Towards Synthetic Cryptography
A String Diagrammatic Manifesto

Martti Karvonen, Robin Piedeleu, Fabio Zanasi

6th March 2026

1 Introduction

Security proofs in game-based cryptography are typically organised as hybrid arguments: one defines a sequence of games and shows that each consecutive pair is indistinguishable to any efficient adversary. A construction is considered secure if it cannot be distinguished from an ideal system whose security is evident or assumed (typically via a complexity-theoretic argument).

In practice, such proofs combine algebraic manipulations, probabilistic reasoning, and careful bookkeeping across multiple rounds of execution and oracle queries. While this style has proven effective, it obscures the compositional structure underlying many cryptographic arguments. Most proof steps are inherently *local*: a component is replaced by another, a bad event is ruled out, or an algebraic identity is applied. Yet these local transformations are embedded in large global game descriptions, making proofs difficult to verify, reuse, or mechanise. As cryptographic systems grow in scale and complexity, this gap between the modular structure of constructions and the global structure of proofs becomes increasingly problematic. If trust is to scale, cryptographic proofs must scale with it.

At the same time, cryptographic constructions themselves are already described compositionally. A Feistel network, for instance, is obtained by wiring together copies of a round pseudorandom function with XOR gates and swaps. Cryptographers routinely depict such constructions using circuit-like diagrams that illustrate how information flows between components. These diagrams provide valuable intuition about composition and dataflow, but they currently play only an informal role in security proofs: they serve as explanatory tools rather than objects of mathematical reasoning.

1.1 Aims

This paper investigates a structural reformulation of game-based cryptography. We treat the circuit-style diagrammatic notation already used informally by cryptographers as a formal language, equipped with a compositional semantics capturing its intended computational behaviour. In this setting, security proofs can be understood as equational reasoning over diagrammatic

representations of cryptographic constructions, making the compositional structure of security arguments explicit. This perspective brings several conceptual and practical benefits, which we highlight below.

A visual language for reasoning about systems. Our approach is based on *string diagrams*, a visual language originating in category theory. Their graphical nature makes the flow of information and resources in a system immediately visible while suppressing unnecessary implementation details. This makes them accessible to a broad range of users and particularly effective for communicating complex constructions and proofs. String diagrams are also *resource sensitive*: operations such as copying and discarding data must be represented explicitly. This feature is especially important in probabilistic computation, where random variables interact subtly with duplication and erasure of information. The explanatory power of string diagrams has been demonstrated across many areas of science, including probabilistic programming, machine learning, and quantum computation.

Making compositional structure explicit. Our primary goal is not merely to reformulate existing proofs, but to expose the compositional structure underlying security arguments. In our framework, replacing a component inside a construction corresponds to a local diagrammatic rewrite. Security proofs therefore become sequences of principled transformations rather than ad hoc manipulations of global games. This makes the locality already present in hybrid arguments explicit at the level of the proof language itself. Category theory provides a natural setting for this perspective because composition is treated as a primitive notion. In particular, compositionality is built into the semantics by design. This makes the framework well suited to reasoning about cryptographic systems, where proofs fundamentally rely on replacing components within larger constructions while preserving security properties.

Separating syntax from semantics. The categorical approach also cleanly separates syntax from semantics. The same diagrammatic language describing constructions and proofs can be interpreted in different computational models—probabilistic, quantum, or others—without modifying the syntax. This portability may facilitate the transfer of security arguments across computational substrates.

An axiomatic account of security arguments. Our approach is deliberately axiomatic. Rather than working directly with low-level probabilistic calculations, we identify the abstract principles used in security arguments and express them as algebraic properties of diagrammatic components. For example, pseudorandomness and collision bounds appear as axioms governing specific generators. Once these principles are isolated, security proofs can be carried out by purely diagrammatic reasoning.

Toward a substrate-independent foundation for cryptography. Because reasoning takes place at the level of diagrammatic structure rather than model-specific calculations, the framework is inherently extensible. Different computational substrates can be accommodated by changing the semantic interpretation while keeping the proof language unchanged. More broadly, we view our work as a step toward a substrate-independent foundation for trust. By isolating structural proof principles from model-specific assumptions, security arguments can arise as instances of algebraic reasoning in a generic diagrammatic language. In such a setting, the same methodology could in principle be applied beyond classical cryptography, including distributed protocols, hardware security primitives, quantum cryptography, and other domains where security properties must be established in a verifiable manner, while recognising that the relevant cryptographic primitives and their interrelationships may depend on the underlying computational model (for instance, in the quantum vs. classical case). The results presented here provide a concrete step toward that goal.

1.2 Methodology

Our approach is based on a variety of tools in categorical modelling.

String Diagrammatic Syntax To formalise cryptographic constructions, we use *string diagrams*, a graphical language originating in category theory that generalises the circuit diagrams already familiar to cryptographers. In this language, components are represented as boxes connected by wires, and diagrams compose in two basic ways: sequentially (connecting outputs to inputs) and in parallel (placing components side by side). This syntax captures the flow of information in a cryptographic construction while supporting rigorous algebraic reasoning about composition. Within this framework, cryptographic constructions are diagrams built from generators representing primitives such as randomness sources, Boolean operations, and pseudorandom functions, using sequential and parallel composition.

Categorical Semantics Like a standard syntax, our diagrams are given a formal semantics in a computational model, based on the (symmetric monoidal) category of efficiently computable substochastic maps. To model interaction and stateful behaviour, we extend this semantics to a category of streams, representing sequences of oracle queries made by an adversary. Our treatment builds on recent categorical treatments of stateful and stream-based computation [DLGR$^+$23, DLDFR25], adapted here to the setting of modern cryptography.

Induction and Coinduction Two notions of equivalence arise naturally in this framework. The first is equality of streams, which is defined without reference to a resource-bounded observer. The second is computational indistinguishability, defined relative to polynomial-time adversaries. These notions require different proof principles. Equality admits a coinductive characterisation based on bisimulation, reasoning step-by-step through the interaction. Com-

putational indistinguishability instead depends on the number of oracle queries made by the adversary, and is established by induction on that number.

Case Study: 3-round Feistel To demonstrate the approach, we analyse a central case study: the security of the 3-round Feistel construction, originally proved by Luby and Rackoff [LR88]. This example is sufficiently rich to exhibit the essential features of modern cryptographic reasoning. In our framework, the proof decomposes into a sequence of local diagrammatic transformations corresponding to familiar cryptographic principles, including pseudorandomness, the birthday bound, and basic algebraic properties of XOR. Our argument follows the structure of the proof presented by Rosulek [Ros26], but recasts its reasoning in a compositional diagrammatic form.

Scope and style. This document is intended as a programmatic report rather than a fully polished research article. Our aim is to articulate a categorical perspective on game-based cryptography using string diagrams. This is work in progress: we develop the core ideas in sufficient detail to demonstrate the feasibility of our programme on a central example, but we do not aim at maximal generality. In keeping with the exploratory style of this report, we leave open several problems, formulate conjectures and leave several proofs as future work.

Related work. The formal verification of cryptographic constructions has seen substantial progress over the past two decades. Systems such as EasyCrypt [BGLZB11], CryptoVerif [Bla06], and frameworks developed in Coq [BGZB09] and F* [ZBPB17] provide expressive logics and proof environments that have been used to mechanise sophisticated game-based proofs in the computational model. These systems typically interpret games as programs and apply techniques from formal verification to establish security properties.

Among these, EasyCrypt and its predecessor CertiCrypt [BGZB09] are particularly close in spirit to our work. They rely on probabilistic relational Hoare logics (pRHL), which support structured reasoning about program equivalence and computational indistinguishability. Security proofs are expressed as sequences of program transformations that reduce the security of a construction to the assumed hardness of an underlying computational problem.

In parallel, tools such as ProVerif [Bla16] and Tamarin [BCDS17] enable automated analysis of cryptographic protocols in the symbolic (Dolev-Yao) model [DY81]. These approaches focus on protocol-level reasoning and offer powerful automation, but operate in a different abstraction from the computational model considered here.

Our work is complementary to these approaches. Rather than introducing a new program logic or verification environment, we aim to identify the algebraic structure underlying game-based security arguments. The goal is a more synthetic account of these proofs, in which the compositional principles guiding hybrid arguments become explicit and amenable to algebraic reasoning. In doing so, we were inspired by the interesting albeit less formal blog post by Lúcás Meier [Mei23].

Outline. The remainder of the report is structured as follows. In Section 2, we recall the Feistel construction and its classical security argument, which serves as our motivating example. Section 3 introduces string diagrams as a formal syntax for cryptographic constructions, identifying the basic generators corresponding to randomness, structural wirings, Boolean operations, and more. Section 4 develops the semantic model, beginning with efficiently computable substochastic maps and extending to a category of streams to capture interactive and stateful behaviour. Here we distinguish two notions of equivalence arising in this setting (equality of streams and computational indistinguishability). In Section 5 we introduce the axioms and proof principles we have identified, prove some intermediate results, and apply the framework to the 3-round Feistel construction, presenting its security proof as a sequence of diagrammatic transformations. We conclude by discussing open problems and future challenges in Section 6. Some proofs and other auxiliary results are relegated to the appendix.

Acknowledgments. We thank Mike Rosulek for many insightful discussions on the topics of this report and for the valuable suggestions on the diagrammatic notation adopted here. This research was supported by ARIA's *Scaling Trust* programme.

2 Motivating Example

Many constructions in cryptography rely heavily on pseudorandom functions (PRFs) and transforming them into other cryptographic primitives. A canonical example of such a construction is the Feistel cipher, which turns a pseudorandom function into a pseudorandom permutation (PRP). PRPs are key building blocks in a number of ciphers and symmetric encryption schemes, where the inverse F^{-1} of some PRP F provides a way to decrypt data encrypted using F.

The proof that the Feistel cipher produces a secure PRP, due to Luby and Rackoff [LR88] is a classic, but challenging, cryptographic proof. It serves as a good test case for our categorical approach, as it is simple enough to describe succinctly, yet its security proof is non-trivial and already exhibits the essential features of modern cryptographic reasoning.

Feistel cipher. Consider a family f of functions $f_\lambda \colon \mathbb{B}^\lambda \times \mathbb{B}^\lambda \to \mathbb{B}^\lambda$ indexed by the security parameter $\lambda \in \mathbb{N}$. We say that f is pseudorandom if it is computationally indistinguishable from a truly random function R, given by a family of lazy random dictionaries $R_\lambda \colon \mathbb{B}^\lambda \to \mathbb{B}^\lambda$ for each $\lambda \in \mathbb{N}$. In other words, no polytime (in λ) adversary with oracle access to f can distinguish it from R with more than negligible advantage.

The aim of the Feistel cipher is to construct a PRP from f, that is, a PRF that admits an inverse. One round of a Feistel cipher operates as follows: given the pair of inputs (L_i, R_i) and the subkey K_i, the output of the current round is given by

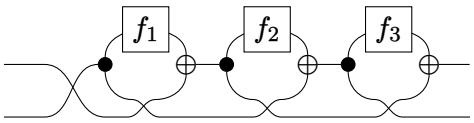
$$L_{i+1} = R_i \quad \text{and} \quad R_{i+1} = L_i \oplus f(R_i, K_i)$$

where $\oplus$ denotes bitwise XOR. Here, we focus on the 3-round Feistel cipher F, which outputs (R_3, L_3). First, $F\colon\ \mathbb{B}^\lambda \times \mathbb{B}^\lambda \to \mathbb{B}^\lambda \times \mathbb{B}^\lambda$ is a permutation, with inverse F^{-1} given by computing the following operations in reverse order for $i = 3, 2, 1$,

$$R_i =: L_{i+1} \quad \text{and} \quad L_i = R_{i+1} \oplus f(L_{i+1}, K_i)$$

It is more difficult to show that F is also pseudorandom. Luby and Rackoff's original proof considers Feistel networks in which each round function is chosen independently and uniformly at random. In this context, they prove that A 3-round Feistel network with independent random round functions is a PRP. The proof is information-theoretic: the adversary is modelled as an oracle algorithm making at most q queries, and the distinguishing advantage for an efficient adversary is bounded explicitly by a term of order $O(q^2/2^\lambda)$ The argument proceeds by a careful analysis of oracle queries and identifying certain *bad events* – collisions among intermediate values for different queries – whose probability is bounded via combinatorial counting arguments.

Often, authors will depict the 3-round Feistel using a circuit-like representation:



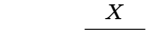
In this report, we will give this notation a formal status using string diagrams and will endow it with a formal semantics that captures its intended behaviour. Moreover, we will be able to reason directly with these diagrams to show the security of the 3-round Feistel (Theorem 2).

Luby and Rackoff also show that a *4-round* Feistel network is a *strong* PRP, that is, it is secure when the adversary is given oracle access to the permutation *and* its inverse. We leave the extension of our work to this proof for future work.

3 Syntax

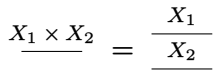
We now describe the formal syntax used to represent cryptographic constructions. The syntax is diagrammatic: constructions are expressed as string diagrams built from basic components and composed by wiring their interfaces together or by placing them side-by-side. Although we ultimately interpret these diagrams in a symmetric monoidal category, we begin with an elementary description that requires no categorical background.

Boxes and wires. String diagrams are depicted as graphs made of boxes and wires. Here, wires represent channels that carry values of a specified type. In the present setting, types correspond to (finite) sets such as bitstrings of some fixed length, or tuples thereof. We write

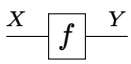


to represent a wire carrying values of type X. In categorical terms, a type is called an *object*, and a plain wire represents the identity id_X over that object, *i.e.*, a channel that simply takes in an input and forwards it to its output, doing nothing.

Multiple wires are treated as ordered tuples: placing two wires in parallel corresponds to taking a product of types. Thus a pair of wires of types X_1 and X_2 is regarded as a single interface of type $X_1 \times X_2$:



Between two objects X and Y, we can have a *morphism* $f\colon X \to Y$, which we depict as a box:



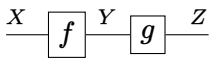
This is intended to represent a (possibly probabilistic, possibly stateful) procedure that consumes inputs of type X and produces outputs of type Y. Morphisms $s\colon 1 \to X$, whose input type is the singleton set $1 = \{\bullet\}$ are represented as boxes with only an output wire:

These can be thought of a processes that produce outputs without the need for any input. A distribution over X is an example of such a process: it outputs some element $x \in X$ with a certain probability. Conversely, morphisms $e\colon X \to 1$ are depicted as boxes with no output wire:

In our case, such a diagram represents a procedure which accepts its input with a certain probability, placing a constraint on its input. The simplest example is a procedure that accepts all inputs and discards them. Another example is a stateful program that fails if it sees the same input twice.

Composition. There are two fundamental ways to compose diagrams.

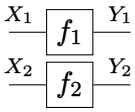
Sequential: given two morphisms $f\colon X \to Y$ and $g\colon Y \to Z$, we can form the morphism $f\,;g\colon X \to Z$ which we depict by chaining the two corresponding diagrams horizontally:



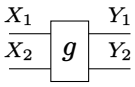
Note that the type of the intermediate wires have to match. This can be thought of as passing the outputs of the first procedure as inputs to the second.

Parallel: given two morphisms $f_1\colon X_1 \to Y_1$ and $f_2\colon X_2 \to Y_2$, we can form their monoidal product $f_1 \otimes f_2\colon X_1 \times X_2 \to Y_1 \times Y_2$, which we depict by juxtaposing the two corresponding

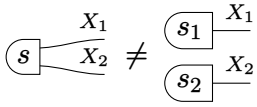
diagrams vertically:[1]



Intuitively, these represent two procedures that operate independently, in parallel. Note that not all diagrams on multiple wires decompose as the monoidal product of multiple diagrams; a generic diagram $g\colon X_1 \times X_2 \to Y_1 \times Y_2$ is represented as box with two input and output wires:

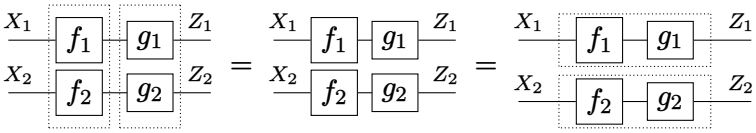


The reader will be familiar with this idea for ordinary functions: a map $X_1 \times X_2 \to Y_1 \times Y_2$ decompose into a monoidal product only if the ith output depends only on the ith input for $i = 1, 2$. In a general SMC, diagrams $s\colon 1 \to X_1 \times X_2$ may not even decompose into the monoidal product of two sub-diagrams:



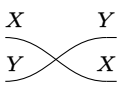
For example, when diagrams represent operations that involve randomness, s may produces values that are not independent; $s_1 \otimes s_2$ on the other hand produces independent outputs.

The diagrammatic notation absorbs equivalences that are not obvious with a symbolic syntax. For example, $(f_1 \otimes f_2)\,;\,(g_1 \otimes g_2)$ and $(f_1\,;\,g_1) \otimes (f_2\,;\,g_2)$ are depicted in the same way:



Similarly, all the axioms of monoidal categories, recalled in Figure 1, thereby become diagrammatic tautologies.

Symmetry and wire crossings. In a *symmetric* monoidal category, we are also allowed to cross an X-wire with a Y-wire using a special morphism σ_X^Y, which we depict as follows:



This can be thought of a special procedure that swaps its two inputs before returning them as outputs. Crucially, these diagrams obey equational axioms guaranteeing that they behave

[1] We use the convention that the composition of two diagrams is depicted horizontally, from left to right, and the monoidal product vertically, from top to bottom. Other authors prefer to use a convention where composition is vertical from top to bottom or bottom to top, and the monoidal product horizontal, from left to right.

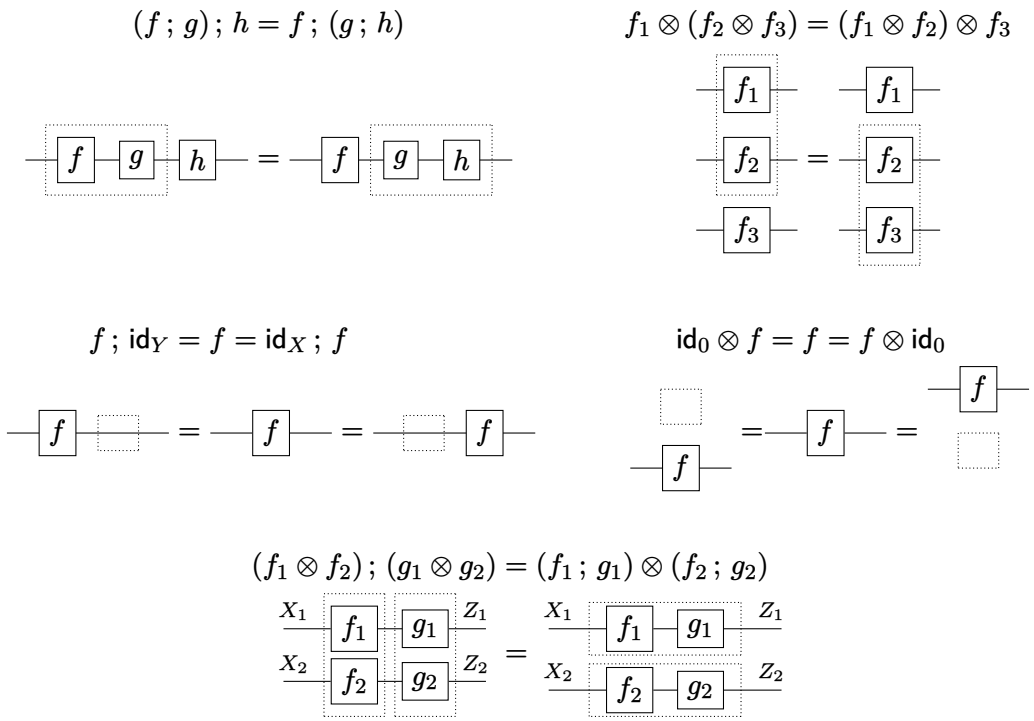


Figure 1: Axioms of monoidal categories and their diagrammatic counterpart.

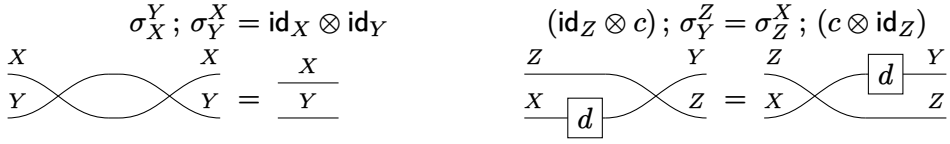
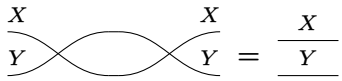
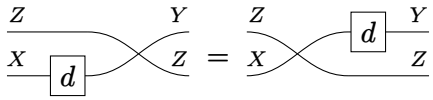


Figure 2: Additional axioms for symmetric monoidal categories and their diagrammatic counterpart.

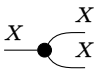
as you would expect from their depiction as wire-crossings. In particular, two wire-crossings composed sequentially are equivalent to (uncrossed) parallel wires:



In addition, diagrams can be pulled through wire-crossings in ways that their depiction as string diagrams make obvious:



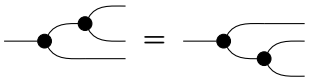
Copying and deleting. For cryptographic applications, we work in a richer setting than the plain symmetric monoidal case, with additional diagrams available, encoding additional structure. The first is a diagram $\Delta_X \colon X \to X \times X$, depicted as a black dot,



and representing an operation which takes in a value, copies it and dispatches it to its two output wires, much like in a standard circuit notation. The second is a diagram $\delta_X \colon X \to 1$, also depicted as a black dot,

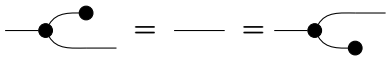
and representing an operation that discards its input (and outputs nothing). Together, these operations satisfy the *cocommutative comonoid axioms*, stating that

- copying the first copy is equivalent to copying the second copy, *i.e.* it produces three identical copies:



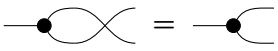
(coassociativity)

- copying and discarding a copy amounts to doing nothing:



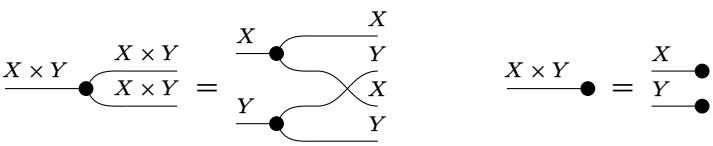
(counitality)

- the two copies are identical so we can swap them:



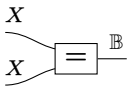
(cocommutativity)

We also require copying and deleting to be compatible with the monoidal product in the following sense:

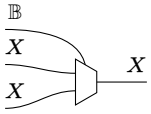


It is helpful to think of a diagram as a structured program whose dataflow is implicit in the wiring. Sequential composition corresponds to function composition; parallel composition corresponds to forming tuples and evaluating components independently; copying and deleting are here to manage variables. We now introduce more complex operations.

Boolean operations. Our case study operates on bitstrings so we will use freely standard Boolean operations, such as $\oplus$ (XOR), $\wedge$ (AND), etc. We will also make extensive use of the equality operation for a given type X, depicted as



which simply outputs the Boolean `true` if its two inputs are equal and `false` otherwise. In addition, we will need an if-then-else (aka multiplexer) gate, depicted as

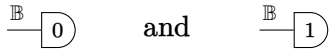


which outputs its second input if the first (Boolean) input is set to `true`, or its third input if the first input is set to `false`.

In what follows, we will use common algebraic identities involving Boolean operations freely, as axiomatisations of these operations already exist and are not our focus.

Probabilistic operations. Cryptography relies heavily on randomness. For this reason, we will use a number of diagrams that manipulate probability distributions. The first is a

representing an operation which outputs uniform samples from X. The other two are



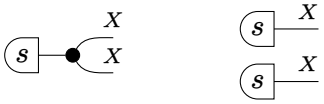
which constrain their inputs to be `false` and `true`, respectively. These can be understood as programs that check whether their input have the specified value, allow the computation to continue when this is the case, and fail otherwise.

We have studied the interplay of these probabilistic primitives with standard Boolean circuit operations extensively in a previous paper [PTSZ25], and will rely freely on this work here.

Interlude: why string diagrams? A central motivation for our use of string diagrams is the observation that cryptographic constructions involve probabilistic operations, and that these form monoidal categories [Fri20]. These categories provide a setting that generalises standard

algebra in which stochastic maps can be composed: as we saw, sequential composition models the chaining of probabilistic operations (for example, sampling then transforming), while monoidal (parallel) composition represents independent processes. This categorical structure captures the essential compositionality of probabilistic computation, as evidenced by the wealth of recent work on categorical probability theory [Fri20, FGP21, Per23, FGPR23].

Crucially, in the probabilistic setting, operations like copying and discarding values are not freely available. Intuitively, this makes sense: if f represents some probabilistic process (such as flipping a coin), running f once and reusing its outcome in two different places is different from running f twice (flipping a coin two times). In diagrammatic terms, the following two diagrams are not necessarily equal:



Similarly, the possibility of failure implies that discarding the output of some operation f does not mean that we could have dispensed with f in the first place. In diagrammatic terms, the following two diagrams are not necessarily equal:

This fundamental distinction means that the semantics of cryptographic constructions cannot be captured by conventional algebraic syntax, which assume that variables can be duplicated or ignored at will. In categorical terms, stochastic maps do not form *Cartesian* monoidal categories: the monoidal product is not the categorical product, unlike in, say, the category of sets and functions. Attempting to model this behaviour in a symbolic syntax instead requires significantly more complex machinery – bound variables, let-bindings, α-equivalence, and the delicate management of linear versus non-linear substitution – all features which complicate formal reasoning and mechanisation.

By contrast, string diagrams provide a natural and rigorous two-dimensional syntax for reasoning about such processes. Diagrams eliminates the need for bound variables and substitution altogether. Notably, structural features such as conditional independence, marginalisation, and factorisation can be expressed and manipulated directly in the diagrammatic language [FGHL$^+$23], making it particularly intuitive for human use.

Remark 1. *The description of string diagrams can be made fully-precise by describing them as equivalence classes of terms built from generators by sequential and parallel composition. Fixing a collection of generating operations with specified input and output types, the diagrammatic language described above may then be defined inductively as the free symmetric monoidal category*[2] *generated by these operations.*

Concretely, one defines: objects as formal tensor products of basic types, morphisms as terms built inductively from generators using sequential composition and monoidal product, and equations imposed only by the axioms of a SMCs (Figures 1 and 2).

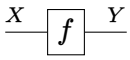
[2]Technically, the free *strict* SMC generated by these operations.

In this presentation, string diagrams are not merely suggestive pictures but representatives of equivalence classes of such terms. The graphical calculus is then sound and complete for reasoning in the free SMC.

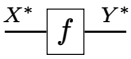
This formal viewpoint allows us to interpret diagrams in some target semantics using a structure-preserving mapping, i.e.*, a symmetric monoidal functor defined inductively on generators and extended to all diagrams using the composition and the monoidal product in the target SMC.*

For the purposes of exposition, however, we continue to reason directly with diagrams, relying on their well-established correspondence with the underlying inductive syntax.

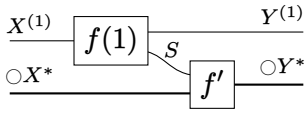
Stateful operations and streams. In line with the computational model, we want to think of our diagrams as components in a protocol that can be called on a number of times by an efficient adversary. For this purpose we will distinguish diagrams with two different kinds of wires in our syntax: diagram with thin wires, representing an operation which can only be used once,



and thick wires, representing operations which can be reused as many times as the adversary queries them,



When we define the semantics rigorously, operations of this type will denote a stream and the objects X^* and Y^* a sequence of objects. One way to understand a stream f is as the composition of a *head*, $f(1)$, specifying the operation of the stream at the first timestep, and of another stream *tail*, f', specifying the remainder of f:

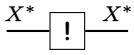


Intuitively, the notation $X^{(1)}$ will indicate a type X of values which can be used at the first timestep of the stream, while $\bigcirc X^*$ denotes a sequence of Xs, like X^*, but shifted by one timestep. The middle wire of type S can be understood as a form of state passed from the head of the stream to its tail. In what follows, we will often specify the behaviour of certain operations using this decomposition of a stream over $X^* \to Y^*$ into a first operation $X^{(1)} \to Y^{(1)} \times S$ followed by a stream $S \times \bigcirc X^* \to \bigcirc Y^*$.

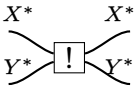
All of the primitives we have mentioned so far can be promoted in this way to reusable operations with thick wires that can be called arbitrarily many times.

So far, however, all of our diagrammatic primitives represent operations that do not keep a state across different rounds of execution. To truly take advantage of the stream semantics, and to reflect cryptographic practice, we will also use stateful operations. We mention two that play a central role in the rest of this report.

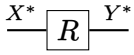
- First, we have



  which constrains all its inputs to be distinct. In other words, we can think of it as a program that keeps its previous inputs in memory and fails if its current input is the same as one it has already seen. This operation will be particularly useful when reasoning about pseudorandomness. Note that we have it for all input types and will make extensive use of it for pairs of inputs, depicting it as follows:

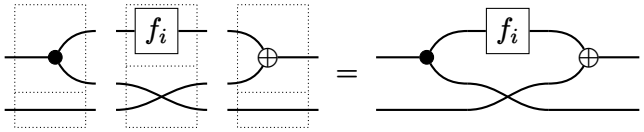


- Second, we have

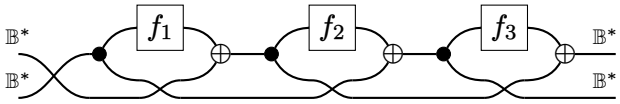


  which behaves like a (lazy) random dictionary: for every new input $x \in X$, it samples some output in Y uniformly and stores the pair (x, y); if it receives x again as input, it outputs the same y. This is also known as a *random oracle* and will serve as an ideal model of a (pseudo)random function in what follows.

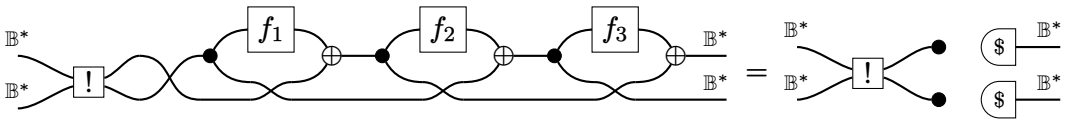
Feistel cipher as a string diagram. Using these operations, we can see how the circuit for the Feistel cipher is built as a string diagram. First, every round is of the form $r_i := (\Delta\mathbb{B}^* \otimes \mathsf{id}_{\mathbb{B}^*})\,;\,(f_i \otimes \sigma^{\mathbb{B}^*}_{\mathbb{B}^*})\,;\,((\oplus) \otimes \mathsf{id}_{\mathbb{B}^*})$, for some pseudorandom function f_i. The string diagram already conveys more directly the function of each round:



Then, with three round function f_1, f_2, f_3, we can form the 3-round Feistel cipher by sequentially composing these rounds (and adding a symmetry/wire-crossing in front to account for the extra-swap of the inputs), $\sigma^{\mathbb{B}^*}_{\mathbb{B}^*}\,;\,r_1\,;\,r_2\,;\,r_3$, or



Security for this construction typically means that it is pseudorandom, *i.e.*, indistinguishable from a random function to efficient adversaries. One way to restate this property is that the Feistel cipher is indistinguishable from true randomness *on distinct inputs*. Diagrammatically, we can express this requirement as follows:



(1)

Notice that the !-box remains on the rhs as it places a constraint on the kind of inputs that we allow (they have to be distinct). Our aim below is to:

- define a symmetric monoidal category in which to interpret the diagrams in (1), *cf.* Section 4;
- identify a manageable set of axioms that we can use to derive (1) by reasoning equationally at the level of the diagrams themselves, *cf.* Section 5.

4 Semantics

We now explain the kind of operations our diagrams are intended to represent and organise these into a symmetric monoidal category, as required to use string diagrams.

4.1 A categorical semantics for cryptographic constructions

Cryptographic constructions and proofs make fundamental use of randomness and it is well-known that various kinds of maps which incorporate randomness can be organised into symmetric monoidal categories. We recall here briefly how this is done, as this will constitute the basis on which we will build progressively more complex semantics, able to accommodate the cryptographic constructions we care about here.

Substochastic maps. In this report, a *subdistribution* over a set X will refer to a finitely-supported map $\varphi\colon X \to [0,1]$ such that $\sum_{x\in X} \varphi(x) \leq 1$. A *distribution* is then a subdistribution φ for which $\sum_{x\in X} \varphi(x) = 1$. We will write (sub)distributions as formal sums $\sum_x \varphi(x)\,|x\rangle$, omitting the elements of X for which $\varphi(x) = 0$, and call $\mathcal{D}X$ (resp. $\mathcal{D}_{\leq 1}X$) the set of all distributions (resp. subdistributions) over some finite set X.

A *substochastic map* $f\colon X \to Y$ is an ordinary map $X \to \mathcal{D}_{\leq 1}Y$. Currying the Y component, we can think of f as a map $f(-|-)\colon Y \times X \to [0,1]$. As with distributions, a substochastic map is *stochastic* when $\sum_y f(y|x) = 1$ for all $x \in X$. Notice that any map $f\colon X \to Y$ can be promoted to a stochastic map $X \to Y$ given by $x \mapsto |f(x)\rangle$ for all $x \in X$.

We can also interpret a substochastic map $f\colon X \to Y$ as specifying a *conditional* subdistribution: $f(-|x)$ gives a subdistribution over Y for each $x \in X$. The composition $f\,;g$ of two substochastic maps $f : X \to Y, g\colon Y \to Z$ is given by summing over the intermediate variable as follows:

$$(f\,;g)(z|x) = \sum_{y\in Y} g(z|y)f(y|x)$$

The identity substochastic map $\mathsf{id}_X\colon X \to X$ is simply the Dirac delta $\delta_x\colon x \mapsto |x\rangle$. Substochastic maps with these operations define a category, which we call $\mathsf{Stoch}_{\leq 1}$ [MP22].

For two subdistributions $\varphi \in \mathcal{D}_{\leq 1}X$ and $\rho \in \mathcal{D}_{\leq 1}Y$, we can form the product subdistribution $\varphi \otimes \rho \in \mathcal{D}_{\leq 1}(X \times Y)$, given by $(\varphi \otimes \rho)(x,y) = \varphi(x) \cdot \rho(y)$. We will write $|xy\rangle = |x\rangle \otimes |y\rangle$ so

that $(\sum_x \varphi(x)\,|x\rangle) \otimes (\sum_y \rho(y)\,|y\rangle) = \sum_{x,y} \varphi(x)\rho(y)\,|xy\rangle$. The same operation can be extended to conditional distributions, that is, to substochastic maps $f_1 \colon X_1 \to Y_1$ and $f_2 \colon X_2 \to Y_2$, giving $(f_1 \otimes f_2)(x_1, x_2) = f_1(x_1) \otimes f_2(x_2)$. This makes $\mathsf{Stoch}_{\leq 1}$ into a *symmetric monoidal* category, with the Cartesian product of sets as monoidal product, the singleton set $1 = \{\bullet\}$ as unit, and the symmetry $\sigma^X_Y : X \times Y \to Y \times X$ given by $\sigma^X_Y(x, y) = |yx\rangle$. Stochastic maps out of the unit 1, *e.g.* $1 \to X$, correspond precisely to distributions over X.

Note that $\mathcal{D}(1) = 1$ and therefore, there is only one stochastic map $\epsilon_X \colon X \to 1$ for any set X, given by $\epsilon_X(x) = |\bullet\rangle$. Moreover, there is a canonical diagonal substochastic map inherited from the Cartesian product of sets: $\Delta_X \colon X \to X \times X$ given by $\Delta_X(x) = |xx\rangle := |x\rangle \otimes |x\rangle$. It is important to note that $f\,;\Delta_Y \neq \Delta_X\,;(f \times f)$ in general—we say that arbitrary substochastic maps cannot be copied. We have hinted at this earlier: sampling twice from a distribution produces two independent outcomes; it is not equivalent to sampling once and using the result twice. Stochastic maps that do satisfy $f\,;\Delta_Y = \Delta_X\,;(f \times f)$ are precisely the *deterministic* maps, that is, maps for which $f(x) = |y\rangle$ for a single $y \in Y$.

Similarly, not every substochastic map $f \colon X \to Y$ satisfies $f\,;\epsilon_Y = \epsilon_X$. Those that do are precisely the *stochastic* maps, since $(f\,;\epsilon_Y)(x) = \sum_{y \in X} f(y|x) = 1 = \epsilon_X(x)$. Such maps are often called *discardable* or *causal* [Fri20, CJ19].

Efficiently-computable substochastic maps. In cryptography, we care about operations that can be performed *efficiently*, that is, with polynomially bounded resources. To follow this requirement, we will work with efficiently-computable (sub)stochastic maps: sequences of substochastic maps in some parameter λ, which we call the *security parameter*, and which can be implemented by a probabilistic Turing machine running in time polynomial in λ [BK23, Section 7].

Definition 1. *Let* $\mathsf{EffStoch}_{\leq 1}$ *be the category whose objects* $X = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ *are sequences of finite sets and morphisms* $f \colon X \to Y$ *are efficiently-computable substochastic maps* $\{f_\lambda \colon X_\lambda \to Y_\lambda\}_{\lambda \in \mathbb{N}}$*, the composition of compatible morphisms* $f; g$ *is computed componentwise* $f_\lambda; g_\lambda$ *in* $\mathsf{Stoch}_{\leq 1}$*, and identities* $\mathsf{id}_X \colon X \to X$ *are the obvious sequences of identities* $X_\lambda \to X_\lambda$*.*

This category can be equipped with the structure of a symmetric monoidal category (SMC) with monoidal product given componentwise in $\mathsf{Stoch}_{\leq 1}$*,* i.e.*, for each value of the security parameter* λ*, by the cartesian product of sets on objects and the tensor product of substochastic maps on morphisms [BK23, Section 7.1]. We write* 1 *for the sequence of singleton sets which constitutes the monoidal unit for this SMC, and* σ^Y_X *for the symmetry* $X \times Y \to Y \times X$ *lifted componentwise from the symmetry in* $\mathsf{Stoch}_{\leq 1}$*.*

Note moreover that $\mathsf{EffStoch}_{\leq 1}$ *inherits the structure of a copy-discard (or gs-monoidal) category from* $\mathsf{Stoch}_{\leq 1}$*, with structural maps* $\Delta_X \colon X \to X \times X$ *(copy) and* $\delta_X \colon X \to 1$ *(discard).*

Remark 2. *To be more rigorous, following the work of Broadbent and Karvonen [BK23], we should define objects themselves as* efficient sequences of finite sets*: sequences of injections* $\{\langle - \rangle_n \colon X_\lambda \to \{0,1\}^\star\}\}_{\lambda \in \mathbb{N}}$ *where* $\{0,1\}^\star$ *denotes set of words over* $\{0,1\}$*, each* X_λ *is a finite*

set and the characteristic function of $\langle X_\lambda \rangle_\lambda \subseteq X_\lambda$ is computable in polynomial time. These serve as encodings of sets in bitstrings in order to define polytime computability more rigorously.

Then, efficiently-computable substochastic maps $f\colon X \to Y$ are those sequence of substochastic maps for which the sequence defined by $\langle f_\lambda \rangle(\langle x \rangle) = \langle y \rangle$ can be computed in polynomial time. In this context, the monoidal product is defined relative to an efficient bijective pairing function $< -, - >\colon \{0,1\}^\star \times \{0,1\}^\star \to \{0,1\}^\star$.

In what follows most of our types are bitstrings, equipped with the obvious injection into $\{0,1\}^\star$. Thus, to avoid complicating notation unnecessarily, we leave out encodings $\langle - \rangle$ from our definitions.

Stateful maps. Most cryptographic protocols unfold over multiple rounds of interactions between the different parties that may maintain some state. Moreover, adversaries can call components adaptively during the course of the protocol. For this reason, we move from (efficiently-computable) substochastic maps to *stateful* substochastic maps that maintain a memory.

We use the state-construction from the work of Di Lavore et al [DLGR$^+$23] to construct a SMC of *stateful*, efficiently-computable substochastic maps.

Definition 2. *A stateful map of type $X \to Y$ is the data of a sequence of (finite) sets S, an efficiently-computable substochastic map of type $X \times S \to Y \times S$ and an element of S (that is, for each S_λ), which we call its initial state.*

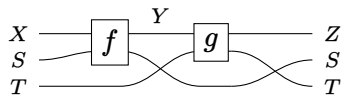
Stateful maps form a SMC $\mathsf{St}(\mathsf{EffStoch}_{\leq 1})$ [DLGR$^+$23, Definition 3.9] with composition given by

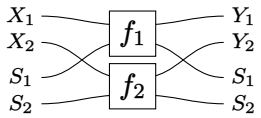
$$(f \times \mathsf{id}_T)\ ;\ \left(\mathsf{id}_Y \times \sigma_S^T\right)\ ;\ (g \times \mathsf{id}_S)\ ;\ \left(\mathsf{id}_Z \times \sigma_T^S\right)$$

and monoidal product given by

$$\left(\mathsf{id}_{X_1} \times \sigma_{X_2}^{S_1} \times \mathsf{id}_{S_2}\right)\ ;\ (f_1 \times f_2)\ ;\ \left(\mathsf{id}_{Y_1} \times \sigma_{S_1}^{Y_2} \times \mathsf{id}_{S_2}\right)$$

As string diagrams in the SMC $\mathsf{EffStoch}_{\leq 1}$, we can visualise composition and monoidal product of stateful maps as follows:





Intuitively, every stateful map f with state space S and initial state s_0 generates a monoidal stream whose head is given by (partially) applying f to the initial state and whose tail is given by applying f thereafter.

Streams. Stateful maps are too intentional: they keep track of information that is invisible to an external agent able to feed it inputs and observe its outputs. Eventually, we want to identify streams that are indistinguishable to an (efficient) adversary. A useful intermediary step is to drop the requirement that adversaries be efficient and think of stateful maps as *monoidal streams*, the definition of which we adapt below for our purposes [DLDFR25].

Definition 3. *[DLDFR25, Definition 7.1] A* stream $f\colon X \to Y$ *is given by a sequence of objects of* $\mathsf{EffStoch}_{\leq 1}$ $\{S(n)\}_{n\in\mathbb{N}}$ *with* $S(0) = 1$*, called the* memory *of* f*, and a sequence of efficiently-computable substochastic maps*[3]

$$\{f(n)\colon X(n) \times S(n-1) \to Y(n) \times S(n)\}_{n\in\mathbb{N},n\geq 1}$$

quotiented by dinaturality in $S(n)$*,* i.e.*, by the equivalence relation* $f \sim g$ *generated by the existence of an efficiently-computable substochastic map* $r\colon T(0) \to S(0)$ *such that*

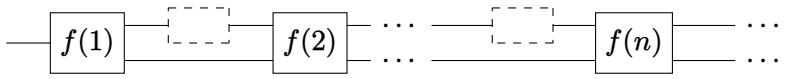
i) $f(1) = g(1)\,;\,(\mathsf{id}_Y \times r)$*,*

ii) $(\mathsf{id}_X \times r) \cdot f' \sim g'$*,*

where $f'(n) = f(n+1)$ *and* $h \cdot f$ *is given by* $(h \cdot f)(1) = h\,;\,f(1)$ *and* $(h \cdot f)(n) = f(n)$ *for* $n > 1$*.*

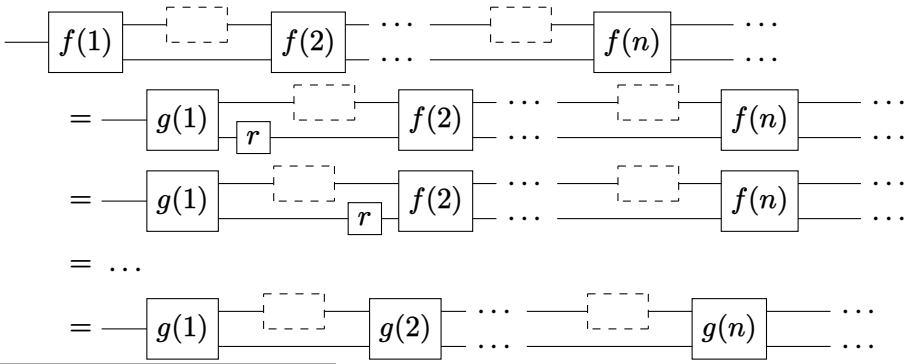
Streams can be organised into a SMC Stream*, whose composition, identities, monoidal product, unit, and symmetries are all computed component-wise as in* $\mathsf{St}(\mathsf{EffStoch}_{\leq 1})$ *[DLDFR25, Theorem 7.12].*

Abusing string diagrams in $\mathsf{EffStoch}_{\leq 1}$, we can visualise streams as diagrams with holes (dashed boxes) extending infinitely to the right:



(2)

Here, the holes indicates where the environment (*e.g.* an adversary, another stream...) may interact with the stream in between two timesteps. This notation also clarifies the meaning behind conditions *i)* and *ii)* in Definition 3: if $f \sim g$ then there exists r such that



[3]Note that efficiently-computable substochastic maps are themselves sequences in the security parameter λ, so we are effectively working with sequences of sequences here. The two layer of indices play different roles and should not be confused, however: the λ allows us to measure and bound the amount of computational resources available, while the second index n represents the number of queries or call to a given stateful component.

Efficient streams. Streams are too general, as they allow for arbitrarily complex changes of types and behaviour between different components of the same stream. In line with the requirement that all parties of a cryptographic protocol only access limited computational resources (*i.e.*, perform operations that are computable in polynomial time in the security parameter), we often prefer to work with streams that are generated by efficiently-computable stateful maps, in the following sense.

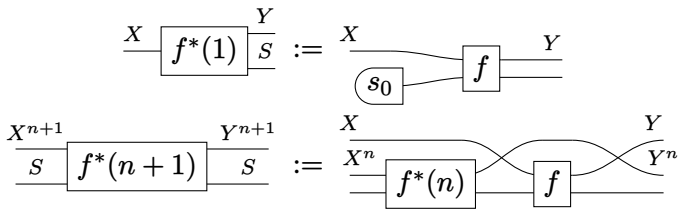
Definition 4. *Given some object of* $\mathsf{EffStoch}_{\leq 1}$*, let* X^* *be the sequence defined by* $X^*(n) = X$ *for all* $n > 0$*.*

The stream $(S, f, s_0)^* \colon X^* \to Y^*$ generated by a stateful map (S, f, s_0) *is the sequence* $\{f(1) \colon X \to Y \times S\} \cup \{f^*(n) \colon X^n \times S \to Y^n \times S\}_{n>1}$ *defined inductively by*

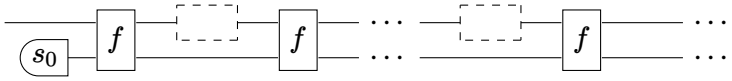
$$f^*(1) := (\mathsf{id}_X \times s_0) \,;\, f$$
$$f^*(n+1) := f^*(n) \,;\, \left(\sigma_X^{Y^n} \times \mathsf{id}_S\right) \,;\, (\mathsf{id}_{Y^n} \times f) \,;\, \left(\sigma_{Y^n}^{Y} \times \mathsf{id}_S\right)$$

We call streams generated by a stateful map efficient*.*

We can picture the definition above using string diagrams in $\mathsf{EffStoch}_{\leq 1}$:



Using the intuitive notation of (2) for streams, we can visualise the stream generated by (S, f, s_0) as follows:



.

Proposition 1. *Given two stateful maps* $(S, f, s_0) \colon X \to Y$ *and* $(T, g, t_0) \colon X \to Z$*, let*

$$\left((S, f, s_0) \,;\, (T, g, t_0)\right)^* = (S, f, s_0)^* \,;\, (T, g, t_0)^*$$

Moreover, given two stateful maps $(S_1, f_1, s_1) \colon X_1 \to Y_1$ *and* $(S_2, f_2, s_2) \colon X_2 \to Y_2$*,*

$$\left((S_1, f_1, s_1) \times (S_2, f_2, s_2)\right)^* = (S_1, f_1, s_1)^* \times (S_2, f_2, s_2)^*$$

Corollary 1. *Efficient streams form a sub-SMC* $\mathsf{EffStream}$ *of the SMC* Stream*.*

We will also need to see any ordinary (non-stateful) efficiently-computable substochastic map $f \colon X \to Y$ as stream $f^{(k)}$ with trivial memory whose only non-trivial component is the kth one. In particular, $f^{(1)}$ acts like a component in a protocol that can only be called once as f at the first timestep and is not available thereafter.

Definition 5. *For some $k \in \mathbb{N}$ and object X, let $X^{(k)}$ be the sequence given by $X^{(k)}(n) = X$ for $n \leq k$ and $X^{(k)}(n) = 1$ for $n > k$.*

Moreover, for some efficiently-computable substochastic map $f\colon X \to Y$, let $f^{(k)}\colon X^{(k)}(n) \to Y^{(k)}(n)$ be the stream with memory 1, $f^{(k)}(n) = f$ for $n \leq k$ and for which all other components are $f^{(k)}(n) = \mathsf{id}_1\colon 1 \to 1$ for $n > k$.

We also use the notation $\bigcirc X$ for the sequence of objects given by $\bigcirc X(1) = 1$ and $\bigcirc X(n) = X(n-1)$ for $n > 1$. Similarly, let $\bigcirc f$ is the stream given by $\bigcirc f(1) = \mathsf{id}_1$ and $\bigcirc f(n) = f(n-1)$ for $n > 1$.

Finally, given some stream $f\colon X \to Y$, let $f^{\leq k}\colon X^{(k)} \to Y^{(k)}$ be the stream given by $f^{\leq k}(n) = f(n)$ for $n \leq k$ and $f^{\leq k}(n) = \mathsf{id}_1$ for $n > k$. In other words, $f^{\leq k}$ only allows k calls to f or truncates it after the kth timestep. Note that for an efficient stream generated by the stateful map (S, f, s_0), we simply write $(S, f, s_0)^{\leq k}$ for the stream $((S, f, s_0)^*)^{\leq k}$.

As we said before, streams are too general for our purpose and we will only consider those that are generated by some efficient stateful and non-stateful substochastic maps, as given by the last two definitions. Characterising precisely the subcategory of Stream generated by these is an interesting open problem.

Open Problem 1. *What is the right category of efficiently-computable streams?*

Conjecture 1. *The right category of efficiently-computable streams are those with a finite prefix whose components are all given by efficiently-computable substochastic maps and whose tail is generated by some stateful efficiently-computable substochastic map.*

Coinduction. Working with streams allows us to reason on our semantics using the principle of *coinduction*. In particular, we will use this principle here to show that two streams f and g are equal by exhibiting some binary relation R – called a *bisimulation* – that contains the pair (f, g) and such that, if $(s, t) \in R$ then $s(1) = t(1)$ and $(s', t') \in R$. When this is the case, we have shown that R is a subset of the equality relation on streams and therefore that $f = g$. In practice, when reasoning with the diagrammatic syntax, we will not need to specify the bisimulation R explicitly; instead, the bisimulation will appear as a *coinductive hypothesis* to which we appeal in the middle of an equational derivation. At that point, coinductive reasoning may feel strange, as if we are missing a base case, but it is a fully rigorous way of reasoning about streams [JR11]. This proof method is particularly useful when the streams that we manipulate are themselves defined coinductively, *i.e.*, by specifying their *head* $f(1)$ and their *tail* f'. This is useful, because, as we will see, some of the key components in our motivating example can be defined in this way. See Section 5.2 for more details and Section 5.3 for examples of this form of reasoning with diagrams.

However, coinduction is only useful to reason about equality of streams; in cryptography, we often care about a coarser form of equivalence, defined in reference to adversaries with bounded computational resources, as we now explain.

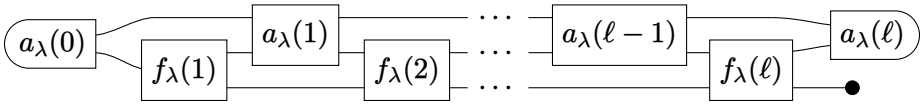
Computational indistinguishability. Game-based cryptography relies on the notion of an efficient adversary whose aim is to distinguish two protocols. We formalise this notion in our setting.

Definition 6. *A* distinguisher $a[-]$ *of type* $X \to Y$ *is a sequence of efficient substochastic maps* $\{a_\lambda(i)\colon A_\lambda(i-1) \times Y_\lambda(i-1) \to A_\lambda(i) \times X_\lambda(i)\}_{\lambda\in\mathbb{N}, 1\leq i\leq \ell-1}$ *(notice the swapping of domain and codomain), an initial state* $a_\lambda(0)\colon 1 \to A_\lambda(0)\times X_\lambda(0)$ *and a final predicate* $a_\lambda^{(\ell)}\colon A_\lambda(\ell)\times Y_\lambda(\ell) \to 1$*, where* $\ell = \ell_\lambda$ *(as a function of* λ*) is bounded by a polynomial in* λ*.*

Definition 7. *Given a stream* $f\colon X \to Y$ *and a distinguisher* $a[-]$ *of the same type, let their* interaction $a[f]$ *be the sequence* $\{a[f]_\lambda\}_{\lambda\in\mathbb{N}}$ *given by:*

$$(a_\lambda(0) \times s_{0,\lambda}); (\mathsf{id}_{A_\lambda(0)} \times f_\lambda); (a_\lambda(1) \times \mathsf{id}_{S_\lambda}); \dots; (\mathsf{id}_{A_\lambda(\ell-1)} \times f_\lambda); (a_\lambda(\ell) \times \delta_{S_\lambda})$$

Using string diagrams in $\mathsf{Stoch}_{\leq 1}$, $a[f]_\lambda : 1 \to 1$ can be depicted as



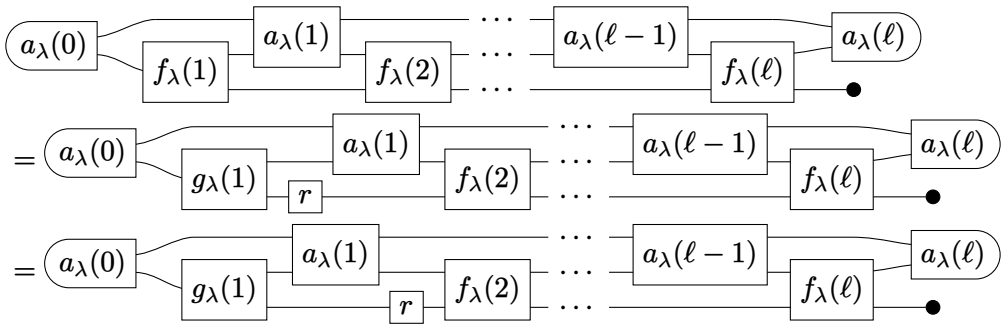
Note that this is a *scalar*, that is, a diagram of type $1 \to 1$ where $1 = \{\bullet\}$ is the unit of the monoidal structure in $\mathsf{Stoch}_{\leq 1}$. In this SMC, scalars are simply real numbers in $[0, 1]$ (a subprobability), so $a[f]$ gives a real number for each $\lambda \in \mathbb{N}$. Specifically, the interaction of f with $a[-]$ at λ is given by the subprobability of the composite in which the distinguisher $a[-]$ calls f ℓ_λ-many times.

To see that $a[f]$ is well-defined, consider g such that $f \sim g$. Then, there exists $r\colon T(0) \to S(0)$ such that

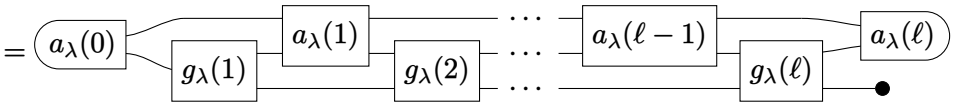
i) $f(1) = g(1)\,;\, (\mathsf{id}_Y \times r)$,

ii) $(\mathsf{id}_X \times r) \cdot f' \sim g'$,

Recall that $f'(n) = f(n+1)$ and $h \cdot f$ is given by $(h \cdot f)(1) = h\,;\, f(1)$ and $(h \cdot f)(n) = f(n)$ for $n > 1$. Then, for every λ, we have



$= \ldots$



Recall that a function (of the security parameter λ) is said to be *negligible* if it is asymptotically less than the inverse of any polynomial.

Definition 8. *We say that two efficient streams $f, g\colon X \to Y$ are computationally* indistinguishable *if, for any distinguisher $a[-]$ of the same type, the function $|a[f] - a[g]|$ is negligible. In this case, we write $f \approx g$.*

Open Problem 2. *Can we find a coinductive characterisation of computational indistinguishability?*

Such a characterisation would help to extend coinductive reasoning to indistinguishable streams. It is likely that finding a coinductive definition of indistinguishability will require us to work in a *quantitative* setting in order to keep track of how much an efficient adversary is allowed to unfold a given stream. We will discuss potential concrete steps in this direction in Section 6. Alternatively, we could define a relation as a greatest fixpoint that 1) is included in $\approx$ and 2) contains the examples we care about. Once again, we leave this for future work.

Recall that a congruence is, roughly-speaking, an equivalence relation that is compatible with some algebraic operation. Thus, $\approx$ is congruence for composition of streams if, whenever $f \approx f'$ and $g \approx g'$ we also have $f\,;g \approx f'\,;g'$. Similarly, $\approx$ is a congruence for the monoidal product of streams if $f_i \approx f_i'$ implies $f_1 \otimes f_2 \approx f_1' \otimes f_2'$. Intuitively, these two properties will allow us to reason about computational indistinguishability compositionally (that is, locally on diagrams). The proof of the following theorem is in Appendix A.1.

Theorem 1. *Computational indistinguishability is a congruence for composition and the monoidal product in* Stream*. In other words, the structure of SMC lifts to streams* up to *the equivalence relation $\approx$.*

The following lemma is immediate and will be useful to reason on streams by *induction* (see Section 5.2)

Lemma 1 (Prefix equivalence)**.** *For any two streams $f\colon X \to Y$, if $f^{\leq n} \approx g^{\leq n}$ for all $n \in \mathbb{N}, n > 0$, then $f \approx g$.*

4.2 Interpreting diagrams as streams

To distinguish syntax and semantics more cleanly, we define a mapping $[\![-]\!]$ from diagrams to Stream, the SMC of streams. As explained in Remark 1, our diagrammatic language can be organised into a formal syntax, freely-generated from the generating operations covered in Section 3. In principle, then, the semantics may be defined formally as a symmetric monoidal

functor from the syntax to Stream, specified by its action on generators and extended inductively using the composition and monoidal product in Stream. The interested reader will find more details in introductory texts [PZ25].

In general, we use thick lines to represent the identity streams over objects which have infinitely many non-trivial components and thin lines for the identity stream over objects that have a single non-trivial component (usually at the first timestep). For example, the identity stream over the object $\mathbb{B}^*(n) = \{\mathbb{B}^\lambda\}_{\lambda \in \mathbb{N}}$ for every $n \in \mathbb{N}$ or, equivalently, the stream generated by the identity stateful map over $\mathbb{B}$, with trivial memory $1 = \{\bullet\}_{\lambda \in \mathbb{N}}$, is represented as follows:

$$\left[\!\!\left[\xrightarrow{\mathbb{B}^*} \right]\!\!\right] = \mathsf{id}_{\mathbb{B}^*} = (1, \mathsf{id}_{\mathbb{B}}, \bullet)^*$$

Similarly, the identity over the object $\bigcirc\mathbb{B}^*$ – the identity stream over $\mathbb{B}^*$ shifted by one timestep – is represented as a labelled thick line:

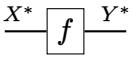
$$\left[\!\!\left[\xrightarrow{\bigcirc\mathbb{B}^*} \right]\!\!\right] = \mathsf{id}_{\bigcirc\mathbb{B}^*}$$

We use thin lines to represent the identity stream over $\mathbb{B}^{(1)}$, given by $\mathbb{B}^{(1)}(1) = \{\mathbb{B}^\lambda\}_{\lambda \in \mathbb{N}}$ and $\mathbb{B}^{(1)}(n) = 1$ for every $n > 1$:

$$\left[\!\!\left[\xrightarrow{\mathbb{B}^{(1)}} \right]\!\!\right] = \mathsf{id}_{\mathbb{B}^{(1)}}$$

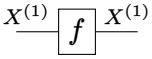
Note that $\mathbb{B}^{(1)} \times \bigcirc\mathbb{B}^* \cong \mathbb{B}^*$.

For any stateful map f, we draw the following diagram with thick wires, to denote its iteration $f^* \colon X^* \to Y^*$:



As it is clear from the type that this diagram represents the iteration f^*, we write f as the label for the box (instead of f^*). Similarly, we depict$\bigcirc\,[\![f]\!]$ and $f^{\leq n}$ for $n > 1$, respectively, as

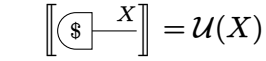
and $f^{\leq 1}$ with thin wires as



The reader familiar with string diagrams or category theory might find it strange that we give the same name to diagrams with different types; this is meant to suggest that these diagrams all represent the same stateful operation, potentially truncated, or shifted by some timesteps, etc., so that their interpretation is fully determined by the label f and the type of the wires.

In what follows, $\bot$ denotes the everywhere-zero subdistribution (for simplicity, we use the same notation for the everywhere-zero subdistribution over any supporting set), and $\delta(x)$ denotes the the Dirac delta at x, the probability distribution which is entirely supported at some element

x. We also write $\mathcal{U}(X)$ for the uniform distribution over some finite set X and represent it with a \$-box (with no input):

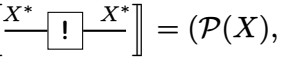


$$= \mathcal{U}(X)$$

We now define a stream which forces its inputs to all be distinct. It will be defined as the stream generated by the following stateful map, for any object X: let $!(-)\colon X \times \mathcal{P}(X) \to X \times \mathcal{P}(X)$ be the stateful map with $\emptyset$ as initial state, and given by

$$!(x, S) = \begin{cases} \delta(x, S \cup \{x\}) & \text{if } x \notin S \\ \bot & \text{otherwise} \end{cases}$$

Here $\mathcal{P}(X)$ acts as a memory of previous inputs; $!(x, S)$ then fails if x has already been seen; otherwise it returns x and adds it to the memory. Note that, despite the presence of the powerset $\mathcal{P}(X)$, this gives an *efficient* sequence of substochastic maps, since $!(-)$ adds at most one element to its memory at each interaction with the environment, resulting in a memory that grows linearly with each input (and the initial state is the empty set). Then, we can define

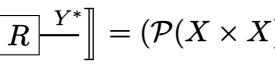


$$= (\mathcal{P}(X), !, \emptyset)^*$$

We now define a stream that behaves as a lazy random dictionary R: when a given new input R samples a new output uniformly and adds the pair to its memory; when given an input it has already seen, it returns the output chosen previously. Formally, let $R\colon X \times \mathcal{P}(X \times Y) \to Y \times \mathcal{P}(X \times Y)$ be the stateful map with $\emptyset$ as initial state for every $\lambda \in \mathbb{N}$, and given by

$$R(x, S) = \begin{cases} \delta(y, S) & \text{if there exists } y \text{ s.t. } (x, y) \in S \\ \mathcal{U}\{(y, S) \cup \{(x, y)\}) \mid y \in Y\} & \text{otherwise} \end{cases}$$

Here $\mathcal{P}(X \times Y)$ acts like a memory of previous input-output pairs. Note that $R(x, S)$ is well-defined: the y in the first case above is necessarily unique if it exists, since only one such y is sampled in the second case. Once again, this can be implemented efficiently, since the memory only grows linearly with each input (and we start from the empty set as initial state). Note that this can also be thought of as random function $X \to Y$ sampled uniformly among functions of this type. Then, we can define



$$= (\mathcal{P}(X \times X), R, \emptyset)^*$$

We can also define the monoidal streams generated by $!$ and R coinductively—these definitions will serve as axioms in the following subsection. For this we first need a few more auxiliary functions that we will use throughout.

As explained in Section 3, we will use standard Boolean operations, such as $\neg$, $\wedge$, etc. We will also use the same notation for their counterpart as stateful maps/monoidal streams, instead

of the more cumbersome $\neg^*, \wedge^*$, etc. The semantics of *if-then-else gate/multiplexer* for each type X, is given by:

$$\left[\!\!\left[\begin{array}{l} \mathbb{B} \\ X \\ X \end{array} \right\rangle\!\!\left. X \right]\!\!\right] (b,x,y) = \begin{cases} \delta(x) & \text{if } b = \texttt{true} \\ \delta(y) & \text{if } b = \texttt{false} \end{cases}$$

The semantics of the map $\{(-) = (-) \colon X \times X \to \mathbb{B}\}_{\lambda \in \mathbb{N}}$ for each type X encodes the equality of elements of type X:

$$\left[\!\!\left[\begin{array}{l} X \\ X \end{array} \right\rangle\!\!\left[= \right]\!\!\left. \mathbb{B} \right]\!\!\right] = \begin{cases} \delta(\texttt{true}) & \text{if } x = y \\ \delta(\texttt{false}) & \text{otherwise} \end{cases}$$

Let $\left\{ \left[\!\!\left[\mathbb{B} \,\middle|\, 1 \right]\!\!\right] : \mathbb{B} \to 1 \right\}_{\lambda \in \mathbb{N}}$ be the (memoryless) stateful map given by

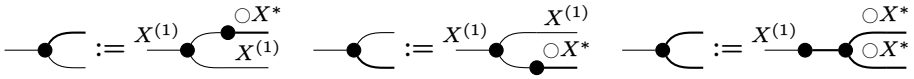
$$\left[\!\!\left[\mathbb{B} \,\middle|\, 1 \right]\!\!\right] (x) = \begin{cases} \delta(\bullet) & \text{if } x = \texttt{true} \\ \bot & \text{if } x = \texttt{false} \end{cases}$$

In other words, $\left[\!\!\left[\mathbb{B} \,\middle|\, 1 \right]\!\!\right]$ is the substochastic map that returns the only possible probability distribution over $1 = \{\bullet\}$ when its input is the Boolean true, and the everywhere-zero subdistribution otherwise. Note also that its domain is $\mathbb{B}$ independently of the value of the security parameter λ. Similarly, we can define $\left[\!\!\left[\mathbb{B} \,\middle|\, 0 \right]\!\!\right]$ as $\left[\!\!\left[\mathbb{B} \,\middle|\, 0 \right]\!\!\right] (x) = \left[\!\!\left[\mathbb{B} \,\middle|\, 1 \right]\!\!\right] (\neg x)$.

The following stream allows us to repeat a value at every position of a stream: it only takes a single input $x \in X$ at the first timestep (and only $\bullet \in 1$ at every subsequent step) and outputs this x at every timestep.

$$\left[\!\!\left[X^{(1)} \bullet \bigcirc X^* \right]\!\!\right] (n)(x,s) = \begin{cases} (x,x) & \text{if } n = 1 \\ (s,s) & \text{if } n > 1 \end{cases}$$

We will commonly combine this stream with copying and use the following syntactic sugar to simplify diagrams:



Finally, any composite diagram can be interpreted using the following rules for composition and monoidal product:

$$\left[\!\!\left[X \,\middle|\, f \,\middle|\, Y \,\middle|\, g \,\middle|\, Z \right]\!\!\right] = \left[\!\!\left[X \,\middle|\, f \,\middle|\, Y \right]\!\!\right] ; \left[\!\!\left[Y \,\middle|\, g \,\middle|\, Z \right]\!\!\right]$$

$$\left[\!\!\left[\begin{array}{l} X_1 \,|\, f_1 \,|\, Y_1 \\ X_2 \,|\, f_2 \,|\, Y_2 \end{array} \right]\!\!\right] = \left[\!\!\left[X_1 \,\middle|\, f_1 \,\middle|\, Y_1 \right]\!\!\right] \otimes \left[\!\!\left[X_2 \,\middle|\, f_2 \,\middle|\, Y_2 \right]\!\!\right]$$

where ';' and '$\otimes$' above denote composition and the monoidal product in the SMC of streams (Definition 3).

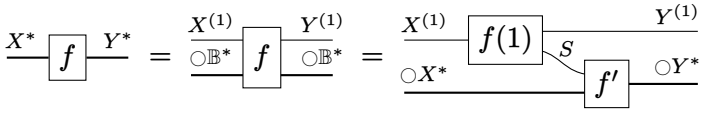
5 Diagrammatic Reasoning

5.1 Equational Theory

In cryptography, many statements concern objects which grow sufficiently fast or are sufficiently large, asymptotically—we define this notion formally below.

Definition 9. *We call an object $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ of* $\mathsf{EffStoch}_{\leq 1}$ sufficiently large *if the sequence $|X_\lambda|$ is asymptotically larger than any polynomial.*

For example, $\mathbb{B} = \{\mathbb{B}^\lambda\}$ is sufficiently large, and so is $\{X^\lambda\}$ for any set of cardinality $|X| > 1$. Note also that $X^{(1)} \times \bigcirc X^* \cong X^*$ for any object X. This equivalence allows us to characterise the behaviour of a given operation, *e.g.*, $f\colon X^* \to Y^*$, coinductively by specifying its head $f(1)\colon X^{(1)} \to Y^{(1)} \times S$ and its tail $f'\colon S \times \bigcirc X^* \to \bigcirc Y^*$. In other words, as explained in Section 3, we can see a stream $f\colon X^* \to Y^*$ as the composite of its head and tail:

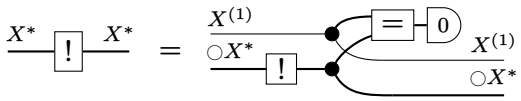


(f-unfold)

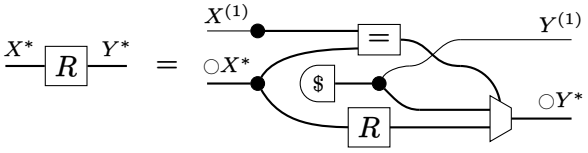
In diagrams, we will freely identify $X^{(1)} \times \bigcirc X^*$ and X^*, even though the two are not strictly equal, only isomorphic. To be more formal, we should introduce the isomorphism between $X^{(1)} \times \bigcirc X^*$ and X^* and its inverse explicitly. These are a well-known trick to deal with string diagrams for non-strict SMCs [WGZ22, Car22], but they would only play a very minor role in this report, not to mention increase the notational burden. For this reason, we omit them.

Axioms. As in our semantics, we use two different equivalence relations on diagrams: $=$ and $\approx$, capturing equality of streams and computational indistinguishability respectively. We will assume that any equality $f = g$ implies $f \approx g$ so that equality refines indistinguishability. Our axioms are the following:

1. any equality between substochastic maps,
2. for any X and Y,

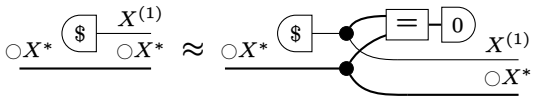


(!-unfold)



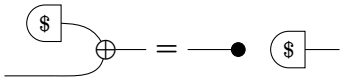
(R-unfold)

3. for any sufficiently large X,



(birthday bound)

Let us offer a few clarifying comments. For the first item, we do not give axioms for substochastic maps explicitly as this is not the focus of this work. The interested reader can find a complete axiomatisation of substochastic maps between sets of the form $\mathbb{B}^n$ in previous work by two of the authors [PTSZ25], or in a recent paper by DiGiorgio, Sobocinski, and Voorneveld [GSV26]. In what follows we will make extensive use of simple Boolean identities involving conjunction, if-then-else gates, `true`, `false`, the ability to delete certain causal maps, and other classic probabilistic equalities, such as the *one-time pad*:

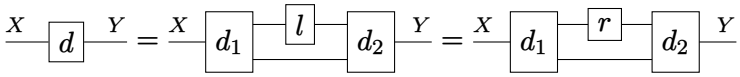


(one-time pad)

The second item lists two axioms that allow us to unfold `!` and the random dictionary[4] R. We think of these axioms as stream equations in the style of Rutten's stream calculus [Rut01], where the first component (on the top wires) defines the head and rest (on the bottom wires) defines the tail of the stream.

Finally, the last item is the only properly cryptographic axiom we use: it is a diagrammatic translation of the fact that, given some value x from some sufficiently large X, if we sample another value uniformly from X we are very unlikely (negligibly so, if the size of X grows faster than any polynomial) to sample x specifically. This last axiom can be seen as a key distinguishing feature of cryptography from plain probability theory and is where the existence of sufficiently-large sets and bounded computational resources comes into play.

Formally, $=$ and $\approx$ are the smallest congruences (with respect to composition and the monoidal product) that contain the axioms above. In practice, this means that we can reason equationally with diagrams as in a two-dimensional generalisation of standard algebra: to apply an equality $l = r$ (or $l \approx r$) in a diagram d, we need to identify the lhs l as a subdiagram within d and replace it with the rhs r:



However, basic equational reasoning is insufficient to handle more intricate security proofs, for which we may need to exploit specific properties of streams. For this reason, we will also reason using induction and coinduction, as we now explain.

[4]The unfolding axiom for the random dictionary R was inspired by a blog post [Mei23] in which several ideas related to this report also appear and for which its author also deserves credit.

5.2 (Co)inductive Proofs with Diagrams

In this work, we use two high-level proof methods. These could be phrased more formally as axiom schemes, but we prefer an intuitive high-level description of how to apply them, and will illustrate their use through examples in Section 5.3.

Coinduction. The first is coinductive and we use it to reason about equality of streams. Assume that we want to show that two streams f and g are equal. As explained in Section 4. To do so coinductively involves building a bisimulation relation R that contains the pairs of diagrams that we want to show equal, usually of the form

$$R = \left\{ \left(\overset{X^*}{\text{---}}\boxed{f}\overset{Y^*}{\text{---}}, \overset{X^*}{\text{---}}\boxed{g}\overset{Y^*}{\text{---}} \right), \left(\overset{\bigcirc X^*}{\text{---}}\boxed{f}\overset{\bigcirc Y^*}{\text{---}}, \overset{\bigcirc X^*}{\text{---}}\boxed{g}\overset{\bigcirc Y^*}{\text{---}} \right), \dots \right\}$$

Proving that R is indeed a bisimulation involves showing that, if $(f, g) \in R$ then $f(1) = g(1)$ and $(f', g') \in R$. However, in practice, we do not build a bisimulation explicitly, but use equational reasoning with what we call a *coinductive hypothesis*: we start from f, use our axioms to unfold the behaviour of f, and, under the assumption that $f' = g'$ (the coinductive hypothesis), derive g. For a concrete use of this principle, we invite the reader to jump to the proof of Lemma 2 below.

Induction. The second principle, induction, will undoubtedly be more familiar to the reader. We use it chiefly to establish that two streams are indistinguishable. At the level of diagrams, to show that $f \approx g$, we first prove

$$\overset{X^{(1)}}{\text{---}}\boxed{f}\overset{Y^{(1)}}{\text{---}} \;\approx\; \overset{X^{(1)}}{\text{---}}\boxed{g}\overset{Y^{(1)}}{\text{---}}$$

for the base case, and then we prove

$$\overset{X^{(n)}}{\text{---}}\boxed{f}\overset{Y^{(n)}}{\text{---}} \;\approx\; \overset{X^{(n)}}{\text{---}}\boxed{g}\overset{Y^{(n)}}{\text{---}} \quad\Rightarrow\quad \overset{X^{(n+1)}}{\text{---}}\boxed{f}\overset{Y^{(n+1)}}{\text{---}} \;\approx\; \overset{X^{(n+1)}}{\text{---}}\boxed{g}\overset{Y^{(n+1)}}{\text{---}}$$

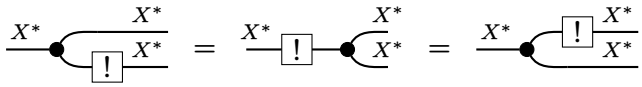
for the inductive case. Note that this shows that $f^{\leq n} \approx g^{\leq n}$ for all $n > 0$; by Lemma 1, this allows us to conclude that $f \approx g$ as we wanted. For a concrete use of induction, we invite the reader to jump to the proof of Lemma 3 below.

Remark 3. *Interestingly, our proofs by coinduction and induction below look very similar. This lends credence to the possibility of finding a common formal framework for both, possibly by developing a (quantitative) coinductive characterisation of computational indistinguishability,* cf. *Open Problem 2.*

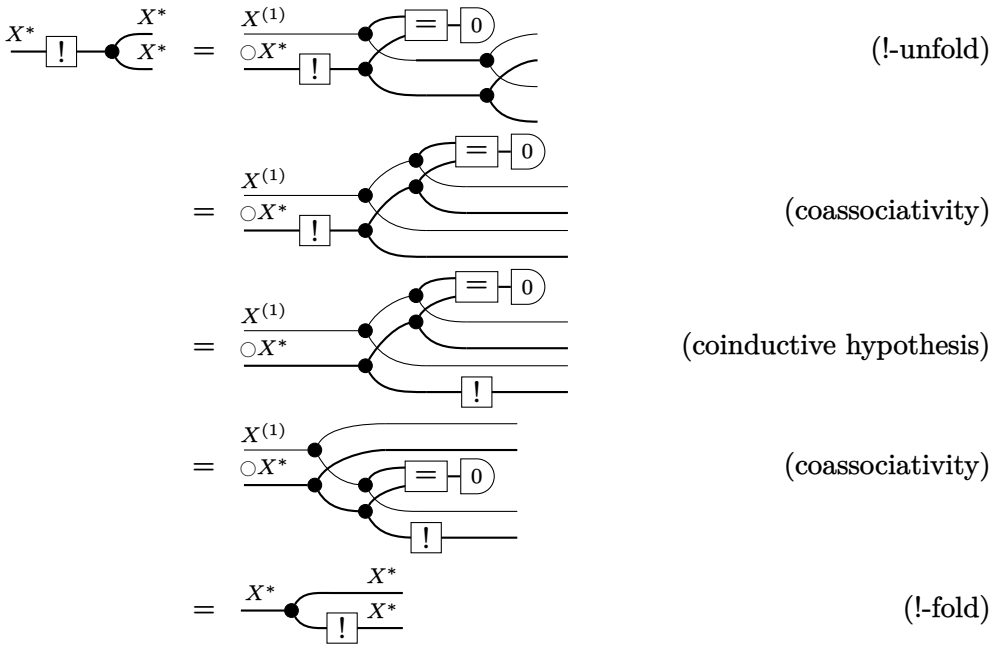
5.3 Auxiliary Results

Before turning back to our motivating example, we demonstrate how (co)inductive diagrammatic reasoning operates on some useful intermediary results.

The first is a simple lemma, stating that it does not matter which branch of a copy a !-box is placed—in all three possible choices, the constraint that all inputs be distinct is enforced in the same way. The proof of this statement proceeds by coinduction, following the pattern laid out in Section 5.2, and serves as a paradigmatic example of this form of reasoning here.

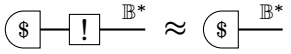
Lemma 2. 

Proof. By coinduction:

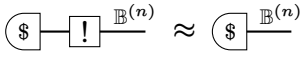


The second equality can be derived from the first one using the cocommutativity of  . □

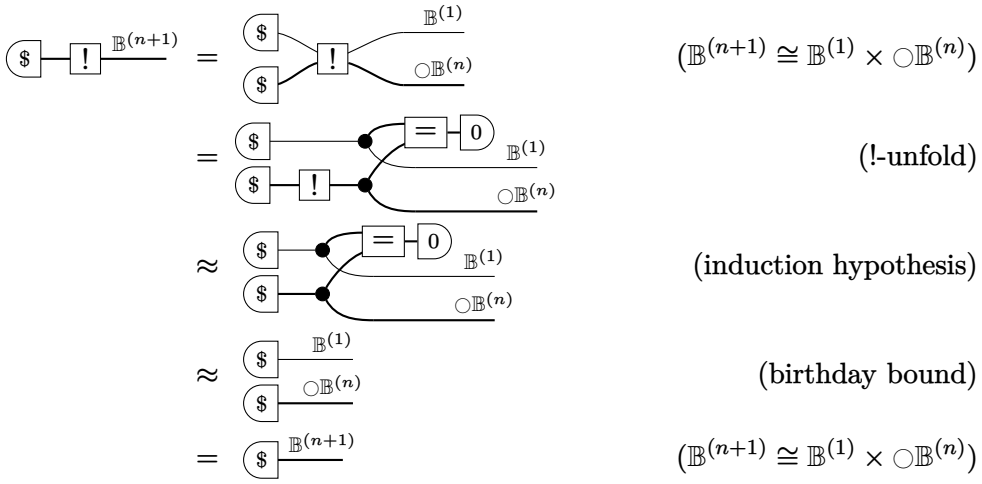
The following lemma is a diagrammatic version of the *birthday bound* argument, stating roughly that, for a sufficiently large set, we can safely assume that we will never pick the same value twice by sampling uniformly (or rather, that the probability of doing so is negligible). Alternatively, it states that sampling *without* replacement is indistinguishable from sampling *with* replacement. Its proof illustrate the induction proof method of Section 5.2.

Lemma 3. 

Proof. We cannot use coinduction here, as this statement involves indistinguishability. We prove the lemma by induction instead. First, the base case is immediate, as $!^{(1)} \colon \mathbb{B}^{(1)} \to \mathbb{B}^{(1)}$ is simply the identity. For the inductive case, assume that



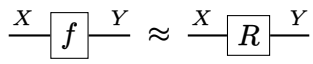
for some natural number $n \geq 1$; then, we have



□

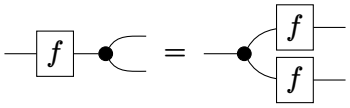
In this report, we are not just concerned with random functions, like R, but with *pseudo*random functions. In diagrammatic terms, we define a PRF to be any diagram $f\colon X \to Y$ which is indistinguishable from $R\colon X \to Y$.

Definition 10. *A diagram $f\colon X \to Y$ is a PRF if*



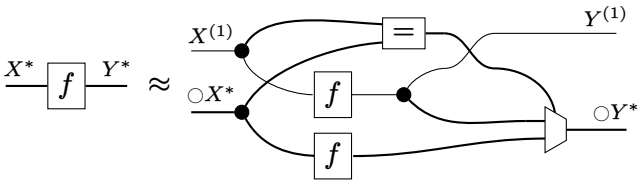
(PRF)

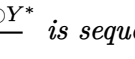
In a SMC equipped with copying, there is a standard notion of *deterministic* maps: they are those that satisfy



Semantically, these correspond to streams whose components map each possible input to a single output with probability 1. As we can see, a random dictionary is not deterministic in this sense, since, for a given input x, it samples its output y randomly. However, when queried a second time with x, it will always respond with y. In this report, we are interested in this second form of determinism, which we call *sequential* determinism. Intuitively, a sequentially-deterministic stream $f\colon X \to Y$ is one which, when fed the same input twice, will produce the same output.

Definition 11. *A diagram $f\colon X^* \to Y^*$ is* sequentially-deterministic *if*



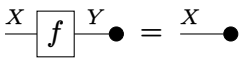
and  *is sequentially-deterministic.*

Note that the two boxes labelled with f in the above definition have different types: the top one denotes $[\![f]\!]^{\leq 1} : X^{(1)} \to Y^{(1)}$, which behaves like the head of $[\![f]\!]$ for the first timestep only (and as $\mathsf{id}_1 : 1 \to 1$ thereafter), while the second represents $\bigcirc [\![f]\!] : \bigcirc X^* \to \bigcirc Y^*$ (Definition 5).

Proposition 2. *Sequentially-deterministic diagrams are closed under composition and monoidal product.*

In a SMC with a deleting operation, there is a standard notion of *causal* map: they are those for which discarding their output is equivalent to not having performed them at all. The term causal refers to the fact that such maps do not impose constraints backwards, on their inputs.

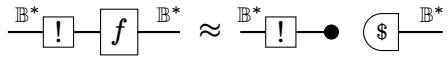
Definition 12. *A diagram $f : X \to Y$ is* causal *if it satisfies:*



For example, R is causal, while  and `!` are *not* causal.

The following characterises what it means for a PRF f to be *secure*: if we feed distinct inputs f, its outputs will appear independent and uniformly distributed to any efficient adversary. In fact, if f is also sequentially-deterministic and causal, this property is equivalent to being PRF. The proof of this lemma is in Appendix A.2.

Lemma 4. *(i) If $f : \mathbb{B}^* \to \mathbb{B}^*$ is a PRF, then*

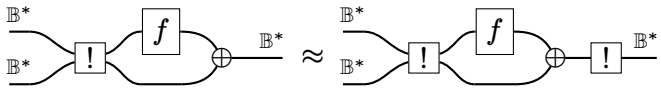


(3)

(ii) Moroever, if f is sequentially-deterministic and causal, then the converse holds, i.e.*, equation* (3) *implies that f is a PRF.*

The following lemma is the most technically challenging step in the proof of security of the Feistel cipher and contains the core of the combinatorial arguments in Luby and Rackoff's original paper [LR88]. In plain English, it states that if f is a PRF and the pair of inputs to the function $g : (x, y) \mapsto f(x) \oplus y$ are all distinct, then its outputs are also all distinct. Another way to phrase this is that g is a *universal hash function* [Ros26, Chap. 11]. As usual, the distinctness of the stream of values represented by a certain diagram is enforced using the `!`-box (for pairs in this case). The reader interested in its diagrammatic proof will find it in Appendix A.2.

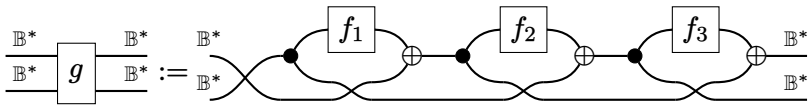
Lemma 5. *For $f : \mathbb{B}^* \to \mathbb{B}^*$ a pseudo-random function, we have*



5.4 Motivating Example Revisited

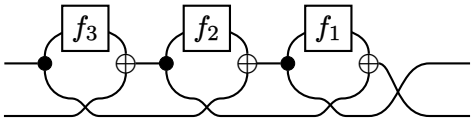
We now turn back to our motivating example, the 3-round Feistel cipher and prove its security.

Theorem 2 (Luby-Rackoff, 1988)**.** *If f_1, f_2, f_3 are PRFs, then*

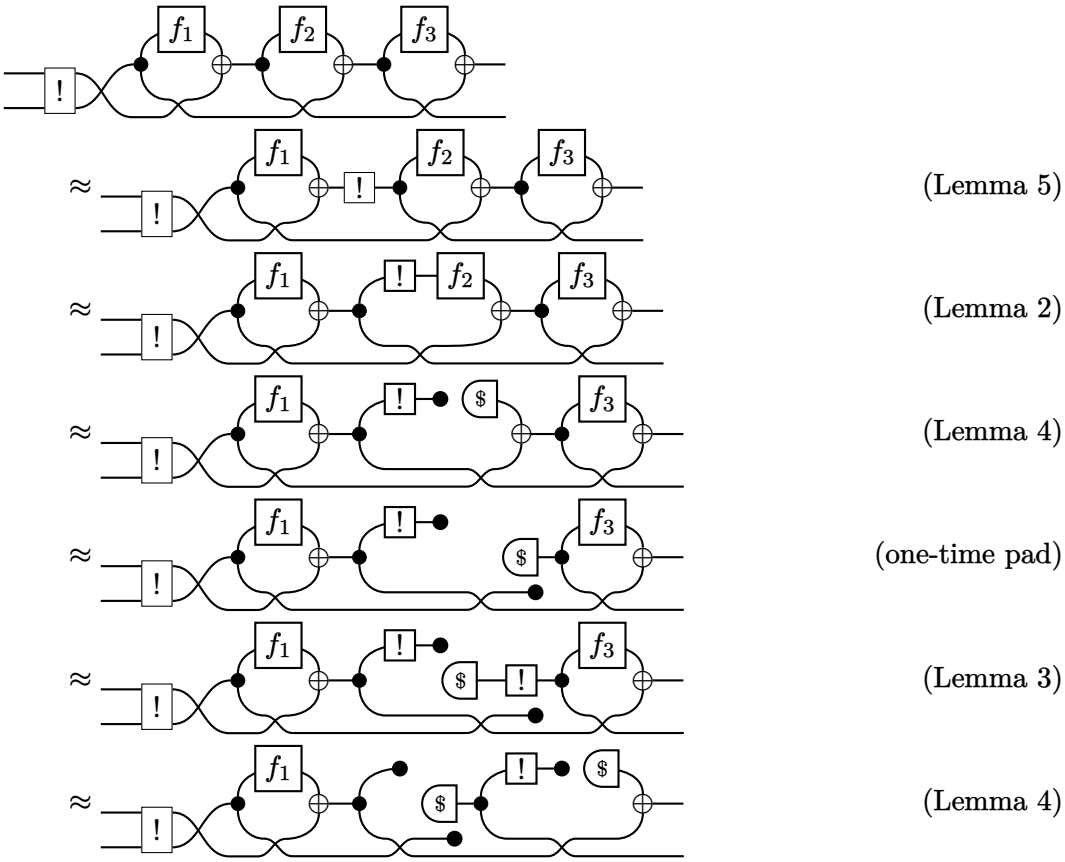


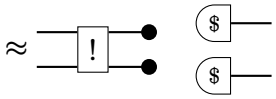
is a PRP.

Proof. That it is a permutation with the following inverse is easy to show:



Moreover, by Proposition 2, g is clearly sequentially-deterministic since all its components are. Thus, by Lemma 4 *(ii)* it suffices to show that, on distinct inputs, the outputs of the Feistel cipher are indistinguishable from uniform randomness:





(one-time pad) ;(delete)

□

6 Discussion and Outlook

The framework developed in this report represents a first step toward an algebraic reformulation of game-based cryptography. While we have focused on the 3-round Feistel construction as a central case study, the diagrammatic and categorical machinery introduced here is more general. In this section we outline several research directions that build on the present work and illustrate its broader potential.

Extending the framework. The Feistel construction illustrates how game-based arguments can be decomposed into local diagrammatic transformations. A natural next step is to extend this methodology to a broader class of constructions, ranging from symmetric-key primitives and hashing schemes to asymmetric cryptography, authentication mechanisms, and zero-knowledge protocols.

Extending the framework in this way would make it possible to identify reusable axiom schemes and proof patterns. Many arguments are expected to reduce to combinations of stream-unfolding laws together with a small number of probabilistic approximations, such as birthday-type bounds. A systematic study along these lines could lead to a library of compositional proof principles for cryptography.

Axiomatic unification across models. Beyond individual constructions lies a more conceptual objective: to isolate which aspects of security arguments are genuinely model-dependent and which are structural. The framework developed here deliberately separates syntax from semantics. The same diagrammatic language may therefore be interpreted in different symmetric monoidal categories: efficiently computable substochastic maps for the classical computational model, unrestricted stochastic maps for information-theoretic settings, algebraic terms for symbolic Dolev-Yao models, or completely positive maps in a quantum setting. Security proofs then take the form of derivations in the free symmetric monoidal category modulo a chosen collection of axioms.

This perspective suggests a programme of comparing models functorially. Given two semantic categories interpreting the same syntax, one may ask which diagrammatic equalities and indistinguishability relations are preserved under change of interpretation. In this way, relationships between information-theoretic and computational security, between symbolic and computational soundness results, or between classical and quantum security notions can be studied at the level of structural principles rather than individual constructions. The long-term ambition is to articulate a shared diagrammatic core spanning multiple cryptographic domains,

with model-specific assumptions appearing explicitly as axioms whose validity can be analysed independently. In this sense, the project aims not only to formalise proofs but also to provide a substrate-independent language for comparing cryptographic models.

Unifying proof methods: towards coinductive indistinguishability. A central technical challenge emerging from this work concerns the nature of computational indistinguishability itself. Equality of streams admits a coinductive characterisation via bisimulation. Computational indistinguishability, by contrast, is defined through quantification over polynomial-time distinguishers. Developing a coinductive characterisation of indistinguishability would therefore allow the different proof principles used in this work to be unified within a single framework.

One promising direction is to enrich the semantics with quantitative structure, for instance by equipping it with a pseudometric that tracks more precisely *how far* two streams are from one another. One of the authors of this report has developed such an enrichment in the setting of efficiently computable substochastic maps [BK23], which we plan to adapt to the present framework. On the syntactic side, this would lead naturally to quantitative equational reasoning with diagrams ([LRSZ25]), where statements of the form $d \overset{\epsilon}{\approx} d'$ guarantee that $[\![d]\!]$ and $[\![d']\!]$ differ by at most ϵ.

References

[BCDS17] David A. Basin, Cas J. F. Cremers, Jannik Dreier, and Ralf Sasse. Symbolically analyzing security protocols using tamarin. *ACM SIGLOG News*, 4:19–30, 2017. 4

[BGLZB11] Gilles Barthe, Benjamin Grégoire, Yassine Lakhnech, and Santiago Zanella-Béguelin. Beyond provable security verifiable ind-cca security of oaep. In *The Cryptographer's Track at RSA Conference*, 2011. 4

[BGZB09] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella-Béguelin. Formal certification of code-based cryptographic proofs. In *ACM-SIGACT Symposium on Principles of Programming Languages*, 2009. 4

[BK23] Anne Broadbent and Martti Karvonen. Categorical composable cryptography: extended version. *Logical Methods in Computer Science*, 19:30:1–30:46, 2023. 16, 34

[Bla06] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 15 pp.–154, 2006. 4

[Bla16] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Found. Trends Priv. Secur.*, 1:1–135, 2016. 4

[Car22] Titouan Carette. Propification and the scalable comonad. 2022. 26

[CJ19] Kenta Cho and Bart Jacobs. Disintegration and Bayesian inversion via string diagrams. *Mathematical Structures in Computer Science*, 29(7):938–971, 2019. 16

[DLDFR25] Elena Di Lavore, Giovanni De Felice, and Mario Román. Coinductive streams in monoidal categories. *Logical Methods in Computer Science*, 21, 2025. 3, 17, 18

[DLGR+23] Elena Di Lavore, Alessandro Gianola, Mario Román, Nicoletta Sabadini, and Paweł Sobociński. Span(Graph): a canonical feedback algebra of open transition systems. *Software and Systems Modeling*, 22(2):495–520, 2023. 3, 17

[DY81] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *22nd Annual Symposium on Foundations of Computer Science (sfcs 1981)*, pages 350–357, 1981. 4

[FGHL+23] Tobias Fritz, Tomáš Gonda, Nicholas Gauguin Houghton-Larsen, Antonio Lorenzin, Paolo Perrone, and Dario Stein. Dilations and information flow axioms in categorical probability. *Mathematical Structures in Computer Science*, 33(10):913–957, 2023. 12

[FGP21] Tobias Fritz, Tomáš Gonda, and Paolo Perrone. De Finetti's Theorem in Categorical Probability. *Journal of Stochastic Analysis*, 2(4):6, 2021. 12

[FGPR23] Tobias Fritz, Tomáš Gonda, Paolo Perrone, and Eigil Fjeldgren Rischel. Representable Markov categories and comparison of statistical experiments in categorical probability. *Theoretical Computer Science*, 961:113896, 2023. 12

[Fri20] Tobias Fritz. A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics. *Advances in Mathematics*, 370:107239, aug 2020. 11, 12, 16

[GSV26] Alessandro Di Giorgio, Pawel Sobocinski, and Niels F. W. Voorneveld. Parametric iteration in resource theories. In Stefano Guerrini and Barbara König, editors, *34th EACSL Annual Conference on Computer Science Logic, CSL 2026, Paris, France, February 23-28, 2026*, volume 363 of *LIPIcs*, pages 29:1–29:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026. 27

[JR11] Bart Jacobs and Jan J. M. M. Rutten. An introduction to (co)algebra and (co)induction. In *Advanced Topics in Bisimulation and Coinduction*, 2011. 20

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.*, 17(2):373–386, 1988. 4, 5, 31

[LRSZ25] Gabriele Lobbia, Wojciech Różowski, Ralph Sarkis, and Fabio Zanasi. Quantitative Monoidal Algebra: Axiomatising Distance with String Diagrams. In Paweł Gawrychowski, Filip Mazowiecki, and Michał Skrzypczak, editors, *50th International Symposium on Mathematical Foundations of Computer Science (MFCS 2025)*, volume 345 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:21, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 34

[Mei23] Lucas (cronokirby) Meier. A sketch of synthetic cryptography. `https://cronokirby.com/posts/a-sketch-of-synthetic-cryptography.html`, June 2023. Accessed: 2026-03-03. 4, 27

[MP22] Sean Moss and Paolo Perrone. Probability monads with submonads of deterministic states. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '22, New York, NY, USA, 2022. Association for Computing Machinery. 15

[Per23] Paolo Perrone. Markov categories and entropy. *IEEE Transactions on Information Theory*, 2023. 12

[PTSZ25] Robin Piedeleu, Mateo Torres-Ruiz, Alexandra Silva, and Fabio Zanasi. A complete axiomatisation of equivalence for discrete probabilistic programming. In Viktor Vafeiadis, editor, *Programming Languages and Systems - 34th European Symposium on Programming, ESOP 2025, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, ON, Canada, May 3-8, 2025, Proceedings, Part II*, volume 15695 of *Lecture Notes in Computer Science*, pages 202–229. Springer, 2025. 11, 27

[PZ25] Robin Piedeleu and Fabio Zanasi. *An Introduction to String Diagrams for Computer Scientists*. Elements in Applied Category Theory. Cambridge University Press, 2025. 23

[Ros26] Mike Rosulek. *The Joy of Cryptography*. MIT Press, 2026. 4, 31

[Rut01] Jan J. M. M. Rutten. Elements of stream calculus (an extensive exercise in coinduction). In *Mathematical Foundations of Programming Semantics*, 2001. 27

[WGZ22] Paul W. Wilson, Dan R. Ghica, and Fabio Zanasi. String diagrams for strictification and coherence. *Log. Methods Comput. Sci.*, 20, 2022. 26

[ZBPB17] Jean Karim Zinzindohoué, Karthikeyan Bhargavan, Jonathan Protzenko, and Benjamin Beurdouche. Hacl*: A verified modern cryptographic library. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017. 4

A Auxiliary Results and Proofs

A.1 Semantics

Theorem 1. *Computational indistinguishability is a congruence for composition and the monoidal product in* Stream*. In other words, the structure of SMC lifts to streams* up to *the equivalence relation* $\approx$*.*

Proof. For composition, let $f \approx p \colon X \to Y$ and $g \approx q \colon Y \to Z$ be four streams, and $a[-]$ be a distinguisher of type $X \to Z$. Then,

$$a[f;g]_\lambda = (g;a)[f]_\lambda$$

where $(g;a)[-]$ with state $A \times T$ (and T is the state space of g) and type $X \to Y$ is given by

$$(g;a)^{(0)}_\lambda = (a^{(0)}_\lambda \times t_{0,\lambda}); (\mathsf{id}_{A^{(0)}_\lambda} \times \sigma^{X_\lambda}_{T_\lambda})$$

$$(g;a)^{(i)}_\lambda = (\mathsf{id}^{(i-1)}_{A_\lambda} \times \sigma^{T_\lambda}_{Y_\lambda}); (\mathsf{id}_{A^{(i-1)}_\lambda} \times g_\lambda); (a^{(i)}_\lambda \times \mathsf{id}_{T_\lambda}); (\mathsf{id}_{A^{(i)}_\lambda} \times \sigma^{X_\lambda}_{T_\lambda})$$

$$(g;a)^{(\ell)}_\lambda = (\mathsf{id}^{(\ell-1)}_{A_\lambda} \times \sigma^{T_\lambda}_{Y_\lambda}); (\mathsf{id}_{A^{(\ell-1)}_\lambda} \times g_\lambda); (a^{(\ell)}_\lambda \times \delta_{T_\lambda})$$

Since f and p are indistinguishable,

$$(g;a)[f]_\lambda = (g;a)[p]_\lambda$$

and

$$(g;a)[p]_\lambda = a[p;g]_\lambda$$

Similarly,

$$a[p;g]_\lambda = (a;p)[g]_\lambda$$

where $(a;p)[-]$ with state $A \times R$ (and R is the state space of p) and type $Y \to Z$ is given by

$$(a;p)^{(0)}_\lambda = (a^{(0)}_\lambda \times r_{0,\lambda}); (\mathsf{id}_{A^{(0)}_\lambda} \times p_\lambda); (\mathsf{id}_{A^{(0)}_\lambda} \times \sigma^{Y_\lambda}_{R_\lambda})$$

$$(a;p)^{(i)}_\lambda = (\mathsf{id}^{(i-1)}_{A_\lambda} \times \sigma^{R_\lambda}_{Z_\lambda}); (a^{(i)}_\lambda \times \mathsf{id}_{R_\lambda}); (\mathsf{id}_{A^{(i)}_\lambda} \times p_\lambda); (\mathsf{id}_{A^{(i)}_\lambda} \times \sigma^{Y_\lambda}_{R_\lambda})$$

$$(a;p)^{(\ell)}_\lambda = (\mathsf{id}^{(\ell-1)}_{A_\lambda} \times \sigma^{R_\lambda}_{Z_\lambda} \times \delta_{T_\lambda}); (a^{(\ell)}_\lambda \times \delta_{R_\lambda})$$

Then, since g and q are indistinguishable,

$$(a;p)[g]_\lambda = (a;p)[q]_\lambda$$

and

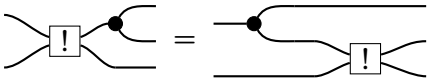
$$(a;p)[q]_\lambda = a[p;q]_\lambda$$

Therefore,

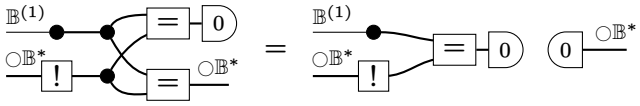
$$a[p;q]_\lambda = a[f;g]_\lambda$$

as we wanted to show. A similar proof works to show that indistinguishability is a congruence for the monoidal product too. □

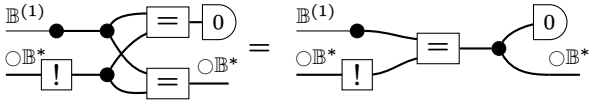
A.2 Diagrammatic Reasoning

Lemma 6. 

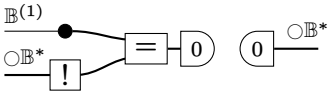
Proof. The proof is similar to that of Lemma 2 and we omit it here. □

Lemma 7. 

Proof.



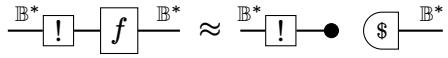
(=-copy)



(propagate observation)

□

Lemma 4. *(i) If* $f\colon \mathbb{B}^* \to \mathbb{B}^*$ *is a PRF, then*



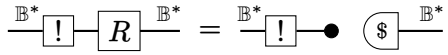
(3)

(ii) Moroever, if f *is sequentially-deterministic and causal, then the converse holds,* i.e.*, equation* (3) *implies that* f *is a PRF.*

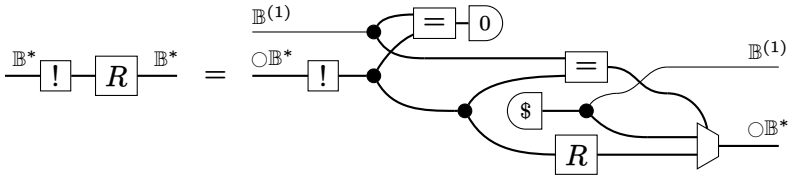
Proof. First, f is pseudo-random and therefore indistinguishable from a random dictionary R of the same type:

(PRF)

So it is sufficient to show that the lemma holds strictly for R, *i.e.* that

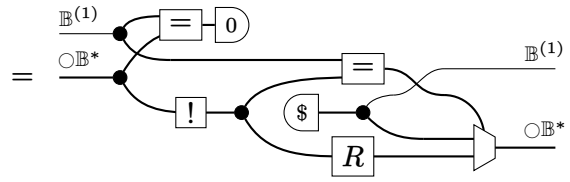


We do so by coinduction:[5]

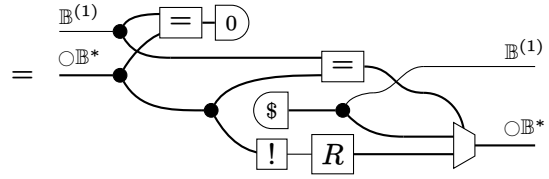


(R-unfold)

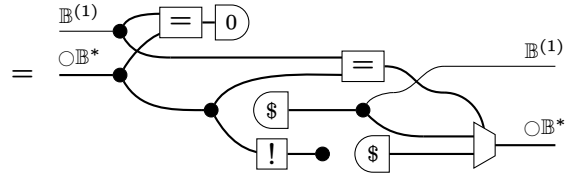
[5]Although the statement of the lemma involves the indistinguishability relation, we only use it to replace f with the random dictionary R. After that, the statement we want to prove is an equality between streams and is therefore amenable to a coinductive proof technique.



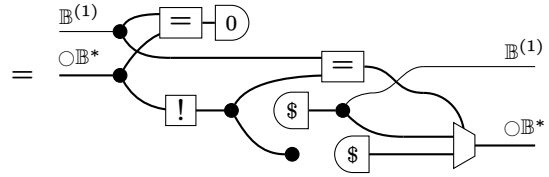
(Lemma 2)



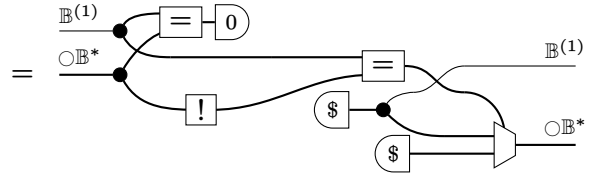
(Lemma 2)



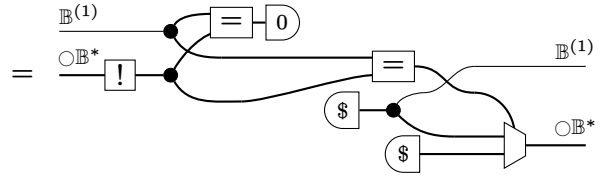
(coind. hyp.)



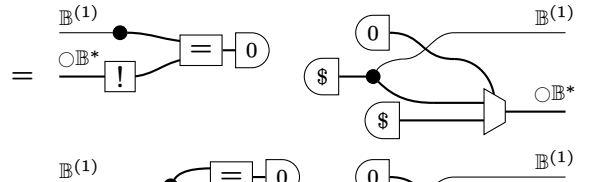
(Lemma 2)



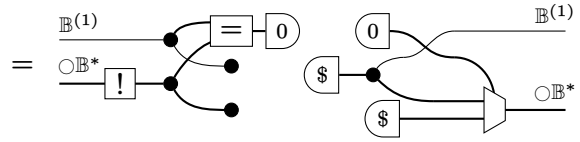
(counitality)



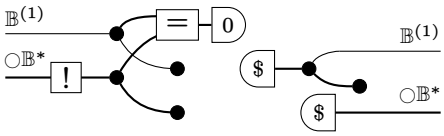
(Lemma 2)



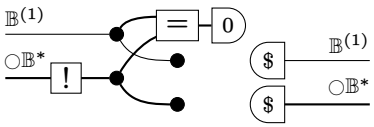
(Lemma 7)



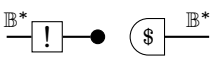
(counitality)



(Boolean equality)

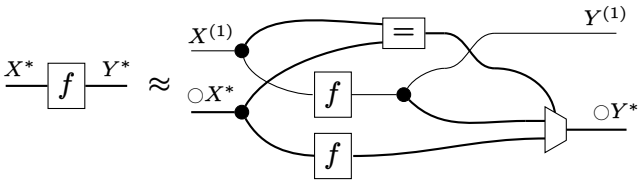


(counitality)



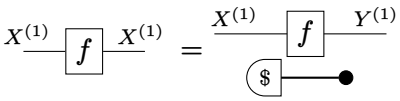
(!-fold)

For *(ii)*, assume that f is sequentially-deterministic:

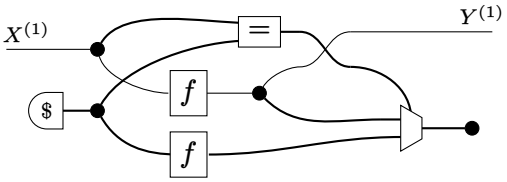


(4)

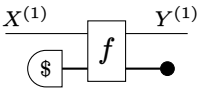
Thus,



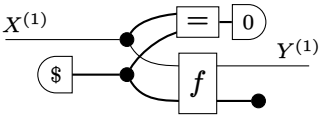
(delete)



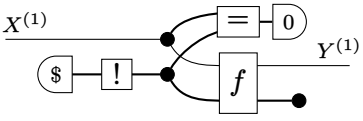
(delete)



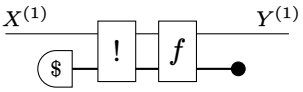
(4)



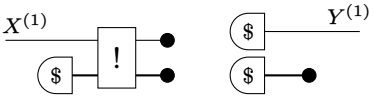
(birthday bound)



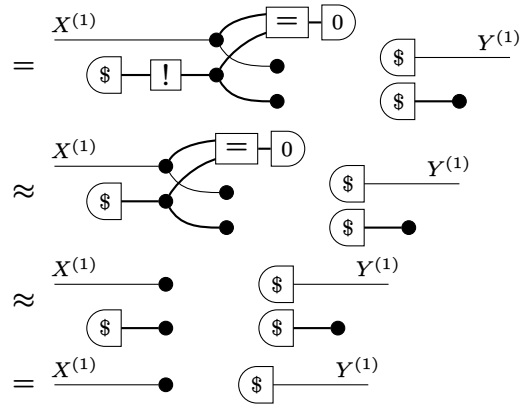
(Lemma 3)



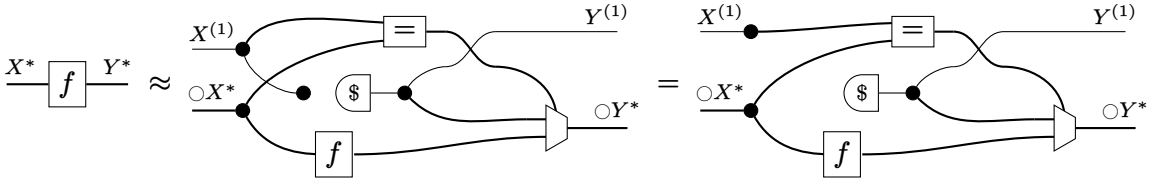
!-fold



(3)

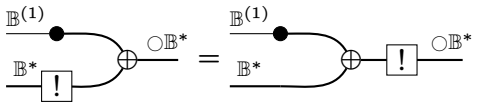


And hence,

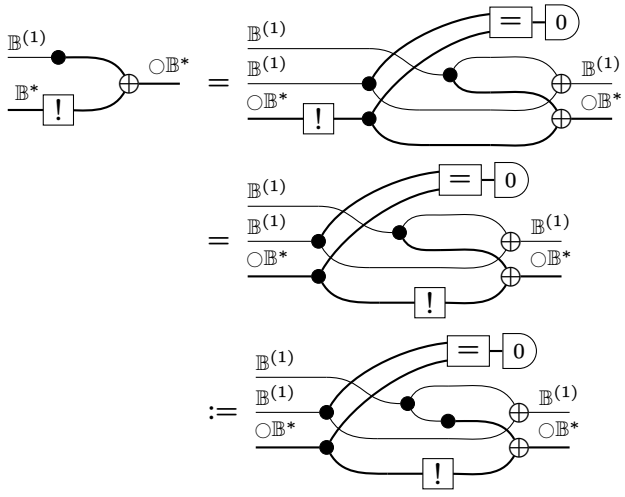


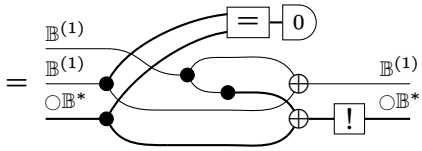
As we can see, this is the same equation as the axiom that characterises the behaviour of the random dictionary (R-unfold). An induction allows us to show that $f \approx R$. □

Lemma 8.

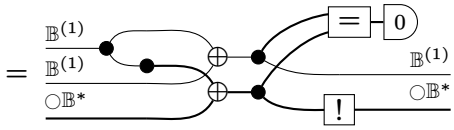


Proof. By coinduction:

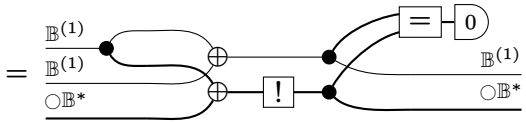




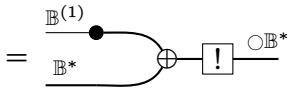
(coind. hyp.)



(Lemma 9)

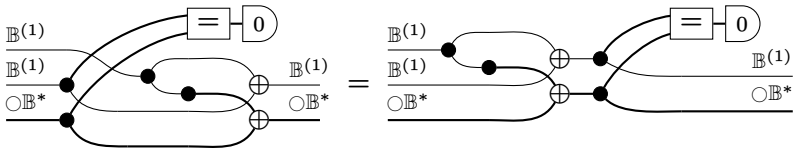


(Lemma 2)

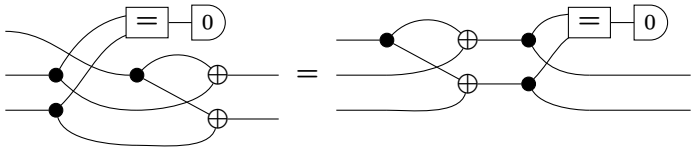


(!-fold)

□

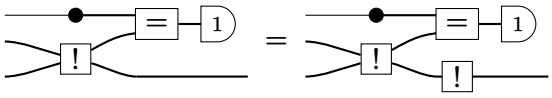
Lemma 9. 

Proof. Can also be shown by coinduction. The non-coinductive step relies on the following equality between substochastic maps:



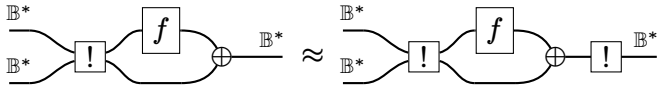
This is a diagrammatic representation of the following fact: $x \oplus y \neq x \oplus y'$ if and only if $y \neq y'$. In other words, it is equivalent to the injectivity of the mapping $x \oplus (-)$. □

The following lemma simply states that, if the first component of a stream of *pairs* is constant and we force all pairs to be distinct, then the elements of the second component have to be distinct.

Lemma 10. 

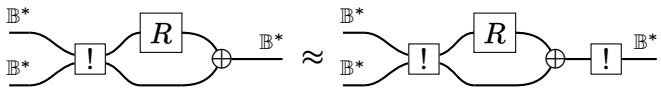
Proof. Also by coinduction. □

Lemma 5. *For $f\colon \mathbb{B}^* \to \mathbb{B}^*$ a pseudo-random function, we have*



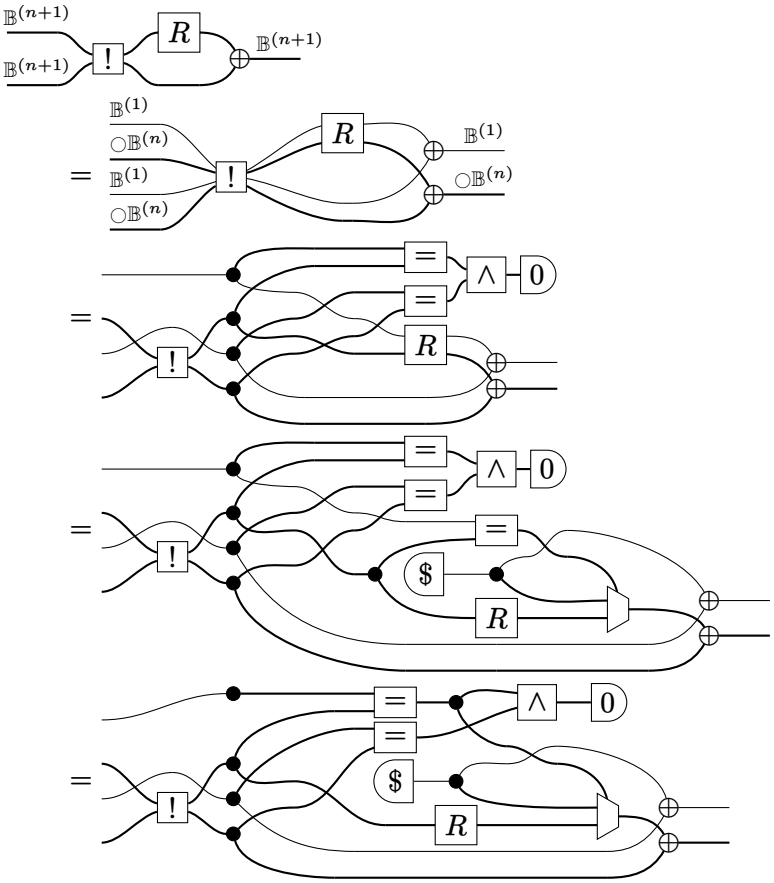
Proof. First, since f is pseudorandom, it is indistinguishable from R, a random dictionary of the same type:

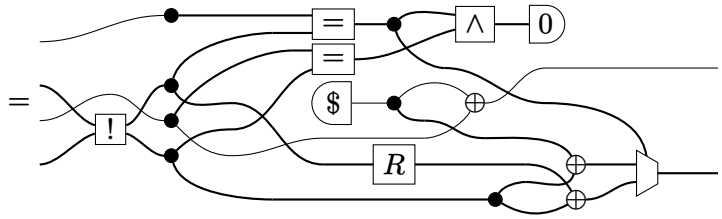
So it is sufficient to show that the lemma holds for R, *i.e.*, that



(5)

We show (5) by induction on the index of the streams that both sides denote. First, for $n = 1$, the !-box does not enforce any constraint (a single input does not have to be distinct from anything else) and so (5) holds trivially. Assume that (5) holds for some natural number $n > 0$; then we have,

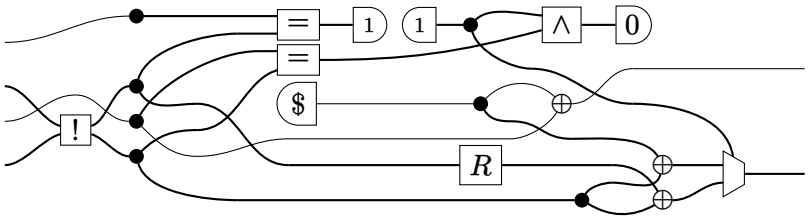


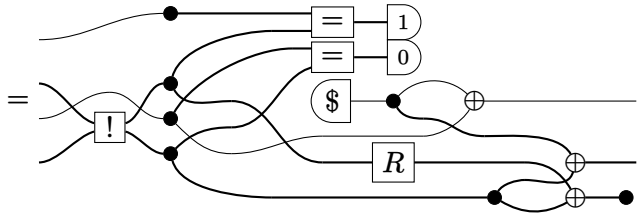


(⊕-distribute-if)

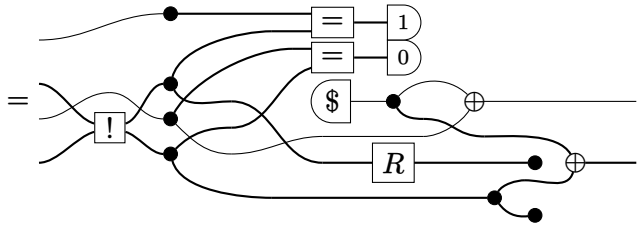
To simplify diagrams, let us break down the analysis into the two cases given by the then- and else-branch of the multiplexer. We do this, by substituting `true`/0 or `false`/1 on the appropriate wire.

1) First, for the then-branch, we have

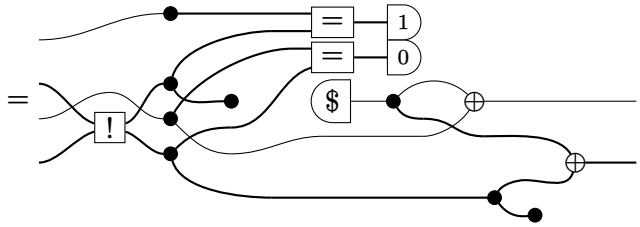




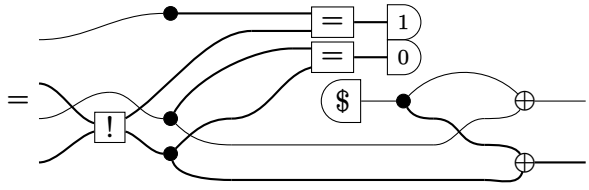
(Boolean equality)



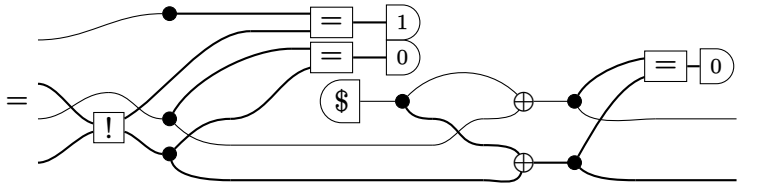
(delete)



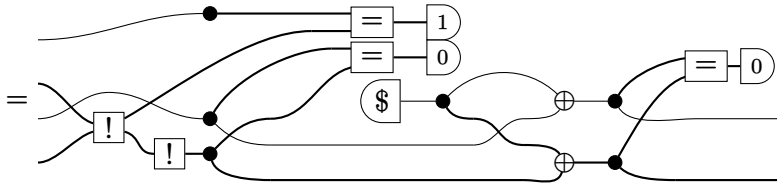
(delete)



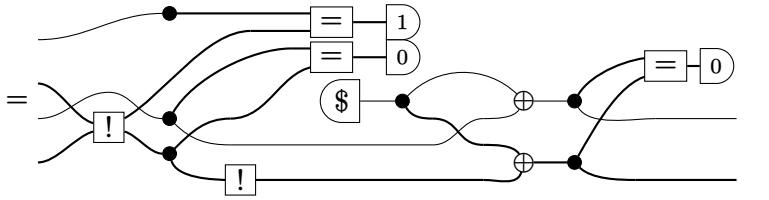
(counitality)



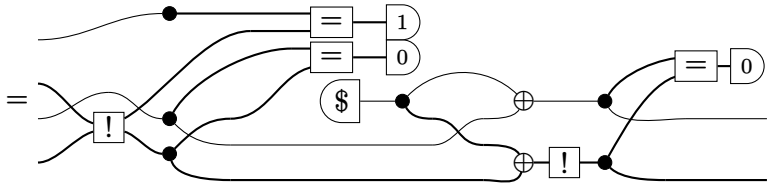
(Lemma 9)



(Lemma 10)

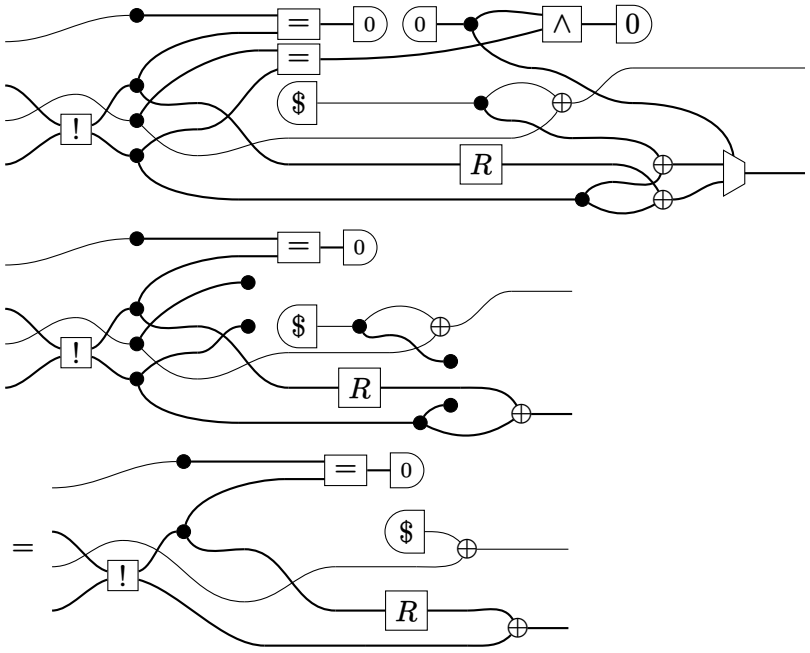


(Lemma 2)

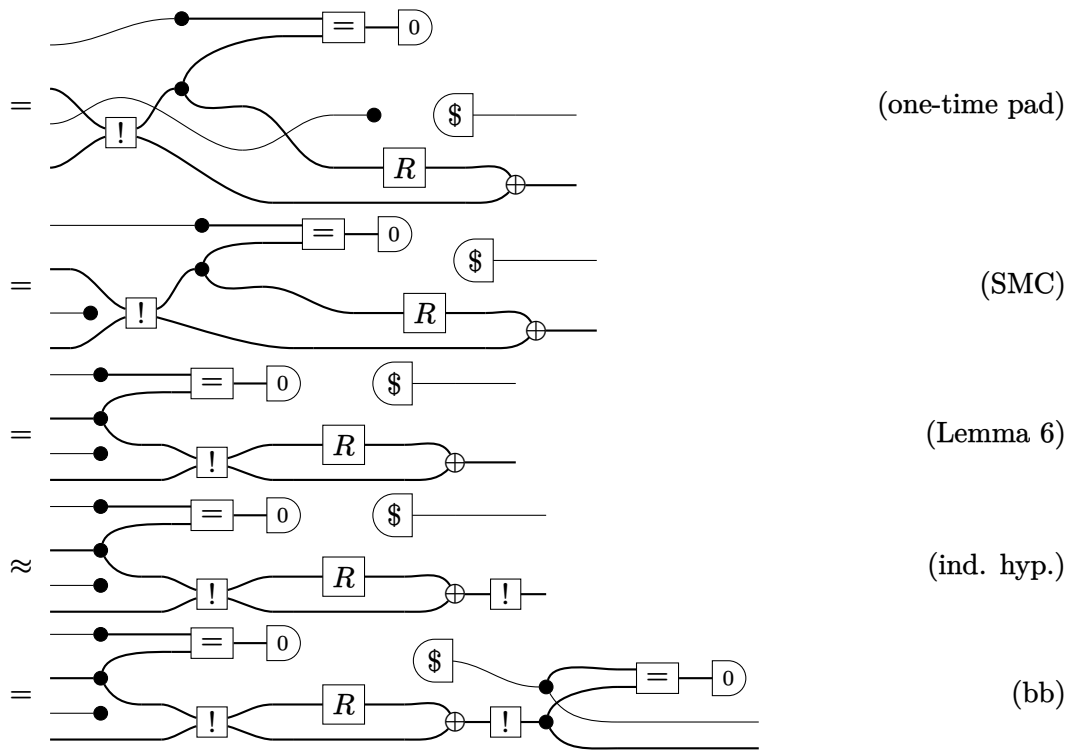


(Lemma 8)

2) Second, the case of the else-branch is slightly easier:



(counitality)



(one-time pad)

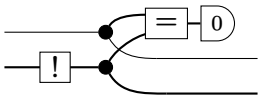
(SMC)

(Lemma 6)

(ind. hyp.)

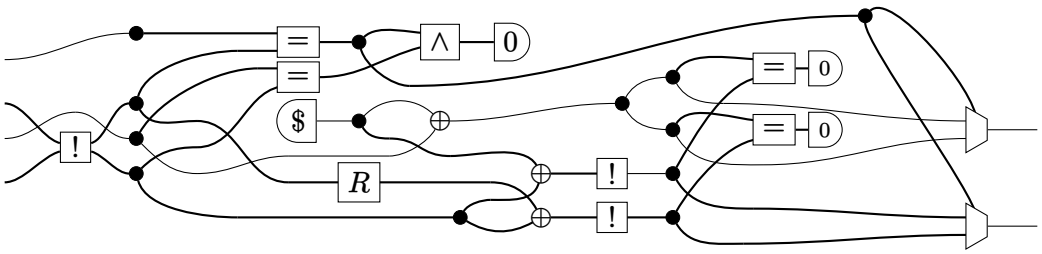
(bb)

Notice that, for both branches, we obtain the same sub-diagram as before post-composed with



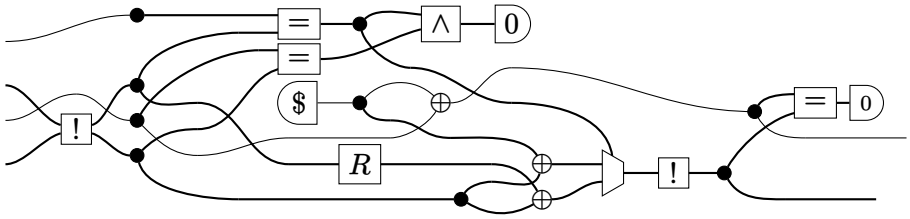
(6)

Therefore, we can put back the two cases together to obtain to following diagram:

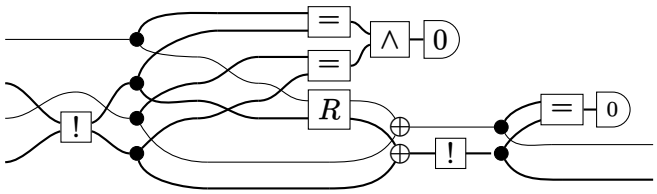


Since they are on both branches of the if-then-else gates, we can push both occurrences of (6)

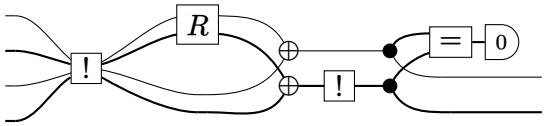
through to the output of the whole diagram:[6]



Then, by performing the first steps of this proof backwards, we obtain

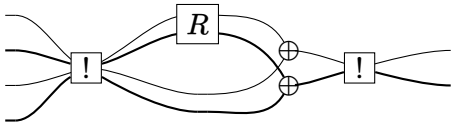


$=$



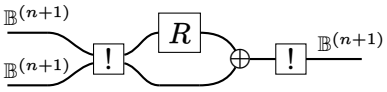
(!-unfold & R-unfold)

$=$



(!-unfold)

$=$



($\mathbb{B}^{(n+1)} \cong \mathbb{B}^{(1)} \times \bigcirc\mathbb{B}^{(n)}$)

thereby concluding the induction. □

[6] In symbolic form, `if` x `then` $t; u$ `else` $t; v$ is equivalent to just $t;$ (`if` x `then` u `else` v).