Public-Seed Pseudorandom Permutations*

Pratik Soni and Stefano Tessaro

University of California, Santa Barbara
{pratik_soni,tessaro}@cs.ucsb.edu

Abstract. A number of cryptographic schemes are built from (keyless) permutations, which are either designed in an ad-hoc fashion or are obtained by fixing the key in a block cipher. Security proofs for these schemes, however, idealize this permutation, i.e., making it random and accessible, as an oracle, to all parties. Finding plausible concrete assumptions on such permutations that guarantee security of the resulting schemes has remained an elusive open question.

This paper initiates the study of standard-model assumptions on permutations – or more precisely, on families of permutations indexed by a *public* seed. We introduce the notion of a *public-seed pseudorandom permutation* (psPRP), which is inspired by the UCE notion by Bellare, Hoang, and Keelveedhi (CRYPTO '13). It considers a two-stage security game, where only the second stage learns the seed, and the first-stage adversary, known as the source, is restricted to prevent trivial attacks – the security notion is consequently parameterized by the class of allowable sources. To this end, we define in particular unpredictable and reset-secure sources analogous to similar notions for UCEs.

We first study the relationship between psPRPs and UCEs. To start with, we provide efficient constructions of UCEs from psPRPs for both reset-secure and unpredictable sources, thus showing that most applications of the UCE framework admit instantiations from psPRPs. We also show a converse of this statement, namely that the five-round Feistel construction yields a psPRP for reset-secure sources when the round function is built from UCEs for reset-secure sources, hence making psPRP and UCE equivalent notions for such sources.

In addition to studying such reductions, we suggest generic instantiations of psPRPs from both block ciphers and (keyless) permutations, and analyze them in ideal models. Also, as an application of our notions, we show that a simple modification of a recent highly-efficient garbling scheme by Bellare et al. (S&P '13) is secure under our psPRP assumption.

Keywords: Symmetric cryptography, UCE, permutation-based cryptography, assumptions, indifferentiability

1 Introduction

Many symmetric cryptographic schemes are built generically from an underlying component, like a hash function or a block cipher. For several recent examples (e.g., hash functions [BDPV08,RS08], authenticated-encryption schemes [AJN15], PRNGs [BDPV10,GT16], garbling schemes [BHKR13]), this component is a (keyless) *permutation*, which is either designed from scratch to meet certain cryptanalytic goals (as in the case of SHA-3 and derived algorithms based on the sponge paradigm) or is instantiated by fixing the key in a block cipher like AES (as in the garbling scheme of [BHKR13]).

The security of these schemes is usually proved in the *ideal-permutation model*, that is, the permutation is randomly chosen, and all parties are given (black-box) access to it. Essentially no non-tautological assumptions on permutations are known which are sufficient to imply security.[1] This situation is in sharp contrast to that of hash functions, where despite the popularity of the random-oracle model, we have a good understanding of plausible security assumptions that can be satisfied by these functions. This is particularly important – not so much because we want to put ideal models in question, but because we would like to assess what is really expected from a good permutation or hash function that makes these schemes secure.

Our contributions, in a nutshell. This paper initiates the study of computational assumptions for permutation-based cryptography. Akin to the case of hash functions, we extend permutations with a *public* seed, that is, π_s is used in lieu of π, where s is a public parameter of the scheme. We introduce a new framework – which we call *public-seed pseudorandom permutations*, or psPRPs, for short – which we investigate, both in terms of realizability, as well as in terms of applications. Our approach takes inspiration from Bellare, Hoang, and Keelveedhi's UCE framework [BHK13], which we extend to permutations. As we will see, psPRPs are both useful and interesting objects of study.

Beyond definitions, we contribute in several ways. First off, we build UCEs from psPRPs via efficient permutation-based hash functions, and show conversely how to build psPRPs from UCEs using the Feistel construction. We also discuss generic instantiations of psPRPs from block ciphers and keyless permutations. Finally, we show a variant of the garbling scheme from [BHKR13] whose security can be based on a psPRP assumption on the underlying block cipher, without compromising efficiency. Our reductions between psPRPs and UCEs are established by general theorems that connect them with a weak notion of indifferentiability, which is of independent interest. We explain all of this in detail in the remainder of this introduction; an overview of the results is in Fig. 1.

The UCE framework: A primer. Bellare, Hoang, and Keelveedhi (BHK) [BHK13] introduced the notion of a *universal computational extractor* (UCE). For a seeded hash function $\mathsf{H} : \{0,1\}^s \times \{0,1\}^* \to \{0,1\}^h$, the UCE framework considers a two-stage security game. First, a *source* S is given oracle access to either $\mathsf{H}(s,\cdot)$ (for a random, and for now secret, seed s), or a random function $\rho : \{0,1\}^* \to \{0,1\}^h$. After a number of queries, the source produces some leakage $L \in \{0,1\}^*$. In the second stage, the distinguisher D learns *both* L and the seed s, and needs to decide whether S was interacting with $\mathsf{H}(s,\cdot)$ or ρ – a task we would like to be hard. Clearly, this is unachievable without restrictions on S, as it can simply set $L = y^*$, where y^* is the output of the oracle on a fixed input x^*, and D then checks whether $\mathsf{H}(s,x^*) = y^*$, or not.

[1] A notable exception is the line of work on establishing good bounds on the PRF-security of MACs derived from sponge-based constructions, as e.g. in [MMH+14,ADMA15,GPT15], where one essentially assumes that the underlying permutation yields a secure Even-Mansour [EM97] cipher.

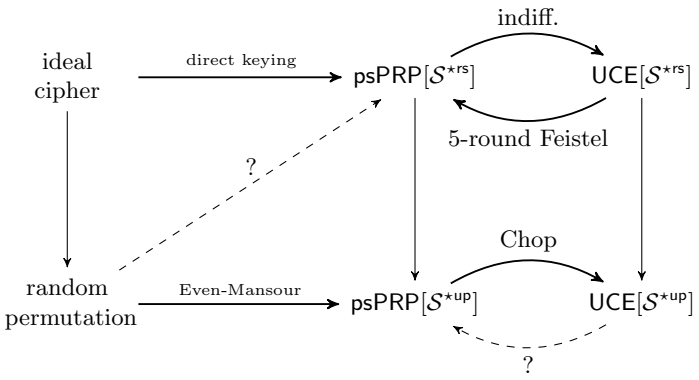

Fig. 1: **Relations established in this paper.** Here, $\star$ is set consistently everywhere either to c or to s. Lack of arrow indicates a separation, dashed lines indicate implications that are open and which we conjecture to hold true. Also note that in the ideal-cipher model, a random permutation is obtained by fixing the cipher key (e.g., to the all-zero string). We do not know whether the converse is true generically – indifferentiable constructions of ideal ciphers from random permutations (e.g., [ABD+13]) do not apply here [RSS11].

BHK propose to restrict the set of allowable sources – the security notion $\mathsf{UCE}[\mathcal{S}]$ corresponds to a function H being secure against all sources within a class $\mathcal{S}$. For example, *unpredictable* sources are those for which a predictor P, given the leakage $L \xleftarrow{\$} S^{\rho}$, cannot guess any of S's queries. They further distinguish between the class of *computationally* unpredictable sources $\mathcal{S}^{\mathsf{cup}}$ and the class of *statistically* unpredictable sources $\mathcal{S}^{\mathsf{sup}}$, depending on the powers of P. A somewhat stronger notion – referred to as *reset-security* – demands that a distinguisher R given $L \xleftarrow{\$} S^{\rho}$ accessing the random function ρ cannot tell whether it is given access to the same oracle ρ, or to a completely independent random oracle ρ'. One denotes as $\mathcal{S}^{\mathsf{srs}}$ and $\mathcal{S}^{\mathsf{crs}}$ the classes of statistical and computational reset-secure sources, respectively.

While $\mathsf{UCE}[\mathcal{S}^{\mathsf{cup}}]$-security (even under meaningful restrictions) was shown impossible to achieve in the standard model [BFM14,BST16] assuming indistinguishability obfuscation (IO) [BGI+01], there is no evidence of impossibility for $\mathsf{UCE}[\mathcal{S}^{\mathsf{sup}}]$ and $\mathsf{UCE}[\mathcal{S}^{\mathsf{srs}}]$, and several applications follow from them. Examples include providing standard-model security for a number of schemes and applications previously only secure in the random-oracle model, including deterministic [BHK13] and hedged PKE [BH15], immunizing backdoored PRGs [DGG+15], message-locked encryption [BHK13], hardcore functions [BHK13], point-function obfuscation [BHK13,BS16] simple KDM-secure symmetric encryption [BHK13], adaptively-secure garbling [BHK13], and CCA-secure encryption [MH14]. Moreover, as also pointed out by Mittelbach [Mit14], and already proved in the original BHK work, $\mathsf{UCE}[\mathcal{S}^{\mathsf{crs}}]$ and $\mathsf{UCE}[\mathcal{S}^{\mathsf{cup}}]$ are achievable in *ideal* models, and act as useful intermediate security notions for two-stage security games, where indifferentiability does not apply [RSS11].

Public-seed PRPs. We extend the UCE approach to the case of a *seeded* permutation $\pi : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^n$, that is, $\pi_s = \pi(s,\cdot)$ is an efficiently invertible permutation on n-bit strings. As in the UCE case, the security game will involve a source making queries to a permutation P *and its inverse* P^{-1}. In the real case, P / P^{-1} give access to π_s and π_s^{-1}, whereas in the ideal case they give access to a random permutation ρ and its inverse ρ^{-1}. Then, S passes on the leakage

L to the distinguisher D, which additionally learns s. The $\mathsf{psPRP}[\mathcal{S}]$ security notion demands indistinguishability of the real and ideal cases for all PPT D and all $S \in \mathcal{S}$.

This extension is straightforward, but it is not clear that it is useful at all. For instance, UCEs naturally generalize the notion of an extractor, yet no such natural "generalization" exists here, except that of extending the PRP game (played by the source) with a public-seed stage (and hence, the name psPRP). In addition, necessary source restrictions are somewhat less intuitive than in the UCE case. For instance, for psPRPs, for statistically/computationally unpredictable sources (we abuse notation, and denote the corresponding source classes also as $\mathcal{S}^{\mathsf{sup}}$ and $\mathcal{S}^{\mathsf{cup}}$) it must be hard for a predictor to guess an input *or* an output of the queries made by S.

UCEs from psPRPs. We first show that psPRPs are not only useful, but essentially allow to recover *all* previous applications of UCEs through simple constructions of UCEs from psPRPs.

Our first result shows that all permutation-based constructions which are *indifferentiable* from a random oracle [MRH04,CDMP05] transform a $\mathsf{psPRP}[\mathcal{S}^{\star\mathsf{rs}}]$-secure permutation into $\mathsf{UCE}[\mathcal{S}^{\star\mathsf{rs}}]$-secure hash function, where $\star \in \{\mathsf{c}, \mathsf{s}\}$.[2] In particular, this implies that the sponge paradigm by Bertoni *et al* [BDPV08], which underlies the SHA-3 hash function, can be used for such transformation, thus giving extra validation for the SHA-3 standard. We note that the permutation underlying SHA-3 is not seeded, but under the assumption that the underlying permutation is $\mathsf{psPRP}[\mathcal{S}^{\star\mathsf{rs}}]$-secure when seeded via the Even-Mansour construction [EM97], it is easy to enhance the sponge construction with a seed.

Note that $\mathcal{S}^{\star\mathsf{rs}}$ is a strictly larger class than $\mathcal{S}^{\star\mathsf{up}}$ (for both psPRP and UCE). Therefore, when an application only needs $\mathsf{UCE}[\mathcal{S}^{\star\mathsf{up}}]$-secure hashing, one may ask whether the assumption on the underlying psPRP can also be reduced. We will prove that this is indeed the case, and show that whenever π is $\mathsf{psPRP}[\mathcal{S}^{\star\mathsf{up}}]$-secure, then the simple construction that on input X outputs $\pi_s(X)[1 \dots r]$, that is, the first r bits of $\pi_s(X)$ is a secure fixed-input length $\mathsf{UCE}[\mathcal{S}^{\star\mathsf{up}}]$ as long as $r < n - \omega(\log \lambda)$. This result can be combined with the domain extender of [BHK14] to obtain a variable-input-length $\mathsf{UCE}[\mathcal{S}^{\star\mathsf{up}}]$-secure hash function.[3]

CP-sequential indifferentiability. The technique behind the above results is inspired by Bellare, Hoang, and Keelveedhi's work [BHK14] on UCE domain extension. They show that every construction that transforms a fixed-input length random oracle into a variable-input length one in the sense of indifferentiability [MRH04,CDMP05] is a good domain extender for UCEs.

We extend their result along three axes. First off, we show that it applies to arbitrary pairs of ideal primitives – e.g., a fixed-input-length or variable-input length random function or a random permutation. For example, a construction using a permutation which is indifferentiable from a random oracle transforms $\mathsf{psPRP}[\mathcal{S}^{\star\mathsf{rs}}]$-secure permutations into $\mathsf{UCE}[\mathcal{S}^{\star\mathsf{rs}}]$-secure functions. Through such a general treatment, our above result on sponges is a corollary of the indifferentiability analysis of [BDPV08].

Second, we show that a weaker version of indifferentiability, which we call *CP-sequential indifferentiability*, suffices. Recall that indifferentiability of a construction M transforming an ideal primitive $\mathbf{I}$ into an ideal primitive $\mathbf{J}$ means that there exists a simulator Sim such that $(\mathsf{M}^{\mathbf{I}}, \mathbf{I})$ and $(\mathbf{J}, \mathsf{Sim}^{\mathbf{J}})$ are indistinguishable. CP-sequential indifferentiability only demands this for distinguish-

[2] We note that the computational case, by itself, is not that useful, given we know that $\mathsf{UCE}[\mathcal{S}^{\mathsf{cup}}]$ and hence also $\mathsf{UCE}[\mathcal{S}^{\mathsf{crs}}]$ security is unachievable, unless IO does not exist. However, we may want to occasionally apply these results in ideal models, where the notion is achievable, and thus they are worth stating.

[3] Their construction pre-processes the arbitrary-long input with an almost universal hash function, as e.g. one based on polynomial evaluation.

ers that make all of their *construction* queries to $\mathsf{M}^{\mathbf{I}}$ / $\mathbf{J}$ *before* they proceed to *primitive* queries to $\mathbf{I}$ / $\mathsf{Sim}^{\mathbf{J}}$. As we will see, this significantly enlarges the set of constructions this result applies to. For example, truncating the permutation output to $r < n$ bits does not achieve indifferentiability, because a simulator on an inverse query Y needs to fix $\pi^{-1}(Y)$ to some X such that, for the given random function $\rho : \{0,1\}^n \to \{0,1\}^r$, $\rho(X)$ is consistent with Y on the first r bits, which is infeasible. Yet, the same construction *is* CP-sequentially indifferentiable, intuitively because there is no way for a distinguisher to catch an inconsistent random X, as this would require an extra query to ρ. CP-sequential indifferentiability is dual to the sequential indifferentiability notion of Mandal, Patarin, and Seurin [MPS12], which considers the opposite order of construction and primitive queries. In fact, the two notions are incomparable, as we explain below.

Finally, we will also show that under suitable restrictions on the construction M, the result extends from reset-secure sources to unpredictable ones. This will allow to lift our result for truncation to unpredictable sources.

Constructing psPRPs. Obviously, a central question is whether the assumption of being a psPRP is, by itself, attainable. Our general theorem answers this question already – existing indifferentiability result for Feistel constructions [HKT11,CHK+16,DSKT16,DS16] imply already that the 8-round Feistel construction transforms a function which is $\mathsf{UCE}[\mathcal{S}^{\star\mathsf{rs}}]$-secure into a $\mathsf{psPRP}[\mathcal{S}^{\star\mathsf{rs}}]$-secure permutation.

It is important however to assess whether simpler constructions achieve this result. *Here, we show that the five-round Feistel construction suffices.* Our proof heavily exploits our connection to CP-indifferentiability. Indeed, the six-round lower bound of [CHK+16] does not apply for CP-indifferentiabiliy, as it requires the ability to ask construction queries *after* primitive queries, and we show that CP-indifferentiability is achieved at five rounds already. Our result is not merely a simplification of earlier ones, and our simulation strategy is novel. In particular, while we still follow the chain-completion approach of previous works, due to the low number of rounds, we need to introduce new techniques to bound the complexity of the simulator. To our rescue will come the fact that no construction queries can be made after primitive queries, and hence only a limited number of chain types will need to be completed.

We also note that the we are not aware of any obvious lower bound that shows that more than four rounds are really necessary – four rounds are necessary alone to reach PRP security in the eyes of the source. We leave it as an open problem to show whether four rounds are sufficient. We also note that our result only deals with reset-secure sources, and we leave it as an open problem to find a similar result for unpredictable sources. For reasons we explain in the body, it seems reasonable to conjecture that a heavily unbalanced Feistel network with $\Omega(n)$ rounds achieves this transformation.

Constructing psPRPs, in ideal models. While the main purpose of the psPRP framework is that of removing ideal model assumptions, it is still valuable to assess how psPRPs are built in the first place. To this end, we also show how to *heuristically* instantiate psPRPs from existing cryptographic primitives, and here validation takes us necessarily back to ideal models. Plus, for ideal-model applications that require psPRP security as an intermediate notion (for instance, because we are analyzing two-stage games), these provide instantiations.

We validate two strategies: (1) Using a block cipher, and seed it through the key, and (2) Using a *keyless* permutation, and seeding via the Even-Mansour construction [EM97]. We prove that the

first approach is $\mathsf{psPRP}[\mathcal{S}^{\mathsf{crs}}]$-secure in the ideal-cipher model, and prove the second $\mathsf{psPRP}[\mathcal{S}^{\mathsf{cup}}]$-secure in the random permutation model.[4]

FIXED-KEY BLOCK-CIPHER BASED GARBLING FROM PSPRPS. As a benchmark for psPRPs, we revisit the garbling schemes from [BHKR13] based on fixed-key block ciphers, which achieve high degrees of efficiency by eliminating re-keying costs. Their original security analysis was in the ideal-cipher model, and their simplicity is unmatched by schemes with standard-model reductions.

We consider a simple variant of their Ga scheme and prove it secure under the assumption the underlying block cipher, when seeded through its key input, is $\mathsf{psPRP}[\mathcal{S}^{\mathsf{sup}}]$-secure. Our construction is slightly less efficient than the scheme from [BHKR13], since a different seed/key is used for every garbling. However, we still gain from the fact that no re-keying is necessary throughout a garbling operation, or the evaluation of a garbled circuit. We also note that our approach also extends to the GaX scheme of [BHKR13] with further optimizations.

EXTRA RELATED WORK. A few works gave UCE constructions. Brzuska and Mittelbach [BM14] gave constructions from auxiliary-input point obfuscation (AIPO) and iO. In a recent paper, under the exponential DDH assumption, Zhandry [Zha16] built a primitive (called an AI-PRG) which is equivalent to a UCE for a subset of $\mathcal{S}^{\mathsf{cup}}$ which is sufficient for instantiating point obfuscators. (The observation is not made explicit in [Zha16], but the definitions are equivalent.) None of these results is sufficiently strong to instantiate our Feistel-based construction of psPRPs.

Farshim and Mittelbach [FM16] also recently proposed an interactive extension of the UCE framework, called ICE. We believe our framework can be extended along similar lines, and leave this for future work.

The cryptanalysis community has studied block-cipher security under known keys, albeit with a different focus. For example, Knuden and Rijmen [KR07] gave attacks against Feistel networks and reduced-round versions of AES that find input-output pairs with specific properties (given the key) in time faster than it should be possible if the block cipher were a random permutation. Several such attacks were later given for a number of block ciphers. We are not aware of these attacks however invalidating psPRP security. Andreeva, Bogdanov, and Mennink [ABM14] gave formal models for known-key security in ideal models based on a weak form of indifferentiability, where construction queries are to the construction *under a known random key*. These are however unrelated.

OUTLINE. Section 2 proposes a general framework for public-seed pseudorandom notions, and Section 3 puts this to use to provide general reduction theorems between pairs of such primitives, and defines in particular CP-sequential indifferentiability. UCE constructions from psPRPs are given in Section 4, whereas Section 5 presents our main result on building psPRPs via the Feistel construction. Heuristic constructions are presented in Section 6, and finally we apply psPRPs to the analysis of garbling schemes in Section 7.

NOTATIONAL PRELIMINARIES. Throughout this paper, we denote by $\mathsf{Funcs}(X, Y)$ the set of functions $X \to Y$, and in particular use the shorthand $\mathsf{Funcs}(m, n)$ whenever $X = \{0,1\}^m$ and $Y = \{0,1\}^n$. We also denote by $\mathsf{Perms}(X)$ the set of permutations on the set X, and analogously, $\mathsf{Perms}(n)$ denotes the special case where $X = \{0,1\}^n$. We say that a function $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is *negligible* if for all $c \in \mathbb{N}$, there exists a λ_0 such that $f(\lambda) \leq \lambda^{-c}$ for all $\lambda \geq \lambda_0$.

Our security definitions and proofs will often use games, as formalized by Bellare and Rogaway [BR06]. Typically, our games will have boolean outputs – that is, either true or false – and

[4] Again, recall that IO-based impossibility for $\mathcal{S}^{\mathsf{cup}}$ and $\mathcal{S}^{\mathsf{crs}}$ do not apply because we are in ideal models.

we use the shorthand $\mathsf{Pr}[\mathsf{G}]$ to denote the probability that a certain game outputs the value true, or occasionally 1 (when the output is binary, rather than boolean).

2 Public-seed Pseudorandomness

We present a generalization of the UCE notion [BHK13], which we term *public-seed pseudorandomness*. We apply this notion to define psPRPs as a special case, but the general treatment will be useful to capture transformations between UCEs and psPRPs in Section 3 via one single set of theorems.

2.1 Ideal Primitives and their Implementations

We begin by formally defining *ideal primitives* using notation inspired by [GT15,BBT16].

IDEAL PRIMITIVES. An *ideal primitive* is a pair $\mathbf{I} = (\Sigma, \mathcal{T})$, where $\Sigma = \{\Sigma_\lambda\}_{\lambda\in\mathbb{N}}$ is a family of sets of functions (such that all functions in Σ_λ have the same domain and range), and $\mathcal{T} = \{\mathcal{T}_\lambda\}_{\lambda\in\mathbb{N}}$ is a family of probability distributions, where $\mathcal{T}_\lambda$'s range is a subset of Σ_λ for all $\lambda \in \mathbb{N}$. The ideal primitive $\mathbf{I}$, once the security parameter λ is fixed, should be thought of as an oracle that initially samples a function I as its initial state according to $\mathcal{T}_\lambda$ from Σ_λ. We denote this sampling as $I \leftarrow\!\!\$\, \mathbf{I}_\lambda$. Then, $\mathbf{I}$ provides access to I via queries, that is, on input $\mathbf{x}$ it returns $I(\mathbf{x})$.[5]

EXAMPLES. We give a few examples of ideal primitives using the above notation. In particular, let $\kappa, m, n : \mathbb{N} \to \mathbb{N}$ be functions.

Example 1. The *random function* $\mathbf{R}_{m,n} = (\Sigma^{\mathbf{R}}, \mathcal{T}^{\mathbf{R}})$ is such that for $\lambda \in \mathbb{N}$, $\Sigma^{\mathbf{R}}_\lambda = \mathsf{Funcs}(m(\lambda), n(\lambda))$, and $\mathcal{T}^{\mathbf{R}}_\lambda$ is the uniform distribution on $\Sigma^{\mathbf{R}}_\lambda$. We also define $\mathbf{R}_{*,n}$ to be the same for $\mathsf{Funcs}(*, n(\lambda))$, that is, when the domain is extended to arbitrary length input strings.[6]

Example 2. The *random permutation* $\mathbf{P}_n = (\Sigma^{\mathbf{P}}, \mathcal{T}^{\mathbf{P}})$ is such that for all $\lambda \in \mathbb{N}$,

$$\Sigma^{\mathbf{P}}_\lambda = \Big\{ P : \{\mathsf{+},\mathsf{-}\} \times \{0,1\}^{n(\lambda)} \to \{0,1\}^{n(\lambda)} \mid \exists \pi \in \mathsf{Perms}(n(\lambda)) :\ P(\mathsf{+},x) = \pi(x), P(\mathsf{-},x) = \pi^{-1}(x) \Big\} ,$$

and moreover, $\mathcal{T}^{\mathbf{P}}_\lambda$ is the uniform distribution on $\Sigma^{\mathbf{P}}_\lambda$.

Example 3. The *ideal cipher* $\mathbf{IC}_{\kappa,n} = (\Sigma^{\mathbf{IC}}, \mathcal{T}^{\mathbf{IC}})$ is such that

$$\Sigma^{\mathbf{IC}}_\lambda = \Big\{ E : \{0,1\}^{\kappa(\lambda)} \times \{\mathsf{+},\mathsf{-}\} \times \{0,1\}^{n(\lambda)} \to \{0,1\}^{n(\lambda)} \mid \forall k \in \{0,1\}^{\kappa(\lambda)} \exists \pi_k \in \mathsf{Perms}(n(\lambda)) :\ E(k,\mathsf{+},x) = \pi_k(x), E(k,\mathsf{-},x) = \pi_k^{-1}(x) \Big\} ,$$

and $\mathcal{T}^{\mathbf{IC}}_\lambda$ is the uniform distribution on $\Sigma^{\mathbf{IC}}_\lambda$.

[5] The reader may wonder whether defining Σ is necessary, but this will allow us to enforce a specific format on valid implementations below.

[6] Note that this requires some care, because Σ_λ is now uncountable, and thus sampling from it requires a precise definition. We will not go into formal details, similar to many other papers, but it is clear that this can easily be done.

MAIN $\mathsf{psPR}^{S,D}_{\mathsf{F},\mathbf{I}}(\lambda)$:	ORACLE $\mathcal{O}(i, \mathbf{x})$:
$(1^n, t) \leftarrow\!\$\ S(1^\lambda, \varepsilon)$	**if** $b = 1$ **then**
$b \leftarrow\!\$\ \{0,1\}$	**return** $\mathsf{F.Eval}(1^\lambda, k_i, \mathbf{x})$
$k_1, \ldots, k_n \leftarrow\!\$\ \mathsf{F.Kg}(1^\lambda)$	**else**
$f_1, \ldots, f_n \leftarrow\!\$\ \mathbf{I}_\lambda$	**return** $f_i(\mathbf{x})$
$L \leftarrow\!\$\ S^{\mathcal{O}}(1^\lambda, t)$	
$b' \leftarrow\!\$\ D(1^\lambda, k_1, \ldots, k_n, L)$	
return $b' = b$	

Fig. 2: Game psPR used to define pspr-security for a primitive F that is Σ-compatible with $\mathbf{I}$. Here, S is the source and D is the distinguisher. Recall that the notation $f \leftarrow\!\$\ \mathbf{I}_\lambda$ indicates picking a function from Σ_λ using $\mathcal{T}_\lambda$.

EFFICIENCY CONSIDERATIONS. Usually, for an ideal primitive $\mathbf{I} = (\Sigma, \mathcal{T})$, the bit-size of the elements of Σ_λ grows exponentially in λ, and thus one would not implement a primitive $\mathbf{I}$ by sampling I from Σ_λ, but rather using techniques such as lazy sampling. An implementation of a primitive $\mathbf{I}$ is a *stateful* randomized PPT algorithm A such that $A(1^\lambda, \cdot)$ behaves as $I \xleftarrow{\$} \mathbf{I}_\lambda$ for all $\lambda \in \mathbb{N}$. We say that $\mathbf{I}$ is *efficiently implementable* if such an A exists. All the above examples – $\mathbf{R}_{m,n}, \mathbf{R}_{*,n}, \mathbf{P}_n$, and $\mathbf{IC}_{\kappa,n}$ – are efficiently implementable as long as m, n, κ are polynomially bounded functions.

Σ-COMPATIBLE FUNCTION FAMILIES. A *function family* $\mathsf{F} = (\mathsf{Kg}, \mathsf{Eval})$ consists of a *key (or seed) generation algorithm* $\mathsf{F.Kg}$ and an *evaluation algorithm* $\mathsf{F.Eval}$. In particular, $\mathsf{F.Kg}$ is a randomized algorithm that on input the unary representation of the security parameter λ returns a *key* k, and we let $[\mathsf{F.Kg}(1^\lambda)]$ denote the set of all possible outputs of $\mathsf{F.Kg}(1^\lambda)$. Moreover, $\mathsf{F.Eval}$ is a deterministic algorithm that takes three inputs; the security parameter in unary form 1^λ, a key $k \in [\mathsf{F.Kg}(1^\lambda)]$ and a query $\mathbf{x}$ such that $\mathsf{F.Eval}(1^\lambda, k, \cdot)$ implements a function that maps queries $\mathbf{x}$ to $\mathsf{F.Eval}(1^\lambda, k, \mathbf{x})$. We say that F is *efficient* if both Kg and Eval are polynomial-time algorithms.

Definition 1 (Σ-compatibility). *A function family* F *is* Σ*-compatible with* $\mathbf{I} = (\Sigma, \mathcal{T})$ *if* $\mathsf{F.Eval}(1^\lambda, k, \cdot) \in \Sigma_\lambda$ *for all* $\lambda \in \mathbb{N}$ *and* $k \in [\mathsf{F.Kg}(1^\lambda)]$.

2.2 Public-seed Pseudorandomness, psPRPs, and Sources

We now define a general notion of public-seed pseudorandom implementations of ideal primitives.

THE GENERAL DEFINITION. Let $\mathsf{F} = (\mathsf{Kg}, \mathsf{Eval})$ be a function family that is Σ-compatible with an ideal primitive $\mathbf{I} = (\Sigma, \mathcal{T})$. Let S be an adversary called the *source* and D an adversary called the *distinguisher*. We associate to them, F and $\mathbf{I}$, the game $\mathsf{psPR}^{S,D}_{\mathsf{F},\mathbf{I}}(\lambda)$ depicted in Fig. 2. The source initially chooses the number of keys n. Then, in the second stage, it is given access to an oracle $\mathcal{O}$ and we require any query $(i, \mathbf{x})$ made to this oracle be valid, that is, $\mathbf{x}$ is a valid query for any $f_i \in \Sigma_\lambda$ and $i \in [n]$, for n output by the first stage of the source. When the challenge bit $b = 1$ ("real") the oracle responds via $\mathsf{F.Eval}$ under the key k_i ($\mathsf{F.Eval}(1^\lambda, k_i, \cdot)$) that is chosen by the game and *not* given to the source. When $b = 0$ ("ideal") it responds via f_i where $f_i \leftarrow\!\$\ \mathbf{I}_\lambda$. After its interaction with the oracle $\mathcal{O}$, the source S communicates the *leakage* $L \in \{0,1\}^*$ to D. The distinguisher is given access to the keys $k_1, \ldots, k_n$ and must now guess $b' \in \{0,1\}$ for b. The game returns true iff $b' = b$ and we describe the pspr-advantage of (S, D) for $\lambda \in \mathbb{N}$ as

$$\mathsf{Adv}^{\mathsf{pspr}[\mathbf{I}]}_{\mathsf{F},S,D}(\lambda) = 2 \Pr\left[\mathsf{psPR}^{S,D}_{\mathsf{F},\mathbf{I}}(\lambda)\right] - 1 \ . \tag{1}$$

MAIN $\mathsf{Pred}^{P}_{\mathbf{I},S}(\lambda)$:
$\mathsf{done} \leftarrow \mathsf{false}; Q \leftarrow \emptyset; (1^n, t) \leftarrow\$ S(1^\lambda, \varepsilon)$
$f_1, \ldots, f_n \leftarrow\$ \mathbf{I}_\lambda$
$L \leftarrow\$ S^{\mathcal{O}}(1^\lambda, t); \mathsf{done} \leftarrow \mathsf{true}$
$Q' \leftarrow\$ P^{\mathcal{O}}(1^\lambda, 1^n, L)$
return $(Q \cap Q' \neq \emptyset)$

ORACLE $\mathcal{O}(i, \mathbf{x})$:
if $\neg\mathsf{done}$ **then** $Q \leftarrow Q \cup \{\mathbf{x}\}$
return $f_i(\mathbf{x})$

MAIN $\mathsf{Reset}^{R}_{\mathbf{I},S}(\lambda)$:
$\mathsf{done} \leftarrow \mathsf{false}; (1^n, t) \leftarrow\$ S(1^\lambda, \varepsilon)$
$f_1^0, f_1^1, \ldots, f_n^0, f_n^1 \leftarrow\$ \mathbf{I}_\lambda$
$L \leftarrow\$ S^{\mathcal{O}}(1^\lambda, t); \mathsf{done} \leftarrow \mathsf{true}$
$b \leftarrow\$ \{0,1\}; b' \leftarrow\$ R^{\mathcal{O}}(1^\lambda, 1^n, L)$
return $b' = b$

ORACLE $\mathcal{O}(i, \mathbf{x})$:
if $\neg\mathsf{done}$ **then return** $f_i^0(\mathbf{x})$
else return $f_i^b(\mathbf{x})$

Fig. 3: Games Pred and Reset are used to define the unpredictability and reset-security of the source S respectively against the ideal primitive $\mathbf{I}$. Here, S is the source, P is the predictor and R is the reset adversary.

In the following, we are going to use the shorthands $\mathsf{UCE}[m, n]$ for $\mathsf{pspr}[\mathbf{R}_{m,n}]$, $\mathsf{UCE}[n]$ for $\mathsf{pspr}[\mathbf{R}_{*,n}]$, and $\mathsf{psPRP}[n]$ for $\mathsf{pspr}[\mathbf{P}_n]$.

Note that our security game captures the multi-key version of the security notions, also considered in past works on UCE, as it is not known to be implied by the single-key version, which is recovered by having the source initially output $n = 1$.

Restricting sources. One would want to define F as secure if $\mathsf{Adv}^{\mathsf{pspr}[\mathbf{I}]}_{\mathsf{F},S,D}(\lambda)$ is negligible in λ for all polynomial time sources S and distinguishers D. However, as shown already in the special case of UCEs [BHK13], this is impossible, as one can always easily construct (at least for non-trivial $\mathbf{I}$'s) a simple source S which leaks the evaluation of $\mathcal{O}$ on a given point, and D can check consistency given k.

Therefore to obtain meaningful and non-empty security definitions we restrict the considered sources to some class $\mathcal{S}$, without restricting the distinguisher class. Consequently, we denote by $\mathsf{psPR}[\mathbf{I}, \mathcal{S}]$ the security notion that asks $\mathsf{Adv}^{\mathsf{pspr}[\mathbf{I}]}_{\mathsf{F},S,D}(\lambda)$ to be negligible for all polynomial time distinguishers D and all sources $S \in \mathcal{S}$. Following [BHK13], we also use $\mathsf{psPR}[\mathbf{I}, \mathcal{S}]$ to denote the set of F's which are $\mathsf{psPR}[\mathbf{I}, \mathcal{S}]$-secure. Note that obviously, if $\mathcal{S}_1 \subseteq \mathcal{S}_2$, then $\mathsf{psPR}[\mathbf{I}, \mathcal{S}_2] \subseteq \mathsf{psPR}[\mathbf{I}, \mathcal{S}_1]$ where $\mathcal{S}_1$ and $\mathcal{S}_2$ are source classes for the ideal primitive $\mathbf{I}$. We will use the shorthands $\mathsf{psPRP}[n, \mathcal{S}]$ for $\mathsf{psPR}[\mathbf{P}_n, \mathcal{S}]$ and $\mathsf{UCE}[m, n, \mathcal{S}]$ for $\mathsf{psPR}[\mathbf{R}_{m,n}, \mathcal{S}]$, where $m = *$ if the domain is unbounded.

Below, we discuss two important classes of restrictions, which are fundamental for the remainder of this paper – unpredictable and reset-secure sources.

Unpredictable sources. Let S be a source. Consider the game $\mathsf{Pred}^{P}_{\mathbf{I},S}(\lambda)$ of Fig. 3 associated to S and an adversary P called the predictor. Given the leakage, the latter outputs a set Q'. It wins if this set contains any $\mathcal{O}$-query of the source. For $\lambda \in \mathbb{N}$ we let

$$\mathsf{Adv}^{\mathsf{pred}[\mathbf{I}]}_{S,P}(\lambda) = \Pr\left[\mathsf{Pred}^{P}_{\mathbf{I},S}(\lambda)\right] . \tag{2}$$

We say that P is a *computational predictor* if it is polynomial time, and it is a *statistical* predictor if there exists polynomials q, q' such that for all $\lambda \in \mathbb{N}$, predictor P makes at most $q(\lambda)$ oracle queries and outputs a set Q' of size at most $q'(\lambda)$ in game $\mathsf{Pred}^{P}_{\mathbf{I},S}(\lambda)$. We stress that in this case the predictor need not be polynomial time, even though it makes a polynomial number of queries. We say S is *computationally unpredictable* if $\mathsf{Adv}^{\mathsf{pred}[\mathbf{I}]}_{S,P}(\lambda)$ is negligible for all computational predictors

MAIN $\mathsf{CP}[\mathbf{I} \to \mathbf{J}]^{A}_{\mathsf{M},\mathsf{Sim}}(\lambda)$:	ORACLE $\mathsf{Func}(\mathbf{x})$:	ORACLE $\mathsf{Prim}(\mathbf{u})$:
$b \leftarrow\!\$\ \{0,1\}$; $f \leftarrow\!\$\ \mathbf{I}_\lambda$; $g \leftarrow\!\$\ \mathbf{J}_\lambda$	**if** $b = 1$ **then**	**if** $b = 1$ **then**
$\mathsf{st} \leftarrow\!\$\ A_1^{\mathsf{Func}}(1^\lambda)$	**return** $\mathsf{M}^f(\mathbf{x})$	**return** $f(\mathbf{u})$
$b' \leftarrow\!\$\ A_2^{\mathsf{Prim}}(1^\lambda, \mathsf{st})$	**else**	**else**
return $b' = b$	**return** $g(\mathbf{x})$	**return** $\mathsf{Sim}^g(\mathbf{u})$

Fig. 4: Game CP used to define cpi-security for a construction M implementing the primitive $\mathbf{J}$ using primitive $\mathbf{I}$. Here, Sim is the simulator and $A = (A_1, A_2)$ is the two-stage distinguisher.

P. We say S is *statistically unpredictable* if $\mathsf{Adv}^{\mathsf{pred}[\mathbf{I}]}_{S,P}(\lambda)$ is negligible for all statistical predictors P. We let $\mathcal{S}^{\mathsf{cup}}$ be the class of all polynomial time, computationally unpredictable sources and $\mathcal{S}^{\mathsf{sup}} \subseteq \mathcal{S}^{\mathsf{cup}}$ be the class of all polynomial time statistically unpredictable sources.[7]

RESET-SECURE SOURCES. Let S be a source. Consider the game $\mathsf{Reset}^R_{\mathbf{I},S}(\lambda)$ of Fig. 3 associated to S and an adversary R called the reset adversary. The latter wins if given the leakage L it can distinguish between f^0 used by the source S and an independent f^1 where $f^0, f^1 \leftarrow\!\$\ \mathbf{I}_\lambda$. For $\lambda \in \mathbb{N}$ we let

$$\mathsf{Adv}^{\mathsf{reset}[\mathbf{I}]}_{S,R}(\lambda) = 2\Pr\left[\mathsf{Reset}^R_{\mathbf{I},S}(\lambda)\right] - 1\ . \tag{3}$$

We say that R is a *computational reset adversary* if it is polynomial time, and it is a statistical reset adversary if there exists a polynomial q such that for all $\lambda \in \mathbb{N}$, reset adversary R makes at most $q(\lambda)$ oracle queries in game $\mathsf{Reset}^R_{\mathbf{I},S}(\lambda)$. We stress that in this case the reset adversary need not be polynomial time. We say S is *computationally reset-secure* if $\mathsf{Adv}^{\mathsf{reset}[\mathbf{I}]}_{S,R}(\lambda)$ is negligible for all computational reset adversaries R. We say S is *statistically reset-secure* if $\mathsf{Adv}^{\mathsf{reset}[\mathbf{I}]}_{S,R}(\lambda)$ is negligible for all statistical reset adversaries R. We let $\mathcal{S}^{\mathsf{crs}}$ be the class of all polynomial time, computationally reset-secure sources and $\mathcal{S}^{\mathsf{srs}} \subseteq \mathcal{S}^{\mathsf{crs}}$ the class of all polynomial time statistically reset-secure sources.

RELATIONSHIPS. For the case of psPRPs, we mention the following fact, which is somewhat less obvious than in the UCE case, and in particular only holds if the permutation's domain grows with the security parameter.

Proposition 1. *For all $n \in \omega(\log \lambda)$, we have $\mathsf{psPRP}[n, \mathcal{S}^{\star\mathsf{rs}}] \subseteq \mathsf{psPRP}[n, \mathcal{S}^{\star\mathsf{up}}]$ where $\star \in \{\mathsf{c}, \mathsf{s}\}$.*

Proof (Sketch). In the reset secure game, consider the event that R queries its oracle $\mathcal{O}$ on input (i, σ, x) which was queried by S already as an $\mathcal{O}(i, \sigma, x)$ query, or it was the answer to a query $\mathcal{O}(i, \overline{\sigma}, y)$. Here (like elsewhere in the paper), we use the notational convention $\overline{+} = -$ and $\overline{-} = +$. The key point here is proving that as long as this bad event does not happen, the $b = 0$ and $b = 1$ case are hard to distinguish. A difference with the UCE case is that due to the permutation property, they will not be perfectly indistinguishable, but a fairly standard (yet somewhat tedious) birthday argument suffices to show that indistinguishability still holds as long as the overall number of $\mathcal{O}$ queries (of S and R) is below $2^{n(\lambda)/2}$, which is super-polynomial for $n(\lambda) = \omega(\log \lambda)$. □

[7] We note that computational unpredictability is only meaningful for sufficiently restricted classes of sources or in ideal models, as otherwise security against $\mathcal{S}^{\mathsf{cup}}$ is not achievable assuming IO, using essentially the same attack as [BFM14].

3 Reductions and Indifferentiability

We present general theorems that we will use to obtain reductions between psPRPs and UCEs. Our general notation for public-seed pseudorandom primitives allows us to capture the reductions through two general theorems.

CP-SEQUENTIAL INDIFFERENTIABILITY. Indifferentiability was introduced in [MRH04] by Maurer, Renner, and Holenstein to formalize reductions between ideal primitives. Following their general treatment, it captures the fact that a (key-less) construction M using primitive $\mathbf{I}$ (which can be queried by the adversary directly) is *as good* as another primitive $\mathbf{J}$ by requiring the existence of a simulator that can simulate $\mathbf{I}$ consistently by querying $\mathbf{J}$.

Central to this paper is a weakening of indifferentiability that we refer to as *CP-sequential indifferentiability*, where the distinguisher A makes all of its *construction* queries to $\mathsf{M}^{\mathbf{I}}$ (or $\mathbf{J}$) *before* moving to making *primitive queries* to $\mathbf{I}$ (or $\mathsf{Sim}^{\mathbf{J}}$, where Sim is the simulator). Note that this remains a non-trivial security goal, since Sim does not learn the construction queries made by A, but needs to simulate correctly nonetheless. However, the hope is that because A has committed to its queries before starting its interaction with Sim, the simulation task will be significantly easier. (We will see that this is indeed the case.)

More concretely, the notion is concerned with constructions which implement $\mathbf{J}$ from $\mathbf{I}$, and need to at least satisfy the following syntactical requirement.

Definition 2 ($(\mathbf{I} \to \mathbf{J})$-compatibility). *Let $\mathbf{I} = (\mathbf{I}.\Sigma, \mathbf{I}.\mathcal{T})$ and $\mathbf{J} = (\mathbf{J}.\Sigma, \mathbf{J}.\mathcal{T})$ be ideal primitives. A construction M is called $(\mathbf{I} \to \mathbf{J})$-compatible if for every $\lambda \in \mathbb{N}$, and every $f \in \mathbf{I}.\Sigma_\lambda$, the construction M implements a function $x \mapsto \mathsf{M}^f(1^\lambda, x)$ which is in $\mathbf{J}.\Sigma_\lambda$.*

The game CP is described in Fig. 4. For ideal primitives $\mathbf{I}, \mathbf{J}$, a two-stage adversary $A = (A_1, A_2)$, an $(\mathbf{I} \to \mathbf{J})$-compatible construction M, and simulator Sim, as well as security parameter $\lambda \in \mathbb{N}$, we define

$$\mathsf{Adv}^{\mathsf{cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},A}(\lambda) = 2 \cdot \Pr\left[\mathsf{CP}[\mathbf{I} \to \mathbf{J}]^A_{\mathsf{M},\mathsf{Sim}}(\lambda)\right] - 1 \ . \tag{4}$$

We remark that the CP-sequential indifferentiability notion is the exact dual of sequential indifferentiability as introduced by Mandal, Patarin, and Seurin [MPS12], which postpones construction queries *to the end*. As we will show below in Section 4.2, there are CP-indifferentiable constructions which are not sequentially indifferentiable in the sense of [MPS12].

REDUCTIONS. We show that CP-sequential indifferentiability yields a reduction between public-seed pseudorandomness notions. A special case was shown in [BHK14] by Bellare, Hoang, and Keelvedhi for domain extension of UCEs. Our result goes beyond in that: (1) It is more general, as it deals with arbitrary ideal primitives, (2) It only relies on CP-sequential indifferentiability, as opposed to full indifferentiability, and (3) The reduction of [BHK14] only considered reset-secure sources, whereas we show that under certain conditions on the construction, the reduction also applies to unpredictable sources. Nonetheless, our proofs follow the same approach of [BHK14], and the main contribution is conceptual.

We let $\mathsf{F} = (\mathsf{F.Kg}, \mathsf{F.Eval})$ be a function family which is Σ-compatible with an ideal primitive $\mathbf{I}$. Further, let M be an $(\mathbf{I} \to \mathbf{J})$-compatible construction. Then, overloading notation, we define the new function family $\mathsf{M}[\mathsf{F}] = (\mathsf{M.Kg}, \mathsf{M.Eval})$, where $\mathsf{M.Kg} = \mathsf{F.Kg}$, and for every $k \in [\mathsf{M.Kg}(1^\lambda)]$, we let

$$\mathsf{M.Eval}(1^\lambda, k, x) = \mathsf{M}^O(1^\lambda, x) \ , \tag{5}$$

where $O(z) = \mathsf{F.Eval}(1^\lambda, k, z)$.

GAME $\mathsf{EXT}^{S,P}_{\mathsf{M},\mathbf{I},\mathsf{Ext}}(\lambda)$:	ORACLE $\mathcal{O}(i,x)$:
$\mathsf{done} \leftarrow \mathsf{false}$ $Q_{\mathbf{I}}, Q_{\mathsf{M}} \leftarrow \emptyset$ $(1^n, \mathsf{st}) \leftarrow\!\!\$\ S(1^\lambda, \varepsilon)$ $f_1, \ldots, f_n \leftarrow\!\!\$\ \mathbf{I}_\lambda$ $L \leftarrow\!\!\$\ S^{\mathcal{O}_{\mathsf{M}}}(1^\lambda, 1^n, \mathsf{st})$ $\mathsf{done} \leftarrow \mathsf{true}$ $Q \leftarrow\!\!\$\ P^{\mathcal{O}}(1^\lambda, 1^n, L)$; $Q^* \leftarrow \mathsf{Ext}^{\mathcal{O}}(Q)$ **return** $((Q \cap Q_{\mathbf{I}} \neq \emptyset) \wedge (Q^* \cap Q_{\mathsf{M}} = \emptyset))$	**if** $\neg\mathsf{done}$ **then** $Q_{\mathbf{I}} \xleftarrow{\cup} \{x\}$ **return** $f_i(x)$ ORACLE $\mathcal{O}_{\mathsf{M}}(i,x)$: **if** $\neg\mathsf{done}$ **then** $Q_{\mathsf{M}} \xleftarrow{\cup} \{x\}$ $y \leftarrow \mathsf{M}^{\mathcal{O}(i,\cdot)}(x)$ **return** y

Fig. 5: Game $\mathsf{EXT}^{S,P}_{\mathsf{M},\mathbf{I},\mathsf{Ext}}(\lambda)$ in the definition of query extractability.

RESET-SECURE SOURCES. The following is our general reduction theorem for the case of reset-secure sources. Its proof follows similar lines as the one in [BHK14] and is in Appendix A.2.

Theorem 1 (Composition theorem, reset-secure case). *Let* M, F, $\mathbf{I}$, *and* $\mathbf{J}$ *be as above. Fix any simulator* Sim. *Then, for every source-distinguisher pair* (S, D), *where* S *requests at most* $N(\lambda)$ *keys, there exists a source-distinguisher pair* $(\overline{S}, \overline{D})$, *and a further distinguisher* A, *such that*

$$\mathsf{Adv}^{\mathsf{pspr}[\mathbf{J}]}_{\mathsf{M}[\mathsf{F}],S,D}(\lambda) \leq \mathsf{Adv}^{\mathsf{pspr}[\mathbf{I}]}_{\mathsf{F},\overline{S},\overline{D}}(\lambda) + N(\lambda) \cdot \mathsf{Adv}^{\mathsf{cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},A}(\lambda)\ . \tag{6}$$

Here, in particular: The complexities of D *and* $\overline{D}$ *are the same. Moreover, if* S, D, *and* M *are polynomial time, and* $\mathbf{I}$, $\mathbf{J}$ *are efficiently implementable, then* A, $\overline{S}$ *and* $\overline{D}$ *are also polynomial-time.*

Moreover, for every reset adversary R, *there exists a reset adversary* R' *and a distinguisher* B *such that*

$$\mathsf{Adv}^{\mathsf{reset}[\mathbf{I}]}_{\overline{S},R}(\lambda) \leq \mathsf{Adv}^{\mathsf{reset}[\mathbf{J}]}_{S,R'}(\lambda) + 3N(\lambda) \cdot \mathsf{Adv}^{\mathsf{cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},B}(\lambda)\ , \tag{7}$$

where R' *makes a polynomial number of query / runs in polynomial time if* R *and* Sim *make a polynomial number of queries / run in polynomial time, and* $\mathbf{I}, \mathbf{J}$ *are efficiently implementable.* ■

QUERY EXTRACTABLE CONSTRUCTIONS. Next, we show that under strong conditions on the construction M, Theorem 1 extends to the case of unpredictability.

In particular, we consider constructions which we term *query extractable*. Roughly, what such constructions guarantee is that every query made by M to an underlying ideal primitive $\mathbf{I}$ can be assigned to a (small) set of possible inputs to M that would result in this query during evaluation. Possibly, this set of inputs may be found by making some additional queries to $\mathbf{I}$. We define this formally through the game $\mathsf{EXT}^{S,P}_{\mathsf{M},\mathbf{I},\mathsf{Ext}}(\lambda)$ in Fig. 5. It involves a *source* S and a *predictor* P, as well as an extractor Ext. Here, S selects an integer n, which results in n instances $f_1, \ldots, f_n$ of $\mathbf{I}$ being spawned, and then makes queries to n instances of M^{f_i}, gives some leakage to the predictor P, and the predictor makes further query to the $\mathbf{I}$-instances, until it outputs a set Q. Then, we run the extractor Ext on Q, and the extractor can also make additional queries to the $\mathbf{I}$-instances, and outputs an additional set Q^*. We are interested in the event that Q contains one of queries made to the f_i's by M in the first stage of the game, yet Q^* does not contain any of S's queries to M^{f_i} for some i. In particular, we are interested in

$$\mathsf{Adv}^{\mathsf{ext}[\mathbf{I}]}_{\mathsf{M},S,P,\mathsf{Ext}}(\lambda) = \mathsf{Pr}\left[\mathsf{EXT}^{S,P}_{\mathsf{M},\mathbf{I},\mathsf{Ext}}(\lambda)\right]\ .$$

We say that M is *query extractable with respect to* $\mathbf{I}$ if there exists a polynomial time Ext such that $\mathsf{Adv}^{\mathsf{ext}[\mathbf{I}]}_{\mathsf{M},S,P,\mathsf{Ext}}(\lambda)$ is negligible for all PPT P and S. We say it is *perfectly* query extractable if the advantage is 0, rather than simply negligible.

The next theorem provides an alternative to Theorem 1 for the case of unpredictable sources whenever M guarantees query extractability. The proof of theorem is in Appendix A.3.

Theorem 2 (Composition theorem, unpredictable case). *Let* M, F, $\mathbf{I}$, *and* $\mathbf{J}$ *be as before. Fix any simulator* Sim. *Then, for every source-distinguisher pair* (S, D), *where* S *requests at most* $N(\lambda)$ *keys, there exists a source-distinguisher pair* $(\overline{S}, \overline{D})$, *and a further distinguisher* A, *such that*

$$\mathsf{Adv}^{\mathsf{pspr}[\mathbf{J}]}_{\mathsf{M}[\mathsf{F}],S,D}(\lambda) \le \mathsf{Adv}^{\mathsf{pspr}[\mathbf{I}]}_{\mathsf{F},\overline{S},\overline{D}}(\lambda) + N(\lambda) \cdot \mathsf{Adv}^{\mathsf{cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},A}(\lambda) \ . \tag{8}$$

Here, in particular: The complexities of D *and* $\overline{D}$ *are the same. Moreover, if* S, D, *and* M *are polynomial time, and* $\mathbf{I}$, $\mathbf{J}$ *are efficiently implementable, then* A, $\overline{S}$ *and* $\overline{D}$ *are also polynomial-time.*

Moreover, for every predictor P *and extractor* Ext, *there exists a predictor adversary* P' *and a distinguisher* B *such that*

$$\mathsf{Adv}^{\mathsf{pred}[\mathbf{I}]}_{\overline{S},P}(\lambda) \le \mathsf{Adv}^{\mathsf{pred}[\mathbf{J}]}_{S,P'}(\lambda) + \mathsf{Adv}^{\mathsf{ext}[\mathbf{I}]}_{\mathsf{M},S,P,\mathsf{Ext}}(\lambda) + N(\lambda) \cdot \mathsf{Adv}^{\mathsf{cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},B}(\lambda) \ , \tag{9}$$

where P' *makes a polynomial number of query / runs in polynomial time if* P, Sim *and* Ext *make a polynomial number of queries / run in polynomial time, and* $\mathbf{I}, \mathbf{J}$ *are efficiently implementable.* ■

4 From psPRPs to UCEs

We consider the problem of building UCEs from psPRPs. On the one hand, we want to show that all applications of UCEs can be recovered modularly by instantiating the underlying UCE with a psPRP-based construction. Second, we want to show that practical permutation-based designs can be instantiated by assuming the underlying permutation (when equipped with a seed) is a psPRP.

4.1 Reset-secure Sources and Sponges

The case of reset-secure sources follows by a simple application of Theorem 1: A number of constructions from permutations have been proved indifferentiable from a random oracle, and all of these yield a construction of a UCE for $\mathcal{S}^{\star\mathsf{rs}}$ when the underlying permutation is a psPRP for $\mathcal{S}^{\star\mathsf{rs}}$, where $\star \in \{\mathsf{c}, \mathsf{s}\}$.[8]

Sponges. A particular instantiation worth mentioning is the sponge construction by Bertoni et al. [BDPV08], which underlies KECCAK/SHA-3. In particular, let $\mathsf{Sponge}_{n,r}$ be the $(\mathbf{P}_n \to \mathbf{R}_{*,r})$-compatible construction which operates as follows, on input 1^λ, $M \in \{0,1\}^*$, and given oracle access to a permutation $\rho : \{0,1\}^{n(\lambda)} \to \{0,1\}^{n(\lambda)}$. The message M is split into r-bit blocks $M[1], \ldots, M[\ell]$, and the computation keeps a state $S_i \,\|\, T_i$, where $S_i \in \{0,1\}^r$ and $T_i \in \{0,1\}^{n-r}$. Then, $\mathsf{Sponge}^{\rho}_{n,r}(1^\lambda, M) = S_\ell[1..r]$, where

$$S_0 \,\|\, T_0 \leftarrow 0^n \ , \quad S_i \,\|\, T_i \leftarrow \rho((S_{i-1} \oplus M[i]) \,\|\, T_{i-1}) \text{ for } i = 1, \ldots, \ell \ .$$

Then, the following theorem follows directly from Theorem 1 and the indifferentiability analysis of [BDPV08]. (We state here only the asymptotic version, but concrete parameters can be obtained from these theorems.)

[8] One caveat is that some of these constructions use a few independent random permutations, whereas Theorem 1 assumes only one permutation is used. We point out in passing that Theorem 1 can easily be adapted to this case.

Theorem 3 (UCE-security for Sponges). *For* $\star \in \{\mathsf{c},\mathsf{s}\}$ *and* $n(\lambda)$ *polynomially bounded in* λ*, if* $\mathsf{F} \in \mathsf{psPRP}[n, \mathcal{S}^{\star\mathsf{rs}}]$*, then* $\mathsf{Sponge}_{n,r}[\mathsf{F}] \in \mathsf{UCE}[*, r, \mathcal{S}^{\star\mathsf{rs}}]$ *whenever* $n(\lambda) - r(\lambda) = \omega(\log \lambda)$. ■

HEURISTIC INSTANTIATION. We wish to say this validates SHA-3 as being a good UCE. One caveat of Theorem 3 is that the actual sponge construction (as used in SHA-3) uses a seedless permutation π. We propose the following assumption on such a permutation π that – if true – implies a simple way to modify an actual Sponge construction to be a secure UCE using Theorem 3. In particular, we suggest using the Even-Mansour [EM97] paradigm to add a seed to π. Given a family of permutations $\Pi = \{\pi_\lambda\}_{\lambda\in\mathbb{N}}$, where $\pi_\lambda \in \mathsf{Perms}(n(\lambda))$, define then $\mathsf{EM}[\Pi] = (\mathsf{EM.Kg}, \mathsf{EM.Eval})$ where $\mathsf{EM.Kg}$ outputs a random $n(\lambda)$-bit string s on input 1^λ, and

$$\mathsf{EM.Eval}(1^\lambda, s, (+, x)) = s \oplus \pi_\lambda(x \oplus s)\ ,\ \mathsf{EM.Eval}(1^\lambda, s, (-, y)) = s \oplus \pi_\lambda^{-1}(y \oplus s)$$

for all $s, x \in \{0,1\}^{n(\lambda)}$. Now, if Π is such that $\mathsf{EM}[\Pi]$ is $\mathsf{psPRP}[n, \mathcal{S}^{\mathsf{srs}}]$-secure, then $\mathsf{Sponge}[\mathsf{EM}[\Pi]]$ is $\mathsf{UCE}[*, r, \mathcal{S}^{\mathsf{srs}}]$-secure by Theorem 3. We discuss the conjecture that EM is $\mathsf{psPRP}[n, \mathcal{S}^{\mathsf{srs}}]$-secure further below in Section 6.

The attractive feature of $\mathsf{Sponge}[\mathsf{EM}[\Pi]]$ is that it can be implemented in a (near) black-box way from $\mathsf{Sponge}[\Pi]$, that is, the original sponge construction run with fixed oracle Π, by setting (1) The initial state $S_0 \,\|\, T_0$ to the seed s (rather than $0^{n(\lambda)}$), and (2) xoring the first r bits $s[1 \ldots r]$ of the seed s to the output. The other additions of the seed s to the inner states are unnecessary, as they cancel out. (A similar observation was made by Chang et al [CDH+12] in the context of keying sponges to obtain PRFs.)

4.2 Unpredictable Sources

Many UCE applications only require (statistical) unpredictability. In this section, we see that for this weaker target a significantly simpler construction can be used. In particular, we will first build a $\mathsf{UCE}[n, r, \mathcal{S}^{\star\mathsf{up}}]$-secure *compression function* from a $\mathsf{psPRP}[n, \mathcal{S}^{\star\mathsf{up}}]$-secure permutation, where $n(\lambda) - r(\lambda) = \omega(\log \lambda)$ and $\star \in \{\mathsf{c},\mathsf{s}\}$. Combined with existing domain extension techniques [BHK14], this can be enhanced to a variable-input-length UCE for the same class of sources.

THE CHOP CONSTRUCTION. Let $r, n : \mathbb{N} \to \mathbb{N}$ be polynomially bounded functions of the security parameter λ, where $r(\lambda) \le n(\lambda)$ for all $\lambda \in \mathbb{N}$. We consider the following construction $\mathsf{Chop}[n, r]$ which is $(\mathbf{P}_n \to \mathbf{R}_{n,r})$-compatible. On input 1^λ, it expects a permutation $\pi : \{0,1\}^n \to \{0,1\}^n$ for $n = n(\lambda)$, and given additionally $x \in \{0,1\}^{n(\lambda)}$, it returns

$$\mathsf{Chop}[n, r]^\pi(1^\lambda, x) = \pi(x)[1 \ldots r(\lambda)]\ , \tag{10}$$

that is, the first $r = r(\lambda)$ bits of $\pi(x)$. It is not hard to see that the construction is (perfectly) query extractable using the extractor Ext which given oracle access to $\mathcal{O}$ and a set Q of queries of the form $(+, x)$ and $(-, y)$, returns a set consisting of all x such that $(+, x) \in Q$, and moreover adds x' to the set obtained by querying $\mathcal{O}(i, -, y)$ for every $i \in [n]$ and $(-, y) \in Q$.

CP-SEQUENTIAL INDIFFERENTIABILITY. The following theorem establishes CP-sequential indifferentiability of the Chop construction. We refer the reader to Appendix B for the proof but give some intuition about it after the theorem.

Theorem 4 (CP-indifferentiability of Chop). *Let* $r, n : \mathbb{N} \to \mathbb{N}$ *be such that* $r(\lambda) \le n(\lambda)$ *for all* $\lambda \in \mathbb{N}$*. Let* $\mathbf{P} = \mathbf{P}_n$ *and* $\mathbf{R} = \mathbf{R}_{n,r}$ *be the random permutation and random function, respectively.*

Then, there exists a simulator Sim *such that for all distinguishers* A *making at most* q *construction and* p *primitive queries,*

$$\mathsf{Adv}^{\mathsf{cpi}[\mathbf{P}\to\mathbf{R}]}_{\mathsf{Chop}[n,r],\mathsf{Sim},A}(\lambda) \leq \frac{(q+p)^2}{2^n} + \frac{p\cdot q}{2^{n-r}} \,. \tag{11}$$

Here, Sim *makes at most one oracle query upon each invocation, and otherwise runs in time polynomial in the number of queries answered.* ■

The dependence on r is necessary, as otherwise the construction becomes invertible and cannot be CP-sequentially indifferentiable. Also, note that we cannot expect full indifferentiability to hold for the Chop construction, and in fact, not even sequential indifferentiability in the sense of [MPS12]. Indeed, a distinguisher A can simply first query $\mathsf{Prim}(\text{-}, y)$, obtaining x, and then query $\mathsf{Func}(x)$, that yields y'. Then, A just checks that the first r bits of y equals y', and if so outputs 1, and otherwise outputs 0. Note that in the real world, A always outputs 1, by the definition of Chop. However, in the ideal world, an arbitrary simulator Sim needs, on input y, to return an x for which the random oracle (to which it access) returns the first r bits of y. This is however infeasible if $n - r = \omega(\log\lambda)$, unless the simulator can make roughly 2^r queries.

The proof in Appendix B shows this problem vanishes for CP-sequential indifferentiability. Indeed, our simulator will respond to queries $\mathsf{Sim}(\text{-}, y)$ with a random (and inconsistent) x. The key point is that due to the random choice, it is unlikely that the distinguisher has already issued a prior query $\mathsf{Func}(x)$. Moreover, it is also unlikely (in the real world) that the distinguisher, after a query $\mathsf{Func}(x)$, makes an inverse query on $\pi(x)$. The combination of these two facts will be enough to imply the statement.

UCE SECURITY. We can now combine Theorem 4 with the fact that the Chop construction is (perfectly) query extractable, and use Theorem 2:

Corollary 1. *For all* n, r *such that* $n(\lambda) - r(\lambda) = \omega(\log\lambda)$*, if* F *is* $\mathsf{psPRP}[n, \mathcal{S}^{\star\mathsf{up}}]$*-secure, then* $\mathsf{Chop}[\mathsf{F}]$ *is* $\mathsf{UCE}[n, r, \mathcal{S}^{\star\mathsf{up}}]$*-secure, where* $\star \in \{\mathsf{c}, \mathsf{s}\}$*.*

The construction of [BHK14] can be used to obtain variable-input-length UCE: It first hashes the arbitrary-long input down to an $n(\lambda)$-bit long input using an almost-universal hash function, and then applies $\mathsf{Chop}[\mathsf{F}]$ to the resulting value.

5 Building psPRPs from UCEs

This section presents our main result on building psPRPs from UCEs, namely that the five-round Feistel construction, when its round functions are instantiated from a $\mathsf{UCE}[\mathcal{S}^{\star\mathsf{rs}}]$-secure function family (for $\star \in \{\mathsf{c}, \mathsf{s}\}$), yields a $\mathsf{psPRP}[\mathcal{S}^{\star\mathsf{rs}}]$-secure permutation family.

CP-INDIFFERENTIABILITY OF FEISTEL. Let $n : \mathbb{N} \to \mathbb{N}$ be a (polynomially bounded) function. We define the following construction Ψ_5, which, for security parameter λ, implements an invertible permutation on $2n(\lambda)$-bit strings, and makes calls to an oracle $f : [5] \times \{0,1\}^{n(\lambda)} \to \{0,1\}^{n(\lambda)}$. In particular, on input 1^λ and $X = X_0 \,\|\, X_1$, where $X_0, X_1 \in \{0,1\}^{n(\lambda)}$, running $\Psi_5^f(1^\lambda, (+, X))$ outputs $X_5 \,\|\, X_6$, where

$$X_{i+1} \leftarrow X_{i-1} \oplus f(i, X_i) \text{ for all } i = 1, \ldots, 5 \,. \tag{12}$$

Symmetrically, upon an inverse query, $\Psi_5^f(1^\lambda, (\text{-}, Y = X_5 \,\|\, X_6))$ simply computes the values backwards, and outputs $X_0 \,\|\, X_1$. Construction Ψ_5 is clearly $(\mathbf{R}^5_{n,n} \to \mathbf{P}_{2n})$-compatible, where we use

the notation $\mathbf{R}_{n,n}^k$ to denote the k-fold combination of independent random functions which takes queries of the form (i, x) that are answered by evaluating on x the i-th function.

The following theorem establishes CP-indifferentiability for Ψ_5. We discuss below its consequences, and give a detailed description of our simulation strategy. The full analysis of the simulation strategy – which employes the randomness-mapping technique of [HKT11] – is found in Appendix C.

Theorem 5 (CP-indifferentiability of Feistel). *Let* $\mathbf{R} = \mathbf{R}_{n,n}^5$ *and* $\mathbf{P} = \mathbf{P}_{2n}$. *Then, there exists a simulator* Sim *(described in Fig. 6) such that for all distinguisher* A *making at most* $q(\lambda)$ *queries,*

$$\mathsf{Adv}^{\mathsf{cpi}[\mathbf{R}\to\mathbf{P}]}_{\Psi_5,\mathsf{Sim},A}(\lambda) \leq \frac{360q(\lambda)^6}{2^{n(\lambda)}} \ . \tag{13}$$

Here, Sim *makes at most* $2q(\lambda)^2$ *queries, and otherwise runs in time polynomial in the number of queries answered, and* n. ■

This, together with Theorem 1, gives us immediately the following corollary: Given a keyed function family $\mathsf{F} = (\mathsf{F.Kg}, \mathsf{F.Eval})$, where for all $\lambda \in \mathbb{N}$, $k \in [\mathsf{F.Kg}(1^\lambda)]$, $\mathsf{F.Eval}(1^\lambda, k, \cdot)$ is a function from $n(\lambda)+3$ bits to $n(\lambda)$ bits, interpreted as a function $[5]\times\{0,1\}^{n(\lambda)} \to \{0,1\}^{n(\lambda)}$, then define the keyed function family $\Psi_5[\mathsf{F}] = (\Psi.\mathsf{Kg}, \Psi.\mathsf{Eval})$ obtained by instantiating the round function using F.

Corollary 2. *For any polynomially bounded* $n = \omega(\log\lambda)$, *if* $\mathsf{F} \in \mathsf{UCE}[n+3, \mathcal{S}^{\star\mathsf{rs}}]$, *then* $\Psi_5[\mathsf{F}] \in \mathsf{psPRP}[2n, \mathcal{S}^{\star\mathsf{rs}}]$, *where* $\star \in \{\mathsf{c}, \mathsf{s}\}$. ■

Remarks. Theorem 5 is interesting in its own right, as part of the line of works on (full-fledged) indifferentiability of Feistel constructions. Coron et al. [CHK+16] show that six rounds are necessary for achieving indifferentiability, and proofs of indifferentiability have been given for 14, 10, and 8 rounds, respectively [HKT11,CHK+16,DSKT16,DS16]. Thus, our result shows that CP-indifferentiability is a strictly weaker goal in terms of round-complexity of the Feistel construction. (Also for sequential indifferentiability as in [MPS12], six rounds are necessary.) As we will see in the next paragraph, our simulation strategy departs substantially from earlier proofs.

Two obvious problems remain open. First off, we know four rounds are necessary (as they are needed for indistinguishability alone [LR88]), but we were unable to make any progress on whether CP-sequential indifferentiability (or psPRP security) is achievable. The second is the case of unpredictable sources. We note that a heavily unbalanced Feistel construction (where each round function outputs one bit) would be query extractable, as the input of the round function leaves little uncertainty on the inner state, and the extractor can evaluate the round functions for other rounds to infer the input/output of the construction. Thus, if we could prove CP-indifferentiabilty, we could combine this with Theorem 2. Unfortunately, such a proof appears beyond our current understanding.

Simulator description. We explain now our simulation strategy, which is described formally in Fig. 6. We note that our approach inherits the chain-completion technique from previous proofs, but it will differ substantially in how and when chains are completed.

Recall that in the ideal case, in the first stage of the CP-indifferentiability game, A_1 makes queries to Func implementing a random permutation, and then passes the control of the game to A_2 which interacts with Sim. Our Sim maintains tables G_k for $k \in [5]$ to simulate the round functions. We denote by $G_k[X] = \perp$ that the table entry for X is undefined, and we assume

```
PROCEDURE Sim(k, X):
1:  if G_k[X] = ⊥ then
2:    if k = 2 then
3:      F^inner(k, X)
4:      foreach (X_1, X_2) ∈ G_1 × {X} do
5:        if (X_1, X_2, 1) ∉ CompletedChains then
6:          X_0 ← F^inner(1, X_1) ⊕ X_2
7:          (X_5, X_6) ← Func(+, X_0||X_1)
8:          C ← (X_1, X_2, 1)
9:          if G_5[X_5] ≠ ⊥ then                          // Immediate Completion
10:           Complete(C, (X_5, X_6))
11:         else                                          // Completion is delayed
12:           X_3 ← F^inner(2, X_2) ⊕ X_1
13:           Chains[3, X_3] ← (5, X_5), Chains[5, X_5] ∪← {(C, (X_5, X_6))}
14:   elseif k = 4 then
15:     F^inner(k, X)
16:     foreach (X_4, X_5) ∈ {X} × G_5 do
17:       if (X_4, X_5, 4) ∉ CompletedChains then
18:         X_6 ← F^inner(4, X_4) ⊕ X_5
19:         (X_0, X_1) ← Func(-, X_5||X_6)
20:         C ← (X_4, X_5, 4)
21:         if G_1[X_1] ≠ ⊥ then                          // Immediate Completion
22:           Complete(C, (X_0, X_1))
23:         else                                          // Completion is delayed
24:           X_3 ← F^inner(4, X_4) ⊕ X_5
25:           Chains[3, X_3] ← (1, X_1), Chains[1, X_1] ∪← {(C, (X_0, X_1))}
26:   elseif k ∈ {1, 5} then
27:     F^inner(k, X)
28:     foreach (C, (U, V)) ∈ Chains[k, X] do
29:       if C ∉ CompletedChains then                     // Delayed Completion
30:         Complete(C, (U, V))
31:   elseif Chains[3, X] ≠ ⊥ then
32:     Sim(Chains[k, X])
33: return F^inner(k, X)
```

Fig. 6: The code for simulator Sim. Sim has access to the Func oracle and maintains data structures G_k, Chains and CompletedChains as global variables.

all values are initially undefined. Also, we refer to a tuple (X_k, X_{k+1}, k) as a *partial chain* where $G_k[X_k] \neq \perp$ and $G_{k+1}[X_{k+1}] \neq \perp$ for $k \in \{1, 4\}$, $X_k, X_{k+1} \in \{0,1\}^n$.

For any query (k, X) by A_2, Sim checks if $G_k[X] = \perp$. If not then the image $G_k[X]$ is returned. Otherwise, depending on the value of k, Sim takes specific steps as shown in Fig. 6. If $k \in \{2, 4\}$ then Sim sets $G_k[X]$ to a uniformly random n-bit string by calling the procedure $\mathsf{F}^{\text{inner}}$. At this point, Sim considers newly formed tuples $(X_1, X_2) \in G_1 \times \{X\}$ (when $k = 2$) and detects partial chains $C = (X_1, X_2, 1)$. The notation $X_1 \in G_1$ is equivalent to $G_1[X_1] \neq \perp$. For every partial chain C that Sim detects, it queries Func on (X_0, X_1) and gets (X_5, X_6) where $X_0 = G_1[X_1] \oplus X_2$. If (X_0, X_1) does not appear in one of the queries/responses by/to A_1 then it is unlikely for A_2 to guess the corresponding (X_5, X_6) pair. Therefore, if $G_5[X_5] \neq \perp$ then Sim assumes that C is a chain that most likely corresponds to a query by A_1. We refer to partial chains that correspond to the queries by A_1 as *relevan*t chains. In this case, Sim *immediately* completes C by calling the

PROCEDURE $\mathsf{F}^{\text{inner}}(i, X_i)$:
34: **if** $G_i[X_i] = \perp$ **then**
35: $G_i[X_i] \leftarrow\!\!\$\ \{0,1\}^n$
36: **return** $G_i[X_i]$

PROCEDURE $\mathsf{ForceVal}(X, Y, l)$:
37: $G_l[X] \leftarrow Y$

PROCEDURE $\mathsf{Complete}(C, (U, V))$:
38: $(X, Y, i) \leftarrow C$
39: **if** $i = 1$ **then**
40: $X_1 \leftarrow X, X_2 \leftarrow Y,\ X_3 \leftarrow \mathsf{F}^{\text{inner}}(2, X_2) \oplus X_1$
41: $(X_5, X_6) \leftarrow (U, V)$
42: $X_4 \leftarrow \mathsf{F}^{\text{inner}}(5, X_5) \oplus X_6$
43: $\mathsf{ForceVal}(X_3, X_4 \oplus X_2, 3)$, $\mathsf{ForceVal}(X_4, X_5 \oplus X_3, 4)$
44: **elseif** $i = 4$ **then**
45: $X_4 \leftarrow X, X_5 \leftarrow Y,\ X_3 \leftarrow \mathsf{F}^{\text{inner}}(4, X_4) \oplus X_5$
46: $(X_0, X_1) \leftarrow (U, V)$
47: $X_2 \leftarrow \mathsf{F}^{\text{inner}}(1, X_1) \oplus X_0$
48: $\mathsf{ForceVal}(X_3, X_4 \oplus X_2, 3)$, $\mathsf{ForceVal}(X_2, X_1 \oplus X_3, 2)$
49: CompletedChains $\overset{\cup}{\leftarrow} \{(X_1, X_2, 1), (X_4, X_5, 4)\}$

Fig. 6: (continued) The code for subroutines used by simulator Sim.

procedure Complete. C is completed by forcing the values of $G_3[X_3]$ and $G_4[X_4]$ to be consistent with the Func query where $X_3 \leftarrow G_2[X_2] \oplus X_1$ and $X_4 \leftarrow G_5[X_5] \oplus X_6$.

If $G_5[X_5] = \perp$ then either C is not a relevant chain or C is a relevant chain but A_2 has not queried $(5, X_5)$ yet. An aggressive strategy would be to complete C, thereby asking Sim to complete every partial chain ever detected. The resulting simulation strategy will however end up potentially managing an exponential number of partial chains, contradicting our goal of efficient simulation. Hence, Sim *delays* the completion and only completes C on A_2's query to either $(3, X_3)$ or $(5, X_5)$ where $X_3 = G_2[X_2] \oplus X_1$. The completion is delayed by storing information about X_3 and X_5, that fall on the chain C, in the table Chains. In particular, Sim stores a pointer to $(5, X_5)$ at Chains$[3, X_3]$. The inputs $((X_1, X_2, 1), (X_5, X_6))$ to the Complete call on C are stored in Chains$[5, X_5]$. As many chains can share the same X_5, we allow Chains$[5, X_5]$ to be a set. The idea of delaying the chain completions is unique to our simulation strategy and it translates to an efficient Sim which consistently completes chains in the eyes of A. Sim works symmetrically when $k = 4$.

For queries of the form (k, X) where $k \in \{1, 5\}$, Sim always assigns $G_k[X]$ to a uniform random n-bit string by calling $\mathsf{F}^{\text{inner}}$. Moreover as discussed earlier, X could be on previously detected partial chains whose completion was delayed. Therefore after the assignment, Sim picks up all partial chains C' (if any) stored in Chains$[k, X]$ and completes them. This is where Sim captures a relevant partial chain which was delayed for completion. Finally for queries $(3, X)$, Sim checks if this X was on a partial chain that was detected but not completed. If Chains$[3, X] = \perp$ then Sim assigns $G_3[X]$ a uniform random n-bit string otherwise it follows the pointer to Chains$[3, X]$ to complete the chain X was on. Since Chains$[3, X]$ just stores a tuple (instead of a set) there can be at most one chain C that Chains$[3, X]$ can point to at any time. In the execution, Chains$[3, X]$

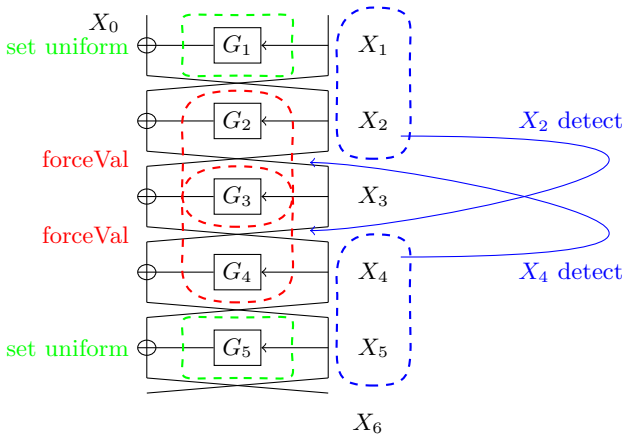

Fig. 7: The 5-round Feistel where Sim sets $G_1[X_1]$ and $G_5[X_5]$ uniformly at random (green). Sim detects chains at either (X_1, X_2) or (X_4, X_5) (blue) and adapts at (X_3, X_4) and (X_2, X_3) respectively (red).

can get overwritten which may lead to inconsistencies in chain completions. However, we show that there are no overwrites in either tables G_k or the data-structure Chains, except with negligible probability. This allows Sim to Complete chains consistently in the eyes of A. Furthermore, to avoid completing the same chains again, Sim maintains a set of all CompletedChains and completes any chain if it is not in CompletedChains. A pictorial description of Sim is found in Fig. 7.

5.1 Proof of Theorem 5

In this Section, we outline the proof of sequential indifferentiability of the 5-round Feistel network Ψ_5. First, we develop some notation and describe the hybrids used in the proof. The detailed analysis of the indistinguishability of the hybrids can be found in Appendix C.

For the entire proof we will let n denote the security parameter and we omit it from definitions of games for ease of notation. To prove our claim, we fix a deterministic distinguisher $A = (A_1, A_2)$ that makes at most q queries overall, i.e., we sum the number of queries in its first and second stage. [9] For ease of description, we further assume that A_1 makes $q_c \leq q$ (construction) queries and denote its queries as $(\sigma, X^i_k, X^i_{k+1})$ for $(\sigma, k) \in \{(+, 0), (-, 5)\}$ and $i \in [q_c]$. We refer to these queries as relevant queries. From now on, any query described with superscript i must be understood as either the query by A_1 or its response. We also assume that A_2 issues primitive queries corresponding to all construction queries by A_1. This can be achieved by allowing A_2 run the procedure AllPrimitive described in Figure 8 at the end and blowing up the number of queries only by a factor of 5.

Our proof considers four games denoted as G_i for $i \in [4]$ which are described in Figures 8, 9. Furthermore we define the quantity $\Delta(\mathsf{G}_i, \mathsf{G}_{i+1})$,

$$\Delta(\mathsf{G}_i, \mathsf{G}_{i+1}) = |\mathsf{Pr}[\mathsf{G}_i] - \mathsf{Pr}[\mathsf{G}_{i+1}]| \ . \tag{14}$$

We describe our games next.

[9] We may assume A is deterministic without loss of generality, as the advantage of any probabilistic distinguisher is at most the advantage of the corresponding deterministic distinguisher obtained by optimally fixing its random tape.

MAIN $\mathsf{G}_1^A(n)$:
$P^{+}, P^{-} \leftarrow \phi$
$\mathsf{st} \leftarrow\$\, A_1^{\mathsf{Func}}(1^n)$
$b' \leftarrow\$\, A_2^{\mathsf{Prim}}(1^n, \mathsf{st})$
return b'

ORACLE $\mathsf{Prim}(k, X)$:
$Y \leftarrow\$\, \mathsf{Sim}(k, X)$
return Y

ORACLE $\mathsf{Func}(+, X_0||X_1)$:
if $P[+, X_0, X_1] = \perp$ **then**
  $(X_5, X_6) \leftarrow\$\, \{0,1\}^n \times \{0,1\}^n \setminus P^{-}$
  $P^{-} \xleftarrow{\cup} \{(X_5, X_6)\}$, $P^{+} \xleftarrow{\cup} \{(X_0, X_1)\}$
  $P[+, X_0, X_1] \leftarrow (X_5, X_6)$
  $P[-, X_5, X_6] \leftarrow (X_0, X_1)$
return $P[+, X_0, X_1]$

ORACLE $\mathsf{Func}(-, X_5||X_6)$
if $P[-, X_5, X_6] = \perp$ **then**
  $(X_0, X_1) \leftarrow\$\, \{0,1\}^n \times \{0,1\}^n \setminus P^{+}$
  $P^{-} \xleftarrow{\cup} \{(X_5, X_6)\}$, $P^{+} \xleftarrow{\cup} \{(X_0, X_1)\}$
  $P[+, X_0, X_1] \leftarrow (X_5, X_6)$
  $P[-, X_5, X_6] \leftarrow (X_0, X_1)$
return $P[-, X_5, X_6]$

MAIN $\mathsf{G}_2^A(n)$:
$(f, \rho) \leftarrow\$\, \mathcal{F} \times \mathcal{R}$
$\mathsf{st} \leftarrow A_1^{\mathsf{Func}}(1^n)$
$b' \leftarrow A_2^{\mathsf{Prim}}(1^n, \mathsf{st})$
AllComplete()
return b'

ORACLE $\mathsf{Prim}(k, X)$:
$Y \leftarrow \mathsf{Sim}(f)(k, X)$
return Y

ORACLE $\mathsf{Func}(+, X_0||X_1)$
if $P[+, X_0, X_1] = \perp$ **then**
  $(X_5, X_6) \leftarrow \rho[+, X_0, X_1]$
  $P[+, X_0, X_1] \leftarrow (X_5, X_6)$
  $P[-, X_5, X_6] \leftarrow (X_0, X_1)$
return $P[+, X_0, X_1]$

ORACLE $\mathsf{Func}(-, X_5||X_6)$
if $P[-, X_5, X_6] = \perp$ **then**
  $(X_0, X_1) \leftarrow \rho[-, X_5, X_6]$
  $P[-, X_5, X_6] \leftarrow (X_0, X_1)$
  $P[+, X_0, X_1] \leftarrow (X_5, X_6)$
return $P[-, X_5, X_6]$

PROCEDURE AllPrimitive():
foreach $i \in [q_c]$ **do**
  $V \leftarrow X_1^i, U \leftarrow X_0^i$
  foreach $i \in [5]$ **do**
    $Y \leftarrow \mathsf{Prim}(i, V) \oplus U$
    $U \leftarrow V$, $V \leftarrow Y$

PROCEDURE AllComplete():
foreach $(k, X) \in$ Chains.keys() **do**
  if $k \in \{1, 5\}$ **then**
    $\mathsf{F}^{\text{inner}}(k, X)$
    foreach $(C, (U, V)) \in \text{Chains}[k, X]$ **do**
      if $C \notin$ CompletedChains **then**
        Complete$(C, (U, V))$

Fig. 8: (Top) Game G_1 is described on the left along with the Prim and Func oracles. Game G_2 is defined on the right along with the Prim and Func oracles where the randomness f, ρ is sampled uniformly at the beginning of the game. (Bottom) The procedure AllPrimitive (run by A_2) making primitive queries corresponding to all construction queries by A_1. Here, (X_0^i, X_1^i) (passed to A_2 through st) is either the ith query by A_1 or its response by Func. The procedure AllComplete (run by Sim) at the end of G_2 issues calls to Complete for all incomplete chains whose completion was delayed. T.keys() returns the set of all keys in the hashtable T.

GAME G_1: G_1 described in Figure 8 is exactly the case when $b = 0$ in the CP-sequential indifferentiability game of Ψ_5. The Func oracle implements a permutation via lazy sampling and Prim oracle invokes the procedure $\mathsf{Sim}(k, X)$ on queries (k, X). In addition, Sim inherits oracle access to Func from Prim.

GAME G_2: REPLACING PERMUTATION WITH A TWO-SIDED RANDOM FUNCTION: In this game, we modify the oracle Func such that now instead of a permutation it acts like a two-sided random

function working with pre-sampled randomness ρ. The description of Func for game G_2 is given in Figure 8. Func maintains a table P to simulate the permutation. For a (forward) query $(+, X_0||X_1)$ to Func, if $P[+, X_0, X_1]$ is already defined then the image $P[+, X_0, X_1]$ is returned as the response. Otherwise, Func gets a $2n$-bit string (X_5, X_6) from $\rho[+, X_0, X_1]$ and maps (X_0, X_1) to (X_5, X_6) and vice versa. This is shown in Figure 8. Similarly for a (backward) query $(-, X_5||X_6)$ to Func such that $P[-, X_5, X_6]$ is not defined Func gets a $2n$-bit string from $\rho[-, X_5, X_6]$ and updates P accordingly. The random table ρ, therefore, contains a $2n$-bit string for all $(+, X_0, X_1)$ and $(-, X_5, X_6)$.

Moreover, we also sample the randomness f used by Sim at the beginning of the game G_2. The table f contains an n-bit string for all indices (k, X) where $k \in [5]$ and $X \in \{0,1\}^n$. We refer to the simulator with presampled randomness f as $\mathsf{Sim}(f)$. Whenever Sim lazily samples randomness to assign $G_k[X]$ (line 37 of Figure 6), $\mathsf{Sim}(f)$ instead accesses $f[k, X]$. Sampling f at the beginning does not change the distribution of the simulation. This is because Sim considers an entry of f at most once. The code for $\mathsf{Sim}(f)$ is shown in Figure 6 alongside the code of Sim.

After A_2 has output its guess b', we allow Sim to run an additional procedure called $\mathsf{AllComplete}$ (described in Figure 8). We refer to the execution of G_2 until the point when A_2 outputs its guess as Phase 1 and the remaining execution as Phase 2. The output guess of A_2 is independent of Phase 2 as A_2 learns nothing about it. Note that in Phase 2, Sim does not make any calls to Func, it just completes incomplete chains that were delayed. This extension to Phase 2 is necessary to argue about the indistinguishability of G_2 with G_3 (defined next).

By $\mathcal{F}$, $\mathcal{R}$ we denote the set of all tables f and ρ respectively as described above. We consider an alternative game $\mathsf{G}_2^{f,\rho}$ where the first step of sampling random tables from game G_2 is omitted and instead we use (f, ρ) as the tables. In other words, game $\mathsf{G}_2^{f,\rho}$ can be thought of as running G_2 with the fixed (f, ρ).

GAME G_3: REPLACING TWO-SIDED RANDOM FUNCTION WITH $\Psi_5(h)$: In G_3, we replace the two-sided random function with the Feistel construction $\Psi_5(h)$ i.e. Func now evaluates the Feistel construction to reply to its queries. Like in the previous game, we again sample the randomness $h \in \mathcal{F}$ at the beginning of the game. One major difference between G_2 and G_3 is that in G_3 the randomness h is shared between Prim (and hence Sim) and Func oracles. The game G_3 maintains tables H_k which have the same semantics as G_k. As shown in Figure 9, whenever Func needs to access the value of the kth round function on X it accesses $H_k[X]$ if it is defined. Otherwise, it sets $H_k[X] \leftarrow h[k, X]$ and then accesses $H_k[X]$. Sim does the same when it needs to set $G_k[X]$. The entire working of Sim is the same as in G_2 (and G_1) except the procedure $\mathsf{F}^{\text{inner}}$ which is shown in Figure 9. Like in G_2, we allow Sim to run the procedure $\mathsf{AllComplete}$ after A_2 has output its guess.

We consider an alternative game G_3^h where the first step of the sampling random table h is omitted from game G_3 and instead we use h as the table. In other words, game G_3^h can be thought of as running G_3 with the fixed h.

GAME G_4: REMOVING THE SIMULATOR Sim: In this game, we modify Prim such that it behaves like a random function as in the case of $b = 1$ in the CP-sequential indifferentiability game of the five-round Feistel construction. The Func oracle which implements the Feistel five-round construction has access to Prim for round functions.

From the choice of our games, it follows that $\mathsf{Adv}^{\mathsf{cpi}[\mathbf{R}\to\mathbf{P}]}_{\Psi_5,\mathsf{Sim},A}(n) \le \Delta(\mathsf{G}_1, \mathsf{G}_4)$. Our focus from now on will be to derive $\Delta(\mathsf{G}_i, \mathsf{G}_{i+1})$ for $i \in [3]$ and then by the triangle inequality we derive a bound on $\Delta(\mathsf{G}_1, \mathsf{G}_4)$. We refer the reader to Appendix C for a detailed analysis.

MAIN $\mathsf{G}_3^A(n)$:
$h \leftarrow\!\!\$\ \mathcal{F}$
$\mathsf{st} \leftarrow A_1^{\mathsf{Func}}(1^n)$
$b' \leftarrow A_2^{\mathsf{Prim}}(1^n, \mathsf{st})$
AllComplete()
return b'

ORACLE $\mathsf{Func}(+, X_0||X_1)$:
if $P[+, X_0, X_1] = \perp$ **then**
  $U \leftarrow X_0$, $V \leftarrow X_1$
  foreach $i \in [1,2,3,4,5]$ **do**
    if $H_i[V] = \perp$ **then**
      $H_i[V] \leftarrow h[i, X]$
    $X \leftarrow H_i[V] \oplus U$
    $U \leftarrow V$, $V \leftarrow X$
  $P[+, X_0, X_1] \leftarrow (U, V)$
  $P[-, U, V] \leftarrow (X_0, X_1)$
return $P[+, X_0, X_1]$

ORACLE $\mathsf{Prim}(k, X)$:
$Y \leftarrow \mathsf{Sim}(k, X)$
return Y

PROCEDURE $\mathsf{F}^{\mathrm{inner}}(k, X)$:
if $G_k[X] = \perp$ **then**
  if $H_k[X] = \perp$ **then**
    $H_k[X] \leftarrow h[k, X]$
  $G_k[X] \leftarrow H_k[X]$
return $G_k[X]$

ORACLE $\mathsf{Func}(-, X_5||X_6)$:
if $P[-, X_5, X_6] = \perp$ **then**
  $V \leftarrow X_5$, $U \leftarrow X_6$
  foreach $i \in [5,4,3,2,1]$ **do**
    if $H_i[V] = \perp$ **then**
      $H_i[V] \leftarrow h[i, X]$
    $A \leftarrow H_i[V] \oplus U$
    $U \leftarrow V$, $V \leftarrow A$
  $P[-, X_5, X_6] \leftarrow (U, V)$
  $P[+, U, V] \leftarrow (X_5, X_6)$
return $P[-, X_5, X_6]$

MAIN $\mathsf{G}_4^A(n)$:
$\mathsf{st} \leftarrow\!\!\$\ A_1^{\mathsf{Func}}(1^n)$
$b' \leftarrow\!\!\$\ A_2^{\mathsf{Prim}}(1^n, \mathsf{st})$
return b'

ORACLE $\mathsf{Func}(+, X_0||X_1)$:
if $P[+, X_0, X_1] = \perp$ **then**
  $U \leftarrow X_0$, $V \leftarrow X_1$
  foreach $i \in [1,2,3,4,5]$ **do**
    $X \leftarrow\!\!\$\ \mathsf{Prim}(i, V) \oplus U$
    $U \leftarrow V$, $V \leftarrow X$
  $P[+, X_0, X_1] \leftarrow (U, V)$
  $P[-, U, V] \leftarrow (X_0, X_1)$
return $P[+, X_0, X_1]$

ORACLE $\mathsf{Prim}(k, X)$:
if $G_k[X] = \perp$ **then**
  $G_k[X] \leftarrow\!\!\$\ \{0,1\}^n$
return $G_k[X]$

ORACLE $\mathsf{Func}(-, X_5||X_6)$:
if $P[-, X_5, X_6] = \perp$ **then**
  $V \leftarrow X_5$, $U \leftarrow X_6$
  foreach $i \in [5,4,3,2,1]$ **do**
    $X \leftarrow\!\!\$\ \mathsf{Prim}(i, V) \oplus U$
    $U \leftarrow V$, $V \leftarrow X$
  $P[-, X_5, X_6] \leftarrow (U, V)$
  $P[+, U, V] \leftarrow (X_5, X_6)$
return $P[-, X_5, X_6]$

Fig. 9: (Top) Game G_3 with its oracles Prim and Func. The Prim oracle behaves exactly like Sim as described in Figure 6 except the $\mathsf{F}^{\mathrm{inner}}$ procedure which is shown here. (Bottom) Game G_4 with its oracle Prim and Func.

6 Ideal-model psPRP Constructions

We discuss two natural approaches to instantiate psPRPs. One is by taking any block cipher, and using its key as a (now public) seed. The second is by using a key-less permutation (e.g., the one within SHA-3), and adding the seed through the Even-Mansour [EM97] construction. While the purpose of our psPRP framework is to remove ideal-model assumptions, the only obvious way to validate (heuristically) these methods *is* via ideal-model proofs, and this is what we do here. Also, note that such ideal-model proofs are useful since, as in the case of UCE, psPRP security can become a powerful intermediate notion within ideal-model proofs for multi-stage security games [Mit14].

PsPRPs from block ciphers. Given a family of block ciphers $\mathsf{E}_\lambda : \{0,1\}^{s(\lambda)} \times \{0,1\}^\lambda \to \{0,1\}^\lambda$, we consider the construction $\mathsf{F} = (\mathsf{F.Kg}, \mathsf{F.Eval})$, where $\mathsf{F.Kg}(1^\lambda)$ outputs a random $k \leftarrow\!\!\$\, \{0,1\}^{s(\lambda)}$, whereas

$$\mathsf{F.Eval}(1^\lambda, (k, +, x)) = \mathsf{E}(k, x) \ , \quad \mathsf{F.Eval}(1^\lambda, (k, -, y)) = \mathsf{E}^{-1}(k, y) \ . \tag{15}$$

The following theorem establishes its security in the ideal cipher model, that is, we assume (without overloading notation) that all parties (i.e., the source, the distinguisher, and the reset adversary in the proof) are given access to a randomly chosen block cipher E, which is also used within F. We refer the reader to Appendix D.1 for the proof which closely follows the proof from [BHK13] that a random oracle is UCE-secure.

Theorem 6 (Ideal Cipher as a psPRP). *Let* $\mathbf{P}$ *be the random permutation with input length* $n(\lambda) = \lambda$*. For every source-distinguisher pair* S, D*, where the source* S*, in its first stage, outputs* n *which is at most* $N(\lambda)$ *and makes* q Prim *queries to its oracle, there exists* R *(described in the proof) such that*

$$\mathsf{Adv}^{\mathsf{psPRP}[n]}_{\mathsf{F},S,D}(\lambda) \le \mathsf{Adv}^{\mathsf{reset}[\mathbf{P}]}_{S,R}(\lambda) + \frac{2qN(\lambda)}{2^{s(\lambda)}} + \frac{2N(\lambda)^2}{2^{s(\lambda)}} \ . \tag{16}$$

In particular, if D *is polynomial time, then so is* R*.* ■

Even-Mansour. We find it practically valuable to assess whether simple constructions can work. To this end, here, we show that the Even-Mansour construction [EM97] yields a $\mathsf{psPRP}[\mathcal{S}^{\mathsf{cup}}]$-secure permutation family in the random permutation model. In particular, we assume we are given a family of permutations $\Pi = \{\pi_\lambda\}_{\lambda\in\mathbb{N}}$, where $\pi_\lambda : \{0,1\}^\lambda \to \{0,1\}^\lambda$, and consider the construction $\mathsf{EM} = (\mathsf{EM.Kg}, \mathsf{EM.Eval})$ as defined in Section 4.1. Similar to the above, the following theorem implicitly assumes all parties are given oracle access to a random permutation which is used to sample the permutation inside EM.

Below, we give a few remarks on why the above approach to show $\mathsf{psPRP}[\mathcal{S}^{\mathsf{crs}}]$ security does not extend to the case of Even-Mansour.

Theorem 7 (Even-Mansour as a psPRP). *Let* $\mathbf{P}$ *be the random permutation with input length* $n(\lambda) = \lambda$*. For every source-distinguisher pair* S, D*, where the source* S*, in its first stage, outputs* n *which is at most* $N(\lambda)$ *and where* S *and* D *jointly make at most* q *queries to their oracles, there exists* P *(described in the proof) such that*

$$\mathsf{Adv}^{\mathsf{psPRP}[n]}_{\mathsf{EM},S,D}(\lambda) \le \mathsf{Adv}^{\mathsf{pred}[\mathbf{P}]}_{S,P}(\lambda) + \frac{3q^2}{2^\lambda} + \frac{2N(\lambda)q^2}{2^\lambda} \ . \tag{17}$$

In particular, if D *runs in polynomial time, then so does* P*.* ■

The proof of Theorem 7 is in Appendix D.2. It resembles the original indistinguishability proof from [EM97], which bounds the advantage via the probability of an intersection query, that is, a direct query (by the source or by the distinguisher) to the random permutation that overlaps with one of the queries to the random permutation made internally by oracle $\mathcal{O}$ invoked by the source. Bounding the probability that S makes an intersection query proceeds as in [EM97] (exploiting lack of knowledge of the seed), whereas bounding the probability that D makes such a query requires a reduction to unpredictability.

WHY NOT RESET-SECURE SOURCES? We would like to extend Theorem 7 to $\mathcal{S}^{\mathsf{crs}}$, as this would provide validation for the assumption from Section 4.1. While we conjecture this to be true, the statement seems to evade a simple proof. The proof approach behind Theorem 6 fails in particular, as it heavily exploits the property that for each distinct seed, the construction F queries a disjoint portion of the domain of the ideal cipher, which is not true for EM.

7 Efficient Garbling from psPRPs

As an application of the psPRP framework, we study the security of the efficient garbling schemes of Bellare, Hoang, Keelveedhi, and Rogaway [BHKR13], and in particular, their simplest scheme (called Ga). It follows Yao's general garbling paradigm [Yao86], but proposes a particular gadget to garble individual gates that only relies on evaluating the underlying block cipher on a fixed key. In terms of efficiency, this has been shown to be advantageous, as it avoids higher re-keying costs. However, its security has only been proved in the ideal-cipher model, and recent work by Gueron et al. [GLNP15] has debated this. Here, we show that a minor variant of Ga (which still largely benefits from the lack of re-keying) is secure assuming the underlying block cipher is $\mathsf{psPRP}[\mathcal{S}^{\mathsf{sup}}]$-secure. While this assumption is undoubtedly strong, it makes it clear what is expected from the permutation. In particular, the main concern of [GLNP15] is the existence of fixed-key distinguishers (as in [KR07]), but these do not seem to affect psPRP-security, while they may invalidate the permutation being ideal.

SIMPLE CIRCUIT DESCRIPTION. For representing circuits we adopt the SCD notation of [BHR12]. A circuit is a 6-tuple $f = (n, m, q, W_1, W_2, G)$ where $n \geq 2$ is the number of inputs, $m \geq 1$ is the number of outputs, $q \geq 1$ is the number of gates, and $n + q$ is the number of wires. We let $\mathsf{Inputs} = [1, \ldots, n]$, $\mathsf{Wires} = [1, \ldots, n + q]$, $\mathsf{OutputWires} = [n + q - m + 1, \ldots, n + q]$ and $\mathsf{Gates} = [n + 1, \ldots, n + q]$. Then $W_1 : \mathsf{Gates} \to \mathsf{Wires} \setminus \mathsf{OutputWires}$ is a function to identify each gate's first incoming wire and $W_2 : \mathsf{Gates} \to \mathsf{Wires} \setminus \mathsf{OutputWires}$ to identify each gate's second incoming wire. Finally $G : \mathsf{Gates} \times \{0,1\}^2 \to \{0,1\}$ is a function that determines the functionality of each gate i.e. G_g is a table storing the output of gate g with input i and j at $G_g[i, j]$. We require that $W_1(g) < W_2(g) < g$ for all gates g.

Following [BHKR13], our definitions will be parameterized by the side information about the circuit obtained from its garbled counterpart. We consider the *topology side information* ϕ_{topo} which maps f to its topology $\phi_{\mathsf{topo}}(f) = (n, m, q, W_1, W_2)$. Another example is ϕ_{xor}, which maps f to a circuit $\phi_{\mathsf{xor}}(f) = (n, m, q, W_1, W_2, G')$ which obscures the functionality of non-xor gates. As shown in [BHR12] and [BHKR13], an important property is that ϕ_{topo} and ϕ_{xor} are efficiently invertible, i.e., there exists an efficient algorithm which given $\phi(f)$ and y, outputs (f', x') such that $\phi(f) = \phi(f')$ and $y = \mathsf{ev}(f', x')$.

GARBLING SCHEMES AND THEIR SECURITY. To describe a garbling scheme we use the notation from [BHR12]. A *garbling scheme* is a tuple of algorithms $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$. The algorithm Gb is probabilistic and others are determinisitic. Gb takes as inputs a circuit $f = (n, m, q, W_1, W_2, G)$ represented in the SCD notation and a security parameter 1^λ and returns a tuple of strings (F, e, d) where F is the garbled circuit, e is the input encoding information and d is the output decoding information. $\mathsf{En}(e, \cdot) : \{0,1\}^n \to \{0,1\}^*$ transforms the n-bit input x to the garbled input X. $\mathsf{Ev}(F, \cdot) : \{0,1\}^* \to \{0,1\}^*$ runs the garbled circuit F on garbled input X and returns the garbled output Y. $\mathsf{De}(d, \cdot) : \{0,1\}^* \to \{0,1\}^m \cup \{\perp\}$ decodes the garbled output Y to return $y \in \{0,1\}^m$. The algorithm ev is the canonical circuit-evaluation function where $\mathsf{ev}(f, x)$ is the m-bit output one

MAIN $\mathsf{PrvInd}^A_{\mathcal{G},\phi}(1^\lambda)$	PROCEDURE $\mathsf{Garble}(f_0, f_1, x_0, x_1)$
$b \leftarrow\!\$\, \{0,1\}$	**if** $\phi(f_0) \neq \phi(f_1)$ **then return** $\bot$
$b' \leftarrow\!\$\, A^{\mathsf{Garble}}(1^\lambda)$	**if** $\{x_0, x_1\} \not\subseteq \{0,1\}^{f_0.n}$ **then return** $\bot$
return $(b' = b)$	**if** $\mathsf{ev}(f_0, x_0) \neq \mathsf{ev}(f_1, x_1)$ **then return** $\bot$
	$(F, e, d) \leftarrow\!\$\, \mathsf{Gb}(1^\lambda, f_b)$, $X \leftarrow \mathsf{En}(e, x_b)$
	return (F, X, d)

Fig. 10: $\mathsf{PrvInd}^A_{\mathcal{G},\phi}$ Game for $\mathcal{G}$ with adversary A.

gets by feeding x to f. Finally, we require that $\mathcal{G}$ is *correct*, that is, if $f \in \{0,1\}^*$, $\lambda \in \mathbb{N}$, $x \in \{0,1\}^n$ and $(F, e, d) \in [\mathsf{Gb}(1^\lambda, f)]$, then $\mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, x))) = \mathsf{ev}(f, x)$. We require that all algorithms run in time polynomial in the security parameter λ.

In this work, we are only concerned with indistinguishability-based privacy, as defined in Game $\mathsf{PrvInd}^A_{\mathcal{G},\phi}$ in Fig. 10. Since both ϕ_{topo} and ϕ_{xor} are efficiently invertible, [BHR12] show that it is sufficient to focus on this target, since simulation-based security is implied. We say that $\mathcal{G}$ is prvind-*secure over side information function* ϕ if for all PPT adversaries A,

$$\mathsf{Adv}^{\mathsf{prvind}[\phi]}_{\mathcal{G},A}(\lambda) = 2\Pr\left[\mathsf{PrvInd}^A_{\mathcal{G},\phi}(\lambda)\right] - 1 \tag{18}$$

is negligible (in λ). Also, it is not hard to see (by a simple hybrid argument) that it is sufficient to prove this for adversaries which make one single query to their oracle.

GARBLING SCHEME $\mathsf{Ga}[\mathsf{P}]$. Our garbling scheme resembles heavily that of [BHKR13]. The only modification is that we assume it uses a function family P meant to be $\mathsf{psPRP}[\mathcal{S}^{\mathsf{sup}}]$-secure (which could be instantiated from a block cipher, by letting the key take the role of the seed.). During the garbling procedure, a fresh seed for P is chosen and made part of the garbled circuit. Clearly, re-keying costs are still largely avoided (especially for large circuits), even though re-keying is necessary when garbling multiple circuits.

Concretely, the garbling scheme $\mathsf{Ga}[\mathsf{P}]$ (Fig. 11) is a tuple of algorithms $\mathsf{Ga}[\mathsf{P}] = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, \mathsf{ev})$ where $\mathsf{P} \in \mathsf{psPRP}[k(\lambda), \mathcal{S}^{\mathsf{sup}}]$ for some polynomial $k(\lambda)$. Algorithm Gb transforms the input circuit f to a tuple of strings (F, e, d) where the seed s for the permutation P sampled independently for each input f is now part of F. Though the Ga scheme of [BHKR13] comes in several variants, where each variant is defined by the dual-key cipher used, we focus on a specific dual-key cipher (namely A1 in [BHKR13]) that leads to the most efficient implementation of Ga.

In the following theorem we prove the prvind-security of $\mathsf{Ga}[\mathsf{P}]$ and later discuss about the more efficient schemes GaX and GaXR from [BHKR13].

Theorem 8 (Garbling from psPRPs). *Let* $\mathsf{P} \in \mathsf{psPRP}[k(\lambda), \mathcal{S}^{\mathsf{sup}}]$ *then* $\mathsf{Ga}[\mathsf{P}]$ *is* prvind-*secure over* ϕ_{topo}. ■

Proof. We start with an overview of the proof, and details are given further below. Let us assume that $\mathsf{Ga}[\mathsf{P}]$ is not prvind-secure then there exists a PPT adversary A that issues circuits with at most $q(\lambda)$ gates and achieves a non-negligible advantage $\varepsilon(\lambda)$ in the $\mathsf{PrvInd}^A_{\mathsf{Ga}[\mathsf{P}],\phi}(\lambda)$ [10] game. Using A we construct a pair (S, D) (Fig. 12) breaking the psPRP security of P, where S is a statistically unpredictable source. Without loss of generality, we can assume that A queries its oracle exactly once.

[10] We drop the subscript topo from ϕ for ease of notation.

PROCEDURE $\mathsf{En}(e, x)$
$(X_1^0, X_1^1, \ldots, X_n^0, X_n^1) \leftarrow e$
$x_1, \ldots, x_n \leftarrow x$
$X \leftarrow (X_1^{x_1}, \ldots, X_n^{x_n})$
return X

PROCEDURE $\mathsf{Ev}(F, X)$
$(n, m, q, W_1, W_2, T, s) \leftarrow F$
$(X_1, \ldots, X_n) \leftarrow X$

foreach $g \in [n+1, \ldots, n+q]$ **do**
  $w_1 \leftarrow W_1(g), w_2 \leftarrow W_2(g)$
  $\alpha \leftarrow \mathrm{lsb}(X_{w_1})$, $\beta \leftarrow \mathrm{lsb}(X_{w_2})$
  $K \leftarrow X_{w_1} \oplus X_{w_2} \oplus g$
  $X_g \leftarrow T[g, \alpha, \beta] \oplus \mathsf{P.Eval}(s, K) \oplus K$
return $(X_{n+q-m+1}, \ldots, X_{n+q})$

PROCEDURE $\mathsf{De}(d, Y)$
$(d_1, \ldots, d_m) \leftarrow d$
$(Y_1, \ldots, Y_m) \leftarrow Y$
foreach $i \in [1, \ldots, m]$ **do**
  $y_i \leftarrow \mathrm{lsb}(Y_i) \oplus d_i$
return $y \leftarrow y_1, \ldots, y_m$

PROCEDURE $\mathsf{Gb}(1^\lambda, f)$
$s \leftarrow\!\$\, \mathsf{P.Kg}(1^\lambda)$
$(n, m, q, A', B', G) \leftarrow f$

foreach $i \in [1, \ldots, n+q]$ **do**
  $t \leftarrow\!\$\, \{0,1\}$
  $X_i^0 \leftarrow\!\$\, \{0,1\}^{k-1} t$
  $X_i^1 \leftarrow\!\$\, \{0,1\}^{k-1} \bar{t}$

foreach $g \in [n+1, \ldots, n+q]$ **do**
  $w_1 \leftarrow W_1(g), w_2 \leftarrow W_2(g)$
  foreach $(i, j) \in \{0,1\}^2$ **do**
    $A \leftarrow X_{w_1}^i, \alpha \leftarrow \mathrm{lsb}(A)$
    $B \leftarrow X_{w_2}^j, \beta \leftarrow \mathrm{lsb}(B)$
    $K \leftarrow X_{w_1}^i \oplus X_{w_2}^j \oplus g$
    $T[g, \alpha, \beta] \leftarrow \mathsf{P.Eval}(s, K) \oplus K \oplus X_g^{G_g[i,j]}$

$F \leftarrow (n, m, q, W_1, W_2, T, s)$
$e \leftarrow (X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$
$d \leftarrow (\mathrm{lsb}(X_{n+1-m+1}^0), \ldots, \mathrm{lsb}(X_{n+q}^0))$
return (F, e, d)

Fig. 11: Scheme $\mathsf{Ga}[\mathsf{P}]$.

Let c be the challenge bit in the psPR game for P, and let Perm be the oracle called by S. We allow S to sample the challenge bit b for the $\mathsf{PrvInd}^A_{\mathsf{Ga}[\mathsf{P}],\phi}$ game. Further, for syntactic reasons we decompose A into (A_0, A_1) where A_0 on input 1^λ outputs (f_0, x_0, f_1, x_1) (inputs for Garble) and forwards a state st to A_1. The result of Garble i.e. (F, X, d) is forwarded to A_1 to guess the challenge bit b in the $\mathsf{PrvInd}^A_{\mathsf{Ga}[\mathsf{P}],\phi}$ game. The source S nearly acts as Gb on input f_b. To satisfy unpredictability, the leakage L must give no information about the queries made by S. Therefore, S refrains from compiling the rows in the garbled table T which can be opened by A. S outputs this partially garbled circuit F^- as leakage in addition to (b, d, st). Moreover, since (S, D) must perfectly simulate the $\mathsf{PrvInd}^A_{\mathsf{Ga}[\mathsf{P}],\phi}$ game for A, leakage also contains the vector X^+ which is the set of all visible tokens (one for each wire). Given s and L, D completes the garbled circuit and invokes A_1 with appropriate inputs. D then outputs $b' \oplus b$ where b' was the guess of A in the $\mathsf{PrvInd}^A_{\mathsf{Ga}[\mathsf{P}],\phi}$ game.

It is easy to see that when $c = 1$, (S, D) simulate the game $\mathsf{PrvInd}^A_{\mathsf{Ga}[\mathsf{P}],\phi}$ for A. Furthermore, when $c = 0$ the leakage L can be transformed to be independent of the bit b by modifying Perm to act like a random function. This allows rows in the garbled table to be independent of the tokens $X_g^{G_g[i,j]}$ which might depend on bit b. Therefore, in this modified game A can do no better than guessing.

To prove that S is statistically unpredictable we need to show that any (possibly unbounded) predictor P making at most $p(\lambda)$ number of queries to the oracle Perm is unlikely to predict a query made by S given $L = (F^-, X^+, b, d, \mathsf{st})$. The idea is to swiftly transition to a game where L is independent of the queries made by S to Perm. This then reduces P to merely guess the queries. To achieve this, we take a similar path as the psPRP game of P ($c = 0$). We transition to a game

SOURCE $S^{\mathsf{Perm}}(1^\lambda)$:
$b \leftarrow\$ \{0,1\}$
$(f_0, x_0, f_1, x_1, \mathsf{st}) \leftarrow A_0(1^\lambda)$
$(n, m, q, W_1, W_2) \leftarrow \phi(f_b)$
$G \leftarrow f_b.G$
foreach $i \in [1, \ldots, n+q]$ **do**
  $v_i \leftarrow \mathsf{ev}(f_b, x_b, i)$; $t_i \leftarrow\$ \{0,1\}$
  $X_i^{v_i} \leftarrow\$ \{0,1\}^{k-1} t_i$
  $X_i^{\overline{v_i}} \leftarrow\$ \{0,1\}^{k-1} \overline{t_i}$
foreach $g \in [n+1, \ldots, n+q]$ **do**
  foreach $(i,j) \in \{0,1\}^2$ **do**
    $w_1 \leftarrow W_1(g)$, $w_2 \leftarrow W_2(g)$
    $A \leftarrow X_{w_1}^i$, $\alpha \leftarrow \mathrm{lsb}(A)$
    $B \leftarrow X_{w_2}^j$, $\beta \leftarrow \mathrm{lsb}(B)$
    $K \leftarrow A \oplus B \oplus g$
    if $i \neq v_{w_1} \vee j \neq v_{w_2}$ **then**
      $T[g, \alpha, \beta] \leftarrow \mathsf{Perm}(K) \oplus K \oplus X_g^{G_g[i,j]}$
$F^- \leftarrow (n, m, q, W_1, W_2, T)$
$X^+ \leftarrow (X_1^{v_1}, \ldots, X_{n+q}^{v_{n+q}})$
return $(F^-, X^+, d, b, \mathsf{st})$

DISTINGUISHER $D(1^\lambda, s, L)$:
$(F^-, X^+, d, b, st) \leftarrow L$
$(n, m, q, W_1, W_2, T) \leftarrow F^-$
$X_1, \ldots, X_{n+q} \leftarrow X^+$
for $g \in [n+1, \ldots, n+q]$ **do**
  $w_1 \leftarrow W_1(g)$, $w_2 \leftarrow W_2(g)$
  $\alpha \leftarrow \mathrm{lsb}(X_{w_1})$, $\beta \leftarrow \mathrm{lsb}(X_{w_2})$
  $K \leftarrow X_{w_1} \oplus X_{w_2} \oplus g$
  $T[g, \alpha, \beta] \leftarrow \mathsf{P.Eval}(s, K) \oplus K \oplus X_g$

$F \leftarrow (n, m, q, W_1, W_2, T, s)$
$X \leftarrow (X_1, \ldots, X_n)$
$b' \leftarrow A_1(1^\lambda, st, F, X, d)$
return $(b' \oplus b)$

Fig. 12: (S, D) in the psPR game of P where A_0's inputs are honest.

G_1 where F^- is independent of bit b. However, unlike the psPRP case, P and S share the same oracle Perm (to which P can also make inverse queries), and therefore it is non-trivial to argue about the independence of L and queries of S as we desire. Therefore, we make a final transition to a game G_2 where Perm returns random strings for queries by S and refrains from storing any information about the queries made by S. The resulting leakage can be viewed as being constructed by S without making any queries to Perm. Then we exploit the fact X^+ information theoretically hides X^- and hence queries by S are hidden from any P making only polynomially many queries to Perm. (We also note that this argument is implicitly contained in the original security proof in the ideal-cipher model.)

We now proceed with more details about the proof. Following up on what discussed above, it is easy to see that when S interacts with P ($c = 1$), A is exactly playing the $\mathsf{PrvInd}^A_{\mathsf{Ga}[\mathsf{P}],\phi}$ game. Therefore,

$$\begin{aligned} \Pr[D \Rightarrow 0 \mid c = 1] &= \frac{1}{2}\Pr[A_1 \Rightarrow 1 \mid b = 1 \wedge c = 1] + \frac{1}{2}\Pr[A_1 \Rightarrow 0 \mid b = 0 \wedge c = 1] \\ &= \frac{1}{2}\mathsf{Adv}^{\mathsf{prvind}[\phi]}_{\mathsf{Ga}[\mathsf{P}],A}(\lambda) + \frac{1}{2}\,. \end{aligned} \tag{19}$$

Let us now consider the case when $c = 0$ in the psPRP game of P i.e. S interacts with the random permutation making forward queries of the form $(+, K)$. We are concerned with $\Pr[D \Rightarrow 0 \mid c = 0]$. This quantity is precisely captured by $\Pr[\mathsf{G}_0]$ described in Figure 13 where to answer a forward query $(+, K)$, Perm samples a random string R and sets $R \oplus K$ as $T_P[+, K]$.[11] If $(+, K)$ was already queried before then G_0 sets bad to true and restores the value of R to be consistent. Moreover, if

[11] Note that the above method of maintaining a permutation might seem a bit strange, but it is easy to see that it is equivalent to sampling a random string U as the image of K and then setting $R \leftarrow U \oplus K$.

ORACLE $\mathsf{Perm}(+, K)$: // $\boxed{\mathsf{G}_0}$,G_1
$R \leftarrow\!\!\$\ \{0,1\}^\lambda$
if $(+, K) \in T_P$ **then**
  $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{R \leftarrow T_P[+, K] \oplus K}$
elseif $(-, R \oplus K) \in T_P$ **then**
  $\mathsf{bad} \leftarrow \mathsf{true}$
  $\boxed{T_P[+, K] \leftarrow\!\!\$\ \{0,1\}^n \setminus P^-}$
  $\boxed{R \leftarrow T_P[+, K] \oplus K}$
$T_P[+, K] \leftarrow R \oplus K$; $T_P[-, R \oplus K] \leftarrow K$
$P^+ \xleftarrow{\cup} \{K\}$; $P^- \xleftarrow{\cup} \{R \oplus K\}$
return $T_P[+, K]$

MAIN $\mathsf{G}_0, \mathsf{G}_1$:
$P^+ \leftarrow \{\}$; $P^- \leftarrow \{\}$
$L \leftarrow S^{\mathsf{Perm}}(1^\lambda)$
$c' \leftarrow D(1^\lambda, s, L)$
return c'

Fig. 13: In Game G_0, Perm behaves like a random permutation and sets bad to true only if S issues a query it had earlier queried or if Perm's choice of R is such that $R \oplus K$ was already sent as a reply. In Game G_1, Perm replies with uniform random string $R \oplus K$. The leakage generated by S in Game G_1 is therefore independent of the bit b. Note that since inverse queries are not made by S we do not describe the simulation of $\mathsf{Perm}(-, \cdot)$.

$(-, R \oplus K)$ was already chosen as a reply then G_0 again sets bad to true and picks a value of R to be consistent with a permutation. It is easy to see that Perm in G_0 behaves as a permutation implemented via lazy sampling. Hence, we have argued that $\Pr[D \Rightarrow 0 \mid c = 0] = \Pr[\mathsf{G}_0]$.

We introduce game G_1 described in Figure 13, where we modify the Perm oracle to not restore the value of R when bad is set to true. It is easy to see that G_0 and G_1 are identical until either game sets bad. However in Game G_1, the leakage L is independent of the bit b in the $\mathsf{PrvInd}^A_{\mathsf{Ga}[\mathsf{P}],\phi}$ game. This is because the entries $T[g, \alpha, \beta]$ constructed by S are now completely independent of the token $X_g^{G_g[i,j]}$ because of the uniform random replies by Perm, i.e, in particular

$$\begin{aligned} T[g, \alpha, \beta] &= \mathsf{Perm}(+, K) \oplus K \oplus X_g^{G_g[i,j]} \\ &= R \oplus K \oplus K \oplus X_g^{G_g[i,j]} \ . \end{aligned} \tag{20}$$

We can easily reformulate G_1[12] to reflect this independence but avoid describing it here. Therefore, $\Pr[\mathsf{G}_1] = 0.5$. Also note that since inverse queries are not made by S we do not describe the simulation of $\mathsf{Perm}(-, \cdot)$. Then,

$$\begin{aligned} \Pr[D \Rightarrow 0 \mid c = 0] &= \Pr[\mathsf{G}_0] \\ &\leq |\Pr[\mathsf{G}_0] - \Pr[\mathsf{G}_1]| + \Pr[\mathsf{G}_1] \\ &= \Pr[\mathsf{G}_1 \text{ sets } \mathsf{bad}] + \frac{1}{2} \ . \end{aligned} \tag{21}$$

Let's bound the probablility of bad being set to true. The game G_1 sets bad if either the query $(+, K)$ is such that it was previously queried by S or if $R \oplus K$ was already sent as a reply by Perm. Since S issues exactly $3q$ queries to Perm (q is the number of gates in circuits given by A_0), by the union bound over all queries we have that $\Pr[(-, R \oplus K) \in T_P]$ is at most $9q^2/2^k$.

[12] A reformulation similar to Fig.14 in [BHKR13] works.

Next, let us consider the probability of bad being set due to S's query $(\mathsf{+}, K) \in T_P$. Let us consider an element $(\mathsf{+}, K^*)$ in T_P. Then necessarily $K^* = A^* \oplus B^* \oplus g^*$. Let $K = A \oplus B \oplus g$. If $A^* = A$ and $B^* = B$ then necessarily $g \neq g^*$. Hence $K^* \neq K$. Therefore it must be the case that either $A \neq A^*$ or $B \neq B^*$. Without loss of generality, let $A \neq A^*$. Furthermore, all but the last bit of A is independent of B, A^*, B^* ($A[1, \ldots, k-1]$ given A^*, B, B^* is still uniform). Therefore $\mathsf{Pr}\left[K = K^*\right] \leq \frac{2}{2^k}$. By the union bound over all entries in T_P and over all queries by S, we have $\mathsf{Pr}\left[(\mathsf{+}, K) \in T_P\right] \leq \frac{2}{2^k} \cdot 9q^2$. By the union bound over the two events which set bad to true,

$$\Pr\left[\mathsf{G}_1 \text{ sets } \mathsf{bad}\right] \leq \frac{27q^2}{2^k} \ . \tag{22}$$

Therefore,

$$\begin{aligned} \mathsf{Adv}^{\mathsf{psPRP}[k]}_{\mathsf{P},S,D}(\lambda) &= \Pr\left[D \Rightarrow 0 \mid c = 1\right] - \Pr\left[D \Rightarrow 0 \mid c = 0\right] \\ &\geq \frac{1}{2}\mathsf{Adv}^{\mathsf{prvind}[\phi]}_{\mathsf{Ga}[\mathsf{P}],A}(\lambda) + \frac{1}{2} - \frac{27q^2}{2^k} - \frac{1}{2} \\ &= \frac{1}{2}\mathsf{Adv}^{\mathsf{prvind}[\phi]}_{\mathsf{Ga}[\mathsf{P}],A}(\lambda) - \frac{27q^2}{2^k} \end{aligned} \tag{23}$$

This contradicts the psPR security of P as long as S is a statistically unpredictable source. To prove that S is statistically unpredictable we need to show that any (possibly unbounded) predictor P making at most $p(\lambda)$ number of queries to the oracle Perm is unlikely to predict a query made by S even if it is allowed to output a set Q' of size at most $p'(\lambda)$ given $L = (F^{\mathsf{-}}, X^{\mathsf{+}}, b, d, \mathsf{st})$.

The Game $\mathsf{G_0}$ (Figure 14) is exactly the $\mathsf{Pred}^P_{\mathbf{P},S}(\lambda)$ where Perm implements the random permutation by lazy sampling. We modify the Game $\mathsf{G_0}$ similar to the case of the psPRP game when $c = 0$ (described above). The games $\mathsf{G_0}$ and $\mathsf{G_1}$ are identical until either game sets bad to true. Therefore by Equation 22 we have $\mathsf{Pr}\left[\mathsf{G_1} \text{ sets } \mathsf{bad}\right]$.

We know that L is independent of the bit b and may want to compute the success probability of P assuming that L gives no information about S's queries to P. However, unlike the psPRP case here P has access to the Perm oracle used by S to output the leakage L. It is not clear how to prove that L gives no information about the queries when both S and P are using the same oracle.

To handle this, we transition to a game $\mathsf{G_2}$ where until done is false the Perm oracle replies randomly like $\mathsf{G_1}$ but does not store the mapping. However it maintains a flag $\mathsf{bad_2}$ which is set to true if any of P's queries intersect with S's. It is easy to see that $\mathsf{G_1}$ and $\mathsf{G_2}$ are identical until $\mathsf{bad_2}$ is set to true. Note that in $\mathsf{G_2}$, S still accesses Perm to construct the leakage but since Perm does not store the map, we can conclude that P learns no information about the queries made by S except one bit of each invisible token inferred from $X^{\mathsf{+}}$. This allows us to look at $\mathsf{G_2}$ where P plays a guessing game where it is allowed to query the Perm oracle p times and has to guess one of the $3q$ queries and can output p' guesses.

$$\Pr\left[\mathsf{G_2}\right] \leq p'(\lambda) \cdot p(\lambda) \cdot 3q \left[\frac{2}{2^{k-1} - p}\right] \ . \tag{24}$$

The factor 2 in the numerator is due to the union bound over both types of queries $(\mathsf{+}, \cdot)$ and $(\mathsf{-}, \cdot)$ and the term $k-1$ in the denominator is to incorporate the one bit leakage of the tokens not visible to P. Then using $p \leq 2^{k-2}$,

MAIN $\mathsf{G}_0(\lambda)$,$\mathsf{G}_1(\lambda)$,$\mathsf{G}_2(\lambda)$:
$P^{+} \leftarrow \{\}$; $P^{-} \leftarrow \{\}$
$\mathsf{Bad}^{+} \leftarrow \{\}$; $\mathsf{Bad}^{-} \leftarrow \{\}$
done $\leftarrow$ false
$Q \leftarrow \{\}$
$L \leftarrow\$\ S^{\mathcal{O}}(1^{\lambda})$
done $\leftarrow$ true
$Q' \leftarrow\$\ P^{\mathcal{O}}(1^{\lambda}, L)$
return $(Q \cap Q' \neq \phi)$

ORACLE Perm$(+, K)$: // $\boxed{\mathsf{G}_0}$,G_1
if ¬done **then**
  $R \leftarrow\$\ \{0,1\}^k$
  if $(+, K) \in T_P$ **then**
    bad $\leftarrow$ true; $\boxed{R \leftarrow T_P[+, K] \oplus K}$
  elseif $(-, R \oplus K) \in T_P$ **then**
    bad $\leftarrow$ true
    $\boxed{T_P[+, K] \leftarrow\$\ \{0,1\}^k \setminus P^{-}}$
    $\boxed{R \leftarrow T_P[+, K] \oplus K}$
  $T_P[+, K] \leftarrow R \oplus K$; $P^{+} \xleftarrow{\cup} \{K\}$
  $T_P[-, R \oplus K] \leftarrow K$; $P^{-} \xleftarrow{\cup} \{R \oplus K\}$
  return $T_P[+, K]$
else
  if $(+, K) \notin T_P$ **then**
    $T_P[+, K] \leftarrow\$\ \{0,1\}^k \setminus P^{-}$
    $T_P[-, T_P[+, K]] \leftarrow \{K\}$
  $P^{+} \xleftarrow{\cup} \{K\}$; $P^{-} \xleftarrow{\cup} \{T_P[+, K]\}$
return $T_P[+, K]$

ORACLE Perm$(-, K)$: // $\boxed{\mathsf{G}_0}$,G_1
if $(-, K) \notin T_P$ **then**
  $T_P[-, K] \leftarrow\$\ \{0,1\}^k \setminus P^{+}$
  $T_P[+, T_P[-, K]] \leftarrow K$
  $P^{-} \xleftarrow{\cup} \{K\}$; $P^{+} \xleftarrow{\cup} \{T_P[-, K]\}$
return $T_P[-, K]$

ORACLE Perm$(+, K)$: // G_2
if ¬done **then**
  $R \leftarrow\$\ \{0,1\}^k$
  $\mathsf{Bad}^{+} \xleftarrow{\cup} \{K\}$
  $\mathsf{Bad}^{-} \xleftarrow{\cup} \{R \oplus K\}$
  return $T_P[+, K]$
else
  if $K \in \mathsf{Bad}^{+}$ **then** $\mathsf{bad}_2 \leftarrow$ true
  if $(+, K) \notin T_P$ **then**
    $T_P[+, K] \leftarrow\$\ \{0,1\}^k \setminus P^{-}$
    $T_P[-, T_P[+, K]] \leftarrow \{K\}$
  $P^{+} \xleftarrow{\cup} \{K\}$, $P^{-} \xleftarrow{\cup} \{T_P[+, K]\}$
return $T_P[+, K]$

ORACLE Perm$(-, K)$: // G_2
if $K \in \mathsf{Bad}^{-}$ **then** $\mathsf{bad}_2 \leftarrow$ true
if $(-, K) \notin T_P$ **then**
  $T_P[-, K] \leftarrow\$\ \{0,1\}^k \setminus P^{+}$
  $T_P[+, T_P[-, K]] \leftarrow K$
  $P^{-} \xleftarrow{\cup} \{K\}$, $P^{+} \xleftarrow{\cup} \{T_P[-, K]\}$
return $T_P[-, K]$

Fig. 14: Game G_0 is the unpredictability game. In Game G_1, Perm when interacting with S replies randomly but mimics a permutation when interacting with the predictor P. In G_2 the leakage is independent of the queries made by S.

$$\begin{aligned} \Pr[\mathsf{G}_2] &\leq p'(\lambda) \cdot p(\lambda) \cdot 3q \left[\frac{2}{2^{k-2}}\right] \\ &= 24 \cdot p'(\lambda) \cdot p(\lambda) \cdot \frac{q}{2^k} \ . \end{aligned} \tag{25}$$

It remains to argue that $\mathsf{Pr}[\mathsf{G}_2 \text{ sets } \mathsf{bad}_2] \leq \frac{24pq}{2^k}$. Then we will have,

$$\mathsf{Adv}^{\mathsf{pred}[\mathbf{P}]}_{S,P}(\lambda) \leq \frac{27q^2}{2^k} + \frac{24pq}{2^k} + \frac{24 \cdot pp'q}{2^k} \ . \tag{26}$$

Hence, S is statistically unpredictable.

The game G_2 sets bad_2 if P's query $(\sigma, K) \in \mathsf{Bad}^{\sigma}$ where $\sigma \in \{+, -\}$. Consider any $K \in \mathsf{Bad}^{+}$ which has a corresponding $R \oplus K \in \mathsf{Bad}^{-}$. Here, $K = A \oplus B \oplus g$ where at least one of A or B is invisible to P. Without loss of generality, let A be invisible. Then given L, all but the last bit of A

is uniformly random and the conditional probability that $K = \alpha$ is at most $1/2^{k-1}$ for any string $\alpha \in \{0,1\}^k$. Consider a query $(+, u)$ by P,

$$\mathsf{Pr}\left[u \text{ hits } K\right] \leq \frac{1}{2^{k-1} - p}\ , \tag{27}$$

where p are the number of queries by P to the permutation. Similar analysis holds for queries $(-, K)$. By the union bound over the S's queries and P's queries, the probability that G_2 sets bad_2,

$$\begin{aligned} \mathsf{Pr}\left[\mathsf{G}_2 \text{ sets } \mathsf{bad}_2\right] &\leq \frac{6pq}{2^{k-1} - p} \\ &\leq \frac{24pq}{2^k}\ . \end{aligned} \tag{28}$$

where the last inequality follows from $p \leq 2^{k-2}$. □

Related schemes. Along with Ga, [BHKR13] propose another scheme GaX which achieves faster garbling and evaluation times using the free-xor technique [KS08]. We consider a variant GaX[P] of GaX where (like Ga[P]) the permutation is replaced by a psPRP[$\mathcal{S}^{\mathsf{sup}}$]-secure permutation P and the seed for P is sampled freshly for every new instantiation of Gb. The security proof of GaX[P] almost readily follows from the security proof of GaX from [BHKR13] with slight modifications as done in the proof of Ga[P].

The third scheme proposed by [BHKR13], called GaXR, further improves over GaX in the size of the garbled table due to the use of row reduction technique at the cost of slower garbling and evaluation times. This means that for every gate, GaXR serves only three rows in the garbled table. Here, we note that adapting GaXR to be proved secure under a suitable psPRP assumption does not appear to have a simple and clear solution, and we leave this as an open problem.

Acknowledgments

We wish to thank John Retterer-Moore for his involvement in an earlier stage of this project. We also thank the EUROCRYPT '17 anonymous reviewers for their insightful feedback. This research was partially supported by NSF grants CNS-1423566, CNS-1528178, CNS-1553758 (CAREER), and IIS-152804, and by a Hellman Fellowship.

References

ABD+13. Elena Andreeva, Andrey Bogdanov, Yevgeniy Dodis, Bart Mennink, and John P. Steinberger. On the indifferentiability of key-alternating ciphers. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 531–550. Springer, Heidelberg, August 2013.

ABM14. Elena Andreeva, Andrey Bogdanov, and Bart Mennink. Towards understanding the known-key security of block ciphers. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 348–366. Springer, Heidelberg, March 2014.

ADMA15. Elena Andreeva, Joan Daemen, Bart Mennink, and Gilles Van Assche. Security of keyed sponge constructions using a modular proof approach. In Gregor Leander, editor, *FSE 2015*, volume 9054 of *LNCS*, pages 364–384. Springer, Heidelberg, March 2015.

AJN15. Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX8 and NORX16: Authenticated encryption for low-end systems. Cryptology ePrint Archive, Report 2015/1154, 2015. http://eprint.iacr.org/2015/1154.

BBT16. Mihir Bellare, Daniel J. Bernstein, and Stefano Tessaro. Hash-function based PRFs: AMAC and its multi-user security. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 566–595. Springer, Heidelberg, May 2016.

BDPV08. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 181–197. Springer, Heidelberg, April 2008.

BDPV10. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Sponge-based pseudo-random number generators. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 33–47. Springer, Heidelberg, August 2010.

BFM14. Christina Brzuska, Pooya Farshim, and Arno Mittelbach. Indistinguishability obfuscation and UCEs: The case of computationally unpredictable sources. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 188–205. Springer, Heidelberg, August 2014.

BGI+01. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

BH15. Mihir Bellare and Viet Tung Hoang. Resisting randomness subversion: Fast deterministic and hedged public-key encryption in the standard model. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 627–656. Springer, Heidelberg, April 2015.

BHK13. Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Instantiating random oracles via UCEs. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 398–415. Springer, Heidelberg, August 2013.

BHK14. Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Cryptography from compression functions: The UCE bridge to the ROM. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 169–187. Springer, Heidelberg, August 2014.

BHKR13. Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society Press, May 2013.

BHR12. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 784–796. ACM Press, October 2012.

BM14. Christina Brzuska and Arno Mittelbach. Using indistinguishability obfuscation via UCEs. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 122–141. Springer, Heidelberg, December 2014.

BR06. Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.

BS16. Mihir Bellare and Igors Stepanovs. Point-function obfuscation: A framework and generic constructions. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 565–594. Springer, Heidelberg, January 2016.

BST16. Mihir Bellare, Igors Stepanovs, and Stefano Tessaro. Contention in cryptoland: Obfuscation, leakage and UCE. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 542–564. Springer, Heidelberg, January 2016.

CDH+12. Donghoon Chang, Morris Dworkin, Seokhie Hong, John Kelsey, and Mridul Nandi. A keyed sponge construction with pseudorandomness in the standard model. In *Proceedings of the Third SHA-3 Candidate Conference*, 2012.

CDMP05. Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård revisited: How to construct a hash function. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 430–448. Springer, Heidelberg, August 2005.

CHK+16. Jean-Sébastien Coron, Thomas Holenstein, Robin Künzler, Jacques Patarin, Yannick Seurin, and Stefano Tessaro. How to build an ideal cipher: The indifferentiability of the Feistel construction. *Journal of Cryptology*, 29(1):61–114, January 2016.

DGG+15. Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. A formal treatment of backdoored pseudorandom generators. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 101–126. Springer, Heidelberg, April 2015.

DS16. Yuanxi Dai and John P. Steinberger. Indifferentiability of 8-round feistel networks. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 95–120. Springer, Heidelberg, August 2016.

DSKT16. Dana Dachman-Soled, Jonathan Katz, and Aishwarya Thiruvengadam. 10-round feistel is indifferentiable from an ideal cipher. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 649–678. Springer, Heidelberg, May 2016.

EM97. Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. *Journal of Cryptology*, 10(3):151–162, 1997.

FM16. Pooya Farshim and Arno Mittelbach. Modeling random oracles under unpredictable queries. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 453–473. Springer, Heidelberg, March 2016.

GLNP15. Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. Fast garbling of circuits under standard assumptions. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 567–578. ACM Press, October 2015.

GPT15. Peter Gazi, Krzysztof Pietrzak, and Stefano Tessaro. The exact PRF security of truncation: Tight bounds for keyed sponges and truncated CBC. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 368–387. Springer, Heidelberg, August 2015.

GT15. Peter Gazi and Stefano Tessaro. Secret-key cryptography from ideal primitives: A systematic overview. In *2015 IEEE Information Theory Workshop, ITW 2015, Jerusalem, Israel, April 26 - May 1, 2015*, pages 1–5. IEEE, 2015.

GT16. Peter Gazi and Stefano Tessaro. Provably robust sponge-based PRNGs and KDFs. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 87–116. Springer, Heidelberg, May 2016.

HKT11. Thomas Holenstein, Robin Künzler, and Stefano Tessaro. The equivalence of the random oracle model and the ideal cipher model, revisited. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 89–98. ACM Press, June 2011.

KR07. Lars R. Knudsen and Vincent Rijmen. Known-key distinguishers for some block ciphers. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 315–324. Springer, Heidelberg, December 2007.

KS08. Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, Heidelberg, July 2008.

LR88. Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2), 1988.

MH14. Takahiro Matsuda and Goichiro Hanaoka. Chosen ciphertext security via UCE. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 56–76. Springer, Heidelberg, March 2014.

Mit14. Arno Mittelbach. Salvaging indifferentiability in a multi-stage setting. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 603–621. Springer, Heidelberg, May 2014.

MMH+14. Nicky Mouha, Bart Mennink, Anthony Van Herrewege, Dai Watanabe, Bart Preneel, and Ingrid Verbauwhede. Chaskey: An efficient MAC algorithm for 32-bit microcontrollers. In Antoine Joux and Amr M. Youssef, editors, *SAC 2014*, volume 8781 of *LNCS*, pages 306–323. Springer, Heidelberg, August 2014.

MPS12. Avradip Mandal, Jacques Patarin, and Yannick Seurin. On the public indifferentiability and correlation intractability of the 6-round Feistel construction. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 285–302. Springer, Heidelberg, March 2012.

MRH04. Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Heidelberg, February 2004.

RS08. Phillip Rogaway and John P. Steinberger. Security/efficiency tradeoffs for permutation-based hashing. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 220–236. Springer, Heidelberg, April 2008.

RSS11. Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 487–506. Springer, Heidelberg, May 2011.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

Zha16. Mark Zhandry. The magic of ELFs. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 479–508. Springer, Heidelberg, August 2016.

A Proofs for Section 3

A.1 Multi-user Indifferentiability

It will also be notationally convenient, for our proofs below, to define a multi-user version of the CP-indifferentiability game, where an additional stage A_0 first chooses the number of instances it intends to attack. This is formalized by the game muCP at the bottom of Fig. 15, for which we define analogously the quantity $\mathsf{Adv}^{\mathsf{m-cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},A}(\lambda)$. Then, the following lemma follows by a simple hybrid argument, and its proof is omitted.

$\underline{\text{MAIN } \mathsf{muCP}[\mathbf{I}\to\mathbf{J}]^{A}_{\mathsf{M},\mathsf{Sim}}(\lambda)}$:
$(1^n, \mathsf{st}_0) \leftarrow\$ A_0(1^\lambda)$; $b \leftarrow\$ \{0,1\}$
$f_1, \ldots, f_n \leftarrow\$ \mathbf{I}_\lambda$
$g_1, \ldots, g_n \leftarrow\$ \mathbf{J}_\lambda$
$\mathsf{st}_1 \leftarrow\$ A_1^{\mathsf{Func}}(1^\lambda, \mathsf{st}_0)$
$b' \leftarrow\$ A_2^{\mathsf{Prim}}(1^\lambda, \mathsf{st}_1)$
return $b' = b$

$\underline{\text{ORACLE } \mathsf{Func}(i, \mathbf{x})}$:
if $b = 1$ **then**
  return $\mathsf{M}^{f_i}(\mathbf{x})$
else
  return $g_i(\mathbf{x})$

$\underline{\text{ORACLE } \mathsf{Prim}(i, \mathbf{u})}$:
if $b = 1$ **then**
  return $f_i(\mathbf{u})$
else
  return $\mathsf{Sim}^{g_i}(\mathbf{u})$

Fig. 15: Game muCP used to define the $\mathsf{m-cpi}$-security for M. Here, Sim is the simulator and $A = (A_0, A_1, A_2)$ is the two-stage distinguisher with an additional A_0 stage to choose the number of instances.

Lemma 1 (Single-user to multi-user CP-indifferentiability). *Let* M *be a* $(\mathbf{I}\to\mathbf{J})$*-compatible construction. For all distinguishers* $A = (A_0, A_1, A_2)$*, such that on input* 1^λ *the output of* A_0 *is bounded by* $N(\lambda)$*, there exists a distinguisher* $A' = (A'_1, A'_2)$ *(given explicitly in the proof) such that for all* $\lambda \in \mathbb{N}$*,*

$$\mathsf{Adv}^{\mathsf{m-cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},A}(\lambda) \le N(\lambda) \cdot \mathsf{Adv}^{\mathsf{cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},A'}(\lambda)\ , \tag{29}$$

In particular, if A *is PPT,* $\mathbf{I}$*,* $\mathbf{J}$ *are efficiently implementable, and* M *is also polynomial-time, then* A' *is polynomial time.* ■

A.2 Proof of Theorem 1

Proof (Theorem 1). The proof follows the lines of [BHK14]. Consider a source-distinguisher pair S, D using at most $N(\lambda)$ keys in the game $\mathsf{psPR}^{S,D}_{\mathsf{M}[\mathsf{F}],\mathbf{J}}(\lambda)$ for $\mathsf{M}[\mathsf{F}]$.

As the first step, we construct a new source-distinguisher pair $(\overline{S}, \overline{D})$ for the game $\mathsf{psPR}^{\overline{S},\overline{D}}_{\mathsf{F},\mathbf{I}}(\lambda)$. In particular, we first let $D = \overline{D}$ (note that the seed for $\mathsf{M}[\mathsf{F}]$ is a seed for F, so this is a syntactically correct distinguisher). Moreover, $\overline{S}$, is described on the left of Fig. 16, and works as follows: given access to its oracle $\mathcal{O}$, $\overline{S}$ simply runs S with a simulated oracle $\mathcal{O}'$. A query $(i, \mathbf{x})$ for $i \in [n]$ made by S to $\mathcal{O}'$ is answered by running construction M, where each oracle query z made by M is in turn answered by $\mathcal{O}(i, z)$. Finally, when S outputs leakage $L \in \{0,1\}^*$, $\overline{S}$ simply makes L its own output.

We can then fairly easily show that

$$\mathsf{Adv}^{\mathsf{pspr}[\mathbf{J}]}_{\mathsf{M}[\mathsf{F}],S,D}(\lambda) \le \mathsf{Adv}^{\mathsf{pspr}[\mathbf{I}]}_{\mathsf{F},\overline{S},\overline{D}}(\lambda) + \mathsf{Adv}^{\mathsf{m-cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},A'}(\lambda)\ ,$$

SOURCE $\overline{S}(1^\lambda, \varepsilon)$:
$(1^n, \mathsf{st}) \xleftarrow{\$} S(1^\lambda, \varepsilon)$
return $(1^n, \mathsf{st})$

SOURCE $\overline{S}^{\mathcal{O}}(1^\lambda, \mathsf{st})$:
// $\mathsf{st} \neq \varepsilon$
$L \xleftarrow{\$} S^{\mathcal{O}'}(1^\lambda, \mathsf{st})$
return L

ORACLE $\mathcal{O}'(i, x)$:
$y \leftarrow \mathsf{M}^{\mathcal{O}(i,\cdot)}(x)$
return y

ADV. $A'_0(1^\lambda)$:
$(1^n, \mathsf{st}) \xleftarrow{\$} S(1^\lambda, \varepsilon)$
return $(1^n, (1^n, \mathsf{st}))$

ADV. $A'_1(1^\lambda, (1^n, \mathsf{st}))$:
$L \xleftarrow{\$} S^{\mathsf{Func}}(1^\lambda, \mathsf{st})$
return $(1^n, L)$

ADV. $A'_2(1^\lambda, (1^n, L))$:
$k_1, \ldots, k_n \xleftarrow{\$} \mathsf{F.Kg}(1^\lambda)$
$b' \xleftarrow{\$} D(1^\lambda, L, k_1, \ldots, k_n)$
return b'

ADV. $R'^{\mathcal{O}}(1^\lambda, 1^n, L)$:
$b' \xleftarrow{\$} R^{\mathcal{O}'}(1^\lambda, 1^n, L)$
return $(1^n, (1^n, \mathsf{st}))$

ORACLE $\mathcal{O}'(i, x)$:
$y \xleftarrow{\$} \mathsf{Sim}_i^{\mathcal{O}(i,\cdot)}(x)$
return y

Fig. 16: Pseudocode descriptions for the proof of Theorem 1. ORACLE Func is as described in Fig. 15 for the muCP game of M. Due to space constraints we use ADV. instead of ADVERSARY.

for an adversary A' we specify in Figure 16 (and explain shortly) and *any* simulator Sim. Indeed, note that the only difference between $\mathsf{psPR}^{\overline{S},\overline{D}}_{\mathsf{F},\mathbf{I}}(\lambda)$ and $\mathsf{psPR}^{S,D}_{\mathsf{M}[\mathsf{F}],\mathbf{J}}(\lambda)$ is that in the *ideal* case, S accesses independent instances of $\mathbf{J}$, whereas in the former S (simulated within $\overline{S}$) has access to independent instances of $\mathsf{M}^{\mathbf{I}}$, i.e., each instance has M query an independent instance of $\mathbf{I}$. Therefore, it is sufficient to consider the multi-user CP-indifferentiability adversary A' as described in Figure 16.

Further, using Lemma 1, we can build from A' an adversary A such that

$$\mathsf{Adv}^{\mathsf{pspr}[\mathbf{J}]}_{\mathsf{M}[\mathsf{F}],S,D}(\lambda) \leq \mathsf{Adv}^{\mathsf{pspr}[\mathbf{I}]}_{\mathsf{F},\overline{S},\overline{D}}(\lambda) + N(\lambda) \cdot \mathsf{Adv}^{\mathsf{cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},A}(\lambda) \ ,$$

given A' always outputs (in its first stage) n such that $n \leq N(\lambda)$ on input 1^λ.

Relating reset-security. It remains to relate the reset-security of S and $\overline{S}$, which is the core of the proof. In particular, fix an arbitrary R for the game $\mathsf{Reset}^R_{\mathbf{I},\overline{S}}(\lambda)$. Then, we are going to build R' for $\mathsf{Reset}^{R'}_{\mathbf{J},S}(\lambda)$ as follows. (A more formal description is on the right of Figure 16.) R' runs R with the given leakage L, however rather than giving R direct access to its oracle $\mathcal{O}$, R is run with a simulated oracle $\mathcal{O}'$. To do this, R' runs $N(\lambda)$ independent instances of the simulator Sim – call them $\mathsf{Sim}_1, \ldots, \mathsf{Sim}_{N(\lambda)}$. A query $\mathcal{O}'(i, x)$ is then answered by querying x to Sim_i, where each of Sim's queries y is answered as $\mathcal{O}(i, y)$.

To continue with our analysis, we denote as $\mathsf{Reset}^R_{\mathbf{I},\overline{S}}(\lambda, c)$ and $\mathsf{Reset}^{R'}_{\mathbf{J},S}(\lambda, c)$ for $c \in \{0, 1\}$, the games obtained from the original games by fixing the challenge b to c, and the game output is the

ADV. $C_0^{(1)}(1^\lambda)$: $(1^n, \mathsf{st}) \leftarrow\!\!\$\, S(1^\lambda, \varepsilon)$ **return** $(1^n, (1^n, \mathsf{st}))$	ADV. $C_0^{(2)}(1^\lambda)$: $(1^n, \mathsf{st}) \leftarrow\!\!\$\, S(1^\lambda, \varepsilon)$ **return** $(1^n, (1^n, \mathsf{st}))$	ADV. $C_0^{(2)}(1^\lambda)$: $(1^n, \mathsf{st}) \leftarrow\!\!\$\, S(1^\lambda, \varepsilon)$ **return** $(1^n, (1^n, \mathsf{st}))$
ADV. $C_1^{(1)}(1^\lambda, (1^n, \mathsf{st}))$: $L \xleftarrow{\$} S^{\mathsf{Func}}(1^\lambda, \mathsf{st})$ **return** $(1^n, L)$	ADV. $C_1^{(2)}(1^\lambda, (1^n, \mathsf{st}))$: $g_1, \ldots, g_n \xleftarrow{\$} \mathbf{J}_\lambda$ $L \xleftarrow{\$} S^{\mathcal{O}''}(1^\lambda, \mathsf{st})$ **return** $(1^n, L)$	ADV. $C_1^{(2)}(1^\lambda, (1^n, \mathsf{st}))$: $L \xleftarrow{\$} S^{\mathsf{Func}}(1^\lambda, \mathsf{st})$ **return** $(1^n, L)$
ADV. $C_2^{(1)}(1^\lambda, (1^n, L))$: $f_1, \ldots, f_n \xleftarrow{\$} \mathbf{I}_\lambda$ $b' \leftarrow R^{\mathcal{O}'}(1^n, L)$ **return** b'	ADV. $C_2^{(2)}(1^\lambda, (1^n, L))$: $b' \leftarrow R^{\mathsf{Prim}}(1^n, L)$ **return** b'	ADV. $C_2^{(2)}(1^\lambda, (1^n, L))$: $b' \leftarrow R^{\mathsf{Prim}}(1^n, L)$ **return** $1 - b'$
ORACLE $\mathcal{O}'(i, x)$: $y \xleftarrow{\$} f_i(x)$ **return** y	ORACLE $\mathcal{O}''(i, x)$: $y \xleftarrow{\$} g_i(x)$ **return** y	

Fig. 17: Pseudocode descriptions for the proof of Theorem 1. ORACLES Func and Prim are as described in Figure 15 for the muCP game of M. Due to space constraints we use ADV. instead of ADVERSARY.

adversary's output bit b'. Then, by a standard argument,

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{reset}[\mathbf{I}]}_{\overline{S},R}(\lambda) &= \Pr\left[\mathsf{Reset}^{R}_{\mathbf{I},\overline{S}}(\lambda,1)\right] - \Pr\left[\mathsf{Reset}^{R}_{\mathbf{I},\overline{S}}(\lambda,0)\right] \\
&= \Pr\left[\mathsf{Reset}^{R}_{\mathbf{I},\overline{S}}(\lambda,1)\right] - \Pr\left[\mathsf{Reset}^{R'}_{\mathbf{J},S}(\lambda,1)\right] \\
&\quad + \Pr\left[\mathsf{Reset}^{R'}_{\mathbf{J},S}(\lambda,1)\right] - \Pr\left[\mathsf{Reset}^{R'}_{\mathbf{J},S}(\lambda,0)\right] \\
&\quad + \Pr\left[\mathsf{Reset}^{R'}_{\mathbf{J},S}(\lambda,0)\right] - \Pr\left[\mathsf{Reset}^{R}_{\mathbf{I},\overline{S}}(\lambda,0)\right] \\
&= \Pr\left[\mathsf{Reset}^{R}_{\mathbf{I},\overline{S}}(\lambda,1)\right] - \Pr\left[\mathsf{Reset}^{R'}_{\mathbf{J},S}(\lambda,1)\right] \\
&\quad + \mathsf{Adv}^{\mathsf{reset}[\mathbf{J}]}_{S,R'}(\lambda) + \Pr\left[\mathsf{Reset}^{R'}_{\mathbf{J},S}(\lambda,0)\right] - \Pr\left[\mathsf{Reset}^{R}_{\mathbf{I},\overline{S}}(\lambda,0)\right]
\end{aligned} \tag{30}$$

We are now going to build a new (multi-user) CP-sequential indifferentiability adversary $\overline{C}$ such that

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{m-cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},\overline{C}}(\lambda) = \frac{1}{3}\Big[&\Pr\left[\mathsf{Reset}^{R}_{\mathbf{I},\overline{S}}(\lambda,1)\right] - \Pr\left[\mathsf{Reset}^{R'}_{\mathbf{J},S}(\lambda,1)\right] \\
&+ \Pr\left[\mathsf{Reset}^{R'}_{\mathbf{J},S}(\lambda,0)\right] - \Pr\left[\mathsf{Reset}^{R}_{\mathbf{I},\overline{S}}(\lambda,0)\right]\Big] .
\end{aligned} \tag{31}$$

The adversary C' proceeds as follows. Initially, it selects $i \xleftarrow{\$} \{1,2,3\}$ uniformly at random, and then runs $C^{(i)}$. The adversaries $C^{(1)}, C^{(2)}$, and $C^{(3)}$ are described in Figure 17. Then, clearly

$$\mathsf{Adv}^{\mathsf{m-cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},C'}(\lambda) = \frac{1}{3}\sum_{i=1}^{3}\mathsf{Adv}^{\mathsf{m-cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},C^{(i)}}(\lambda)$$

MAIN $\mathsf{G}_0(\lambda)$,$\mathsf{G}_1(\lambda)$:	ORACLE $\mathsf{Func}(i,x)$: // G_0	ORACLE $\mathsf{Func}(i,x)$: // G_1
$\mathsf{done} \leftarrow \mathsf{false}$	$Q_{\mathsf{M}} \xleftarrow{\cup} \{x\}$	$Q_{\mathsf{M}} \xleftarrow{\cup} \{x\}$
$Q_{\mathbf{I}} \leftarrow \emptyset$	$y \leftarrow \mathsf{M}^{\mathsf{Prim}(i,\cdot)}(x)$	$y \leftarrow g_i(x)$
$(1^n, t) \leftarrow\!\$\, S(1^\lambda, \varepsilon)$	**return** y	**return** y
$f_1, \ldots, f_n \leftarrow\!\$\, \mathbf{I}_\lambda$	ORACLE $\mathsf{Prim}(i,x)$: // G_0	ORACLE $\mathsf{Prim}(i,x)$: // G_1
$g_1, \ldots, g_n \leftarrow\!\$\, \mathbf{J}_\lambda$	**if** $\neg\mathsf{done}$ **then** $Q_{\mathbf{I}} \xleftarrow{\cup} \{x\}$	**if** $\neg\mathsf{done}$ **then** $Q_{\mathbf{I}} \xleftarrow{\cup} \{x\}$
$L \leftarrow\!\$\, S^{\mathsf{Func}}(1^\lambda, t)$	**return** $f_i(x)$	**return** $\mathsf{Sim}_i^{g_i}(x)$
$\mathsf{done} \leftarrow \mathsf{true}$		
$Q \leftarrow\!\$\, P^{\mathsf{Prim}}(1^\lambda, 1^n, L)$		
$Q^* \leftarrow\!\$\, \mathsf{Ext}^{\mathsf{Prim}}(Q)$		
return $(Q_{\mathsf{M}} \cap Q^* \neq \emptyset)$		

Fig. 18: Games used in the proof of Theorem 2.

To expand this expression, let us first introduce a hybrid experiment $\overline{\mathsf{Reset}}^R_{\mathbf{I},S}(\lambda)$ where in the first stage, S interacts with independent copies of $\mathbf{J}$, and in the second stage, R interacts with independent copies of $\mathbf{I}$. Then,

$$\mathsf{Adv}^{\mathsf{m-cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},C^{(1)}}(\lambda) = \mathsf{Pr}\left[\mathsf{Reset}^R_{\mathbf{I},\overline{S}}(\lambda,1)\right] - \mathsf{Pr}\left[\overline{\mathsf{Reset}}^R_{\mathbf{I},S}(\lambda)\right] .$$

Further,

$$\mathsf{Adv}^{\mathsf{m-cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},C^{(2)}}(\lambda) = \mathsf{Pr}\left[\overline{\mathsf{Reset}}^R_{\mathbf{I},S}(\lambda)\right] - \mathsf{Pr}\left[\mathsf{Reset}^{R'}_{\mathbf{J},S}(\lambda,1)\right] .$$

Finally, it is not hard to see that

$$\mathsf{Adv}^{\mathsf{m-cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},C^{(3)}}(\lambda) = \mathsf{Pr}\left[\mathsf{Reset}^{R'}_{\mathbf{J},S}(\lambda,0)\right] - \mathsf{Pr}\left[\mathsf{Reset}^R_{\mathbf{I},\overline{S}}(\lambda,0)\right] .$$

Note that in $C^{(3)}$ the flipping of the bits is necessary because the execution of $\mathsf{Reset}^{R'}_{\mathbf{J},S}(\lambda,0)$ corresponds to the ideal world in the indifferentiability experiment, whereas $\mathsf{Reset}^R_{\mathbf{I},\overline{S}}(\lambda,0)$ corresponds to the real world, and thus we have to switch signs. To conclude the proof, we can apply Lemma 1 to obtain, from C', an attacker against the single-user CP-indifferentiability of C. □

A.3 Proof of Theorem 2

Proof (Theorem 2). Note that (8) is established exactly as in the proof of Theorem 1; the definitions of $\overline{S}$ and $\overline{D}$ are identical, and the statement is independent of the specific properties of the source.

Therefore, the rest of this proof relates the unpredictability of S with that of $\overline{S}$. In particular, given P, we construct P' as follows. On input 1^λ, 1^n and L, P' runs $P(1^\lambda, 1^n, L)$, but answers its oracle calls (i,x) with $\mathsf{Sim}^{\mathcal{O}(i,\cdot)}(x)$, i.e., using the simulator Sim. When finally P outputs Q, P' runs $\mathsf{Ext}^{\mathcal{O}}(Q)$, and outputs its output Q^*.

To establish (9), we consider two games, G_0 and G_1, as depicted in Figure 18. Game G_0 is the same as game $\mathsf{Pred}^P_{\mathbf{I},\overline{S}}(\lambda)$, except that the winning condition is set when Q^* extracted by Ext contains one of the queries to Func made by S, rather than one of the Prim queries made by M while evaluating S's queries. Therefore, the fact that $\mathsf{Pr}[A] \le \mathsf{Pr}[B] + \mathsf{Pr}[A \wedge \neg B]$ for all events A, B yields

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{pred}[\mathbf{I}]}_{\overline{S},P}(\lambda) &= \mathsf{Pr}\left[Q_{\mathbf{I}} \cap Q \neq \emptyset\right] \\
&\le \mathsf{Pr}\left[Q_{\mathsf{M}} \cap Q^* \neq \emptyset\right] + \mathsf{Pr}\left[(Q_{\mathbf{I}} \cap Q \neq \emptyset) \wedge (Q_{\mathsf{M}} \cap Q^* = \emptyset)\right] \\
&\le \mathsf{Pr}\left[\mathsf{G}_0\right] + \mathsf{Adv}^{\mathsf{ext}[\mathbf{I}]}_{\mathsf{M},S,P,\mathsf{Ext}}(\lambda) .
\end{aligned}$$

PROCEDURE $\mathsf{Sim}(+, x)$:
if $T_P[+, x] = \perp$ **then**
  $y \xleftarrow{\$} \mathsf{Func}(x)$
  $z \xleftarrow{\$} \{0,1\}^{n-r}$
  $T_P[+, x] \leftarrow y \,\|\, z$
  $T_P[-, y \,\|\, z] \leftarrow x$
return $T_P[+, x]$

PROCEDURE $\mathsf{Sim}(-, y)$:
if $T_P[-, y] = \perp$ **then**
  $x \xleftarrow{\$} \{0,1\}^n$
  $T_P[-, y] \leftarrow x$
  $T_P[+, x] \leftarrow y$
return $T_P[-, y]$

Fig. 19: Simulator Sim for the proof of Theorem 4. The simulator has access to an oracle Func which takes n-bit inputs and returns r-bit inputs.

Game G_1 simply modifies oracle Func to reply to queries (i, x) using g_i, where $g_1, \ldots, g_n \xleftarrow{\$} \mathbf{J}_\lambda$. Similarly, it runs n copies of the simulator Sim – call them $\mathsf{Sim}_1, \ldots, \mathsf{Sim}_n$ – and queries $\mathsf{Prim}(i, x)$ are answered by $\mathsf{Sim}_i^{g_i}$. it is now straightforward to build a distinguisher B' for the multi-user CP-sequential indifferentiability game such that

$$\mathsf{Adv}^{\mathsf{m-cpi}[\mathbf{I}\to\mathbf{J}]}_{\mathsf{M},\mathsf{Sim},B'}(\lambda) = \mathsf{Pr}\left[\mathsf{G}_0\right] - \mathsf{Pr}\left[\mathsf{G}_1\right] \ .$$

This is because B' can easily keep track of all sets Q_{M} and $Q_{\mathbf{I}}$ and simulate the rest of G_0 or G_1 using the given oracles Func and Prim. The final distinguisher B can be obtained from B' by applying Lemma 1. □

B Proof of Theorem 4

We start with the description of our simulator Sim. We define our simulator formally in Figure 19 but give an overview of the simulation strategy next.

Simulator description. The simulator Sim provides the interface for *forward* queries $(+, x)$ and backward queries $(-, y)$. The simulator has access to an oracle Func, which implements a random function that maps n-bits to r-bits. In order to mimic the permutation, the simulator internally maintains a table T_P that has entries $(+, x)$ and $(-, y)$. Now, queries are handled as follows

Forward queries. This case is straightforward. Whenever $(+, x)$ is queried, Sim checks if $T_P[+, x] = y \neq \perp$ is defined, and answers with y if it is. Otherwise, it queries Func on input x and gets an r-bit uniform random string y. It additionally generates a random $(n - r)$-bit string z. The concatenation $y \,\|\, z$ is the n-bit string that acts as a response to the original query $(+, x)$. The table T_P is updated to contain $T_P[+, x] = y \,\|\, z$ and $T_P[-, y \,\|\, z] = x$.

Backward queries. The difficulty stems from dealing with an inversion query $(-, y)$. Indeed, here, the simulator has no sense with which x it needs to be consistent with, but our strategy will decide to simply ignore any possible contraint, and simply respond with a random value x. This may not look like a sound strategy as the distinguisher A can have knowledge possibly from construction queries (A knows that $\mathsf{Chop}(x)$ is not a substring of y) that could help it in distinguishing the simulated world from the real world.

The main sequence of games. We are going to prove the theorem using a sequence of games $\mathsf{G}_0(\lambda), \mathsf{G}_1(\lambda), \ldots, \mathsf{G}_5(\lambda)$ (and often omit the security parameter λ for ease of notation). These games are detailed in Figures 20 and 21.

MAIN $\mathsf{G}_i(\lambda)$:
$P^+, P^- \leftarrow \emptyset$
$\mathsf{st} \leftarrow\!\!\$\ A_1^{\mathsf{Func}}(1^\lambda)$
$b' \leftarrow\!\!\$\ A_2^{\mathsf{Prim}}(1^\lambda, \mathsf{st})$
return b'

ORACLE $\mathsf{Func}(x)$:
$y \xleftarrow{\$} \mathsf{Prim}(\mathsf{+}, x)$
return $y[1 \dots r]$

ORACLE $\mathsf{Prim}(\sigma, x)$:
// $\mathsf{G}_0(\lambda)$
if $T_P[\sigma, x] = \bot$ **then**
  $y \xleftarrow{\$} \{0,1\}^n \setminus P^\sigma$
  $T_P[\sigma, x] \leftarrow y$
  $T_P[\overline{\sigma}, y] \leftarrow x$
  $P^\sigma \xleftarrow{\cup} \{y\}$; $P^{\overline{\sigma}} \xleftarrow{\cup} \{x\}$
return $T_P[\sigma, x]$

ORACLE $\mathsf{Prim}(\sigma, x)$:
// $\boxed{\mathsf{G}_1(\lambda)}$, $\mathsf{G}_2(\lambda)$
if $T_P[\sigma, x] = \bot$ **then**
  $y \xleftarrow{\$} \{0,1\}^n$
  if $y \in P^\sigma$ **then**
    $\mathsf{bad}_1 \leftarrow \mathsf{true}$
    $\boxed{y \xleftarrow{\$} \{0,1\}^n \setminus P^\sigma}$
  $T_P[\sigma, x] \leftarrow y$
  $T_P[\overline{\sigma}, y] \leftarrow x$
  $P^\sigma \xleftarrow{\cup} \{y\}$; $P^{\overline{\sigma}} \xleftarrow{\cup} \{x\}$
return $T_P[\sigma, x]$

Fig. 20: Games $\mathsf{G}_0(\lambda)$, $\mathsf{G}_1(\lambda)$, and $\mathsf{G}_2(\lambda)$ in the proof of Theorem 4 where Func and MAIN remain same through all three games.

It is easy to verify that G_0 represents the interaction of A in the real world, i.e., Func queries are answered by the Chop construction (which makes a call to Prim), and Prim itself behaves like a proper random permutation, implemented using lazy sampling. This game is slightly modified in game G_1, where this lazy sampling first samples a random value y from all of $\{0,1\}^n$, and then checks whether it collides with a value that was already used, and if so, the value y is resampled with proper restrictions. Furthermore, if such re-sampling occurs, a flag bad_1 is set. Clearly, this does not modify the behavior of the game in the eyes of the distinguisher, as the final distribution of the sampled y is still uniform of the set of possible values. Thus, we have just argued that $\mathsf{Pr}[\mathsf{G}_0] = \mathsf{Pr}[\mathsf{G}_1]$.

Then, G_2 modifies G_1 further by *removing* this re-sampling. In particular, note that this means that values may be overwritten in T_P, creating inconsistent entries,[13] but this will be only after the flag bad_1 has been set. In fact, the games G_1 and G_2 are *identical* until the flag bad_1 is set, which means in particular that

$$\mathsf{Pr}[\mathsf{G}_0] - \mathsf{Pr}[\mathsf{G}_2] = \mathsf{Pr}[\mathsf{G}_1] - \mathsf{Pr}[\mathsf{G}_2] \leq \mathsf{Pr}[\mathsf{G}_2 \text{ sets } \mathsf{bad}_1] \ . \tag{32}$$

It is not hard to upper bound the probability $\mathsf{Pr}[\mathsf{G}_2 \text{ sets } \mathsf{bad}_1]$. We note in particular that Prim samples a new y at most $q+p$ times (recall Prim is also called within the initial q Func calls by A), and therefore, by the union bound,

$$\mathsf{Pr}[\mathsf{G}_2 \text{ sets } \mathsf{bad}_1] \leq \frac{(q+p)^2}{2^n} \ . \tag{33}$$

GAME G_3. The next transitions will now modify the games further into the actual ideal world game where A interacts with a random oracle and the simulator Sim. First off, in G_3 (Fig. 21), we are going to modify the behavior of Func queries. In particular, rather than invoking Prim, Func is going to perform the same actions Prim would have executed directly. In particular, Func keeps a table T_F of the values it returned. And upon being invoked on x, it is going to set $T_P[\mathsf{+}, x]$ to a random $y \,\|\, z$, and set $T_P[\mathsf{-}, y \,\|\, z] = x$, and return only the y part, to which $T_F[x]$ is also set. Moreover, a *forward* query $(\mathsf{+}, x)$ to Prim will now be processed by calling $\mathsf{Func}(x)$ directly whenever $T_P[\mathsf{+}, x]$ is undefined.

[13] For example, we may have $T_P[\mathsf{+}, x] = y$ and $T_P[\mathsf{-}, y] = x$, and a later query for $x' \neq x$ ends up setting $T_P[\mathsf{+}, x'] = y$ and $T_P[\mathsf{-}, y] = x'$, while still $T_P[\mathsf{+}, x] = y$.

Note that moving from G_2 to G_3 does *not* change the behavior of the experiment – this is because the very same instructions are executed in any execution, with the only difference that Func does not check whether $T_P[+, x]$ is defined before setting it to a random value, but only checks for $T_F[x]$ being defined. However, it cannot be that $T_F[x] = \perp$ but $T_P[+, x] \neq \perp$ because A makes all its Func queries *before* its Prim queries, and otherwise Func is only later called within Prim *after* it has been confirmed that $T_P[+, x] = \perp$. Therefore, $\mathsf{Pr}[\mathsf{G}_2] = \mathsf{Pr}[\mathsf{G}_3]$.

A few syntactical properties will soon be important in game G_3. The game also maintains a set Y as part of its global state. At any point in time, Y contains those $y \,\|\, z$ that have been generated within Func queries which however have not been returned by *forward* Prim queries by A_2. Should A_2 make a backward query $(-, y \,\|\, z)$ for $y \,\|\, z \in Y$, the game is going to set a flag bad_2 to true. Second, it will be convenient to think of the completions $z \in \{0,1\}^{n-r}$ as being kept in a table $T_C[x]$ which is globally accessible through the procedure $\mathsf{Complete}$.

GAME G_4. The final game will simply shift away the task of correctly setting $T_P[+, x]$ and $T_P[-, y \,\|\, z]$ away from Func, and place it back into Prim. Clearly, this modifies the game, however we will observe that as long as bad_2 does not occur, games G_3 and G_4 behave *identically*. Indeed, assume that all random coins are fixed for a second, and during the resulting execution, game G_3 sets $T_P[+, x] = y \,\|\, z$ and $T_P[-, y \,\|\, z] = x$ within $\mathsf{Func}(x)$, whereas G_4 does not. Clearly, this makes no difference if the call is within $\mathsf{Prim}(+, x)$, so we can assume without loss of generality this happens within one of A_1's direct calls to Func. Now, the only way this difference is noticed if later A_2 issues a call $\mathsf{Prim}(-, y \,\|\, z)$, and $\mathsf{Prim}(+, x)$ was never invoked, as this will result in $T_P[-, y \,\|\, z] \neq \perp$ in G_3, but $T_P[-, y \,\|\, z] = \perp$ in G_4. This however means exactly that $y \,\|\, z \in Y$, which implies bad_2 being set to true. Therefore,

$$\mathsf{Pr}[\mathsf{G}_3] - \mathsf{Pr}[\mathsf{G}_4] \leq \mathsf{Pr}[\mathsf{G}_4 \text{ sets } \mathsf{bad}_2] \ . \tag{34}$$

Also, we see that the behavior of G_4 is identical to that of the ideal-world experiment where A interacts with a random Func and the simulator Sim. Therefore,

$$\begin{aligned} \mathsf{Adv}^{\mathsf{cpi}[\mathbf{P}\to\mathbf{R}]}_{\mathsf{Chop},\mathsf{Sim},A}(\lambda) &= \mathsf{Pr}[\mathsf{G}_0] - \mathsf{Pr}[\mathsf{G}_4] \\ &= \mathsf{Pr}[\mathsf{G}_1] - \mathsf{Pr}[\mathsf{G}_2] + \mathsf{Pr}[\mathsf{G}_3] - \mathsf{Pr}[\mathsf{G}_4] \\ &\leq \frac{(q+p)^2}{2^n} + \mathsf{Pr}[\mathsf{G}_4 \text{ sets } \mathsf{bad}_2] \ . \end{aligned} \tag{35}$$

BOUNDING THE BAD PROBABILITY. It is easier to reason about bad_2 being set to true by considering the following alternative to game G_4, which we call G_5 (Figure 22). G_5 looks identical (in the eyes of A) to G_4, but keeps a different flag bad_3. While the semantics of bad_2 in G_4 and bad_3 in G_5 are very different, we will explain below that the probability that bad_2 is set in G_4 during the whole execution equals the probability that bad_3 is set in an execution in G_5.

The main idea is that we want to postpone the sampling of $T_C[x]$ inside $\mathsf{Complete}(x)$ to the latest possible point, and make the bad event only depend on this sampling. To this end, Func will not call $\mathsf{Complete}$ any more, but rather G_5 keeps track (in a set Q) of the Func queries made by A_1. A call to $\mathsf{Complete}(x)$ is then made only within a call $\mathsf{Prim}(+, x)$. In which case, we check whether an earlier query $\mathsf{Prim}(-, y \,\|\, z)$ would have provoked the bad event. As there are $x \in Q$ which may never be queried to Prim, the game iterates over these at the end. Thus, we have argued above that $\mathsf{Pr}[\mathsf{G}_4 \text{ sets } \mathsf{bad}_2] = \mathsf{Pr}[\mathsf{G}_5 \text{ sets } \mathsf{bad}_3]$. Note that every time $\mathsf{Complete}$ is called in G_5, it returns a fresh random $(n-r)$-bit string. Then, bad_3 is set to true if, for the given y, we have $y \,\|\, z \in Q^-$.

ORACLE $\mathsf{Func}(x)$:
if $T_F[x] = \perp$ **then**
$y \xleftarrow{\$} \{0,1\}^r$
$T_F[x] \leftarrow y$
$z \xleftarrow{\$} \mathsf{Complete}(x)$
$\boxed{T_P[\mathsf{+}, x] \leftarrow y \,\|\, z}$
$\boxed{T_P[\mathsf{-}, y \,\|\, z] \leftarrow x}$
$Y \xleftarrow{\cup} \{y \,\|\, z\}$
return $T_F[x]$

PROCEDURE $\mathsf{Complete}(x)$:
if $T_C[x] = \perp$ **then**
$T_C[x] \xleftarrow{\$} \{0,1\}^{n-r}$
return $T_C[x]$

ORACLE $\mathsf{Prim}(\mathsf{+}, x)$:
// $\boxed{\mathsf{G}_3}$
if $T_P[\mathsf{+}, x] = \perp$ **then**
$y \leftarrow \mathsf{Func}(x)$
$Y \leftarrow Y \setminus T_P[\mathsf{+}, x]$
return $T_P[\mathsf{+}, x]$

ORACLE $\mathsf{Prim}(\mathsf{-}, y)$:
// $\boxed{\mathsf{G}_3}$, G_4
if $y \in Y$ **then**
$\mathsf{bad}_2 \leftarrow \mathsf{true}$
if $T_P[\mathsf{-}, y] = \perp$ **then**
$x \xleftarrow{\$} \{0,1\}^n$
$T_P[\mathsf{-}, y] \leftarrow x$
$T_P[\mathsf{+}, x] \leftarrow y$
return $T_P[\mathsf{-}, y]$

ORACLE $\mathsf{Prim}(\mathsf{+}, x)$:
// G_4
if $T_P[\mathsf{+}, x] = \perp$ **then**
$y \leftarrow \mathsf{Func}(x)$
$z \xleftarrow{\$} \mathsf{Complete}(x)$
$T_P[\mathsf{+}, x] \leftarrow y \,\|\, z$
$T_P[\mathsf{-}, y \,\|\, z] \leftarrow x$
$Y \leftarrow Y \setminus T_P[\mathsf{+}, x]$
return $T_P[\mathsf{+}, x]$

Fig. 21: Games $\mathsf{G}_3(\lambda)$ and $\mathsf{G}_4(\lambda)$ in the proof of Theorem 4 where MAIN is as described in Figure 20.

Given there are q chances for this to occur, and that $|Q^-| \leq p$, the union bound yields

$$\mathsf{Pr}\left[\mathsf{G}_4 \text{ sets } \mathsf{bad}_2\right] = \mathsf{Pr}\left[\mathsf{G}_5 \text{ sets } \mathsf{bad}_3\right] \leq \frac{q \cdot p}{2^{n-r}}\,, \tag{36}$$

hence concluding the proof. □

C Proof of Theorem 5

In Section 5.1, we described the hybrids G_1,G_2,G_3 and G_4 that will be used to establish the sequential indifferentiability of Ψ_5. Next, we prove that each of the adjacent hybrids G_j and G_{j+1} are indistinguishable, thereby concluding the proof of Theorem 5.

C.1 Indistinguishability of G_1 and G_2

In this section, we derive a bound for $\Delta(\mathsf{G}_1, \mathsf{G}_2)$ by invoking a standard argument about replacing a permutation with a two-sided random function. We will consider (Sim, A) as a coupled distinguisher D that makes q' queries to the permutation/two-sided random function and therefore has advantage $q'^2/2^{2n}$ in distinguishing the random permutation from the two-sided random function. In Lemma 2, we will prove Sim's query complexity and thereby establish $\Delta(\mathsf{G}_1, \mathsf{G}_2)$ in Lemma 3.

Lemma 2. *At any point in* G_2, $|G_k| \leq q + 2q^2$ *and* $|P| \leq q_c + 2q^2$ *where* $k \in [5]$ *and* $|G_k|$ *and* $|P|$ *is the size of the respective tables.* ■

Proof. In Phase 1 of the game G_2, for every element that gets added to G_1 and G_5 there exists a query by A_2 to Prim. This is because elements get added to G_1 (in Phase 1) either due to a query $(1, X_1)$ by A_2 such that $G_1[X_1] = \perp$ or due to a query $(3, X_3)$ by A_2 such that $\mathrm{Chains}[3, X_3] = (1, X_1)$ and $G_3[X_3] = \perp$ (lines 31-32 in Figure 6). Therefore, $|G_1| \leq q$ and $|G_5| \leq q$ at any point in Phase 1.

MAIN $\mathsf{G}_5(\lambda)$:
$Q, Q^- \leftarrow \emptyset$
$\mathsf{bad}_3 \leftarrow \mathsf{false}$; $\mathsf{done} \leftarrow \mathsf{false}$
$\mathsf{st} \leftarrow\!\!\$\ A_1^{\mathsf{Func}}(1^\lambda)$
$\mathsf{done} \leftarrow \mathsf{true}$
$b' \leftarrow\!\!\$\ A_2^{\mathsf{Prim}}(1^\lambda, \mathsf{st})$
for all $x \in Q$ **do**
  $z \xleftarrow{\$} \mathsf{Complete}(x)$
  if $T_F[x] \,\|\, z \in Q^-$ **then**
    $\mathsf{bad}_3 \leftarrow \mathsf{true}$
return b'

ORACLE $\mathsf{Func}(x)$:
if $\neg\mathsf{done}$ **then** $Q \xleftarrow{\cup} \{x\}$
if $T_F[x] = \perp$ **then**
  $y \xleftarrow{\$} \{0,1\}^r$
  $T_F[x] \leftarrow y$
return $T_F[x]$

ORACLE $\mathsf{Prim}(\mathsf{+}, x)$:
if $T_P[\mathsf{+}, x] = \perp$ **then**
  $y \leftarrow \mathsf{Func}(x)$
  $z \xleftarrow{\$} \mathsf{Complete}(x)$
  if $(x \in Q) \wedge (y \,\|\, z \in Q^-)$ **then**
    $Q \leftarrow Q \setminus \{x\}$
    $\mathsf{bad}_3 \leftarrow \mathsf{true}$
  $T_P[\mathsf{+}, x] \leftarrow y \,\|\, z$
  $T_P[\mathsf{-}, y \,\|\, z] \leftarrow x$
return $T_P[\mathsf{+}, x]$

ORACLE $\mathsf{Prim}(\mathsf{-}, y)$:
$Q^- \xleftarrow{\cup} \{y\}$
if $T_P[\mathsf{-}, y] = \perp$ **then**
  $x \xleftarrow{\$} \{0,1\}^n$
  $T_P[\mathsf{-}, y] \leftarrow x$
  $T_P[\mathsf{+}, x] \leftarrow y$
return $T_P[\mathsf{-}, y]$

Fig. 22: Game $\mathsf{G}_5(\lambda)$ in the proof of Theorem 4.

Now, we bound the size of the table P. Sim issues queries to Func only when A_2 queries $\mathsf{Prim}(k, X)$ where $k \in \{2, 4\}$ and $G_k[X] = \perp$. Since queries to Func are only issued in Phase 1 and $|G_1| \le q$, then for every query $(2, X)$ by A_2 there are at most q queries to Func. Over q queries by A_2, there are at most q^2 queries to Func by Sim. Following the same analysis for Prim queries of the form $(4, \cdot)$, Sim issues at most $2q^2$ queries to Func in G_2.

Therefore, $|P| \le q_c + 2q^2 \le 3q^2$ where the inequality is due to $q_c \le q$.

Next we bound $|G_k|$ and $|\mathrm{Chains}[j]|$ where $k \in [5]$ and $j \in \{1, 3, 5\}$ where

$$\begin{aligned} \mathrm{Chains}[l] &= \bigcup_{(l,X):\mathrm{Chains}[l,X]\neq\perp} \mathrm{Chains}[l, X] \qquad \text{where } l \in \{1, 5\}\ , \\ \mathrm{Chains}[3] &= \bigcup_{(3,X):\mathrm{Chains}[3,X]} \{\mathrm{Chains}[3, X]\}\ . \end{aligned} \tag{37}$$

At any point in G_2 elements get added to $\mathrm{Chains}[l]$ only after a Func query and a Func query can add at most one element in $\mathrm{Chains}[j]$. Hence it is easy to see that $|\mathrm{Chains}[j]| \le 2q^2$. In G_2, for any call to Complete (immediate or delayed) there is exactly one Func query by Sim. As proved earlier, there are at most $2q^2$ calls to Complete and each call adds (via ForceVal) at most one element in $|G_k|$. In Phase 1, elements get added to G_k either due to a query to Prim by A_2 or due to ForceVal calls inside Complete. Moreover in Phase 2, elements also get added to G_k due to ForceVal inside Complete or due to $\mathsf{F}^{\mathrm{inner}}$ calls just before calling Complete. Therefore, we have that $|G_k| \le q + 2q^2$ because there are at most $2q^2$ calls to Complete. □

By Lemma 2 we have the following lemma.

Lemma 3. $\Delta(\mathsf{G}_1, \mathsf{G}_2) \le 9q^4/2^{2n}$. ■

Proof. Let $D = (\mathsf{Sim}, A)$ be a distinguisher that makes at most $3q^2$ $(\ge q_c + 2q^2)$ to the oracle Func. By a standard argument, the lemma holds. □

C.2 Indistinguishability of G_2 and G_3

In this section, we prove the indistinguishability of games G_2 and G_3 by defining a map τ that maps (f, ρ) to h such that the output distribution of A in G_2 is not very different from that in G_3. Infact, we prove that for certain pairs (f, ρ) the image $h = \tau(f, \rho)$ is such that the output distributions of both games are identical. We refer to such pairs as good and the absence of certain bad events in $\mathsf{G}_2^{f,\rho}$ makes (f, ρ) a good pair. We define these events next and later in Lemma 10 show that most pairs (f, ρ) are good.

Definition of good pairs

For the following definitions, consider the game G_2 using a pair $(f, \rho) \in \mathcal{F} \times \mathcal{R}$. i.e. $\mathsf{G}_2^{f,\rho}$. We are going to define a few bad events that occur when executing G_2 with randomness fixed to such a pair (f, ρ). We say that a pair (f, ρ) is *good* if no such event occurs in an execution, and otherwise it is *bad*.

Definition 3. *The event* BadP *occurs in an execution of* $\mathsf{G}_2^{f,\rho}$ *if one the following holds,*

1. *Immediately after a call* $(X_5, X_6) \leftarrow \rho[+, X_0, X_1]$ *either* $G_5[X_5] \neq \perp$ *or* $P[-, X_5, X_6] \neq \perp$.
2. *Immediately after a call* $(X_0, X_1) \leftarrow \rho[-, X_5, X_6]$ *either* $G_1[X_1] \neq \perp$ *or* $P[+, X_0, X_1] \neq \perp$.

Definition 4. *The event* $\mathsf{BadOutside}$[14] *occurs in an execution of* $\mathsf{G}_2^{f,\rho}$ *if one of the following holds,*

1. *Immediately after an assignment* $G_1[X] \leftarrow f[1, X]$ *there exist* X_2 *and* $(+, X_0, X_1)$ *such that* $G_2[X_2] \neq \perp$, $P[+, X_0, X_1] \neq \perp$ *and* $G_1[X] = X_2 \oplus X_0$.
2. *Immediately after an assignment* $G_5[X] \leftarrow f[5, X]$ *there exist* X_4 *and* $(-, X_5, X_6)$ *such that* $G_4[X_4] \neq \perp$, $P[-, X_5, X_6] \neq \perp$ *and* $G_5[X] = X_4 \oplus X_6$.

Definition 5. *The event* $\mathsf{BadGutShot}$ *occurs in an execution of* $\mathsf{G}_2^{f,\rho}$ *if one of the following holds,*

1. *Immediately after an assignment* $G_2[X] \leftarrow f[2, X]$ *there exist* X_1 *and* X_3 *such that* $G_1[X_1] \neq \perp$, $G_3[X_3] \neq \perp$ *and* $G_2[X] = X_1 \oplus X_3$.
2. *Immediately after an assignment* $G_2[X] \leftarrow f[2, X]$ *there exist* X_1 *and* X_3 *such that* $G_1[X_1] \neq \perp, \mathit{Chains}[3, X_3] \neq \perp$ *and* $G_2[X] = X_1 \oplus X_3$.
3. *Immediately after an assignment* $G_4[X] \leftarrow f[4, X]$ *there exist* X_5 *and* X_3 *such that* $G_5[X_5] \neq \perp$, $G_3[X_3] \neq \perp$ *and* $G_4[X] = X_5 \oplus X_3$.
4. *Immediately after an assignment* $G_4[X] \leftarrow f[4, X]$ *there exist* X_5 *and* X_3 *such that* $G_5[X_5] \neq \perp, \mathit{Chains}[3, X_3] \neq \perp$ *and* $G_4[X] = X_5 \oplus X_3$.

Let the i-th query issued by A_1 be $(\sigma, X_k^i, X_{k+1}^i)$ where $(\sigma, k) \in \{(+, 0), (-, 5)\}$. Before the control of G_2 is passed to A_2, the tuples $(X_0^i, X_1^i, X_5^i, X_6^i)$ corresponding to the the ith query by A_1 are defined. Since (f, ρ) is sampled at the beginning of G_2 and Sim always replies to queries $(1, X)$ and $(5, X)$ using the values $f[1, X]$ and $f[5, X]$, respectively, we can further define

$$\begin{aligned} X_2^i &= f[1, X_1^i] \oplus X_0^i \ , \\ X_4^i &= f[5, X_5^i] \oplus X_6^i \ . \end{aligned} \tag{38}$$

The following events are assuming $(X_0^i, X_1^i, X_2^i, X_4^i, X_5^i, X_6^i)$ are defined for $i \in [q_c]$. We call such tuples *relevant chains* or *relevant tuples*.

[14] The names of the events $\mathsf{BadOutside}$ and $\mathsf{BadGutShot}$ are adopted from straight draws in the card game of Poker.

Definition 6. *The event* BadlyCollide *occurs in an execution of* $\mathsf{G}_2^{f,\rho}$ *if one of the following holds,*

1. $X_2^{i_1} = X_2^{i_2}$ *for* $i_1 \neq i_2 \in [q_c]$.
2. $X_4^{i_1} = X_4^{i_2}$ *for* $i_1 \neq i_2 \in [q_c]$.

Definition 7. *The event* BadP2 *occurs in an execution of* $\mathsf{G}_2^{f,\rho}$ *if one the following holds,*

1. *Immediately after a call* $(X_5, X_6) \leftarrow \rho[+, X_0, X_1]$ *where* $(X_0, X_1) \neq (X_0^i, X_1^i)$ *for all* $i \in [q_c]$ *there exists* $i_1 \in [q_c]$ *such that* $X_5 = X_5^{i_1}$.
2. *Immediately after a call* $(X_0, X_1) \leftarrow \rho[-, X_5, X_6]$ *where* $(X_5, X_6) \neq (X_5^i, X_6^i)$ *for all* $i \in [q_c]$ *there exists* $i_1 \in [q_c]$ *such that* $X_1 = X_1^{i_1}$.

Definition 8. *The event* BadOutside2 *occurs in an execution of* $\mathsf{G}_2^{f,\rho}$ *if one of the following holds,*

1. *Immediately after an assignment* $G_1[X] \leftarrow f[1, X]$ *where* $X \neq X_1^i$ *for all* $i \in [q_c]$ *there exist* $((X_4, X_5, 4), (X_0, X)) \in \mathit{Chains}[1, X]$ *and* $i_1 \in [q_c]$ *such that* $X_2^{i_1} = G_1[X] \oplus X_0$.
2. *Immediately after an assignment* $G_5[X] \leftarrow f[5, X]$ *where* $X \neq X_5^i$ *for all* $i \in [q_c]$ *there exist* $((X_1, X_2, 1), (X, X_6)) \in \mathit{Chains}[5, X]$ *and* $i_1 \in [q_c]$ *such that* $X_4^{i_1} = G_5[X] \oplus X_6$.

Most pairs (f, ρ) are good

We next show that most pairs $(f, \rho) \in \mathcal{F} \times \mathcal{R}$ are good, i.e., that a randomly sampled (f, ρ) pair is good except with small probability.

For all the following proofs we consider the occurrence of any of the bad events defined above at any point in the execution of G_2. Also, in order to simplify the calculation of the probability that (f, ρ) is not good, we will often alternate viewing G_2 as lazily sampling (f, ρ) (occasionally only partially), rather than pre-sampling it. (This will obviously not affect the probability, as the experiments are equivalent.) For the following calculations, it will be convenient to first observe the following: from Lemma 2 we know that $|P| \leq t(q)$, $|G_i| \leq t(q)$ for all $i \in [5]$ and $|\mathrm{Chains}[l]| \leq t(q)$ where $l \in \{1, 3, 5\}$ and $t(q) = 3q^2$. (We will occasionally write t rather than $t(q)$.)

Lemma 4. $\mathsf{Pr}[\mathsf{BadP}] \leq 2t^2/2^n$. ■

Proof. For any query to $(X_5, X_6) \leftarrow \rho[+, X_0, X_1]$, BadP occurs if either $P[-, X_5, X_6] \neq \perp$ which happens with probability $t/2^{2n}$ or if $G_5[X_5] \neq \perp$ which happens with probability $t/2^n$. Since $|P| \leq t$ at any point in G_2, there can be at most t such calls and hence by the union bound we have the lemma. □

Lemma 5. $\mathsf{Pr}[\mathsf{BadOutside}] \leq 2t^3/2^n$. ■

Proof. For any assignment of the form $G_1[X] \leftarrow f[1, X]$, BadOutside occurs if there exists X_2 and $(+, X_0, X_1)$ such that $G_2[X_2] \neq \perp$, $P[+, X_0, X_1] \neq \perp$ and $G_1[X] = X_2 \oplus X_0$. Since $|P| \leq t$ and $|G_2| \leq t$, there are at most t^2 such pairs for which BadOutside happens with probability $1/2^n$. The symmetric argument holds for assignments of the form $G_5[X] \leftarrow f[5, X]$.

Since $|G_1|, |G_5| \leq t$ at any point in G_2, there can be at most $2t$ such assignments and hence by the union bound we have the lemma. □

Lemma 6. $\mathsf{Pr}[\mathsf{BadGutShot}] \leq 4t^3/2^n$. ■

Proof. For any assignment of the form $G_2[X] \leftarrow f[2, X]$, BadGutShot occurs if there exists X_1 and X_3 such that $G_1[X_1] \neq \perp$, $G_3[X_3] \neq \perp$ (Chains$[3, X_3] \neq \perp$) and $G_1[X] = X_2 \oplus X_0$. Since $|\text{Chains}[3]| \leq t$, $|G_3| \leq t$ and $|G_1| \leq t$, there are at most $2t^2$ such pairs for which BadGutShot happens with probability $1/2^n$. Since $|G_2|, |G_4| \leq t$ at any point in G_2, there can be at most $2t$ such assignments and hence by the union bound we have the lemma. □

Lemma 7. $\mathsf{Pr}[\text{BadlyCollide}] \leq 2q^2/2^n$. ■

Proof. To find the probability of BadlyCollide, we visualize the game G_2 in the following way. At the beginning of the game, we just sample $\rho \leftarrow\!\!\$\, \mathcal{R}$ and run the game with A_1. After A_1 is done with all its q_c queries, we define X_2^i and X_4^i as follows:

$$\begin{aligned} X_2^i &\leftarrow r_i \oplus X_0^i \\ X_4^i &\leftarrow s_i \oplus X_6^i, \end{aligned} \tag{39}$$

where $r_i, s_i \leftarrow\!\!\$\, \{0,1\}^n$ for $i \in [q_c]$. Note that the distribution of X_2^i as defined above is the same as defined in Equation 38.

It is easy to see that the probability of collision among the X_2^i values is at most $q^2/2^n$. Similarly, probability of collision among the X_4^i values is at most $q^2/2^n$. Hence by the union bound the lemma holds. □

For the following proofs, we will visualize G_2 as sampling randomness lazily until A_1 is done with its queries. At this point, we can define the respective relevant tuples $(X_0^i, X_1^i, X_2^i, X_4^i, X_5^i, X_6^i)$. Now the control is passed to A_2 where we sample randomness lazily again. Note that BadP2 and BadOutside2 are defined for calls/assignments not involving the relevant tuples. Hence resorting to lazy sampling in the second stage of the game does not change the output distribution in this variant of G_2.

Lemma 8. $\mathsf{Pr}[\text{BadP2}] \leq tq/2^n$. ■

Proof. For any query to $(X_5, X_6) \leftarrow \rho[+, X_0, X_1]$ where $(X_0, X_1) \neq (X_0^i, X_1^i)$ for all $i \in [q_c]$, BadP2 occurs if there exists $i_1 \in [q_c]$ such that $X_5 = X_5^{i_1}$. This happens with probability at most $q/2^n$. Since $|P| \leq t$ at any point in G_2, there can be at most t such calls to ρ and hence by the union bound we have the lemma. □

Lemma 9. $\mathsf{Pr}[\text{BadOutside2}] \leq 2qt^2/2^n$. ■

Proof. For any assignment of the form $G_1[X] \leftarrow f[1, X]$ where $X \neq X_1^i$ for all $i \in [q_c]$, BadOutside2 occurs if there exists $((X_4, X_5, 4), (X_0, X)) \in \text{Chains}[1, X]$ and $i_1 \in [q_c]$ such that $G_1[X] = X_2^{i_1} \oplus X_0$. Since $|\text{Chains}[1]| \leq t$ at any point in G_2, the above happens with probability $tq/2^n$. Similar reasoning holds for assignment of the form $G_5[X] \leftarrow f[5, X]$ where $X \neq X_5^{i_1}$.

Since $|G_1|, |G_5| \leq t$ at any point in G_2, there can be at most $2t$ such assignments and hence by the union bound we have the lemma. □

Using the union bound with all the above lemmas, and combining this with Lemma 2, we obtain the following final lemma giving us an overall bound on the probability that (f, ρ) is bad.

Lemma 10. $\mathsf{Pr}[(f, \rho) \leftarrow\!\!\$\, \mathcal{F} \times \mathcal{R} : (f, \rho)\ \textit{is bad}\ ] \leq 13t^3/2^n \leq 351q^6/2^n$. ■

Properties of good executions

By Lemma 10, we know that most pairs (f, ρ) are good, and now we want to focus on executions in $\mathsf{G}_2^{f,\rho}$ for such good pairs (so-called "good executions"). This then allows us to define a map τ that maps (f, ρ) to h such that the executions of $\mathsf{G}_2^{f,\rho}$ and G_3^h are identical. To formalize this, we first define a notion of a transcript of such an execution.

Transcript of Game G_i: We define the transcripts of an execution of G_i as a sequence of tuples $T_1, T_2, \ldots$ recording each access to the tables P and G made during this execution. In particular, each T_j is one of the following:

1. $T_j = (Y_j, k_j, X_j)$ where $k_j \in [5]$ such that $\mathsf{F}^{\mathsf{inner}}$ was called on input (k_j, X_j) and Y_j was the value returned by $\mathsf{F}^{\mathsf{inner}}$.
2. $T_j = (Y_j, k_j, X_j)$ where $k_j \in \{2, 3, 4\}$ such that $\mathsf{ForceVal}$ was called on input (X_j, Y_j, k_j).
3. $T_j = (\sigma, X_k^j, X_{k+1}^j, X_{k'}^j, X_{k'+1}^j)$, where $(\sigma, k, k') \in \{(+, 0, 5), (-, 5, 0)\}$ such that Func was called on input $(\sigma, X_k^j || X_{k+1}^j)$ and $(X_{k'}^j, X_{k'+1}^j)$ was the value returned by Func.

The subtranscript $T[1 : j]$ which is the sequence $T_1, \ldots, T_j$. Next we describe a few properties of the execution and its transcript corresponding to a good pair (f, ρ).

Some notational conventions. In the following proofs we make references to line numbers. If not specified then it must be assumed that the line numbers are w.r.t. the Sim's pseudocode shown in Figure 6. Also, for ease of notation, we use $x \in T$ and $T[x] \neq \perp$ interchangeably but both denote that the key x is in the table T. For eg: $X \in G_k$ implies $G_k[X] \neq \perp$.

No Chains overwrites. Our first case is going to deal entering values in $\mathsf{Chains}[3, X]$ for some X. We show that this value is never overwritten during a good execution.

Lemma 11. *For any assignment of the form* $\mathsf{Chains}[3, X] \leftarrow (k, X_k)$ *in* $\mathsf{G}_2^{f,\rho}$ *(lines 13,25 in Figure 6),* $\mathsf{Chains}[3, X] = \perp$ *just before the assignment.* ■

Proof. Consider an assignment $\mathsf{Chains}[3, X] \leftarrow (5, X_5)$. This occurs only on a query $(2, X_2)$ by A_2 such that $G_2[X_2] = \perp$ just before the query. To reply to $(2, X_2)$, Sim assigns $G_2[X_2] \leftarrow f[2, X_2]$ and considers all tuples in $G_1 \times \{X_2\}$. For each of these tuples (X_1, X_2) it either schedules the partial chain $(X_1, X_2, 1)$ for immediate completion (line 9-10) or delays its completion (lines 11-13). The assignment to $\mathsf{Chains}[3, X]$ can only occur when the chain's completion gets delayed (lines 11,23).

Now consider $X = G_2[X_2] \oplus X_1$, since (f, ρ) is a good pair and the event $\mathsf{BadGutShot}$ does not occur $\mathsf{Chains}[3, X] = \perp$. Otherwise on assignment $G_2[X_2] \leftarrow f[2, X_2]$ there would exist X_1 and X such that $G_1[X_1] \neq \perp$, $\mathsf{Chains}[3, X] \neq \perp$ where $X = G_2[X_2] \oplus X_1$. The other assignment $\mathsf{Chains}[3, X] \leftarrow (1, X_1)$ is symmetrically handled. Hence the lemma holds. □

No overwrites in G_k. Next we prove that in $\mathsf{G}_2^{f,\rho}$ where (f, ρ) is a good pair, there are no overwrites in G_k. In particular, this will mean that whenever Sim is attempting to force a value (via $\mathsf{ForceVal}$) it can do so without making anything inconsistent.

Lemma 12. *For any call to* $\mathsf{ForceVal}(X, \cdot, l)$ *in* $\mathsf{G}_2^{f,\rho}$*, we have* $G_l[X] = \perp$ *just before the call.* ■

Proof. A call to $\mathsf{ForceVal}$ is triggered only from inside $\mathsf{Complete}$ (lines 43,48). Moreover, Sim calls the procedure $\mathsf{Complete}$ only when it is attempting to complete a partial chain which was not completed

before. If the lemma fails to hold then there exists a call to Complete during which it fails. Consider such a call to Complete and let C be the corresponding partial chain that is being completed. Furthermore, note that $C \notin$ CompletedChains just before the Complete call. We analyse the case when the partial chain C is of the form $(X_1, X_2, 1)$. The other instance when $C = (X_4, X_5, 4)$ can be handled symmetrically, and is omitted.

Therefore, let us assume that during the completion of $C = (X_1, X_2, 1)$, the lemma fails to hold for one of the ForceVal calls. There are two cases in which Complete can get called on $C = (X_1, X_2, 1)$ and we analyse them next.

- Case A: Complete$(C = (X_1, X_2, 1), \cdot)$ was called from line 10 of Figure 6. We refer to such call to Complete as an *immediate* chain completion. Immediate chain completions happen only in Phase 1 (before Sim runs AllComplete), as they are triggered due to a Prim query by A_2.

  Note that just before the call to Complete$((X_1, X_2, 1), \cdot)$ there must have been a query $(2, X_2)$ to Prim by A_2 such that $G_2[X_2] = \perp$ before the query. To respond to $(2, X_2)$ query of A_2, Sim assigns $G_2[X_2] \leftarrow f[2, X_2]$. After the assignment, Sim iterates through every tuple in $G_1 \times \{X_2\}$ (line 4 of Figure 6) to either schedule an immediate completion of C (lines 9-10) or delay its completion (lines 11-13). Since this call to Complete was made from line 10, we must have $G_1[X_1] \neq \perp$, $G_5[X_5] \neq \perp$ where $(X_5, X_6) = P[+, X_0, X_1]$, $X_0 = G_2[X_2] \oplus X_1$. Furthermore, let $X_3 = G_2[X_2] \oplus X_1$ and $X_4 = G_5[X_5] \oplus X_6$ where $G_2[X_2]$ is assigned to $f[2, X_2]$ just before this call to Complete. To complete the chain C, Complete will force $G_3[X_3]$ and $G_4[X_4]$ through ForceVal$(X_3, \cdot, 3)$ and ForceVal$(X_4, \cdot, 4)$ respectively. If the lemma fails to hold for C then either $G_3[X_3] \neq \perp$ or $G_4[X_4] \neq \perp$ before their respective ForceVal calls.

  First, we argue that $G_3[X_3] = \perp$ before the call ForceVal$(X_3, \cdot, 3)$. This is because the pair (f, ρ) is good and the event BadGutShot does not occur on assignment $G_2[X_2] \leftarrow f[2, X_2]$. Furthermore, Chains$[3, X_3] = \perp$ by the same argument. This ensures that by forcing $G_3[X_3]$ to a value (inside ForceVal) we are not making any previously completed chains inconsistent ($G_3[X_3] = \perp$) nor running into the risk of making this chain completion inconsistent due to a future delayed chain completion (Chains$[3, X_3] = \perp$).

  Therefore, if the lemma fails to hold for C, then it must be that $G_4[X_4] \neq \perp$ before the call to ForceVal$(X_4, \cdot, 4)$. Note that in order to complete the chain C, Sim queries Func with $(+, X_0||X_1)$ (line 7). It can either be that $(+, X_0||X_1)$ was a fresh query, that is, $P[+, X_0, X_1] = \perp$ before Sim's query to Func or that it was already in the table P before Sim's query. Furthermore, a tuple gets added to the table P either when Sim makes a query to Func or A_1 makes a query to Func. Assuming, $G_4[X_4] \neq \perp$, we are going to analyse when/how the tuple $(+, X_0, X_1)$, corresponding to query by Sim (to Func in line 7), was added to the table P and arrive at a contradiction to either the goodness of the pair (f, ρ) or that $(X_1, X_2, 1) \notin$ CompletedChains.

  - Case A.1: $(+, X_0, X_1)$ gets added to P due to the Func query in line 7. In this case, the query $(+, X_0||X_1)$ to Func is such that $P[+, X_0, X_1] = \perp$. Therefore, to reply to $(+, X_0||X_1)$, Func accesses $\rho[+, X_0, X_1]$ (as $P[+, X_0, X_1] = \perp$) to get (X_5, X_6) and returns it. However, since we are considering an immediate chain completion, we know that $G_5[X_5] \neq \perp$. This means the event BadP occurs when $\rho[+, X_0, X_1]$ is accessed, which is a contradiction as (f, ρ) is a good pair. Therefore, $(+, X_0, X_1) \in P$ even before Func was called in line 7.

- Case A.2: $(+, X_0, X_1)$ was added to P because of an earlier query to Func by Sim. Sim's queries to Func are distinct therefore if $(+, X_0, X_1)$ got added to P due to a Sim query then Sim must have previously queried $\mathsf{Func}(-, X_5||X_6)$. This could only happen on a query $(4, X_4)$ by A_2 where $X_4 = G_5[X_5] \oplus X_6$. Sim assigns $G_4[X_4] \leftarrow f[4, X_4]$. After the assignment, Sim would have iterated through all tuples in $\{X_4\} \times G_5$ (line 16). It would also consider (X_4, X_5) and make the $\mathsf{Func}(-, X_5, X_6)$ query.
  If $X_1 \in G_1$ at this point, this chain $(X_4, X_5, 4)$ would have been immediately completed (line 21-22) and hence $C = (X_1, X_2, 1)$ would already be completed even before the query $(2, X_2)$ by A_2 which is a contradiction as $C \notin$ CompletedChains.
  If $X_1 \notin G_1$ at this point then $(X_4, X_5, 4)$'s completion is delayed (line 23-25) where Chains$[3, X_3] \leftarrow (1, X_1)$ and $((X_4, X_5, 4), (X_0, X_1))$ gets inserted in Chains$[1, X_1]$. But as $G_1[X_1] \neq \perp$ before the A_2's query $(2, X_2)$ and we are still in Phase 1, it can only be the case that A_2 queried $(1, X_1)$ to Prim or $(3, X'_3)$ to Prim such that Chains$[3, X'_3] = (1, X_1)$ and $G_3[X'_3] = \perp$ whichever earlier. Here, X'_3 corresponds to one of the chains C' such that $(C', (U', V')) \in$ Chains$[1, X_1]$ [15]. In either case, Complete gets called on $(X_4, X_5, 4)$ and hence $(X_1, X_2, 1)$ would be completed even before the query $(2, X_2)$ by A_2 which is again contradiction to $C \notin$ CompletedChains. Therefore, the tuple $(+, X_0, X_1)$ corresponding to the query by Sim (to Func) can only get added to P due to a query (to Func) by A_1.

  Infact, recall that a $(2, X_2)$ query by A_2 can trigger more than one immediate chain completions as Sim loops over all newly formed partial chains $C' = (X'_1, X_2, 1)$ where $X'_1 \in G_1$. It follows from the arguments made in cases A.1 and A.2 that all of Sim's queries $(+, X'_0||X'_1)$ are such that the tuple $(+, X'_0, X'_1)$ was added to P due to a query by A_1.

- Case A.3: $(+, X_0, X_1)$ was added to P due to a query by A_1. The only remaining case for the tuple $(+, X_0, X_1)$ is that it was added to P due to a query by A_1. Therefore, the tuple $(X_0, X_1, X_3, X_4, X_5, X_6)$ falls on a relevant chain, that is, there exists $i \in [q_c]$ such that $X_j = X^i_j$ for $j \in \{0, 1, 2, 4, 5, 6\}$, thereby making C a relevant partial chain. As stated above, a $(2, X_2)$ query by A_2 can trigger immediate completion of more than one partial chains. As discussed at the end of the case A.2, all these partial chains are relevant chains.
  Recall that if the lemma fails to hold for C, then $G_4[X^i_4] \neq \perp$ before the $\mathsf{ForceVal}(X^i_4, \cdot, 4)$ call. Note that an element X gets added to a table G_4 either due to ForceVal call (issued by Sim from inside a call to Complete) or due to a direct query $(4, X)$ by A_2. Assuming that $G_4[X^i_4] \neq \perp$ before the ForceVal call (i.e., X^i_4 is in the table P), we are going to analyse when/how X^i_4 was added to G_4 and arrive at a contradiction to either the goodness of the pair (f, ρ) or that $(X_1, X_2, 1) \notin$ CompletedChains.
  - Case A.3.1: X^i_4 was added due to one of the Complete calls due to $(2, X_2)$. A query to $(2, X_2)$ by A_2 may trigger multiple calls to Complete. However, we just observed that all these calls must correspond to relevant chains. Therefore, if X^i_4 gets added to G_4 due to a Complete call triggered by the query $(2, X_2)$ on some chain $C' \neq C$, this would imply that two relevant chains C and C' share their X_4s and the event BadlyCollide occurs. This contradicts the fact that (f, ρ) is a good pair.

[15] By Lemma 11 there are no overwrites in Chains$[3, X]$. This ensures that a Prim query to $(3, X'_3)$ such that $G_3[X'_3] = \perp$ will lead to a call to $\mathsf{Sim}(5, X_5)$ (line 32). We will use this argument in the future without repeating this explanation.

* Case A.3.2: X_4^i was added due to a Complete call before A_2's query $(2, X_2)$. Let us assume that it was due to a call to Complete on some $(X_1', X_2', 1) \neq (X_1, X_2, 1)$. Again as (f, ρ) is a good pair, $(X_1', X_2', 1)$ cannot be a relevant chain (relevant chains donot share X_2s). Hence, when Sim queries $(+, X_0'||X_1')$ where $X_0' = G_1[X_1'] \oplus X_2'$, Func replies to $(+, X_0'||X_1')$ by accessing $\rho[+, X_0', X_1']$. Then, (f, ρ) being good, the events BadP and BadP2 do not occur. This ensures that $X_5' \notin G_5$ and also that $X_5' \neq X_5^{i_1}$ for all $i_1 \in [q_c]$. Since $X_5' \notin G_5$, chain $(X_1', X_2', 1)$'s completion gets delayed (lines 11-13). We are inside the call to Complete on $(X_1', X_2', 1)$ and Sim must be executing lines 27-30 for $(5, X_5')$. $G_5[X_5']$ is assigned to $f[5, X_5']$ and then Complete is called on C' where $(C', (X_5', X_6')) \in \text{Chains}[5, X_5']$. If $G_5[X_5'] \oplus X_6' = X_4^i$ (as we have assumed) then BadOutside2 will occur on assignment $G_5[X_5']$ where $X_5 \neq X_5^{i_1}$, which is a contradiction as (f, ρ) is a good pair.

* Case A.3.3: X_4^i gets added to G_4 due to a query $(4, X_4^i)$ by A_2. The only remaining case in which X_4^i can get added to G_4 is if A_2 makes a Prim query $(4, X_4^i)$. Note that such a query $(4, X_4^i)$ (if at all) must occur before $(2, X_2)$. Furthermore, when A_2 queries $(4, X_4^i)$, X_5^i must already be in G_5. Otherwise, BadOutside occurs on the assignment $G_5[X_5^i] \leftarrow f[5, X_5^i]$. This leaves us with the state that $X_5^i \in G_5$ when A_2 queries $(4, X_4^i)$ and $G_4[X_4^i] = \perp$ just before A_2's query $(4, X_4^i)$. We have already argued for a similar case in Case A.2 where we derive a contradiction to $(X_1, X_2, 1) \notin$ CompletedChains before the query $(2, X_2)$ by A_2.

Therefore, in all the cases where $G_4[X_4^i] \neq \perp$ before the ForceVal$(X_4^i, \cdot, 4)$ leads to a contradiction.

Therefore, if the lemma fails to hold for C then it must be the case that the Complete called on C is from line 30. We analyse this case next.

- Case B: Complete$(C = (X_1, X_2, 1), \cdot)$ was called from line 30 of Figure 6. We refer such a call to Complete as *delayed* chain completion. Delayed chain completions can happen in either of the Phases in G_2.

When C's completion was delayed (lines 11-13), $\text{Chains}[3, X_3] = (5, X_5)$ and $(X_1, X_2, 1) \in \text{Chains}[5, X_5]$ where $X_3 = G_2[X_2] \oplus X_1$ and $(X_5, X_6) = P[+, X_0, X_1]$ and $X_0 = G_1[X_1] \oplus X_2$. To complete the chain C, Complete will force $G_3[X_3]$ and $G_4[X_4]$ (X_4 is not yet defined) through ForceVal$(X_3, \cdot, 3)$ and ForceVal$(X_4, \cdot, 4)$ respectively. If the lemma fails to hold for C then either $G_3[X_3] \neq \perp$ or $G_4[X_4] \neq \perp$ before their respective ForceVal calls.

First, we argue that before the call ForceVal$(X_3, \cdot, 3)$, $G_3[X_3] = \perp$. Prior to this (delayed) call to Complete, $\text{Chains}[3, X_3] = (5, X_5)$ (because there are no overwrites in Chains$[3, \cdot]$ by Lemma 11). Such a (delayed) call to Complete is triggered on an A_2 query to either $(5, X_5)$ or $(3, X_3')$ or within AllComplete procedure in the Phase 2 such that $\text{Chains}[3, X_3'] = (5, X_5)$, whichever earlier. Here X_3' corresponds to any chain C' such that $(C', \cdot) \in \text{Chains}[5, X_5]$. Let us consider the earliest point in the execution when either A_2 queries $(5, X_5)$ or $(3, X_3')$ or there is a call to AllComplete such that $\text{Chains}[3, X_3'] = (5, X_5)$. We assume that $\text{Chains}[3, X_3']$ was assigned to $(5, X_5)$ due to a Func query on some (X_0', X_1') where $G_1[X_1'] \neq \perp$, $G_2[X_2'] \neq \perp$, $X_0' = G_1[X_1'] \oplus X_2'$ and $X_3' = G_2[X_2'] \oplus X_1'$. The other case when Func query is on some (X_5', X_6') is symmetrically handled.

We claim that for all X_3' such that $\text{Chains}[3, X_3'] = (5, X_5)$, $G_3[X_3'] \neq \perp$. The claim may not hold because X_3' got added to G_3 during a $\mathsf{Complete}$ call on some chain C'' by Sim. Note that C'' can be of the form $(X_1'', X_2'', 1)$ or $(X_4'', X_5'', 4)$ as $\mathsf{Complete}$ calls on either can lead to $\mathsf{ForceVal}(X_3', \cdot, 3)$. We refute the case when $C'' = (X_1'', X_2'', 1)$ and the other case can be similarly handled. More concretely for C'' it is the case that $G_2[X_2''] \oplus X_1'' = X_3'$. The chain C'' could be considered for completion only on a query $(2, X_2'')$ by A_2. At this point $G_1[X_1''] \neq \perp$ and $G_2[X_2'']$ was assigned to $f[2, X_2'']$. If A_2 issues $(2, X_2'')$ after $(2, X_2')$ and $G_2[X_2''] \oplus X_1'' = G_2[X_2'] \oplus X_1' = X_3'$ then the event $\mathsf{BadGutShot}$ is triggered on assignment $G_2[X_2''] \leftarrow f[2, X_2'']$. Similarly, if A_2 issues $(2, X_2'')$ earlier then $\mathsf{BadGutShot}$ is triggered on the assignment $G_2[X_2'] \leftarrow f[1, X_2']$. Since (f, ρ) is a good pair, our claim holds for all X_3'. Hence $G_3[X_3] = \perp$ before its $\mathsf{ForceVal}$ call.

Now, the partial chain C can either be a relevant chain or a non-relevant chain. Below, we analyse both cases separately.

- Case B.1: $C = (X_1, X_2, 1)$ is a non-relevant chain. If C is not a relevant chain then $X_5 \neq X_5^i$ for all $i \in [q_c]$. Otherwise the event $\mathsf{BadP2}$ would have occured on the Func query $(+, X_0 || X_1)$ made by Sim to schedule the delayed completion of C, where $X_0 = G_1[X_1] \oplus X_2$. Since this is a non-relevant chain's delayed completion, $G_5[X_5]$ gets assigned to $f[5, X_5]$ (line 27) just before the $\mathsf{Complete}$ call. Then for all $(C', (X_5, X_6')) \in \text{Chains}[5, X_5]$ we must have $G_4[X_4'] = \perp$ where $X_4' = G_5[X_5] \oplus X_6'$. Otherwise there would exist $(-, X_5, X_6') \in P$ and $X_5 \in G_5$ immediately after the assignment $G_5[X_5] \leftarrow f[5, X_5]$ such that $G_4[X_4'] \neq \perp$, which contradicts the goodness of the pair (f, ρ). Since for every X_4', $G_4[X_4'] = \perp$, therefore $G_4[X_4] = \perp$ as well.

  Therefore, if the lemma fails to hold for C, then it can only be that the $\mathsf{Complete}$ call (for the lemma does not hold) was a delayed completion of a relevant chain C. We analyse this case next.

- Case B.2: $C = (X_1, X_2, 1)$ is a relevant chain. If C is a relevant chain then $(X_5 = X_5^i \ \exists \ i \in [q_c])$ $\text{Chains}[5, X_5^i]$ only consists of tuples $(C', (X_5^i, X_6'))$ such that the partial chain C' is relevant. Otherwise, if there exist a non-relevant partial chain C' then the event $\mathsf{BadP2}$ would have occured on the Func query made by Sim to schedule the completion of C'. Now, since all C' are relevant chains then $X_4' = G_5[X_5^i] \oplus X_6'$ ($G_5[X_5^i]$ is defined before the control is transferred to A_2) for every chain C' in $\text{Chains}[5, X_5^i]$ are distinct, as $\mathsf{BadlyCollide}$ does not occur.
  Therefore, if the lemma fails to hold for C, then $G_4[X_4^i] \neq \perp$ before the $\mathsf{ForceVal}(X_4^i, \cdot, 4)$ call where $X_4^i = G_5[X_5^i] \oplus X_6^i$. We, infact, argue something stronger, that is, for all $(C', (X_5^i, X_6')) \in \text{Chains}[5, X_5^i]$, that $G_4[X_4'] = \perp$ where $X_4' = G_5[X_5^i] \oplus X_6'$. Let us assume that $G_4[X_4'] \neq \perp$ before the corresponding call to $\mathsf{Complete}$ on C'. As seen in case $A.3$, X_4' can get added to G_4 either during a chain completion performed by Sim or a direct query $(4.X_4')$ by A_2. Similar to the argument presented in case A.3.3. (for direct query) and the fact that (f, ρ) is a good pair (relevant chains cannot share X_4s), if $G_4[X_4'] \neq \perp$ then X_4' got added to G_4 during the completion of some non-relevant chain $C'' = (X_1'', X_2'', 1)$. Furthermore, the corresponding X_5'' (of chain C'') is such that $X_5'' \neq X_5^i$ as the event $\mathsf{BadP2}$ does not occur in a good execution. Therefore during the completion of chain C'', $G_5[X_5'']$ is assigned to $f[5, X_5'']$ and if $G_5[X_5''] \oplus X_6'' = X_4'$ then the event $\mathsf{BadOutside2}$ would occur, contradicting that (f, ρ) is a good pair. Therefore, we conclude that $G_4[X_4'] = \perp$ for every X_4' (as described

PROCEDURE $\mathsf{isTrue}(T)$:
foreach $(\sigma, X_k, X_{k+1}, X_{k'}, X_{k'+1}) \in T$ **do**
  if $\neg\mathsf{Check}(\sigma, X_k, X_{k+1}, X_{k'}, X_{k'+1})$ **then return** false
return true

PROCEDURE $\mathsf{Check}(\sigma, X_k, X_{k+1}, X_{k'}, X_{k'+1})$
if $\sigma = +$ **then**
  $U \leftarrow X_k, V \leftarrow X_{k+1}$
  for $i \in [1,2,3,4,5]$ **do**
    if $(Y, i, V) \notin T$ **then return** false
    $A \leftarrow Y \oplus U$, $U \leftarrow V$, $V \leftarrow A$
  if $(U, V) \neq (X_{k'}, X_{k'+1})$ **then return** false

else
  $V \leftarrow X_k, U \leftarrow X_{k+1}$
  for $i \in [5,4,3,2,1]$ **do**
    if $(Y, i, V) \notin T$ **then return** false
    $A \leftarrow Y \oplus U$, $U \leftarrow V$, $V \leftarrow A$
  if $(U, V) \neq (X_{k'}, X_{k'+1})$ **then return** false

return true

Fig. 23: The predicate isTrue defined on the transcript T.

above). Hence, $G_4[X_4] = \perp$ before the call $\mathsf{ForceVal}(X_4, \cdot, 4)$ during the delayed completion of $C = (X_1, X_2, 1)$.

Therefore even for a delayed chain completion of $C = (X_1, X_2, 1)$, the calls to $\mathsf{ForceVal}(X_l, \cdot, l)$ are such that $G_l[X_l] = \perp$ before the call.

We have now analysed all the cases for that call to Complete on C for which we assumed the lemma fails to hold. Since, we have arrived at a contradiction in each of these cases, the lemma indeed holds. □

Next, we define a predicate isTrue on the transcript (described in Figure 23) which captures that all queries made to Func $(\sigma, X_k, X_{k+1}, X_{k'}, X_{k'+1})$ are such that $(X_{k'}, X_{k'+1})$ look like the result of a Feistel Ψ_5 evaluation on (σ, X_k, X_{k+1}) given the transcript T of the execution. We prove that the isTrue returns true on a transcript T corresponding to an execution with a good pair (f, ρ).

Lemma 13. *For all* $(\sigma, X_k, X_{k+1}, X_{k'}, X_{k'+1}) \in T$ *that correspond to* Func *queries by* A_1, $\mathsf{Check}(\sigma, X_k, X_{k+1}, X_{k'}, X_{k'+1})$ *returns* true *at the end of the execution for* $(\sigma, k, k') \in \{(+, 0, 5), (-, 5, 0)\}$. ■

Proof. Consider any tuple $(\sigma, X_k, X_{k+1}, X_{k'}, X_{k'+1}) \in T$ which corresponds to the query by A_1. We prove the lemma for $(\sigma, k, k') = (+, 0, 5)$, that is, tuples of the form $(+, X_0, X_1, X_5, X_6)$. The other case of $(\sigma, k, k') = (+, 0, 5)$ can be handled symmetrically, and is omitted.

Since A_2 makes all primitive queries corresponding to the construction queries by A_1, then there exists at least one occurence of $(Y_1, 1, X_1)$ and $(Y_5, 5, X_5)$ in T. Consider the first occurrences of both the tuples, furthermore consider the earliest of the two.

Let us w.l.o.g. assume that the earlier tuple in the sequence T is $(Y_1, 1, X_1)$. Since this is the earliest call to $\mathsf{F}^{\mathrm{inner}}(1, X_1)$, inside $\mathsf{F}^{\mathrm{inner}}$, $G_1[X_1]$ gets assigned to $f[1, X_1]$. This defines $X_2 =$

$G_1[X_1] \oplus X_0$ such that $G_2[X_2] = \perp$ (otherwise on the assignment $G_1[X_1] \leftarrow f[1, X_1]$ the event $\mathsf{BadOutside}$ occurs in an execution with a good pair (f, ρ)). Hence the tuple $(Y_2, 2, X_2)$ hasn't occured in T at this point. Since A_2 makes all primitive queries there is at least one occurrence of $(Y_2, 2, X_2)$ in T.

At this point there are two cases possible: one where $(Y_5, 5, X_5)$ is the earliest tuple after $(Y_1, 1, X_1)$ in T and the other where $(Y_2, 2, X_2)$ is the earliest tuple after $(Y_1, 1, X_1)$ in T. We consider both these cases and show that Check returns true.

- Case A: $(Y_5, 5, X_5)$ is earlier. Since this is also the earliest call to $\mathsf{F}^{\mathrm{inner}}(5, X_5)$, we have the assignment $G_5[X_5] \leftarrow f[5, X_5]$. This defines $X_4 = G_5[X_5] \oplus X_6$ such that $G_4[X_4] = \perp$ (otherwise on the assignment $G_5[X_5] \leftarrow f[5, X_5]$ the event $\mathsf{BadOutside}$ would occur (in an execution with a good pair (f, ρ))). Hence the tuple $(\cdot, 4, X_4)$ hasn't occured in T yet. Since A_2 makes all primitive queries, there exists at least one ocurrence of $(Y_4, 4, X_4)$ in T. Finally, we consider the earlier of $(Y_2, 2, X_2)$ and $(Y_4, 4, X_4)$. We analyse the case when $(Y_2, 2, X_2)$ is the earlier tuple in T. The other case when $(Y_4, 4, X_2)$ occurs earlier than $(Y_2, 2, X_2)$ is symmetrical, and is omitted.

  We claim (prove below) that the tuple $(\cdot, 2, X_2)$ corresponds to a call to $\mathsf{F}^{\mathrm{inner}}$ on $(2, X_2)$. Moreover it was due to a query $(2, X_2)$ by A_2 to Prim. And since this is the earliest occurrence of $(\cdot, 2, X_2)$, $G_2[X_2] = \perp$ before the call to $\mathsf{F}^{\mathrm{inner}}$. To respond to A_2's query, Sim (lines 3-7 in Figure 6) issues a call to $\mathsf{F}^{\mathrm{inner}}$ where $G_2[X_2]$ is assigned to $f[2, X_2]$. After the call to $\mathsf{F}^{\mathrm{inner}}$, Sim considers all possible tuples in $G_1 \times \{X_2\}$ (line 4) and makes appropriate Func queries. Therefore, it issues $\mathsf{Func}(+, X_0 || X_1)$ and gets (X_5, X_6). Since $G_5[X_5] \neq \perp$ (as the tuple $(Y_5, 5, X_5)$ has already occurred in T), $\mathsf{Complete}$ is called on input $((X_1, X_2, 1), (X_5, X_6))$ (lines 9-10 in Figure 6). It is easy to see that after $\mathsf{Complete}$ returns, Check on $(+, X_0, X_1, X_5, X_6)$ is returns true. From Lemma 12 we know that there are no overwrites in G_k. Hence, Check will return true even at the end of the execution.

  Now, we get back to proving our claim that $(\cdot, 2, X_2)$ corresponds to a call to $\mathsf{F}^{\mathrm{inner}}$. Moreover this was because A_2 queries $(2, X_2)$. Let us assume that $(\cdot, 2, X_2)$ does not correspond to a call to $\mathsf{F}^{\mathrm{inner}}$. Therefore, it corresponds to a call to $\mathsf{ForceVal}(X_2, \cdot, 2)$. Since $\mathsf{ForceVal}$ is called only from inside $\mathsf{Complete}$, there exists a partial chain $C' = (X'_4, X'_5, 4)$ during whose completion the $\mathsf{ForceVal}$ call was made. Since (f, ρ) is a good pair, all relevant chains have distinct X_2s (as the event $\mathsf{BadlyCollide}$ does not occur). Since $(X_1, X_2, 1)$ is a relevant chain (because we are only concerned with relevant chains in this proof) then C' cannot be a relevant chain.
  Since C' is not a relevant chain, Sim at some point must have queried $\mathsf{Func}(-, X'_5, X'_6)$ to get (X'_0, X'_1). The query is answered by Func using ρ. Since (f, ρ) is a good pair it is the case that $G_1[X'_1] = \perp$ ($\overline{\mathsf{BadP}}$) and $X'_1 \neq X^i_1$ for $i \in [q_c]$ ($\overline{\mathsf{BadP2}}$). Moreover $G_1[X'_1] = \perp$ ensures that $(X'_4, X'_5, 4)$ is delayed for completion. Currently we are inside the call to $\mathsf{Complete}$ on $(X'_4, X'_5, 4)$. Hence, $G_1[X'_1] \leftarrow f[1, X'_1]$ and if $G_1[X'_1] \oplus X'_0 = X_2$ we would trigger the event $\mathsf{BadOutside2}$ which contradicts that the pair (f, ρ) is good. Hence $(Y_2, 2, X_2)$ cannot correspond to a call to $\mathsf{ForceVal}(X_2, \cdot, 2)$.
  Therefore for Case A, we have proved that the lemma holds.

- Case B: $(Y_2, 2, X_2)$ is earlier. By the exact same argument as above, we know that $(Y_2, 2, X_2)$ corresponds to a call to $\mathsf{F}^{\mathrm{inner}}$ and this was due to the query $(2, X_2)$ by A_2. Since this is the earliest call $\mathsf{F}^{\mathrm{inner}}(2, X_2)$, inside the call to $\mathsf{F}^{\mathrm{inner}}$, $G_2[X_2]$ is assigned to $f[2, X_2]$ and then Sim

considers all tuples (lines 4-7) in $G_1 \times \{X_2\}$ and makes appropriate queries to Func. Since $(Y_2, 2, X_2)$ is the earlier query we know that $G_5[X_5] = \perp$ at this point. Therefore $(X_1, X_2, 1)$ is delayed for completion (lines 11-13). Moreover, Chains$[3, X_3] = (5, X_5)$ where $X_3 = G_2[X_2] \oplus X_1$ and $((X_1, X_2, 1)), (X_5, X_6)) \in$ Chains$[5, X_5]$. Since A_2 issues all primitive queries then there exists at least one occurence of $(Y_3, 3, X_3)$ in T. Like before, we will consider the earlier of the two $(Y_3, 3, X_3)$ and $(Y_5, 5, X_5)$. Unlike before, the cases here are not symmetrically and hence we explicitly consider them in the following:

- Case B.1: $(Y_5, 5, X_5)$ is earlier. Since this is the earliest occurence of $(Y_5, 5, X_5)$ in T or equivalently the earliest call to $\mathsf{F}^{\text{inner}}$ on $(5, X_5)$, $G_5[X_5] = \perp$. This call to $\mathsf{F}^{\text{inner}}$ could be because of a direct query $(5, X_5)$ by A_2 to Prim or due to a direct query $(3, X_3')$ by A_2 to Prim such that Chains$[3, X_3'] = (5, X_5)$. It cannot be the case that $(Y_5, 5, X_5)$ was due to a call to $\mathsf{F}^{\text{inner}}$ by Sim from inside AllComplete. This is because A_2 makes all primitives queries and hence it would query $(5, X_5)$ before Sim gets to run AllComplete. In either case, Sim is going to execute lines 27-30 on $(5, X_5)$. Hence, Complete is called on $((X_1, X_2, 1), (X_5, X_6))$ in Chains$[5, X_5]$. It is easy to see that after Complete returns, Check on $(+, X_0, X_1, X_5, X_6)$ returns true. From Lemma 12 we know that there are no overwrites in G_k. Hence, Check will return true even at the end of the execution.

- Case B.2: $(Y_3, 3, X_3)$ is earlier. The tuple $(Y_3, 3, X_3)$ occurs in T either due to a $\mathsf{F}^{\text{inner}}$ query $(3, X_3)$ or due to a ForceVal call. We consider these two cases separately below:

  * Case B.2.1: $(Y_3, 3, X_3)$ corresponds to a ForceVal call. A call to ForceVal occurs only from inside Complete. Let us assume that Complete was called on $(C', (U, V))$. Furthermore, we let $C' = (X_1', X_2', 1)$ and $(U, V) = (X_5', X_6')$ (the other case is symmterical). Therefore, the chain C' would have been considered by Sim for completion on a query $(2, X_2')$ by A_2. Infact this must correspond to an $\mathsf{F}^{\text{inner}}$ call to $(2, X_2')$ and hence the tuple $(Y_2', 2, X_2')$ occurs in T.
    If $(Y_2', 2, X_2')$ occurs later in the sequence T than $(Y_2, 2, X_2)$ and $G_2[X_2'] \oplus X_1' = X_3$ then on assignment $G_2[X_2'] \leftarrow f[2, X_2']$ there exists $X_1' \in G_1$ and $(3, X_3) \in$ Chains such that $X_3 = G_2[X_2'] \oplus X_1'$. Since this (f, ρ) is a good pair, the event BadGutShot does not occur. Therefore, $(Y_2', 2, X_2')$ occurs before $(Y_2, 2, X_2)$ in T. If $(Y_2', 2, X_2')$ occurs earlier then depending on whether $X_5' \in G_5$, either X_3' gets added to G_3 or Chains$[3, X_3'] \leftarrow (5, X_5) \neq \perp$ where $X_3' \leftarrow G_2[X_2'] \oplus X_1'$. Finally, moving ahead in the T to the point where $(Y_2, 2, X_2)$ occurs and $G_2[X_2] \leftarrow f[2, X_2]$ if $X_3 = X_3'$ where $X_3 \leftarrow G_2[X_2] \oplus X_1$ we would trigger BadGutShot.
    Since the pair (f, ρ), the tuple $(Y_3, 3, X_3)$ in T cannot correspond to a call to ForceVal. The only other case, that is, $(Y_3, 3, X_3)$ corresponding to $\mathsf{F}^{\text{inner}}$ call is discussed next.

  * Case B.2.2: $(Y_3, 3, X_3)$ corresponds to $\mathsf{F}^{\text{inner}}$ query. The only case where the tuple $(Y_3, 3, X_3)$ may correspond to an $\mathsf{F}^{\text{inner}}$ query is due to a direct query $(3, X_3)$ by A_2. However, as Chains$[3, X_3] = (5, X_5)$, on a direct query $(3, X_3)$ Sim executes lines 31-32 and ends up calling itself on $(5, X_5)$. Therefore, the tuple $(Y_5, 5, X_5)$ occurs earlier than $(Y_3, 3, X_3)$ which is a contradiction. Hence, the tuple $(Y_3, 3, X_3)$ cannot correspond to $\mathsf{F}^{\text{inner}}$ query as well.

  Therefore, the tuple $(Y_3, 3, X_3)$ (in an execution with a good pair (f, ρ)) cannot occur in T before $(Y_5, 5, X_5)$. And the other case is already analysed in case B.1.

Therefore for case B, we have proved that the lemma holds.

□

Lemma 14. *For all* $(\sigma, X_k, X_{k+1}, X_{k'}, X_{k'+1}) \in T$, $\mathsf{Check}(\sigma, X_k, X_{k+1}, X_{k'}, X_{k'+1})$ *returns* true *at the end of the execution where* $(\sigma, k, k') \in \{(+, 0, 5), (-, 5, 0)\}$. *Moreover,* $\mathsf{isTrue}(T) = \mathsf{true}$. ■

Proof. In Lemma 13 we have already argued about the tuple $(\sigma, X_k, X_{k+1}, X_{k'}, X_{k'+1})$ that correspond to queries by A_1. Therefore, we will focus only on tuples that correspond to queries by Sim to Func. Moreover, when proving Lemma 13 we have seen that for every tuple corresponding to A_1 there is a tuple in T corresponding to the Func query made by Sim.

Therefore, we focus on tuples $(\sigma, X_k, X_{k+1}, X_{k'}, X_{k'+1})$ which correspond to non-relevant chains. We prove the lemma for $(\sigma, k, k') = (+, 0, 5)$, that is, tuples of the form $(+, X_0, X_1, X_5, X_6)$. Since this tuple corresponds to a non-relevant chain, there exists a direct query $(2, X_2)$ by A_2 (such that $G_2[X_2] = \perp$) which lead to Sim query Func on $(+, X_0||X_1)$. Since $G_2[X_2] = \perp$, $G_2[X_2]$ is assigned to $f[2, X_2]$ on A_2 and then Sim would have considered all tuples in $G_1 \times \{X_2\}$ and issued appropriate queries to Func. Since $(X_1, X_2, 1)$ is not a relevant chain, Func replies to $(+, X_0||X_1)$ using ρ and hence $G_5[X_5] = \perp$. This chain is then delayed for completion. Moreover, $\mathsf{Chains}[3, X_3] = (5, X_5)$ and $((X_1, X_2, 1), (X_5, X_6))$ is added to the set $\mathsf{Chains}[5, X_5]$ and $G_3[X_3] = \perp$.

Since Sim runs $\mathsf{AllComplete}$ in Phase 2, we know that at the end of the execution, $G_3[X_3] \neq \perp$ and $G_5[X_5] \neq \perp$. However, X_3 (resp., X_5) might get added to the table G_3 (resp. G_5) in Phase 1 as well. We analyse each of these possibilities below and show that there is a call to $\mathsf{Complete}$ on $(X_1, X_2, 1)$.

- Case A: Either $G_3[X_3] \neq \perp$ or $G_5[X_5] \neq \perp$ at the end of Phase 1. If at the end of Phase 1 either $X_3 \in G_3$ or $X_5 \in G_5$ then there must occur a tuple $(Y_3, 3, X_3)$ or $(Y_5, 5, X_5)$ in T. Consider the first occurrences of the two tuple and moreover consider the earlier among them.

  - Case A.1: $(Y_5, 5, X_5)$ is earlier. This tuple corresponds to a call to $\mathsf{F}^{\mathsf{inner}}$ on $(5, X_5)$. This could either be due to a direct query $(5, X_5)$ by A_2 or due to a direct query $(3, X_3')$ by A_2 such that $\mathsf{Chains}[3, X_3'] \leftarrow (5, X_5)$. In any case, Sim will be executing lines 27-30 and there is a call to $\mathsf{Complete}$ on $((X_1, X_2, 1), (X_5, X_6))$.

  - Case A.2: $(Y_3, 3, X_3)$ is earlier. This tuple may correspond to an $\mathsf{F}^{\mathsf{inner}}$ call to $(3, X_3)$ which is a result of a query $(3, X_3)$ by A_2. In this case, Sim executes lines 31-32 and ends up calling itself on $(5, X_5)$. We know by Lemma 11 that there are no overwrites in $\mathsf{Chains}[3, X_3]$ and hence it calls to precisely this $(5, X_5)$. Therefore by the above argument there is a call to $\mathsf{Complete}$. This tuple cannot correspond to a call to $\mathsf{ForceVal}(X_3, Y_3, 3)$ (an earlier argument made in Case B.2.1 of Lemma 13 suffices, we will trigger the event $\mathsf{BadGutShot}$).

- Case B: $G_3[X_3] = \perp$ and $G_5[X_5] = \perp$ at the end of Phase 1. If at the end of Phase 1, $G_3[X_3] = \perp$ and $G_5[X_5] = \perp$. Then in Phase 2, Sim runs $\mathsf{AllComplete}$ and hence there occurs a call to $\mathsf{Complete}$ on $(X_1, X_2, 1)$ as $((X_1, X_2, 1), (X_5, X_6)) \in \mathsf{Chains}[5, X_5]$.

Therefore in all cases, we have proved that there is a call to $\mathsf{Complete}$. It is easy to see that after $\mathsf{Complete}$ returns, Check on $(+, X_0, X_1, X_5, X_6)$ is going to return true. From Lemma 12 we know that there are no overwrites in G_k. Hence, Check will return true even at the end of the execution.

□

The randomness-mapping argument

After proving the properties of good executions, that is, executions of $\mathsf{G}_2^{f,\rho}$ with a good pair (f,ρ), we are ready to define the map τ that maps (f,ρ) to h such that the executions $\mathsf{G}_2^{f,\rho}$ and G_3^{h} are identical. This allows us to show the indistinguishability of G_2 and G_3 (proved in Lemma 17).

THE MAP τ:

Definition 9. *For $(f,\rho) \in \mathcal{F} \times \mathcal{R}$, $h \leftarrow \tau(f,\rho)$ is defined as follows:*

1. *Set $h = f$ and run $\mathsf{G}_2^{f,\rho}$ with $A = (A_1, A_2)$.*
2. *If for some (k, X) there is a $\mathsf{ForceVal}(X, Y, k)$ call then set $h[k, X] \leftarrow Y$ for the first such call.*

In the following lemma we finally prove that for a good pair (f,ρ) the transcript of $\mathsf{G}_2^{f,\rho}$ is identical to the transcript of G_3^{h} where $h = \tau(f,\rho)$.

Lemma 15. $T(\mathsf{G}_2^{f,\rho}) = T(\mathsf{G}_3^{h=\tau(f,\rho)})$. *Hence* $\mathsf{Pr}\left[\mathsf{G}_2^{f,\rho}\right] = \mathsf{Pr}\left[\mathsf{G}_3^{h}\right]$. ■

Proof. Let for ease of notation $T_2 = T(\mathsf{G}_2^{f,\rho})$ and $T_3 = T(\mathsf{G}_3^{h})$ and let T_{2j} (T_{3j}) denote the jth tuple in the sequence T_2 (T_3).

The first tuple in both transcripts is going to correspond to the query by A_1. We analyse when $T_{21} = (+, X_0, X_1, X_5, X_6)$ and the other case is symmetric. A_1 queries $\mathsf{Func}(+, X_0||X_1)$ which returns (X_5, X_6) by accesing $\rho[+, X_0, X_1]$. Since (f,ρ) is good and $\mathsf{isTrue}(T_2)$ is true (Lemma 14), we know that (X_5, X_6, X_0, X_1) obey the following relation:

$$X_2 = G_1[X_1] \oplus X_0;\ X_3 = G_2[X_2] \oplus X_1;\ X_4 = G_3[X_3] \oplus X_2;$$
$$X_5 = G_4[X_4] \oplus X_3;\ X_6 = G_5[X_5] \oplus X_4;$$

Since A is deterministic, it must be the case that $T_{31} = (+, X_0, X_1, X_5', X_6')$. Func performs the following computations on (X_0, X_1) to return (X_5', X_6').

$$X_2' = H_1[X_1] \oplus X_0;\ X_3' = H_2[X_2'] \oplus X_1;\ X_4 = H_3'[X_3'] \oplus X_2';$$
$$X_5' = H_4[X_4'] \oplus X_3';\ X_6' = H_5[X_5'] \oplus X_4';$$

Since (f,ρ) is good and by Lemma 12 there are no overwrites in the tables G_k (in G_2), the image h of the map is such that $G_k[X] = h[k, X]$. Since there are no overwrites in H_k, it is also the case that $H_k[X] = h[k, X]$. Therefore, $(X_5', X_6') = (X_5, X_6)$ and $T_{21} = T_{31}$.

Let us assume that $T_{2i} = T_{3i}$ for $i \in [j-1]$. We next argue about the equality of the jth tuple. Since the transcripts are identical till $(j-1)$, it must be the case that the jth tuple in both transcripts have the same format i.e. both of them correspond to either $\mathsf{F}^{\text{inner}}$,$\mathsf{Func}$ or $\mathsf{ForceVal}$.

- Case A: $T_{2j} = (Y, k, X)$ and it corresponds to $\mathsf{ForceVal}(X, Y, k)$. As transcripts are equivalent till $(j-1)$, and the view of Sim is identical till now it must be the case that if T_{2j} corresponds to a $\mathsf{ForceVal}$ call then $T_{2j} = T_{3j}$.

- Case B: $T_{2j} = (Y, k, X)$ and it corresponds to an $\mathsf{F}^{\text{inner}}$ call on (k, X). If this is not the first occurrence of the tuple $(\cdot, k, X)$ then the state of tables G_k is identical in both games (as transcripts are identical until this point). Therefore, $T_{2j} = T_{3j}$. Hence, let us consider that this is the first occurrence of the tuple (Y, k, X) in T_2. Since the transcripts are identical until this point so is the view of Sim, then $T_{3j} = (Y', k, X)$. As this is the first occurrence of (Y, k, X) in T_2 we have $Y = f[k, X] = G_k[X]$. Similarly in T_3 we have $Y' = h[k, X]$. By definition of the map τ, $Y = Y'$.

- Case C: $T_{2j} = (\sigma, X_k, X_{k+1}, X_{k'}, X_{k'+1})$. If Func replies to (σ, X_k, X_{k+1}) without accessing ρ then it must be the case that $T_{2j} = T_{3j}$. If Func replies using ρ, then by the same argument of the equality of T_{21} and T_{31}, we have that $T_{2j} = T_{3j}$.

Therefore, $T_2 = T_3$. Consider the last tuple in T_2 after which A_2 outputs the bit b'. Let it be the mth tuple. Since $T_2[1 \dots m] = T_3[1 \dots m]$, A_2 in G_3^h necessarily outputs the same guess b'. Hence the output distributions of $\mathsf{G}_2^{f,\rho}$ and G_3^h are identical. □

Next, we consider a key combinatorial property of the map τ restricted to the set $\mathsf{GOOD} \subseteq \mathcal{F} \times \mathcal{R}$.

$$\mathsf{GOOD} = \{(f, \rho) \in \mathcal{F} \times \mathcal{R} : (f, \rho) \text{ is a good pair}\} .$$

Lemma 16. *$\tau : \mathsf{GOOD} \to \mathcal{F}$ is $|\mathcal{R}|$-regular.* ■

Proof. Let $h \in \mathcal{F}$ have at least one good pre-image (f_h, ρ_h). In the game $\mathsf{G}_2^{f_h,\rho_h}$, both A_1 and Sim issue queries to Func. Some of these calls to Func may access the randomness ρ_h. Let α be the number of such calls to Func that access ρ_h. Therefore, from the entire table ρ_h only α indices are accessed during the entire execution. Let these α indices be $\{I_1, \dots, I_\alpha\}$.

For every chain completed by Sim there exists a unique query to Func which accesses the randomness ρ_h. If the completed chain is relevant then Func query (that accesses ρ_h) is by A_1. For non-relevant chains that are completed the Func query (that accesses ρ_h) is by Sim. Moreover, because the Sim at the end executes the $\mathsf{AllComplete}$ function and the predicate isTrue on the transcript T of the execution is true (Lemma 14), Sim completes chains corresponding to all its Func queries. This implies that there are exactly α chains that are completed during the execution of $\mathsf{G}_2^{f_h,\rho_h}$. Therefore, there were exactly 2α calls to $\mathsf{ForceVal}$. Since there are no overwrites in an execution (Lemma 12) with a good pair, there are 2α indices in f that are not carried over to h (in the definition of τ). These correspond to all the $\mathsf{ForceVal}(X, \cdot, k)$ calls. Let these 2α indices be $\{J_1, \dots, J_{2\alpha}\}$.

Consider a $\mathcal{FR}_h \subseteq \mathcal{F} \times \mathcal{R}$ such that every element $(f, \rho) \in \mathcal{FR}_h$ are as follows:

$$\begin{aligned} f[x] &= f_h[x] && \forall x \in [5] \times \{0,1\}^n \setminus \{J_1, \dots, J_{2\alpha}\} \\ \rho[X] &= \rho_h[X] && \forall X \in \{I_1, \dots, I_\alpha\}. \end{aligned} \tag{40}$$

We claim that $|\mathcal{FR}_h| = |\mathcal{R}|$. To show that τ is $|\mathcal{R}|$-regular, we need to show that $\mathcal{FR}_h \subseteq \mathsf{GOOD}$ and that $\tau(f, \rho) \neq h$ where $(f, \rho) \notin \mathcal{FR}$. Since $(f, \rho) \in \mathcal{FR}_h$ agree with (f_h, ρ_h) on all indices that are accessed (which define the goodness of the pair), therefore if (f_h, ρ_h) is a good pair then so is (f, ρ). Consider $(f, \rho) \notin \mathcal{FR}$, then either f disagrees with f_h on an index other than J_i or ρ disagrees with ρ_h on an index I_j. It is easy to see that in either case there exists an index L such that $h[L] \neq \tau(f, \rho)[L]$. □

We are now ready to consider $\Delta(\mathsf{G}_2, \mathsf{G}_3)$.

$$\begin{aligned}
\Pr[\mathsf{G}_2] &= \Pr\left[(f,\rho) \leftarrow\!\!\$\ \mathcal{F}\times\mathcal{R} : \mathsf{G}_2^{f,\rho}\right] \\
&= \sum_{(f,\rho)\in\mathsf{GOOD}} \Pr[(f,\rho)]\cdot\Pr\left[\mathsf{G}_2^{f,\rho}\right] + \sum_{(f,\rho)\in\overline{\mathsf{GOOD}}} \Pr[(f,\rho)]\cdot\Pr\left[\mathsf{G}_2^{f,\rho}\right] \\
&\le \sum_{(f,\rho)\in\mathsf{GOOD}} \Pr[(f,\rho)]\cdot\Pr\left[\mathsf{G}_2^{f,\rho}\right] + \Pr\left[(f,\rho)\leftarrow\!\!\$\ \mathcal{F}\times\mathcal{R} : (f,\rho)\in\overline{\mathsf{GOOD}}\right] \qquad (41)\\
&= \sum_{(f,\rho)\in\mathsf{GOOD}} \Pr[(f,\rho)]\cdot\Pr\left[\mathsf{G}_3^{h=\tau(f,\rho)}\right] \\
&\quad + \Pr\left[(f,\rho)\leftarrow\!\!\$\ \mathcal{F}\times\mathcal{R} : (f,\rho)\in\overline{\mathsf{GOOD}}\right] ,
\end{aligned}$$

where the last equality is due to Lemma 15.

$$\begin{aligned}
\Pr[\mathsf{G}_2] &\le \frac{1}{|\mathcal{F}|\times|\mathcal{R}|}\sum_{(f,\rho)\in\mathsf{GOOD}} \Pr\left[\mathsf{G}_3^{h=\tau(f,\rho)}\right] + \Pr\left[(f,\rho)\leftarrow\!\!\$\ \mathcal{F}\times\mathcal{R} : (f,\rho)\in\overline{\mathsf{GOOD}}\right] \\
&= \frac{1}{|\mathcal{F}|\times|\mathcal{R}|}\cdot|\mathcal{R}|\cdot\sum_{h\in\tau(\mathsf{GOOD})}\Pr\left[\mathsf{G}_3^{h}\right] + \Pr\left[(f,\rho)\leftarrow\!\!\$\ \mathcal{F}\times\mathcal{R} : (f,\rho)\in\overline{\mathsf{GOOD}}\right] ,
\end{aligned} \qquad (42)$$

where the last equality is due to Lemma 16.

$$\begin{aligned}
\Pr[\mathsf{G}_2] &\le \frac{1}{|\mathcal{F}|}\cdot\sum_{h\in\tau(\mathsf{GOOD})}\Pr\left[\mathsf{G}_3^{h}\right] + \Pr\left[(f,\rho)\leftarrow\!\!\$\ \mathcal{F}\times\mathcal{R} : (f,\rho)\in\overline{\mathsf{GOOD}}\right] \\
&= \sum_{h\in\tau(\mathsf{GOOD})}\Pr[h]\cdot\Pr\left[\mathsf{G}_3^{h}\right] + \Pr\left[(f,\rho)\leftarrow\!\!\$\ \mathcal{F}\times\mathcal{R} : (f,\rho)\in\overline{\mathsf{GOOD}}\right] \qquad (43)\\
&\le \Pr[\mathsf{G}_3] + \Pr\left[(f,\rho)\leftarrow\!\!\$\ \mathcal{F}\times\mathcal{R} : (f,\rho)\in\overline{\mathsf{GOOD}}\right]
\end{aligned}$$

Lemma 17. $\Delta(\mathsf{G}_2, \mathsf{G}_3) \le 351q^6/2^n$. ■

Proof. From Equation 43 and Lemma 10 the lemma holds. □

C.3 Indistinguishability of G_3 and G_4

Lemma 18. *In* G_3, Prim *always replies with* $h[k, X]$ *for any query* (k, X). ■

Proof. Sim in G_3^h responds to (k, X) with $G_k[X]$. During the execution, Sim sets $G_k[X] \leftarrow H_k[X]$ for all $k \in \{1, 5\}$. Therefore for such assignments the lemma holds as $H_k[X] = h[k, X]$ throughout the execution of G_3. For $k \in \{2, 3, 4\}$ there are two cases: (a) $G_k[X]$ is assigned to $H_k[X]$ or (b) $G_k[X]$ is forced to a value Y in a call to $\mathsf{ForceVal}(X, Y, k)$. For the case (a) the lemma holds by the same argument as $k \in \{1, 5\}$. For case (b) some analysis is in order. A call to ForceVal occurs only when Complete gets called on some partial chain C. This is true for both type of calls to Complete i.e. direct or immediate. We will consider the case when $C = (X_1, X_2, 1)$ and hence argue about $k \in \{3, 4\}$. The other case $C = (X_4, X_5, 4)$ can be symmetrically handled.

For $\mathsf{Complete}(X_1, X_2, 1)$ there was a query to Func i.e. $\mathsf{Func}(+, X_0||X_1)$ where $X_0 = G_1[X_1] \oplus X_2$. To answer this query Func evalutes the Feistel construction forward by accessing H (hence h). Then it must be the case that both calls to $\mathsf{ForceVal}(X, Y, k)$ it holds that $Y = h[k, X]$. Therefore $G_k[X] = H_k[X] = h[k, X]$ for all (k, X) such that $G_k[X] \neq \perp$. Hence the lemma holds.

□

Lemma 19. $\Delta(\mathsf{G}_3, \mathsf{G}_4) = 0$. ■

Proof. Due to Lemma 18 the lemma holds. □

From Lemmas 3,17,19 we have that $\Delta(\mathsf{G}_1, \mathsf{G}_4) \leq 360q^6/2^n$ where Sim makes at most $2q^2$ queries.

D Proofs for Section 6

Proof (Theorem 6). The proof proceeds via a sequence of five games $\mathsf{G}_0(\lambda), \ldots, \mathsf{G}_4(\lambda)$, which are all described in Figure 24. In all games, the oracle Prim models the ideal cipher, where for convenience the canonical way of efficiently simulating such cipher is relegated into $\mathsf{InnerPrim}$. We start the initial game, $\mathsf{G}_0(\lambda)$, which models the real-world interaction of S, D in the ideal cipher model for the psPRP game $\mathsf{psPR}^{S,D}_{\mathsf{F},\mathbf{P}}(\lambda)$, i.e., when the source S interacts with $\mathsf{F.Eval}(k_i, \cdot, \cdot)$. The only difference is that the keys (like in all subsequent games) are sampled from $\mathcal{D}(2^{s(\lambda)}, n)$, which is the set of all vectors of n *distinct* $s(\lambda)$-bit strings.

Then, Game $\mathsf{G}_1(\lambda)$ introduces a modified oracle $\mathcal{O}$, which behaves as a family of independent permutations, i.e., $\mathcal{O}(i, \cdot, \cdot)$ replies corresponding to forward and backward queries to a fresh random permutation. However, we also modify the game so that ideal-cipher queries $\mathsf{Prim}(k_i, \cdot, \cdot)$ are going to be answered by $\mathcal{O}(i, \cdot, \cdot)$. Note that this is not affecting the behavior of the experiment *at all*, we have just modified the location where the permutation implemented by $\mathsf{Prim}(k_i, \cdot, \cdot)$ and $\mathcal{O}(i, \cdot, \cdot)$ is managed. Therefore, we have just argued that

$$\mathsf{Pr}\left[\mathsf{G}_0(\lambda)\right] = \mathsf{Pr}\left[\mathsf{G}_1(\lambda)\right] \ . \tag{44}$$

In $\mathsf{G}_1(\lambda)$, we are also going to add a bad flag which does not affect the behavior of the game, but is set when the source S makes a Prim-query of the form $(k_i, \star, \star)$, for $i \in [n]$. (In contrast, note that D can easily make such a query, thus the bad flag is set only as long as done is false, which indicates the control flow has not been passed on to D yet.)

D.1 Proof of Theorem 6

One first key transition happens in $\mathsf{G}_2(\lambda)$. The main difference is that we are going to now decouple the ideal cipher from the oracle $\mathcal{O}$ *as long as* done *is false*. In the case the bad flag is set while answering $\mathsf{Prim}(k_i, \sigma, x)$, we are not going to respond any more using $\mathcal{O}(i, \sigma, x)$ as in $\mathsf{G}_2(\lambda)$. Rather, we are going to create an independent entry in the table T_{E}, and use that one for our response. Clearly, this modification in behavior is only noticed if the source S provokes bad to be set, as otherwise $\mathsf{G}_1(\lambda)$ and $\mathsf{G}_2(\lambda)$ are equivalent until bad. Thus,

$$\mathsf{Pr}\left[\mathsf{G}_1(\lambda)\right] - \mathsf{Pr}\left[\mathsf{G}_2(\lambda)\right] \leq \mathsf{Pr}\left[\mathsf{G}_1(\lambda) \text{ sets } \mathsf{bad}\right] = \mathsf{Pr}\left[\mathsf{G}_2(\lambda) \text{ sets } \mathsf{bad}\right] \ . \tag{45}$$

MAIN $\mathsf{G}_i(\lambda)$:
$\mathsf{done} \leftarrow \mathsf{false}$
$(1^n, t) \leftarrow\!\!\$\ S^{\mathsf{Prim}}(1^\lambda, \varepsilon)$
$k_1, \ldots, k_n \leftarrow\!\!\$\ \mathcal{D}(2^{s(\lambda)}, n)$
$L \leftarrow\!\!\$\ S^{\mathcal{O},\mathsf{Prim}}(1^\lambda)$
$\mathsf{done} \leftarrow \mathsf{true}$
$b' \leftarrow\!\!\$\ D^{\mathsf{Prim}}(1^\lambda, k_1, \ldots, k_n, L)$
return b'

ORACLE $\mathcal{O}(i, \sigma, x)$:
// $\mathsf{G}_0(\lambda)$
return $\mathsf{Prim}(k_i, \sigma, x)$

ORACLE $\mathcal{O}(i, \sigma, x)$:
// $\mathsf{G}_1(\lambda)$ – $\mathsf{G}_4(\lambda)$
if $T_{\mathsf{P}}[i, \sigma, x] = \perp$ **then**
  $y \xleftarrow{\$} \{0,1\}^\lambda \setminus P_i^\sigma$
  $T_{\mathsf{P}}[i, \sigma, x] \leftarrow y$
  $T_{\mathsf{P}}[i, \overline{\sigma}, T_{\mathsf{P}}[k, \sigma, x]] \leftarrow x$
  $P_i^\sigma \xleftarrow{\cup} \{y\}$; $P_i^{\overline{\sigma}} \xleftarrow{\cup} \{x\}$
return $T_{\mathsf{P}}[i, \sigma, x]$

ORACLE $\mathsf{Prim}(k, \sigma, x)$:
// $\mathsf{G}_0(\lambda)$
return $\mathsf{InnerPrim}(k, \sigma, x)$

ORACLE $\mathsf{Prim}(k, \sigma, x)$:
// $\mathsf{G}_1(\lambda)$
if $\exists i \in [n]$: $k = k_i$ **then**
  if $\neg$done **then**
    bad $\leftarrow$ true
  return $\mathcal{O}(i, \sigma, x)$
return $\mathsf{InnerPrim}(k, \sigma, x)$

ORACLE $\mathsf{Prim}(k, \sigma, x)$:
// $\mathsf{G}_2(\lambda)$
if $\exists i \in [n]$: $k = k_i$ **then**
  if $\neg$done **then**
    bad $\leftarrow$ true
  else return $\mathcal{O}(i, \sigma, x)$
return $\mathsf{InnerPrim}(k, \sigma, x)$

ORACLE $\mathsf{Prim}(k, \sigma, x)$: // $\mathsf{G}_3(\lambda)$
if $\exists i \in [n]$: $k = k_i$ **then**
  if $\neg$done **then**
    bad $\leftarrow$ true
  else return $\mathcal{O}'(i, \sigma, x)$
return $\mathsf{InnerPrim}(k, \sigma, x)$

ORACLE $\mathsf{Prim}(k, \sigma, x)$:
// $\mathsf{G}_4(\lambda)$
if $\exists i \in [n]$: $k = k_i$ **then**
  if $\neg$done **then**
    bad $\leftarrow$ true
  return $\mathcal{O}'(i, \sigma, x)$
return $\mathsf{InnerPrim}(k, \sigma, x)$

PROC. $\mathsf{InnerPrim}(k, \sigma, x)$:
// $\mathsf{G}_0(\lambda)$ - $\mathsf{G}_4(\lambda)$
if $T_{\mathsf{E}}[k, \sigma, x] = \perp$ **then**
  $y \xleftarrow{\$} \{0,1\}^\lambda \setminus E_k^\sigma$
  $T_{\mathsf{E}}[k, \sigma, x] \leftarrow y$
  $T_{\mathsf{E}}[k, \overline{\sigma}, T_{\mathsf{E}}[k, \sigma, x]] \leftarrow x$
  $E_k^\sigma \xleftarrow{\cup} \{y\}$; $E_k^{\overline{\sigma}} \xleftarrow{\cup} \{x\}$
return $T_{\mathsf{E}}[k_i, \sigma, x]$

Fig. 24: Description of games from the proof of Theorem 6. Here, $\overline{+} = -$ and $\overline{-} = +$, for notational convenience. The MAIN procedure remains the same throughout $\mathsf{G}_0(\lambda), \ldots, \mathsf{G}_4(\lambda)$. Also, $\mathcal{O}'$ denotes an oracle behaving identical to $\mathcal{O}$, but using independent randomness and tables.

It is easiest to bound $\mathsf{Pr}\left[\mathsf{G}_2(\lambda) \text{ sets } \mathsf{bad}\right]$, as in this experiment, the source S's view up to its output is independent of $k_1, \ldots, k_n$, and thus by a standard argument, the probability any of its q queries hits one of these keys is at most $N(\lambda)q/2^{s(\lambda)}$, i.e.,

$$\mathsf{Pr}\left[\mathsf{G}_2(\lambda) \text{ sets } \mathsf{bad}\right] \leq \frac{N(\lambda)q}{2^{s(\lambda)}} \ . \tag{46}$$

Now, the two final games $\mathsf{G}_3(\lambda)$ and $\mathsf{G}_4(\lambda)$ are obtained from $\mathsf{G}_2(\lambda)$ and $\mathsf{G}_1(\lambda)$, respectively, by replacing calls to $\mathcal{O}$ within Prim queries with calls to $\mathcal{O}'$, an independent copy of $\mathcal{O}$, which keeps its own table $T_{\mathsf{P}'}$. It is in particular easy to see that $\mathsf{G}_4(\lambda)$ behaves as the ideal-world case ($b = 0$) in the psPRP game $\mathsf{psPR}^{S,D}_{\mathsf{F},\mathsf{P}}(\lambda)$, since S only interacts with $\mathcal{O}$ (which is random) and Prim behaves consistently with an independent ideal cipher, *except* that keys are sampled distinct. Also, by a similar argument to the one above, G_3 and G_4 are identical until bad, and the probability of setting bad can be bounded similarly, and thus

$$\mathsf{Pr}\left[\mathsf{G}_3(\lambda)\right] - \mathsf{Pr}\left[\mathsf{G}_4(\lambda)\right] \leq \mathsf{Pr}\left[\mathsf{G}_3(\lambda) \text{ sets } \mathsf{bad}\right] = \mathsf{Pr}\left[\mathsf{G}_4(\lambda) \text{ sets } \mathsf{bad}\right] \leq \frac{N(\lambda)q}{2^{s(\lambda)}} \ . \tag{47}$$

Thus, combining (44), (45), (46), and (47),

$$\begin{aligned}\mathsf{Adv}^{\mathsf{pspr}[\mathbf{P}]}_{\mathsf{F},S,D}(\lambda) &\leq \frac{2N(\lambda)^2}{2^{s(\lambda)}} + \mathsf{Pr}\left[\mathsf{G}_0(\lambda)\right] - \mathsf{Pr}\left[\mathsf{G}_4(\lambda)\right] \\ &\leq \frac{2N(\lambda)^2}{2^{s(\lambda)}} + \frac{2N(\lambda)q}{2^{s(\lambda)}} + \mathsf{Pr}\left[\mathsf{G}_2(\lambda)\right] - \mathsf{Pr}\left[\mathsf{G}_3(\lambda)\right] \ .\end{aligned} \tag{48}$$

We are going to show now the existence of a reset adversary R for the source S (in the ideal cipher model, i.e., with access to an oracle Prim implementing an ideal cipher), such that

$$\mathsf{Pr}\left[\mathsf{G}_2(\lambda)\right] - \mathsf{Pr}\left[\mathsf{G}_3(\lambda)\right] = \mathsf{Adv}^{\mathsf{reset}[\mathbf{P}]}_{S,R}(\lambda) \ . \tag{49}$$

The adversary R is straightforward: Given leakage $L \in \{0,1\}^*$ from S, inputs 1^λ, 1^n, and oracle access to $\mathcal{O}$ (i.e., this is the oracle $\mathcal{O}$ from the reset-security game) and Prim, R initially generates random keys $k_1, \ldots, k_n \xleftarrow{\$} \mathcal{D}(2^{s(\lambda)}, n)$. Then, it runs D on input 1^λ, $k_1, \ldots, k_n$, and L, and whenever D makes a Prim query (k, σ, x), R answers it with its own Prim oracle if $k \notin \{k_1, \ldots, k_n\}$, and answers it as $\mathcal{O}(i, \sigma, x)$ otherwise. Then, (49) easily follows by inspection, since the simulated experiments are identical. (In particular, it is crucial to observe that $\mathcal{O}'$ in $\mathsf{G}_3(\lambda)$ is only used *after* done is set to true!) □

D.2 Proof of Theorem 7

Proof (Theorem 7). The proof proceeds via a sequence of five games $\mathsf{G}_0, \ldots, \mathsf{G}_4$, which are all described in Figure 25. In all games, the oracle Prim models the random permutation accessed by S and D. The initial game $\mathsf{G}_0(\lambda)$ which models the real-world interaction of S, D in the ideal cipher model for the psPRP game $\mathsf{psPR}^{S,D}_{\mathsf{EM},\mathbf{P}}(\lambda)$, i.e., when the source S interacts with $\mathsf{EM.Eval}(k_i, \cdot, \cdot)$.

Then, Game G_1 solely modifies how Prim queries are answered, in particular dropping the permutation requirement. Note that here it is possible that for example a forward query $\mathsf{Prim}(+, x)$ returns a y which was already returned earlier for a different query $\mathsf{Prim}(+, x')$, or it was queried as $\mathsf{Prim}(-, y)$, and returned x'. In this case, the table value is overwritten, and the next $\mathsf{Prim}(-, y)$ query will return x, instead of x'. Still, by a standard argument, it is not hard to see

$$\mathsf{Pr}\left[\mathsf{G}_0(\lambda)\right] - \mathsf{Pr}\left[\mathsf{G}_1(\lambda)\right] \leq \frac{q^2}{2^\lambda} \ , \tag{50}$$

where remember that q is the overall number of queries in the whole game, to either $\mathcal{O}$ or Prim, by S and D.

Then, Game $\mathsf{G}_2(\lambda)$ modifies the oracle Prim to keep an own table T_P. In particular, $T_P[i, \sigma, x]$ contains the value returned upon query $\mathcal{O}(i, \sigma, x)$. However, this will be implemented in a way that things are kept consistent, in particular:

- If $T_P[i, \sigma, x] \neq \perp$ and $T_\pi[\sigma, x \oplus k_i]$ are defined, then they are equal.
- If $\mathcal{O}(i, \sigma, x)$ and $\mathcal{O}(i', \sigma, x')$ are both queried such that $x \oplus k_i = x' \oplus k_{i'}$, then $T_P[i, \sigma, x] \oplus T_P[i', \sigma, x'] = k_i \oplus k_j$.

This ensures in particular that G_2 and G_1 behave *identically*, and thus $\mathsf{Pr}\left[\mathsf{G}_1(\lambda)\right] = \mathsf{Pr}\left[\mathsf{G}_2(\lambda)\right]$. In G_2, we also introduce a flag bad which is set to true whenever the code needs to take care of setting T_π and T_P consistently.

MAIN $\mathsf{G}_i(\lambda)$:
$\mathsf{done} \leftarrow \mathsf{false}$
$(1^n, t) \leftarrow\!\!\$\ S^{\mathsf{Prim}}(1^\lambda, \varepsilon)$
$k_1, \ldots, k_n \xleftarrow{\$} \{0,1\}^\lambda$
$L \leftarrow\!\!\$\ S^{\mathcal{O},\mathsf{Prim}}(1^\lambda)$
$\mathsf{done} \leftarrow \mathsf{true}$
$b' \leftarrow\!\!\$\ D^{\mathsf{Prim}}(1^\lambda, k_1, \ldots, k_n, L)$
return b'

ORACLE $\mathcal{O}(i, \sigma, x)$: // $\mathsf{G}_2(\lambda)$
if $T_P[i, \sigma, x] = \bot$ **then**
 if $T_\pi[\sigma, x \oplus k_i] \neq \bot$ **then**
  $\mathsf{bad} \leftarrow \mathsf{true}$
  $T_P[i, \sigma, x] \leftarrow T_\pi[\sigma, x \oplus k_i] \oplus k_i$
  $T_P[i, \overline{\sigma}, T_P[i, \sigma, x]] \leftarrow x$
 else
  if $\exists i : T_P[\sigma, x \oplus k_i \oplus k_j] \neq \bot$ **then**
   $\mathsf{bad} \leftarrow \mathsf{true}$
   $y \leftarrow T_P[\sigma, x \oplus k_i \oplus k_j] \oplus k_i \oplus k_j$
  else $y \xleftarrow{\$} \{0,1\}^\lambda$
  $T_P[i, \sigma, x] \leftarrow y$
  $T_P[i, \overline{\sigma}, y] \leftarrow x$
return $T_P[i, \sigma, x]$

ORACLE $\mathcal{O}(i, \sigma, x)$: // $\mathsf{G}_3(\lambda)$
if $T_P[i, \sigma, x] = \bot$ **then**
 if $T_\pi[\sigma, x \oplus k_i] \neq \bot$ **then**
  $\mathsf{bad} \leftarrow \mathsf{true}$
 if $\exists i : T_P[\sigma, x \oplus k_i \oplus k_j] \neq \bot$ **then**
  $\mathsf{bad} \leftarrow \mathsf{true}$
 $y \xleftarrow{\$} \{0,1\}^\lambda$
 $T_P[i, \sigma, x] \leftarrow y$
 $T_P[i, \overline{\sigma}, y] \leftarrow x$
return $T_P[i, \sigma, x]$

ORACLE $\mathcal{O}(i, \sigma, x)$: // $\mathsf{G}_4(\lambda)$
if $T_P[i, \sigma, x] = \bot$ **then**
 $T_P[i, \sigma, x] \leftarrow y \setminus P_i^\sigma$
 $T_P[i, \overline{\sigma}, y] \leftarrow x$
 $P_i^\sigma \xleftarrow{\cup} \{x\}$; $P_i^{\overline{\sigma}} \xleftarrow{\cup} \{y\}$
return $T_P[i, \sigma, x]$

ORACLE $\mathcal{O}(i, \sigma, x)$:
// $\mathsf{G}_0(\lambda)$, $\mathsf{G}_1(\lambda)$
return $k_i \oplus \mathsf{Prim}(\sigma, x \oplus k_i)$

ORACLE $\mathsf{Prim}(\sigma, x)$:
// $\mathsf{G}_0(\lambda), \mathsf{G}_4(\lambda)$
if $T_\pi[\sigma, x] = \bot$ **then**
 $y \xleftarrow{\$} \{0,1\}^\lambda \setminus \Pi^\sigma$
 $T_\pi[\sigma, x] \leftarrow y$
 $T_\pi[\overline{\sigma}, y] \leftarrow x$
 $P^\sigma \xleftarrow{\cup} \{x\}$; $P^{\overline{\sigma}} \xleftarrow{\cup} \{y\}$
return $T_\pi[\sigma, x]$

ORACLE $\mathsf{Prim}(\sigma, x)$: // $\mathsf{G}_1(\lambda)$
if $T_\pi[\sigma, x] = \bot$ **then**
 $y \xleftarrow{\$} \{0,1\}^\lambda$
 $T_\pi[\sigma, x] \leftarrow y$
 $T_\pi[\overline{\sigma}, y] \leftarrow x$
return $T_\pi[\sigma, x]$

ORACLE $\mathsf{Prim}(\sigma, x)$: // $\mathsf{G}_2(\lambda)$
if $T_\pi[\sigma, x] = \bot$ **then**
 if $\exists i : T_\pi[\sigma, x \oplus k_i] \neq \bot$ **then** $\mathsf{bad} \leftarrow \mathsf{true}$
  $T_\pi[\sigma, x] \leftarrow T_P[i, \sigma, x \oplus k_i] \oplus k_i$
  $T_\pi[\overline{\sigma}, T_\pi[\sigma, x]] \leftarrow x$
 else
  $y \xleftarrow{\$} \{0,1\}^\lambda$
  $T_\pi[\sigma, x] \leftarrow y$
  $T_\pi[\overline{\sigma}, y] \leftarrow x$
return $T_\pi[\sigma, x]$

ORACLE $\mathsf{Prim}(\sigma, x)$: // $\mathsf{G}_3(\lambda)$
if $T_\pi[\sigma, x] = \bot$ **then**
 if $\exists i : T_\pi[\sigma, x \oplus k_i] \neq \bot$ **then** $\mathsf{bad} \leftarrow \mathsf{true}$
 $y \xleftarrow{\$} \{0,1\}^\lambda$
 $T_\pi[\sigma, x] \leftarrow y$
 $T_\pi[\overline{\sigma}, y] \leftarrow x$
return $T_\pi[\sigma, x]$

Fig. 25: Description of games from the proof of Theorem 7. Here, $\overline{+} = -$ and $\overline{-} = +$, for notational convenience. The MAIN procedure remains the same throughout $\mathsf{G}_0(\lambda), \ldots, \mathsf{G}_4(\lambda)$.

Game G_3 is identical to G_2, except that the game does not attempt to keep T_P and T_π consistent. Whenever there is need for setting values consistently, however, the game still sets the flag bad to true. Therefore, G_2 and G_3 are equivalent until bad, and thus

$$\mathsf{Pr}\left[G_2(\lambda)\right] - \mathsf{Pr}\left[\mathsf{G}_3(\lambda)\right] \leq \mathsf{Pr}\left[\mathsf{G}_3(\lambda) \text{ sets } \mathsf{bad}\right] \ . \tag{51}$$

We will return to analyzing $\mathsf{Pr}\left[\mathsf{G}_2(\lambda) \text{ sets } \mathsf{bad}\right]$ later. We first conclude the sequence of games with Game G_4, which re-introduces the permutation contraint in T_P and T_π, thus making the resulting game identical to the ideal-world execution of Game $\mathsf{psPR}^{S,D}_{\mathsf{EM},\mathbf{P}}(\lambda)$. Also, again by a simple collision argument, it is not hard to see that

$$\mathsf{Pr}\left[\mathsf{G}_3(\lambda)\right] - \mathsf{Pr}\left[\mathsf{G}_4(\lambda)\right] \leq \frac{q^2}{2^\lambda} \ . \tag{52}$$

Therefore, we can combine (50), (51), and (52), and see that

$$\mathsf{Adv}^{\mathsf{pspr}[\mathbf{P}]}_{\mathsf{EM},S,D}(\lambda) = \mathsf{Pr}\left[G_0(\lambda)\right] - \mathsf{Pr}\left[\mathsf{G}_4(\lambda)\right] \leq \frac{2q^2}{2^\lambda} + \mathsf{Pr}\left[\mathsf{G}_3(\lambda) \text{ sets } \mathsf{bad}\right] \ . \tag{53}$$

Upper bounding the probability of setting bad. We consider now game G_3, and in particular differentiate between provoking bad while S queries $\mathcal{O}$ or Prim, and provoking bad as a result of one of D's Prim queries. We refer to the first event as Bad_1, to the latter as Bad_2.

For Bad_1, let us look at the experiment up to the point where S terminates with its leakage output L. In particular, in G_3 the distribution of the outputs of S queries does not depend on the keys $k_1, \ldots, k_n$, and on bad being set. Therefore, we can just assume that all queries have been made by S, and then check what is the probability that bad would have been set as a result of these queries when we sample $k_1, \ldots, k_n$ at the end of S's execution.

Concretely, for every one of the (at most q) Prim queries (σ, x), the probability that $(i, \sigma, x \oplus k_i)$ for $i \in [n]$ is set in T_P is at most $N(\lambda)q/2^\lambda$. Moreover, we need to establish the probability that for an $\mathcal{O}$ query (i, σ, x), there exists j such that T_P is already defined on $(j, \sigma, x \oplus k_i \oplus k_j)$. Again, this probability can easily be upper bounded by $N(\lambda)q/2^\lambda$. The union bound yields

$$\mathsf{Pr}\left[\mathsf{Bad}_1\right] \leq \frac{2N(\lambda)q^2}{2^\lambda} \ .$$

Now, we move to Bad_2, and this is where we finally make use of the unpredictability of the source. In particular, we consider a game G_5 (Figure 26), which is equivalent to G_4, but just takes into account the analogue of the event Bad_2 happening through a corresponding flag bad_2: Again, by a simple collision argument, it is not hard to see that

$$\mathsf{Pr}\left[\mathsf{Bad}_2\right] \leq \mathsf{Pr}\left[\mathsf{G}_5(\lambda) \text{ sets } \mathsf{bad}_2\right] + \frac{q^2}{2^\lambda} \ .$$

Finally, we upper bound $\mathsf{Pr}\left[\mathsf{G}_5(\lambda) \text{ sets } \mathsf{bad}_2\right]$ with $\mathsf{Adv}^{\mathsf{pred}[\mathbf{P}]}_{S,P}(\lambda)$ for a suitable predictor P. The predictor is based on the simple observation that in order for bad_2 to be set to true, D needs to issue a query (σ, x), returning y, such that for some i, either $\mathcal{O}(i, \sigma, x \oplus k_i)$ or $\mathcal{O}(i, \overline{\sigma}, y \oplus k_i)$ was queried by S. Therefore, P, on input 1^λ, n, and L, first generates keys $k_1, \ldots, k_n$, and run $D(1^\lambda, k_1, \ldots, k_n, L)$, answering D's Prim queries with P's own Prim oracle. Also, at the end of the

MAIN $\mathsf{G}_5(\lambda)$:
$\mathsf{done} \leftarrow \mathsf{false}$
$(1^n, t) \leftarrow\!\!\$\ S^{\mathsf{Prim}}(1^\lambda, \varepsilon)$
$k_1, \ldots, k_n \xleftarrow{\$} \{0,1\}^\lambda$
$L \leftarrow\!\!\$\ S^{\mathcal{O},\mathsf{Prim}}(1^\lambda)$
$\mathsf{done} \leftarrow \mathsf{true}$
$b' \leftarrow\!\!\$\ D^{\mathsf{Prim}}(1^\lambda, k_1, \ldots, k_n, L)$
return b'

ORACLE $\mathcal{O}(i, \sigma, x)$:
if $T_P[i, \sigma, x] = \bot$ **then**
  $T_P[i, \sigma, x] \leftarrow y \setminus P_i^\sigma$
  $T_P[i, \overline{\sigma}, y] \leftarrow x$
  $P_i^\sigma \xleftarrow{\cup} \{x\}$; $P_i^{\overline{\sigma}} \xleftarrow{\cup} \{y\}$
return $T_P[i, \sigma, x]$

ORACLE $\mathsf{Prim}(\sigma, x)$:
if $T_\pi[\sigma, x] = \bot$ **then**
  if $\exists i : T_P[i, \sigma, x \oplus k_i] \neq \bot$ **then**
    $\mathsf{bad}_2 \leftarrow \mathsf{true}$
  $y \xleftarrow{\$} \{0,1\}^\lambda \setminus \Pi^\sigma$
  $T_\pi[\sigma, x] \leftarrow y$
  $T_\pi[\overline{\sigma}, y] \leftarrow x$
  $\Pi^\sigma \xleftarrow{\cup} \{x\}$; $\Pi^{\overline{\sigma}} \xleftarrow{\cup} \{y\}$
return $T_\pi[\sigma, x]$

Fig. 26: Game $\mathsf{G}_5(\lambda)$ used in the proof of Theorem 7.

execution, consider the set T of all (x, y)'s such that D queried either $(+, x)$ obtaining y, or $(-, y)$ obtaining x. Then, P concludes by outputting

$$Q' = \{(+, x \oplus k_i), (-, y \oplus k_i) \ : \ i \in [n], (x, y) \in T\} \ .$$

By the above observation, note that $\mathsf{Adv}^{\mathsf{pred}[\mathbf{P}]}_{S,P}(\lambda) = \mathsf{Pr}\left[\mathsf{G}_5(\lambda) \text{ sets } \mathsf{bad}_2\right]$. □