The Layered Games Framework

for Specifications and Analysis of Security Protocols

Amir Herzberg and Igal Yoffe

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

Abstract. The layered games framework provides a solid foundation to the accepted methodology of building complex distributed systems, as a 'stack' of independently-developed protocols. Each protocol in the stack, realizes a corresponding 'layer' model, over the 'lower layer'. We define layers, protocols and related concepts. We then prove the *fundamental lemma of layering*. The lemma shows that given a stack of protocols $\{\pi_i\}_{i=1}^{u}$, s.t. for every $i \in \{1, \ldots u\}$, protocol π_i realizes layer L_i over layer L_{i-1}, then the entire stack can be composed to a single protocol $\pi_{u||\ldots||1}$, which realizes layer L_u over layer L_0.

The fundamental lemma of layering allows precise specification, design and analysis of each layer independently, and combining the results to ensure properties of the complete system. This is especially useful when considering (computationally-bounded) adversarial environments, as for security and cryptographic protocols.

Our specifications are based on *games*, following many works in applied cryptography. This differs from existing frameworks allowing compositions of cryptographic protocols, which are based on *simulatability of ideal functionality*.

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* (or modules-). 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); see e.g. Kurose and Ross [30]. The present work is part of an effort, described in Herzberg and Yoffe [25], 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 [25].

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*: when discussing and analyzing a protocol π_i for layer i, running in multiple nodes, we abstract the

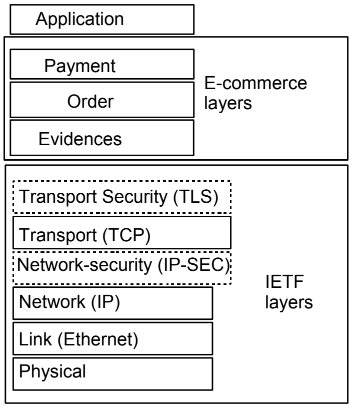


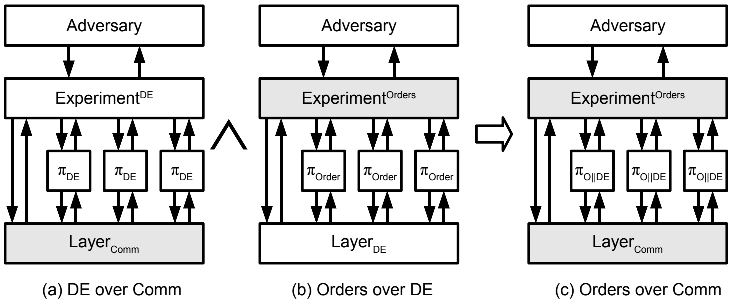
Fig. 1. IETF and e-commerce layers; (optional) security sub-layers marked with dotted contour.

satisfactory behaviors of the lower layers by a single abstract *layer model* L_{i-1}, and the satisfactory behaviors of layer i into abstract layer model L_i. Protocol π_i *realizes layer model* L_i *over layer model* L_{i-1}, if the behavior of (multiple instances of) π_i running over layer model L_{i-1}, satisfies layer model L_i (except with negligible probability). We write this as: $\mathsf{L}_i \vdash \begin{bmatrix} \pi_i \\ \mathsf{L}_{i-1} \end{bmatrix}$.

A pair of protocols π_i and π_{i-1}, of layers $i, i+1$, can be composed into a single protocol, which we denote as $\pi_{i||i-1}$. Our main result is the *fundamental lemma of layering*, showing that by composing protocols of multiple layers, we can implement a high-layer model directly over a low-layer model. Given layer models $\{\mathsf{L}_i\}_{i=0}^{l}$, and protocols $\pi_1, \ldots, \pi_l$, where $\mathsf{L}_i \vdash \begin{bmatrix} \pi_i \\ \mathsf{L}_{i-1} \end{bmatrix}$ for $i = 1, \ldots, l$, their layered composition $\pi_{1||\ldots||l}$ implements L_l over L_0, i.e. $\mathsf{L}_l \vdash \begin{bmatrix} \pi_{1||\ldots||l} \\ \mathsf{L}_0 \end{bmatrix}$. This provides firm foundations to the security of modular and layered architectures, as in Figure 1.

For example, in Herzberg and Yoffe [27] we define the *delivery evidences layer* model L_{DE}, and the lower *communication layer* model $\mathsf{L}_{\mathrm{Comm}}$; and we show a protocol π_{DE} s.t. $\mathsf{L}_{\mathrm{DE}} \vdash \begin{bmatrix} \pi_{\mathrm{DE}} \\ \mathsf{L}_{\mathrm{Comm}} \end{bmatrix}$. Similarly, in Herzberg and Yoffe [26] we define the *orders layer* model $\mathsf{L}_{\mathrm{Orders}}$, and show protocol π_{Order} s.t. $\mathsf{L}_{\mathrm{Orders}} \vdash \begin{bmatrix} \pi_{\mathrm{Order}} \\ \mathsf{L}_{\mathrm{DE}} \end{bmatrix}$. Using the fundamental lemma of layering, the composite protocol $\pi_{\mathrm{DE}||\mathrm{O}}$ realizes the orders layer directly over the communication layer, i.e. $\mathsf{L}_{\mathrm{Orders}} \vdash \begin{bmatrix} \pi_{\mathrm{DE}||\mathrm{O}} \\ \mathsf{L}_{\mathrm{Comm}} \end{bmatrix}$. This is illustrated in Figure 2, where we outline the games each of the protocols

($\pi_{\text{DE}}, \pi_{\text{Order}}$ and their composition $\pi_{\text{DE}||\text{O}}$, the two lower layers (Comm and DE), the two experiments protocols (DE and Orders), and the adversary protocol.



(a) DE over Comm (b) Orders over DE (c) Orders over Comm

Fig. 2. Layering of realizations of the Order and Delivery Evidences (DE) layers

The layered games framework provides solid foundations to the accepted methodology, of using layered architectures (also called reference models), to specify, design, analyze, implement and test each layer independently. In spite of the extensive use of layered architectures, such foundations did exist prior to this work. For example, the IP (Internet Protocol) layer is essentially only required to provide a vaguely-described 'best effort' service. Existing proposals and standard of specifications of layers are only stated informally, often by partial-specification for the *operation* of the protocols, rather than to the *service* the higher layer can rely on. Composition of protocols is 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 [2] 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 an (infinite) state machine with polynomial-time transition function, 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, in developing the layered games framework, we found that some definitional choices could have subtle but critical impact on composability. Details within.

Precise specifications of models for network layers can be hard to write and analyze, since they depend on many implementation and environment aspects. However, such rigorous specifications, and analysis, are critical, at least for security and cryptographic protocols, which must resist adversarial attacks. The layered games framework allows meaningful models, and analysis of implementations (protocols), using standard reduction techniques and composition of protocols (layers).

Compositions and reductions are standard techniques in design and analysis of cryptographic functions and protocols. As noted above, polynomial-time algorithms trivially compose well. However, composition of cryptographic protocols is more challenging. Several frameworks were shown to ensure secure composition, including *universal composability* (UC) by Canetti [14], *reactive simulatability* by Backes, Pfitzmann, and Waidner [5], Pfitzmann and Waidner [34], *observational equivalence* by Lincoln, Mitchell, Mitchell, and Scedrov [32], and more. These frameworks all follow the *ideal functionality paradigm*.

The ideal functionality paradigm is elegant and powerful, and resulted in many significant results, including proofs that arbitrary functions and functionalities can be computed securely, e.g. Goldreich, Micali, and Wigderson [21], Canetti [12, 14]. Grossly simplifying, an 'ideal functionality' for layer i is a single program or ITM F_i, which has multiple copies of the interfaces to layer $i + 1$. Protocol π_i is considered secure, if executions of multiple copies of it over F_{i-1}, are *indistinguishable* from executions of F_i.

However, it may not always be feasible to define an ideal functionality capturing the possible behaviors of a realistic network layer. In fact, even defining the behaviors of each layer is challenging; transforming this into a program, would be impractical or impossible, and may result in over-specification. Note that over-specification of layers (or protocols) is usually considered harmful by practitioners, see e.g. Bradner [9].

This inability to use ideal functionalities as specifications for networking and e-commerce layer models, is our motivation in developing the layered games framework. The layered games framework allows protocol compositions with realistic specifications for network and e-commerce layer models, and with emphasis on simplicity and usability, even at some reduction in scope and generality.

As the name implies, 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. [21], Goldreich [20]. Game playing supports strong analytical tools, e.g. Bellare and Rogaway [8], and may facilitate the use of (semi) automated proof-checking tools, see e.g. Halevi [24].

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 much more, e.g., Goldwasser and Micali [22], Goldwasser, Micali, and Yao [23], Goldreich [20].

Some primitives have secure implementations for game-based specifications, where the corresponding ideal functionalities are not realizable, see Datta, Derek, Mitchell, Ramanathan, and Scedrov [17], Canetti, Kushilevitz, and Lindell [11], Canetti and Fischlin [13]. This provides another motivation for investigating compositions of protocols satisfying game-playing specifications. However, our focus is different: allowing realistic models for network layers, without trying to define them as 'ideal functionality'.

Further related works. Our execution model is closely related to the execution models of I/O Automata of Lynch and Tuttle [33], especially the Probabilistic I/O Automata model of Canetti et al. [15], and to the Reactive Simulatability framework [5, 6, 35]. In an especially related work, Backes et al. [4] define a relaxed notion of *conditional reactive simulatability*, where simulation is required only if the environment fulfills some constraints; however, there are significant differences between the works, most notably their constraints are on the environment and not on the lower layers.

The layered games framework follows the *computational* approach to cryptography, which treats protocols and cryptographic schemes as programs/machines, operating on arbitrary stings (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 [18], Burrows, Abadi, and Needham [10]. Several works investigate compositions of cryptographic protocols with the symbolic approach, e.g. Datta et al. [16] and Backes at al. [3]. We believe that it may be possible and 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 [1]. This may facilitate the use of verification tools; notice also that we use state machines as the basic computational model, which can also be helpful in applying verification tools.

Organization. In Section 2 we define protocols, configurations (of protocols), and executions (of configurations). In Section 3 we define layer games, models and realizations. In Section 4 we present and prove the fundamental lemma of layering. We conclude and discuss future work in Section 5.

For space limitations, the proof and detailed examples of applications of the framework are deferred to the full version of this paper [28]; see also [27, 26].

2 Protocols, configurations and executions

2.1 Protocols

Our basic element of computation is a *protocol*. We use protocols to model all the entities compromising the systems we investigate, including even adversarial entities ('the adversary'). Protocols are state machines[1] that accept input on one

[1] We use state machines, rather than e.g. ITM as in Universal Composability [14], 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.

of few *input interfaces*, and produce output on one or more *output interfaces*. The transition function δ maps the input (interface and value), current state and random bits, 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 δ can depend on two additional inputs: random bits and a security parameter. The random bits may be ignored to define deterministic protocols, including analysis of protocols using 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 π is a tuple $\langle S, I_{IN}, I_{OUT}, \delta\rangle$ where:*

1. *S is a set of states, where $\perp \in S$ is the initial state,*
2. *I_{IN} is a set of input interface identifiers,*
3. *I_{OUT} is a 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 $OUT = S \times \prod_{i \in I_{OUT}} (\{0,1\}^* \cup \{\perp\})$. The outputs consist of a new state, denoted $\delta.S \in S$, and output values $\delta.ov[\iota] \in \{0,1\}^* \cup \{\perp\}$ for each interface $\iota \in I_{OUT}$.*

The protocol is polynomial *if δ is polynomial-time computable, and if the length of the outputs is the same as the length of the inputs*[2]*, plus a polynomial in the security parameter, i.e. $\exists c \in \mathbb{N}$ s.t. $\forall (1^k, s, \iota_i, x, r) \in IN, \iota_o \in I_{OUT}$: $|\delta.ov[\iota_o](k, s, \iota_i, x, r)| \leq |x| + |k|^c$.*

Notations:

Π, Π_{poly}**:** Let Π denote the set of all protocols, and Π_{poly} denote the set of polynomial protocols.

Dot notation: the range of δ is a set of pairs $(s, ov[\iota])$, where $s \in S$ is the new state and $ov[\iota] \in \{0,1\}^* \cup \{\perp\}$ is the output on each output interface $\iota \in I_{OUT}$. To refer directly to the state or the outputs, we use dot notation

[2] 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 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 to prevent exponential blow-up were presented by Küsters [31] describing a general model for systems which satisfy certain acyclic conditions, Canetti [14] and Hofheinz, Müller-Quade, and Unruh [29] for UC, and Backes et al. [6] for reactive simulatability.

as in $\delta.s(\cdot)$ and $\delta.ov[\iota](\cdot)$ respectively. We similarly use dot notation in other places, i.e. $\alpha.\beta$ refers to element β of a record or tuple α.

We can connect protocols, via their interfaces, in different *configurations*, as we define next. We can also connect from an output interface of a protocol, to an input interface of the same protocol; this makes it trivial to compose several protocols into a single protocol, which is useful (see Section 4). Note that if we compose several polynomial protocols in this manner, then the resulting protocol is also polynomial.

2.2 Configuration

We study interactions of multiple protocols, connected via their interfaces; we call the set of interconnected protocols a *configuration*. Configuration are a *directed graph*, whose nodes P are identifiers for 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'. Identification of the input and output interfaces, corresponds to the awareness of the network-layer, e.g. of router or firewall, to the identification of the network interface card on which a packet was received. For example, Figure 2, shows three (homomorphic) configurations. The definition follows.

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

P *is a set of protocol instance identifiers,*

iI, oI *map identifiers in* P *to input and output interfaces, respectively,*

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* $p' = \perp$ *or* $p' \in \mathsf{P}$ *(another instance),*

nI *maps from instance identifier* $p \in \mathsf{P}$ *and an output interface* $\iota \in \mathsf{oI}(p)$*, to input interface* ι'*, where if* $\mathsf{nP}(p, \iota) \in \mathsf{P}$ *then* $\iota' \in \mathsf{iI}(\mathsf{nP}(p, \iota))$*,*

Above, we defined configurations without any 'size' parameter, as required e.g. to analyze protocols and distributed algorithms designed for networks with a variable number of parties (and where complexities may depend on the number of parties). This is for simplicity and to avoid clutter; the extensions to (uniform or non-uniform) 'configuration families' seem quite obvious. Notice that for many applications, e.g. in [27, 26], it may be sufficient to consider a small fixed set of parties.

Still, 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.

2.3 Executions

An *execution* is a sequence of events, each event corresponding to one transition of a protocol π 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 $\pi = \mathit{\Gamma}(p)$ from the protocol identifiers P to the protocols realizing each node.

An important design goal, is that the set of executions of a given configuration C, with a specific mapping to protocols $\mathit{\Gamma}$, 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 further simplify such reductions, we require that executions be a *deterministic* function of explicit random-tape inputs. Specifically, the i^{th} event in the execution, denoted ξ_i, is defined by the (deterministic) transition function of the protocol $\mathit{\Gamma}(p_i)$ 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 $\mathbb{R} = \{R_i \equiv \{0,1\}^*\}_{i=1,2,\ldots}$ be the sequence whose elements are the sets of all binary strings $\{0,1\}^*$; each execution is a deterministic function of the specific sequence $R \in \mathbb{R}$ used in that execution (i.e. $R = \{R_i\}_{i=1,2,\ldots}$ s.t. $(\forall i) R_i = \{0,1\}^*$).

Each protocol instance has its own state, and in each round may decide to invoke interfaces of multiple other protocol instances; see for example the configurations in Figure 2. Therefore, some scheduling mechanism for events is required. To ensure well-defined executions, without any non-deterministic choice (except for the explicit use of the random input strings $R \in \mathbb{R}$), we use a deterministic *schedule* $\mathcal{S}$ (cf. [15]).

A schedule $\mathcal{S}$ of configuration $C = \langle \mathsf{P}, \mathsf{iI}, \mathsf{oI}, \mathsf{nP}, \mathsf{nI} \rangle$, is a sequence of pairs $\mathcal{S} = \{\langle p_i, \iota_i \rangle\}_{i \in \mathbb{N}}$ where $p_i \in \mathsf{P}$. We (later) require protocols to perform correctly for *any* schedule, therefore, the schedule can be considered as adversarial (and not even limited by computational assumptions). On the other hand, the schedule, is defined in advance and cannot depend on the execution (or on the random bits $R \in \mathbb{R}$); in a sense, we separated the adversarial mechanisms into a non-adaptive, computationally-unlimited element (the schedule), and an adaptive, usually computationally-limited element (modeled as a protocol, or multiple protocols, in the configuration, and aware of only inputs on its interfaces). A schedule could, of course, prevent events from happening; to prevent this from being a trivial method to cause executions where the adversary wins, our definitions of games (later) consider the adversary as winning only if some event happens, rather than by the absence of some event.

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 ι, with regular semantics for the *enqueue*, *dequeue*, and *is_non_empty* operations. Other choices may be possible.

Definition 3 (Execution). *Let* $C = \langle \mathsf{P}, \mathsf{iI}, \mathsf{oI}, \mathsf{nP}, \mathsf{nI} \rangle$ *be a configuration. 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*. Let* $\Gamma : \mathsf{P} \rightarrow \Pi$ *be a mapping of the protocol identifiers* P *to specific protocols.*

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

```
FOR ALL $p \in \mathsf{P}$: $s[p] := \perp$;
$\mathrm{Q}[p_1,\iota_1]$.ENQUEUE$(0)$; $X := \{\}$
FOR $i := 1$ TO $\infty$ DO:
  IF ($p_i \in \mathsf{P}$, $\iota_i \in I_{IN}(p_i)$ AND $\mathrm{Q}[p_i, \iota_i]$.IS_NON_EMPTY()) THEN:
    1. $iv_i := \mathrm{Q}[p_i, \iota_i]$.DEQUEUE();
    2. $\langle S, I_{IN}, I_{OUT}, \delta \rangle := \Gamma(p_i)$.
    3. $\langle s[p_i], ov_i[\iota \in I_{OUT}] \rangle := \delta(k, s[p_i], \iota_i, iv_i; R_i)$;
    4. $\forall\, \iota \in I_{OUT}$: IF $ov_i[\iota] \neq \perp$
                          THEN: $\mathrm{Q}[\mathsf{nP}(p_i, \iota), \mathsf{nI}(p_i, \iota)]$.ENQUEUE$(ov_i[\iota])$;
```

Let $X_k(C, \Gamma, \mathcal{S})$ be the random variable $X_k(C, \Gamma, \mathcal{S}; R)$ for $R \in_R \mathbb{R}$.

If all protocols in the range of Γ are *polynomial*, we say that Γ is *polynomial.* If Γ is polynomial, then $X_k(C, \Gamma, \mathcal{S})[l]$ is sampleable in time polynomial in k and l, where $X_k(C, \Gamma, \mathcal{S})[l]$ denotes the l first events of $X_k(C, \Gamma, \mathcal{S})$. 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* $\Gamma : \mathsf{P} \rightarrow \Pi_{\mathsf{poly}}$ *be a mapping of the protocol identifiers* P *to specific polynomial protocols. Then* $X_k(C, \Gamma, \mathcal{S})[l]$ *is sampleable in probabilistic polynomial time (as a function of k and l).*

3 Layer Games, Models and Realizations

From this section, our discussion is focused, for simplicity, on *layered architectures*, as in Figure 1. We believe that it is not too difficult to generalize our concepts and results, but that this will cause (mostly technical) complexities, that may make the resulting definitions less easy to understand and use.

The basic idea of layered architectures, is *abstraction.* Namely, the designer of protocol π_i for layer i, is oblivious to details of lower layers, and only cares about the *layer model* of layer $i-1$, denoted L_{i-1}. The layer model L_{i-1} defines all possible behaviors observable to layer i, resulting from the operation of layer $i-1$ protocols and of all lower layers. The goal of the designer of protocol π_i, for layer i, is to ensure that when instances of π_i operate over any instantiation of Γ_{i-1} of layer model L_{i-1}, the resulting operation satisfies layer model L_i.

In the first subsection below, we give a game-based definition of a *layer model*, with conditions on the outcomes of the game, defining when a protocol Γ_{L} is considered to satisfy layer model L; we denote this by $\mathsf{L} \models \Gamma_{\mathsf{L}}$. In the second subsection, we define the *realization* relation, denoted $\mathsf{L}_U \vdash \left[\overset{\pi_U}{\mathsf{L}_L}\right]$, indicating that protocol π_U, when running over lower layer L_L, realizes layer model L_U.

3.1 Layer Models

We define the layer model L, by a simple zero-sum (win-lose) game between an *adversary protocol*, with identifier A, and a *layer protocol*, with identifier I_{L}. These protocols interact only via a third protocol, the *experiment protocol*, with identifier Exp, as shown in Figure 3. The experiment protocol defines the 'rules of the game', and in particular the outcome, which Exp produces on a designated output interface $\mathsf{outcome}$. Specifically, in every execution, Exp outputs a value on $\mathsf{outcome}$ (at most) once, and this value is a single bit: 1 if the adversary wins (protocol failed the game), and 0 if the adversary losses (protocol passed the game). The game includes an *expected winning rate* $\alpha \in [0, 1]$ (typically $\alpha = 0$ or $\alpha = \frac{1}{2}$), defining the expected (or permitted) probability that the adversary will win, i.e. eventually have 1 on $\mathsf{outcome}$.

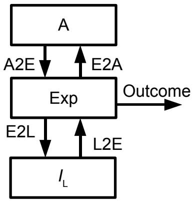


Fig. 3. Layer Model Configuration. If for every Γ_{A} holds $\Pr(\mathsf{outcome} = 1) \leq \alpha +$ $+\mathsf{negl}(k)$, then the layer protocol Γ_{L} satisfies $\mathsf{L} = (\Gamma_{\mathsf{Exp}}, \alpha)$, or: $\mathsf{L} \models \Gamma_{\mathsf{L}}$.

We later implement layer i over layer $i - 1$, by multiple instances of protocol π_i, one in each processor in the network. For simplicity, we assume a constant number of instances n; it seems straightforward to extend the results to allow n to be a parameter. It is convenient to define a separate input and output interfaces between the experiment and each instance. Namely, for $j \in \{1, \ldots, n\}$, the configuration includes interface $\mathsf{E2L}_j$ from Exp to I_{L}, and interface $\mathsf{L2E}_j$ from I_{L} to Exp. Finally, we use a single interface $\mathsf{E2A}$ from Exp to A, and a single interface $\mathsf{A2E}$ from A to Exp. This completes the definition of the *layer modeling game configuration* C_{LM} (for some constant number n of instances).

For $\phi \in \{\mathsf{Exp}, \mathsf{A}\}$, let $\Gamma(\phi) = \Gamma_\phi$ be the protocol instantiating node ϕ; similarly, let $\Gamma(I_\mathsf{L}) = \Gamma_\mathsf{L}$ be a protocol realizing I_L. Given schedule $\mathcal{S}$, let $\mathsf{Exp}^{\Gamma_{\mathsf{Exp}}}_{\Gamma_\mathsf{A},\Gamma_\mathsf{L},\mathcal{S}}(k, l; R)$ denote the output of $\mathsf{outcome}$ after l events in the execution $X_k(C_{LM}, \Gamma, \mathcal{S}; R)$, for $R \in \mathbb{R}$, or $\perp$ if there was no such output.

Definition 4 (Layer model). *A (polynomial)* layer model *is a pair* $\mathsf{L} = (\Gamma_{\mathsf{Exp}}, \alpha)$*, where* Γ_{Exp} *is a (polynomial) protocol and* $\alpha \in [0, 1]$*. We say that protocol* $\Gamma_\mathsf{L} \in \Pi_{\mathsf{poly}}$ computationally satisfies *layer model* L*, and write* $\mathsf{L} \models_{\mathsf{poly}} \Gamma_\mathsf{L}$*, if for every* $\Gamma_\mathsf{A} \in \Pi_{\mathsf{poly}}$*, schedule* $\mathcal{S}$*, polynomial* l *and large enough* k*, holds:*

$$\Pr_{R \in \mathbb{R}}\left(\mathsf{Exp}^{\Gamma_{\mathsf{Exp}}}_{\Gamma_\mathsf{A},\Gamma_\mathsf{L},\mathcal{S}}(k, l(k); R) = 1\right) \leq \alpha + \mathit{negl}(k)$$

where negl *is some negligible function (asymptotically smaller than any strictly positive polynomial), and* $\mathsf{Exp}^{\Gamma_{\mathsf{Exp}}}_{\Gamma_\mathsf{A},\Gamma_\mathsf{L},\mathcal{S}}(k, l; R)$ *is defined as above.*

Protocol Γ_L statistically satisfies L*, if the above holds when protocols are not required to be polynomial, and* perfectly satisfies L *if this holds even when we remove the* $\mathit{negl}(k)$ *term. These notions are denoted* $\mathsf{L} \models_{\mathsf{stat}} \Gamma_\mathsf{L}$ *and* $\mathsf{L} \models_{\mathsf{perf}} \Gamma_\mathsf{L}$*, respectively.*

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

Proposition 2. *For any layer model* L *and any protocol* Γ_L *holds:*

$$\mathsf{L} \models_{\mathsf{perf}} \Gamma_\mathsf{L} \Rightarrow \mathsf{L} \models_{\mathsf{stat}} \Gamma_\mathsf{L} \Rightarrow \mathsf{L} \models_{\mathsf{poly}} \Gamma_\mathsf{L}$$

Notation: we may write $\mathsf{L} \models \Gamma_\mathsf{L}$, when it is obvious that we refer to $\models_{\mathsf{poly}}$.

3.2 Layer Realization Indistinguishability Game

We now define and investigate another game, which we call *indistinguishable layer realization games*, which is similar to indistinguishability games used in many cryptographic definitions, e.g. pseudo-random functions [19], and especially to the 'left-or-right indistinguishability' (LOR) of [7]. Layer realization games are convenient for the common layered and modular ('top-down') design methodologies. As in previous sections, we had to tradeoff generality for simplicity and ease-of-use.

The configuration of layer realization indistinguishability games is illustrated in Figure 4. Like in layer model games, the configuration contains nodes A, Exp and I_L, where A and I_L are connected only via Exp. There are $n + 1$ additional nodes, where n is the (constant) number of instances: n *realization nodes* (instances) $\{\mathsf{R}_j\}_{j=1,\ldots,n}$, and one *lower layer node* I_{LL}.

As in the layer model games, without loss of generality, we use a single input and output interface from the experiment (or 'higher layer') to each instance in I_L, and therefore we will have the interfaces $\mathsf{E2L}_j$, $\mathsf{L2E}_j$, $\mathsf{E2A}$ and $\mathsf{A2E}$ as before. The configuration also includes interfaces $\mathsf{E2R}_j$, $\mathsf{R2E}_j$, $\mathsf{R2L}_j$ and $\mathsf{L2R}_j$, connecting between Exp and R, and between R and I_{LL}. This completes the definition of the *layer realization configuration* C_{LR} (for a fixed number n of instances).

All the realization nodes are instantiated by (mapped to) the same protocol π, which is tested for realization of layer L over lower layer LL. Namely, $(\forall j \in \{1,\ldots,n\})\mathit{\Gamma}(\mathsf{R}_j) = \pi$, where $\mathit{\Gamma}$ is the mapping we will use in the execution of the game (with n instances).

In layer realization indistinguishability games, we use a specific experiment protocol $\mathsf{Exp}^{\mathsf{IND}}$, which we define below, i.e. $\mathit{\Gamma}(\mathsf{Exp}) = \mathsf{Exp}^{\mathsf{IND}}$. Here are some basic details about $\mathsf{Exp}^{\mathsf{IND}}$. Upon initialization, $\mathsf{Exp}^{\mathsf{IND}}$flips a fair coin $b \in_R \{L, R\}$, where L stands for either Layer or Left, and R stands for either Realization or Right. The game ends when $\mathsf{Exp}^{\mathsf{IND}}$ receives a guess b' of either L or R from the adversary A, which arrives on a dedicated Guess input interface. Upon receiving the guess b', $\mathsf{Exp}^{\mathsf{IND}}$outputs on its $\mathsf{outcome}$ output interface 1 if $b = b'$, and 0 otherwise.

Given adversary protocol $\mathit{\Gamma}(\mathsf{A}) = \mathit{\Gamma}_{\mathsf{A}}$, protocols for the two layers $\mathit{\Gamma}(I_{\mathsf{L}}) = \mathit{\Gamma}_{\mathsf{L}}$, $\mathit{\Gamma}(I_{\mathsf{LL}}) = \mathit{\Gamma}_{\mathsf{LL}}$, sequence of random bit sequences $R \in \mathbb{R}$ and schedule $\mathcal{S}$, let ${\mathsf{Exp}^{\mathsf{IND}}}_{\mathit{\Gamma}_{\mathsf{A}},\mathit{\Gamma}_{\mathsf{L}},\mathit{\Gamma}_{\mathsf{LL}},\pi,\mathcal{S}}(k, l; R)$ denote the output of $\mathsf{outcome}$ after l events in the execution $X_k(C_{LR}, \mathit{\Gamma}, \mathcal{S}; R)$, or $\perp$ if there was no such output.

Definition 5 (Layer realization). *Let* L, LL *be two polynomial layer models. Protocol* π computationally realizes *layer model* L *over layer model* LL*, which we denote by* $\mathsf{L} \vdash_{\mathsf{poly}} \begin{bmatrix} \pi \\ \mathsf{LL} \end{bmatrix}$*, if for every polynomial algorithm* $\mathit{\Gamma}_{\mathsf{LL}}$ *s.t.* $\mathsf{LL} \models \mathit{\Gamma}_{\mathsf{LL}}$*, there exists a polynomial algorithm* $\mathit{\Gamma}_{\mathsf{L}}$ *s.t.* $\mathsf{L} \models \mathit{\Gamma}_{\mathsf{L}}$*, s.t. every polynomial algorithm* $\mathit{\Gamma}_{\mathsf{A}}$ *and for every schedule* $\mathcal{S}$ *and every polynomial* l*, for sufficiently large* k *holds*

$$\Pr_{R\in\mathbb{R}}\left({\mathsf{Exp}^{\mathsf{IND}}}_{\mathit{\Gamma}_{\mathsf{A}},\mathit{\Gamma}_{\mathsf{L}},\mathit{\Gamma}_{\mathsf{LL}},\pi,\mathcal{S}}(k, l(k); R) = 1\right) \le \frac{1}{2} + \mathsf{negl}(k)$$

Protocol π statistically realizes *layer model* L *over layer model* LL*, which we denote by* $\mathsf{L} \vdash_{\mathsf{stat}} \begin{bmatrix} \pi \\ \mathsf{LL} \end{bmatrix}$*, if the above holds when protocols are not required to be polynomial, and* perfectly realizes L *over* LL*, which we denote by* $\mathsf{L} \vdash_{\mathsf{perf}} \begin{bmatrix} \pi \\ \mathsf{LL} \end{bmatrix}$ *if this holds even when we remove the* $\mathsf{negl}(k)$ *term.*

In summary, protocol π *realizes* layer model Lover layer model LL, if for every adversary protocol $\mathit{\Gamma}_{\mathsf{A}}$ and every lower-layer protocol $\mathit{\Gamma}_{\mathsf{LL}}$, there is some protocol $\mathit{\Gamma}_{\mathsf{L}}$ satisfying layer L, s.t. the $\mathit{\Gamma}_{\mathsf{A}}$ cannot distinguish between interacting with $\mathit{\Gamma}_{\mathsf{L}}$ and interacting with π operating over $\mathit{\Gamma}_{\mathsf{LL}}$, where $\mathit{\Gamma}_{\mathsf{A}}$ interacts only via $\mathsf{Exp}^{\mathsf{IND}}$. Intuitively, $\begin{bmatrix} \pi \\ \mathit{\Gamma}_{\mathsf{LL}} \end{bmatrix}$ is a good implementation of L, if the adversary A cannot distinguish between it and between some protocol $\mathit{\Gamma}_{\mathsf{L}}$ which satisfies L, when interacting via $\mathsf{Exp}^{\mathsf{IND}}$, better than the trivial winning rate of $\frac{1}{2}$. To complete the description, we now present the *indistinguishability experiment* $\mathsf{Exp}^{\mathsf{IND}}$.

Definition 6 (Layer realization indistinguishability experiment). *Let* $\mathsf{Exp}^{\mathsf{IND}}= \langle S, I_{IN}, I_{OUT}, \delta\rangle$ *be the following protocol:*

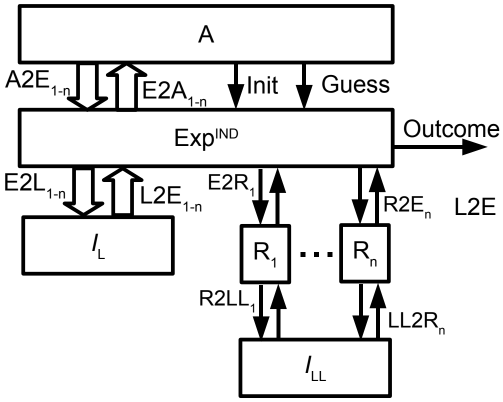


Fig. 4. The Layer Realization Indistinguishability game. Protocol π realizes layer L over layer LL, if for every adversary Γ_{A} and every lower-layer protocol Γ_{LL}, there is some protocol Γ_{L} satisfying layer model L, s.t. the adversary cannot distinguish between Γ_{L} and between the composition of n instances of π over Γ_{LL}.

$S = \{\perp, \mathsf{testing}, \mathsf{done}\}$
$I_{IN} = \{\mathsf{Init}, \mathsf{Guess}\} \cup \{\mathsf{A2E}_j\}_{j=1,\ldots,n} \cup \{\mathsf{L2E}_j\}_{j=1,\ldots,n} \cup \{\mathsf{R2E}_j\}_{j=1,\ldots,n}$
$I_{OUT} = \{\mathsf{outcome}\} \cup \{\mathsf{E2A}_j\}_{j=1,\ldots,n} \cup \{\mathsf{E2L}_j\}_{j=1,\ldots,n} \cup \{\mathsf{E2R}_j\}_{j=1,\ldots,n}$
δ:

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 interface $\mathsf{A2E}_i$, for $i \in \{1, \ldots, n\}$, to corresponding output event on output interface $\mathsf{E2L}_i$ (if $b = \mathsf{L}$) or $\mathsf{E2R}_i$ (if $b = \mathsf{R}$), and all input events on interfaces $\mathsf{L2E}_i$ (if $b = \mathsf{L}$) or $\mathsf{R2E}_i$ (if $b = \mathsf{R}$), to corresponding output events on interface $\mathsf{E2A}_i$.*
3. *When, in* testing *state, the guess input interface* Guess *is invoked with input (guess) $b' \in \{\mathsf{L}, \mathsf{R}\}$, output on* outcome *the value 1 if $b = b'$, and 0 otherwise ($b \neq b'$). Move to the* done *state (and ignores all further inputs).*

4 The Fundamental Lemma of Layering

We now show the fundamental lemma of layering, allowing compositions of protocols of multiple layers. 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 first need to define layering of *protocols*. We actually consider two different variants of protocol layering:

- Layering of two realization protocols $\pi_{\mathsf{L}}, \pi_{\mathsf{LL}}$. As discussed, we assumed (for simplicity) that there are n instantiations of the realization protocol of each layer; each of these has two input interfaces and two output interfaces, one for the higher layer and one for the lower layer. We define $\pi_{\mathsf{LL}||\mathsf{L}} = \begin{bmatrix} \pi_{\mathsf{L}} \\ \pi_{\mathsf{LL}} \end{bmatrix}$ in the obvious way.
- Layering of the n instances of the realization protocol π_{L}, on top of a protocol realizing the lower-layer model Γ_{LL}. We define $\Gamma_{\mathsf{LL}||\mathsf{L}} = \begin{bmatrix} \pi_{\mathsf{L}} \\ \Gamma_{\mathsf{LL}} \end{bmatrix}$ in the obvious way.

Note our convention of using π_x for protocols instantiating realizations (of n instances), and Λ_x for instantiations of a (lower) layer model. Also, note that if π_{L} and π_{LL} (or Γ_{LL}) are polynomial, then $\Gamma_{\mathsf{LL}||\mathsf{L}}$ is also polynomial.

We first present the *'composition preserves satisfaction' lemma*, which justifies considering abstraction of all lower layers, into a single 'virtual protocol'. For both this and the fundamental lemma of layering (below), we present only the computational version (the statistical and perfect versions are similar).

Lemma 1 (Composition preserves satisfaction). *Let* L, LL *be two polynomial layer models, and* $\pi_{\mathsf{L}}, \Gamma_{\mathsf{LL}}$ *be polynomial protocols, such that* π_{L} *computationally realizes* L *over* LL*, namely* $\mathsf{L} \vdash_{\mathsf{poly}} \begin{bmatrix} \pi_{\mathsf{L}} \\ \mathsf{LL} \end{bmatrix}$*, and and* Γ_{LL} *computationally satisfies* LL*, namely* $\mathsf{LL} \models_{\mathsf{poly}} \Gamma_{\mathsf{LL}}$*. Then the composite protocol* $\Gamma_{\mathsf{LL}||\mathsf{L}}$ *satisfies* L*, namely* $\mathsf{L} \models_{\mathsf{poly}} \Gamma_{\mathsf{LL}||\mathsf{L}}$*. Or, as a formula:*

$$\left(\mathsf{L} \vdash_{\mathsf{poly}} \begin{bmatrix} \pi_{\mathsf{L}} \\ \mathsf{LL} \end{bmatrix}\right) \bigwedge \left(\mathsf{LL} \models_{\mathsf{poly}} \Gamma_{\mathsf{LL}}\right) \Rightarrow \left(\mathsf{L} \models_{\mathsf{poly}} \Gamma_{\mathsf{LL}||\mathsf{L}}\right)$$

The *composite realization* 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. We state the lemma for only three layers - generalization for an arbitrary stack is immediate.

Lemma 2 (The Fundamental Lemma of Layering). *Let* $\mathsf{L}_3, \mathsf{L}_2, \mathsf{L}_1$ *be three polynomial layer models, and* π_2, π_3 *be polynomial protocols, such that* π_3 *computationally realizes* L_3 *over* L_2*, and* π_2 *computationally realizes* L_2 *over* L_1*. Then* $\pi_{2||3} = \begin{bmatrix} \pi_3 \\ \pi_2 \end{bmatrix}$ *computationally realizes* L_3 *over* L_1*.*

Furthermore, let Γ_{L_1} *be a polynomial protocol that computationally satisfies* L_1*, namely* $\mathsf{L}_1 \models_{\mathsf{poly}} \Gamma_{\mathsf{L}_1}$*. Then* $\Gamma_{1||2||3} = \begin{bmatrix} \pi_{2||3} \\ \Gamma_1 \end{bmatrix}$ *satisfies* L_3*, i.e.* $\mathsf{L}_3 \models_{\mathsf{poly}} \Gamma_{1||2||3}$.

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, but it is clearly a very ambitious goal, possibly overambitious, and certainly beyond the reach of a single publication. 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 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 interested to explore the relationships between the layered games framework, and other formal frameworks for study of distributed algorithms and protocols (see introdcution).
- The framework is based on the computational approach to security, where attackers can compute arbitrary functions on information available to it (e.g. ciphertext). Many results and tools are based on symbolic analysis, see introduction (and [18, 10, 1]). It can be very useful to find how to apply such techniques and tools, within the framework.

Acknowledgments

We would like to thank Yehuda Lindell, Ran Canetti, Dominique Unruh, Alejandro Hevia, Mark Manulis and Dennis Hofheinz for interesting discussions and helpful comments.

Bibliography

[1] Abadi and Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *JCRYPTOL: Journal of Cryptology*, 15, 2002.

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

[3] Backes, Datta, Derek, Mitchell, and Turuani. Compositional analysis of contract-signing protocols. *TCS: Theoretical Computer Science*, 367, 2006.

[4] 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.

[5] 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.

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

[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] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *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`.

[9] 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`.

[10] Burrows, Abadi, and Needham. A logic of authentication. *ACMTCS: ACM Transactions on Computer Systems*, 8, 1990.

[11] Canetti, Kushilevitz, and Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *JCRYPTOL: Journal of Cryptology*, 19, 2006.

[12] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[13] 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.

[14] 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.

[15] 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, *DISC*, 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`.

[16] 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.

[17] 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.

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

[19] Goldreich, Goldwasser, and Micali. How to construct random functions. *JACM: Journal of the ACM*, 33, 1986.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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`.

[25] 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.

[26] 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.

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

[28] Amir Herzberg and Igal Yoffe. Layered specifications, design and analysis of security protocols. Cryptology ePrint Archive, Report 2006/398, 2006.

[29] 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.

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.