# On Higher-Order Cryptography (Long Version)

Boaz Barak Raphaëlle Crubillé Ugo Dal Lago

**Abstract**

Type-two constructions abound in cryptography: adversaries for encryption and authentication schemes, if active, are modeled as algorithms having access to oracles, i.e. as second-order algorithms. But how about making cryptographic schemes *themselves* higher-order? This paper gives an answer to this question, by first describing why higher-order cryptography is interesting as an object of study, then showing how the concept of probabilistic polynomial time algorithm can be generalized so as to encompass algorithms of order strictly higher than two, and finally proving some positive and negative results about the existence of higher-order cryptographic primitives, namely authentication schemes and pseudorandom functions.

## 1 Introduction

Higher-order computation generalizes classic first-order one by allowing algorithms and functions to not only take *strings* but also *functions* in input. It is well-known that this way of computing gives rise to an interesting computability and complexity theory [26, 25, 31], and that it also constitutes a conceptual basis for the functional programming paradigm, in which higher-order subroutines allow for a greater degree of modularity and conciseness in programs.

In cryptography [19, 20, 24], computation is necessarily randomized, and being able to restrict the time complexity of adversaries is itself crucial: most modern cryptographic schemes are insecure against computationally *unbounded* adversaries. Noticeably, higher-order constructions are often considered in cryptography, in particular when modeling active adversaries, which have access to oracles for the underlying encryption, decryption, or authentication functions, and can thus naturally be seen as second-order algorithms. Another example of useful cryptographic constructions which can be spelled out at different type orders, are *pseudorandom* primitives. Indeed, pseudorandomness can be formulated on (families of) strings, giving rise to so-called pseudorandom *generators* [5], but also on (families of) first-order functions on strings, giving rise to the so-called pseudorandom *functions* [21]. In the former case, again, adversaries (i.e., distinguishers) are ordinary polynomial time algorithms, while in the latter case, they are polytime oracle machines.

Given the above, it is natural to wonder whether standard primitives like encryption, authentication, hash functions, or pseudorandom functions, could be made higher-order. As discussed in Section 2 below, that would represent a way of dealing with code-manipulating programs and their security in a novel, fundamentally interactive, way. Before even looking at the feasibility of this goal, there is one challenge we are bound to face, which is genuinely definitional: how could we even *give* notions of security (e.g. pseudorandomness, unforgeability, and the like) for *second-order* functions, given that those definitions would rely on a notion of *third-order* probabilistic polynomial time adversary, itself absent from the literature? Indeed, although different proposals exist for classes of feasible *deterministic* functionals [9, 25], not much is known if the underlying algorithm has access to a source of randomness. Moreover, the notion of feasibility cryptography relies on is based on the so-called *security parameter*, a global numerical value which controls the complexity of all the involved parties. In Section 3, we give a definition of higher-order probabilistic polynomial time by way of concepts borrowed from game semantics [22, 23, 2], and being

inspired by recent work by Ferée [13]. We give evidence to the fact that the provided definition is general enough to capture a broad class of adversaries of order strictly higher than two.

After having introduced the model, we take a look at whether any concrete instance of a secure higher-order cryptographic primitive can be given. The results we provide are about pseudorandom functions and (deterministic) authentication. We prove on the one hand that those constructions are *not* possible if one insists on them having the expected type (see Section 4.2). On the other hand, we prove (in Section 4.3 below) that second-order pseudorandomness *is* possible if the argument function takes as input a string of *logarithmic* length.

# 2 The Why and How of Authenticating Functions

Encryption and authentication, arguably the two simplest cryptographic primitives, are often applied to *programs* rather than mere data. But when this is done, programs are treated as ordinary data, i.e., as *strings of symbols*. In particular, two different but equivalent programs are seen as different strings, and their encryptions or authentication tags can be completely different objects. It is natural to ask the following: would it be possible to deal with programs as *functions* and not as *strings*, in a cryptographic scenario? Could we, e.g., encrypt or authenticate programs seeing them as *black boxes*, thus without any access to their code?

For the sake of simplicity, suppose that the program $\mathtt{P}$ we deal with has a very simple IO behaviour, i.e. it takes as input a binary string of length $n$ and returns a boolean. Authenticating $\mathtt{P}$ could in principle be done by querying $\mathtt{P}$ on some of its inputs and, based on the outputs to the queries, compute a tag for $\mathtt{P}$. As usual, such an authenticating scheme would be secure if no efficient adversary $\mathcal{A}$ could produce a tag for $\mathtt{P}$ without knowing the underlying secret key $k$ (such that $|k| = n$), unless with negligible probability. Please notice that the adversary, contrarily to the scheme itself, could have access to the code of $\mathtt{P}$, even if that code has not been used during the authenticating process.

But how could security be *defined* in a setting like the above? The three entities at hand have the following types, where $\mathtt{MAC}$ is the authentication algorithm, $\mathbb{S} = \{0,1\}^*$ is the set of binary strings, and $\mathbb{B} = \{0,1\}$ is the set of boolean values:

$$\begin{aligned} \mathtt{P} &: \mathbb{S} \to \mathbb{B} \\ \mathtt{MAC} &: \mathbb{S} \to (\mathbb{S} \to \mathbb{B}) \to \mathbb{S} \\ \mathcal{A} &: ((\mathbb{S} \to \mathbb{B}) \to \mathbb{S}) \to (\mathbb{S} \to \mathbb{B}) \times \mathbb{S} \end{aligned}$$

The first argument of $\mathtt{MAC}$ is the key $k$, which is of course not passed to the adversary $\mathcal{A}$. The latter can query $\mathtt{MAC}_k$ and produce a function and its tag. Its type, as expected, has order three. The above is not an accurate description of the input-output behaviour of the involved algorithms, and in particular of the fact that the length of the input string to $\mathtt{P}$ might be in a certain relation to the length of $k$, i.e., the underlying security parameter. Reflecting all this in the types above is however possible by replacing occurrences of the type $\mathbb{S}$ with *refinements* of $\mathbb{S}$, as follows:

$$\begin{aligned} \mathtt{P} &: \mathbb{S}[n] \to \mathbb{B} \\ \mathtt{MAC} &: \mathbb{S}[n] \to (\mathbb{S}[r(n)] \to \mathbb{B}) \to \mathbb{S}[p(n)] \\ \mathcal{A} &: ((\mathbb{S}[r(n)] \to \mathbb{B}) \to \mathbb{S}[p(n)]) \to (\mathbb{S}[r(n)] \to \mathbb{B}) \times \mathbb{S}[p(n)] \end{aligned}$$

Observe how $\mathtt{MAC}$ is allowed to output a tag of length $p(n)$ where $p(n) \geq n$. Moreover, please note how the input to $\mathtt{P}$ is assumed to have length linear in $n$.

But how could the time complexity of the three algorithms above be defined? While polynomial time computability of the function $\mathtt{P}$ and the authenticating algorithm $\mathtt{MAC}$ can be captured in a standard way using, e.g., oracle Turing machines, the same cannot be said about $\mathcal{A}$. How to, e.g., appropriately account for the time $\mathcal{A}$ needs to "cook" a function $f$ in $\mathbb{S}[n] \to \mathbb{B}$ to be passed to its argument functional? Appealing as it is, our objective of studying higher-order forms of cryptography is actually bound to be nontrivial, even from a purely *definitional* perspective.

Given the above discussion, the contributions of this paper can be described in greater detail, as follows:

- On the one hand, we give a *definition* of a polynomial-time higher-order probabilistic algorithm whose time complexity depends on a global security parameter and which is based on games and strategies, in line with game semantics [22, 23, 2]. This allows to discriminate satisfactorily between efficient and non-efficient adversaries, and accounts for the complexity of first-and-second-order algorithms consistently with standard complexity theory.
- On the other hand, we give some *positive* and *negative* results about the possibility of designing second-order cryptographic primitives, and in particular pseudorandom functions and authentication schemes. In particular we prove, by an essentially information-theoretic argument, that secure deterministic second-order authentication schemes of the kind sketched above *cannot* exist. A simple and direct reduction argument shows that a more restricted form of pseudorandom function exists under standard cryptographic assumptions. Noticeably, the adversaries we prove the existence of are of a very peculiar form, while the ones which we prove impossible to build are quite general.

# 3 Higher-Order Probabilistic Polynomial Time Through Parametrized Games

In this section, we introduce a framework inspired by game semantics, in which one can talk about the efficiency of probabilistic higher-order programs in presence of a global security parameter. While the capability of interpreting higher-order programs is a well-established feature of game semantics, dealing *at the same time* with probabilistic behaviors *and* efficiency constraints has—to the best of the authors' knowledge—not been considered so far. The two aspects have however been tackled *independently*. Several game models of probabilistic languages have been introduced: we can cite here, for instance, the fully abstract model of probabilistic Idealized Algol by Danos and Harmer [11], or the model of probabilistic PCF by Clairambault at al. [8]. About efficiency, we can cite the work by Férée [13] on higher-order complexity and game semantics, in which the cost of evaluating higher-order programs is measured parametrically on the size of *all* their inputs, including functions, thus in line with type-two complexity [9]. We are instead interested in the efficiency of higher-order definitions with respect to the *security parameter*. Unfortunately, existing probabilistic game models do not behave well when restricted to capture feasibility: *polytime computable* probabilistic strategies in the spirit of Danos and Harmer do not *compose* , as we are going to show in Section 3.3.

Contrary to most works in game semantics, we do not aim at building a model of a *particular* programming language, but we take game semantics as our model of computation. As a consequence, we are not bound by requirements to interpret particular programming features or to reflect their discriminating power, and the resulting notions of games and strategies will be very simple.

We present our game-based model of computation in three steps: first, we define a category of deterministic games and strategies called $\mathscr{PG}$—for *parametrized games*—which capture computational agents whose behavior is parametrized by the security parameter. This model ensures that computational agents are *total*: they *always* answer any request by the opponent. In a second step, we introduce $\mathscr{PPG}$, as a sub-category of $\mathscr{PG}$ designed to model those agents whose time complexity is *polynomially bounded* with respect to the security parameter. Finally, we deal with *randomized agents* by allowing them to interact with a *probabilistic oracle*, that outputs (a bounded amount of) random bits.

## 3.1 Parametrized Deterministic Games

Our game model has been designed so as to be able to deal with security properties that—as exemplified by *computational indistinguishability*—are expressed by looking at the behavior of adversaries *at the limit*, i.e., when the security parameter tends towards infinity. The *agents* we

consider are actually *families* of functions, indexed by the security parameter. As such, our game model can be seen as a parametrized version of Hyland's simple games [22], where the set of plays is replaced by a *family* of sets of plays, indexed by the natural numbers. Moreover, we require the total length of any interaction between the involved parties to be polynomially bounded in the security parameter.

We need a few preliminary definitions before delving into the definition of a game. Given two sets $X$ and $Y$, we define $\mathrm{Alt}(X,Y)$ as $\{(a_1,\ldots,a_n) \mid a_i \in X \text{ if } i \text{ is odd}, a_i \in Y \text{ if } i \text{ is even}\}$, i.e., as the set of finite alternating sequences whose first element is in $X$. Given any set of sequences $X$, $\mathsf{Odd}(X)$ (respectively, $\mathsf{Even}(X)$) stands for the subset of $X$ containing the sequences of odd (respectively, even) length. From now on, we implicitly assume that any (opponent or player) move $m$ can be faithfully encoded as a string in an appropriate, fixed alphabet. This way, moves and plays implicitly have a length, that we will indicate through the unary operator $|\cdot|$. We fix a set **Pol** of unary polynomially-bounded total functions on the natural numbers, which includes the identity $\iota$, base-2 logarithm $\lfloor \lg \rfloor$, addition, multiplication, and closed under composition. **Pol** can be equipped with the pointwise partial order: $p \leq q$ when $\forall n \in \mathbb{N}, p(n) \leq q(n)$.

**Definition 1** (Parametrized Games)**.** *A* parametrized game $G = (O_G, P_G, L_G)$ *consists of sets* $O_G$, $P_G$ *of opponent and player moves, respectively, together with a family of non-empty prefix-closed sets* $L_G = \{L_G^n\}_{n\in\mathbb{N}}$, *where* $L_G^n \subseteq Alt(O_G, P_G)$, *such that there is* $p \in \mathbf{Pol}$ *with* $\forall n \in \mathbb{N}. \forall s \in L_G^n. |s| \leq p(n)$. *The union of* $O_G$ *and* $P_G$ *is indicated as* $M_G$, *and is said to be set of* moves *of* $G$.

For every $n \in \mathbb{N}$, $L_G^n$ represents the set of legal plays, when $n$ is the current value of the security parameter. Observe that the first move is always played by *the opponent*, and that for any fixed value of the security parameter $n$, the length of legal plays is bounded by $p(n)$, where $p \in \mathbf{Pol}$. In the following, we often form plays from moves coming from different games or from different copies of the same game. If $s$ is such a play, we indicate, e.g., the sub-play of $s$ consisting of the moves from $G$ as $s_G$.

**Example 2** (Ground Games)**.** *We present here some games designed to model* data-types. *The simplest game is probably the* unit game $\mathbf{1} = (\{?\}, \{*\}, \{L_{\mathbf{1}}^n\}_{n\in\mathbb{N}})$ *with just one opponent move and one player move, where* $L_{\mathbf{1}}^n = \{\varepsilon, ?, ?*\}$ *for every* $n$. *Just slightly more complicated than the unit game is the* boolean game $\mathbf{B}$ *in which the two moves* 0 *and* 1 *take the place of* $*$. *In the two games introduced so far, parametrization is not really relevant, since* $L_G^n = L_G^m$ *for every* $n, m \in \mathbb{N}$. *The latter is not true in* $\mathbf{S}[p] = (\{?\}, \{0,1\}^*, \{L_{\mathbf{S}[p]}^n\}_{n\in\mathbb{N}})$ *with* $L_{\mathbf{S}[p]}^n = \{\varepsilon, ?\} \cup \{?s \mid |s| = p(n)\}$, *which will be our way of capturing strings. A slight variation of* $\mathbf{S}[p]$ *is* $\mathbf{L}[p]$, *in which the returned string can have length* smaller or *equal to* $p(n)$.

**Example 3** (Oracle Games)**.** *As another example, we describe how to construct* polynomial boolean oracles *as games. For every polynomial* $p \in \mathbf{Pol}$ *we define a game* $\mathbf{O}^p$ *as* $(\{?\}, \{0,1\}, \{L_{\mathbf{O}^p}^n\}_{n\in\mathbb{N}})$ *with*

$$L_{\mathbf{O}^p}^n = \{?\} \cup \{?b_1?b_2\ldots?b_m \mid b_i \in \{0,1\} \wedge m \leq p(n)\} \cup \{?b_1?b_2\ldots?b_m? \mid b_i \in \{0,1\} \wedge m < p(n)\}.$$

Our oracle games are actually a special case of a more general construction, that amounts to building, from any game $G$, and any polynomial $p$, a game which consists in playing $G$ at most $p(n)$ times. That is itself nothing more than a bounded version [18] of the exponential construction from models of linear logic [16].

**Definition 4** (Bounded Exponentials)**.** *Let* $G = (O_G, P_G, L_G)$ *be a parametrized game. For every* $p \in \mathbf{Pol}$, *we define a new parametrized game* $!_pG := (O_{!_pG}, P_{!_pG}, L_{!_pG})$ *as follows:*

- $O_{!_pG} = \mathbb{N}_{>0} \times O_G$, *and* $P_{!_pG} = \mathbb{N}_{>0} \times P_G$;
- *For* $n \in \mathbb{N}$, $L_{!_pG}^n$ *is the set of those plays* $s \in Alt(O_{!_pG}, P_{!_pG})$ *such that:*
  - *for every* $i$, *the* $i$*-th projection* $s_i$ *of* $s$ *is in* $L_G^n$.
  - *if a move* $(i+1, z)$ *appears in* $s$ *for* $i \in \mathbb{N}_{>0}$, *then a move* $(i, x)$ *appears at some earlier point of* $s$, *and* $i + 1 \leq p(n)$.

We do not need any switching condition as in so-called AJM games [1]: the impossibility for the observer to switch between the various copies of $G$ when playing in $!_pG$ is a byproduct of our very definition of a game. Observe that the game $\mathbf{O}^p$ is isomorphic to the game $!_p\mathbf{B}$—we can build a bijection between legal plays having the same length.

Games specify *how* agents could play in a certain interactive scenario. As such, they do not represent *one* such agent, this role being the one of *strategies*. Indeed, a strategy on a game is precisely a way of specifying the *deterministic* behavior of an agent, i.e. how the agent plans to react to any possible move by the opponent. We moreover ask our strategies to be total, i.e., that the player cannot refuse to play when it is her turn.

**Definition 5** (Strategies)**.** *A* strategy *on a parametrized game $G = (O_G, P_G, L_G)$ consists of a family $\mathsf{f} = \{\mathsf{f}_n\}_{n\in\mathbb{N}}$, where $\mathsf{f}_n$ is a partial function from $\mathsf{Odd}(L_G^n)$ to $P_G$ such that:*

- *for every $s \in \mathsf{Odd}(L_G^n)$, if $\mathsf{f}_n(s) = x$ is defined, then $sx \in L_G^n$;*
- *$sxy \in Dom(\mathsf{f}_n)$ implies that $x = \mathsf{f}_n(s)$;*
- *for every $s \in \bar{\mathsf{f}}_n$, if $sx \in L_G^n$ then $sx \in Dom(\mathsf{f}_n)$;*

*where $\bar{\mathsf{f}}$ represents the* set of plays characterising $\mathsf{f}$*, defined as the family $\{\bar{\mathsf{f}}_n\}$ where $\bar{\mathsf{f}}_n = \{\varepsilon\} \cup \{s\mathsf{f}_n(s) \mid s \in Dom(\mathsf{f}_n)\} \subseteq L_G^n$.*

Any strategy $\mathsf{f}$ is entirely characterized by its set of plays $\bar{\mathsf{f}}$. As such, it does not need to be effective, i.e., it is entirely possible that $\mathsf{f}$, seen as a function of the history and the security parameter, is an uncomputable function. It should also be noted that any strategy $\mathsf{f}$ can decide how to proceed in the game, namely how to reply to the opponent's last move looking *at the entire history $s$*, and not on just the last move. As a consequence, our strategies are far from being history-free, and there is a reason for that: we would like to be as permissive as possible as for the class of adversaries strategies can model. This being said, the concrete adversaries and constructions we will build in the coming sections actually need only a relatively small portion of the history when deciding how to play next.

Up to now, the games we have described are such that their strategies are meant to represent concrete data: think about how a strategy for, e.g., $\mathbf{B}$ or $\mathbf{O}^p$ could look like. It is now time to build games modeling *functions*, this being embodied by the following construction on games:

**Definition 6** (Constructing Functional Games)**.** *The game $G \multimap H$ is given as $O_{G\multimap H} = P_G + O_H$, $P_{G\multimap H} = O_G + P_H$, and $L_{G\multimap H} = \{s \in Alt(O_{G\multimap H}, P_{G\multimap H}) \mid s_G \in L_G, s_H \in L_H\}$.*

Strategies for the game $G \multimap H$ are meant to model any agent that, when interacting with a strategy in $G$, behaves like a strategy for $H$. When $G$ and $H$ are ground games, this can indeed be seen as a function between the corresponding sets.

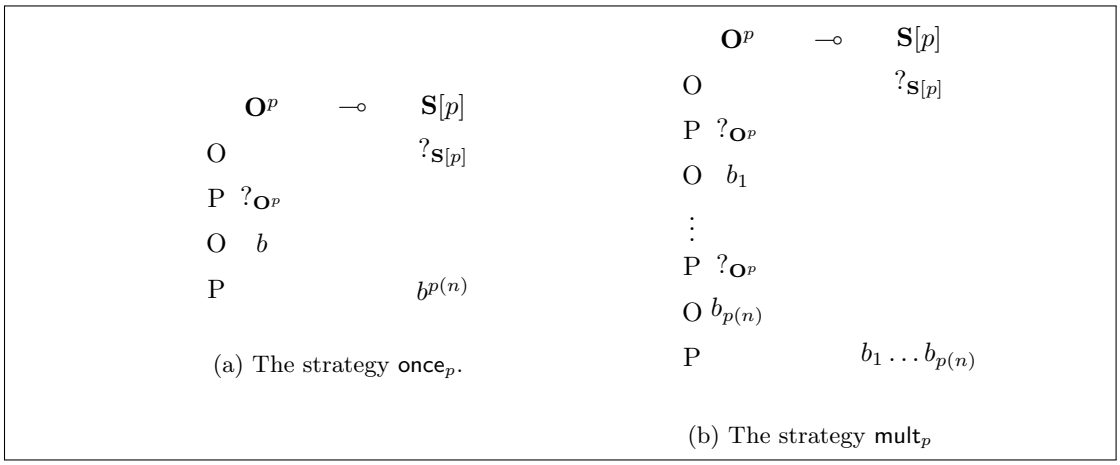
**Example 7.** *As an example, we look at the game $\mathbf{O}^p \multimap \mathbf{S}[p]$, which captures functions returning a string of size $p(n)$ after having queried a binary oracle at most $p(n)$ times. First, observe that:*

$$\mathbf{O}^p \multimap \mathbf{S}[p] = (\{?_{\mathbf{S}[p]}, 0, 1\}, \{?_{\mathbf{O}^p}\} \cup \{0,1\}^\star, \{L^n_{\mathbf{O}^p\multimap\mathbf{S}[p]}\}_{n\in\mathbb{N}}),$$

*where $L_n^{\mathbf{O}^p\multimap\mathbf{S}[p]}$ is generated by the following grammar:*

$$\begin{aligned}
q \in L_n^{\mathbf{O}^p\multimap\mathbf{S}[p]} ::= {} & ?_{\mathbf{S}[p]}o \;\Big|\; ?_{\mathbf{S}[p]}e \;\Big|\; ?_{\mathbf{S}[p]}es && s \in \{0,1\}^* \text{ with } |s| \leq p(n)\\
e ::= {} & \epsilon \;\Big|\; ?_{\mathbf{O}^p}b_1 \ldots ?_{\mathbf{O}^p}b_m && b_i \in \{0,1\}, m \leq p(n)\\
o ::= {} & ?_{\mathbf{O}^p} \;\Big|\; ?_{\mathbf{O}^p}b_1 \ldots ?_{\mathbf{O}^p}b_{m-1}?_{\mathbf{O}^p} && b_i \in \{0,1\}, m \leq p(n)
\end{aligned}$$

*Of course there are* many *strategies for this game, and we just describe two of them here, both making use of the oracle: the first one—that we will call $\mathsf{once}_p$— queries the oracle for a random boolean, and returns the string $0^{p(n)}$ or the string $1^{p(n)}$ depending on the obtained value. It is represented in Figure 1a. The second strategy—denoted $\mathsf{mult}_p$ and represented in Figure 1b—generates a random key of length $p(n)$ by making $p(n)$ calls to the probabilistic oracle.*




(a) The strategy $\mathsf{once}_p$.

(b) The strategy $\mathsf{mult}_p$

Figure 1: Two Distinct Strategies on the Game $\mathbf{O}^p \multimap \mathbf{S}[p]$

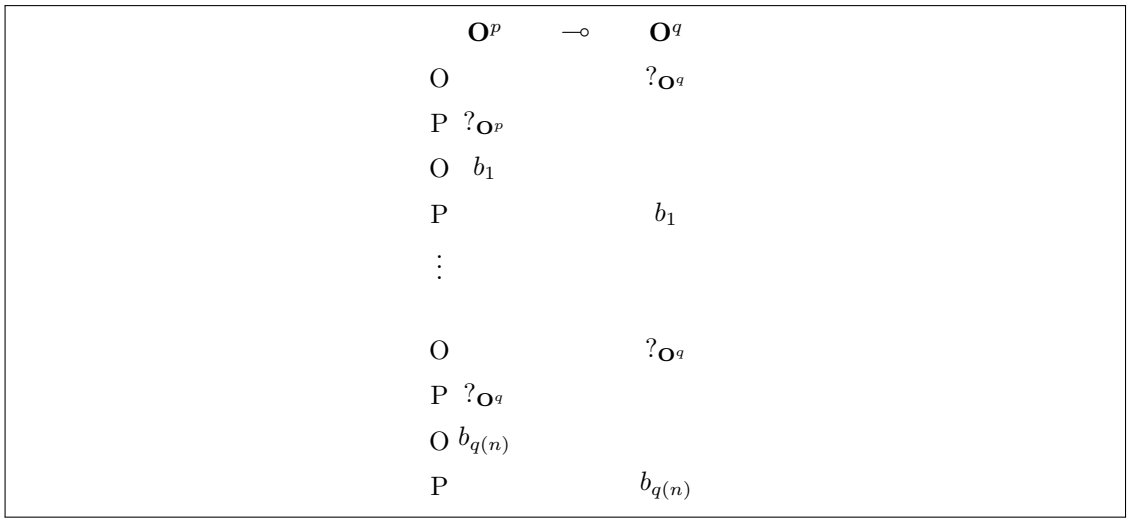



Figure 2: The $w^{p,q}$ strategy on the game $\mathbf{O}^p \multimap \mathbf{O}^q$ with $q \leq p$

**Example 8.** *In this example, we look at a way of* weakening *our probabilistic oracle: more precisely, if we have an oracle that makes $p(n)$ boolean values available, then we can build from it an oracle that gives only $q(n)$ boolean values (whenever $q \leq p$). To formalize this idea in our model, we build a canonical strategy $w^{q,p}$ on the game $\mathbf{O}^p \multimap \mathbf{O}^q$, that we represent in Figure 2.*

We now look at how to *compose* strategies: given a strategy on $G \multimap H$, and $H \multimap K$, we want to build a strategy on $G \multimap K$ that combines them. We define composition as in [22, 32], except that we need to take into account the security parameter $n$.

**Definition 9** (Composition of Strategies)**.** *Let $G, H, K$ be parametrized games, and let $\mathsf{f}$, $\mathsf{g}$ be two strategies on $G \multimap H$ and $H \multimap K$ respectively. We first define the set of* interaction sequences of $\mathsf{f}$ and $\mathsf{g}$ *as:*

$$(\mathsf{f} \parallel\!\parallel \mathsf{g})_n = \{s \in (M_G + M_H + M_K)^\star \mid s_{G,H} \in \overline{\mathsf{f}}_n \wedge s_{H,K} \in \overline{\mathsf{g}}_n\},$$

*From there, we define the composition of $\mathsf{f}$ and $\mathsf{g}$ as the unique strategy $\mathsf{f};\mathsf{g}$ such that:*

$$\overline{\mathsf{f};\mathsf{g}}_n = \{s_{G,K} \mid s \in (\mathsf{f} \parallel\!\parallel \mathsf{g})_n\}.$$

We can check that $\mathsf{f};\mathsf{g}$ is indeed a strategy on $G \multimap K$, and that moreover composition, seen as an operation on strategies, is associative and admits an identity. We can thus define $\mathscr{PG}$ as the

category whose objects are parametrized games, and whose set of morphisms $\mathscr{PG}(G, H)$ consists of the parametrized strategies on the game $G \multimap H$.

## 3.2 Polytime Computable Strategies

Parametrized games have been defined so as to have polynomially bounded *length*. However, there is no guarantee on the effectiveness of its strategies, i.e., that the next player move, can be computed algorithmically from the history, uniformly in the security parameter. This can be however tackled by considering a subcategory of $\mathscr{PG}$ in which strategies are not merely functions, but can be (efficiently) computed:

**Definition 10** (Polytime Computable Strategies)**.** *Let $G$ be a parametrized game, and $\mathsf{f}$ be a strategy on $G$. We say that $\mathsf{f}$ is* polytime computable *when there exists a polynomial time Turing machine which on input $(1^n, s)$ returns $\mathsf{f}(n)(s)$.*

All strategies we have given as examples in the previous section are polytime computable. For example, the two strategies from Example 7 are both computable in linear time. More generally, observe how the polytime constraint is formulated on an algorithm computing an ordinary first-order function, so the problem of giving a notion of feasible higher-order function is reduced, by way of games, to the same problem for first-order functions, for which a canonical notion of polytime bounded computation exists. The same idea has been pursued also by Férée [13] who, however, works in a setting in which no global security parameter is available.

**Proposition 11** (Stability of Polytime Computable Strategies)**.** *Let $G, H, K$ be polynomially bounded games. If $\mathsf{f}, \mathsf{g}$ are polytime computable strategies, respectively on $G \multimap H$, and $H \multimap K$, then $\mathsf{f}; \mathsf{g}$ is itself a polytime computable strategy.*

*Proof.* Recall that both $G$ and $H$ are polynomially bounded: it means that there exist two polynomials $P$ and $Q$ that bound the length of any play in $G$ and $H$, respectively. Moreover $\mathsf{f}$ and $\mathsf{g}$ are polytime computable: there exists a polynomials $R$ such that for every $n \in \mathbb{N}$, $\mathsf{h} \in \{\mathsf{f}, \mathsf{g}\}$, and $s \in Dom(\mathsf{h}_n)$, $\mathsf{h}_n(s)$ can be computed in less than $R(n)$ time. We need to show that for any play $s$ of odd length in the domain of $(\mathsf{f}; \mathsf{g})_n$, we are able to compute *in polynomial time* the unique player move $x$ such that $(\mathsf{f}; \mathsf{g})_n(s) = x$. The difficulty here is that we need a more constructive way of building the strategy $\mathsf{f}; \mathsf{g}$ than the one given in Definition 9. Intuitively, what we need here is an algorithm to find in polynomial time—with respect to $n$—the sequence $t \in (\mathsf{f} \parallel\!\!\mid \mathsf{g})_n$ of minimal length such that $t_{G,K} = sx$ with $x \in P_{G \multimap K}$. Observe that the length of $t$ is bounded by $P(n) + Q(n)$, since its projection on $G, H$ must be a valid play on $G \multimap H$, and its projection on $H, K$ a valid play on $H \multimap K$. We will show that the cost of building each move in $t$ is also polynomially bounded, thus the whole algorithm runs in polynomial time. We describe now our algorithm to build $t$: we need an intermediate partial function:

$$\phi : (\mathsf{f} \parallel\!\!\mid \mathsf{g})_n \times (O_G + P_K)^\star \to (\mathsf{f} \parallel\!\!\mid \mathsf{g})_n \times (O_G + P_K)^\star.$$

Intuitively, $\phi$ takes an already built sequence $t \in (\mathsf{f} \parallel\!\!\mid \mathsf{g})_n$, and a list of opponent moves $l \in (M_G + M_K)^\star$, and build the sequence in $(\mathsf{f} \parallel\!\!\mid \mathsf{g})_n$ after one supplementary move: depending on the last position in $t$, this next move can either have been produced by applying $\mathsf{f}$ or $\mathsf{g}$, or by taking the next opponent move given in the list $l$. We now give the definition of the partial function $\phi$, by case analysis over the form of its argument. Our case analysis is based on the fact that the set $M_G + M_H + M_K$ can be decomposed as $(O_G + P_G) + O_{G \multimap H} + O_{H \multimap K}$.

$$\phi(\epsilon, o \cdot l) = (o, l) \qquad \text{when } o \in O_K \tag{1}$$
$$\phi(t \cdot x, l) = (t \cdot x \cdot \mathsf{f}(t.x_{|G,H}), l) \text{ when } x \in O_{G \multimap H}, \text{ and } \mathsf{f}(t.x_{|G,H}) \in P_H \tag{2}$$
$$\phi(t \cdot x, o.l) = (t \cdot x \cdot \mathsf{f}(t.x_{|G,H}), l) \text{ when } x \in O_{G \multimap H}, \text{ and } \mathsf{f}(t.x_{|G,H}) \in O_G \tag{3}$$
$$\phi(t \cdot x, l) = (t \cdot x \cdot \mathsf{g}(t.x_{|G,H}), l) \text{ when } x \in O_{H \multimap K}, \text{ and } \mathsf{g}(t.x_{|G,H}) \in O_H \tag{4}$$
$$\phi(t \cdot x, o.l) = (t \cdot x \cdot \mathsf{g}(t.x_{|G,H}), l) \text{ when } x \in O_{H \multimap K}, \text{ and } \mathsf{g}(t.x_{|G,H}) \in P_K \tag{5}$$
$$\phi(t \cdot x, o.l) = (t \cdot x \cdot o, l) \text{ when } x \in O_G + P_K \tag{6}$$

We are interested by the action of $\phi$ on the following set:

$$S_n := \{(t, l) \mid t \in (\mathsf{f} \parallel\!\!\parallel \mathsf{g})_n \text{ and } t_{|G,K} \cdot l \in \overline{\mathsf{f};\mathsf{g}}\}.$$

An important point to remark in the definition of $\phi_{|S_n}$ is that in the cases (3) and (5), it is actually *always true* that $\mathsf{f}(t.x_{|G,H}) = o$ and $\mathsf{g}(t.x_{|G,H}) = o$ respectively: it is because we deal with *deterministic* strategies. From there, we can see that the following invariant holds for $\phi$: if $\phi(t, l) = (t', l')$, then $t_{|G,K} \cdot l = t'_{|G,K} \cdot l'$. We denote $\mathrm{inv}(t, l) = t_{|G,K} \cdot l$. As a consequence, $\phi$ sends $S_n$ into $S_n$.

We denote by $NF^{\mathrm{odd}}(\phi)$ the $(t, l)$ in $S_n$ such that $\phi$ is not defined on $(t, l)$, and moreover $\mathrm{inv}(t, l)$ is of odd length. We can show that if $(t, l) \in NF^{\mathrm{odd}}(\phi)$, then $l$ is empty, and besides– when denoting $x$ the last move of $t$:

- or $x \in O_{G \multimap H}$, and $\mathsf{f}(t_{|G,H}) \in O_G$. In this case, we denote $\mathrm{next}(t, l) = \mathsf{f}(t_{|G,H})$;
- or $x \in O_{H \multimap K}$, and $\mathsf{g}(t_{|G,H}) \in P_K$. In this case, we denote $\mathrm{next}(t, l) = \mathsf{g}(t_{|H,K})$;

As a consequence, we are able to compute $(\mathsf{f};\mathsf{g})_n(s)$ the following way: we apply $\phi$ from $(\varepsilon, s)$ until obtaining an element $(t, \varepsilon) \in NF^{odd}(\phi)$, and then we return $\mathrm{next}(t, \varepsilon)$.

We still need to show that this procedure terminates, and that moreover it requires polynomial time. Observe first that whenever we apply $\phi$ on $(t, l)$, the length of $t$ increases, so the number of times we need to apply $\phi$ before obtaining a normal form is bounded by the maximal length of any $t \in \mathsf{f} \parallel\!\!\parallel \mathsf{g}$, which is smaller that $P(n) + Q(n)$. Moreover, when applying $\phi$ itself, we need first to do the case analysis on $t$, and then to apply $\mathsf{f}$ or $\mathsf{g}$: this takes at most $R(n) + P(n) + Q(n)$ steps. After having obtained the normal form, we need to apply once more either $\mathsf{f}$ or $\mathsf{g}$ to obtain the next move: the time to do that is once again bounded by $R(n) + Q(n) + S(n)$. To sum up, we obtain a procedure that computes the next move in $O((R + P + Q)^2(n))$. □

For elementary reasons, the identity strategy on any game $G$ is polytime computable. We can thus write $\mathscr{PPG}$ for the the sub-category of $\mathscr{PG}$ whose objects are paramterized games, and whose morphisms are polytime computable strategies.

Let us now consider the algorithm `MAC` from Section 2. Its type can be turned into the parametrized game $\mathbf{S}[\iota] \multimap !_q(\mathbf{S}[r] \multimap \mathbf{B}) \multimap \mathbf{S}[p]$. The bounded exponential $!_q$ serves to model the fact that the argument function can be accessed a number of times which is *polynomially bounded* on $n$. As a consequence, `MAC` can only query the argument function a number of times which is negligibly smaller than the number of possible queries, itself exponential in $n$ (if $r(n) \geq n$). As we will see in Section 4, this is the key ingredient towards proving security of such a message authentication code to be unattainable.

### 3.3 Probabilistic Strategies

Both in $\mathscr{PG}$ and in $\mathscr{PPG}$, strategies on any game $G$ are ultimately *functions*, and the way they react to stimuli from the environment is completely deterministic. How could we reconcile all this with our claim that the framework we are introducing adequately models *randomized* higher-order computation? Actually, one could be tempted to define a notion of *probabilistic strategy* in which the underlying function family $\{f_n\}_{n \in \mathbb{N}}$ is such that $f_n$ returns *a probability distribution* $f_n(s)$ of player moves when fed with the history $s$. This, however, would lead to some technical problems when *composing strategies*: it would be hard to keep the composition of two efficient strategies itself efficient. We illustrate the problem in the example below: we build two probabilistic strategies that are polytime computable, but whose composition is not polytime computable. Our construction is actually dependent on the existence of a *one-way permutation*; the existence of such a function is a standard cryptographic assumption.

**Example 12.** *Let $h : \{0,1\}^\star \to \{0,1\}^\star$ be a one-way permutation. It means that for every $n$, $h$ is a permutation over $\{0,1\}^n$, and moreover it is* easy to compute*, i.e.,there exist a polynomial $P$ and an algorithm that computes $h(s)$ from any input $s$ in time $\leq P(|s|)$, but hard to inverse,*

*in the sense that no polytime algorithm can compute a pseudo-inverse for $h$ with non-negligible probability. From there, we build two polytime computable strategies:*

- *Using the one-way function $h$, we first build the strategy $\mathsf{f}_h$ on the game $\mathbf{S}[\iota] \multimap !^2\mathbf{S}[\iota]$, represented in Figure 3a: it corresponds to an agent that takes a string as argument, and that must produce two strings. Its behaviour is as follows: first it asks for its argument $s$, then the first string it returns is $h(s)$, and the second is $s$. Since $h$ can be computed in polynomial time, we can see that $\mathsf{f}_h$ is indeed a polytime (deterministic) strategy.*
- *We then build a probabilistic strategy Unif on the game $\mathbf{S}[\iota]$, that, for each value $n$ of the security parameter, returns a string uniformly distributed on $\{0,1\}^n$–it is represented in Figure 3b. Observe that this strategy is also polytime computable.*

*We now look at the composition $\mathsf{f}_h \circ Unif$, and we are going to show that it is* not *polytime computable. This probabilistic stratgey is represented in Figure 3c. When we represent formally $\mathsf{f}_h \circ Unif$ as a partial function $f : \mathsf{Odd}(L^n_{!^2\mathbf{S}[\iota]}) \to P_{!2\mathbf{S}[\iota]}$, we obtain:*

$$f(1, ?_{\mathbf{S}[\iota]}) = \sum_{l \in \{0,1\}^n} \frac{1}{2^n} \cdot \{(1, h(l))^1\};$$

$$f((1, ?_{\mathbf{S}[\iota]}) \cdot (1, l) \cdot (2, ?_{\mathbf{S}[\iota]})) = \{(2, l')^1\}, \qquad \text{with } h(l') = l.$$

*But since $h$ is a one way permutation, we see that there is no polytime algorithm able to compute $f((1, ?_{\mathbf{S}[\iota]}) \cdot (1, l) \cdot (2, ?_{\mathbf{S}[\iota]}))$, thus $\mathsf{f}_h \circ Unif$ is not polytime computable: as a consequence probabilistic polytime computable strategies are not stable by composition under the one-way-permutation assumption (actually, it is possible to extend the reasoning with the weaker one-way function assumption).*

It turns out that a much more convenient route consists, instead, in defining a *probabilistic* strategy on $G$ simply as a *deterministic* polytime strategy on $\mathbf{O}^p \multimap G$, namely as an ordinary strategy having access to polynomially many random bits. Actually, we have already encountered strategies of this kind, namely $\mathsf{once}_p$ and $\mathsf{mult}_p$ from Example 7. This will be our way of modeling higher-order probabilistic computations.

Parametrized games and probabilistic strategies can be themselves seen as a category whose morphisms (from the game $G$ to the game $H$) are pairs in the form $(q, \mathsf{f})$, where $\mathsf{f}$ is a strategy in $\mathbf{O}^q \multimap (G \multimap H)$. This category can be proved to be symmetric monoidal closed, although cartesian closure fails: duplication is not available in its full generality, but only in bounded form, which, we conjecture, is enough to get the structure of a bounded exponential situation [6].

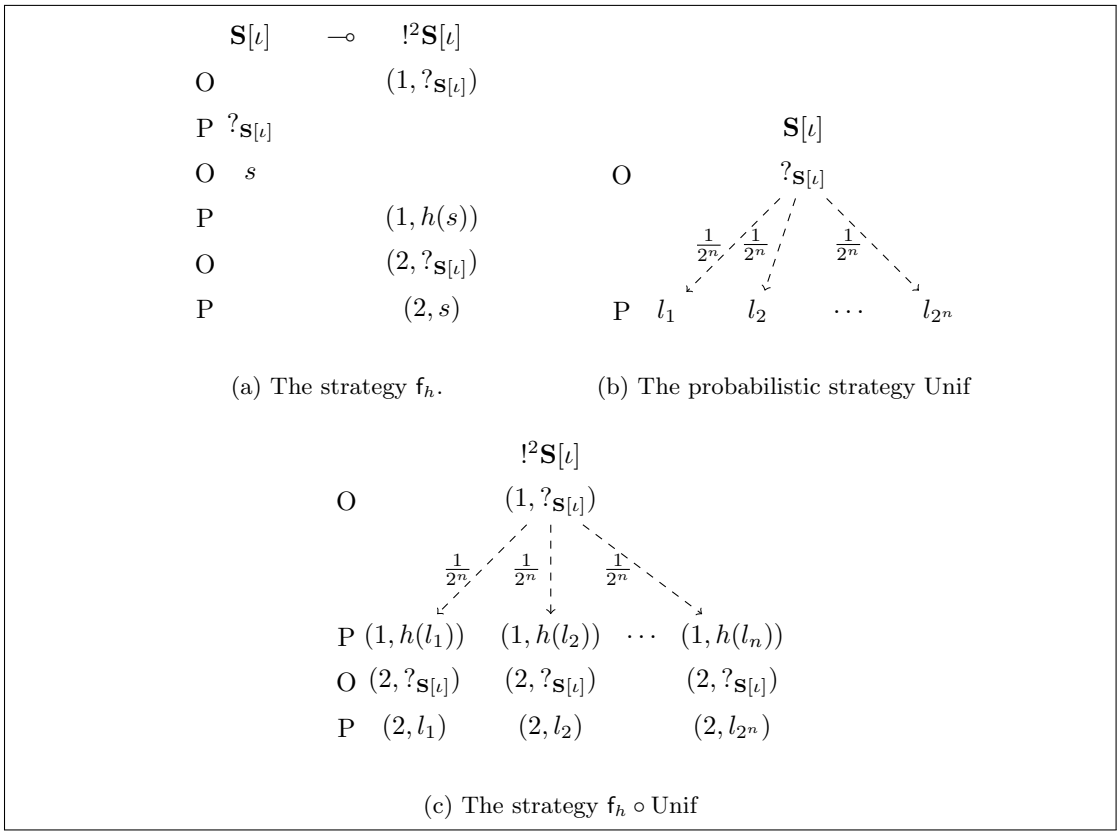
Given a probabilistic strategy $\mathsf{f}$ on $G$ (i.e. a strategy on $\mathbf{O}^q \multimap G$) and $p \in \mathbf{Pol}$, we indicate as $!_p\mathsf{f}$ the strategy in which $p(n)$ copies of $\mathsf{f}$ are played, *but in which* randomness is resolved just once and for all, i.e. $!_p\mathsf{f}$ is the naturally defined strategy on $\mathbf{O}^q \multimap !_pG$ in which the $q(n)$ random bits are all queried for at the beginning of the play, after the first opponent move.

We want now to be able to *observe* the result of a computation. Similarly to what is done for instance in [32], we observe only at ground types: for instance for the boolean type, the observable will be a pair $(p_0, p_1)$, where $p_0$ is the probability that the program returns false, and $p_1$ the probability that the program returns true. In our model, the ground types are those games where the only legal plays are the initial question by the opponent, followed by a terminal answer of the player.

**Definition 13.** *Let $G$ a game in SG. We say that $G$ is* observable *when for every $n \in \mathbb{N}$:*

$$L^n_G \subseteq \{?_G\} \cup \{?_G r \mid r \in P_G\}.$$

Observe that the games $\mathbf{B}$ and $\mathbf{S}[p]$ that we considered in Example 2 are indeed observable games. However, it is not the case of the game build using the $\multimap$ construct, for instance $\mathbf{S}[p] \multimap \mathbf{S}[2p]$.




(a) The strategy $\mathsf{f}_h$.

(b) The probabilistic strategy Unif

(c) The strategy $\mathsf{f}_h \circ \mathrm{Unif}$

Figure 3: Polytime probabilistic strategies are not compositional

Recall that in our framework, a *probabilistic* program of some ground type $G$ is actually represented by a strategy $\mathsf{f}$ on the game $\mathbf{O}^p \multimap G$, for some polynomials $p$. At this point, it should be noted that this polynomial $p$ is somehow arbitrary, since if we take a polynomial $q \geq p$, we can easily transform $\mathsf{f}$ in a strategy on $\mathbf{O}^q \multimap G$, using the strategy on $\mathbf{O}^q \multimap \mathbf{O}^p$ highlighted in Example 7. In order to recover information about the probabilistic behavior of the program, we compute the probability that the return value will be $r$: to do that we look at all the possible sequences of booleans outputed by the oracle that lead to the player returning $r$ under the strategy $\mathsf{f}$, and then we suppose that those sequences are uniformly distributed, in a probabilistic sense.

**Definition 14.** *Let $G$ be an observable game, and $\mathsf{f}$ a strategy on the game $\mathbf{O}^p \multimap G$. For any $r \in P_G$, we define the* probability of observing $r$ when executing $\mathsf{f}$ *as the function:*

$$Prob_{\mathsf{f}}(r) : \mathbb{N} \to [0,1]n \mapsto \sum_{k \leq p(n)} \sum_{\substack{(b_1,\ldots,b_k)\in\{0,1\}^k \\ with\ (?\cdot?_{\mathbf{B}}\cdot b_1 \ldots ?_{\mathbf{B}}\cdot b_k \cdot r)\in(\bar{\mathsf{f}})_n}} \frac{1}{2^k}$$

Observe that if $\mathsf{f}$ is a strategy on $\mathbf{O}^p \multimap \mathsf{f}$, and $w_{q,p}$ the canonical strategy on $\mathbf{O}^q \multimap \mathbf{O}^p$– as defined in Example 7, then for every value $r$, $\mathrm{Prob}_{\mathsf{f}}(r) = \mathrm{Prob}_{w_{q,p};\mathsf{f}}(r)$: it means that the probability of observing $r$ is independant of the choice of the polynomial oracle–as soon as it produces *at least* as many random booleans as asked by the agent.

### 3.4 On the Expressive Power of Probabilistic Strategies

A few words about the expressive power of probabilistic strategies—seen as a model of higher-order randomized computation—are now in order. For trivial reasons, every probabilistic strategy for the game $\mathbf{S}[\iota] \multimap \mathbf{L}[p]$ can be precisely simulated by a a probabilistic Turing machine working in polynomial time. Conversely, every such machine can be turned into a probabilistic strategy for the aforementioned game, once $p$ is chosen as a sufficiently large polynomial. Similarly, behind any probabilistic strategy for the game $\mathbf{S}[\iota] \multimap !_q(\mathbf{L}[p] \multimap \mathbf{L}[r]) \multimap \mathbf{L}[s]$ there is an probabilistic *oracle* Turing machine working in polynomial time. The converse statement, however, can be proved *only assuming* the oracle with which the machine interacts to produce outputs polynomially related (in size) to the inputs.

More generally, the intrinsic restriction parametrized games impose on the *length* of any interaction indeed poses some limitations as to what strategies can do, and in particular to how they can interact with the environment. This implies that our framework is fundamentally inadequate as a characterization of, say, the basic feasible functionals [9]. We claim, however, that cryptography most often deals with situations in which, even if some of the parties can be computationally *unbounded*, the length of the interaction between them, but also the size of the exchanged messages, are *polynomially* bounded. The interested reader is invited to take a look at, e.g., the cryptographic experiments in [24]. Even in interactive proofs, in which no restrictions is put on complexity of the prover, the amount and size of the exchanged messages is by definition polynomially bounded.

## 4 The (In)feasibility of Higher-Order Cryptography

In this section, we give both negative and positive results about the possibility of defining a deterministic polytime strategy for the game $\mathbf{S}[\iota] \multimap !_q(\mathbf{S}[r] \multimap \mathbf{B}) \multimap \mathbf{S}[p]$ which could serve to authenticate functions. When $r$ is linear, this is impossible, as proved in Section 4.2 below. When, instead, $r$ is logarithmic (and $q$ is at least linear), a positive result can be given, see Section 4.3.

But how would a strategy for the game $!_q(\mathbf{S}[r] \multimap \mathbf{B}) \multimap \mathbf{S}[p]$ look like? Plays for this game are in Figure 4. A strategy for such a game is required to determine the value of the query $s_{i+1} \in \mathbb{S}[r(n)]$ based on $t_1, \ldots, t_i \in \mathbb{B}$. Moreover, based on $t_1, \ldots, t_m$ (where $m \leq q(n)$), the strategy should be able to produce the value $v \in \mathbb{S}[p(n)]$. Strictly speaking, the strategy should also be able to respond to a situation in which the opponent directly replies to a move $(i, ?_{\mathbf{B}})$ by

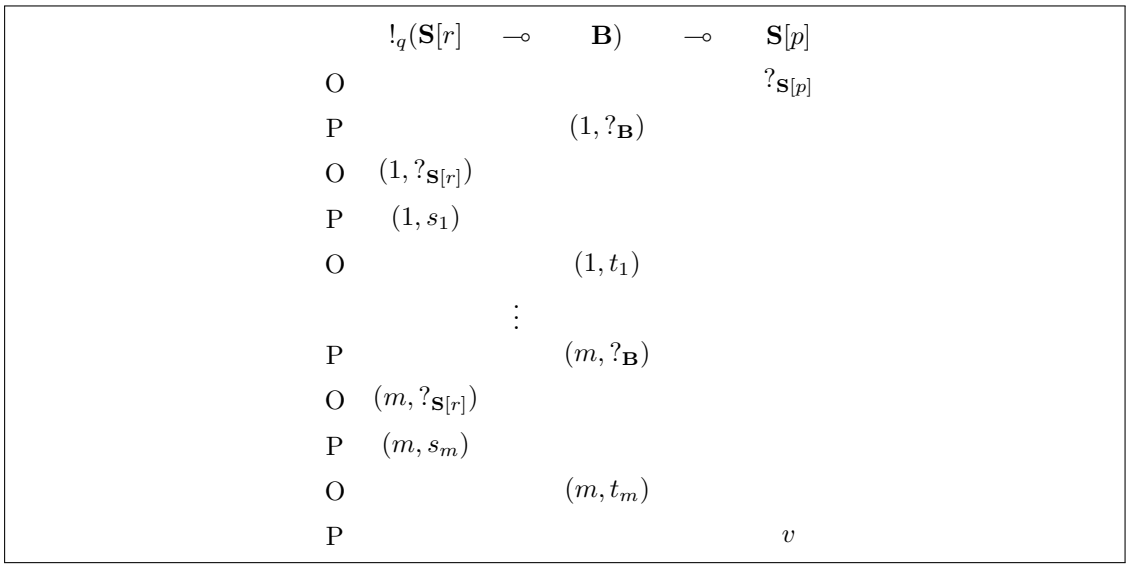



Figure 4: Plays (of Maximal Length) for the game $!_q(\mathbf{S}[r] \multimap \mathbf{B}) \multimap \mathbf{S}[p]$

way of a truth value $(i, t_i)$, without querying the argument. This is however a signal that the agent with which the strategy is interacting represents a *constant* function, and we will not consider it in the following.

The way we will prove deterministic authentication impossible when $r$ is linear consists in showing that since $q$ is polynomially bounded (thus negligibly smaller than the number of possible queries of type $\mathbf{S}[r]$ any function is allowed to make to its argument), there are many argument functions $\mathbb{S}[r(n)] \to \mathbb{B}$ which are indistinguishable, and would thus receive the same tag. In the following, we prove that the (relatively few) coordinates on which the argument function is queried can even be efficiently determined.

## 4.1 Efficiently Determining Influential Variables

A key step towards proving our negative result comes from the theory of influential variables in decision trees. In this section, we are going to give some preliminary definitions about it, without any aim at being comprehensive (see, e.g., [29]).

From now on, metavariables like $N, M, L$ stand for natural number unrelated to the security parameter, unless otherwise specified. Given a natural number $N \in \mathbb{N}$, $[N]$ denotes the set $\{1, \ldots, N\}$. Whenever $j \in [N]$, $e_j \in \mathbb{S}[N]$ is the binary string which is everywhere 0 except on the $j$-th component, in which it is 1.

**Definition 15** (Variance and Influence)**.** *For every distribution $\mathcal{D}$ over $\mathbb{S}[N]$, and $F : \mathbb{S}[N] \to \mathbb{B}$, we write $\mathbf{Var}_{\mathcal{D}}(F)$ for the value $\mathbb{E}(F(\mathcal{D})^2) - \mathbb{E}(F(\mathcal{D}))^2 = \Pr_{x,y\sim\mathcal{D}}(F(x) \neq F(y))$, called the* variance of $F$ under $\mathcal{D}$. *For every distribution $\mathcal{D}$ over $\mathbb{S}[N]$, $F : \mathbb{S}[N] \to \mathbb{B}$, and $j \in [N]$, we define the* influence of $j$ on $F$ under $\mathcal{D}$, *written $\mathbf{Inf}^j_{\mathcal{D}}(F)$, as $\Pr_{x\sim D}[F(x) \neq F(x \oplus e_j)]$.*

The quantity $\mathbf{Inf}^j_{\mathcal{D}}(F)$ measures how much, on the average, changing the $j$-th input to $F$ influences its output. If $F$ does not depend too much on the $j$-th input, then $\mathbf{Inf}^j_{\mathcal{D}}(F)$ is close to 0, while it is close to 1 when switching the $j$-th input has a strong effect on the output.

**Example 16.** *Let $PARITY_N : \mathbb{S}[N] \to \mathbb{B}$ be the parity function on $N$ bits. It holds that*

$$\begin{aligned}\mathbf{Inf}^j_{\mathcal{D}}(PARITY_N) &= \Pr_{x\sim\mathcal{D}}[PARITY_N(x) \neq PARITY_N(x \oplus e_j)] \\ &= \sum_x \mathcal{D}(x) \cdot |PARITY_N(x) - PARITY_N(x \oplus e_j)| = \sum_x \mathcal{D}(x) = 1\end{aligned}$$

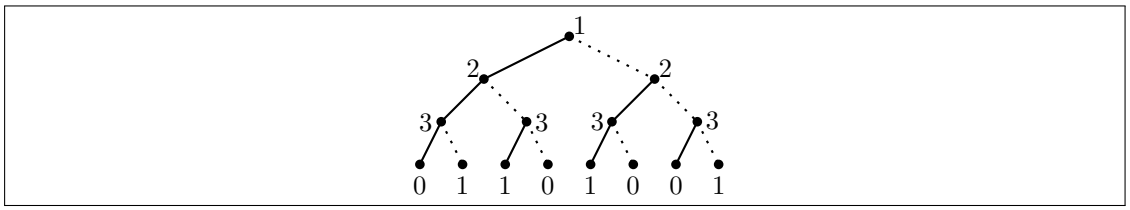



Figure 5: A Decision Tree for $PARITY_3$

*This indeed matches the intuition: changing any one coordinate makes the output to change, independently on the distribution from which the input is drawn.*

If $F : A \to \mathbb{S}[L]$, and $t \in [L]$, we define $F_t : A \to \mathbb{B}$ to be the function that on input $x \in A$ outputs the $t$-th bit of $F(x)$. The kind of distributions over $\mathbb{S}[N]$ we will be mainly interested at are the so-called *semi-uniform* ones, namely those in which some of the $N$ bits have a fixed value, while the others take all possible values with equal probability. It is thus natural to deal with them by way of partial functions. For every partial function $g : [N] \to \mathbb{B}$ we define $Dom(g) \subseteq [N]$ to be the set of inputs on which $g$ is defined, and $U_g$ to be the uniform distribution of $x$ over $\mathbb{S}[N]$ *conditioned on* $x_j = g(j)$ for every $j \in Dom(g)$, i.e., the distribution defined as follows:

$$U_g(x) = \begin{cases} \frac{1}{2^{N-|Dom(g)|}} & \text{if } x_j = g(j) \text{ for every } j \in Dom(g); \\ 0 & \text{otherwise.} \end{cases}$$

If a distribution $\mathcal{D}$ can be written as $U_g$, where $g : [N] \to \mathbb{B}$, we say that $\mathcal{D}$ is an $Dom(g)$-*distribution*, or a *semi-uniform* distribution. Given a distribution $\mathcal{D} : \mathbb{S}[N] \to \mathbb{R}_{[0,1]}$, some index $j \in [N]$ and a bit $b \in \mathbb{B}$, the expression $\mathcal{D}[j \leftarrow b]$ stands for the conditioning of $\mathcal{D}$ to the fact that the $j$-th boolean argument is $b$. Note that if $\mathcal{D}$ is an $S$-distribution and $j \in [N] \setminus S$, then $\mathcal{D}[j \leftarrow b]$ is an $S \cup \{j\}$-distribution.

A crucial concept in the following is that of a decision tree, which is a model of computation for boolean functions in which the interactive aspects are put upfront, while the merely computational aspects are somehow abstracted away.

**Definition 17** (Decision Tree)**.** *Given a function $F$, a* decision tree $T$ for $F$ *is a finite ordered binary tree whose internal nodes are labelled with an index $i \in [N]$, whose leaves are labelled with a bit $b \in \mathbb{B}$, and such that whenever a path ending in a leaf labelled with $b$ is consistent with $x \in \mathbb{S}[N]$, it holds that $F(x) = b$. The* depth *of any decision tree $T$ is defined the same as that of any tree.*

**Example 18.** *An example of a decision tree that computes the function $PARITY_3 : \mathbb{S}[3] \to \mathbb{B}$ defined in Example 16 can be found in Figure 5.*

The following result, which is an easy corollary of some well-known results in the literature (i.e. Corollary 1.2 from [29]), put the variance and the influence in relation whenever the underlying function can be computed by way of a decision tree of limited depth.

**Lemma 19.** *Suppose that $F$ is computable by a decision tree of depth at most $q$ and $g : [N] \to \mathbb{B}$ is a partial function. Then there exists $j \in [N] \setminus Dom(g)$ such that*

$$\mathbf{Inf}^j_{U_g}(F) \geq \frac{\mathbf{Var}_{U_g}(F)}{q}$$

Every decision tree $T$ makes on any input a certain number of queries, which of course can be different for different inputs. If $\mathcal{D}$ is a distribution, $S$ is a subset of $[N]$ and $T$ is a decision tree, we define $\Delta_{\mathcal{D},S}(T)$ as the expectation over $x \sim \mathcal{D}$ of the number of queries that $T$ makes on input $x$ outside of $S$, which is said to be *the average query complexity of $T$ on $\mathcal{D}$ and $S$*. The following result relates the query complexity before and after the underlying semi-uniform distribution is updated: if we fix the value of a variable, then the average query complexity goes down (on the average) by at least the variable's influence:

**Lemma 20.** *For every decision tree $T$ computing a function $F$, $S \subseteq [N]$, $j \in [N] \setminus S$, and $S$-distribution $\mathcal{D}$, it holds that*

$$\tfrac{1}{2}\Delta_{\mathcal{D}[j\leftarrow 0],S\cup\{j\}}(T) + \tfrac{1}{2}\Delta_{\mathcal{D}[j\leftarrow 1],S\cup\{j\}}(T) \leq \Delta_{\mathcal{D},S}(T) - \mathbf{Inf}^j_{\mathcal{D}}(F)$$

*Proof.* We start by showing that

$$\Delta_{D,S\cup\{j\}}(T) \leq \Delta_{D,S}(T) - \mathbf{Inf}^j_{\mathcal{D}}(F) \tag{7}$$

For every $L \in [N]$, let $I_L$ be the random variable that is equal to one if $L$ is queried by $T$ on $x \sim D$ and equal to 0 otherwise. Therefore we can rewrite (7) as

$$\sum_{L\notin S\cup\{j\}} \mathbb{E}(I_L) \leq \sum_{L\notin S} \mathbb{E}(I_L) - \mathbf{Inf}^j_{\mathcal{D}}(F)$$

or

$$\mathbb{E}(I_j) \geq \mathbf{Inf}^j_{\mathcal{D}}(F)\ . \tag{8}$$

But for every $x$ on which $j$ is not queried by the tree $T$, $T(x) = T(x \oplus e_j)$. Hence if $j$ is influential w.r.t. $x$ then certainly $j$ is queried, and (8) follows. For obvious reasons $\mathcal{D} = \frac{1}{2}\mathcal{D}[j \leftarrow 0] + \frac{1}{2}\mathcal{D}[j \leftarrow 1]$ and so in particular

$$\Delta_{\mathcal{D},S\cup\{j\}}(T) = \tfrac{1}{2}\Delta_{\mathcal{D}[j\leftarrow 0],S\cup\{j\}}(T) + \tfrac{1}{2}\Delta_{\mathcal{D}[j\leftarrow 1],S\cup\{j\}}(T)$$

from which the thesis easily follows, given (7). □

By somehow iterating over Lemma 20, we can get the following result, which states that fixing enough coordinates, the variance can be made arbitrarily low, and that those coordinates can be efficiently determined:

**Theorem 21.** *For every $F : \mathbb{S}[N] \to \mathbb{S}[L]$ such that for every $t \in [L]$, $F_t$ is computable by a decision tree of depth at most $Q$, and every $\varepsilon > 0$, there exist a natural number $m \leq LQ^2/\varepsilon$ and a partial function $g : [N] \to \mathbb{B}$ where $|Dom(g)| \leq m$ such that*

$$\mathbf{Var}_{U_g}(F_t) \leq \varepsilon \tag{9}$$

*for every $t \in \{1, \ldots, L\}$. Moreover, there is a polytime randomized algorithm $\mathsf{A}$ that on input $F$, $\delta > 0$, and $\varepsilon > 0$, makes at most $\mathcal{O}(LN) \cdot poly(Q/(\delta\varepsilon))$ queries to $F$ and outputs such a partial function $g$ with $|Dom(g)| \leq \mathcal{O}((LQ^2)/(\varepsilon\delta))$ with probability at least $1 - \delta$.*

*Proof.* We here give the main ingredients of the proof, referring to [3] for a more detailed account. The algorithm $\mathsf{A}$ proceeds by iteratively fixing new coordinates between the $N$ many ones the function $F$ depends on, stopping when the variance of all the functions $F_t$ on the obtained semi-uniform distribution falls significantly below $\varepsilon$. The next coordinate to be fixed is chosen by estimating, using statistical methods, the influence of *all* the possible coordinates. Using similar methods, $\mathsf{A}$ can also estimate accurately the variance, and stop when enough coordinates are fixed. The role of Lemma 19 is to guarantee that if the variance is not too low, an influential variable can always be found, while the one of Lemma 20 consists in guaranteeing that a bounded number of iterations is enough. □

*Proof.* For every $t \in [L]$ let $T_t$ be a tree computing $F_t$ and having depth at most $q$. The algorithm $\mathbf{A}$ works as follows:

**Algorithm $\mathbf{A}(F, \varepsilon, \delta)$ :**

1. $i \leftarrow 0$;
2. $g_0 \leftarrow \emptyset$;
3. $D_0 \leftarrow U_{g_0}$;

4. $\gamma \leftarrow \delta/(NLq^2/(\delta\varepsilon))$
5. For every $t \in [L]$, perform $v_t \leftarrow \textbf{EstimateVar}(D_i, F_t, \varepsilon/3, \gamma)$;
6. If for every $t \in [L]$, it holds that $v_t \leq (2/3)\varepsilon$, then return $g_i$, otherwise continue.
7. Let $t_\uparrow$ be a $t$ such that $v_t \geq 2\varepsilon/3$;
8. For every $j \in [N]$, perform $n_j \leftarrow \textbf{EstimateInf}(D_i, F_{t_\uparrow}, \varepsilon/(10q), \gamma)$;
9. Let $j$ be such that $n_j \geq 0.2 \cdot \varepsilon/q$;
10. $b \leftarrow \{0, 1\}$;
11. $g_{i+1} \leftarrow g_i \cup \{(j, b)\}$;
12. $D_{i+1} \leftarrow U_{g_{i+1}}$;
13. $i \leftarrow i + 1$.
14. Go back to 5.

The algorithm can be studied in two steps:

- We first of all analyse the complexity of the algorithm **A**. An analysis of the runtime gives us the following result:

  **Proposition 22.** *The number of times Algorithm* **A** *executes its main loop is, in expectation, at most* $O(Lq^2/\varepsilon)$.

  *Proof.* We define the potential function

  $$\varphi(i) = \sum_{t=1}^{L} \Delta_{D_i, s(g_i)}(T_t)$$

  where $T_t$ is the tree computing $F_t$. The algorithm does not of course have access to this tree but this potential function is only used in the analysis. Initially, we are guaranteed that $\varphi(0) \leq Lq$ and by definition $\varphi(i) \geq 0$ for every $i$. Hence to show that we end in expected number of steps $O(Lq^2/\varepsilon)$ it is enough to show that in expectation

  $$\mathbb{E}[\varphi(i) - \varphi(i+1)] \geq \Omega(\varepsilon/q)$$

  for every step $i$ in which we do not stop. Indeed, let $i$ be such a step and let $t_0$ be the index such that our estimate for $Var(F_{t_0}(D_i)) \geq (2/3)\varepsilon$ and $j_0$ the influential variable for $F_{t_0}$ dud we find. Recall that $g_{i+1}$ is obtained by extending $g_i$ to satisfy $g_{i+1}(j_0) = b$ for a random $b \in \mathbb{B}$. For every $t \neq t_0$, the expectation over $b$ of $\Delta_{D_{i+1}, s(g_{i+1})}(T_t)$ is equal to the expected number of queries that $T_t$ makes on $D_i$ that do not touch the set $s(g_i) \cup \{j\}$. This number can only be smaller than $\Delta_{D_i, s(g_i)}(T_t)$. For $t = t_0$, by Lemma 20, $\mathbb{E}[\Delta_{D_{i+1}, s(g_{i+1})}(F_t)] \leq \mathbb{E}[\Delta_{D_{i+1}, s(g_{i+1})}(F_t)] + \Omega(\varepsilon/q)$. Hence, in expectation $\varphi(i+1)$ is smaller than $\varphi(i)$ by an additive factor of $\Omega(\varepsilon/q)$ which is what we wanted to prove. □

- A further discussion is needed to ensure that the desired level of accuracy is actually attained by the algorithm. The reason why $\gamma$ is set to $\delta/T$ (where $T = O(NLq^2/(\delta\varepsilon))$ is our bound on the number of times we need to use these estimates) is to ensure $\gamma$ to be small enough so that we can take a union bound over the chances that any of our estimates fail. Thus the algorithm will only stop when it reach $D_i = U_{g_i}$ for which the variance of $F_t$ is smaller than $\varepsilon$ for every $t \in [L]$.

- Once we have Proposition 22, the theorem follows by noting that if the expected number of iterations of the main loop is $m$, the probability that we make more than $m/\delta$ iterations is at most $\delta$. (Since the potential function undergoes a biased random walk, a more refined analysis can be used to show that $O(m \log(1/\delta))$ iterations will suffice, but this does not matter much for our final application and so we use the cruder bound of $m/\delta$.)

□

## 4.2 On the Impossibility of Authenticating Functions

Theorem 21 tells us that for every first-order boolean function which can be computed by a decision tree of low depth, there exist relatively few of its coordinates that, once fixed, determine the function's output with very high probability. If $N$ is exponentially larger than $Q$, in particular, there is no hope for such a function to be a secure message authentication code. In this section, we aim at proving the aforementioned claim. In order to do it, we build a third-order randomized algorithm, which will be shown to fit into our game-theoretic framework.

More specifically, we are concerned with the cryptographic properties of strategies for the parametrized game $SOF_{q,r,p} = !_q(\mathbf{S}[r] \multimap \mathbf{B}) \multimap \mathbf{S}[p]$ and, in particular, with the case in which $r$ is the identity $\iota$, i.e. we are considering the game $LINSOF_{q,p} = SOF_{q,\iota,p}$. Any such strategy, when deterministic, can be seen as computing a family of functions $\{F_n\}_{n\in\mathbb{N}}$ where $F_n : (\mathbb{S}[n] \to \mathbb{B}) \to \mathbb{S}[p(n)]$. How could we fit all this into the hypotheses of Theorem 21?

The definitions of variance, influence, and decision tree can be easily generalised to functions in the form $F : (\mathbb{S}[N] \to \mathbb{B}) \to \mathbb{S}[M]$. Of course the underlying distribution $\mathcal{D}$ must be a distribution over functions $\mathbb{S}[N] \to \mathbb{B}$. The parameter $N$ can be fixed in such a way that $n < N < 2^n$, where $n$ is the security parameter. For simplicity we will choose $N$ to be a power of 2, which hence divides $2^n$. In other words, $N = 2^p$, where $p$ is strictly between $\log_2(n)$ and $n$. Eventually we will set $N$ to be some large enough polynomial in $n$, i.e., $p$ will be in the form $a \cdot \log_2(n) + b$.

**Definition 23** (Extensions)**.** *For every $x \in \mathbb{S}[N]$, we define the* extension of $x$*, denoted by $f_x$ as the function $f_x : \mathbb{S}[n] \to \mathbb{B}$ such that for every $i \in [2^n]$ (identifying $\mathbb{S}[n]$ with the numbers $\{0, \ldots, 2^n - 1\}$ in the natural way), it holds that $f_x(i) = x_{\lfloor i/N \rfloor + 1}$. That is, $f_x$ is the function that outputs $x_1$ on the first $2^n/N$ inputs, outputs $x_2$ on the second $2^n/N$ inputs, and so on and so forth. If $F : (\mathbb{S}[n] \to \mathbb{B}) \to \mathbb{B}$ is a function, then we define $\tilde{F} : \mathbb{S}[N] \to \mathbb{B}$ as follows: for every $x \in \mathbb{S}[N]$, $\tilde{F}(x) = F(f_x)$. Given a distribution $\mathcal{D}$ over $\mathbb{S}[N]$, a distribution over functions $\mathbb{S}[n] \to \mathbb{B}$ can be formed in the natural way as $f_{\mathcal{D}}$.*

We will also make use of the following slight variation on the classic notion of Hamming distance: define $H(\cdot, \cdot)$ to be the so-called *normalized Hamming distance*. In fact, we overload the symbol $H$ and use it for both *strings* in $\mathbb{S}[N]$ and *functions* in $\mathbb{S}[n] \to X$ for some set $X$. That is, if $x, y \in \{0,1\}^N$ then $H(x,y) = \Pr_{j\in[N]}[x_j \neq y_j]$ while if $f, g \in \mathbb{S}[n] \to X$ then $H(f,g) = \Pr_{i\in\mathbb{S}[n]}[f(i) \neq g(i)]$. We use the following simple lemma

**Lemma 24.** *For every $x, y \in \mathbb{S}[N]$, $H(f_x, f_y) = H(x,y)$*

*Proof.* If we choose $i \in \mathbb{S}[n]$ then (identifying $\mathbb{S}[n]$ with $\{0, \ldots 2^n - 1\}$), since $N$ divides $2^n$, the distribution $\lfloor i/N \rfloor$ is uniform over $\{0, \ldots, N-1\}$ and so the two probabilities in the definition of the Hamming distance are the same. More formally:

$$\begin{aligned} H(f_x, f_y) &= \Pr_{i\sim\mathbb{S}[n]}[f_x(i) \neq f_y(i)] = \sum_{i\in\mathbb{S}[n]} \frac{1}{2^n} |f_x(i) - f_y(i)| \\ &= \sum_{i\in\mathbb{S}[n]} \frac{1}{2^n} \left|x_{\lfloor i/N \rfloor + 1} - y_{\lfloor i/N \rfloor + 1}\right| = 2^{n-p} \sum_{j\in[N]} \frac{1}{2^n} |x_j - y_j| \\ &= \sum_{j\in[N]} \frac{1}{2^p} |x_j - y_j| = \Pr_{j\sim[N]}[x_j \neq y_j] = H(x,y). \end{aligned}$$

□

Finally, the following ground game will be useful in the following as a way to represent partial functions as ground objects. The game $\mathbf{T}[q]$ is a slight variation on $\mathbf{S}[q]$ in which the returned string is in the ternary alphabet $\{0, 1, \bot\}$. Any strategy for $\mathbf{T}[q]$ can thus be seen as representing a (family of) partial functions from $[q(n)]$ to $\mathbb{B}$.

**Theorem 25.** *For every $\varepsilon, \delta > 0$, there is a polytime probabilistic strategy $\mathsf{infvar}_{\varepsilon,\delta}$ on the game $!_s(LINSOF_{q,p}) \multimap \mathbf{T}[t]$ such that for every deterministic strategy $\mathsf{f}$ on $LINSOF_{q,p}$ computing $\{F_n\}_{n\in\mathbb{N}}$, the composition $(!_s\mathsf{f}); \mathsf{infvar}_{\varepsilon,\delta}$, with probability at least $1-\delta$, computes some functions $g_n : [t(n)] \to \mathbb{B}$ such that $\mathbf{Var}_{f_{U_{g_n}}}(F_n) \leq \varepsilon$ and $|Dom(g_n)| \leq \mathcal{O}(p(n)q^2(n)/(\delta\epsilon))$.*

*Proof.* Please observe that $\tilde{F}_n$ is a *function* from $\mathbb{S}[N]$ to $\mathbb{S}[L]$ which, since $F_n$ is computable by a decision tree of depth $q(n)$, can be computed itself by a decision tree of the same depth. The result follows by just looking at the function $\tilde{F}$ and applying to it Theorem 21. Actually, the queries that algorithm $\mathbf{A}$ makes to $\tilde{F}_n$ correctly translate into queries of the form $f_i = f_{x_i}$. □

Remarkably, the strategy $\mathsf{infvar}_{\varepsilon,\delta}$ infers the "influential variables" of $\mathsf{f}$ *without* looking at how the latter queries its argument function, something which would anyway be available in the history of the interaction. This is reminiscent of *innocence* [23], a key concept in game semantics. We can now state the main result of this section.

**Theorem 26.** *For every $\delta$ there is a polytime probabilistic strategy $\mathsf{coll}_\delta$ on a game $!_s(LINSOF_{q,p}) \multimap (\mathbf{S}[\iota] \multimap \mathbf{B}) \otimes (\mathbf{S}[\iota] \multimap \mathbf{B})$ such that for every deterministic strategy $\mathsf{f}$ on $LINSOF_{q,p}$ computing $\{F_n\}_{n\in\mathbb{N}}$, the composition $(!_s\mathsf{f}); \mathsf{coll}_\delta$, with probability at least $1-\delta$, computes two function families $g, h$ with $g_n, h_n : \mathbb{S}[n] \to \mathbb{B}$ such that*

1. *$H(g_n, h_n) \geq 0.1$ for every $n$.*
2. *$F_n(g_n) = F_n(h_n)$ for every $n$.*
3. *For every function $f$ on which $\mathsf{coll}_\delta$ queries its argument, it holds that $H(f, g_n) \geq 0.1$ and $H(f, h_n) \geq 0.1$.*

*Proof.* The strategy $\mathsf{coll}_\delta$ can be easily built from $\mathsf{infvar}_{\varepsilon,\delta}$: the former calls the latter, and then draws two strings independently at random from $U_{k_n}$, where $k_n$ is the function the latter produces in output, and obtaining $x, y$. The two required outputs are thus $f_x$ and $f_y$, and have all the required properties. □

This shows that $\mathsf{coll}_\delta$ finds a *collision* for $F_n$ as a pair of functions that are different from each other (and in fact significantly different in Hamming distance) but for which $F_n$ outputs the same value, and hence $F$ cannot be a collision-resistant hash function. Moreover, because the functions are far from those queried, this means that $F_n$ cannot be a secure message authentication code either, since by querying $F_n$ on $g_n$, the adversary can predict the value of the tag on $h_n$.

*Proof.* We run the algorithm of Theorem 25 and obtain a partial function $g$ such that $|S(g)| \leq 10Lq^2/(\delta^2\epsilon)$ and such that the variance of every one of the functions $F_1, F_2, \ldots, F_L$ is smaller than $\delta\epsilon$. We choose $N$ to be large enough so that the bound $10Lq^2/(\delta^2\epsilon)$ on $|S(g)|$ is smaller than $\delta N/100$. Once we achieve this, we sample $x'$ and $x''$ independently from $U_g$ and set $g = f_{x'}$ and $h = f_{x''}$. Since we fixed only a $0.01\delta$ fraction of the coordinates, and $x'$ and $x''$ are random over the remaining ones, using the standard Chernoff bounds with high probability $g$ and $h$ will differ on at least $1 - 2^{-L} - \delta/10$ fraction of the remaining coordinates from each other and also from all other previous queries. (Specifically, the probability of the difference being smaller than this is exponentially small in $\delta N$ and we can make $N$ big enough so that this is much smaller than $\delta/(Lm)$ and so we can take a union bound on all queries.) On the other hand, because of the variance, the probability that $F_i(g) \neq F_i(h)$ for every $i$ is less than $\delta/L$ and so we can take a union bound over all $L$ $i$'s to conclude that $G(g) = G(h)$ with probability at least $1 - \delta$. □

### 4.3 A Positive Result on Higher-Order Pseudorandomness

We conclude this paper by giving a positive result. More specifically, we prove that pseudorandomness can indeed be attained at second order, but at a high price, namely by switching to the type $LOGSOF_p = SOF_{\iota,\lfloor \lg \rfloor,p}$. This indeed has the same structure of $LINSOF_{q,p}$, but the argument function takes in input strings of *logarithmic size* rather than linear size. Moreover, the argument function can be accessed a linear number of times, which is enough to query it on *every possible* coordinate.

The fact that a strategy on $LOGSOF_p$ can query its argument on every possible coordinate renders the attacks described in the previous section unfeasible. Actually, $LOGSOF_p$ can be seen as an *interactive* variation of the game $\mathbf{S}[\iota] \multimap \mathbf{S}[p]$, for which pseudorandomness is well known to be attainable starting from standard cryptographic assumptions [19]: simply, instead of taking in input *the whole* string *at once*, it queries it *bit by bit*, in a certain order. A *random strategy* of that type, then, would be one that, using the notation from Figure 4,

- Given $t_1, \ldots, t_i \in \mathbb{B}$, returns a string $s_{i+1}$ uniformly chosen at random from $\mathbb{S}[r(n)] - \{s_1, \ldots, s_i\}$, this for every $i < q(n)$.
- Moreover, based on $t_1, \ldots, t_{r(n)}$, it produces a string $v$ chosen uniformly at random from $\mathbb{S}[p(n)]$.

Please notice that this random strategy can be considered as a *random* functional in $(\mathbb{S}[r(n)] \to \mathbb{B}) \to \mathbb{S}[p(n)]$ only if $r(n)$ is logarithmic, because this way the final result $v$ is allowed to depend on the value of the input function in *all possible coordinates*. The process of generating such a random strategy uniformly can be seen[1] as a probabilistic strategy $\mathsf{randsof}$. We are now ready to formally define pseudorandom functions:

**Definition 27** (Second-Order Pseudorandom Function)**.** *A deterministic polytime strategy* $\mathsf{f}$ *on* $\mathbf{S}[\iota] \multimap LOGSOF_p$ *is said to be* pseudorandom *iff for every probabilistic polytime strategy* $\mathcal{A}$ *on* $!_s LOGSOF_p \multimap \mathbf{B}$ *there is a negligible function* $\varepsilon : \mathbb{N} \to \mathbb{R}_{[0,1]}$ *such that*

$$|\Pr(!_s(\mathsf{mult}_\iota; \mathsf{f}); \mathcal{A} \Downarrow^n 1) - \Pr(!_s\mathsf{randsof}; \mathcal{A} \Downarrow^n 1)| \leq \varepsilon(n)$$

The way we build a pseudorandom function consists in constructing a *deterministic* polytime strategy $\mathsf{fo2so}$ for the game $!_{\iota+1}(\mathbf{L}[\iota] \multimap \mathbf{S}[w]) \multimap LOGSOF_p$, where $w \in \mathbf{Pol}$ is such that $w \geq \lfloor \lg \rfloor$ and $w \geq p$. The strategy is represented in Figure 6. The function $\alpha$ interprets its first argument (a string in $\mathbb{S}[w(n)]$), as an element of $\mathbb{S}[\lfloor \lg \rfloor(n)]$ distinct from those it takes as second argument, and distributing the probabilities uniformly. The function $\beta$, instead, possibly discards some bits of the input and produces a possibly shorter string.

The way the strategy $\mathsf{fo2so}$ is defined makes the composition $\mathsf{f}; \mathsf{fo2so}$ statistically very close to the random strategy whenever $\mathsf{f}$ is chosen uniformly at random among the strategies for the parametric game $\mathbf{L}[\iota] \multimap \mathbf{S}[w]$. This allows us to prove the following:

**Theorem 28.** *Let* $F : \{0,1\}^n \times \{0,1\}^{\leq n} \to \{0,1\}^{w(n)}$ *be a pseudorandom function and let* $\mathsf{f}_F$ *be the deterministic polytime strategy for the game* $\mathbf{S}[\iota] \multimap !_{\iota+1}(\mathbf{L}[\iota] \multimap \mathbf{S}[w])$ *obtained from* $F$*. Then,* $\mathsf{f}_F; \mathsf{fo2so}$ *is second-order pseudorandom.*

# 5 Related Work

Game semantics and the geometry of interaction are among the best-studied program semantic frameworks (see, e.g. [17, 2, 23]), and can also be seen as computational models, given their operational flavor. This is particularly apparent in the work on abstract machines [10, 14], but also in the so-called geometry of synthesis [15]. In this paper, we are particularly interested in the latter use of game semantics, and take it as the underlying computational model. The definition of

[1] the strategy at hand would, strictly speaking, need exponentially many random bits, which are not allowed in our model; this could be accomodated without any major problem.

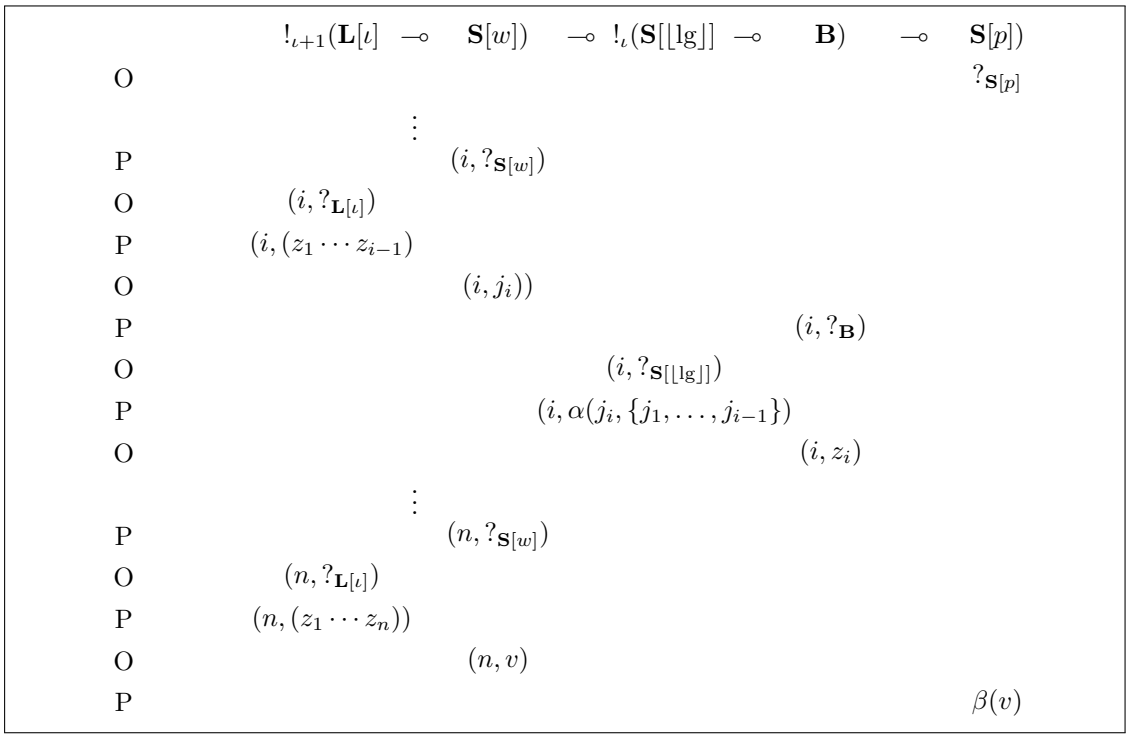



Figure 6: Plays in $\overline{\mathsf{fo2so}}$

our game model has been strongly inspired by works by Hyland [22] and Wolverson [32], the main novelties being parametrization and the bounded exponential construction, which together allows us to account for efficient randomized higher-order computations of the kinds used in cryptography. As a consequence, our definition of an acceptable strategy is more permissive than the ones from so-called AJM games [2] and HO games [23], the former being history-free, the latter essentially relying on so-called justification pointers.

This is certainly not the first paper in which cryptography is generalized to computational models beyond the one of first-order functions. One should of course cite Canetti's universally composable security [7], but also Mitchell et al.'s framework, the latter based on process algebras [28]. None of them deals with security properties of higher-order functions, though. A precursor of the aforementioned work [27] deals with first-order probabilistic polynomial time by way of oracles in an higher-order calculus, but lacks any claim about how probabilistic polynomial time would look like for genuinely higher-order functions.

Various ways to generalize the so-called formal model [12] to higher-order computation have been proposed. As an example, this is what Sumii and Pierce [30] do with their system of logical relations, which is shown to guarantee a form of non-interference. Similarly for Bhargavan et al.'s type system [4] in which cryptographic primitives are seen as libraries for the language F#. All this is is however fundamentally different from what we do here, namely extending the so-called *computational* model to higher-order computation: randomized behaviours and time-bounds are abstracted away.

## Acknowledgments

We thank Juspreet Singh Sandhu for helpful discussions during the early stages of this project. The first author is supported by NSF awards CCF 1565264 and CNS 1618026 and by a Simons Investigator Fellowship. The third author is supported by the ERC CoG 818616 "DIAPASoN", and by the ANR grants 19CE480014 "PPS" and 16CE250011 "REPAS".

# References


[1] Samson Abramsky and Radha Jagadeesan. Games and full completeness for multiplicative linear logic. *J. Symb. Log.*, 59(2):543–574, 1994.

[2] Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Inf. Comput.*, 163(2):409–470, 2000.

[3] Boaz Barak, Raphaëlle Crubillé, and Ugo Dal Lago. On higher-order cryptography (long version). *CoRR*, abs/2002.07218, 2020.

[4] Karthikeyan Bhargavan, Cédric Fournet, and Andrew D. Gordon. Modular verification of security protocol code by typing. In *Proc. of POPL 2010*, pages 445–456, 2010.

[5] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, 1984.

[6] Aloïs Brunel, Marco Gaboardi, Damiano Mazza, and Steve Zdancewic. A core quantitative coeffect calculus. In *Proc. of ESOP 2014*, pages 351–370, 2014.

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

[8] Pierre Clairambault and Hugo Paquet. Fully abstract models of the probabilistic lambda-calculus. In *Proc. of CSL 2018*, pages 16:1–16:17, 2018.

[9] Stephen A. Cook and Bruce M. Kapron. Characterizations of the basic feasible functionals of finite type. In *Proc. of FOCS 1989*, pages 154–159, 1989.

[10] Pierre-Louis Curien and Hugo Herbelin. Abstract machines for dialogue games. *CoRR*, abs/0706.2544, 2007.

[11] Vincent Danos and Russell Harmer. Probabilistic game semantics. *ACM Trans. Comput. Log.*, 3(3):359–382, 2002.

[12] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Trans. Inf. Theory*, 29(2):198–207, 1983.

[13] Hugo Férée. Game semantics approach to higher-order complexity. *J. Comput. Syst. Sci.*, 87:1–15, 2017.

[14] Olle Fredriksson and Dan R. Ghica. Abstract machines for game semantics, revisited. In *Proc. of LICS 2013*, pages 560–569, 2013.

[15] Dan R. Ghica. Geometry of synthesis: a structured approach to VLSI design. In *Proc. of POPL 2007*, pages 363–375, 2007.

[16] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

[17] Jean-Yves Girard. Geometry of interaction 1: Interpretation of system F. *Logic Colloquium 88*, 1989.

[18] Jean-Yves Girard, Andre Scedrov, and Philip J. Scott. Bounded linear logic: A modular approach to polynomial-time computability. *Theor. Comput. Sci.*, 97(1):1–66, 1992.

[19] Oded Goldreich. *Foundations of Cryptography: Volume 1.* Cambridge University Press, 2006.

[20] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications.* Cambridge University Press, 2009.

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

[22] Martin Hyland. Game semantics. In Andy Pitts and Peter Dybjer, editors, *Semantics and Logics of Computation*. Cambridge University Press, 1997.

[23] Martin Hyland and Luke Ong. On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 163(2):285–408, 2000.

[24] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2007.

[25] Akitoshi Kawamura and Stephen A. Cook. Complexity theory for operators in analysis. *TOCT*, 4(2):5:1–5:24, 2012.

[26] John Longley and Dag Normann. *Higher-Order Computability*. Theory and Applications of Computability. Springer, 2015.

[27] John C. Mitchell, Mark Mitchell, and Andre Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. of FOCS 1998*, pages 725–733, 1998.

[28] John C. Mitchell, Ajith Ramanathan, Andre Scedrov, and Vanessa Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theor. Comput. Sci.*, 353(1-3):118–164, 2006.

[29] Ryan O'Donnell, Michael E. Saks, Oded Schramm, and Rocco A. Servedio. Every decision tree has an in.uential variable. In *Proc. of FOCS 2005*, pages 31–39, 2005.

[30] Eijiro Sumii and Benjamin C. Pierce. Logical relations for encryption. *Journal of Computer Security*, 11(4):521–554, 2003.

[31] Klaus Weihrauch. *Computable Analysis: An Introduction*. Springer Publishing Company, Incorporated, 2013.

[32] Nicholas Wolverson. *Game semantics for an object-oriented language*. PhD thesis, University of Edinburgh, UK, 2009.