative specification', defining tests for unacceptable systems, maybe more appropriate than a 'positive specification' as with ideal functionalities. Notice that designers of practical network layers, *intentionally* avoid unnecessary restrictions of lower layers. This is critical, to allow flexible, independent design and implementation of each layer. Over-specification is considered harmful, see e.g. Bradner [12].

Our results. In addition to presenting the layered games framework, in this paper we prove few basic results regarding it. In particular, we prove the *layering lemma*, which shows that layers compose well: given protocols $\{\pi_i\}_{i=1}^{u}$, if every protocol $\pi_i$ realizes model $\mathcal{M}_i$ over model $\mathcal{M}_{i-1}$, then the composite protocol $\pi_{1||\ldots||u}$ realizes model $\mathcal{M}_u$ over $\mathcal{M}_0$. We only handle compositions by layering, but this is sufficient for many practical network architectures and protocols.

We also prove several other properties, such as the *indistinguishability lemma*. The indistinguishability lemma shows that if two systems $\Gamma_L, \Gamma_R$ are indistinguishable, and $\Gamma_L$ satisfies some model $\mathcal{M}$, then $\Gamma_R$ also satisfies $\mathcal{M}$. This allows use of complex reduction arguments, e.g. hybrid arguments, to facilitate proofs of security of complex protocols; see for example [9].

Organization. In Section 2 we define a specific execution model which we use in the framework, consisting of protocols, systems, configurations (of protocols and systems), and executions (of configurations). In Section 3 we define models and realizations, and present the layering lemma. In Section 4 we define emulation and indistinguishability of systems, based on the indistinguishability game, and the indistinguishability lemma. We conclude and discuss future work in Section 5.

In the rest of this introduction, we present the following subsections. In subsection 1.1, we give an example of layering of two secure electronic commerce layers (combining the order layer over the deliver evidences layer). In subsection 1.2 we discuss the need for a theoretical basis for layering. Finally, in subsection 1.3, we discuss related works.

## 1.1 Example: secure e-commerce layers

In Herzberg and Yoffe [30] we define the *communication* model $\mathcal{M}_{\mathsf{Comm}}$; this is a formalization of a commonly-used model of communication systems such as the Internet, using a reliable transport protocol like TCP over an unreliable lower-layer service (e.g. the Internet Protocol). We then define the *delivery evidences* model $\mathcal{M}_{\mathsf{DE}}$, and show a protocol $\pi_{\mathsf{DE}}$ s.t. $\mathcal{M}_{\mathsf{DE}} \vdash \begin{bmatrix} \pi_{\mathsf{DE}} \\ \mathcal{M}_{\mathsf{Comm}} \end{bmatrix}$. Similarly, in Herzberg and Yoffe [29] we define the *orders* model $\mathcal{M}_{\mathrm{Orders}}$, and show protocol $\pi_{\mathsf{Orders}}$ s.t. $\mathcal{M}_{\mathsf{Orders}} \vdash \begin{bmatrix} \pi_{\mathsf{Orders}} \\ \mathcal{M}_{\mathsf{DE}} \end{bmatrix}$. Using the fundamental lemma of layering, the composite protocol $\pi_{\mathsf{DE}||\mathsf{Orders}}$ realizes the orders model directly over the communication model, i.e. $\mathcal{M}_{\mathsf{Orders}} \vdash \begin{bmatrix} \pi_{\mathsf{DE}||\mathsf{Orders}} \\ \mathcal{M}_{\mathsf{Comm}} \end{bmatrix}$.

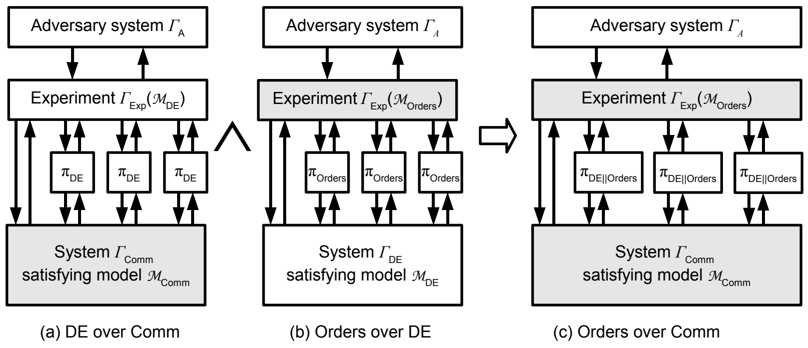

Adversary system $\Gamma_A$
Experiment $\Gamma_{Exp}(\mathcal{M}_{DE})$
$\pi_{DE}$
System $\Gamma_{Comm}$ satisfying model $\mathcal{M}_{Comm}$
(a) DE over Comm
Experiment $\Gamma_{Exp}(\mathcal{M}_{Orders})$
$\pi_{Orders}$
System $\Gamma_{DE}$ satisfying model $\mathcal{M}_{DE}$
(b) Orders over DE
$\pi_{DE||Orders}$
(c) Orders over Comm

**Fig. 2.** Layering of realizations of the Order and Delivery Evidences (DE) layers. This figure contains three simple configurations ((a), (b) and (c)), each of them containing three copies of a protocol ($\pi_{\mathsf{DE}}$, $\pi_{\mathsf{Orders}}$ and $\pi_{DE||Orders}$, respectively). Part (a) illustrates that $\pi_{\mathsf{DE}}$ realizes $\mathcal{M}_{\mathsf{DE}}$ over $\mathcal{M}_{\mathsf{Comm}}$; part (b) illustrates that $\pi_{\mathsf{Orders}}$ realizes $\mathcal{M}_{\mathsf{Orders}}$ over $\mathcal{M}_{\mathsf{DE}}$; and part (c) illustrates that protocol $\pi_{\mathsf{DE||Orders}}$ realizes $\mathcal{M}_{\mathsf{Orders}}$ over $\mathcal{M}_{\mathsf{Comm}}$.

This is illustrated in Figure 2. In each of the three parts of the figure, we see three nodes (processors) running a protocol, realizing one model over another. For example, in (a) we see protocol $\pi_{\mathsf{DE}}$, which realizes model $\mathcal{M}_{\mathsf{DE}}$ over model $\mathcal{M}_{\mathsf{Comm}}$. Namely, when $\pi_{\mathsf{DE}}$ is combined over any system $\Gamma_{\mathsf{Comm}}$ that satisfies $\mathcal{M}_{\mathsf{Comm}}$, the resulting combined system $\Gamma_{\mathsf{DE}} = \begin{bmatrix} \pi_{\mathsf{DE}} \\ \Gamma_{\mathsf{Comm}} \end{bmatrix}$ satisfies $\mathcal{M}_{\mathsf{DE}}$.

Formally, the model $\mathcal{M}_{\mathsf{DE}}$ includes an *experiment system* $\Gamma_{\mathsf{Exp}}(\mathcal{M}_{\mathsf{DE}})$, which interacts with the composite system $\Gamma_{\mathsf{DE}} = \begin{bmatrix} \pi_{\mathsf{DE}} \\ \Gamma_{\mathsf{Comm}} \end{bmatrix}$ and with an *adversary* system $\Gamma_{\mathsf{A}}$ (see Figure 2 (a)). The experiment has direct interfaces with the 'lower layer' system (in (a), this is $\Gamma_{\mathsf{Comm}}$), which we use in [30] to allow the experiment $\Gamma_{\mathsf{Exp}}(\mathcal{M}_{\mathsf{DE}})$ to control delays and faults, and to eavesdrop on communication. The system $\Gamma_{\mathsf{DE}} = \begin{bmatrix} \pi_{\mathsf{DE}} \\ \Gamma_{\mathsf{Comm}} \end{bmatrix}$ satisfies $\mathcal{M}_{\mathsf{DE}}$, if the probability that $\mathsf{Exp}(\mathcal{M}_{\mathsf{DE}})$ outputs 1 is negligible, for any efficient (polynomial time) $\Gamma_{\mathsf{A}}$.

Similarly, in Figure 2 (b), we see protocol $\pi_{\mathsf{Orders}}$, which realizes model $\mathcal{M}_{\mathsf{Orders}}$ over model $\mathcal{M}_{\mathsf{DE}}$. Namely, when $\pi_{\mathsf{Orders}}$ is combined over any system $\Gamma_{\mathsf{DE}}$ that satisfies $\mathcal{M}_{\mathsf{DE}}$, the resulting combined system $\Gamma_{\mathsf{Orders}} = \begin{bmatrix} \pi_{\mathsf{Orders}} \\ \Gamma_{\mathsf{DE}} \end{bmatrix}$ satisfies $\mathcal{M}_{\mathsf{Orders}}$.

Finally, in Figure 2 (c), we see protocol $\pi_{\mathsf{DE||Orders}}$, which is the composition of protocol $\pi_{\mathsf{Orders}}$ 'on top of' protocol $\pi_{\mathsf{DE}}$, denoted $\pi_{\mathsf{DE||Orders}} \equiv \begin{bmatrix} \pi_{\mathsf{Orders}} \\ \pi_{\mathsf{DE}} \end{bmatrix}$. The layering lemma of shows that $\pi_{\mathsf{DE||Orders}}$ realizes model $\mathcal{M}_{\mathsf{Orders}}$ directly over model $\mathcal{M}_{\mathsf{Comm}}$. Namely, when $\pi_{\mathsf{DE||Orders}}$ is combined over any system $\Gamma_{\mathsf{Comm}}$ that satisfies $\mathcal{M}_{\mathsf{Comm}}$, the resulting combined system $\Gamma_{\mathsf{Orders}} = \begin{bmatrix} \pi_{\mathsf{DE}||\mathsf{Orders}} \\ \Gamma_{\mathsf{Comm}} \end{bmatrix}$ satisfies $\mathcal{M}_{\mathsf{Orders}}$.

## 1.2 The need for theoretical foundations for layering

Modular design is a basic engineering principle, applied in all large scale systems. In particular, layering is a common, simple form of modular design, which is widely used in distributed systems and networks, e.g. the Internet. However, existing proposals and standards of specifications of layers are only stated informally, often by (partial) specification for the *operation* of the protocols, rather than (full) specification for the *service model* that the higher layer can rely on. For example, the IP (Internet Protocol) layer is required to provide a vaguely-described 'best effort' service.

Layered compositions of protocols are also used without formal definition or proof. A possible explanation for the fact that layering was not yet based on formal foundations, in spite of its wide use, is the fact that similar compositions work as expected for many models, often trivially. For example, the composition of two polynomial time algorithms is trivially also a polynomial time algorithm. However, as Abadi and Lamport [1] argue, composition properties require proof, and may not hold for all (natural) models. For example, the composition of two polynomial time interactive Turing machines (ITM), or of (infinite) state machines with polynomial-time transition functions, may not be polynomial-time, in the natural setting where the outputs of each machine is considered part of the inputs of the other. Indeed, our proof of the layering lemma is simple; however, we found that some definitional choices could have subtle but critical impact on composability. This motivated the precise execution model, which we present in Section 2.

## 1.3 Related areas

**The game-playing paradigm** The layered games framework is based on the *game-playing paradigm*, instead of following the ideal functionality paradigm. The game playing paradigm is central to the theory of cryptography, see e.g. Goldreich et al. [23], Goldreich [21]. Game playing supports strong analytical tools, e.g. Bellare and Rogaway [9]. This may facilitate the use of (semi) automated proof-checking tools, see e.g. Halevi's proposal [26] and and Blanchet's CryptoVerif tool [10].

In the game-playing paradigm, one specifies an interactive game between a component and an adversary, where security is defined by the probability of the adversary winning in the game. With information-theoretic games the adversarial entity is allowed unbounded computational resources, while *concrete* and *probabilistic polynomial time* games assume certain limitations on adversarial resources, e.g. available time. Game-based specifications are widely used, and available for many cryptographic primitives such as digital signature and encryption schemes, pseudo-random functions, and many more, e.g., Goldwasser and Micali [24], Goldwasser, Micali, and Yao [25], Goldreich [21].

**Execution model** Our execution model is closely related to the execution models of I/O Automata of Lynch and Tuttle [36], especially the Probabilistic I/O Automata model of Canetti et al. [16], and to the Reactive Simulatability framework [4, 5, 38].

**Computational vs. symbolic security** The layered games framework follows the *computational* approach to cryptography, which treats protocols and cryptographic schemes as programs/machines, operating on arbitrary strings (bits). This is in contrast to the *symbolic* approach, where cryptographic operations are seen as functions on a space of symbolic (formal) expressions, and security properties are stated as symbolic expressions; see Dolev and Yao [20], Burrows, Abadi, and Needham [13]. Several works investigate compositions of cryptographic protocols with the symbolic approach, e.g. Datta et al. [18] and Backes at al. [6]. It may be beneficial, to extend the layered games framework to support symbolic/formal analysis, possibly building on recent results on the relationships between the two approaches, such as Abadi and Rogaway [2].

**Worst-case analysis** Our focus and main motivation is specification and analysis of security and cryptographic protocols. However, the layered games framework can be applicable when the goal is to analyze performance of protocols under hard to predict scenarios. In particular, the layered games framework allows generalization of the the adversarial queuing theory of Borodin, Kleinberg, Raghavan, Sudan, and Williamson [11].

## 2 Execution Model: Protocols, Systems, Configurations and Executions

In this section, we present an execution model for distributed systems, which we later use to define models and games, and to prove the layering lemmas. Such precise execution model is essential, to ensure that the analysis is not invalidated due to some 'technical', subtle issues; we handled several such issues when developing the framework, some of them found by careful readers (see acknowledgments). The execution model defines protocols, systems, configurations and executions. Some readers may prefer to skip this section at first reading, and read the rest of the paper based on their intuitive interpretation of these concepts.

## 2.1 Protocols

Our basic element of computation is a *protocol*. Protocols are state machines[2] that accept input on one of the *input interfaces*, and produce output on one or more *output interfaces*.

The protocol also includes a mapping $\mathsf{SeCo}$ of 'self-connections', from output interfaces to input interfaces (or to $\perp$, signaling an output interfaces which is not connected to an input interface of the same protocol). We use self-connections to merge two protocols into one (by transforming the connections between the two protocols into self-connections of the merged protocol).

The protocol includes a transition function $\delta$, which maps the input (interface and value) and current state, to a new state and to outputs on the different output interfaces. We use $\perp$ to denote a special value which is not a binary string ($\perp \not\in \{0,1\}^*$); a protocol outputs $\perp$ on some output interface to signal 'no output'.

The transition function $\delta$ can depend on two additional inputs: random bits and a security parameter. A *deterministic protocol* ignores the random bits; deterministic protocols can be useful, e.g. to analyze the provision and use of pseudo-random bits. The (unary) security parameter, allows to define computational properties of the protocol and of specifications, such as security against computationally-bounded adversary. Specifically, we use the security parameter to define a *polynomial* protocol.

**Definition 1 (Protocol).** *A protocol $\pi$ is a tuple $\langle S, I_{IN}, I_{OUT}, \delta, \mathsf{SeCo}\rangle$ where:*

1. *$S$ is a set of states, where $\perp \in S$ is the initial state,*
2. *$I_{IN}$ is a finite sequence of input interface identifiers,*
3. *$I_{OUT}$ is a finite set of output interface identifiers,*
4. *$\delta : IN \rightarrow OUT$ is a transition function, with:*
   - *Domain $IN = 1^* \times S \times I_{IN} \times \{0,1\}^* \times \{0,1\}^*$ (security parameter, current state, input interface, input value, random bits).*
   - *Range $OUT = S \times (\{0,1\}^* \cup \{\perp\})^{|I_{OUT}|}$. The outputs consist of a new state, denoted $\delta.s \in S$, and output values $\delta.ov[\iota_o] \in \{0,1\}^* \cup \{\perp\}$ for each interface $\iota_o \in I_{OUT}$.*
5. *$\mathsf{SeCo}$ is a mapping of 'self connections', from $I_{OUT}$ to $I_{IN} \cup \{\perp\}$*

*The protocol is* polynomial *if $\delta$ is polynomial-time computable, and if the length of the outputs is the same as the length of the inputs*[3]*, plus a polynomial in the security parameter, i.e. $\exists c \in \mathbb{N}$ s.t. $\forall (1^k, s, \iota, x; r) \in IN, \iota_o \in I_{OUT} : |\delta.ov[\iota_o](1^k, s, \iota, x; r)| \leq |x| + |k|^c$.*

[2] We use state machines, rather than e.g. ITM as in Universal Composability [15], since we found it simpler, and easier to ensure that an execution involving multiple protocols, some of which are adversarial, will have well-defined scheduling and distribution of events. Also, in many cases protocols may be represented by *finite* state machines, which may have advantages including possible use of automated verification tools.

[3] This restriction of the output length to be the same as input length, plus some 'overhead' which depends only on the security parameter, is a simple method to prevent

Notations:

**$\mathbf{\Pi}$, $\mathbf{\Pi}_{\mathsf{poly}}$:** we denote the set of all protocols by $\mathbf{\Pi}$, and the set of polynomial protocols by $\mathbf{\Pi}_{\mathsf{poly}}$.

**Dot notation:** The range of $\delta$ is a set of pairs $(s, ov[\iota_o])$, where $s \in S$ is the new state and $ov[\iota_o] \in \{0,1\}^* \cup \{\bot\}$ is the output on each output interface $\iota_o \in I_{OUT}$. To refer directly to the state or the outputs, we use dot notation as in $\delta.s(\cdot)$ and $\delta.ov[\iota_o](\cdot)$ respectively. We also use dot notation to refer to a specific part of a protocol, e.g. $\pi.I_{IN}$ is the set of input interface identifiers $I_{IN}$ of protocol $\pi$.

**Range, Domain:** We use Domain($\mathsf{SeCo}$) and Range($\mathsf{SeCo}$) to refer to range and domain, respectively, of $\mathsf{SeCo}$ (or other mapping), i.e. $\mathsf{SeCo} = \text{Domain}(\mathsf{SeCo}) \rightarrow \text{Range}(\mathsf{SeCo})$.

## 2.2 Configurations

We study executions of *configurations*, containing multiple protocols connected via their interfaces. A configuration is a *directed graph*, whose nodes $\mathsf{P}$ are identifiers of protocols, and whose edges are defined by mappings $p' = \mathsf{nP}(p, \iota)$ (for 'next protocol') and $\iota' = \mathsf{nI}(p, \iota)$ (for 'next interface'), mapping *output interface* $\iota \in \mathsf{oI}(p)$ of node $p$, to *input interface* $\iota' \in \mathsf{iI}(p')$ of node $p'$.

**Definition 2 (Configuration).** *A* configuration *is a tuple* $C = \langle \mathsf{P}, \mathsf{iI}, \mathsf{oI}, \mathsf{nP}, \mathsf{nI} \rangle$*, where:*

$\mathsf{P}$ *is a set of protocol instance identifiers,*

$\mathsf{iI}, \mathsf{oI}$ *map identifiers in* $\mathsf{P}$ *to input and output interfaces, respectively,*

$\mathsf{nP}$ *maps from instance identifier* $p \in \mathsf{P}$ *and an output interface* $\iota \in \mathsf{oI}(p)$*, to* $p' = \mathsf{nP}(p, \iota)$*, where either* $p' = \bot$ *or* $p' \in \mathsf{P}$ *(another instance),*

$\mathsf{nI}$ *maps from instance identifier* $p \in \mathsf{P}$ *and an output interface* $\iota \in \mathsf{oI}(p)$*, to input interface* $\iota'$*, s.t. if* $\mathsf{nP}(p, \iota) \in \mathsf{P}$ *then* $\iota' \in \mathsf{iI}(\mathsf{nP}(p, \iota))$*,*

Configurations as defined above, are quite general. In particular, we intentionally avoided assuming any specific communication or synchronization mechanisms. This allows use of the framework in diverse scenarios, e.g. with or without assumptions on synchronization, communication and failures.

exponential blow-up in input and output lengths, as outputs of one protocol become inputs to another protocol during execution. This restriction is reasonable in practice, and sufficient for our needs; for example, it allows a protocol to 'duplicate' input from one interface, to multiple output interfaces, but maintains a polynomial bound on the length of the inputs and outputs on each interface during the execution. More elaborate ways to prevent exponential blow-up were presented by Küsters [34] describing a general model for systems which satisfy certain acyclic conditions, Canetti [15] and Hofheinz, Müller-Quade, and Unruh [32] for UC, and Backes et al. [5] for reactive simulatability.

### 2.3 Schedules and Executions

An *execution* is defined by a sequence of transitions of a protocol $\pi$ running in one node $p \in \mathsf{P}$ inside a configuration $C = \langle \mathsf{P}, \mathsf{iI}, \mathsf{oI}, \mathsf{nP}, \mathsf{nI} \rangle$. To define the execution, we use a mapping $\pi = \mathit{\Pi}(p)$ from the protocol identifiers $\mathsf{P}$ to the protocols implementing each node.

An important design goal, is that the set of executions of a given configuration $C$, with a specific mapping to protocols $\mathit{\Pi}$, would be a well-defined random variable. This makes it easier to use an execution as a 'subroutine', to facilitate reduction-based reasoning and proofs. To simplify such reductions, we require that executions be a *deterministic* function of explicit random-tape inputs. Specifically, the $i^{\text{th}}$ event in the execution, denoted $\xi_i$, is defined by the (deterministic) transition function of the protocol $\mathit{\Pi}(p_i)$ invoked at this event (where $p_i$ is the identifier of that node). We allow the protocol to make random choices, but only using uniformly-selected random bits $R_i \in_R \{0,1\}^*$, provided as input to the transition function. Let $\mathcal{R} \equiv \{\{0,1\}^*\}_{i=1,2,...}$ be the sequence whose elements are the sets of all binary strings $\{0,1\}^*$; each execution is a deterministic function of the specific sequence $R \in \mathcal{R}$ used in that execution (i.e., $R = \{R_i\}_{i=1,2,...}$ s.t. $(\forall i) R_i \in \{0,1\}^*$).

Each protocol instance has its own state, and in each round it may decide to invoke interfaces of multiple other protocol instances; see for example the configurations in Figure 2. Recall that some of the output interfaces of a protocol may also be connected to input interfaces of the same protocol (self-connections).

At any point in the execution, there may be multiple 'pending' values sent from some output interface of one protocol $p \in \mathsf{P}$, to some input interface of another protocol $q \in \mathsf{P}$ (possibly $q = p$, for self-connection). To define an execution, we need to decide on the order (scheduling) in which we handle these 'pending' input values. To ensure well-defined executions, without any non-deterministic choice (except for the explicit use of the random input strings $R \in \mathcal{R}$), we use a *schedule* $\mathcal{S}$.

**Definition 3 (Schedule).** *A schedule $\mathcal{S}$ of configuration $C = \langle \mathsf{P}, \mathsf{iI}, \mathsf{oI}, \mathsf{nP}, \mathsf{nI} \rangle$ and protocol mapping $\mathit{\Pi}$, is a sequence of pairs $\mathcal{S} = \{< p_i, \iota_i >\}_{i \in \mathbb{N}}$ where $p_i \in \mathsf{P}$ and $\iota_i \in \mathit{\Pi}(p_i).I_{IN}$.*

The schedule is defined in advance and cannot depend on the execution (or on the random bits). Schedules can be completely adversarial, as in [16], or (partially) specified, as in Section 3.

A similar issue, where we tried to avoid non-determinism, involves how we handle multiple pending inputs, submitted on the same input interface. Our definition delivers inputs on an interface, in the order in which they were submitted. We do this by keeping a *FIFO queue* $Q[p, \iota]$, for protocol instance $p$ and input interface $\iota$, with regular semantics for the ENQUEUE, DEQUEUE, and IS_EMPTY operations. This choice is natural, although other choices may be possible.

**Definition 4 (Execution).** *Let $C = \langle \mathsf{P}, \mathsf{iI}, \mathsf{oI}, \mathsf{nP}, \mathsf{nI} \rangle$ be a configuration, and let $\mathit{\Pi} : \mathsf{P} \rightarrow \mathbf{\Pi}$ be a mapping of the protocol identifiers $\mathsf{P}$ to specific protocols. Let $\mathcal{S} = \{\langle p_i \in \mathsf{P}, \iota_i \in \mathsf{iI}(p_i) \rangle\}_{i \in \mathbb{N}}$ be a* schedule *of $C$ and $\mathit{\Gamma}$.*

*The* execution $X_k(C, \Pi, \mathcal{S}; R)$ *of security parameter* $k \in 1^*$*, configuration* $C$*, protocol mapping* $\Gamma$*, schedule* $\mathcal{S}$ *and sequence (of random bits)* $R = \{R_i\} \in \mathcal{R}$*, is the sequence of* execution events $\{\xi_i\} = \{\langle p_i, \iota_i, iv_i, ov_i[\cdot]\rangle\}$ *resulting from the execution process in Figure 3.*

```
FOR ALL $p \in \mathsf{P}$:
        $s[p] := \bot$;
$Q[p_1, \iota_1]$.ENQUEUE$(0)$;
FOR $i := 1$ TO $\infty$ DO:
        IF $Q[p_i, \iota_i]$.IS_EMPTY() THEN $iv_i = \bot,\ ov_i[\cdot] = \bot$;
        ELSE:
          1. $iv_i := Q[p_i, \iota_i]$.DEQUEUE();
          2. $\langle S, I_{IN}, I_{OUT}, \delta, \mathsf{SeCo}\rangle := \Pi(p_i)$;
          3. $\langle s[p_i], ov_i[\iota \in I_{OUT}]\rangle := \delta(k, s[p_i], \iota_i, iv_i; R_i)$;
          4. FOR ALL $\iota_o \in I_{OUT}$ S.T. $ov_i[\iota_o] \neq \bot$:
                  IF $\mathsf{nP}(p_i, \iota_o) \neq \bot$
                    THEN $Q[\mathsf{nP}(p_i, \iota_o), \mathsf{nI}(p_i, \iota_o)]$.ENQUEUE$(ov_i[\iota_o])$;
                    ELSE IF $\mathsf{SeCo}(\iota_o) \neq \bot$
                      THEN $Q[p_i, \mathsf{SeCo}(\iota_o)]$.ENQUEUE$(ov_i[\iota_o])$;
```

**Fig. 3.** Execution process defining $\{\langle p_i, \iota_i, iv_i, ov_i[\cdot]\rangle\} = X_k(C, \Pi, \mathcal{S}; R)$

*For given* $p \in \mathsf{P}, \iota \in \mathsf{oI}(p)$*, let:*

$$i_{k,p,\iota,l}(C, \Pi, \mathcal{S}; R) \equiv \max\left[0 \cup \{i \leq l | (p_i = p) \wedge ov_i[\iota] \neq \bot\}\right]$$

*and*

$$X_{k,l,l,p,\iota,l}(C, \Pi, \mathcal{S}; R) \equiv \begin{cases} ov_{i_{k,l,p,\iota,l}(C,\Pi,\mathcal{S};R)}[\iota] & \text{if } i_{k,l,p,\iota,l}(C, \Pi, \mathcal{S}; R) > 0 \\ \bot & \text{otherwise} \end{cases}$$

If all protocols in the range of $\Pi$ are *polynomial*, we say that $\Pi$ is *polynomial*. If $\Pi$ is polynomial, then $X_k(C, \Pi, \mathcal{S})[l]$ is sampleable in time polynomial in $k$ and $l$. This allows a polynomial protocol to run polynomial number of steps of an execution containing polynomial protocols, as part of its computational process (e.g. for reduction proofs). We restate this observation in the following proposition.

**Proposition 1 (Executions of polynomial protocols are efficiently sampleable).** *Let* $C = \langle \mathsf{P}, \mathsf{iI}, \mathsf{oI}, \mathsf{nP}, \mathsf{nI}\rangle$ *be a configuration and* $\Pi : \mathsf{P} \rightarrow \mathbf{\Pi}_{\mathsf{poly}}$ *be a mapping of the protocol identifiers* $\mathsf{P}$ *to specific polynomial protocols. Then* $X_k(C, \Pi, \mathcal{S})[l]$ *is sampleable in probabilistic polynomial time (as a function of* $k$ *and* $l$*).*

## 3 Layered Protocols, Systems, Models and Realizations

From this section, our discussion is focused, on *layered architectures* [4]. Layered architectures are based on *abstraction.* Namely, the designer of protocol $\pi_i$ for layer $i$, is oblivious to details of lower layers, and only cares that the *system* $\Gamma_{i-1}$, representing layer $i-1$ and all lower layers, will satisfy the layer $i-1$ *model*, denoted $\mathcal{M}_{i-1}$. The goal of the designer of protocol $\pi_i$, for layer $i$, is to realize model $\mathcal{M}_i$, over any system $\Gamma_{i-1}$ satisfying model $\mathcal{M}_{i-1}$.

In the first subsection below, we define layered protocols and systems. Next, in Section 3.2, we give a game-based definition of a *model*, with conditions on the outcomes of the game, defining when a (layered) system $\Gamma_{\mathsf{L}}$ is considered to satisfy model $\mathcal{M}$; we denote this by $\mathcal{M} \models \Gamma_{\mathsf{L}}$. Finally, in Section 3.3, we define the *realization* relation, denoted $\mathcal{M}_U \vdash \begin{bmatrix} \pi_U \\ \mathcal{M}_L \end{bmatrix}$, indicating that protocol $\pi_U$ realizes model $\mathcal{M}_U$, when running over any system satisfying lower layer model $\mathcal{M}_L$, and present the layering lemma.

### 3.1 Layered protocols and systems

For simplicity, in this work we focus on analysis of *layered* protocols and systems, as common in practice; see Figure 4. A layered protocol $\pi$ has only two external input interfaces, $\{\mathsf{I_H}, \mathsf{I_L}\} = \pi.I_{IN}/\text{Domain}(\pi.\mathsf{SeCo})$, for higher and lower layer, respectively. Similarly, $\pi$ has only two external output interfaces, $\{\mathsf{O_H}, \mathsf{O_L}\} = \pi.I_{OUT}/\text{Range}(\pi.\mathsf{SeCo})$, for higher and lower layer, respectively. Layered protocols are a simplification of practical protocols, which usually have several interfaces to and from the higher layer, and several interfaces to and from the lower layer; these can usually be 'merged' to the interfaces above.

We investigate executions of $n$ copies of the analyzed layered protocol $\pi$, each running in one of $n$ 'processors'. The $n$ copies of $\pi$ interact with some network-wide protocols, for the adversary, experiments, and 'environment' (e.g., lower layers). We found it convenient to refer to such network-wide protocols as *systems*; this allows us to refer to layered protocols simply as 'protocols'. The parameter $n$ can impact the number of interfaces of the systems, or their transition functions.

As a convention, we use $\pi$ for (layered) protocols, and $\Gamma$ for systems. Specifically, we use $\Gamma_{\mathsf{A}}$ for the adversary system, $\Gamma_{\mathsf{Exp}}$ for the experiment system, and $\Gamma_i$ for the *layer i system.* The layer $i$ system is the combination of the environment and all protocols, from layer $i$ and below.

The layer $i$ system $\Gamma_i(n)$, for $n$ processors, has input interfaces $\{\mathsf{I}_j\}_{j=1}^n$, and output interfaces $\{\mathsf{O}_j\}_{j=1}^n$, one pair $(\mathsf{I}_j, \mathsf{O}_j)$ for each processor $j \in \{1, \ldots, n\}$. In addition, the layer has input interface $\mathsf{I_E}$ and output interface $\mathsf{O_E}$, for inputs to

[4] We believe that it is possible to generalize our concepts and results to modular, but not layered, architectures. However, this will cause (at least technical) complexities, that may make the resulting definitions less easy to understand and use. We therefore leave this for future investigation.

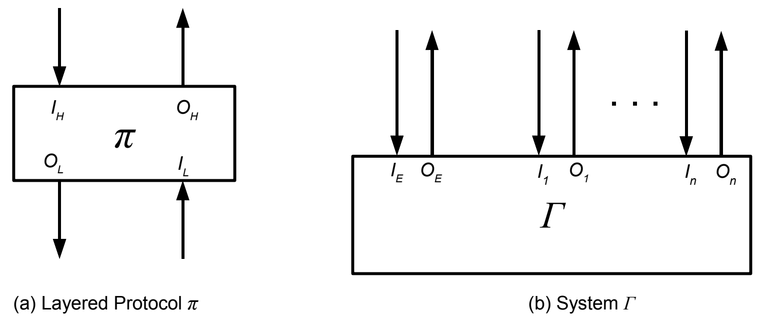


**Fig. 4.** Layered protocol (a) and system (b).

(respectively, outputs from) the 'environment', e.g. to define delays and to report on 'low level' events, observable via the environment (e.g. packet transmission). In addition, the system $\Gamma_i(n)$ can contain self-connections.

Usually, we investigate families of configurations $C = \{C(n)\}$. This allows, in particular, to consider $n$ protocols, and systems $\Gamma(n)$ whose set of interfaces is a function of $n$ (e.g. to interact with $n$ protocols). For convenience, we use the term 'configuration' also when referring to a family of configurations $C = \{C(n)\}$.

Given two (layered) protocols $\pi_2, \pi_1$, we define the composite protocol $\pi_{1||2} \equiv \begin{bmatrix} \pi_2 \\ \pi_1 \end{bmatrix}$ by 'merging' the two protocols, as shown in Figure 5 (a). Similarly, given protocol $\pi_1$ and system $\Gamma_0$, we define the composite system $\Gamma_1 \equiv \begin{bmatrix} \pi_1 \\ \Gamma_0 \end{bmatrix}$, as in Figure 5 (b). We omit the (trivial) details of both compositions.

The following proposition states a trivial, yet important, observation, namely that composition is associative.

**Proposition 2 (Layering is associative).** *Let $\pi_1, \pi_2, \pi_3$ be three protocols. The layering of $\pi_1, \pi_2$ and $\pi_3$ results in the same protocol $\pi_{1||2||3}$, regardless of the order, i.e.*

$$\pi_{1||2||3} = \pi_{1||(2||3)} = \begin{bmatrix} \pi_{2||3} \\ \pi_1 \end{bmatrix} = \begin{bmatrix} \pi_3 \\ \pi_{1||2} \end{bmatrix} = \pi_{(1||2)||3}$$

*Similarly, let $\Gamma_0$ be a system; the layering of $\pi_1$ and $\pi_2$ on top of $\Gamma_0$ results in the same system $\Gamma_2$, regardless of the order, i.e.:*

$$\Gamma_2 = \begin{bmatrix} \pi_2 \\ \begin{bmatrix} \pi_1 \\ \Gamma_0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} \pi_2 \\ \pi_1 \end{bmatrix} \\ \Gamma_0 \end{bmatrix}$$

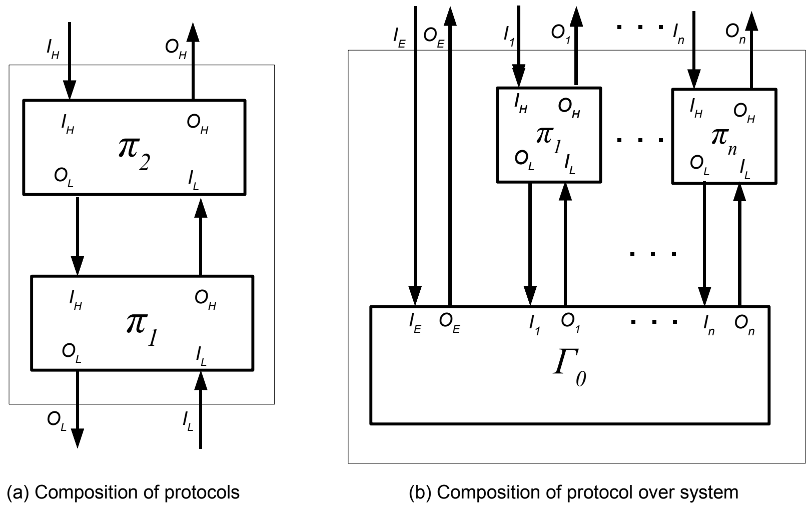


**Fig. 5.** Composite protocol $\pi_{1||2}$ (a) and system $\Gamma_1$ (b) .

The next proposition states that composition preserves efficiency, in the sense of polynomial-time complexity. A system $\Gamma$ is *polynomial* if for every $n$, $\Gamma(n)$ is a polynomial protocol; if $\Gamma(n)$ is efficiently computable, we say that the system $\Gamma$ is *uniformly polynomial*, otherwise, it is *non-uniformly polynomial*.

**Proposition 3 (Layering preserves efficiency).** *The layering of polynomial protocols, results in a polynomial protocol. The layering of a polynomial protocol over a (uniformly) polynomial system, results in an (uniformly) polynomial system.*

## 3.2 Layered Models

We use the term (layered) *model* for the specifications of a (layered) system $\Gamma_{\mathsf{L}}$. We define a (layered) model $\mathcal{M}$ by a simple zero-sum (win-lose) game between an *adversary system* $\Gamma_{\mathsf{A}}$ and a system $\Gamma_{\mathsf{L}}$. These systems interact only via a third system, the *experiment* $\Gamma_{\mathsf{Exp}}(\mathcal{M})$, as shown in Figure 6. When the relevant model $\mathcal{M}$ is clear, we sometimes omit it and write simply $\Gamma_{\mathsf{Exp}}$.

The experiment system defines the 'rules of the game', and outputs 1, on a designated output interface outcome, if the adversary 'won'; if this happens (with probability over some *winning threshold* $\alpha(\mathcal{M}) \in [0, 1]$), then the system $\Gamma_{\mathsf{L}}$ does not satisfy model $\mathcal{M}$. Without loss of generality, the experiment outputs only 1 on outcome, and does so at most once in every execution. Typically, $\alpha(\mathcal{M}) = 0$ (e.g. for forgery games) or $\alpha(\mathcal{M}) = \frac{1}{2}$ (e.g. for indistinguishability games).

In Proposition 5, we show that, for computational security, we can always use $\alpha(\mathcal{M}) = \frac{1}{2}$, without loss of generality.

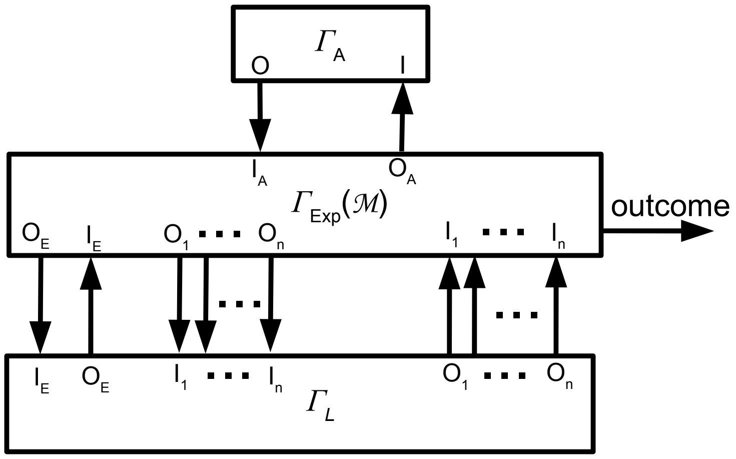


**Fig. 6.** Layer Model Configuration.

Figure 6 shows the configuration $C_{LM}(n) = \langle \mathsf{P}_n, \mathsf{iI}_n, \mathsf{oI}_n, \mathsf{nP}_n, \mathsf{nI}_n \rangle$ of the layer modeling game, for $n$ processors. The configuration contains only three protocol identifiers, $P_n = \{\mathsf{A}, \mathsf{Exp}, \mathsf{L}\}$, which we map to the three systems $\Gamma_{\mathsf{A}}$ (adversary), $\Gamma_{\mathsf{Exp}}(\mathcal{M})$ (experiment) and $\Gamma_{\mathsf{L}}$ (layered system), respectively.

For layered models, we only allow simple, 'round robin' schedules $\mathcal{S}_{RR}(n) = (\mathcal{S}_{\mathsf{A}}(n)||\mathcal{S}_{\mathsf{Exp}}(n)||\mathcal{S}_{\mathsf{L}}(n))^*$, where, for $\phi \in \{\mathsf{A}, \mathsf{Exp}, \mathsf{L}\}$, the 'sub-schedule' $\mathcal{S}_\phi$ is defined by the sequence of input interface of $\phi$, i.e. $\mathcal{S}_\phi(n) \equiv \phi \times \Gamma_\phi(n).I_{IN}$.

**Definition 5 (Layer model satisfaction game).** *A* model $\mathcal{M}$ *is a tuple* $\mathcal{M} = (\Gamma_{\mathsf{Exp}}(\mathcal{M}), \alpha(\mathcal{M}))$*, where* $\alpha(\mathcal{M}) \in [0,1]$*,* $\Gamma_{\mathsf{Exp}}(\mathcal{M})$ *is a polynomial system that output on interface* outcome *at most once (and only the value* 1*).*

*We say that* system $\Gamma_{\mathsf{L}} \in \mathbf{\Pi}_{\mathsf{poly}}$ computationally satisfies model $\mathcal{M}$*, and write* $\mathcal{M} \models_{\mathsf{poly}} \Gamma_{\mathsf{L}}$*, if for every* $n$*,* $\Gamma_{\mathsf{A}} \in \mathbf{\Pi}_{\mathsf{poly}}$*, polynomial* $l$*, constant* $c$ *and large enough* $k$*, holds:*

$$\Pr_{R \in \mathcal{R}} (X_{k,\mathsf{Exp},\mathsf{outcome},l}(C_{LM}(n), \Pi_n, \mathcal{S}_{RR}(n)) = 1) \leq \alpha(\mathcal{M}) + k^c$$

*Where* $\Pi_n(\mathsf{A}) = \Gamma_{\mathsf{A}}(n), \Pi_n(\mathsf{Exp}) = \Gamma_{\mathsf{Exp}}(n), \Pi_n(\mathsf{L}) = \Gamma_{\mathsf{L}}(n)$ *and* $\mathcal{S}_{RR}(n) = (\mathcal{S}_{\mathsf{A}}(n)||\mathcal{S}_{\mathsf{Exp}}(n)||\mathcal{S}_{\mathsf{L}}(n))^*$ *as above.*

*System* $\Gamma_{\mathsf{L}}$ non-uniformly satisfies *model* $\mathcal{M}$*, which we denote by* $\mathcal{M} \models_{\mathsf{nu}} \Gamma_{\mathsf{L}}$*, if the above holds even for non-uniform* $\Gamma_{\mathsf{A}}$*. System* $\Gamma_{\mathsf{L}}$ perfectly satisfies *model* $\mathcal{M}$*, which we denote by* $\mathcal{M} \models_{\mathsf{perf}} \Gamma_{\mathsf{L}}$*, if the above holds when protocols are not required to be polynomial, and without the* $k^c$ *term.*

Shorthand: we may write $\mathcal{M} \models \Gamma_{\mathsf{L}}$, when it is obvious that we refer to $\mathcal{M} \models_{\mathsf{poly}} \Gamma_{\mathsf{L}}$. We also may omit the word 'computational', i.e. say that $\Gamma_{\mathsf{L}}$ *satisfies* layer model $\mathcal{M}$.

We observe the trivial relation among the three notions of satisfaction.

**Proposition 4.** *For any model $\mathcal{M}$ and system $\Gamma_{\mathsf{L}}$ holds:*

$$(\mathcal{M} \models_{\mathsf{perf}} \Gamma_{\mathsf{L}}) \Rightarrow (\mathcal{M} \models_{\mathsf{poly}} \Gamma_{\mathsf{L}}) \Rightarrow (\mathcal{M} \models_{\mathsf{nu}} \Gamma_{\mathsf{L}})$$

Model $\mathcal{M}_{\mathsf{L}}$ is (polynomially / non-uniformly / perfectly) *weaker than* model $\mathcal{M}_{\mathsf{R}}$, if every system $\Gamma_{\mathsf{L}}$ that satisfies $\mathcal{M}_{\mathsf{R}}$, also satisfies $\mathcal{M}_{\mathsf{L}}$. We denote this by $\mathcal{M}_{\mathsf{L}} \prec_{\phi} \mathcal{M}_{\mathsf{R}}$, where $\phi \in \{\mathsf{poly}, \mathsf{nu}, \mathsf{perf}\}$ respectively. Models $\mathcal{M}_{\mathsf{L}}$ and $\mathcal{M}_{\mathsf{R}}$ are *$\phi-$equivalent*, if $\mathcal{M}_{\mathsf{L}} \prec_{\phi} \mathcal{M}_{\mathsf{R}}$ and $\mathcal{M}_{\mathsf{R}} \prec_{\phi} \mathcal{M}_{\mathsf{L}}$; we denote this by $\mathcal{M}_{\mathsf{L}} \equiv_{\phi} \mathcal{M}_{\mathsf{R}}$. Models $\mathcal{M}_{\mathsf{L}}$ and $\mathcal{M}_{\mathsf{R}}$ are *fully equivalent* if they are polynomially, non-uniformly and perfectly equivalent; we denote this by $\mathcal{M}_{\mathsf{L}} \equiv \mathcal{M}_{\mathsf{R}}$. Clearly, the $\prec$ relations define a partial order, while the $\equiv$ relations define equivalence classes.

We next show that for computational and non-uniform satisfaction, we can consider, without loss of generality, only models $\mathcal{M}$ with winning threshold $\alpha(\mathcal{M}) = \frac{1}{2}$.

**Proposition 5 (WLOG $\alpha = \frac{1}{2}$).** *Every model $\mathcal{M}$, has a polynomially and non-uniformly equivalent model $\mathcal{M}'$, s.t. $\alpha(\mathcal{M}') = \frac{1}{2}$.*

*Proof:* We first show that there exists such $\mathcal{M}'$ s.t. $\alpha(\mathcal{M}') \leq \frac{1}{2}$. This is trivial; let $\alpha(\mathcal{M}') = \alpha(\mathcal{M}) \cdot \frac{1}{2}$, and $\Gamma_{\mathsf{Exp}}(\mathcal{M}') = \Gamma_{\mathsf{Exp}}(\mathcal{M})$, except that if $\Gamma_{\mathsf{Exp}}(\mathcal{M})$ outputs 1 on $\mathsf{outcome}$, then $\Gamma_{\mathsf{Exp}}(\mathcal{M}')$ flips a fair coin $b \in_R \{0,1\}$, and returns 1 only if $b = 1$.

Hence, we assume, without loss of generality, that $\alpha(\mathcal{M}) \leq \frac{1}{2}$, and define $\mathcal{M}'$ s.t. $\alpha(\mathcal{M}') = \frac{1}{2}$. Again, the experiment system $\Gamma_{\mathsf{Exp}}(\mathcal{M}')$ is a modification of $\Gamma_{\mathsf{Exp}}(\mathcal{M})$, as follows.

As its first step in the execution, $\Gamma_{\mathsf{Exp}}(\mathcal{M}')$ flips a *biased* coin $b \in \{0,1\}$, with $\Pr(b = 1) = \frac{\frac{1}{2} - \alpha(\mathcal{M})}{1 - \alpha(\mathcal{M})}$. If $b = 1$ then $\Gamma_{\mathsf{Exp}}(\mathcal{M}')$ immediately outputs 1 on $\mathsf{outcome}$. Otherwise, if $b = 0$, then $\Gamma_{\mathsf{Exp}}(\mathcal{M}')$ proceeds exactly like $\Gamma_{\mathsf{Exp}}(\mathcal{M})$.

For fixed $l, k, R, n, \mathcal{S}, \Gamma_{\mathsf{A}}$ and $\Gamma_{\mathsf{L}}$, let:

$$W = \begin{cases} true & \text{if } X_{k,\mathsf{Exp},\mathsf{outcome},l}(C_{LM}, \Pi, \mathcal{S}) = 1 \\ false & \text{else} \end{cases}$$

And:

$$W' = \begin{cases} true & \text{if } X_{k,\mathsf{Exp},\mathsf{outcome},l}(C_{LM}, \Pi', \mathcal{S}) = 1 \\ false & \text{else} \end{cases}$$

Where $\Pi'(\mathsf{A}) = \Pi(\mathsf{A}) = \Gamma_{\mathsf{A}}$, $\Pi'(\mathsf{L}) = \Pi(\mathsf{L}) = \Gamma_{\mathsf{L}}$, $\Pi(\mathsf{Exp}) = \Gamma_{\mathsf{Exp}}(\mathcal{M})$ and $\Pi'(\mathsf{Exp}) = \Gamma_{\mathsf{Exp}}(\mathcal{M}')$.

It follows that:

$$\begin{aligned}
\Pr(W') &= \Pr\left((b=1) \vee ((b=0) \wedge W)\right) \\
&= \frac{\frac{1}{2} - \alpha(\mathcal{M})}{1 - \alpha(\mathcal{M})} + \left(1 - \frac{\frac{1}{2} - \alpha(\mathcal{M})}{1 - \alpha(\mathcal{M})}\right) \Pr(W) \\
&= \frac{\frac{1}{2} - \alpha(\mathcal{M}) + (1 - \alpha(\mathcal{M}) \Pr(X^{\mathcal{M}}) - \left(\frac{1}{2} - \alpha(\mathcal{M})\right) \Pr(W)}{1 - \alpha(\mathcal{M})} \\
&= \frac{\frac{1}{2} - \alpha(\mathcal{M}) + \frac{1}{2} \Pr(W)}{1 - \alpha(\mathcal{M})} \\
&= \frac{1}{2} + \frac{\Pr(W) - \alpha(\mathcal{M})}{1 - \alpha(\mathcal{M})}
\end{aligned}$$

It follows that for every layered system $\Gamma_{\mathsf{L}}$, holds:

$$\mathcal{M} \models \Gamma_{\mathsf{L}} \Leftrightarrow \mathcal{M}' \models \Gamma_{\mathsf{L}}$$

Which shows that $\mathcal{M} \equiv_{\mathsf{poly}} \mathcal{M}'$ and $\mathcal{M} \equiv_{\mathsf{nu}} \mathcal{M}'$. □

## 3.3 Model Realization and the Layering Lemma

The model satisfaction game defines specifications for systems, i.e. defines when a system satisfies a model. However, we normally design only the protocols for a specific layer, assuming they operate over some 'underlying' system, representing the combination of lower layers and the underlying environment. We next define such specification for a protocol.

**Definition 6.** *Let $\mathcal{M}_1, \mathcal{M}_0$ be two models, and let $\pi$ be a protocol. We say that* $\pi$ computationally realizes $\mathcal{M}_1$ over $\mathcal{M}_0$, *and denote this by* $\mathcal{M}_1 \vdash \begin{bmatrix} \pi \\ \mathcal{M}_0 \end{bmatrix}$, *if* $\mathcal{M}_1 \models \begin{bmatrix} \pi \\ \Gamma_0 \end{bmatrix}$, *for every $\Gamma_0$ s.t. $\mathcal{M}_0 \models \Gamma_0$.*

*Similarly,* $\pi$ non-uniformly realizes $\mathcal{M}_1$ over $\mathcal{M}_0$, *which we denote by* $\mathcal{M}_1 \vdash_{\mathsf{nu}} \begin{bmatrix} \pi \\ \mathcal{M}_0 \end{bmatrix}$, *if* $\mathcal{M}_1 \models_{\mathsf{nu}} \begin{bmatrix} \pi \\ \Gamma_0 \end{bmatrix}$, *for every $\Gamma_0$ s.t. $\mathcal{M}_0 \models_{\mathsf{nu}} \Gamma_0$; and* $\pi$ perfectly realizes $\mathcal{M}_1$ over $\mathcal{M}_0$, *which we denote by* $\mathcal{M}_1 \vdash_{\mathsf{perf}} \begin{bmatrix} \pi \\ \mathcal{M}_0 \end{bmatrix}$, *if* $\mathcal{M}_1 \models_{\mathsf{perf}} \begin{bmatrix} \pi \\ \Gamma_0 \end{bmatrix}$, *for every $\Gamma_0$ s.t. $\mathcal{M}_0 \models_{\mathsf{perf}} \Gamma_0$.*

We now present the layering lemma, allowing compositions of protocols of multiple layers. The layering lemma shows that we can prove realization of each layer separately, and the composition of the realizations will be a realization of the highest layer over the lowest layer. This provides firm foundations to the accepted methodology of designing, implementing, analyzing and testing of each layer independently, yet relying on their composition to ensure expected properties. We state the lemma for only three layers - generalization for an arbitrary stack is immediate.

**Lemma 1 (The Layering Lemma).** *Let $\mathcal{M}_2, \mathcal{M}_1, \mathcal{M}_0$ be three polynomial models, and $\pi_1, \pi_2$ be two polynomial protocols, such that for $i = 1, 2$, protocol $\pi_i$ computationally (non-uniformly, perfectly) realizes $\mathcal{M}_i$ over $\mathcal{M}_{i-1}$. Then $\pi_{1||2} = \begin{bmatrix} \pi_2 \\ \pi_1 \end{bmatrix}$ computationally (respectively, non-uniformly, perfectly) realizes $\mathcal{M}_2$ over $\mathcal{M}_0$. Namely:*

$$\left((i=1,2)\mathcal{M}_i \vdash_{\mathsf{poly}} \begin{bmatrix} \pi_i \\ \mathcal{M}_{i-1} \end{bmatrix}\right) \Rightarrow \mathcal{M}_2 \vdash_{\mathsf{poly}} \begin{bmatrix} \pi_{1||2} \\ \mathcal{M}_0 \end{bmatrix} \tag{1}$$

$$\left((i=1,2)\mathcal{M}_i \vdash_{\mathsf{nu}} \begin{bmatrix} \pi_i \\ \mathcal{M}_{i-1} \end{bmatrix}\right) \Rightarrow \mathcal{M}_2 \vdash_{\mathsf{nu}} \begin{bmatrix} \pi_{1||2} \\ \mathcal{M}_0 \end{bmatrix} \tag{2}$$

$$\left((i=1,2)\mathcal{M}_i \vdash_{\mathsf{perf}} \begin{bmatrix} \pi_i \\ \mathcal{M}_{i-1} \end{bmatrix}\right) \Rightarrow \mathcal{M}_2 \vdash_{\mathsf{perf}} \begin{bmatrix} \pi_{1||2} \\ \mathcal{M}_0 \end{bmatrix} \tag{3}$$

*Furthermore, let $\Gamma_0$ be a polynomial system that computationally ( non-uniformly, perfectly) satisfies $\mathcal{M}_0$. Then $\Gamma_2 = \begin{bmatrix} \pi_{1||2} \\ \Gamma_0 \end{bmatrix}$ computationally (respectively, non-uniformly, perfectly) satisfies $\mathcal{M}_2$.*

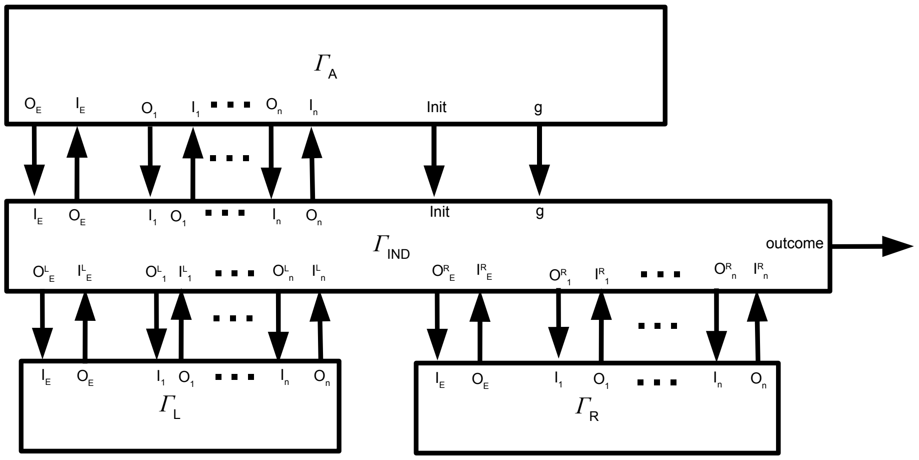
*Proof:* We prove the computational claims; the non-uniform and perfect variants follow exactly in the same way. Let $\Gamma_0$ be a system that computationally satisfies model $\mathcal{M}_0$. Since $\mathcal{M}_1 \vdash_{\mathsf{poly}} \begin{bmatrix} \pi_1 \\ \mathcal{M}_0 \end{bmatrix}$, by definition 6, the composite protocol $\Gamma_1 = \begin{bmatrix} \pi_1 \\ \Gamma_0 \end{bmatrix}$ satisfies $\mathcal{M}_1$, namely $\mathcal{M}_1 \models \Gamma_1$.

Protocol layering is associative (Proposition 2), hence $\Gamma_2 = \begin{bmatrix} \pi_{1||2} \\ \Gamma_0 \end{bmatrix} = \begin{bmatrix} \pi_2 \\ \Gamma_1 \end{bmatrix}$. Since $\mathcal{M}_2 \vdash_{\mathsf{poly}} \begin{bmatrix} \pi_2 \\ \mathcal{M}_1 \end{bmatrix}$ and $\mathcal{M}_1 \models \Gamma_1$, it follows from definition 6 that $\Gamma_2 = \begin{bmatrix} \pi_2 \\ \Gamma_1 \end{bmatrix}$ satisfies $\mathcal{M}_2$, namely $\mathcal{M}_2 \models \Gamma_2$, proving the second part of the lemma. The first part also follows, from definition 6, since this holds for every $\Gamma_0$ that computationally satisfies model $\mathcal{M}_0$. □

## 4 Indistinguishable Systems

Indistinguishability games are used in many cryptographic definitions, e.g. pseudo-random functions Goldreich, Goldwasser, and Micali [22]. We next define such game for systems. Intuitively, two systems are (polynomially) indistinguishable, if no (polynomial time) adversary can distinguish between them. We use the 'left-or-right' style of Bellare et al. [7].

The configuration $C_{LR}(n) = \langle \mathsf{P}_{LR}, \mathsf{iI}_n, \mathsf{oI}_n, \mathsf{nP}_n, \mathsf{nI}_n \rangle$ of the system indistinguishability games is illustrated in Figure 7, for some fixed number of instances $n$. The main difference between $C_{LR}$ and the configuration $C_{LM}$ of the model



Γ_A
O_E I_E O_1 I_1 O_n I_n Init g
I_E O_E I_1 O_1 I_n O_n Init g
Γ_IND
outcome
O^L_E I^L_E O^L_1 I^L_1 O^L_n I^L_n O^R_E I^R_E O^R_1 I^R_1 O^R_n I^R_n
I_E O_E I_1 O_1 I_n O_n
Γ_L
I_E O_E I_1 O_1 I_n O_n
Γ_R

**Fig. 7.** The Systems Indistinguishability game. System $\Gamma_L$ is indistinguishable from system $\Gamma_R$, if no (polytime) adversary can distinguish between interacting with $\Gamma_L$ and interacting with $\Gamma_R$ (with significant probability).

game, is the addition of the node $\mathsf{R}$, mapped to the system $\Gamma_{\mathsf{R}}$; we also use the name $\mathsf{IND}$ for the experiment here, i.e. $\mathsf{P}_{LR} = \{\mathsf{A}, \mathsf{IND}, \mathsf{L}, \mathsf{R}\}$.

In the layer realization indistinguishability game, we use the specific experiment system $\Gamma_{\mathsf{IND}}$ below, i.e. $\Pi_n(\mathsf{IND}) = \Gamma_{\mathsf{IND}}(n)$. Upon initialization, $\Gamma_{\mathsf{IND}}$ flips a fair coin $b \in_R \{L, R\}$. The game ends when $\Gamma_{\mathsf{IND}}$ receives a guess $b'$ of either $L$ or $R$ from the adversary $\mathsf{A}$, on the $\mathsf{g}$ (guess) input interface. Upon receiving the guess $b'$, $\Gamma_{\mathsf{IND}}$ outputs on its $\mathsf{outcome}$ output interface 1 if $b = b'$ (otherwise, it never produces output on $\mathsf{outcome}$). Details follow.

**Definition 7 (System indistinguishability experiment).** *Let* $\Gamma_{\mathsf{IND}}(n) = \langle S, I_{IN}(n), I_{OUT}(n), \delta_n\rangle$ *be the following system:*

$S = \{\perp, \mathsf{testing}, \mathsf{done}\}$
$I_{IN}(n) = \{\mathsf{Init}, \mathsf{g}, \mathsf{I_E}\} \cup \{\mathsf{I}_i, \mathsf{I^L}_i, \mathsf{I^R}_i\}_{i=1,\ldots,n}$
$I_{OUT}(n) = \{\mathsf{outcome}\} \cup \{\mathsf{O}_i, \mathsf{O^L}_i, \mathsf{O^R}_i\}_{i=1,\ldots,n}$
$\delta_n$*:*

1. *In initialization state* $\perp$*, upon any input, select randomly* $b \in_R \{\mathsf{L}, \mathsf{R}\}$*, and move to* testing *state.*
2. *In* testing *state, pass all input events on interfaces* $\mathsf{I}_i$*, for* $i \in \{\mathsf{E}, 1, \ldots, n\}$*, to corresponding output event on output interfaces* $\mathsf{O^L}_i$ *(if* $b = \mathsf{L}$*), or* $\mathsf{O^R}_i$ *(if* $b = \mathsf{R}$*). Similarly, pass all input events on interfaces* $\mathsf{I^L}_i$ *(if* $b = \mathsf{L}$*) or* $\mathsf{I^R}_i$ *(if* $b = \mathsf{R}$*), to corresponding output events on interface* $\mathsf{O}_i$*.*
3. *When, in* testing *state, the guess input interface* $\mathsf{g}$ *is invoked with input (guess)* $b' \in \{\mathsf{L}, \mathsf{R}\}$*, output on* outcome *the value* 1 *if* $b = b'$*. Move to the* done *state (and ignore all further inputs).*

The system indistinguishability experiment is the core of the system indistinguishability game, which we define next. The game tests whether an adversarial system $\varGamma_{\mathsf{A}}$ is able to distinguish between two systems, $\varGamma_{\mathsf{L}}$ and $\varGamma_{\mathsf{R}}$.

**Definition 8 (System indistinguishability game).** *Let* $\mathsf{P}_{LR} = \{\mathsf{A}, \mathsf{IND}, \mathsf{L}, \mathsf{R}\}$ *and* $C_{LR}(n) = \langle \mathsf{P}_{LR}, \mathsf{iI}_n, \mathsf{oI}_n, \mathsf{nP}_n, \mathsf{nI}_n \rangle$*, as above.*

*System* $\varGamma_{\mathsf{R}}$ computationally emulates *system* $\varGamma_{\mathsf{L}}$*, if for every adversary system* $\varGamma_{\mathsf{A}}$*, integer* $n$*, constant* $c$ *and polynomial* $l$*, for sufficiently large* $k$ *holds:*

$$\Pr\left(X_{k,\mathsf{IND},\mathsf{outcome},l}(C_{LR}(n), \varPi_n, \mathcal{S}(n)) = 1\right) \le \frac{1}{2} + k^c$$

*Where* $\varPi_n(\mathsf{A}) = \varGamma_{\mathsf{A}}(n), \varPi_n(\mathsf{IND}) = \varGamma_{\mathsf{IND}}(n), \varPi_n(\mathsf{L}) = \varGamma_{\mathsf{L}}(n), \varPi_n(\mathsf{R}) = \varGamma_{\mathsf{R}}(n)$ *and* $\mathcal{S}(n) = (\mathcal{S}_{\mathsf{A}}(n) || \mathcal{S}_{\mathsf{IND}}(n) || \mathcal{S}_{\mathsf{L}}(n) || \mathcal{S}_{\mathsf{R}}(n) || \mathcal{S}_{\mathsf{IND}}(n))^*$*, with* $\mathcal{S}_\phi(n) = \phi \times \varGamma_\phi(n).I_{IN}$*, for* $\phi \in \{\mathsf{A}, \mathsf{IND}, \mathsf{L}, \mathsf{R}\}$*.*

*System* $\varGamma_{\mathsf{R}}$ non-uniformly emulates *system* $\varGamma_{\mathsf{L}}$ *if the above holds even for non-uniform* $\varGamma_{\mathsf{A}}$*. System* $\varGamma_{\mathsf{L}}$ perfectly emulates *system* $\varGamma_{\mathsf{L}}$ *if the above holds even for non-polynomial* $\varGamma_{\mathsf{A}}$*, and without the* $k^c$ *term.*

*We say that system* $\varGamma_{\mathsf{R}}$ non-uniformly emulates *system* $\varGamma_{\mathsf{L}}$*, if the above holds when the adversary system* $\varGamma_{\mathsf{A}}$ *is non-uniformly polynomial. We say that* $\varGamma_{\mathsf{R}}$ perfectly emulates *system* $\varGamma_{\mathsf{L}}$*, if the above holds when the adversary system* $\varGamma_{\mathsf{A}}$ *is not limited to polynomial time, and without the* $k^c$ *term. We denote the emulation relation by* $\varGamma_{\mathsf{L}} \prec_{\mathsf{poly}} \varGamma_{\mathsf{R}}$ *(respectively,* $\varGamma_{\mathsf{L}} \prec_{\mathsf{nu}} \varGamma_{\mathsf{R}}$ *or* $\varGamma_{\mathsf{L}} \prec_{\mathsf{perf}} \varGamma_{\mathsf{R}}$*).*

*We say that system* $\varGamma_{\mathsf{R}}$ *is* computationally (non-uniformly, perfectly) indistinguishable *from system* $\varGamma_{\mathsf{L}}$*, if each of the two systems computationally (respectively, non-uniformly or perfectly) emulates the other. We denote this by* $\varGamma_{\mathsf{L}} \equiv_{\mathsf{poly}} \varGamma_{\mathsf{R}}$ *(respectively,* $\varGamma_{\mathsf{L}} \equiv_{\mathsf{nu}} \varGamma_{\mathsf{R}}$ *or* $\varGamma_{\mathsf{L}} \equiv_{\mathsf{perf}} \varGamma_{\mathsf{R}}$*).*

We observe the trivial relation among the three notions of emulation and indistinguishability.

**Proposition 6.** *For any two systems* $\varGamma_{\mathsf{L}}, \varGamma_{\mathsf{R}}$ *holds:*

$$(\varGamma_{\mathsf{L}} \prec_{\mathsf{perf}} \varGamma_{\mathsf{R}}) \Rightarrow (\varGamma_{\mathsf{L}} \prec_{\mathsf{nu}} \varGamma_{\mathsf{R}}) \Rightarrow (\varGamma_{\mathsf{L}} \prec_{\mathsf{poly}} \varGamma_{\mathsf{R}})$$

$$(\varGamma_{\mathsf{L}} \equiv_{\mathsf{perf}} \varGamma_{\mathsf{R}}) \Rightarrow (\varGamma_{\mathsf{L}} \equiv_{\mathsf{nu}} \varGamma_{\mathsf{R}}) \Rightarrow (\varGamma_{\mathsf{L}} \equiv_{\mathsf{poly}} \varGamma_{\mathsf{R}})$$

We next present the *system emulation lemma*, which shows that if $\varGamma_{\mathsf{R}}$ emulates $\varGamma_{\mathsf{L}}$, and $\varGamma_{\mathsf{L}}$ satisfies model $\mathcal{M}$, then $\varGamma_{\mathsf{R}}$ also satisfies $\mathcal{M}$. This allows us to use emulation and indistinguishability relations among systems, e.g. in hybrid arguments.

**Lemma 2 (System emulation lemma).** *Let* $\mathcal{M}$ *be a model and* $\varGamma_{\mathsf{L}} \prec_{\mathsf{poly}} \varGamma_{\mathsf{R}}$ *be two systems. If* $\mathcal{M} \models_{\mathsf{poly}} \varGamma_{\mathsf{L}}$*, then* $\mathcal{M} \models_{\mathsf{poly}} \varGamma_{\mathsf{R}}$*.*

*Proof:* We prove the equivalent claim:

$$(\mathcal{M} \models_{\mathsf{poly}} \varGamma_{\mathsf{L}}) \bigwedge (\neg [\mathcal{M} \models_{\mathsf{poly}} \varGamma_{\mathsf{R}}]) \Rightarrow (\neg [\varGamma_{\mathsf{L}} \prec_{\mathsf{poly}} \varGamma_{\mathsf{R}}])$$

Namely, assume $\Gamma_{\mathsf{L}}$ satisfies $\mathcal{M}$, but $\Gamma_{\mathsf{R}}$ does not satisfy $\mathcal{M}$. We will construct an adversary $\Gamma_{\mathsf{A}}^{LR}$ that distinguishes between $\Gamma_{\mathsf{R}}$ and $\Gamma_{\mathsf{L}}$, which completes the proof.

Without loss of generality, assume that $\mathcal{M} = (\Gamma_{\mathsf{Exp}}(\mathcal{M}), \frac{1}{2})$, i.e. the winning threshold $\alpha(\mathcal{M})$ is $\frac{1}{2}$. Since $\Gamma_{\mathsf{R}}$ does *not* computationally satisfy $\mathcal{M}$, i.e. $\neg\left[\mathcal{M} \models_{\mathsf{poly}} \Gamma_{\mathsf{R}}\right]$, it follows that for some $n$, there exists some $\Gamma_{\mathsf{A}} \in \mathbf{\Pi}_{\mathsf{poly}}$, polynomial $l$, and constant $c$, s.t. for infinitely many integers $k$, holds:

$$\Pr_{R \in \mathcal{R}}\left(X_{k,\mathsf{Exp},\mathsf{outcome},l}\left(C_{LM}(n), \Pi_n, \mathcal{S}(n)\right) = 1\right) > \frac{1}{2} + k^c$$

Where $\Pi_n(\mathsf{A}) = \Gamma_{\mathsf{A}}(n), \Pi_n(\mathsf{Exp}) = \Gamma_{\mathsf{Exp}}(\mathcal{M})(n), \Pi_n(\mathsf{L}) = \Gamma_{\mathsf{R}}(n)$ and $\mathcal{S}(n) = (\mathcal{S}_{\mathsf{A}}(n)||\mathcal{S}_{\mathsf{Exp}}(n)||\mathcal{S}_{\mathsf{R}}(n))^*$, with $\mathcal{S}_\phi(n) = \phi \times \Gamma_\phi(n).I_{IN}$ for $\phi \in \{\mathsf{A}, \mathsf{Exp}\}$ and $\mathcal{S}_{\mathsf{R}}(n) = \mathsf{L} \times \Gamma_{\mathsf{R}}(n).I_{IN}$.

Since the result of the execution depends only on the output after $l_{\mathsf{R}}(k)$ steps, we can assume, without loss of generality, that $\mathcal{S}$ is of length at most $l_{\mathsf{R}}(k)$.

We now use $\Gamma_{\mathsf{A}}$, to design the distinguishing adversary $\Gamma_{\mathsf{A}}^{LR}$. Specifically, let $\Gamma_{\mathsf{A}}^{LR}$ be the natural composition of $\Gamma_{\mathsf{A}}$ and $\Gamma_{\mathsf{Exp}}(\mathcal{M})$, except for following change in the output on $\mathsf{outcome}$. If $\Gamma_{\mathsf{Exp}}(\mathcal{M})$ outputs 1 on $\mathsf{outcome}$, then $\Gamma_{\mathsf{A}}^{LR}$ outputs $\mathsf{R}$ on its $\mathsf{g}$ (guess) output interface. On the other hand, if $\Gamma_{\mathsf{Exp}}(\mathcal{M})$ does not output 1 by the end of the run, after $l(k)$ steps, then $\Gamma_{\mathsf{A}}^{LR}$ flips a coin $b' \in_R \{\mathsf{L}, \mathsf{R}\}$ and outputs 1 if $b' = 1$.

It remains to show, that $\Gamma_{\mathsf{A}}^{LR}$ distinguishes between runs where $\Gamma_{\mathsf{IND}}(n)$ flips $\mathsf{R}$, i.e. $\Gamma_{\mathsf{A}}^{LR}$ interacts with $\Gamma_{\mathsf{R}}$, and between runs where $\Gamma_{\mathsf{IND}}(n)$ flips $\mathsf{L}$, i.e. $\Gamma_{\mathsf{A}}^{LR}$ interacts with the system $\Gamma_{\mathsf{L}}$. Specifically, we complete the proof by showing that, for the same $n, c, l$, and for as above [5] , and for infinitely many integers $k$, holds:

$$\Pr\left(X_{k,\mathsf{IND},\mathsf{outcome},2l}(C_{LR}(n), \Pi_n^{LR}, \mathcal{S}^{LR}(n)) = 1\right) > \frac{1}{2} + k^{c-2} \tag{4}$$

Where $\Pi_n^{LR}(\mathsf{A}) = \Gamma_{\mathsf{A}}^{LR}(n), \Pi_n^{LR}(\mathsf{IND}) = \Gamma_{\mathsf{IND}}(n), \Pi_n^{LR}(\mathsf{L}) = \Gamma_{\mathsf{L}}(n), \Pi_n^{LR}(\mathsf{R}) = \Gamma_{\mathsf{R}}(n)$ and $\mathcal{S}^{LR}(n) = \left(\mathcal{S}_{\mathsf{A}}^{LR}(n)||\mathcal{S}_{\mathsf{IND}}^{LR}(n)||\mathcal{S}_{\mathsf{L}}^{LR}(n)||\mathcal{S}_{\mathsf{R}}^{LR}(n)||\mathcal{S}_{\mathsf{IND}}(n)\right)^*$, where $\mathcal{S}_\phi^{LR}(n) = \phi \times \Gamma_\phi(n).I_{IN}$ for $\phi \in \{\mathsf{A}, \mathsf{L}, \mathsf{R}\}$ and $\mathcal{S}_{\mathsf{IND}}(n) \equiv \{< \mathsf{IND}, \iota > | \iota \in \Gamma_{\mathsf{IND}}(n).I_{IN}\}$.

We prove that Equation 4 holds, by proving that:

$$\Pr\left(X_{k,\mathsf{IND},\mathsf{outcome},2l}(C_{LR}, \Pi, \mathcal{S}^{LR}) = 1 | b = \mathsf{R}\right) > \frac{3}{4} + k^c \tag{5}$$

And that, for every $c_L$ (and sufficiently large $k$), holds:

$$\Pr\left(X_{k,\mathsf{IND},\mathsf{outcome},2l}(C_{LR}, \Pi, \mathcal{S}^{LR}) = 1 | b = \mathsf{L}\right) > \frac{1}{4} - \frac{k^{c_L}}{2} \tag{6}$$

The layer realization experiment flips a fair coin to select $\mathsf{L}$ or $\mathsf{R}$, hence the expected outcome of the experiment is the average of the left-hand-side of

[5] Assuming, without loss of generality, that $|\Gamma_{\mathsf{L}}(n).I_{IN}| \leq |\Gamma_{\mathsf{R}}(n).I_{IN}|$. This is needed only to bound the length of the execution by $2l$.

equations 5 and 6, which is at least $\frac{1}{2} + \frac{1}{4}(2k^c - k^{c_L})$. Since this holds for every $c_L$, we set $c_L = c - 1$; by assuming $k > 4$, we have that:

$$\Pr\left(X_{k,\mathsf{IND},\mathsf{outcome},2l}(C_{LR}(n), \Pi_n^{LR}, \mathcal{S}^{LR}(n)) = 1\right) > \frac{1}{2} + \frac{1}{4}k^{c-1}(k-1) > \frac{1}{2} + k^{c-2}$$

Which shows that equation 4 holds, and therefore completes the proof.

It remains to prove the two equations; we begin with Equation 5. Consider an execution of the indistinguishability game, where the indistinguishability experiment protocol $\Gamma_{\mathsf{IND}}$ picks $b = \mathsf{R}$. This execution is equivalent to that of the model experiment above (except for the adaptation of $\mathsf{outcome}$ by $\Gamma_{\mathsf{A}}^{LR}$), hence for some $c$ and infinitely many values $k$, holds:

$$\begin{aligned}\Pr\Big(X_{k,\mathsf{IND},\mathsf{outcome},2l}(C_{LR}(n), \Pi_n^{LR}, \mathcal{S}^{LR}(n)) = 1\Big| b = \mathsf{R}\Big) \geq \\ \geq \Pr\left(X_{k,\mathsf{Exp},\mathsf{outcome},l}(C_{LM}, \Pi, \mathcal{S}) = 1\right) + \Pr\left(X_{k,\mathsf{Exp},\mathsf{outcome},l}(C_{LM}, \Pi, \mathcal{S}) = 0\right) \cdot \tfrac{1}{2} > \\ > \tfrac{3}{4} + k^c\end{aligned}$$

Hence, Equation 5 holds, for $c^{LR} \leq c_R - 1$.

It remains to prove Equation 6. Recall that $\Gamma_{\mathsf{L}}(n)$ computationally satisfies $\mathcal{M}$, i.e. $\mathcal{M} \models_{\mathsf{poly}} \Gamma_{\mathsf{L}}$. Consider an execution of the indistinguishability game, where the indistinguishability experiment protocol $\Gamma_{\mathsf{IND}}$ picks $b = \mathsf{L}$. This execution is equivalent to that of the layer model experiment above (except for the adaptation of $\mathsf{outcome}$ by $\Gamma_{\mathsf{A}}^{LR}$), hence for every $c_{\mathsf{L}}$:

$$\begin{aligned}\Pr\Big(X_{k,\mathsf{IND},\mathsf{outcome},2l}(C_{LR}(n), \Pi_n^{LR}, \mathcal{S}^{LR}(n)) = 1\Big| b = \mathsf{L}\Big) \geq \\ \geq \Pr\left(X_{k,\mathsf{Exp},\mathsf{outcome},l}(C_{LM}, \Pi, \mathcal{S}) = \perp\right) \cdot \tfrac{1}{2} > \\ > \tfrac{1}{4} - \tfrac{k^c}{2}\end{aligned}$$

This completes the proof. □

## 5 Conclusions and Research Directions

In this work, we try to lay solid, rigorous foundations, to the important methodology of layered decomposition of distributed systems and network protocols, particularly concerning security in adversarial settings. The framework is built on previous works on modeling and analysis of (secure) distributed systems, as described in the introduction. There are many directions that require further research. Here are some:

– The best way to test and improve such a framework, is simply by using it to analyze different problems and protocols; there are many interesting and important problems, that can benefit from such analysis. As one important example, consider the *secure channel layer* problem. Many protocols and applications assume they operate over 'secure, reliable connections'. In practice, this is often done using the standard layers in Figure 1, in one of

two methods. In the first method, we use TLS (for security) over TCP (for reliability) over the 'best effort' service of IP. In the second method, we use TCP (for reliability) over IP-Sec (for security), again over 'best effort' (IP). It would be interesting to define a 'secure, reliable connection' layer, and to analyze these two methods with respect to it.
– There are many desirable extensions to the framework, including: support for corruptions of nodes, including adaptive and/or mobile corruptions (proactive security and forward security); adaptive control of the number of nodes; support for side channels such as timing and power.
– In this work, we have focused on layered configurations. These are sufficient for many scenarios. However, there are other scenarios. It would be interesting to identify important non-layered scenarios, and find appropriate games, specifications and composition properties, which will support them, possibly as generalizations of our definitions and results.
– It would be interesting to explore the relationships between the layered games framework, and other formal frameworks for study of distributed algorithms and protocols (see introduction).
– The framework is based on the computational approach to security, where attackers can compute arbitrary functions on information available to them (e.g. ciphertext). Many results and tools are based on symbolic analysis, see introduction (and [20, 13, 2]). It can be very useful to find how to apply such techniques and tools, within the framework.

## Acknowledgments

We would like to thank Ran Canetti, Alejandro Hevia, Yehuda Lindell, Mark Manulis, Michael Sedletsky and Haya Shulman for interesting discussions and helpful comments. Special thanks to Dominique Unruh, Oded Goldreich and Dennis Hofheinz. Along with other comments, Dominique noted the need for restricting the schedule of the layer realization game (motivating the definition of fair schedules), and Dennis showed us we should restrict the schedule to be efficiently computable.

# Bibliography

[1] M. Abadi and L. Lamport. Composing specifications. *ACM Trans. Program. Lang. Syst.*, 15(1):73–132, 1993.

[2] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[3] M. Backes, M. Dürmuth, D. Hofheinz, and R. Küsters. Conditional Reactive Simulatability. In *ESORICS 2006, 11th European Symposium on Research in Computer Security*, volume 4189 of *Lecture Notes in Computer Science*, pages 424–443. Springer, 2006.

[4] M. Backes, B. Pfitzmann, and M. Waidner. A General Composition Theorem for Secure Reactive Systems. In *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2004.

[5] M. Backes, B. Pfitzmann, and M. Waidner. Secure Asynchronous Reactive Systems. Cryptology ePrint Archive, Report 2004/082, 2004. `http://eprint.iacr.org/`.

[6] Michael Backes, Anupam Datta, Ante Derek, John C. Mitchell, and Mathieu Turuani. Compositional analysis of contract-signing protocols. *TCS: Theoretical Computer Science*, 367, 2006.

[7] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS-97)*, pages 394–405, Los Alamitos, October 20–22 1997. IEEE Computer Society Press. ISBN 0-8186-8197-7.

[8] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. In *CRYPTO '98: Proceedings of the 18th Annual International Cryptology Conference on Advances in Cryptology*, pages 26–45, London, UK, 1998. Springer-Verlag.

[9] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, 2006. ISBN 3-540-34546-9. URL `http://dx.doi.org/10.1007/11761679_25`.

[10] Bruno Blanchet. Computationally sound mechanized proofs of correspondence assertions. In *CSF*, pages 97–111. IEEE Computer Society, 2007. URL `http://dx.doi.org/10.1109/CSF.2007.16`.

[11] Allan Borodin, Jon Kleinberg, Prabhakar Raghavan, Madhu Sudan, and David P. Williamson. Adversarial queuing theory. *JACM: Journal of the ACM*, 48(1):13–38, January 2001.

[12] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119 (Best Current Practice), March 1997. URL `http://www.ietf.org/rfc/rfc2119.txt`.

[13] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems (TOCS)*, 8(1):18–36, 1990.

[14] R. Canetti and M. Fischlin. Universally Composable Commitments. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 19–40, London, UK, 2001. Springer-Verlag.

[15] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001. updated version: Cryptology ePrint Archive, Report 2000/067.

[16] Ran Canetti, Ling Cheung, Dilsun Kirli Kaynar, Moses Liskov, Nancy A. Lynch, Olivier Pereira, and Roberto Segala. Time-bounded task-PIOAs: A framework for analyzing security protocols. In Shlomi Dolev, editor, *Distributed Computing, 20th International Symposium, DISC 2006, Stockholm, Sweden, September 18-20, 2006, Proceedings*, volume 4167 of *Lecture Notes in Computer Science*, pages 238–253. Springer, 2006. ISBN 3-540-44624-9. URL `http://dx.doi.org/10.1007/11864219_17`.

[17] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *J. Cryptology*, 19(2):135–167, 2006.

[18] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *J. Comput. Secur.*, 13(3): 423–482, 2005.

[19] A. Datta, A. Derek, J. C. Mitchell, A. Ramanathan, and A. Scedrov. Games and the impossibility of realizable ideal functionality. In *Theory of Cryptography, 3rd Theory of Cryptography Conference, TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 360–379. Springer, 2006.

[20] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[21] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press, New York, NY, USA, 2004.

[22] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

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

[24] S. Goldwasser and S. Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 365–377, New York, NY, USA, 1982. ACM Press.

[25] S. Goldwasser, S. Micali, and A. Yao. Strong signature schemes. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 431–439, New York, NY, USA, 1983. ACM Press.

[26] Shai Halevi. A plausible approach to computer-aided cryptographic proofs. Report 2005/181, Cryptology ePrint Archive, June 2005. URL `http://eprint.iacr.org/2005/181.pdf`.

[27] A. Herzberg. Secure asynchronous channels.

[28] A. Herzberg and I. Yoffe. Layered Architecture for Secure E-Commerce Applications. In *SECRYPT'06 - International Conference on Security and Cryptography*, pages 118–125. INSTICC Press, 2006.

[29] A. Herzberg and I. Yoffe. On Secure Orders in the Presence of Faults. In *Proceedings of Secure Communication Networks (SCN)*, volume 4116 of *LNCS*, pages 126–140. Springer-Verlag, 2006. New version: Foundations of Secure E-Commerce: The Order Layer, in Cryptology ePrint Archive, Report 2006/352.

[30] A. Herzberg and I. Yoffe. The delivery and evidences layer. Cryptology ePrint Archive, Report 2007/139, 2007. `http://eprint.iacr.org/`.

[31] Amir Herzberg and Igal Yoffe. The layered games framework for specifications and analysis of security protocols. In Ran Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 125–141. Springer, 2008. ISBN 978-3-540-78523-1. URL `http://dx.doi.org/10.1007/978-3-540-78524-8_8`. Draft of full version in cryptology ePrint Archive Report 2006/398.

[32] D. Hofheinz, J. Müller-Quade, and D. Unruh. Polynomial Runtime in Simulatability Definitions. In *CSFW '05: Proceedings of the 18th IEEE Computer Security Foundations Workshop (CSFW'05)*, pages 156–169, Washington, DC, USA, 2005. IEEE Computer Society.

[33] J.F. Kurose and K.W. Ross. *Computer networking: a top-down approach featuring the Internet*. Addison-Wesley, 2003.

[34] R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *CSFW '06: Proceedings of the 19th IEEE Workshop on Computer Security Foundations*, pages 309–320, Washington, DC, USA, 2006. IEEE Computer Society.

[35] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic polytime framework for protocol analysis. In *CCS '98: Proceedings of the 5th ACM conference on Computer and communications security*, pages 112–121, New York, NY, USA, 1998. ACM Press.

[36] N. A. Lynch and M. R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 137–151, New York, NY, USA, 1987. ACM Press.

[37] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 245–254, New York, NY, USA, 2000. ACM Press.

[38] B. Pfitzmann and M. Waidner. A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. In *SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 184–200, Washington, DC, USA, 2001. IEEE Computer Society.

[39] A. C. Yao. Protocols for Secure Computation. In *Proceedings of 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.