The Game-Playing Technique

M. Bellare * P. Rogaway †

July 12, 2005

(Draft 0.5)

Abstract

In the *game-playing technique*, one writes a pseudocode *game* such that an adversary's advantage in attacking some cryptographic construction is bounded above by the probability that the game sets a flag *bad*. This probability is then upper bounded by making stepwise, syntactical refinements to the pseudocode—a *chain* of games. The approach was first used by Kilian and Rogaway (1996) and has been used repeatedly since, but it has never received a systematic treatment. In this paper we provide one. We develop the foundations for game-playing, formalizing a general framework for doing game-playing proofs and providing general and useful lemmas that justify various kinds of game-refinement steps. We illustrate the use of the game-playing framework to provide simpler and more easily verifiable proofs of some classic results, including the PRP/PRF switching lemma, the security of the CBC MAC, and the chosen-plaintext-attack security of OAEP.

Keywords: Cryptographic analysis techniques, games, provable security.

*Department of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093 USA. E-mail: mihir@cs.ucsd.edu WWW: www.cse.ucsd.edu/users/mihir/

†Department of Computer Science, University of California at Davis, Davis, California, 95616, USA; and Department of Computer Science, Faculty of Science, Chiang Mai University, Chiang Mai 50200, Thailand. E-mail: rogaway@cs.ucdavis.edu WWW: www.cs.ucdavis.edu/~rogaway/

Contents

1 Introduction

This paper is about the game-playing approach for analyzing cryptographic constructions. We develop a theory of game-playing, elevating it from examples to a general and readily usable technique, and we showcase the use of the method with some illustrative applications. Our work supports the thesis that game-playing, done right, is a powerful tool, capable of delivering more complete and easily verifiable proofs of strong results than are obtainable by competing conventional methods.

1.1 The game-playing approach

The first step in our program is to distill from different approaches in the literature a single paradigm to capture what we want to call game playing. Roughly it works like this. Suppose we wish to upper bound the *advantage* of an adversary A in attacking some cryptographic construction. This is a number between 0 and 1 that is computed as the difference between the probabilities that A outputs 1 in two different "worlds."[1] We proceed as follows:

(1) Write some *pseudocode*—a *game*—that captures the behavior of world 1. The game initializes variables, interacts with the adversary, and then runs some more.

(2) Write another piece of pseudocode—a second game—that captures the behavior of world 0. Arrange that games 1 and 0 are *syntactically identical programs* apart from statements that follow the setting of a flag *bad* to true.

(3) Invoke a *fundamental lemma of game playing* to say that, in this setup, the adversary's advantage is upper-bounded by the probability that *bad* gets set (in either game).

(4) Choose one of the two games and slowly transform it, modifying it in ways that increase or leave unchanged the probability that *bad* gets set, or decrease the probability that *bad* gets set by a bounded amount.

(5) In this way you produce a *game chain*, ending at some *terminal* game. Bound the probability that *bad* gets set in the terminal game.

It is central to our approach that games are *code*, not abstract probabilistic functions; the method, as we develop it, centers around making disciplined transformations to code to get a cryptographic bound.

1.2 Foundations of game playing

We begin by giving a general framework for game-playing proofs. A game G is formalized as a tuple of programs, each written in some programming language $\mathcal{L}$. We use an intentionally limited pseudocode in lieu of a formally specified programming language. We could have formally specified the language $\mathcal{L}$, and we expect there to be some advantages gained by doing this, but a pseudocode $\mathcal{L}$ makes an appropriate first step. Programs over $\mathcal{L}$ employ a common set of global, static variables (and no local variables). A game G can be *run* with an adversary A (look ahead to Figure 2), the adversary calling out to the programs that are provided. We define what it means for two games to be identical-until-*bad*-is-set, where *bad* is a boolean variable in the games. This is a syntactical condition. We prove a *fundamental lemma* for game-playing that says that if two games are identical-until-*bad*-is-set then the difference in the probabilities of a given outcome is bounded by the probability that *bad* gets set (in either game). The fundamental lemma is the central tool justifying the game-playing technique.

We go on to give some general lemmas and techniques for analyzing the probability that *bad* gets set. Principle among these is a simple lemma that lets you change anything you want *after* the flag *bad* gets set, and a lemma that justifies, in some cases, a commonly-used technique of "lazy" coin-flipping. We comment that while elements of this framework have been used before, nothing has been done with much care or formality.

[1] Sometimes the advantage might be something else, such as the probability that the adversary forges, but the case we consider is very common.

1.3 Applications

The applications we provide are chosen to illustrate the applicability of games in a wide variety of environments: they range across the standard model and the random oracle model [BR93], and across both symmetric and asymmetric cryptographic primitives.

PRP/PRF SWITCHING LEMMA. We begin with a motivating observation, due to Tadayoshi Kohno, that the standard proof of the *PRP/PRF switching lemma*, as given in [BKR94, HWKS98], contains an error in reasoning about conditional probabilities. (The lemma says that an adversary that asks at most q queries can distinguish with advantage at most $q^2/2^{n+1}$ a random permutation on n-bits from a random function of n-bits to n-bits. It is frequently employed in the analysis of constructions that use blockciphers and model them as PRPs.) We regard this as evidence that reasoning about cryptographic constructions via conditional probabilities can be subtle and error-prone even in the simplest of settings, and motivates the use of games as an alternative. We re-prove the switching lemma with a very simple game-based proof.

CBC MAC. Let $\mathbf{Adv}^{\mathrm{cbc}}_{n,m}(q)$ denote the maximum advantage that an adversary restricted to making at most q oracle queries can obtain in distinguishing between (1) the m-block CBC MAC, keyed by a random permutation on n bits, and (2) a random function from mn-bits to n-bits. A result of [BKR94] says that $\mathbf{Adv}^{\mathrm{cbc}}_{n,m}(q) \le 2m^2q^2/2^n$. The constant of 2 was reduced to 1 in [Ma02]. The proof of [BKR94] was very complex and did not directly capture the intuition behind the security of the scheme. In this paper we use games to give an elementary proof of the $m^2q^2/2^n$ bound, the proof directly capturing, in our view, the underlying intuition.

OAEP. Finally, we give an example of using games in the public-key, random-oracle setting by proving that OAEP [BR94] with any trapdoor permutation is an IND-CPA secure encryption scheme. The original proof [BR94] of this (known) result was hard to follow or verify; the new proof is simpler and clearer, and illustrates the use of games in a computational rather than information-theoretic setting.

1.4 Syntactic games vs. semantic games

Different authors use the term *game playing* to mean different things. We broadly distinguish two "types" of games: syntactic games and semantic ones. This paper is about *syntactic* games. Here games are specified by code, written in some programming language. Either the language is formalized or is obviously formalizable. Cryptographic reasoning proceeds by syntactic manipulation of the code. We reason with the aid of low-level statements about code, like "games P and Q differ only at line 20", or "line 20 of game P is the second part of a compound statement whose first part is *bad* ← `true`.". Syntactic games begin with Kilian and Rogaway [KR96], who used the approach to analyse DESX. The method soon became the favored one of Rogaway, who used it, along with coauthors, in some ten or so subsequent papers [BKR98, BR00, RBB01, BR02, BRS02, R02, HR03, BRW04, HR04, R04].

The alternative, *semantic* games, regards a game as an arbitrary interaction between an adversary and its environment. The environment can be specified in pseudocode or in any other way. The game-playing technique is understood to be the stepwise mutation of the adversary's environment in order to carry out an analysis. A semantic game-playing proof is fundamentally no different from a hybrid argument, the idea for which goes back to Goldwasser and Micali [GM84]. Bellare and Goldwasser provide an early example of semantic games, giving an intricate hybrid argument to prove security of a signature scheme that uses multiple cryptographic primitives [BG89]. In recent years Victor Shoup has used semantic game-playing extensively. He analyzed a public-key encryption scheme with a game chain [Sh00] and (semantic-style) game-playing soon became a favored technique of his [SS00, Sh01a, Sh01b, CS02, CS03a, CS03b, GS04]. Nowadays many further authors develop their proofs in terms of game chains; see [BBKN01, BCP03, BK05, Bo01, DFKY03, FOPS04, GMMV03, KD04, PP04] as a sample.

Clearly syntactic games are a special case of semantic ones, and, conversely, it seems likely that any semantic game-playing proof can be recast as a syntactic one, so perhaps the difference between syntactic and semantic games might seen inconsequential, or a matter of taste. It is a thesis of this paper that the difference between syntactic and semantic games is a significant one, and a choice that will have a long-term impact on our field. We advocate the syntactic approach because we believe that this more structured technique will engender a richer and more rigorous theory. Henceforth in this paper, *game playing* refers strictly to syntactic games.

1.5 Additional related work

With motivation similar to our own, Maurer develops a framework for the analysis of cryptographic constructions and applies it to the CBC MAC and other examples [Ma02]. Vaudenay has likewise developed a general framework for the analysis of blockciphers and blockcipher-based constructions, and has applied it to the encrypted CBC MAC [Va01]. Neither Maurer's nor Vaudenay's approach are widely employed, and neither is geared towards making stepwise, code-directed refinements for computing a probability. Vaudenay's work, and that of others [PR00, Va01, BR00], includes analyses of variants of the basic CBC MAC.

A more limited and less formal version of our fundamental lemma (Lemma 5) appears in [BKR98, Lemma 7.1]. A lemma by Shoup [Sh01a, Lemma 1] functions in a similar way for semantic games.

Shoup has independently and contemporaneously prepared a manuscript on the game-playing technique [Sh04]. It is more pedagogically-oriented than this paper. Shoup makes no attempt to develop a theory for game playing beyond [Sh01a, Lemma 1] that we mentioned above and reappears here. As with us, one of Shoup's examples is the PRP/PRF switching lemma.

In response to a web distribution of this paper, Bernstein offers his own proof for the CBC MAC [Be05], re-obtaining the conventional bound. Bernstein sees no reason for games, and offers his own explanation for why cryptographic proofs are often complex and hard to verify: author incompetence with probability.

Subsequent to the first posting of this paper, Bellare, Pietrzak and Rogaway [BPR05] improved the bound $\mathbf{Adv}^{\mathrm{cbc}}_{n,m}(q) \le m^2q^2/2^n$ of [BKR94, Ma02] to $\mathbf{Adv}^{\mathrm{cbc}}_{n,m}(q) \le 20mq^2/2^n$ for $m \le 2^{n/3}$. More generally, they show that this bound holds when the messages are at most m blocks and no message is a blockwise prefix of any other. This improves on the result of Petrank and Rackoff [PR00], who had earlier observed that a bound of a constant times $m^2q^2/2^n$ held also in this setting. The proof of [BPR05] relies on the games approach of this paper, refining the game used here to reprove the $\mathbf{Adv}^{\mathrm{cbc}}_{n,m}(q) \le m^2q^2/2^n$ bound and then applying and extending some techniques from [DGHKR04].

1.6 Discussion and outline

Why games? We advocate the game-playing paradigm for several reasons. First, we believe that the approach can lead to more easily verified, less error-prone proofs than those grounded in more conventional probabilistic language. In our opinion, many proofs in cryptography are essentially unverifiable, and this situation has been getting worse and worse. Our field may be approaching a crisis of rigor, where well-known results can be unverified and unverifiable. Second, we believe that game-playing is very widely applicable. Games can be used in the standard model, the random-oracle model, the ideal-blockcipher model, and more; they can be used symmetric settings, public-key settings, and further trust models; they can be used for simple schemes (eg, justifying the Carter-Wegman MAC) and complex protocols (eg, proving the correctness of a key-distribution protocol). Third, game-playing is easily applied and quickly mastered: one needn't spend weeks to learn some supporting theory. Indeed the second author has been using game-playing in his graduate crypto class for years, increasingly employing it to provide a unifying structure for proofs. Students do well at following game-based proofs, perhaps because the incremental character of constructing a game chain meshes well with the mechanics of a blackboard talk, and perhaps too because the approach is relatively "forgiving" if a student misses some particular step. Finally the game-playing technique can lead to significant new results that would seem to be hard to get to using any other technique.

Why should this work? It is fair to ask if anything is actually "going on" when using games—couldn't you recast everything into more conventional mathematical language and drop all that ugly pseudocode? Our experience is that it does not work to do so. The kind of probabilistic statements and thought encouraged by the game-playing paradigm seems to be a better fit, for many cryptographic problems, than that which is encouraged by (just) defining random-variables, writing conventional probability expressions, conditioning, and the like. The power of the approach ultimately stems from the fact that pseudocode is the most precise and easy-to-understand language we know for describing the sort of probabilistic, reactive environments encountered in cryptography, and by remaining in that domain to do ones reasoning you are better able to see what is happening, manipulate what is happening, and validate the changes. In short, form matters.

Challenges. The extent to which games deliver easily verifiable proofs depends on the way they are used. One should make small, easily-checked adjustments as one moves from one game to the next; longer game chains with small changes between adjacent games are easier to verify than short chains with big jumps between adjacent games. This can be tedious and lead to lengthy proofs. To be fully rigorous, each adjustment to a game should be justified by a formally proven rule—the sort of rule that an optimizing compiler might employ to justify reusing a register or doing some code motion. There is not yet a rich enough theory to support all of the modifications to the code that you might want to make in a game. We believe that this will get better in time; this paper is one step.

2 The PRP/PRF Switching Lemma

Let Perm(n) be the set of all permutations on $\{0,1\}^n$. Let Rand(n) be the set of all functions from $\{0,1\}^n$ to $\{0,1\}^n$. By $A^f \Rightarrow 1$ we refer to the event that adversary A, equipped with an oracle f, outputs the bit 1. In what follows, assume that π is randomly sampled from Perm(n) and ρ is randomly sampled from Rand(n).

Lemma 1 [PRP/PRF Switching Lemma] *Let $n \geq 1$ be an integer. Let A be an adversary that asks at most q oracle queries. Then* $|\Pr[A^\pi \Rightarrow 1] - \Pr[A^\rho \Rightarrow 1]| \leq q(q-1)/2^{n+1}$. ▮

The result is folklore, and is used extensively. Its value is the following. In analyzing a blockcipher-based construction C we need to bound how well an adversary A can do in breaking $\mathsf{C}[\pi]$, for a random permutation π on n bits. But it is often technically easier to upper bound how well the adversary can do in attacking $\mathsf{C}[\rho]$, for a random function ρ from n bits to n bits. Doing this suffices because we can then apply the Switching Lemma to conclude that the difference is small.

In this section we point to some subtleties in the "standard" proof, as given for example in [HWKS98, BKR94], of this apparently simple result, showing that one of the claims made in these proofs is incorrect. We then show how to prove the lemma using games. This example provides a gentle introduction to the game-playing technique and a warning about perils of following ones intuition when dealing with conditional probability in provable-security cryptography.

The standard analysis proceeds as follows. Let Coll ("collision") be the event that an adversary, interacting with an oracle ρ, asks distinct queries X and X' that return the same answer. Let Dist ("distinct") be the complementary event. Now

$$\Pr[A^\pi \Rightarrow 1] \;=\; \Pr[A^\rho \Rightarrow 1 \mid \mathsf{Dist}] \qquad (1)$$

since a random permutation is indistinguishable from a random function in which one observes no collisions. Letting x be this common value and $y = \Pr[A^\rho \Rightarrow 1 \mid \mathsf{Coll}]$ we have

$$\begin{aligned} |\Pr[A^\pi \Rightarrow 1] - \Pr[A^\rho \Rightarrow 1]| \;&=\; |x - x\Pr[\mathsf{Dist}] - y\Pr[\mathsf{Coll}]| \;=\; |x(1-\Pr[\mathsf{Dist}]) - y\Pr[\mathsf{Coll}]| \\ &=\; |x\Pr[\mathsf{Coll}] - y\Pr[\mathsf{Coll}]| \;=\; |(x-y)\Pr[\mathsf{Coll}]| \;\leq\; \Pr[\mathsf{Coll}] \end{aligned}$$

where the final inequality follows because $x, y \in [0,1]$. One next argues that $\Pr[\mathsf{Coll}] \leq q(q-1)/2^{n+1}$ and so the Switching Lemma follows.

Where is the error in the simple proof above? It's at (1); it needn't be the case that $\Pr[A^\pi \Rightarrow 1] = \Pr[A^\rho \Rightarrow 1 \mid \mathsf{Dist}]$, and the sentence we gave by way of justification was mathematically meaningless. Here is a simple example to demonstrate that $\Pr[A^\pi \Rightarrow 1]$ can be different from $\Pr[A^\rho \Rightarrow 1 \mid \mathsf{Dist}]$. Let $n = 2$, name the four points of $\{0,1\}^2$ as 0, 1, 2, and 3, and consider the following adversary A with oracle f:

if $f(0) = 0$ **then return** 1 **else if** $f(1) = 1$ **then return** 1 **else return** 0.

Then $\Pr[A^\pi \Rightarrow 1] = 5/12 \approx 0.42$ because there are 12 possibilities for $\pi(0)\pi(1)$ and A returns 1 for five of them: $01, 02, 03, 21, 31$. On the other hand, $\Pr[A^\rho \Rightarrow 1 \mid \mathsf{Dist}] = \Pr[A^\rho \Rightarrow 1 \wedge \mathsf{Dist}] / \Pr[\mathsf{Dist}] = (6/16)/(13/16) = 6/13 \approx 0.46$ because there are 16 possible values $\rho(0)\rho(1)$ and $A^\rho \Rightarrow 1 \wedge \mathsf{Dist}$ is true for six of them, 00, 01, 02, 03, 21, 31, while Dist is true for 13 of them: 00, 01, 02, 03, 10, 12, 13, 20, 21, 23, 30, 31, 32.

Notice that the number of oracle queries made by the adversary of our counterexample varies, being either one or two, depending on the reply it receives to its first query. As we show in Appendix A (this was also pointed out by Kohno), if A always makes exactly q oracle queries (regardless of A's coins and the answers returned to its queries) then (1) is true. Since one can always first modify A to make exactly q queries, we

Initialize	**On query** $f(X)$ Game S1
100 $bad \leftarrow$ false	110 $Y \xleftarrow{\$} \{0,1\}^n$
101 **for** $X \in \{0,1\}^n$ **do** $\pi(X) \leftarrow$ undefined	111 **if** $Y \in \mathbf{image}(\pi)$ **then** $bad \leftarrow$ true , $Y \xleftarrow{\$} \overline{\mathbf{image}}(\pi)$
	112 **return** $\pi(X) \leftarrow Y$ omit ↑ for Game S0

Figure 1: Games used in the proof of the Switching Lemma.

would be loth to say that the proofs in [HWKS98, BKR94] are incorrect, but the authors make claim (1), and view it as "obvious," without restricting the adversary to exactly q queries, masking a subtlety that is not apparent at a first (or even second) glance.

The fact that one can write something like (1) and people assume this to be correct, and even obvious, suggests to us that the language of conditional probability may often be unsuitable for thinking about and dealing with the kind of probabilistic scenarios that arise in cryptography. Games may more directly capture the desired intuition. Let's use them to give a correct proof. Assume without loss of generality that A never asks an oracle query twice.

We imagine answering A's queries by running one of two games. Instead of thinking of A interacting with a random permutation oracle $\pi \xleftarrow{\$} \mathrm{Perm}(n)$ think of A interacting with the Game S1 shown in Figure 1. Instead of thinking of A interacting with a random function oracle $\rho \xleftarrow{\$} \mathrm{Rand}(n)$ think of A interacting with the game S0 shown in the same figure. Game S0 is game S1 without the shaded statement.

In both games S1 and S0 we start off performing the initialization step, setting a flag *bad* to false and setting a variable π to be undefined at every n-bit string. (We will soon establish conventions that eliminate the need to write these steps.) As the game runs, we fill-in values of $\pi(X)$ with n-bit strings. At any point in time, we let $\mathbf{image}(\pi)$ be the set of all n-bit strings Y such that $\pi(X) = Y$ for some X. Let $\overline{\mathbf{image}}(\pi)$ be the complements of this set relative to $\{0,1\}^n$.

Notice that the adversary never sees the flag *bad*. The flag will play a central part in our analysis, but it is not something that the adversary can observe. It's only there for our bookkeeping. What *does* adversary A see as it plays game S0? Whatever query X it asks, the game returns a random n-bit string Y. So game S0 perfectly simulates a random function $\rho \xleftarrow{\$} \mathrm{Rand}(n)$ (remember that the adversary isn't allowed to repeat a query) and $\Pr[A^\rho \Rightarrow 1] = \Pr[A^{\mathrm{S0}} \Rightarrow 1]$. Similarly, if we're in game S1, then what the adversary gets in response to each query X is a random point Y that has not already been returned to A. The behavior of a random permutation oracle is exactly this, too. (This is guaranteed by what we will call the "principle of lazy sampling.") So $\Pr[A^\pi \Rightarrow 1] = \Pr[A^{\mathrm{S1}} \Rightarrow 1]$. At this point we have that $|\Pr[A^\pi \Rightarrow 1] - \Pr[A^\rho \Rightarrow 1]| = |\Pr[A^{\mathrm{S1}} \Rightarrow 1] - \Pr[A^{\mathrm{S0}} \Rightarrow 1]|$. We next claim that $|\Pr[A^{\mathrm{S1}} \Rightarrow 1] - \Pr[A^{\mathrm{S0}} \Rightarrow 1]| \le \Pr[A^{\mathrm{S0}} \text{ sets } bad\,]$. We refer to the lemma that makes this step possible as the *fundamental lemma of game playing*. The lemma says that whenever two games are written so as to be syntactically identical except for things that immediately follow the setting of *bad*, the difference in the probabilities that A outputs 1 in the two games is bounded by the probability that *bad* is set in either game. (It actually says something a bit more general, as we will see.) So we have left only to bound $\Pr[A^{\mathrm{S0}} \text{ sets } bad\,]$. By the union bound, the probability that a Y will ever be in $\mathbf{image}(\pi)$ at line 111 is at most $(1 + 2 + \cdots + (q-1))/2^n = q(q-1)/2^{n+1}$. This completes the proof.

3 The Game-Playing Framework

3.1 Game syntax

A *program* P is a finite, valid sequence of *statements* written in some *programming language*, $\mathcal{L}$. We identify a program with its *parse tree*. Programs take zero or more strings as input and produce zero or more strings as output. We only consider programs that always terminate. We will not formally specify the programming language $\mathcal{L}$; our language will be pseudocode and we will keep it simple enough that there won't be any ambiguity about how to run a program. Certainly one could rigorously define the programming language that one wanted to use for specifying games, and one could then endow it with a proper execution semantics. We won't do so just now, but we do need to explain some basic characteristics and conventions for our pseudocode.

We include the usual repertoire of constructs one finds in a procedural programming language: variables, assignment statements, **if**-statements, **for**-statements, and so forth. We also include a sample-then-assign operator $\xleftarrow{\$}$ where $X \xleftarrow{\$} \mathcal{X}$ means to select a random element from the finite set $\mathcal{X}$ (all elements equally

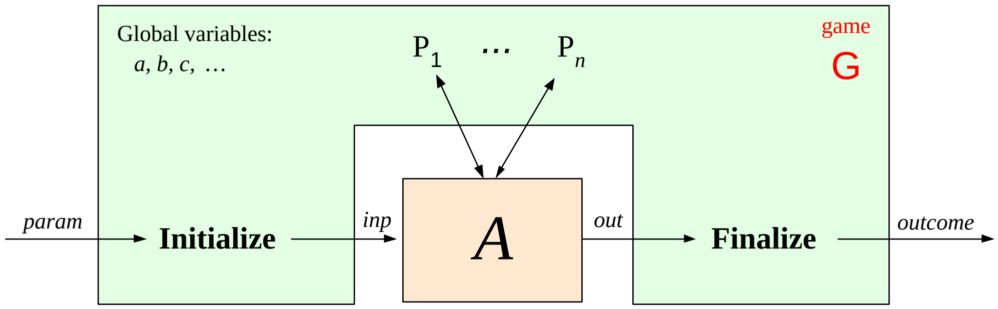

Figure 2: Running a game G with an adversary A. The game—the box that surrounds A—consists of pseudocode procedures **Initialize**, $\mathrm{P}_1, \ldots, \mathrm{P}_n$, and **Finalize**. The adversary A receives an (optional) input from the game, interacts with its oracles $\mathrm{P}_1, \ldots, \mathrm{P}_n$, and produces an output. The outcome of the game is determined by **Finalize**.

---

probable) and assign the resulting value to the variable X. This is the only source of randomness in programs, so probabilities are taken over the choices associated to sample-then-assign statements. Variables in programs are understood to be static and global: their values "hang around" from call to call and have a scope of all programs in an associated *game*, which we will define shortly. We'll assume a relatively rich set of types: booleans, integers, strings, arrays (including arrays indexed by strings), finite sets, and partial functions from finite sets to finite sets. We won't explicitly declare variables, but each variable will have a fixed type, that type being clear from the context. We'll use a comma or newline as a statement separator, and S, S' is a statement when S and S′ are. The empty statement ε is also a statement, and we regard S and S, ε as the same. We use indentation to indicate grouping. Boolean variables are automatically initialized to false and other variables are initially everywhere undefined (an array is undefined for all possible indices and a function is undefined at all domain points).

Definition 2 **[Games]** *A* ***game*** $\mathrm{G} = (\textbf{Initialize}, \mathrm{P}_1, \mathrm{P}_2, \ldots, \mathrm{P}_n, \textbf{Finalize})$ *is a sequence of programs.* ▌

Programs $\mathrm{P}_1, \ldots, \mathrm{P}_n$ are the *oracles* of the game. If we omit specifying **Initialize** or **Finalize** it means that the program does nothing: it computes the identity function. We let *param* denote the input to **Initialize** and we let *inp* denote its output. We let *out* be the input to **Finalize** and we let *outcome* be its output. If we describe a game by giving a single unlabeled program, that program is the **Finalize** program. For all of our games, the **Initialize** and **Finalize** programs will have those names, but we will choose suggestive names for $\mathrm{P}_1, \ldots, \mathrm{P}_n$. To see examples of games, look ahead to any of the games appearing later in this paper, which we name as in C4 or S1.

3.2 Running a game

To run a game G we need an adversary A to interact with it. See Figure 2. An *adversary* is a probabilistic algorithm equipped with the ability to query some number $n \geq 0$ of *oracles*. For convenience, we assume that an adversary is described by a program—in particular, its source of randomness is sample-then-assign statements $X \xleftarrow{\$} \mathcal{X}$ where the adversary has constructed the finite set $\mathcal{X}$ using the constructs of the programming language.[2] The pair consisting of a game G and an adversary A is called a *runnable game*. We will refer to a runnable game between G and A by writing either G_A or A^{G}. We'll use the first notation if we want to emphasize what the game is doing, and we'll use the second notation if we want to emphasize what the adversary is doing.

To run $\mathrm{G} = (\textbf{Initialize}, \mathrm{P}_1, \mathrm{P}_2, \cdots, \mathrm{P}_n, \textbf{Finalize})$ with A and string parameter *param*, begin by calling program **Initialize** with input *param*. (In the asymptotic setting, this might be a security parameter k. For all of our non-asymptotic examples *param* is empty.) We now run A, passing it any (string) return value *inp* produced by **Initialize**. When adversary A calls its i^{th} oracle with a given string, we pass that string to program P_i and run it. We return to A whatever string the program P_i says to return. We assume that an adversary eventually terminates, regardless of what it receives from its environment. (That is, adversary A should terminate even if we were to run it in some other, arbitrary game.) When A halts, possibly with

[2] This definition excludes the possibility of an adversary being able to to flip a coin with bias $p = 1/\pi$, for example. It is possible to show that an optimal adversary for a game G need not flip coins with irrational biases; in that sense, assuming an adversary's source of randomness to be to be sample-then-assign statements is without loss of generality.

some output *out*, we call **Finalize**, providing it any output produced by A. The *outcome* of the game is the string value returned by **Finalize**. The outcome of a game can be regarded as a random variable, the randomness taken over the sample-then-assign statements of the adversary A and the game G. Often the outcome of the game is the return value of A, procedure **Finalize** not doing anything beyond passing on its input as its output.

We write $\Pr[\mathrm{G}_A \Rightarrow 1]$ for the probability that the outcome of game G is 1 when we run G_A. We say that games G and H are *equivalent* if for any adversary A it is the case that $\Pr[\mathrm{G}_A \Rightarrow 1] = \Pr[\mathrm{H}_A \Rightarrow 1]$.

We write $\Pr[A^{\mathrm{G}} \Rightarrow 1]$ to refer to the probability that the adversary A outputs 1 when we run G_A. The *advantage* of A in distinguishing games G and H is the real number $\mathbf{Adv}^{\mathrm{dist}}_{\mathrm{G,H}}(A) = \Pr[A^{\mathrm{G}} \Rightarrow 1] - \Pr[A^{\mathrm{H}} \Rightarrow 1]$. We say that games G and H are (perfectly) *adversarially indistinguishable* if for any adversary A it is the case that $\Pr[A^{\mathrm{G}} \Rightarrow 1] = \Pr[A^{\mathrm{H}} \Rightarrow 1]$.

3.3 Identical-until-bad-is-set games

A boolean variable *bad* in a game G is called a *flag* if starts off as false and changes values at most once: once a flag becomes true, it can never revert to false. We are interested in programs that are syntactically identical until a flag *bad* has been set to true. The formal definition is as follows.

Definition 3 [Identical-until-*bad*-is-set] *Let* P *and* Q *be programs and let* bad *be a flag in each of them. Then* P *and* Q *are* ***identical-until-bad-is-set*** *if their parse trees are the same except for the following: wherever program P has a statement* $bad \leftarrow \mathsf{true},\ \mathrm{S}$ *in its parse tree, program Q has at the corresponding position of its parse tree that statement* $bad \leftarrow \mathsf{true},\ \mathrm{T}$ *for a* T *that is possibly different from* S. *Games* $\mathrm{G} = (\mathbf{Initialize}, \mathrm{P}_1, \ldots, \mathrm{P}_n, \mathbf{Finalize})$ *and* $\mathrm{H} = (\mathbf{Initialize}', \mathrm{Q}_1, \ldots, \mathrm{Q}_n, \mathbf{Finalize}')$ *are* ***identical-until-bad-is-set*** *if each of their corresponding programs are identical-until-bad-is-set.* ▮

As an example, games S0 and S1 from Figure 1 are identical-until-*bad*-is-set. For one of these games, S0, we have the empty statement following $bad \leftarrow \mathsf{true}$ in the parse tree of S0; for S1, we have the statement $Y \xleftarrow{\$} \overline{\mathbf{image}}(\pi)$. Since this is the only difference in the programs, the games are identical-until-*bad*-is-set.

We'll also say that G and H are are identical-until-*bad*-is-set if one game has the statement **if** *bad* **then** S where the other has the empty statement ε. One can consider **if** *bad* **then** S to be the same as **if** *bad* **then** $bad \leftarrow \mathsf{true},\ \mathrm{S}$ and one can consider the empty statement ε to be the same as **if** *bad* **then** $bad \leftarrow \mathsf{true},\ \varepsilon$ and under this convention the games are identical-until-*bad*-is-set under the given definition.

We write $\Pr[\mathrm{G}_A \text{ sets } bad\,]$ to refer to the probability that the flag *bad* is true at the end of the execution of the runnable game G_A, when **Finalize** terminates. The following is easy to see:

Proposition 4 *Identical-until-bad-is-set is an equivalence relation on games.* ▮

3.4 The fundamental lemma

The lemma that justifies the game-playing technique is the following.

Lemma 5 [Fundamental lemma of game-playing] *Let* G *and* H *be identical-until-bad-is-set games, and let A be an adversary. Then*

$$\Pr[\mathrm{G}_A \Rightarrow 1] - \Pr[\mathrm{H}_A \Rightarrow 1] \;\leq\; \Pr[\mathrm{G}_A \textit{ sets } bad\,]\ .$$

More generally, $|\Pr[\mathrm{G}_A \Rightarrow 1] - \Pr[\mathrm{H}_A \Rightarrow 1]| \leq \Pr[\mathrm{I}_A \textit{ sets } bad\,]|$ *for any identical-until-bad-is-set games* G, H, I. ▮

Proof of Lemma 5: Ignore for now the second statement in the lemma; it will follow immediately from the first statement by using Proposition 6.

We have assumed that the adversary and all programs comprising a game always terminate, and so there exists a smallest number b such that A and G_A and G_B perform no more than b sample-then-assign statements, each of these sample-then-assign statements sampling from a set of size at most b. Let $C = \mathrm{Coins}(A, \mathrm{G}, \mathrm{H}) = [1 \mathinner{..} b!]^b$ be the set of b-tuples of numbers, each number between 1 and $b!$. We call C the *coins* for $(A, \mathrm{G}, \mathrm{H})$. A random execution of G_A can be determined in the following way. First, draw a random sample $c = (c_1, \ldots, c_b)$ from C. Then, using c, deterministically execute G_A as follows: On the i^{th} sample-then-assign statement, $X_i \xleftarrow{\$} \{X_0, \ldots, X_{n_i-1}\}$, let X_i be $X_{c_i \bmod n_i}$ This way to perform sample-then-assign statements is done regardless of whether A is the one performing the sample-then-assign statement or one of the programs from G is performing the statement. Now notice that n_i divides $b!$ and so the mechanism above will return a uniform point X_i from $\{X_0, \ldots, X_{n_i-1}\}$. The return values for each sample-then-assign statement are independent, so we have properly simulated G_A using the random point from C and no other source of randomness. Similarly, starting from a random point $(c_1, \ldots, c_b)$ from C we can run H_A without any further coins by performing the i^{th} sample-then-assign statement $X_i \xleftarrow{\$} \{X_0, \ldots, X_{n_i-1}\}$ statement as before. From now on in the proof, assume that we realize G_A and H_A as we have described, by sampling $(c_1, \ldots, c_b)$ from the coins C for $(A, \mathrm{G}, \mathrm{H})$. We let $\mathrm{G}_A(c)$ and $\mathrm{H}_A(c)$ denote the run of G and H, respectively, with A and the indicated coins $c \in C$.

Let $CG_{\text{one}} = \{c \in C\colon \mathrm{G}_A(c) \Rightarrow 1\}$ be the coins that cause G_A to output 1, and similarly define CH_{one} for H_A. Partition CG_{one} into $CG^{\text{bad}}_{\text{one}}$ and $CG^{\text{good}}_{\text{one}}$ according to whether *bad* is set to `true` in the run, and similarly define $CH^{\text{bad}}_{\text{one}}$ and $CH^{\text{good}}_{\text{one}}$. Define $CG^{\text{bad}} = \{c \in C\colon \mathrm{G}_A(c) \text{ sets } \mathit{bad}\}$. Observe that because games H and G are identical-until-*bad*-is set games, an element $c \in C$ is in $CG^{\text{good}}_{\text{one}}$ iff it is in $CH^{\text{good}}_{\text{one}}$, so $|CG^{\text{good}}_{\text{one}}| = |CH^{\text{good}}_{\text{one}}|$. Thus

$$\begin{aligned}
\Pr[\mathrm{G}_A \Rightarrow 1] - \Pr[\mathrm{H}_A \Rightarrow 1] &= \frac{|CG_{\text{one}}| - |CH_{\text{one}}|}{|C|} = \frac{|CG^{\text{bad}}_{\text{one}}| + |CG^{\text{good}}_{\text{one}}| - |CH^{\text{good}}_{\text{one}}| - |CH^{\text{bad}}_{\text{one}}|}{|C|} \\
&= \frac{|CG^{\text{bad}}_{\text{one}}| - |CH^{\text{bad}}_{\text{one}}|}{|C|} \le \frac{|CG^{\text{bad}}_{\text{one}}|}{|C|} \le \frac{|CG^{\text{bad}}|}{|C|} = \Pr[\mathrm{G}_A \text{ sets } \mathit{bad}\,] \,.
\end{aligned}$$

The final claim in the lemma, that $|\Pr[A^{\mathrm{G}} \Rightarrow 1] - \Pr[A^{\mathrm{H}} \Rightarrow 1] \le \Pr[A^{\mathrm{I}} \text{ sets } \mathit{bad}\,]|$ when G, H, and I are identical-until-*bad*-is-set, follows directly from Lemma 6 (to be given later). That lemma ensures that $\Pr[\mathrm{G}_A \text{ sets } \mathit{bad}\,] = \Pr[\mathrm{H}_A \text{ sets } \mathit{bad}\,] = \Pr[\mathrm{I}_A \text{ sets } \mathit{bad}\,]$ and so $\Pr[A^{\mathrm{G}} \Rightarrow 1] - \Pr[A^{\mathrm{H}} \Rightarrow 1] \le \Pr[A^{\mathrm{I}} \text{ sets } \mathit{bad}\,]$ and, by symmetry, $\Pr[A^{\mathrm{H}} \Rightarrow 1] - \Pr[A^{\mathrm{G}} \Rightarrow 1] \le \Pr[A^{\mathrm{I}} \text{ sets } \mathit{bad}\,]$. This completes the proof. ▮

TERMINOLOGY. The power of the game-playing technique stems, in large part, from our ability to incrementally rewrite games, constructing chains of games that are at the center of a game-playing proof. Using the fundamental lemma, you first arrange that the analysis you want to carry out amounts to bounding $\epsilon = \Pr[\mathrm{G1}_A \text{ sets } \mathit{bad}\,]$ for some first game G1 and some adversary A.[3] You want to bound ϵ as a function of the resources expended by A. To this end, you modify the game G1, one step at a time, constructing a chain of games $\mathrm{G1} \longrightarrow \mathrm{G2} \longrightarrow \mathrm{G3} \longrightarrow \cdots \longrightarrow \mathrm{G}n$. Game G1 is the *initial* game and game Gn is the *terminal* game. Game G1 is played against A; other games may be played against other adversaries (though they usually are not). Consider a transition $\mathrm{G}_A \to \mathrm{H}_B$. Let $p_G = \Pr[\mathrm{G}_A \text{ sets } \mathit{bad}\,]$ and let $p_H = \Pr[\mathrm{H}_B \text{ sets } \mathit{bad}\,]$. We want to bound p_G in terms of p_H. **(1)** Sometimes we show that $p_G \le p_H$. In this case, the transformation is said to be *safe*. A special case of this is when $p_G = p_H$, in which case the transformation is said to be *conservative*. **(2)** Sometimes we show that $p_G \le p_H + \epsilon$ or $p_G \le c \cdot p_H$ for some particular $\epsilon > 0$ or $c > 1$. Either way, we call the transformation *lossy*. For an additive lossy transformation, ϵ is the *loss term*; for a multiplicative lossy transformation, c is the *dilation* term. When a chain of safe and additively lossy transformations is performed, a bound for *bad* getting set in the initial game is obtained by adding up all the loss terms and the bound for *bad* getting set in the terminal game. If there are multiplicative losses then we bound *bad* getting set in the initial game in the natural way. We use the words *conservative*, *safe*, and *lossy* to apply to pairs of games even in the absence of an adversary: the statement is then understood to apply to all adversaries, or to all adversaries with understood resources. For example, the transformation $G \to H$ is conservative if for all adversaries A we have that $\Pr[G_A \text{ sets } \mathit{bad}\,] = \Pr[H_A \text{ sets } \mathit{bad}\,]$.

4 Game-Rewriting Techniques

In this section we name, describe, and justify some game-transformation techniques that seem universally useful. Our enumeration is not comprehensive, only aiming to hit some of the most interesting or widely applicable techniques. We suggest that a reader might want to skip Sections 4.2–4.3 on a first reading.

[3] In fact, a game chain may be used also for this first phase, before we apply the fundamental lemma; an example is given in our OAEP analysis.

4.1 After bad is set, nothing matters

One of the most common manipulations of games is to modify what happens after *bad* gets set to true. Quite often the modification consists of dropping some code, but it is also fine to insert alternative code. Any modification following the setting of *bad* is conservative. The formal result is as follows.

Proposition 6 [After *bad* is set, nothing matters] *Let* G *and* H *be identical-until-bad-is-set games. Let A be an adversary. Then* $\Pr[\mathrm{G}_A$ *sets bad* $] = \Pr[\mathrm{H}_A$ *sets bad* $]$.

Proof of Proposition 6: Using the definition from the proof of Lemma 5, fix coins $C = \mathrm{Coins}(A, \mathrm{G}, \mathrm{H})$ and execute G_A and H_A in the manner we described using these coins. Let $CG^{\mathrm{bad}} \subseteq C$ be the coins that result in *bad* getting set to true when we run G_A, and let $CH^{\mathrm{bad}} \subseteq C$ be the coins that result in *bad* getting set to true when we run H_A. Since G and H are identical-until-*bad*-is-set, each $c \in C$ causes *bad* to be set to true in G_A iff it causes *bad* to be set to true in H_A. Thus $CG^{\mathrm{bad}} = CH^{\mathrm{bad}}$ and hence $|CG^{\mathrm{bad}}| = |CH^{\mathrm{bad}}|$ and $|CG^{\mathrm{bad}}|/|C| = CH^{\mathrm{bad}}|/|C|$, which is to say that $\Pr[\mathrm{G}_A$ sets *bad* $] = \Pr[\mathrm{H}_A$ sets *bad* $]$. ▮

4.2 Coin fixing

Consider a game G with an oracle P. The adversary A hopes, running with G, to set *bad*. It adaptively asks P strings $X_1, \ldots, X_q$ getting back strings $Y_1, \ldots, Y_q$. We would like to change G to a different game H in which $X_1, \ldots, X_q$, $Y_1, \ldots, Y_q$ are all *fixed*, *constant* strings. We do this—when we can—using the *coin-fixing* technique. It stems from a classical method in complexity theory to eliminate coins [Ad78], hardwiring them in, as in the proof that BPP $\subseteq$ P/poly.

One can't always apply the coin-fixing; we now describe a sufficient condition in which one can. We first describe the basic setup. Suppose that the runnable game G_A has the following characteristics. There is a single oracle P There is no input *param* supplied to A and no output *out* received from it. The game contains a flag *bad*. Adversary A asks, in sequence, exactly q string queries to P, which the program stores in write-once variables $X_1, \ldots, X_q$; and the program computes in response write-once string variables $Y_1, \ldots, Y_q$, providing these answers, one-by-one, to A. That there is a single oracle and that X_i and Y_i are in write-once variables are without loss of generality in our current context. Let $\mathcal{C}$ be a set of $(X_1, \ldots, X_q, Y_1, \ldots, Y_q)$ tuples such that every vector of queries $X_1, \ldots, X_q$ and their responses $Y_1, \ldots, Y_q$ that could arise in an execution of G_A occurs in $\mathcal{C}$. We call $\mathcal{C}$ a *query/response set* for G_A. A query/response set does not need to be the smallest set that includes all possible queries and their response, it only has to include it.

Let $\mathcal{Y}$ be the set of all variables $Y \notin \{X_1, \ldots, X_q, Y_1, \ldots, Y_q\}$ in the game G for which some Y_i depends on Y (here we speak of "depends on" in the information-flow sense of programming-language theory). We say that G_A is *oblivious* if the variable *bad* does not depend on any variable in $\mathcal{Y}$.

Informally, a game is oblivious if it doesn't use anything about how the Y_i-values were made in order to compute *bad*: no variable that influenced a Y_i-value (excluding X_i- and Y_i- values) also influences *bad*. A special case of an oblivious games is when the vector $(Y_1, \ldots, Y_q)$ is chosen at random from some finite set $\mathcal{V}$ Note that in an oblivious program the X_i and Y_i values themselves may influence *bad*.

Given an oblivious game G_A, a query/response set $\mathcal{C}$ for G_A, and a point $\mathtt{C} = (\mathtt{X}_1, \ldots, \mathtt{X}_q, \mathtt{Y}_1, \ldots, \mathtt{Y}_q) \in \mathcal{C}$, we form a new game $\mathrm{H}^{\mathtt{C}}$ as follows. Game $\mathrm{H}^{\mathtt{C}}$ is like G except it has no oracle P. Each (R-value) use of an X_i or Y_i in G is replaced by the corresponding constant $\mathtt{X}_i$ or $\mathtt{Y}_i$. Each (R-value) use of a variable $Y \in \mathcal{Y}$ is replaced by an arbitrary constant of the correct type. At the beginning of the **Finalize** program for $\mathrm{H}^{\mathtt{C}}$ a **for**-loop is executed simulating the arrival of the sequence of P-queries $\mathtt{X}_1, \ldots, \mathtt{X}_q$ and doing whatever program P would have done on receipt of each of these queries (apart from the changes we have already mandated). This completes the description of $\mathrm{H}^{\mathtt{C}}$.

Let $\mathrm{H} = \mathsf{CoinFix}_A^{\mathcal{C}}(\mathrm{G})$ be $\mathrm{H}^{\mathtt{C}}$ for the lexicographically first $\mathtt{C} \in \mathcal{C}$ that maximizes $\Pr[\mathrm{H}_A^{\mathtt{C}}$ sets *bad* $]$. Since H_A no longer depends on A, we may omit mention of it and still have a runnable game. We can now state the coin-fixing lemma.

Lemma 7 [Coin-fixing technique] *Let* G_A *be an oblivious game and let* $\mathcal{C}$ *be a query/response set for it. Let* $\mathrm{H} = \mathsf{CoinFix}_A^{\mathcal{C}}(\mathrm{G})$. *Then* $\Pr[\mathrm{G}_A$ *sets bad* $] \leq \Pr[\mathrm{H}$ *sets bad* $]$. ▮

Proof of Lemma 7: Using the technique of Lemma 5, define coin sets for the runnable game G_A as follows: let C_A be coins for running A; let C_Y be coins for sample-then-assign statements to variables Y_i and variables in $\mathcal{Y}$; and let C_B be any further coins used by G. Each of these is a finite set, and all that is required is that by choosing one random point from each of these sets, $c_A \stackrel{\$}{\leftarrow} C_A$, $c_Y \stackrel{\$}{\leftarrow} C_Y$, and $c_B \stackrel{\$}{\leftarrow} C_B$, one can deterministically run G_A, determining a final value for *bad* for this run, which we'll denote $bad(c_A, c_Y, c_B)$. Coins c_B also determine an execution of H, determining, in particular, if *bad* gets set there: call the final value of that variable $bad(c_B)$. Since some number in a set of real numbers must be at least as large as the average, there must exist a $(\mathtt{c_A}, \mathtt{c_Y}) \in C_A \times C_Y$ such that $\Pr_{c_A,c_Y,c_B}[\mathrm{G}_A(c_A, c_Y, c_B) \text{ sets } bad\,] \le \Pr_{c_B}[\mathrm{G}_A(\mathtt{c_A}, \mathtt{c_Y}, c_B) \text{ sets } bad\,]$. Let $\mathtt{C} = (\mathtt{X}'_1, \ldots, \mathtt{X}'_q, \mathtt{Y}'_1, \ldots, \mathtt{Y}'_q)$ be the queries and responses that result from running G_A with coins $\mathtt{c_A}, \mathtt{c_Y}$. Our notion of obliviousness ensures that $\Pr_{c_B}[\mathrm{G}_A(\mathtt{c_A}, \mathtt{c_Y}, c_B) \text{ sets } bad\,] = \Pr_{c_B}[\mathrm{H}^{\mathtt{C}}(c_B) \text{ sets } bad\,]$, with notation as in the paragraph preceding the lemma. This is because coins $\mathtt{C}$ result in oracle queries $\mathtt{X}'_1, \ldots, \mathtt{X}'_q$, responses $\mathtt{Y}'_1, \ldots, \mathtt{Y}'_q$), and unspecified additional values to variables, and the execution of $\mathrm{H}^{\mathtt{C}}$ proceeds identically apart for "incorrect" values for variables in $\mathcal{Y}$ and the variables these impact, but, by definition of obliviousness, these incorrect values are not relevant when it comes to determining whether or not *bad* gets set. Now $\Pr_{c_B}[\mathrm{H}^{\mathtt{C}}(c_B) \text{ sets } bad\,] \le \Pr[\mathrm{H} \text{ sets } bad\,]$ because $\mathtt{C} \in \mathcal{C}$ must be in the query/response set by our definition of it. This completes the proof. ▮

Coin-fixing is our primary method for eliminating adversarial adaptivity. Many times in analyzing a game, adaptivity is at the center of the analytic difficulty. It is worth pointing out that in using coin-fixing to banish adaptivity, one never establishes that the best non-adaptive adversary for the original game—or any other game—does no better than the best adaptive one. This may be false (or at least not ostensibly true) even though the coin-fixing technique can be used to expunge adaptivity in the analysis.

4.3 Lazy sampling

Instead of making random choices up front, it is often convenient rewrite a game so as to delay making random choices until they are actually needed. We call such "just-in-time" flipping of coins *lazy sampling*.

As a simple but frequently used example—let's call it example 1—consider a game that surfaces to the adversary a random permutation π on n bits. One way to realize this game is to choose π at random from $\mathrm{Perm}(n)$ during **Initialize** and then, when asked a query $X \in \{0,1\}^n$, answer $\pi(X)$. The alternative, lazy, method for implementing π would start with a partial permutation π from n bits to n bits that is everywhere undefined. When asked a query X not yet in the domain of π, the oracle would choose a value Y randomly from the co-range of π, define $\pi(X) \leftarrow Y$, and return Y.

You can think of the current partial function π as imposing the "constraint" that $\pi(X) \notin \mathbf{image}(\pi)$ on our choice of $\pi(X)$. We choose $\pi(X)$ at random from all points respecting the constraint.

For example 1, it seems obvious that the two ways to simulate a random permutation are equivalent. (Recall that *equivalent* is a technical term we have defined: it means that no adversary can distinguish, with any advantage, which of the two games it is playing.) But lazy sampling methods can get more complex and prospective methods for lazy sampling often fail to work. One needs to carefully verify any prospective use of lazy sampling. To see this, consider the following example 2. The game provides the adversary with permutations $\pi_1, \pi_2 \colon \{1,2,3\} \to \{1,2,3\}$ subject to the constraint that $\pi_1(x) \neq \pi_2(x)$ for all $x \in \{1,2,3\}$. The eager way to simulate the pair of oracles is to choose π_1, π_2 uniformly at random from the set of pairs of permutations that obey the constraint. A possible lazy way, where we answer an oracle query with a random point not violating any constraint on already-defined points, may proceed like this. On query $\pi_1(1)$ we would return a random point in $\{1,2,3\}$. Say this is 1. On query $\pi_1(2)$ we would return a random point in $\{2,3\}$, say 2. On query $\pi_1(3)$ we would be forced to return 3. On query $\pi_2(1)$ we would return a random point in $\{2,3\}$, say 2. On query $\pi_2(2)$ we would return a random point in $\{1,3\}$, say 1. But now we are stuck, for on query $\pi_2(3)$ there is nothing correct to return. Here lazy sampling, at least in the way we just implemented it, didn't work.

We now say, more precisely, what we mean by lazy sampling, and then give some conditions under which it works, in particular justifying the first example above while identifying what makes the second one fail.

Let $\mathcal{X}, \mathcal{Y}$ be finite, non-empty sets. A *constraint function* with *locality parameter* t is a function F that assigns a boolean output to any input of the form $i_1, x_1, y_1, \ldots, i_s, x_s, y_s$, where $i_j \in [1\,..\,k]$, $x_j \in \mathcal{X}$, $y_j \in \mathcal{Y}$ and $s \in [1\,..\,t]$. Let $\mathcal{P} = \mathrm{rand}(\mathcal{X}, \mathcal{Y})$ be the set of all partial functions from $\mathcal{X}$ to $\mathcal{Y}$ and let $\mathcal{T} = \mathrm{Rand}(\mathcal{X}, \mathcal{Y})$ be the set of all total functions from $\mathcal{X}$ to $\mathcal{Y}$. We say that a set $\mathcal{F}$ of k-vectors of functions in $\mathcal{T}$ is *described*

Initialize **Game** $\text{Eager}_{\mathcal{F}}$	**Initialize** **Game** $\text{Lazy}_{\mathcal{F}}$
$(f_1, \ldots, f_k) \xleftarrow{\$} \mathcal{F}$	$f_i : \mathcal{X} \to \mathcal{Y}$ is everywhere undefined for each $i \in [1\,..\,k]$
On query $f_i(x)$	**On query** $f_i(x)$
return $f_i(x)$	**if** $f_i(x)$ **then return** $f_i(x)$
	return $f_i(x) \xleftarrow{\$} \text{Ans}_F^{f_1,\ldots,f_k}(i, x)$

Figure 3: Eager and lazy sampling games associated to $\mathcal{F}$, where $\mathcal{F}$ is given by constraint function F.

by F if $\mathcal{F}$ is exactly the set of all $(\bar{f}_1, \ldots, \bar{f}_k) \in \mathcal{T}^k$ such that

$$(\forall s \leq t)\,(\forall i_1, \ldots, i_s \in [1\,..\,k])\,(\forall x_1, \ldots, x_s \in \mathcal{X})\ \left[F(i_1, x_1, \bar{f}_{i_1}(x_1), \ldots, i_s, x_s, \bar{f}_{i_s}(x_s)) = 1\right]\ .$$

The framework we consider is that we provide adversary A with a sequence of oracles $(f_1, \ldots, f_k) \xleftarrow{\$} \mathcal{F}$ drawn at random, with uniform distribution, from a set $\mathcal{F}$ that is described via a constraint function.

The examples we have given above can be put into this framework. For example 1 we have $\mathcal{X} = \mathcal{Y} = \{0,1\}^n$ and for example 2 we have $\mathcal{X} = \mathcal{Y} = \{1, 2, 3\}$. The constraint function for example 1 has locality $t = 2$ and is defined by $F_1(i, x_1, y_1, i, x_2, y_2) = 1$ iff $(x_1 \neq x_2) \Rightarrow (y_1 \neq y_2)$. The constraint function for example 2 has locality $t = 2$ and is defined by $F_3(1, x_1, y_1, 2, x_2, y_2) = 1$ iff (a) $(x_1 = x_2) \Rightarrow (y_1 \neq y_2)$ and (b) $F_1(i, x, y, i, x', y') = 1$ for all $i \in \{1, 2\}$.

Now we explain how lazy sampling works. If $\bar{f} \in \mathcal{T}$ is consistent with $f \in \mathcal{P}$ (meaning the two are equal on all points where partial function f is defined) then we write $\bar{f} \geq f$. For $f_1, \ldots, f_k \in \mathcal{P}$, $i \in [1\,..\,k]$, $x \in \mathcal{X}$, and $y \in \mathcal{Y}$ let

$$\text{Ext}_F^{f_1,\ldots,f_k}(i, x, y) \ = \ \{\, (\bar{f}_1, \ldots, \bar{f}_k) \in \mathcal{F} \ : \ \bar{f}_j \geq f_j \ (1 \leq j \leq k) \text{ and } \bar{f}_i(x) = y \,\}$$

be the set of *extensions* of $f_1, \ldots, f_k$ relative to (i, x, y). This can be viewed as the set of all possible ways to assign values to the as-yet-undefined points of the partial functions $(f_1, \ldots, f_k)$ subject to the constraint that $f_i(x)$ is assigned y. Let $\text{Ans}_F^{f_1,\ldots,f_k}(i, x)$ be the set of all $y \in \mathcal{Y}$ such that $\text{Ext}_F^{f_1,\ldots,f_k}(i, x, y) \neq \emptyset$. This is the set of *possible answers* to query $f_i(x)$, meaning those that have non-zero probability of occurring.

Figure 3 shows two games, one describing eager sampling and the other lazy sampling. We claim that lazy sampling, as formally described in this game, captures the way it was done in our first example, in that the set of possible answers is exactly the set of points that do not violate any constraint. In example 1, from the description of F_1 we see that $\text{Ans}_F^{\pi}(1, x)$ is exactly $\overline{\textbf{image}}(\pi)$. However, the way we sampled in example 2 fails to implement what we have now formally defined as lazy sampling, explaining why it failed. To see this consider the stage where $\pi_1(i) = i$ for $i \in \{1, 2, 3\}$ and $\pi_2(1) = 2$. Then $\text{Ans}_F^{\pi_1,\pi_2}(2, 2) = \{3\}$, while in the example we said that the candidate set from which to draw $\pi_2(2)$ was $\{1, 3\}$. This shows that determining the set of possible answers purely by looking at the constraints on defined points does not work.

Now we move to saying under what conditions lazy sampling works. We say that F is *admissible* if for all $f_1, \ldots, f_k \in \mathcal{P}$, all $i \in [1\,..\,k]$, and all $x \in \mathcal{X}$

$$\forall\, y_1, y_2 \in \text{Ans}_F^{f_1,\ldots,f_k}(i, x)\ \left(|\text{Ext}_F^{f_1,\ldots,f_k}(i, x, y_1)| = |\text{Ext}_F^{f_1,\ldots,f_k}(i, x, y_2)|\right)\ .$$

In other words, the number of ways to extend $f_1, \ldots, f_k$ relative to (i, x, y) does not depend on y as long as y is allowed, or, intuitively, any two allowed values are equi-probable as answers to an oracle query. We say that $\mathcal{F}$ is *admissible* if it is described by an admissible constraint function F. Our result about eager versus lazy sampling is the following. Its proof is given later.

Lemma 8 [Principle of lazy sampling] *Let $\mathcal{F} \neq \emptyset$ be an admissible set. Then games $\text{Eager}_{\mathcal{F}}$ and $\text{Lazy}_{\mathcal{F}}$ are equivalent.* ▌

We claim that the constraint functions F_1 of example 1 is admissible, which explains why lazy sampling worked in these cases. Verifying this claim is quite easy. Suppose π has been defined on some $m-1$ points and $\pi(x)$ is the m-th query. Then for every $y \in \overline{\textbf{image}}(\pi)$ there are $(N-m)!$ possible ways to assign values to the undefined points while setting $\pi(x) = y$, meaning $|\text{Ext}_{F_1}^{\pi}(1, x, y)| = (N-m)!$ for every $y \in \text{Ans}_F^{\pi}(1, x)$.

Proof of Lemma 8: Suppose the adversary has made some number of oracle queries, resulting in the partial functions $f_1, \ldots, f_k$. Now it makes another query, $f_i(x)$. We consider the probability that a particular point $y \in \mathcal{Y}$ is returned in response, and show this is the same in both games.

Any $y \notin \mathrm{Ans}_F^{f_1,\ldots,f_k}(i,x)$ has zero probability of being returned in either game. Suppose $y \in \mathrm{Ans}_F^{f_1,\ldots,f_k}(i,x)$. Let

$$\mathcal{F}(f_1,\ldots,f_k) \;=\; \{\, (\bar{f}_1,\ldots,\bar{f}_k) \in \mathcal{F} \;:\; \bar{f}_j \geq f_j \; (1 \leq j \leq k) \,\} \;.$$

The assumption $\mathcal{F} \neq \emptyset$ implies that $\mathrm{Ans}_F^{f_1,\ldots,f_k}(i,x) \neq \emptyset$. Now the probability that y is returned as the answer to query $f_i(x)$ in $\mathrm{Eager}_{\mathcal{F}}$ is

$$\begin{aligned}
\frac{|\mathrm{Ext}_F^{f_1,\ldots,f_k}(i,x,y)|}{|\mathcal{F}(f_1,\ldots,f_k)|} &= \frac{|\mathrm{Ext}_F^{f_1,\ldots,f_k}(i,x,y)|}{\sum_{y' \in \mathrm{Ans}_F^{f_1,\ldots,f_k}(i,x)} |\mathrm{Ext}_F^{f_1,\ldots,f_k}(i,x,y')|} \\
&= \frac{|\mathrm{Ext}_F^{f_1,\ldots,f_k}(i,x,y)|}{|\mathrm{Ans}_F^{f_1,\ldots,f_k}(i,x)| \cdot |\mathrm{Ext}_F^{f_1,\ldots,f_k}(i,x,y)|} && (2) \\
&= \frac{1}{|\mathrm{Ans}_F^{f_1,\ldots,f_k}(i,x)|} \;. && (3)
\end{aligned}$$

The assumption that F is admissible justifies 2. The proof is complete because (3) is the probability that y is returned as the answer to query $f_i(x)$ in $\mathrm{Lazy}_{\mathcal{F}}$. ∎

4.4 Basic techniques

We briefly survey some other interesting or commonly used techniques. Use of most of these techniques is illustrated in the examples of this paper.

Swapping dependent and independent variables. Instead of choosing a random value $X \stackrel{\$}{\leftarrow} \{0,1\}^n$ and then defining $Y \leftarrow X \oplus C$, one can choose $Y \stackrel{\$}{\leftarrow} \{0,1\}^n$ and define $X \leftarrow Y \oplus C$. This can be generalized in natural ways. Swapping dependent and independent variables is invariably a conservative change (it doesn't affect the probability that *bad* gets set).

Resampling idiom. Let $\mathcal{S} \subseteq T$ be finite, nonempty sets. Then the code fragment $X \stackrel{\$}{\leftarrow} \mathcal{S}$ can be replaced by the equivalent code fragment $X \stackrel{\$}{\leftarrow} \mathcal{T}$, **if** $X \notin S$ **then** $X \stackrel{\$}{\leftarrow} S$. We call this motif *resampling.* It is a basic "idiom" employed in games, often with *bad* getting set, too: $X \stackrel{\$}{\leftarrow} \mathcal{T}$, **if** $X \notin S$ **then** $bad \leftarrow \mathsf{true}$, $X \stackrel{\$}{\leftarrow} S$. Introducing or removing resampling is invariably a conservative change.

Code motion. It is often convenient to move around statements, as an optimizing compiler might. Permissible code motion is usually trivial to verify because games need not need to employ the programming-language constructs (aliasing and side-effects) that complicate seeing whether or not code motion is permissible. One particular form of code motion that is often used is to postpone until **Finalize** making random choices that had been made earlier. Permissible code motion conservative.

Marking instead of recording. Suppose that a variable π is being used in a game to record a lazily-defined permutation: we start off with π everywhere undefined, and then we set some first value $\pi(X_1)$ to Y_1, and later we set some second value $\pi(X_2)$ to Y_2, and so forth. Sometimes an inspection of the code will reveal that all we are paying attention to is which points are in the domain of π and which points are in the range. In such a case, we didn't need to record the association of Y_i to X_i; we could just as well have "marked" X_i as being a now-used domain-point, and marked Y_i as being a now-used range-point. Dropping the use of Y_i may now permit other changes in the code, like code motion. The method is conservative.

Derandomizing a variable. Suppose a game G chooses a variable $X \stackrel{\$}{\leftarrow} \mathcal{X}$ and never re-defines it. We may eliminate the sample-then-assign statement $X \stackrel{\$}{\leftarrow} \mathcal{X}$ and replace all uses of X by a fixed constant $\mathtt{X} \in X$, obtaining a new game $\mathrm{G}^{\mathtt{X}}$. Given an adversary A, let H be $\mathrm{G}^{\mathtt{X}}$ for the lexicographically first $\mathtt{X} \in \mathcal{X}$ that maximizes $\Pr[\mathrm{G}_A^{\mathtt{X}}$ sets $bad\,]$. We say that game H is obtained by derandomizing the variable X. It is easy to see that $\Pr[\mathrm{G}_A$ sets $bad\,] \leq \Pr[\mathrm{H}_A$ sets $bad\,]$; that is, derandomizing a variable is a safe transformation. Derandomizing a variable is reminiscent of coin-fixing, but it is simpler to explain and justify. It does nothing to eliminate adaptivity.

Unplayable games. The games in a game chain do not normally have to be efficient: a game chain is a thought experiment that, typically, is not performed by any user or adversary. We refer to a game that seems to have no efficient implementation as an *unplayable game*. In many cases, it is perfectly fine to use unplayable games.

4.5 Further advice

We have already given a number of pieces of advice concerning the construction of games chains, such as (1) keeping each game-transition simple (at the expense of having more games) and (2) being particularly cautious with the use of lazy sampling, which doesn't apply as often as one might wish. We briefly give a few other pieces of practical advice. (3) A game chain is most easily verified when, as much as possible, games are typeset side-by-side and on as few pages as possible. Doing this may require special-purpose conventions for achieving compact notation, as well as clever typesetting, but it seems to be worth it. (4) Avoid use of **else**-clauses in **if**-statements that set *bad*; they only cause confusion. (5) At least for the "major" games in an proof chain, we suggest to number the lines and put each game in a box. (6) Avoid deeply nested **for** loops and any nontrivial flow-of-control: either makes a game much harder to understand. (7) When the analysis of a terminal game requires a case analysis, as it often seems to, it is easy to get careless by this point in a proof and make errors, overlooking necessary cases or mishandling a case.

5 Elementary Proof for the CBC MAC

Fix $n \geq 1$. A *block* is a string of length n, and M is a *string of blocks* if $|M|$ is divisible by n. If $M \in (\{0,1\}^n)^*$ is a string of blocks we let $\|M\| = |M|/n$ be the number of blocks in M and we let $M_{1\to i}$ be the first i blocks of M. By "Parse M as $M_1 \ldots M_m$" we mean to let $m \leftarrow \|M\|$ and then let M_i denote the i-th block of M. If $\pi\colon \{0,1\}^n \to \{0,1\}^n$ is a function and M is a string of n-bit blocks then we define

algorithm $\mathrm{CBC}_\pi(M)$
  Parse M as $M_1 \ldots M_m$
  $C_0 \leftarrow 0^n$
  for $i \leftarrow 1$ **to** m **do** $C_i \leftarrow C_{i-1} \oplus \pi(M_i)$
  return C_m.

Let Perm(n) denote the set of all permutations on $\{0,1\}^n$ and let Rand(mn, n) denote the set of all functions from $\{0,1\}^{mn}$ to $\{0,1\}^n$. For $m \geq 1$ and $\pi \in \mathrm{Perm}(n)$ let CBC^m_π be the restriction of CBC_π to the domain $\{0,1\}^{mn}$. Given an algorithm A having oracle-access to a function $F\colon \{0,1\}^{mn} \to \{0,1\}^n$ let

$$\mathbf{Adv}^{\mathrm{cbc}}_{n,m}(A) \;=\; \Pr[\pi \stackrel{\$}{\leftarrow} \mathrm{Perm}(n):\; A^{\mathrm{CBC}^m_\pi(\cdot)} \Rightarrow 1] - \Pr[\rho \stackrel{\$}{\leftarrow} \mathrm{Rand}(mn,n):\; A^{\rho(\cdot)} \Rightarrow 1]$$

denote the *advantage* of A. Let

$$\mathbf{Adv}^{\mathrm{cbc}}_{n,m}(q) \;=\; \max\left\{\mathbf{Adv}^{\mathrm{cbc}}_{n,m}(A)\right\}$$

where the maximum is over all adversaries A that ask at most q queries, regardless of oracle responses. To avoid working out uninteresting special cases, we assume throughout that the adversary asks $q \geq 2$ oracle queries and each has $m \geq 2$ blocks.

Here we use games to give a straightforward proof of the (known) $m^2q^2/2^n$ bound for the CBC MAC. Namely we show:

Theorem 9 [CBC MAC, standard bound] *Suppose $m, q \geq 2$ and $n \geq 1$. Then*

$$\mathbf{Adv}^{\mathrm{cbc}}_{n,m}(q) \;\leq\; \frac{m^2q^2}{2^n}\;. \;\blacksquare$$

The proof we give here is simple enough to do in a classroom lecture, and it follows rather closely the intuition for "why" the CBC MAC is secure.

Proof of Lemma 9: Let A be an adversary that asks exactly q queries and assume without loss of generality that it never repeats a query. Refer to games C0–C9 in Figure 4. Let us begin by explaining the notation used there. Each query M^s in the games is required to be a string of blocks, and we silently parse M^s to $M^s = M^s_1 M^s_2 \cdots M^s_m$ where each M_i is a block. Recall that $M^s_{1\to i} = M^s_1 \cdots M^s_i$. The function

On the s^{th} query $F(M^s)$ Game C1

$P \leftarrow \mathsf{Prefix}(M^1, \ldots, M^s)$
$C \leftarrow \mathbb{Y}[P]$
for $j \leftarrow \|P\| + 1$ **to** m **do**
  $X \leftarrow C \oplus M^s_j$
  $C \xleftarrow{\$} \{0,1\}^n$
  if $C \in \mathbf{image}(\pi)$ **then** $bad \leftarrow \mathsf{true}$, $C \xleftarrow{\$} \overline{\mathbf{image}}(\pi)$
  if $X \in \mathbf{domain}(\pi)$ **then** $bad \leftarrow \mathsf{true}$, $C \leftarrow \pi(X)$
  $\pi(X) \leftarrow C$
  $\mathbb{Y}[M^s_{1\to j}] \leftarrow C$ omit for Game C0
return C

On the s^{th} query $F(M^s)$ Game C2

$P \leftarrow \mathsf{Prefix}(M^1, \ldots, M^s)$
$C \leftarrow \mathbb{Y}[P]$
for $j \leftarrow \|P\| + 1$ **to** m **do**
  $X \leftarrow C \oplus M^s_j$
  $C \xleftarrow{\$} \{0,1\}^n$
  if $X \in \mathbf{domain}(\pi)$ **then** $bad \leftarrow \mathsf{true}$
  $\pi(X) \leftarrow C$
  $\mathbb{Y}[M^s_{1\to j}] \leftarrow C$
return C

On the s^{th} query $F(M^s)$ Game C3

$P \leftarrow \mathsf{Prefix}(M^1, \ldots, M^s)$
$C \leftarrow \mathbb{Y}[P]$
for $j \leftarrow \|P\| + 1$ **to** m **do**
  $X \leftarrow C \oplus M^s_j$
  $C \xleftarrow{\$} \{0,1\}^n$
  if $X \in \mathbf{domain}(\pi)$ **then** $bad \leftarrow \mathsf{true}$
  $\pi(X) \leftarrow \mathsf{defined}$
  $\mathbb{Y}[M^s_{1\to j}] \leftarrow C$
return C

On the s^{th} query $F(M^s)$ Game C4

$P \leftarrow \mathsf{Prefix}(M^1, \ldots, M^s)$
$C \leftarrow \mathbb{Y}[P]$
for $j \leftarrow \|P\| + 1$ **to** m **do**
  $X \leftarrow C \oplus M^s_j$
  if $X \in \mathbf{domain}(\pi)$ **then** $bad \leftarrow \mathsf{true}$
  $\pi(X) \leftarrow \mathsf{defined}$
  $C \leftarrow \mathbb{Y}[M^s_{1\to j}] \xleftarrow{\$} \{0,1\}^n$
$Z^s \xleftarrow{\$} \{0,1\}^n$
return Z^s

Game C5

for $s \leftarrow 1$ **to** q **do**
  $\mathtt{P}^s \leftarrow \mathsf{Prefix}(\mathtt{M}^1, \ldots, \mathtt{M}^s)$
  $C \leftarrow \mathbb{Y}[\mathtt{P}^s]$
  for $j \leftarrow \|\mathtt{P}^s\| + 1$ **to** m **do**
    $X \leftarrow C \oplus \mathtt{M}^s_j$
    if $X \in \mathbf{domain}(\pi)$ **then** $bad \leftarrow \mathsf{true}$
    $\pi(X) \leftarrow \mathsf{defined}$
    $C \leftarrow \mathbb{Y}[\mathtt{M}^s_{1\to j}] \xleftarrow{\$} \{0,1\}^n$

Game C6

for $s \leftarrow 1$ **to** q **do**
  $\mathtt{P}^s \leftarrow \mathsf{Prefix}(\mathtt{M}^1, \ldots, \mathtt{M}^s)$
  $C \leftarrow \mathbb{Y}[\mathtt{P}^s]$
  $X \leftarrow C \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|+1}$
  if $X \in \mathbf{domain}(\pi)$ **then** $bad \leftarrow \mathsf{true}$
  $\pi(X) \leftarrow \mathsf{defined}$
  $C \leftarrow \mathbb{Y}[\mathtt{M}^s_{1\to\|\mathtt{P}^s\|+1}] \xleftarrow{\$} \{0,1\}^n$
  for $j \leftarrow \|\mathtt{P}^s\| + 2$ **to** m **do**
    $X \leftarrow C \oplus \mathtt{M}^s_j$
    if $X \in \mathbf{domain}(\pi)$ **then** $bad \leftarrow \mathsf{true}$
    $\pi(X) \leftarrow \mathsf{defined}$
    $C \leftarrow \mathbb{Y}[\mathtt{M}^s_{1\to j}] \xleftarrow{\$} \{0,1\}^n$

Game C7

for $X \in \{0,1\}^+$ **do** $\mathbb{Y}[X] \xleftarrow{\$} \{0,1\}^n$
for $s \leftarrow 1$ **to** q **do**
  $\mathtt{P}^s \leftarrow \mathsf{Prefix}(\mathtt{M}^1, \ldots, \mathtt{M}^s)$
  if $\mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|+1} \in \mathbf{domain}(\pi)$ **then** $bad \leftarrow \mathsf{true}$
  $\pi(\mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|+1}) \leftarrow \mathsf{defined}$
  for $j \leftarrow \|\mathtt{P}^s\| + 2$ **to** m **do**
    if $\mathbb{Y}[\mathtt{M}^s_{1\to j-1}] \oplus \mathtt{M}^s_j \in \mathbf{domain}(\pi)$ **then** $bad \leftarrow \mathsf{true}$
    $\pi(\mathbb{Y}[\mathtt{M}^s_{1\to j-1}] \oplus \mathtt{M}^s_j) \leftarrow \mathsf{defined}$

Game C8

for $X \in \{0,1\}^+$ **do** $\mathbb{Y}[X] \xleftarrow{\$} \{0,1\}^n$
for $s \leftarrow 1$ **to** q **do**
  $\mathtt{P}^s \leftarrow \mathsf{Prefix}(\mathtt{M}^1, \ldots, \mathtt{M}^s)$
  if $\mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|+1} \in \mathbf{domain}(\pi)$ **then** $bad \leftarrow \mathsf{true}$
  $\pi(\mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|+1}) \leftarrow \mathsf{defined}$
  for $j \leftarrow \|\mathtt{P}^s\| + 1$ **to** $m - 1$ **do**
    if $\mathbb{Y}[\mathtt{M}^s_{1\to j}] \oplus \mathtt{M}^s_{j+1} \in \mathbf{domain}(\pi)$ **then** $bad \leftarrow \mathsf{true}$
    $\pi(\mathbb{Y}[\mathtt{M}^s_{1\to j}] \oplus \mathtt{M}^s_{j+1}) \leftarrow \mathsf{defined}$

Game C9

for $X \in \{0,1\}^+$ **do** $\mathbb{Y}[X] \xleftarrow{\$} \{0,1\}^n$
for $s \leftarrow 1$ **to** q **do** $\mathtt{P}^s \leftarrow \mathsf{Prefix}(\mathtt{M}^1, \ldots, \mathtt{M}^s)$
$bad \leftarrow \exists (r,i) \neq (s,j)(r \le s)(i \ge \|\mathtt{P}^r\|+1)(j \ge \|\mathtt{P}^s\|+1)$
  $\mathbb{Y}[\mathtt{P}^r] \oplus \mathtt{M}^r_{\|\mathtt{P}^r\|+1} = \mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|+1}$ **and** $r < s$ **or**
  $\mathbb{Y}[\mathtt{M}^r_{1\to i}] \oplus \mathtt{M}^r_{i+1} = \mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|+1}$ **or**
  $\mathbb{Y}[\mathtt{M}^r_{1\to i}] \oplus \mathtt{M}^r_{i+1} = \mathbb{Y}[\mathtt{M}^s_{1\to j}] \oplus \mathtt{M}^s_{j+1}$ **or**
  $\mathbb{Y}[\mathtt{P}^r] \oplus \mathtt{M}^r_{\|\mathtt{P}^r\|+1} = \mathbb{Y}[\mathtt{M}^s_{1\to j}] \oplus \mathtt{M}^s_{j+1}$

Figure 4: Games used in the CBC MAC analysis. Let $\mathsf{Prefix}(M^1, \ldots, M^s)$ be ε if $s = 1$, else the longest string $P \in (\{0,1\}^n)^*$ s.t. P is a prefix of M^s and M^r for some $r < s$. In each game, **Initialize** sets $\mathbb{Y}[\varepsilon] \leftarrow 0^n$.

$\pi\colon \{0,1\}^n \to \{0,1\}^n$ is initially undefined at each point. The set $\mathbf{domain}(\pi)$ grows as we define points $\pi(X)$, while $\overline{\mathbf{image}}(\pi)$, initially $\{0,1\}^n$, correspondingly shrinks. The table $\mathbb{Y}$ stores blocks and is indexed by strings of blocks P having at most m blocks. A random block will come to occupy selected entries $\mathbb{Y}[X]$ except for $\mathbb{Y}[\varepsilon]$, which is initialized to the constant block 0^n and is never changed. The value $\mathsf{defined}$ (introduced at line 306) is an arbitrary point of $\{0,1\}^n$, say 0^n. Finally, $\mathsf{Prefix}(M^1,\ldots,M^s)$ is the longest string of blocks $P = P_1 \cdots P_p$ that is a prefix of M^s and is also a prefix of M^r for some $r < s$. If Prefix is applied to a single string the result is the empty string, $\mathsf{Prefix}(P^1) = \varepsilon$. As an example, letting A, B, and C be distinct blocks, $\mathsf{Prefix}(\mathtt{ABC}) = \varepsilon$, $\mathsf{Prefix}(\mathtt{ACC},\ \mathtt{ACB},\ \mathtt{ABB},\ \mathtt{ABA}) = \mathtt{AB}$, and $\mathsf{Prefix}(\mathtt{ACC},\ \mathtt{ACB},\ \mathtt{BBB}) = \varepsilon$.

We briefly explain the game chain up until the terminal game. Game C0 is obtained from game C1 by dropping the assignment statements that immediately follow the setting of *bad*. Game **C1** is a realization of $\mathsf{CBC}^m[\mathrm{Perm}(n)]$ and game **C0** is a realization of $\mathrm{Rand}(mn,n)$. The games use lazy sampling of a random permutation (as described in Section 4.3) and the resampling idiom (as described in Section 4.4). Games C1 and C0 are designed so that the fundamental lemma applies, so the advantage of A in attacking the CBC construction is at most $\Pr[A^{\mathrm{C0}} \text{ sets } bad]$. **C0→C2**: The C0 → C2 transition is a lossy transition that takes care of *bad* getting set at line 105, which clearly happens with probability at most $(0+1+\cdots+(mq-1))/2^n \le 0.5\,m^2q^2/2^n$, so $\Pr[A^{\mathrm{C0}} \text{ sets } bad] \le \Pr[A^{\mathrm{C2}} \text{ sets } bad] + 0.5\,m^2q^2/2^n$. **C2→C3**: Next notice that in game C2 we never actually use the values assigned to π, all that matters is that we *record* that a value had been placed in the domain of π, and so game C3 does just that, dropping a fixed value $\mathsf{defined} = 0^n$ into $\pi(X)$ when we want X to join the domain of π. This is the technique we called "marking instead of recording" in Section 4.4. The change is conservative. **C3→C4**: Now notice that in game C3 the value returned to the adversary, although dropped into $\mathbb{Y}[M_1^s \cdots M_m^s]$, is never subsequently used in the game so we could as well choose a random value Z^s and return it to the adversary, doing nothing else with Z^s. This is the change made for game C4. The transition is conservative. **C4→C5**: Changing game C4 to C5 is by the coin-fixing technique of Section 4.2. It is a particularly simple application of the technique: game C4 is oblivious since no variables are used to form Z^s-values. Coin-fixing in this case amounts to letting the adversary choose the sequence of queries $\mathtt{M}^1,\ldots,\mathtt{M}^m$ it asks and the sequence of answers returned to it. The queries still have to be valid: each M^s is an mn-bit string different from all prior ones: that is the query/response set. For the worst $\mathtt{M}^1,\ldots,\mathtt{M}^m$, which the coin-fixing technique fixes, $\Pr[A^{\mathrm{C4}} \text{ sets } bad] \le \Pr[\mathrm{C5} \text{ sets } bad]$. Remember that, when applicable, coin-fixing is safe. **C5→C6**: Game C6 unrolls the first iteration of the loop at lines 503–507. This transformation is conservative. **C6→C7**: Game C7 is a rewriting of game C6 that omits mention of the variables C and X, directly using values from the $\mathbb{Y}$-table instead, whose values are now chosen at the beginning of the game. The change is conservative. **C7→C8**: Game C8 simply re-indexes the for loop at line 705. The change is conservative. **C8→C9**: Game C9 restructures the setting of *bad* inside the loop at 802–807 to set *bad* in a single statement. Points were into the domain of π at lines 804 and 807 and we checked if any of these points coincide with specified other points at lines 803 and 806. The change is conservative.

At this point, we have only to bound $\Pr[A^{\mathrm{C9}} \text{ sets } bad]$. We do this using the sum bound and a case analysis. Fix any $r, i,\ s, j$ as specified in line 902. Consider the following ways that *bad* can get set to true.

Line 903. We first bound $\Pr[\mathbb{Y}[\mathtt{P}^r] \oplus \mathtt{M}^r_{\|\mathtt{P}^r\|+1} = \mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|+1}]$. If $\mathtt{P}^r = \mathtt{P}^s = \varepsilon$ then $\Pr[\mathbb{Y}[\mathtt{P}^r] \oplus \mathtt{M}^r_{\|\mathtt{P}^r\|+1} = \mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|+1}] = \Pr[\mathtt{M}^r_1 = \mathtt{M}^s_1] = 0$ because $\mathtt{M}^r$ and $\mathtt{M}^s$, having only ε as a common block prefix, must differ in their first block. If $\mathtt{P}^r = \varepsilon$ but $\mathtt{P}^s \ne \varepsilon$ then $\Pr[\mathbb{Y}[\mathtt{P}^r] \oplus \mathtt{M}^r_{\|\mathtt{P}^r\|+1} = \mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|+1}] = \Pr[\mathtt{M}^r_1 = \mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|+1}] = 2^{-n}$ since the probability expression involves the single random variable $\mathbb{Y}[\mathtt{P}^s]$ that is uniformly distributed in $\{0,1\}^n$. If $\mathtt{P}^r \ne \varepsilon$ and $\mathtt{P}^s = \varepsilon$ the same reasoning applies. If $\mathtt{P}^r \ne \varepsilon$ and $\mathtt{P}^s \ne \varepsilon$ then $\Pr[\mathbb{Y}[\mathtt{P}^r] \oplus \mathtt{M}^r_{\|\mathtt{P}^r\|+1} = \mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|+1}] = 2^{-n}$ unless $\mathtt{P}^r = \mathtt{P}^s$, so assume that to be the case. Then $\Pr[\mathbb{Y}[\mathtt{P}^r] \oplus \mathtt{M}^r_{\|\mathtt{P}^r\|+1} = \mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|+1}] = \Pr[\mathtt{M}^r_{\|\mathtt{P}^r\|+1} = \mathtt{M}^s_{\|\mathtt{P}^s\|+1}] = 0$ because $\mathtt{P}^r = \mathtt{P}^s$ is the *longest* block prefix that coincides in $\mathtt{M}^r$ and $\mathtt{M}^s$.

Line 904. We want to bound $\Pr[\mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|+1} = \mathbb{Y}[\mathtt{M}^r_{1\to i}] \oplus \mathtt{M}^r_{i+1}]$. If $\mathtt{P}^s = \varepsilon$ then $\Pr[\mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|+1} = \mathbb{Y}[\mathtt{M}^r_{1\to i}] \oplus \mathtt{M}^r_{i+1}] = \Pr[\mathtt{M}^s_{\|\mathtt{P}^s\|+1} = \mathbb{Y}[\mathtt{M}^r_{1\to i}] \oplus \mathtt{M}^r_{i+1}] = 2^{-n}$ because it involves a single random value $\mathbb{Y}[\mathtt{M}^r_{1\to i}]$. So assume that $\mathtt{P}^s \ne \varepsilon$. Then $\Pr[\mathbb{Y}[\mathtt{P}^s] \oplus \mathtt{M}^s_{\|\mathtt{P}^s\|+1} = \mathbb{Y}[\mathtt{M}^r_{1\to i}] \oplus \mathtt{M}^r_{i+1}] = 2^{-n}$ unless $\mathtt{P}^s = \mathtt{M}^r_{1\to i}$ in which case we are looking at $\Pr[\mathtt{M}^s_{\|\mathtt{P}^s\|+1} = \mathtt{M}^r_{\|\mathtt{P}^s\|+1}]$. But this is 0 because $\mathtt{P}^s = \mathtt{M}^r_{1\to i}$ means that the longest prefix that $\mathtt{M}^s$ shares with $\mathtt{M}^r$ is $\mathtt{P}^s$ and so $\mathtt{M}^s_{\|\mathtt{P}^s\|+1} \ne \mathtt{M}^r_{\|\mathtt{P}^s\|+1}$.

Line 905. What is $\mathbb{Y}\,[\mathtt{M}^s_{1\to j}] \oplus \mathtt{M}^s_{j+1} = \mathbb{Y}\,[\mathtt{M}^r_{1\to i}] \oplus \mathtt{M}^r_{i+1}$. It is 2^{-n} unless $i = j$ and $\mathtt{M}^s_{1\to j} = \mathtt{M}^r_{1\to i}$. In that case $\|\mathtt{P}^s\| \geq j$ and $\|\mathtt{P}^r\| \geq i$, contradicting our choice of allowed values for i and j at line 902.

Line 906. We must bound $\Pr[\mathbb{Y}\,[\mathtt{P}^r] \oplus \mathtt{M}^r_{\|\mathtt{P}^r\|+1} = \mathbb{Y}\,[\mathtt{M}^s_{1\to j}] \oplus \mathtt{M}^s_{j+1}]$. As before, this is 2^{-n} unless $\mathtt{P}^r = \mathtt{M}^s_{1\to j}$ but we can not have that $\mathtt{P}^r = \mathtt{M}^s_{1\to j}$ because $j \geq \|\mathtt{P}^s\| + 1$.

There are at most $0.5m^2q^2$ tuples (r, i, s, j) considered at line 902 and we now know that for each of them *bad* gets set with probability at most 2^{-n}. So Pr[Game C9 sets *bad*] $\leq 0.5m^2q^2/2^n$. Combining with the loss from the C0→C2 transition we have that Pr[Game C0 sets *bad*] $\leq m^2q^2/2^n$, completing the proof. ▌

6 A Game-Based Proof for OAEP

We recall the needed background for the asymmetric encryption scheme OAEP [BR94]. A trapdoor-permutation generator with associated security parameter k is a randomized algorithm $\mathcal{F}$ that takes a number, the security parameter, as input and returns a pair (f, f^{-1}) where $f\colon \{0,1\}^k \to \{0,1\}^k$ is (the encoding of) a permutation and f^{-1} is (the encoding of) its inverse. Let

$$\mathbf{Adv}^{\mathrm{owf}}_{\mathcal{F}}(I) \;=\; \Pr[(f, f^{-1}) \stackrel{\$}{\leftarrow} \mathcal{F}(k);\, x \stackrel{\$}{\leftarrow} \{0,1\}^k :\; I(f, f(x)) = x]$$

be the advantage of adversary I in inverting $\mathcal{F}$. Let $\rho < k$ be an integer. The key-generation algorithm of asymmetric encryption scheme $\mathsf{OAEP}^\rho[\mathcal{F}]$ is simply $\mathcal{F}$, meaning it returns f as the public key and f^{-1} as the secret key. The encryption and decryption algorithms have oracles $G\colon \{0,1\}^\rho \to \{0,1\}^{k-\rho}$ and $H\colon \{0,1\}^{k-\rho} \to \{0,1\}^\rho$ and work as follows (for the basic, no-authenticity scheme):

algorithm $\mathcal{E}^{G,H}_f(M)$ /* $M \in \{0,1\}^{k-\rho}$ */	**algorithm** $\mathcal{D}^{G,H}_{f^{-1}}(Y)$ /* $Y \in \{0,1\}^k$ */
$R \stackrel{\$}{\leftarrow} \{0,1\}^\rho,\ S \leftarrow G(R) \oplus M,\ T \leftarrow H(S) \oplus R$	$X \leftarrow f^{-1}(Y),\ S \leftarrow X[1\,..\,k-\rho],\ T \leftarrow X[k-\rho+1\,..\,k]$
$Y \leftarrow f(S \,\|\, T)$	$R \leftarrow H(S) \oplus T,\ M \leftarrow G(R) \oplus S$
return Y	**return** M

Security of an asymmetric encryption scheme $\mathsf{AE} = (\mathcal{F}, \mathcal{E}, \mathcal{D})$ is defined via the following game. Keys (f, f^{-1}) are chosen by running $\mathcal{F}$, and a bit b is chosen at random. Adversary A is given input f and a left-or-right oracle $E(\cdot,\cdot)$ which on input a pair M_0, M_1 of equal-length messages computes $Y \stackrel{\$}{\leftarrow} \mathcal{E}_f(M_b)$ and returns Y. The output of adversary is a bit b' and $\mathbf{Adv}^{\text{fg-cpa}}_{\mathsf{AE}}(A) = 2\Pr[b' = b] - 1$.

Theorem 10 *Let A be an adversary with running time t_A, making at most q_G queries to its G oracle, q_H to its H oracle, and exactly one query to its left-or-right oracle. Then there is an adversary I with running time t_I such that*

$$\mathbf{Adv}^{\mathrm{owf}}_{\mathcal{F}}(I) \;\geq\; \frac{1}{2}\mathbf{Adv}^{\text{fg-cpa}}_{\mathsf{OAEP}^\rho[\mathcal{F}]}(A) - \frac{2q_G}{2^\rho} - \frac{q_H}{2^{k-\rho}} \qquad \text{and} \qquad t_I \;\leq\; t_A + c\,q_G q_H\, t_{\mathcal{F}}$$

where $t_{\mathcal{F}}$ is the time for one computation of a function output by $\mathcal{F}$ and c is an absolute constant depending only on details of the model of computation. ▌

Proof of Theorem 10: The proof is based on games shown in Figures 5 and 6. As usual, we have striven to makes steps between adjacent games small at the cost of a somewhat longer game chain, a tradeoff that we believe increases easy verifiability. For the analysis let $p_i = \Pr[\, out = b$ in Ri $]$ $(0 \leq i \leq 5)$. **R0**: Game R0 perfectly mimics the game defining the security of $\mathsf{OAEP}^\rho[\mathcal{F}]$. Thus

$$\frac{1}{2} + \frac{1}{2}\mathbf{Adv}^{\text{fg-cpa}}_{\mathsf{OAEP}^\rho[\mathcal{F}]}(A) \;=\; p_0 \;=\; p_1 + (p_0 - p_1) \;\leq\; p_1 + \Pr[\text{R0 sets } \mathit{bad}\,]\,,$$

the last step by the fundamental lemma. Since game R0 chooses R^*, S^* at random, Pr[R0 sets *bad*] $\leq q_G/2^\rho + q_H/2^{k-\rho}$. **R2:** Game R2 differs from game R1 only in the setting of *bad*, so $p_1 = p_2$, and using the fundamental lemma again we have

$$p_1 \;=\; p_2 \;=\; p_3 + (p_2 - p_3) \;\leq\; p_3 + \Pr[\text{R3 sets } \mathit{bad}\,]\,.$$

On query $E(M_0, M_1)$ — Game R0

000 $R^* \stackrel{\$}{\leftarrow} \{0,1\}^\rho$
001 $GR^* \stackrel{\$}{\leftarrow} \{0,1\}^{k-\rho}$
002 **if** $G[R^*]$ **then** $bad \leftarrow \mathsf{true}$, $GR^* \leftarrow G[R^*]$
003 $S^* \leftarrow GR^* \oplus M_b$
004 $HS^* \stackrel{\$}{\leftarrow} \{0,1\}^\rho$
005 **if** $H[S^*]$ **then** $bad \leftarrow \mathsf{true}$, $HS^* \leftarrow H[S^*]$
006 $T^* \leftarrow R^* \oplus HS^*$
007 **return** $Y^* \leftarrow f(S^* \,\|\, T^*)$

$\downarrow$ *Eliminate 002, 005 with loss* $q_G/2^\rho + q_H/2^{k-\rho}$

On query $G(R)$
010 **if** $R = R^*$ **then return** $G[R^*] \leftarrow GR^*$
011 **return** $G[R] \stackrel{\$}{\leftarrow} \{0,1\}^{k-\rho}$

On query $H(S)$
020 **if** $S = S^*$ **then return** $H[S^*] \leftarrow HS^*$
021 **return** $H[S] \stackrel{\$}{\leftarrow} \{0,1\}^\rho$

On query $E(M_0, M_1)$ — Game R1

100 $R^* \stackrel{\$}{\leftarrow} \{0,1\}^\rho$
101 $GR^* \stackrel{\$}{\leftarrow} \{0,1\}^{k-\rho}$
102 $S^* \leftarrow GR^* \oplus M_b$
103 $HS^* \stackrel{\$}{\leftarrow} \{0,1\}^\rho$
104 $T^* \leftarrow R^* \oplus HS^*$
105 **return** $Y^* \leftarrow f(S^* \,\|\, T^*)$

$\downarrow$ *Introduce bad at 110*

On query $G(R)$
110 **if** $R = R^*$ **then return** $G[R^*] \leftarrow GR^*$
111 **return** $G[R] \stackrel{\$}{\leftarrow} \{0,1\}^{k-\rho}$

On query $H(S)$
120 **if** $S = S^*$ **then return** $H[S^*] \leftarrow HS^*$
121 **return** $H[S] \stackrel{\$}{\leftarrow} \{0,1\}^\rho$

On query $E(M_0, M_1)$ — Game R2

200 $R^* \stackrel{\$}{\leftarrow} \{0,1\}^\rho$
201 $GR^* \stackrel{\$}{\leftarrow} \{0,1\}^{k-\rho}$
202 $S^* \leftarrow GR^* \oplus M_b$
203 $HS^* \stackrel{\$}{\leftarrow} \{0,1\}^\rho$
204 $T^* \leftarrow R^* \oplus HS^*$
205 **return** $Y^* \leftarrow f(S^* \,\|\, T^*)$

$\downarrow$ *Eliminate statement after bad, apply fund. lemma*

On query $G(R)$
210 **if** $R = R^*$ **then** $bad \leftarrow \mathsf{true}$, **return** $G[R^*] \leftarrow GR^*$
211 **return** $G[R] \stackrel{\$}{\leftarrow} \{0,1\}^{k-\rho}$

On query $H(S)$
220 **if** $S = S^*$ **then return** $H[S^*] \leftarrow HS^*$
221 **return** $H[S] \stackrel{\$}{\leftarrow} \{0,1\}^\rho$

On query $E(M_0, M_1)$ — Game R3

300 $R^* \stackrel{\$}{\leftarrow} \{0,1\}^\rho$
301 $GR^* \stackrel{\$}{\leftarrow} \{0,1\}^{k-\rho}$
302 $S^* \leftarrow GR^* \oplus M_b$
303 $HS^* \stackrel{\$}{\leftarrow} \{0,1\}^\rho$
304 $T^* \leftarrow R^* \oplus HS^*$
305 **return** $Y^* \leftarrow f(S^* \,\|\, T^*)$

$\downarrow$ *Swap rand/ind vars at 301, 302. Eliminate unused var*

On query $G(R)$
310 **if** $R = R^*$ **then** $bad \leftarrow \mathsf{true}$
311 **return** $G[R] \stackrel{\$}{\leftarrow} \{0,1\}^{k-\rho}$

On query $H(S)$
320 **if** $S = S^*$ **then return** $H[S^*] \leftarrow HS^*$
321 **return** $H[S] \stackrel{\$}{\leftarrow} \{0,1\}^\rho$

On query $E(M_0, M_1)$ — Game R4

400 $R^* \stackrel{\$}{\leftarrow} \{0,1\}^\rho$
401 $S^* \stackrel{\$}{\leftarrow} \{0,1\}^{k-\rho}$
402 $HS^* \stackrel{\$}{\leftarrow} \{0,1\}^\rho$
403 $T^* \leftarrow R^* \oplus HS^*$
404 **return** $Y^* \leftarrow f(S^* \,\|\, T^*)$

On query $G(R)$
410 **if** $R = R^*$ **then** $bad \leftarrow \mathsf{true}$
411 **return** $G[R] \stackrel{\$}{\leftarrow} \{0,1\}^{k-\rho}$

On query $H(S)$
420 **if** $S = S^*$ **then return** $H[S^*] \leftarrow HS^*$
421 **return** $H[S] \stackrel{\$}{\leftarrow} \{0,1\}^\rho$

$\downarrow$ *Rewrite 410, breaking into two cases*

On query $E(M_0, M_1)$ — Game R5

500 $R^* \stackrel{\$}{\leftarrow} \{0,1\}^\rho$
501 $S^* \stackrel{\$}{\leftarrow} \{0,1\}^{k-\rho}$
502 $HS^* \stackrel{\$}{\leftarrow} \{0,1\}^\rho$
503 $T^* \leftarrow R^* \oplus HS^*$
504 **return** $Y^* \leftarrow f(S^* \,\|\, T^*)$

On query $G(R)$
510 **if** $H[S^*]$ **and** $R = R^*$ **then** $bad \leftarrow \mathsf{true}$
511 **if** $\neg H[S^*]$ **and** $R = R^*$ **then** $bad \leftarrow \mathsf{true}$
512 **return** $G[R] \stackrel{\$}{\leftarrow} \{0,1\}^{k-\rho}$

On query $H(S)$
520 **if** $S = S^*$ **then return** $H[S^*] \leftarrow HS^*$
521 **return** $H[S] \stackrel{\$}{\leftarrow} \{0,1\}^\rho$

$\swarrow \searrow$ *Separate analysis for bad set at 510 (game* A0*) and 511 (game* B0*)*

Figure 5: Games used in the analysis of OAEP. **Initialize** is the same in all of these games: $(f, f^{-1}) \stackrel{\$}{\leftarrow} \mathcal{F}(k)$, $b \stackrel{\$}{\leftarrow} \{0,1\}$, **return** $inp \leftarrow f$. **Finalize** is also the same: **return** $out = b$.

Game A0

On query $E(M_0, M_1)$

000 $R^* \xleftarrow{\$} \{0,1\}^\rho$ 001 $S^* \xleftarrow{\$} \{0,1\}^{k-\rho}$ 002 $HS^* \xleftarrow{\$} \{0,1\}^\rho$ 003 $T^* \leftarrow R^* \oplus HS^*$

004 **return** $Y^* \leftarrow f(S^* \,\|\, T^*)$

↓ Swap rand/ind vars at 002, 003

On query $G(R)$

010 **if** $H[S^*]$ **and** $R = R^*$ **then** $bad \leftarrow \mathsf{true}$

011 **return** $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

On query $H(S)$

020 **if** $S = S^*$ **then return** $H[S^*] \leftarrow HS^*$

021 **return** $H[S] \xleftarrow{\$} \{0,1\}^\rho$

Game A1

On query $E(M_0, M_1)$

100 $R^* \xleftarrow{\$} \{0,1\}^\rho$ 101 $S^* \xleftarrow{\$} \{0,1\}^{k-\rho}$ 102 $T^* \xleftarrow{\$} \{0,1\}^\rho$ 103 $HS^* \leftarrow R^* \oplus T^*$

104 **return** $Y^* \leftarrow f(S^* \,\|\, T^*)$

↓ Eliminate use of HS at 120 and its defn at 103*

On query $G(R)$

110 **if** $H[S^*]$ **and** $R = R^*$ **then** $bad \leftarrow \mathsf{true}$

111 **return** $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

On query $H(S)$

120 **if** $S = S^*$ **then return** $H[S^*] \leftarrow HS^*$

121 **return** $H[S] \xleftarrow{\$} \{0,1\}^\rho$

Game A2

On query $E(M_0, M_1)$

200 $S^* \xleftarrow{\$} \{0,1\}^{k-\rho}$ 201 $T^* \xleftarrow{\$} \{0,1\}^\rho$ 202 $R^* \xleftarrow{\$} \{0,1\}^\rho$ 203 **return** $Y^* \leftarrow f(S^* \,\|\, T^*)$

On query $G(R)$

210 **if** $H[S^*]$ **and** $R = R^*$ **then** $bad \leftarrow \mathsf{true}$

211 **return** $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

On query $H(S)$ *↓ Defer selection of R* until needed*

220 **if** $S = S^*$ **then return** $H[S^*] \leftarrow R^* \oplus T^*$

221 **return** $H[S] \xleftarrow{\$} \{0,1\}^\rho$

Game A3

On query $E(M_0, M_1)$

300 $S^* \xleftarrow{\$} \{0,1\}^{k-\rho}$ 301 $T^* \xleftarrow{\$} \{0,1\}^\rho$ 302 **return** $Y^* \leftarrow f(S^* \,\|\, T^*)$

On query $G(R)$

310 **if** $H[S^*]$ **and** $R = R^*$ **then** $bad \leftarrow \mathsf{true}$

311 **return** $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

On query $H(S)$ *↓ Use optimistic sampling for 320–321*

320 **if** $S = S^*$ **then** $R^* \xleftarrow{\$} \{0,1\}^\rho$, **return** $H[S^*] \leftarrow R^* \oplus T^*$

321 **return** $H[S] \xleftarrow{\$} \{0,1\}^\rho$

Game A4

On query $E(M_0, M_1)$

400 $S^* \xleftarrow{\$} \{0,1\}^{k-\rho}$ 401 $T^* \xleftarrow{\$} \{0,1\}^\rho$ 402 **return** $Y^* \leftarrow f(S^* \,\|\, T^*)$

On query $G(R)$

410 **if** $H[S^*]$ **and** $R = R^*$ **then** $bad \leftarrow \mathsf{true}$

411 **return** $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

On query $H(S)$

420 $H[S] \xleftarrow{\$} \{0,1\}^\rho$

421 **if** $S = S^*$ **then** $R^* \leftarrow H[S^*] \oplus T^*$

422 **return** $H[S]$ *↓ Replace R* at 410 by its defn and simplify*

Game A5

On query $E(M_0, M_1)$

500 $S^* \xleftarrow{\$} \{0,1\}^{k-\rho}$ 501 $T^* \xleftarrow{\$} \{0,1\}^\rho$ 502 **return** $Y^* \leftarrow f(S^* \,\|\, T^*)$

On query $G(R)$

510 **if** $R = H[S^*] \oplus T^*$ **then** $bad \leftarrow \mathsf{true}$

511 **return** $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

On query $H(S)$

520 **return** $H[S] \xleftarrow{\$} \{0,1\}^\rho$

↓ Eliminate S at 510*

Game A6

On query $E(M_0, M_1)$

600 $S^* \xleftarrow{\$} \{0,1\}^{k-\rho}$ 601 $T^* \xleftarrow{\$} \{0,1\}^\rho$ 602 **return** $Y^* \leftarrow f(S^* \,\|\, T^*)$

On query $G(R)$

610 **if** $\exists\, S$ s.t. $f(S \,\|\, T^*) = Y^*$ **and** $R = H[S] \oplus T^*$ **then** $bad \leftarrow \mathsf{true}$

611 **return** $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

On query $H(S)$

620 **return** $H[S] \xleftarrow{\$} \{0,1\}^\rho$

↓ Eliminate T at 610. Replace 600–602 by equivalent*

Game A7

On query $E(M_0, M_1)$

700 **return** $Y^* \xleftarrow{\$} \{0,1\}^k$

On query $G(R)$

710 **if** $\exists\, S$ s.t. $f(S \,\|\, H[S] \oplus R) = Y^*$ **then** $bad \leftarrow \mathsf{true}$

711 **return** $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

On query $H(S)$

720 **return** $H[S] \xleftarrow{\$} \{0,1\}^\rho$

Bound bad getting set by $\mathbf{Adv}^{\mathrm{owf}}_{\mathcal{F}}(I)$ □

Figure 6: Games used in the analysis of OAEP. **Initialize** is the same in all of these games: $(f, f^{-1}) \xleftarrow{\$} \mathcal{F}(k)$, $b \xleftarrow{\$} \{0,1\}$, **return** $inp \leftarrow f$. **Finalize** is also the same: **return** $out = b$.

On query $E(M_0, M_1)$ — Game B0

000 $R^* \xleftarrow{\$} \{0,1\}^\rho$ 001 $S^* \xleftarrow{\$} \{0,1\}^{k-\rho}$ 002 $HS^* \xleftarrow{\$} \{0,1\}^\rho$ 003 $T^* \leftarrow R^* \oplus HS^*$

004 **return** $Y^* \leftarrow f(S^* \,\|\, T^*)$

$\downarrow$ *Eliminate 020: coins that set bad never have* $S = S^*$

On query $G(R)$

010 **if** $\neg H[S^*]$ **and** $R = R^*$ **then** $bad \leftarrow$ true

011 **return** $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

On query $H(S)$

020 **if** $S = S^*$ **then return** $H[S^*] \leftarrow HS^*$

021 **return** $H[S] \xleftarrow{\$} \{0,1\}^\rho$

On query $E(M_0, M_1)$ — Game B1

100 $R^* \xleftarrow{\$} \{0,1\}^\rho$ 101 $S^* \xleftarrow{\$} \{0,1\}^{k-\rho}$ 102 $HS^* \xleftarrow{\$} \{0,1\}^\rho$ 103 $T^* \leftarrow R^* \oplus HS^*$

104 **return** $Y^* \leftarrow f(S^* \,\|\, T^*)$

On query $G(R)$

110 **if** $\neg H[S^*]$ **and** $R = R^*$ **then** $bad \leftarrow$ true

111 **return** $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

On query $H(S)$

120 **return** $H[S] \xleftarrow{\$} \{0,1\}^\rho$

$\downarrow$ *Conservative replacement of 110: drop first conjunct*

On query $E(M_0, M_1)$ — Game B2

200 $R^* \xleftarrow{\$} \{0,1\}^\rho$ 201 $S^* \xleftarrow{\$} \{0,1\}^{k-\rho}$ 202 $T^* \xleftarrow{\$} \{0,1\}^k$ 203 **return** $Y^* \leftarrow f(S^* \,\|\, T^*)$

On query $G(R)$

210 **if** $R = R^*$ **then** $bad \leftarrow$ true

211 **return** $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

On query $H(S)$

220 **return** $H[S] \xleftarrow{\$} \{0,1\}^\rho$

bad set with prob at most $q_R/2^\rho$ □

Figure 7: Games used in the analysis of OAEP, continued. **Initialize** is the same in all of these games: $(f, f^{-1}) \xleftarrow{\$} \mathcal{F}(k)$, $b \xleftarrow{\$} \{0,1\}$, **return** $inp \leftarrow f$. **Finalize** is also the same: **return** $out = b$.

R4: In game R4 the string GR^* is chosen but not referred to in responding to any oracle queries of the adversary. Thus R4 is a conservative replacement for R3, $p_3 = p_4$, and $\Pr[\text{R3 sets } bad\,] = \Pr[\text{R4 sets } bad\,]$. However, the bit b is not used in R4, and hence $p_4 = 1/2$. In summary

$$p_3 + \Pr[\text{R3 sets } bad\,] \;=\; p_4 + \Pr[\text{R4 sets } bad\,] \;=\; \frac{1}{2} + \Pr[\text{R4 sets } bad\,]\;.$$

Putting all this together we have

$$\frac{1}{2}\mathbf{Adv}^{\text{fg-cpa}}_{\mathsf{OAEP}^\rho[\mathcal{F}]}(A) - \frac{q_G}{2^\rho} - \frac{q_H}{2^{k-\rho}} \;\le\; \Pr[\text{R4 sets } bad\,]\;. \tag{4}$$

We proceed to upper bound the right-hand-side of the above. We have

$$\Pr[\text{R4 sets } bad\,] \;=\; \Pr[\text{R5 sets } bad\,] \;\le\; q_G/2^\rho + \Pr[\text{A0 sets } bad\,]\;.$$

Next we have a series of conservative changes, giving A0, A1, A2, A3, A4, A5 leading to

$$\Pr[\text{A0 sets } bad\,] \;=\; \Pr[\text{A5 sets } bad\,] \;\le\; \Pr[\text{A6 sets } bad\,] \;=\; \Pr[\text{A7 sets } bad\,]\;.$$

To conclude the proof we design I so that

$$\Pr[\text{A7 sets } bad\,] \;\le\; \mathbf{Adv}^{\text{owf}}_{\mathcal{F}}(I)\;. \tag{5}$$

On input f, Y^*, inverter I runs A on input public key f, responding to its oracle queries as follows.

On query $E(M_0, M_1)$ — Inverter I

000 **return** Y^*

On query $G(R)$

010 **if** $\exists\, S$ s.t. $f(S \,\|\, H[S] \oplus R) = Y^*$ **then**
    $bad \leftarrow$ true, $S^* \,\|\, T^* \leftarrow S \,\|\, H[S] \oplus R$

011 **return** $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

On query $H(S)$

020 **return** $H[S] \xleftarrow{\$} \{0,1\}^\rho$

When A halts, inverter I returns $S^* \,\|\, T^*$ if this has been defined. By comparison with A7 we see that (5) is true, completing the proof. ▌

Acknowledgments

We thank Tadayoshi Kohno for permission to use his observations about the standard proof of the PRP/PRF Switching Lemma noted in Section 2 and Appendix A. We thank Krzysztof Pietrzak for his insightful observation concerning a bug in an earlier draft of this paper, and Shai Halevi for likewise identifying the lemma as buggy a couple of days later. Bart Preneel was the first we heard to ask, back in 1994, if the m^2 term can be improved in the CBC MAC bound of $m^2q^2/2^n$.

Phil Rogaway was supported in part by NSF 0208842 and a gift from Intel Corp. Mihir Bellare was supported in part by NSF 0098123, ANR-0129617, NSF 0208842, and an IBM Faculty Partnership Development Award.

References

[Ad78] L. Adleman. Two theorems on random polynomial time. *FOCS 78.*

[BBKN01] M. Bellare, A. Boldyreva, L. Knudsen and C. Namprempre. On-line ciphers and the hash-CBC constructions. *Crypto '01.*

[BCP03] E. Bresson, O. Chevassut, and D. Pointcheval. Security proofs for an efficient password-based key exchange. *ACM CCS '03.*

[Be05] D. Bernstein. A short proof of the unpredictability of cipher block chaining. Manuscript, January 2005. Available on Bernstein's web page.

[BG89] M. Bellare and S. Goldwasser. New paradigms for digital signatures and message authentication based on non-interative zero knowledge proofs. *Crypto 89*, pp. 194–211, 1989.

[BK05] D. Boneh and J. Katz. Improved efficiency for CCA-secure cryptosystems built using identity-based encryption. *CT-RSA '05* (to appear).

[BKR94] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences* (JCSS), vol. 61, no. 3, pp. 362–399, 2000. Earlier version in *Crypto '94.*

[BKR98] M. Bellare, T. Krovetz, and P. Rogaway. Luby-Rackoff backwards: increasing security by making block ciphers non-invertible. *Eurocrypt '98.*

[Bo01] D. Boneh. Simplified OAEP for the RSA and Rabin functions. *Crypto '01.*

[BPR05] M. Bellare, K. Pietrzak, and P. Rogaway. Improved security analyses for CBC MACs. *Crypto 05.*

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. *ACM CCS '93.*

[BR94] M. Bellare and P. Rogaway. Optimal asymmetric encryption. *Eurocrypt '94.*

[BR00] J. Black and P. Rogaway. CBC MACs for arbitrary-length messages: the three-key constructions. *Crypto '00.*

[BR02] J. Black and P. Rogaway. A block-cipher mode of operation for parallelizable message authentication. *Eurocrypt '02.*

[BRS02] J. Black, P. Rogaway, and T. Shrimpton. Encryption-scheme security in the presence of key-dependent messages. *SAC 2002.*

[BRW04] M. Bellare, P. Rogaway, and D. Wagner. The EAX mode of operation (a two-pass authenticated encryption scheme). *FSE '04*

[CS02] R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public key encryption. *Eurocrypt '02.*

[CS03a] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logs. *Crypto '03.*

[CS03b] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. of Computing*, vol. 33, pp. 167–226, 2003.

[DFKY03] Y. Dodis, N. Fazio, A. Kiayias, and M. Yung. Scalable public-key tracing and revoking. *PODC '03.*

[DGHKR04] Y. Dodis, R. Gennaro, J. Håstad, H. Krawczyk, and T. Rabin. Randomness extraction and key derivation using the CBC, Cascade, and HMAC modes. *Crypto '04.*

[FOPS04] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. *J. of Cryptology*, vol. 17, no. 2, pp. 81–104, 2004.

[GMMV03] D. Galindo, S. Martin, P. Morillo, and J. Villar. Fujisaki-Okamoto IND-CCA hybrid encryption revisited. Cryptology ePrint archive report 2003/107, 2003.

[GS04] R. Gennaro and V. Shoup. A note on an encryption scheme of Kurosawa and Desmedt. Cryptology ePrint report 2004/194, 2004.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, 1984. Earlier version in *STOC '82*.

[HR03] S. Halevi and P. Rogaway. A tweakable enciphering mode. *Crypto '03*.

[HR04] S. Halevi and P. Rogaway. A parallelizable enciphering mode. *CT-RSA '04*.

[HWKS98] C. Hall, D. Wagner, J. Kelsey, and B. Schneier. Building PRFs from PRPs. Available on Wagner's web page. Earlier version in *Crypto '98*.

[JJV02] E. Jaulmes, A. Joux, and F. Valette. On the security of randomized CBC-MAC beyond the birthday paradox limit: a new construction. *FSE '02*.

[KD04] K. Kurosawa and Y. Desmedt. A new paradigm of hybrid encryption scheme. *Crypto '04*.

[KR96] J. Kilian and P. Rogaway. How to protect DES against exhaustive key search (an analysis of DESX). *Journal of Cryptology*, vol. 14, no. 1, pp. 17–35, 2001. Earlier version in *Crypto '96*.

[Ma02] U. Maurer. Indistinguishability of random systems. *Eurocrypt '02*.

[PP04] D. Phan and D. Pointcheval. OAEP 3-round: a generic and secure asymmetric encryption padding. *Asiacrypt '04*.

[PR00] E. Petrank and C. Rackoff. CBC MAC for real-time data sources. *J. Cryptology*, vol. 13, no. 3, pp. 315–338, 2000.

[R02] P. Rogaway. Authenticated-encryption with associated-data. *ACM CCS '02*.

[R04] P. Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. *Asiacrypt '04*.

[RBB01] P. Rogaway, M. Bellare, and J. Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Transactions on Information and System Security*, vol. 6, no. 3, pp. 365–403, 2003. Earlier version in *ACM CCS '01*.

[Sh49] C. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.

[Sh00] V. Shoup. Using hash function as a hedge against chosen ciphertext attack. *Eurocrypt '00*.

[Sh01a] V. Shoup. OAEP reconsidered. *J. of Cryptology*, vol. 15, no. 4, pp. 223–249, 2002. Earlier version in *Crypto '01*.

[Sh01b] V. Shoup. A proposal for an ISO standard for public key encryption. Cryptology ePrint report 2001/112, 2001.

[Sh04] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint report 2004/332, November 30, 2004.

[SS00] T. Schweinberger and V. Shoup. ACE: the advanced cryptographic engine. Cryptology ePrint report 2000/022, 2000.

[Va01] S. Vaudenay. Decorrelation over infinite domains: the encrypted CBC-MAC case. *Communications in Information and Systems* (CIS), vol. 1, pp. 75–85, 2001.

A Fixing the PRP/PRF Switching Lemma Without Games

Let adversary A and other notation be as in Section 2, where we showed by example that if the number of oracle queries made by A depends on the answers it receives in response to previous queries, then (1) may not hold. Here we show that if the number of oracle queries made by A is always *exactly* q—meaning the number of queries is this value regardless of A's coins and the answers to the oracle queries—then (1) is true.

Note that given any adversary A_1 making at most q queries, it is easy to modify it to an A_2 that has the same advantage as A_1 but makes exactly q oracle queries. (A_2 will run A_1 until it halts, counting the number of oracle queries the latter makes. Calling this number q_1, it now makes some $q - q_1$ oracle queries, whose answers it ignores, outputting exactly what A_1 outputs.) In other words, if an adversary is assumed to make at most q queries, one can assume wlog that the number of queries is exactly q. This means that one can in fact obtain a correct proof of the PRP/PRF Switching Lemma based on (1). The bug we highlighted in Section 2 thus amounts to having claimed (1) for all A making at most q queries rather than those making exactly q queries.

Let us now show that if the number of oracle queries made by A is always exactly q then (1) is true. Since A is computationally unbounded, we may assume wlog that A is deterministic. We also assume it never repeats an oracle query. Let $V = (\{0,1\}^n)^q$ and for a q-vector $a \in V$ let $a[i] \in \{0,1\}^n$ denote the i-th coordinate of a, $1 \le i \le q$. We can regard A as a function $f\colon V \to \{0,1\}$ that given a q-vector a of replies to its oracle queries returns a bit $f(a)$. Let $\mathbf{a}$ denote the random variable that takes value the q-vector of replies returned by the oracle to the queries made by A. Also let

$$\begin{aligned} \mathrm{dist} &= \{\, a \in V \,:\, a[1], \ldots, a[n] \text{ are distinct} \,\} \\ \mathrm{one} &= \{\, a \in V \,:\, f(a) = 1 \,\}\,. \end{aligned}$$

Let $\Pr_{\mathrm{rand}}[\,\cdot\,]$ denote the probability in the experiment where $\rho \stackrel{\$}{\leftarrow} \mathrm{Rand}(n)$. Then

$$\begin{aligned} \Pr[\,A^\rho \Rightarrow 1 \mid \mathsf{Dist}\,] &= \Pr_{\mathrm{rand}}[\,f(\mathbf{a}) = 1 \mid \mathbf{a} \in \mathrm{dist}\,] \;=\; \frac{\Pr_{\mathrm{rand}}[\,f(\mathbf{a}) = 1 \wedge \mathbf{a} \in \mathrm{dist}\,]}{\Pr_{\mathrm{rand}}[\,\mathbf{a} \in \mathrm{dist}\,]} \\ &= \frac{\sum_{a \in \mathrm{dist} \cap \mathrm{one}} \Pr_{\mathrm{rand}}[\,\mathbf{a} = a\,]}{\sum_{a \in \mathrm{dist}} \Pr_{\mathrm{rand}}[\,\mathbf{a} = a\,]} \;=\; \frac{\sum_{a \in \mathrm{dist} \cap \mathrm{one}} 2^{-nq}}{\sum_{a \in \mathrm{dist}} 2^{-nq}} \;=\; \frac{|\mathrm{dist} \cap \mathrm{one}|}{|\mathrm{dist}|}\,. \end{aligned}$$

On the other hand let $\Pr_{\mathrm{perm}}[\,\cdot\,]$ denote the probability in the experiment where $\pi \stackrel{\$}{\leftarrow} \mathrm{Perm}(n)$. Then

$$\begin{aligned} \Pr[\,A^\pi \Rightarrow 1\,] &= \Pr_{\mathrm{perm}}[\,f(\mathbf{a}) = 1\,] \;=\; \sum_{a \in \mathrm{dist} \cap \mathrm{one}} \Pr_{\mathrm{perm}}[\,f(\mathbf{a}) = a\,] \\ &= \sum_{a \in \mathrm{dist} \cap \mathrm{one}} \prod_{i=0}^{q-1} \frac{1}{2^n - i} \;=\; \sum_{a \in \mathrm{dist} \cap \mathrm{one}} \frac{1}{|\mathrm{dist}|} \;=\; \frac{|\mathrm{dist} \cap \mathrm{one}|}{|\mathrm{dist}|}\,. \end{aligned}$$