Code-Based Game-Playing Proofs and the Security of Triple Encryption*

Mihir Bellare † Phillip Rogaway ‡

February 27, 2006

(Draft 2.2)

Abstract

The game-playing technique is a powerful tool for analyzing cryptographic constructions. We illustrate this by using games as the central tool for proving security of three-key triple-encryption, a long-standing open problem. Our result, which is in the ideal-cipher model, demonstrates that for DES parameters (56-bit keys and 64-bit plaintexts) an adversary's maximal advantage is small until it asks about 2^{78} queries. Beyond this application, we develop the foundations for game playing, formalizing a general framework for game-playing proofs and discussing techniques used within such proofs. To further exercise the game-playing framework we show how to use games to get simple proofs for the PRP/PRF Switching Lemma, the security of the basic CBC MAC, and the chosen-plaintext-attack security of OAEP.

Keywords: Cryptographic analysis techniques, games, provable security, triple encryption.

* Earlier versions of this paper were entitled "The Game-Playing Technique" and "The Game-Playing Technique and its Application to Triple Encryption."

† 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 **3**

2 The Game-Playing Framework **7**

3 The Security of Three-Key Triple-Encryption **11**

Acknowledgments **17**

References **17**

A The PRP/PRF Switching Lemma **19**

B Fixing the PRP/PRF Switching Lemma Without Games **22**

C An Example Programming Language for Games **23**

D Game-Rewriting Techniques **26**
D.1 Game chains 26
D.2 Basic techniques 26
D.3 Coin fixing 28
D.4 Lazy sampling 29

E Proofs for Triple-Encryption **31**
E.1 Proof of Lemma 5 31
E.2 Proof of Lemma 6 35
E.3 Proof of Lemma 7 37

F Elementary Proof for the CBC MAC **37**

G A Game-Based Proof for OAEP **41**

1 Introduction

Foundations and applications. The game-playing technique has become a popular approach for doing proofs in cryptography. We will explain the method shortly. In this paper we take the initial steps in developing a theory of game-playing proofs. We believe that such a theory will prove beneficial for our field. Then we demonstrate the utility of game-playing by providing examples of the technique, the most striking of which is the first proof that triple-encryption (using three independent keys) is far more secure than single or double encryption. The result, which is in the ideal-cipher model, is the first to prove that the cascade of blockciphers can markedly improve security. Other examples that we work out with games include the PRP/PRF Switching Lemma, the PRF-security of the CBC MAC, and the chosen-plaintext-attack security for OAEP.

Why games? There are several reasons why we take a fresh look at the game-playing technique. First, the method is widely applicable, easily employed, and provides a unifying structure for diverse proofs. Games can be used in the standard model, the random-oracle model, the ideal-blockcipher model, and more; in the symmetric setting, the public-key setting, and further trust models; for simple schemes (eg, justifying the Carter-Wegman MAC) and complex protocols (eg, proving the correctness of a key-distribution protocol).

Second, the game-playing technique can lead to significant new results. We demonstrate this by developing a game-based proof for three-key triple encryption. Proving security for triple encryption is a well-known problem, but technical difficulties have always frustrated attempts at a solution.

Finally, we believe that the game-playing approach can lead to proofs that are less error-prone and more easily verifiable, even mechanically verifiable, than proofs grounded solely in more conventional probabilistic language. In our opinion, many proofs in cryptography have become essentially unverifiable. Our field may be approaching a crisis of rigor. While game-playing is not a panacea to this problem (which has at its core a significant cultural element), game-playing may play a role in the answer.

The cascade construction. The security of the cascade construction, where two or more independently keyed blockciphers are composed with one another, is a nearly 30-year-old problem [19, 34]. Even and Goldreich refer to it as a "critical question" in cryptography [21, p. 109]. They showed that the cascade of ciphers is at least as strong as the *weakest* cipher in the chain [21], while Maurer and Massey showed that, in a weaker attack model, it is at least as strong as the *first* cipher in the chain. We know that double encryption (the two-stage cascade) can't strengthen security much, due to the classic meet-in-the-middle attack [19], although Aiello, Bellare, Di Creczenzo, and Venkatesan show that the "shape" of the security curve is slightly improved [3]. This means that triple encryption (the three-stage cascade) is the shortest potentially "good" cascade. And, indeed, triple DES is the cascade that is widely standardized and used [36].

Triple encryption "works." In this paper we prove that triple-encryption vastly improves security over single or double encryption. Given a blockcipher $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ with inverse D we consider $\mathsf{Cascade}_E^{\mathrm{eee}}(K_0K_1K_2,\ X) = E_{K_2}(E_{K_1}(E_{K_0}(X)))$ and $\mathsf{Cascade}_E^{\mathrm{ede}}(K_0K_1K_2,\ X) = E_{K_2}(D_{K_1}(E_{K_0}(X)))$. Our results are the same for both versions of triple encryption. Following [22, 30, 42], we model E as a family of random permutations, one for each key, and we provide the adversary with oracle access to the blockcipher $E(\cdot,\cdot)$ and its inverse $E^{-1}(\cdot,\cdot)$ Given such oracles, the adversary is asked to distinguish between (a) $\mathsf{Cascade}_E^{\mathrm{eee}}(K_0K_1K_2,\ \cdot\,)$ and its inverse, for a random key $K_0K_1K_2$, and (b) a random permutation on n bits and its inverse. We show that the adversary's advantage in making this determination, $\mathbf{Adv}_{k,n}^{\mathrm{eee}}(q)$, remains small until it asks about $q = 2^{k+0.5\min\{k,n\}}$ queries (the actual expression is more complex). The bound we get is

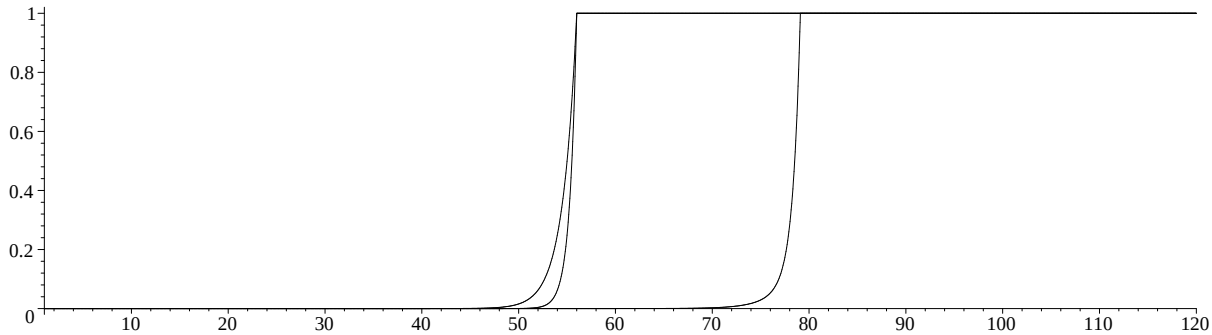

Figure 1: Upper bound on adversarial advantage (proven security) verses $\log_2 q$ (where q=number of queries) for the cascade construction, assuming key length $k = 56$ and block length $n = 64$. Single encryption is the leftmost curve, double encryption is the middle curve [3], and triple encryption in the rightmost curve, as given by Theorem 3.

plotted as the rightmost curve of Figure 1 for DES parameters $k = 56$ and $n = 64$. In this case an adversary must ask more than $2^{78.5}$ queries to get advantage 0.5. Also plotted are the security curves for single and double encryption, where the adversary must ask 2^{55} and $2^{55.5}$ queries to get advantage 0.5. For a blockcipher with $k = n = 64$, the adversary must ask more than 2^{89} queries to get advantage 0.5. As there are matching attacks and security bounds for single and double encryption, our result proves that, in the ideal-cipher model, triple encryption is more secure than single or double encryption.

Our proof for triple-encryption uses game-playing in an integral way, first to recast the advantage we wish to bound to a simpler game, and later to analyze that game by investigating another one. Ultimately one is left with a game where conventional probabilistic reasoning (a special-purpose occupancy bound) can be applied. Game playing does not replace conventional probabilistic reasoning; it supplements it.

As for the cascade of $\ell \geq 4$ blockciphers, the maximal advantage in our attack model is no worse than it is for triple encryption, so our result proves that cascade "works" (provides improved security over single and double encryption) for all $\ell \geq 3$. It is an open question if security actually increases with increasing ℓ.

What is the game-playing technique? One complication in any discussion about game-playing proofs is that the term means different things to different people. To some, a game-playing proof in cryptography is any proof where one conceptualizes the adversary's interaction with its environment as a kind of game, the proof proceeding by stepwise refinement to that game. Viewed in this way, game-playing proofs have their origin in the earliest hybrid arguments, which began with Goldwasser and Micali [24] and Yao [48]. Bellare and Goldwasser provide an early example of an intricate proof of this flavor, demonstrating the security of a signature scheme that uses multiple cryptographic primitives [4]. In recent years Shoup has come to use such game-based proofs extensively [1, 16–18, 41, 43, 44, 46], as have other authors.

We believe that game-playing proofs can be most effectively studied and systematized by imposing some discipline on the process and, in particular, regarding games as code. This viewpoint begins in 1994 with Kilian and Rogaway [30]. Code-based game-playing soon became the favored technique of Rogaway, who, along with coauthors, used it in many subsequent papers [6, 10, 12–14, 26, 27, 38–40]. Code-based game-playing typically works like this. Suppose you wish to upper bound the advantage of an adversary A in attacking some cryptographic construction. This is the difference between the probability that A outputs 1 in each of two different "worlds." First, write some code—a game—that captures the behavior of world 0. The code initializes variables, interacts with the adversary, and then runs some more. Then write another piece of code—a second

game—that captures the behavior of world 1. Arrange that games 0 and 1 are *syntactically identical* apart from statements that follow the setting of a flag *bad* to true. Now invoke the "fundamental lemma of game playing" (which we formalize and prove in this paper) to say that, in this setup, the adversary's advantage is upper-bounded by the probability that *bad* gets set (in either game). Next, 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. In this way you produce a *chain* of games, ending at some *terminal* game. Bound the probability that *bad* gets set in the terminal game using conventional (not game-based) techniques.

Formalizing the foundations. In our treatment games are code and not abstract environments; as we develop it, game-playing centers around making disciplined transformations to code. The code can be written in pseudocode or a formalized programming language $\mathcal{L}$. We will describe a sample programming language for writing games, using it (apart from some "syntactic sugar") in our examples.

Under our framework, a game G is a program that is *run* with an adversary A, which is also a program (look ahead to Figure 2). The adversary calls out to procedures, called *oracles*, specified by the game. We define what it means for two games to be identical-until-*bad*, where *bad* is a boolean variable in the game. This is a syntactical condition. We prove that if two games are identical-until-*bad* then the difference in the probabilities of a given outcome is bounded by the probability that *bad* gets set (in either game). This result, the *fundamental lemma of game-playing*, is the central tool justifying the technique.

We go on to give describe some general lemmas and techniques for analyzing the probability that *bad* gets set. Principle among these is a lemma that lets you change anything you want after the flag *bad* gets set. Other techniques speak to eliminating adaptivity, de-randomization, making "lazy" probabilistic choices, resampling, using "poisoned" points, and so forth.

Further applications. We illustrate the applicability of games in a wide variety of settings, providing results in the standard model, the random-oracle model [9], and the ideal-cipher model, and in both the symmetric and asymmetric settings.

We begin with a motivating observation, due to Tadayoshi Kohno, that the standard proof of the PRP/PRF Switching Lemma [5, 28] contains an error in reasoning about conditional probabilities. (The lemma says that an adversary that asks 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.) 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. This motivates our use of games as an alternative. We re-prove the Switching Lemma with a simple game-based proof.

Next we look at the 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 Bellare, Kilian, and Rogaway [5] says that $\mathbf{Adv}^{\mathrm{cbc}}_{n,m}(q) \leq 2m^2q^2/2^n$. But the proof [5] is very complex and does not directly capture the intuition behind the security of the scheme. Here we use games to give an elementary proof for an $m^2q^2/2^n$ bound, the proof directly capturing, in our view, the underlying intuition.

Finally, we give an example of using games in the public-key, random-oracle setting by proving that OAEP [8] with any trapdoor permutation is an IND-CPA secure encryption scheme. The original proof of this result [8] 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.

Further related work. The best-known attack on three-key triple-encryption is due to Lucks [31]. He does not work out an explicit lower bound for $\mathbf{Adv}^{\mathrm{eee}}_{k,n}(q)$ but in the case of triple-DES the advantage becomes large by $q = 2^{90}$ queries. We prove security to about 2^{78} queries, so there is no contradiction.

The DESX construction has been proven secure up to about $2^{k+n-\lg m}$ blockcipher queries for key length k, block length n, and m queries to the construction [30]. This is stronger than our bound for triple encryption when the adversary can obtain few points encrypted by the construction, a weaker bound otherwise.

Double encryption and two-key triple encryption were analyzed Aiello, Bellare, Di Crescenzo, and Venkatesan [3], where it is shown that the meet-in-the-middle attack is optimal (in the ideal-cipher model). Their result is the first to show that the cascade construction buys you *something* (half a bit of security for advantage 0.5), but what it buys is inherently limited, because of the meet-in-the-middle attack. We comment that games provide an avenue to a much simpler proof of their result.

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 [32]. Vaudenay has likewise developed a framework for the analysis of blockciphers and blockcipher-based constructions, and has applied it to the encrypted CBC MAC [47]. Neither Maurer's nor Vaudenay's approach is geared towards making stepwise, code-directed refinements for computing a probability.

A more limited and less formal version of the Fundamental Lemma appears in [6, Lemma 7.1]. A lemma by Shoup [43, Lemma 1] functions in a similar way for games that are not necessarily code-based.

Shoup has independently and contemporaneously prepared a manuscript on game playing [45]. It is more pedagogically-oriented than this paper. Shoup does not try to develop a theory for game playing beyond [43, Lemma 1]. 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 [11], 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.

In work derivative of an earlier version of this paper, Bellare, Pietrzak, and Rogaway [7] improve the bound $\mathbf{Adv}^{\mathrm{cbc}}_{n,m}(q) \le m^2q^2/2^n$ of [5, 32] to about $mq^2/2^n$, and consider generalizations to this claim as well. The proof of [7] springs from games, refining the game used here for the CBC MAC and then analyzing it using techniques derivative of [20].

Following the web distribution of this paper, Halevi argues for the creation of an automated tool to help write and verify game-based proofs [25]. We agree. The possibility for such tools has always been one of our motivations, and one of the reasons why we focused on code-based games.

Why should game-playing work? It is fair to ask if anything is actually "going on" when using games—couldn't you recast everything into more conventional probabilistic language and drop all that ugly code? 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. Part of the power of the approach 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.

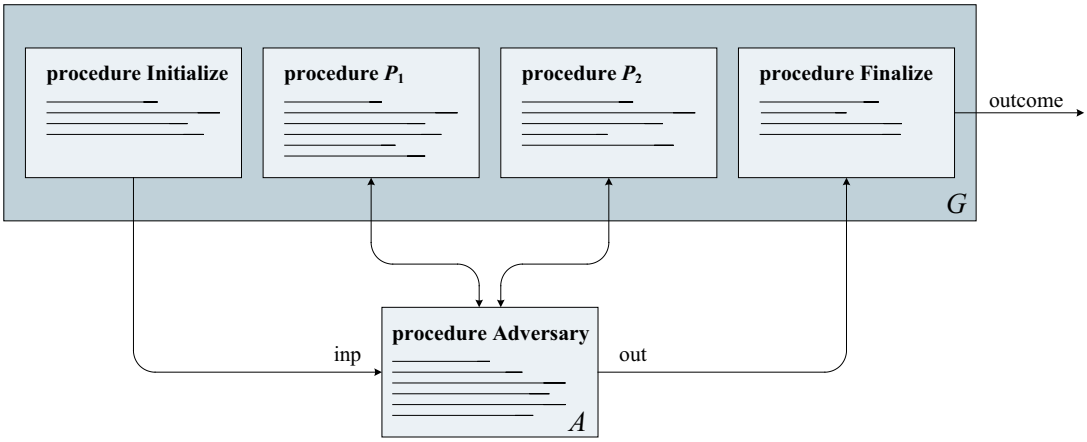

Figure 2: Running a game G with an adversary A. The game is the code at the top, the adversary is the code at the bottom. The adversary interacts with the game by calling the oracles provided (two of which are shown).

2 The Game-Playing Framework

Programming language. A game is a program, viewed as a collection of procedures, and the adversary is likewise a program, but one consisting of a single procedure. We will, for the moment, regard games and adversaries as being written in pseudocode. Below we outline some elements of our pseudocode. We find that a pseudocode-based descriptive language is adequate to make game-playing unambiguous and productive. To make a truly rigorous theory one should, in fact, fully specify the underlying programming language. In Appendix C we provide an example language $\mathcal{L}$ suitable for describing games and adversaries (we specify the syntax but dispense with the operational semantics, which should be clear). The games of this paper conform to the syntax of $\mathcal{L}$ apart from some minor matters.

Our programming language is strongly typed, with the type of each variable apparent from its usage (we dispense with explicit declarations). We will have variables of type integer, boolean, string, set, and array. A set is a finite set of strings and an array is an associative array, one taking on values of strings. The semantics of a boolean variable, which we will also call a *flag*, is that once true it stays true.

We allow conventional statements like *if* statements, *for* statements, and *assignment* statements. There is also a *random-assignment* statement, which is the only source of randomness in programs. Such a statement has the form $s \xleftarrow{\$} S$ where S is a finite set. The result is to uniformly select a random element from the set S and assign it to s. If $S = \emptyset$ or $S = \mathsf{undefined}$ (we regard $\mathsf{undefined}$ as a possible value for a variable) then the result of the random-assignment statement is to set s to $\mathsf{undefined}$. A comma or newline serves as a statement separator and indentation is used to indicate grouping.

A game has three kinds of procedures: an initialization procedure (Initialize), a finalization procedure (Finalize), and named oracles (each one a procedures). The adversary can make calls to the oracles, passing in values from some finite domain associated to each oracle. The initialization or finalization procedures may be absent, and often are, and there may be any number of oracles, including none. All variables in a game are global variables and are not visible to the adversary's code. All variables in adversary code are local.

Running a game. We can *run* a game G with an adversary A. To begin, variables are given initial values. Integer variables are initialized to 0; boolean variables are initialized to false; string

variables are initialized to the empty string ε; set variables are initialized to the empty set $\emptyset$; and array variables hold the value undefined at every point. These conventions facilitate omitting explicit initialization code in many cases.

The Initialize procedure is the first to execute, possibly producing an output *inp*. This is provided as input to the procedure Adversary, denoted A, which now runs. The adversary code can make oracle queries via statements of the form $y \leftarrow P(\cdots)$ for any oracle P that has been defined in the game. The result is to assign to y the value returned by the procedure call. We assume that the game and adversary match syntactically, meaning that all the oracle calls made by the adversary are to oracles specified in the game, and with arguments that match in type or quantity. The semantics of a call is call-by-value; the only way for an oracle to return a value to the adversary is via a return statement. When adversary A halts, possibly with some output *out*, we call the Finalize procedure, providing it any output produced by A. The Finalize procedure returns a string that is the outcome of the game. If we omit specifying Initialize or Finalize it means that these procedures do nothing: they compute the identity function.

Termination and finite randomness. We assume that an underlying execution model provides a notion for the number of steps (the running time) of a program. We require that both the adversary and the game always terminate in finite time. By this we mean that, for any adversary A there must exist an integer T such that A always halts within T steps (regardless of the random choices A makes and the answers it receives to its oracle queries). Similarly, for any game G there must exist an integer T such that G always halts within T steps (regardless of the random choices made, Initialize halts within T steps, and, regardless of the inputs they are provided, Finalize and the oracles halt within T steps). The finite-termination requirement is guaranteed automatically by our sample programming language $\mathcal{L}$.

Since the adversary and game terminate in finite time, there must be an integer T such that they each execute at most T random-assignment statements, and there must be an integer B such that the size of the set S in any random-assignment statement $s \xleftarrow{\$} S$ executed by the adversary or the game is at most B. Taken together, this means that the execution of G with A uses *finite randomness*, meaning G and A are underlain by a finite sample space Ω. Thus probabilities are well-defined and henceforth we can talk about the probabilities of various events in the execution.

Adversary and game outputs. We associate two outputs to the process of running a game with an adversary. The first, called the adversary output, is the value *out* returned by A after it has completed its interaction with the oracles provided by the game. The second, called the game output, is the value *outcome* returned by the Finalize procedure. Often the two outputs are the same, because the Finalize procedure is not specified (whence we define it to do nothing but pass on its input as its output).

The adversary and game outputs can be regarded as random variables. We write $\Pr[A^G \Rightarrow 1]$ for the probability that the adversary output is 1 when we run game G with adversary A, and $\Pr[G^A \Rightarrow 1]$ for the probability that the game output is 1 when we run game G with adversary A.

Advantages. If G and H are games and A is an adversary, let

$$\mathbf{Adv}(A^G, A^H) \;=\; \Pr[A^G \Rightarrow 1] - \Pr[A^H \Rightarrow 1] \quad \text{and} \quad \mathbf{Adv}(G^A, H^A) \;=\; \Pr[G^A \Rightarrow 1] - \Pr[H^A \Rightarrow 1]\ .$$

These represent the advantage of the adversary in distinguishing the games, the first measured via adversary output and the second via game output. We refer to the first as the *adversarial advantage* and the second as the *game advantage*. We say that G, H are *adversarially indistinguishable* if for any adversary A it is the case that $\mathbf{Adv}(A^G, A^H) = 0$, and *equivalent* if, for any adversary A it is

the case that $\mathbf{Adv}(G^A, H^A) = 0$. We will often use the fact that

$$\mathbf{Adv}(A^G, A^I) = \mathbf{Adv}(A^G, A^H) + \mathbf{Adv}(A^H, A^I) \quad (1)$$

$$\mathbf{Adv}(G^A, I^A) = \mathbf{Adv}(G^A, H^A) + \mathbf{Adv}(H^A, I^A) \quad (2)$$

for any games G, H, I and any adversary A. These will be referred to as the *triangle equalities*.

We will usually be interested in adversarial advantage (eg, this is the case in the game-playing proof of the PRP/PRF Switching Lemma). Game advantage is useful when we are interested in how the adversary's output relates to some game variable such as a hidden bit chosen by the game (this happens in our proof of the security of OAEP).

IDENTICAL-UNTIL-*bad* GAMES. We are interested in programs that are syntactically identical except for statements that follow the setting of a flag *bad* to true. Somewhat more precisely, let G and H be programs and let *bad* be a flag that occurs in both of them. Then we say that G and H are *identical-until-bad* if their code is the same except that there might be places where G has a statement $bad \leftarrow \mathsf{true},\ S$ while game H has a corresponding statement $bad \leftarrow \mathsf{true},\ T$ for some T that is different from S. As an example, in the games S_0 and S_1 from Figure 6, the former has the empty statement following $bad \leftarrow \mathsf{true}$ while in S_1 we have $Y \xleftarrow{\$} \overline{\text{image}}(\pi)$ following $bad \leftarrow \mathsf{true}$. Since this is the only difference in the programs, the games are identical-until-*bad*. One could also say that G and H are are identical-until-*bad* if one has the statement if *bad* then S where the other has the empty statement, for this can be rewritten in the form above.

A fully formal definition of identical-until-*bad* requires one to pin down the programming language and talk about the parse trees of programs in the language. We establish the needed language in Appendix C but, in fact, such formality isn't needed in applications: for any two games one writes down, whether or not they are identical-until-*bad* is obvious. We emphasize that identical-until-*bad* is a purely "syntactic" requirement.

We write $\Pr[A^G \text{ sets } bad\,]$ or $\Pr[G^A \text{ sets } bad\,]$ to refer to the probability that the flag *bad* is true at the end of the execution of the adversary A with game G, namely at the point when the Finalize procedure terminates. It is easy to see that, for any flag *bad*, identical-until-*bad* is an equivalence relation on games. When we say that a sequence of games $G_1, G_2, \ldots$ are identical-until-*bad*, we mean that each pair of games in the sequence are identical-until-*bad*.

THE FUNDAMENTAL LEMMA. The fundamental lemma says that the advantage that an adversary can obtain in distinguishing a pair of identical-until-*bad* games is at most the probability that its execution sets *bad* in one of the games (either game will do).

Lemma 1 [Fundamental lemma of game-playing] *Let G and H be identical-until-bad games and let A be an adversary. Then*

$$\mathbf{Adv}(A^G, A^H) \le \Pr[A^G \text{ sets } bad\,] \quad \textit{and} \quad (3)$$

$$\mathbf{Adv}(G^A, H^A) \le \Pr[G^A \text{ sets } bad\,]\,. \quad (4)$$

More generally, let G, H, I be identical-until-bad games. Then

$$\left|\mathbf{Adv}(A^G, A^H)\right| \le \Pr[A^I \text{ sets } bad\,] \quad \textit{and} \quad (5)$$

$$\left|\mathbf{Adv}(G^A, H^A)\right| \le \Pr[I^A \text{ sets } bad\,]\,. \quad (6)$$

Proof: Statement (3) follows from (4) by applying the latter to games G', H' formed by replacing the Finalize procedure of games G, H, respectively, with the trivial one that simply returns the

adversary output. Similarly, (5) follows from (6). We will now prove (4) and then derive (6) from it.

We have required that the adversary and game always terminate in finite time, and also that there is an integer that bounds the size of any set S in any random-assignment statement $s \xleftarrow{\$} S$ executed by the adversary or game. This means that there exists an integer b such that the execution of G with A and the execution of H with A perform no more than b random-assignment statements, each of these sampling from a set of size at most b. Let $C = \mathrm{Coins}(A, G, H) = [1\,..\,b!]^b$ be the set of b-tuples of numbers, each number between 1 and $b!$. We call C the *coins* for (A, G, H). For $c = (c_1, \ldots, c_b) \in C$, the *execution of G with A on coins c* is defined as follows: on the ith random-assignment statement, call it $X \xleftarrow{\$} S$, if $S = \{a_0, \ldots, a_{m-1}\}$ is nonempty and $a_0 < a_1 < \cdots < a_{m-1}$ in lexicographic order then let X take on the value $a_{c_i \bmod m}$. If $S = \emptyset$ then let X take on the value undefined. This way to perform random-assignment statements is done regardless of whether it is A or one of the procedures from G that is is performing the random-assignment statement. Notice that m will divide $b!$ and so if c is chosen at random from C then the mechanism above will return a point X drawn uniformly from S, and also the return values for each random-assignment statement are independent. For $c \in C$ we let $G^A(c)$ denote the output of G when G is executed with A on coins c. We define the execution of H with A on coins $c \in C$, and $H^A(c)$, similarly.

Let $CG_{\text{one}} = \{c \in C : G^A(c) \Rightarrow 1\}$ be the set of coins $c \in C$ such that G outputs 1 when executed with A on coins c. Partition CG_{one} into $CG_{\text{one}}^{\text{bad}}$ and $CG_{\text{one}}^{\text{good}}$, where $CG_{\text{one}}^{\text{bad}}$ is the set of all $c \in CG_{\text{one}}$ such that the execution of G with A on coins c sets *bad* and $CG_{\text{one}}^{\text{good}} = CG_{\text{one}} \setminus CG_{\text{one}}^{\text{bad}}$. Similarly define CH_{one}, $CH_{\text{one}}^{\text{bad}}$ and $CH_{\text{one}}^{\text{good}}$. Observe that because games G and H are identical-until-*bad*, an element $c \in C$ is in $CG_{\text{one}}^{\text{good}}$ iff it is in $CH_{\text{one}}^{\text{good}}$. Thus these sets are equal and in particular have the same size. Now we have

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

This completes the proof of (4). Now, if G, H are identical-until-*bad* then (4) tells us that

$$\mathbf{Adv}(G^A, H^A) \le \Pr[G^A \text{ sets } bad\,] \quad \text{and} \quad \mathbf{Adv}(H^A, G^A) \le \Pr[H^A \text{ sets } bad\,]\,.$$

However, if G, H, I are all identical-until-*bad*, then Proposition 2 says that

$$\Pr[G^A \text{ sets } bad\,] = \Pr[H^A \text{ sets } bad\,] = \Pr[I^A \text{ sets } bad\,]\,.$$

Thus we have established (6). ▮

We have used finite randomness in our proof of the Fundamental Lemma, but we comment that this is more for simplicity than necessity: probabilities over the execution of G with A can be defined quite generally, even when the underlying sample space is infinite, and the Fundamental Lemma can still be proved. But we have never encountered any situation where such an extension is useful.

After bad is set, nothing matters. One of the most common manipulations of games along a game chain is to change what happens after *bad* gets set to true. Often one expunges code that follows the setting of *bad*, as we did in the PRP/PRF Switching Lemma,but it is also fine to insert alternative code. Any modification following the setting of *bad* leaves unchanged the probability of setting *bad*, as the following result shows.

Proposition 2 [After *bad* is set, nothing matters] *Let G and H be identical-until-bad games. Let A be an adversary. Then* $\Pr[G^A \text{ sets } bad\,] = \Pr[H^A \text{ sets } bad\,]$.

Proof: Using the definition from the proof of Lemma 1, fix coins $C = \mathrm{Coins}(A, G, H)$ 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*, 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 $\Pr[G^A \text{ sets } bad\,] = \Pr[H^A \text{ sets } bad\,]$. ▌

Besides the lemma above, many other ways to manipulate games are illustrated by our examples and our discussion in Section D.

Game inputs. The setting discussed above can be extended to allow a game to take an input parameter: the Initialize procedure would take an optional input that is a string parameter *input*. The adversary and game outputs will now be denoted $\Pr[A^G(input) \Rightarrow 1]$ and $\Pr[G^A(input) \Rightarrow 1]$ respectively. Similarly, the advantages become $\mathbf{Adv}(A^G(input), A^H(input))$ and $\mathbf{Adv}(G^A(input), H^A(input))$, these being defined in the obvious ways. The definition of identical-until-*bad* obviously extends to games with inputs, as does the Fundamental Lemma.

We can imagine that it might be convenient for games to have inputs, for example in the asymptotic setting where *input* might be the security parameter, but our experience has been that it is not really necessary. Rather than giving a game an input *input*, we can usually imagine a family of games, one for each value of *input*, and reason about these; since the games are involved only in the analysis, this usually suffices. Accordingly our treatment of games omits explicit game inputs.

3 The Security of Three-Key Triple-Encryption

Triple encryption goes back to the early attempts to strengthen DES against key-search attacks [19]. We now show that the method increases security, in the ideal-cipher model, resolving a long-standing open problem.

Definitions. Let $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be a blockcipher with key length k and block length n. For $K \in \{0,1\}^k$ and $X \in \{0,1\}^n$ let $E_K(X) = E(K, X)$. Let $E^{-1}\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$ be the blockcipher that is the inverse of E. We also denote it by D. We associate to E two blockciphers formed by composition. Denoted $\mathsf{Cascade}_E^{\mathrm{eee}}, \mathsf{Cascade}_E^{\mathrm{ede}}\colon \{0,1\}^{3k} \times \{0,1\}^n \to \{0,1\}^n$, they are defined as

$$\begin{aligned} \mathsf{Cascade}_E^{\mathrm{eee}}(K_0K_1K_2,\ X) &= E_{K_2}(E_{K_1}(E_{K_0}(X))) \quad \text{and} \\ \mathsf{Cascade}_E^{\mathrm{ede}}(K_0K_1K_2,\ X) &= E_{K_2}(D_{K_1}(E_{K_0}(X))) \end{aligned}$$

for all $K_0, K_1, K_2 \in \{0,1\}^k$ and $X \in \{0,1\}^n$. These blockciphers have key length $3k$ and block length n and are sometimes referred to as the *three-key* forms of triple encryption. We will call the two methods EEE and EDE, respectively. There is also a *two-key* variant of triple encryption, obtained by setting $K_0 = K_2$, but we do not investigate it since the method admits comparatively efficient attacks [34].

We will be working in the ideal-blockcipher model, as in works like [3, 22, 30]. Let $\mathrm{Bloc}(k, n)$ be the set of all blockciphers $E\colon \{0,1\}^k \times \{0,1\}^n \to \{0,1\}^n$. Thus $E \xleftarrow{\$} \mathrm{Bloc}(k, n)$ means that $E_K\colon \{0,1\}^n \to \{0,1\}^n$ is a random permutation on n-bit strings for each $K \in \{0,1\}^k$. We consider an adversary A that can make four types of oracle queries: $\boldsymbol{T}(X)$, $\boldsymbol{T^{-1}}(Y)$, $\boldsymbol{E}(K, X)$, and

$\boldsymbol{E}^{-1}(K,Y)$, where $X,Y \in \{0,1\}^n$ and $K \in \{0,1\}^k$. (As for our syntax, $\boldsymbol{T}$, $\boldsymbol{T}^{-1}$, $\boldsymbol{E}$, $\boldsymbol{E}^{-1}$ are formal symbols, not specific functions.) The *advantage of A against EEE* and the *maximal advantage against EEE obtainable using q queries* are defined as

$$\mathbf{Adv}^{\mathrm{eee}}_{k,n}(A) \;=\; \mathbf{Adv}(A^{C_0}, A^{R_0}) \quad \text{and} \quad \mathbf{Adv}^{\mathrm{eee}}_{k,n}(q) \;=\; \max_{A}\left\{\, \mathbf{Adv}^{\mathrm{eee}}_{k,n}(A) \,\right\}$$

where the games C_0, R_0 are shown in Figure 3 and the maximum is over all adversaries A that make at most q oracle queries (that is, a total of q across all oracles). The advantage of A measures its ability to tell whether $\boldsymbol{T}(\cdot)$ is a random permutation or is $\mathsf{Cascade}^{\mathrm{eee}}_{E}(K_0K_1K_2,\ \cdot\,)$ for $K_0K_1K_2 \xleftarrow{\$} \{0,1\}^{3k}$, when $\boldsymbol{E}$ realizes a random blockcipher $E \xleftarrow{\$} \mathrm{Bloc}(k,n)$ and $\boldsymbol{T}^{-1}, \boldsymbol{E}^{-1}$ realize inverses of $\boldsymbol{T}, \boldsymbol{E}$, respectively.

Define the *query threshold* $\mathbf{QTh}^{\mathrm{eee}}_{1/2}(k,n)$ as the largest integer q for which $\mathbf{Adv}^{\mathrm{eee}}_{k,n}(q) \le 1/2$. We will speak of EEE being secure up to $\mathbf{QTh}^{\mathrm{eee}}_{1/2}(k,n)$ queries. Let $\mathbf{Adv}^{\mathrm{ede}}_{k,n}(A), \mathbf{Adv}^{\mathrm{ede}}_{k,n}(q)$, and $\mathbf{QTh}^{\mathrm{ede}}_{1/2}(k,n)$ be defined in the analogous way.

Results. The main result of this section is the following:

Theorem 3 [Security of triple-encryption] *Let $k,n \ge 2$. Let $\alpha = \max(2e2^{k-n}, 2n+k)$. Then*

$$\mathbf{Adv}^{\mathrm{eee}}_{k,n}(q) \;\le\; 4\alpha\,\frac{q^2}{2^{3k}} + 10.7\left(\frac{q}{2^{k+n/2}}\right)^{2/3} + \frac{12}{2^k}\,. \quad \square \tag{7}$$

We display the result graphically in Figure 1 for DES parameters $k = 56$ and $n = 64$. Our bound implies that $\mathbf{QTh}^{\mathrm{eee}}_{1/2}(k,n)$ is, very roughly, about $2^{k+\min(k,n)/2}$, meaning that EEE is secure up to this many queries. We remark that Lucks [31] provides a key-recovery attack on EEE that succeeds in about $2^{k+n/2}$ queries, indicating that our threshold value is reasonably tight for $n \le k$.

For EDE the result is the same, meaning that $\mathbf{Adv}^{\mathrm{ede}}_{k,n}(q)$ is also bounded by the quantity on the right-hand-side of (7). This can be shown by mostly-notational modifications to the proof of Theorem 3.

Conventions. The rest of this section, as well as Appendix E, is for proving Theorem 3. We begin with some conventions. Recall that an adversary A against EEE or EDE can make oracle queries $\boldsymbol{T}(X)$, $\boldsymbol{T}^{-1}(Y)$, $\boldsymbol{E}(K,X)$, or $\boldsymbol{E}^{-1}(K,Y)$ for any $X,Y \in \{0,1\}^n$ and $K \in \{0,1\}^k$. We will assume that any adversary against EEE or EDE is deterministic and never makes a *redundant* query. A query is redundant if it has been made before; a query $\boldsymbol{T}^{-1}(Y)$ is redundant if A has previously received Y in answer to a query $\boldsymbol{T}(X)$; a query $\boldsymbol{T}(X)$ is redundant if A has previously received X in answer to a query $\boldsymbol{T}^{-1}(Y)$; a query $\boldsymbol{E}^{-1}(K,Y)$ is redundant if A has previously received Y in answer to a query $\boldsymbol{E}(K,X)$; a query $\boldsymbol{E}(K,X)$ is redundant if A has previously received X in answer to a query $\boldsymbol{E}^{-1}(K,Y)$. Assuming A to be deterministic and not to ask redundant queries is without loss of generality in the sense that for any A that asks q queries there is an A' asking at most q queries that satisfies these assumptions and achieves the same advantage as A. Finally, recall that our general conventions imply that A never asks a query with arguments outside of the intended domain, meaning $\{0,1\}^k$ for keys and $\{0,1\}^n$ for messages. We say that an adversary is *simplified* if it does not make $\boldsymbol{T}(\cdot)$, $\boldsymbol{T}^{-1}(\cdot)$ queries (that is, it makes only $\boldsymbol{E}(\cdot,\cdot)$, $\boldsymbol{E}^{-1}(\cdot,\cdot)$ queries).

Simplifying the adversary. The first step in our proof is to reduce the problem of bounding the advantage of an adversary against EEE to the problem of bounding advantage for a simplified adversary that aims to distinguish between a new pair of games (obviously not the original pair of games since an adversary can get no advantage in distinguishing R_0 and C_0 if it makes no $\boldsymbol{T}(\cdot)$ or $\boldsymbol{T}^{-1}(\cdot)$ queries). Consider the games in Figure 3. The R-games (where R stands for random) omit the boxed assignment statements while the C-games (where C stands for construction) include them. $\mathrm{Distinct}^k_3$ denotes the set of all triples $(K_0,K_1,K_2) \in (\{0,1\}^k)^3$ such that $K_0 \ne K_1$ and

procedure Initialize Game R_0 $\boxed{\text{Game } C_0}$

$K_0, K_1, K_2 \xleftarrow{\$} \{0,1\}^k$, $E \xleftarrow{\$} \mathrm{Bloc}(k,n)$, $T \xleftarrow{\$} \mathrm{Perm}(n)$, $\boxed{T \leftarrow E_{K_2} \circ E_{K_1} \circ E_{K_0}}$

procedure $\boldsymbol{T}(P)$
return $T[P]$

procedure $\boldsymbol{T^{-1}}(S)$
return $T^{-1}[S]$

procedure $\boldsymbol{E}(K,X)$
return $E_K[X]$

procedure $\boldsymbol{E^{-1}}(K,Y)$
return $E_K^{-1}[Y]$

procedure Initialize Game R_1 $\boxed{\text{Game } C_1}$

$(K_0, K_1, K_2) \xleftarrow{\$} \mathrm{Distinct}_3^k$, $E \xleftarrow{\$} \mathrm{Bloc}(k,n)$, $T \xleftarrow{\$} \mathrm{Perm}(n)$, $\boxed{T \leftarrow E_{K_2} \circ E_{K_1} \circ E_{K_0}}$

procedure $\boldsymbol{T}(P)$
return $T[P]$

procedure $\boldsymbol{T^{-1}}(S)$
return $T^{-1}[S]$

procedure $\boldsymbol{E}(K,X)$
return $E_K[X]$

procedure $\boldsymbol{E^{-1}}(K,Y)$
return $E_K^{-1}[Y]$

procedure Initialize Game R_1 $\boxed{\text{Game } C_2}$

$(K_0, K_1, K_2) \xleftarrow{\$} \mathrm{Distinct}_3^k$, $E \xleftarrow{\$} \mathrm{Bloc}(k,n)$, $T \xleftarrow{\$} \mathrm{Perm}(n)$, $\boxed{E_{K_2} \leftarrow T \circ E_{K_0}^{-1} \circ E_{K_1}^{-1}}$

procedure $\boldsymbol{T}(P)$
return $T[P]$

procedure $\boldsymbol{T^{-1}}(S)$
return $T^{-1}[S]$

procedure $\boldsymbol{E}(K,X)$
return $E_K[X]$

procedure $\boldsymbol{E^{-1}}(K,Y)$
return $E_K^{-1}[Y]$

procedure Initialize Game R_2 $\boxed{\text{Game } C_T}$

300 $(K_0, K_1, K_2) \xleftarrow{\$} \mathrm{Distinct}_3^k$, $E \xleftarrow{\$} \mathrm{Bloc}(k,n)$, $\boxed{E_{K_2} \leftarrow T \circ E_{K_0}^{-1} \circ E_{K_1}^{-1}}$

procedure $\boldsymbol{E}(K,X)$
310 return $E_K[X]$

procedure $\boldsymbol{E^{-1}}(K,Y)$
320 return $E_K^{-1}[Y]$

Figure 3: Games used for triple encryption. The C_i games include the boxed statements, the R_i games do not.

$K_1 \neq K_2$ and $K_0 \neq K_2$. Game C_T is parameterized by a permutation $T \in \mathrm{Perm}(n)$, meaning we are effectively defining one such game for every T. Now we claim the following:

Lemma 4 *Let A_1 be an adversary that makes at most q oracle queries. Then there is a simplified adversary B and a permutation $S \in \mathrm{Perm}(n)$ such that*

$$\mathbf{Adv}_{k,n}^{\mathrm{eee}}(A_1) \;\leq\; \mathbf{Adv}(B^{C_S}, B^{R_2}) + \frac{6}{2^k}\ .$$

Furthermore, B also makes at most q oracle queries. ▌

Proof: The only change between games R_0 and R_1 is to draw the keys K_0, K_1, K_2 from $\mathrm{Distinct}_3^k$ rather than from $(\{0,1\}^k)^3$, and similarly for games C_0 and C_1. So

$$\mathbf{Adv}(A_1^{C_0}, A_1^{C_1}) \leq \frac{3}{2^k} \quad \text{and} \quad \mathbf{Adv}(A_1^{R_1}, A_1^{R_0}) \leq \frac{3}{2^k}\ .$$

(The above can easily be shown with games, but it is so simple that we do not bother.) Now using

triangle equality (1) and the above we have

$$\begin{aligned}\mathbf{Adv}^{\mathrm{eee}}_{k,n}(A_1) &= \mathbf{Adv}(A_1^{C_0}, A_1^{R_0}) = \mathbf{Adv}(A_1^{C_0}, A_1^{C_1}) + \mathbf{Adv}(A_1^{C_1}, A_1^{R_1}) + \mathbf{Adv}(A_1^{R_1}, A_1^{R_0}) \\ &\leq \mathbf{Adv}(A_1^{C_1}, A_1^{R_1}) + \frac{6}{2^k} \,. \end{aligned} \tag{8}$$

Game C_2 is obtained from game C_1 as follows: instead of defining T as $E_2 \circ E_1 \circ E_0$ for random E_0, E_1, E_2, we define E_2 as $T \circ E_{K_0}^{-1} \circ E_{K_1}$ for random T, E_{K_0}, E_{K_1}. These two processes are identical, and so

$$\mathbf{Adv}(A_1^{C_1}, A_1^{R_1}) = \mathbf{Adv}(A_1^{C_2}, A_1^{R_1}) \,. \tag{9}$$

Game C_T is parameterized by a permutation $T \in \mathrm{Perm}(n)$. For any such T we consider an adversary A_T that has T hardwired in its code and is simplified, meaning can make queries $\boldsymbol{E}(K, X)$ and $\boldsymbol{E^{-1}}(K, Y)$ only. This adversary runs A_1, answering the latter's $\boldsymbol{E}(K, X)$ and $\boldsymbol{E^{-1}}(K, Y)$ queries via its own oracles, and answering $\boldsymbol{T}(X)$ and $\boldsymbol{T^{-1}}(Y)$ queries using T. Note that A_T makes at most q oracle queries. Choose $S \in \mathrm{Perm}(n)$ such that

$$\mathbf{Adv}(A_S^{C_S}, A_S^{R_2}) = \max_{T \in \mathrm{Perm}(n)} \left\{ \mathbf{Adv}(A_T^{C_T}, A_T^{R_2}) \right\}$$

and let $B = A_S$. Then, with the expectation taken over $T \xleftarrow{\$} \mathrm{Perm}(n)$ we have

$$\mathbf{Adv}(A_1^{C_2}, A_1^{R_1}) = \mathbf{E}\left[\mathbf{Adv}(A_T^{C_T}, A_T^{R_2})\right] \leq \mathbf{Adv}(A_S^{C_S}, A_S^{R_2}) = \mathbf{Adv}(B^{C_S}, B^{R_2}) \,. \tag{10}$$

This concludes the proof. ∎

Pseudorandomness of three correlated permutations. Towards bounding the advantage of a simplified adversary in distinguishing between games C_3 and R_3 we posit a new problem. Consider games G and H defined in Figure 4. An adversary may make queries $\boldsymbol{\Pi}(i, X)$ or $\boldsymbol{\Pi^{-1}}(i, Y)$ where $i \in \{0, 1, 2\}$ and $X, Y \in \{0, 1\}^n$. The oracles realize three permutations and their inverses, the function realized by $\boldsymbol{\Pi^{-1}}(i, \cdot)$ being the inverse of the one realizing $\boldsymbol{\Pi}(i, \cdot)$ for all $i \in \{0, 1, 2\}$. In both games permutations π_0, π_1 underlying $\boldsymbol{\Pi}(0, \cdot)$ and $\boldsymbol{\Pi}(1, \cdot)$ are random and independent permutations. In game G, the permutation π_2 underlying $\boldsymbol{\Pi}(2, \cdot)$ is also random and independent of π_0 and π_1, but in game H it is equal to $\pi_1^{-1} \circ \pi_0^{-1}$.

Notice that it is easy for an adversary to distinguish between games G and H by making queries that form a "chain" of length three: for any $P \in \{0, 1\}^n$, let the adversary ask and be given $Q \leftarrow \pi_0(P)$, then $R \leftarrow \pi_1(Q)$, then $P' \leftarrow \pi_2(R)$, and then have the adversary output 1 if $P = P'$ (a "triangle" has been found) or 0 if $P \neq P'$ (the "three-chain" is not in fact a triangle). What we will establish is that, apart from such behavior—extending a known "2-chain"—the adversary is not able to gain much advantage. To capture this, as the adversary A makes its queries and gets replies, the games form an edge-labeled directed graph $\mathcal{G}$. The graph, whose vertex set is $\{0, 1\}^n$, is initially without edges (we omit explicit initialization for the graph $\mathcal{G}$). An arc $X \xrightarrow{i} Y$ is created when a query $\boldsymbol{\Pi}(i, X)$ returns the value Y or a query $\boldsymbol{\Pi^{-1}}(i, Y)$ returns the value X. The boolean flag *x2ch* is set in the games if the adversary "extends a 2-chain," meaning that a path $P \xrightarrow{i+1} Q \xrightarrow{i+2} R$ exists in the graph and the adversary asks either $\boldsymbol{\Pi}(i, R)$ or $\boldsymbol{\Pi^{-1}}(i, P)$, where the indicated addition is modulo 3. We will be interested in the game outputs rather than the adversary outputs. If the flag *x2ch* gets set to true then the adversary effectively loses: the game's output is defined as a constant (rather than the adversary's output) and the adversary will gain no advantage form this run. We comment that we have described games G and H as working with a graph, which is not actually a type in our formal programming language $\mathcal{L}$, but it is easy to recast the games to use a set instead of a graph, it is simply that the code may be slightly less readable. We show, again using a game-based proof, that:

procedure Initialize $\pi_0, \pi_1, \pi_2 \xleftarrow{\$} \mathrm{Perm}(n)$, $\boxed{\pi_2 \leftarrow \pi_1^{-1} \circ \pi_0^{-1}}$		Game G $\boxed{\text{Game } H}$
procedure $\boldsymbol{\Pi}(i, X)$ if $\exists\, P \xrightarrow{i+1} Q \xrightarrow{i+2} X \in \mathcal{G}$ then *x2ch* $\leftarrow$ true add $X \xrightarrow{i} \pi_i[X]$ to $\mathcal{G}$ return $\pi_i[X]$	**procedure $\boldsymbol{\Pi^{-1}}(i, Y)$** if $\exists\, Y \xrightarrow{i+1} Q \xrightarrow{i+2} R \in \mathcal{G}$ then *x2ch* $\leftarrow$ true add $\pi_i^{-1}[Y] \xrightarrow{i} Y$ to $\mathcal{G}$ return $\pi_i^{-1}[Y]$	
procedure Finalize(*out*) if *x2ch* then return 1 else return *out*		

procedure $\boldsymbol{E}(K, X)$ return $E_K[X] \xleftarrow{\$} \overline{\mathrm{image}}(E_K)$	**procedure $\boldsymbol{E^{-1}}(K, Y)$** $E_K^{-1}[Y] \xleftarrow{\$} \overline{\mathrm{domain}}(E_K)$	Game L
procedure Finalize $K_0, K_1, K_2 \xleftarrow{\$} \{0,1\}^k$ if $(\exists P)\ [E_{K_2}[E_{K_1}[E_{K_0}[P]]]]$ then *bad* $\leftarrow$ true		

Figure 4: Games used in the analysis of triple encryption. Game H includes the boxed statement, game G does not. Game L captures improbability of making three chains.

Lemma 5 *If* $\Pr\left[B^G \text{ makes} \geq h \text{ oracle queries}\right] \leq \delta$ *then* $\mathbf{Adv}(H^B, G^B) \leq 2.5\, h^2/2^n + \delta$. ▮

The proof is in Section E.1. We remark that the lemma makes no (explicit) assumption about the probability that B^H makes h or more oracle queries.

The improbability of three-chains. Consider an adversary B that can make $\boldsymbol{E}(K, X)$ or $\boldsymbol{E^{-1}}(K, Y)$ queries. Game L of Figure 4 implements the oracles as a random blockcipher and its inverse, respectively, but samples these lazily, defining points as they are needed. Write $X \xrightarrow{K} Y$ to mean that that B has made query $\boldsymbol{E}(K, X)$ and obtained Y as a result, or made query $\boldsymbol{E^{-1}}(K, Y)$ and obtained X as a result, for $K \in \{0,1\}^k$ and $X, Y \in \{0,1\}^n$. The Finalize procedure picks keys K_0, K_1, K_2 at random, and sets *bad* if the adversary's queries have formed a three chain, meaning that there exist points $P, Q, R, S \in \{0,1\}^n$ such that $P \xrightarrow{K_0} Q \xrightarrow{K_1} R \xrightarrow{K_2} S$. The conditional at line 111 means that, for some P, $E_{K_0}[P]$ is defined and $E_{K_1}[E_{K_0}[P]]$ is defined and $E_{K_2}[E_{K_1}[E_{K_0}[P]]]$ is defined. Our next lemma bounds the probability of this happening. The proof is in Section E.2.

Lemma 6 *Let* $k, n \geq 1$. *Let* B *be an adversary that asks at most* q *queries. Let* $\alpha = \max(2e\, 2^{k-n}, 2n + k)$. *Then* $\Pr[B^L \text{ sets } bad] < 2\alpha\, q^2/2^{3k}$. ▮

Reducing to three correlated permutations. Let B be a simplified adversary and $S \in \mathrm{Perm}(n)$ a permutation. We associate to B, S a pair of adversaries $B_{S,1}$ and $B_{S,2}$ that make $\boldsymbol{\Pi}(i, X)$ or $\boldsymbol{\Pi^{-1}}(i, Y)$ queries, where $i \in \{0,1,2\}$ and $X, Y \in \{0,1\}^n$, as follows. For $b \in \{1,2\}$, adversary $B_{S,b}$ picks (K_0, K_1, K_2) at random from $\mathrm{Distinct}_3^k$ and picks E at random from $\mathrm{Bloc}(k, n)$. It then runs B, replying to its oracle queries as follows. If B makes a query $\boldsymbol{E}(K, X)$, adversary $B_{S,b}$ returns $E[K, X]$ if $K \notin \{K_0, K_1, K_2\}$; returns $\boldsymbol{\Pi}(i, X)$ if $K = K_i$ for $i \in \{0,1\}$; and returns $S \circ \boldsymbol{\Pi}(3, X)$ if $i = 3$. Similarly, if B makes a query $\boldsymbol{E^{-1}}(K, Y)$, adversary $B_{S,b}$ returns $E^{-1}[K, Y]$ if $K \notin \{K_0, K_1, K_2\}$; returns $\boldsymbol{\Pi^{-1}}(i, Y)$ if $K = K_i$ for $i \in \{0,1\}$; and returns $\boldsymbol{\Pi^{-1}}(3, Y) \circ S^{-1}$ if $i = 3$. Adversaries $B_{S,1}, B_{S,2}$ differ only in their output, the first always returning 0 and the second returning the output *out* of B.

procedure Initialize — Game R_3

500 $(K_0, K_1, K_2) \xleftarrow{\$} \text{Distinct}_3^k$, $E \xleftarrow{\$} \text{Bloc}(k,n)$, $\boxed{E_{K_2} \leftarrow S \circ E_{K_0}^{-1} \circ E_{K_1}^{-1}}$ — $\boxed{\text{Game } D_S}$

procedure $\boldsymbol{E}(K, X)$

510 if $\exists\, i \in \{0,1,2\}$ such that $K = K_i$ then

511 $\quad Q \leftarrow E_{K_{i+2}}^{-1}[X], \quad P \leftarrow E_{K_{i+1}}^{-1}[Q]$

512 $\quad$ if $P \xrightarrow{i+1} Q \xrightarrow{i+2} X$ then $x2ch \leftarrow \mathsf{true}$

513 $\quad$ Add arc $X \xrightarrow{i} E_K[X]$

514 return $E_K[X]$

procedure $\boldsymbol{E^{-1}}(K, Y)$

520 if $\exists\, i \in \{0,1,2\}$ such that $K = K_i$ then

521 $\quad Q \leftarrow E_{K_{i+1}}[Y], \quad R \leftarrow E_{K_{i+2}}[Q]$

522 $\quad$ if $Y \xrightarrow{i+1} Q \xrightarrow{i+2} R$ then $x2ch \leftarrow \mathsf{true}$

523 $\quad$ Add arc $E_K^{-1}[Y] \xrightarrow{i} Y$

524 return $E_K^{-1}[Y]$

procedure Finalize

530 if $x2ch$ then return 1 else return out

Figure 5: Games used in the triple-encryption analysis. Game D_S includes the boxed statement, game R_3 does not.

Lemma 7 *Let B be a simplified adversary that makes at most q oracle queries, and let $S \in \text{Perm}(n)$. Let $B_{S,1}, B_{S,2}$ be defined based on B, S as above. Let $K = 2^k$. Then for $b \in \{1,2\}$ and every real number $c > 0$,*

$$\Pr\left[\, B_{S,b}^G \text{ makes} \geq 3cq/K \text{ oracle queries} \,\right] \leq \frac{1}{c}\,. \qquad \blacksquare$$

The proof is in Appendix E.3.

Proof of Theorem 3: We now show how to prove Theorem 3 given the above lemmas. Let A be an adversary against EEE that makes at most q oracle queries. Let B be the simplified adversary, and S the permutation, given by Lemma 4, and let $B_{S,1}, B_{S,2}$ be the adversaries associated to it as described above. Consider the games R_3, D_S of Figure 5 and note that

$$\begin{aligned} \Pr[D_S^B \text{ sets } x2ch] &= \Pr[H^{B_{S,1}} \Rightarrow 1] \quad \text{and} \quad \Pr[R_3^B \text{ sets } x2ch] = \Pr[G^{B_{S,1}} \Rightarrow 1] \\ \Pr[D_S^B \Rightarrow 1] &= \Pr[H^{B_{S,2}} \Rightarrow 1] \quad \text{and} \quad \Pr[R_3^B \Rightarrow 1] = \Pr[G^{B_{S,2}} \Rightarrow 1]\,. \end{aligned} \tag{11}$$

Let $\alpha = \max(2e2^{k-n}, 2n+k)$. Then for any $c > 0$ we have the following:

$$
\begin{aligned}
&\mathbf{Adv}^{\mathrm{eee}}_{k,n}(A) \\
&\leq \mathbf{Adv}(B^{C_S}, B^{R_2}) + \frac{6}{2^k} && (12)\\
&= \mathbf{Adv}(C^B_S, D^B_S) + \mathbf{Adv}(D^B_S, R^B_3) + \mathbf{Adv}(R^B_3, R^B_2) + \frac{6}{2^k} && (13)\\
&\leq \Pr\left[\, D^B_S \text{ sets } x2ch \,\right] + \Pr\left[\, R^B_3 \text{ sets } x2ch \,\right] + \mathbf{Adv}(D^B_S, R^B_3) + \frac{6}{2^k} && (14)\\
&= 2 \cdot \Pr\left[\, R^B_3 \text{ sets } x2ch \,\right] + \Pr\left[\, D^B_S \text{ sets } x2ch \,\right] - \Pr\left[\, R^B_3 \text{ sets } x2ch \,\right] + \mathbf{Adv}(D^B_S, R^B_3) + \frac{6}{2^k} \\
&= 2 \cdot \Pr\left[\, R^B_3 \text{ sets } x2ch \,\right] + \mathbf{Adv}(H^{B_{S,1}}, G^{B_{S,1}}) + \mathbf{Adv}(H^{B_{S,2}}, G^{B_{S,2}}) + \frac{6}{2^k} && (15)\\
&\leq 2 \cdot \left(\frac{3}{2^k} + \Pr[B^L \text{ sets } bad\,] \right) + \mathbf{Adv}(H^{B_{S,1}}, G^{B_{S,1}}) + \mathbf{Adv}(H^{B_{S,2}}, G^{B_{S,2}}) + \frac{6}{2^k} && (16)\\
&\leq 2\left(\frac{3}{2^k} + 2\alpha\,\frac{q^2}{2^{3k}} \right) + \frac{5}{2^n}\left(\frac{3cq}{2^k}\right)^2 + \frac{2}{c} + \frac{6}{2^k} && (17)
\end{aligned}
$$

Above, (12) is by Lemma 4, and (13) is by triangle equality (2). To justify (14) we note that game C_S can be easily transformed into an equivalent game such that this game and game D_S are identical-until-*bad*, and, similarly, game R_2 can be easily transformed into an equivalent game such that this game and game R_3 are identical-until-*bad*, and thus (14) follows from the Fundamental Lemma. To justify (15) we use (11). To justify (16) we note that the probability that R^B_3 extends a 2-chain is at most the probability that L^B forms a 3-chain. The extra term is because L picks the keys K_0, K_1, K_2 independently at random while R_3 picks them from $\mathrm{Distinct}^k_3$. To get (17) we first applied Lemma 6 and then, for each $b \in \{1, 2\}$, applied Lemma 5 in conjunction with Lemma 7. Now, since the above is true for any $c > 0$, we pick a particular one that minimizes the function $f(c) = 45\,c^2 q^2\, 2^{-n-2k} + 2c^{-1}$. The derivative is $f'(c) = 90\,cq^2\, 2^{-n-2k} - 2c^{-2}$, and the only real root of the equation $f'(c) = 0$ is $c = (2^{n+2k}/45q^2)^{1/3}$, for which we have $f(c) = 3(45q^2/2^{n+2k})^{1/3}$. Plugging this into the above yields (7) and concludes the proof of Theorem 3. ∎

Acknowledgments

We thank Tadayoshi Kohno for permission to use his observations about the standard proof of the PRP/PRF Switching Lemma noted in Section A and Appendix B.

Mihir Bellare was supported by NSF 0098123, ANR-0129617, NSF 0208842, and an IBM Faculty Partnership Development Award. Phil Rogaway was supported by NSF 0208842 and a gift from Intel Corp. Much of the work on this paper was carried out while Phil was hosted by Chiang Mai University, Thailand.

References

[1] M. Abe, R. Gennaro, K. Kurosawa and V. Shoup. Tag-KEM/DEM: A new framework for hybrid encryption and a new analysis of Kurosawa-Desmedt KEM. *Eurocrypt '05*.

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

[3] W. Aiello, M. Bellare, G. Di Crescenzo, and R. Venkatesan. Security amplification by composition: the case of doubly-iterated, ideal ciphers. *Crypto '98*.

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

[5] 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*.

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

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

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

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

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

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

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

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

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

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

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

[17] 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.

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

[19] W. Diffie and M. Hellman. Exhaustive cryptanalysis of the data encryption standard. *Computer*, vol. 10, pp. 74–84, 1977.

[20] 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*.

[21] S. Even and O. Goldreich. On the power of cascade ciphers. *ACM Transactions on Computer Systems*, vol. 3, no. 2, pp. 108–116, 1985.

[22] S. Even and Y. Mansour. A construction of a cipher from a single pseudorandom permutation. *Asiacrypt '91*. LNCS 739, Springer-Verlag, pp. 210–224, 1992.

[23] 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.

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

[25] S. Halevi. A plausible approach to computer-aided cryptographic proofs. Cryptology ePrint archive report 2005/181, 2005.

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

[27] S. Halevi and P. Rogaway. A tweakable enciphering mode. *Crypto '03*. LNCS 2729, pp. 482–499, 2004.

[28] C. Hall, D. Wagner, J. Kelsey, and B. Schneier. Building PRFs from PRPs. Available on

Wagner's web page. Earlier version in *Crypto '98*.
[29] E. Jaulmes, A. Joux, and F. Valette. On the security of randomized CBC-MAC beyond the birthday paradox limit: a new construction. *FSE '02*. LNCS 2365, pp. 237–251, 2002.
[30] J. Kilian and P. Rogaway. How to protect DES against exhaustive key search (an analysis of DESX). *J. of Cryptology*, vol. 14, no. 1, pp. 17–35, 2001. Earlier version in *Crypto '96*.
[31] S. Lucks. Attacking triple encryption. *FSE '98*. LNCS 1372, pp. 239–253, 1998.
[32] U. Maurer. Indistinguishability of random systems. *Eurocrypt '02*. LNCS 2332, Springer-Verlag, pp. 110–132, 2002.
[33] U. Maurer and J. Massey. Cascade ciphers: the importance of being first. *J. of Cryptology*, vol. 6, no. 1, pp. 55–61, 1993.
[34] R. Merkle and M. Hellman. On the security of multiple encryption. *Communications of the ACM*, vol. 24, pp. 465–467, 1981.
[35] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
[36] National Institute of Standards and Technology. FIPS PUB 46-3, Data Encryption Standard (DES), 1999. Also ANSI X9.52, Triple Data Encryption Algorithm modes of operation, 1998, and other standards.
[37] E. Petrank and C. Rackoff. CBC MAC for real-time data sources. *J. of Cryptology*, vol. 13, no. 3, pp. 315–338, 2000.
[38] P. Rogaway. Authenticated-encryption with associated-data. *ACM CCS '02*.
[39] P. Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. *Asiacrypt '04*.
[40] 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*.
[41] T. Schweinberger and V. Shoup. ACE: the advanced cryptographic engine. Cryptology ePrint report 2000/022, 2000.
[42] C. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.
[43] V. Shoup. OAEP reconsidered. *J. of Cryptology*, vol. 15, no. 4, pp. 223–249, 2002. Earlier version in *Crypto '01*.
[44] V. Shoup. A proposal for an ISO standard for public key encryption. Cryptology ePrint report 2001/112, 2001.
[45] V. Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint report 2004/332, November 30, 2004.
[46] V. Shoup. Using hash function as a hedge against chosen ciphertext attack. *Eurocrypt '00*.
[47] S. Vaudenay. Decorrelation over infinite domains: the encrypted CBC-MAC case. *Communications in Information and Systems* (CIS), vol. 1, pp. 75–85, 2001.
[48] A. Yao. Theory and applications of trapdoor functions. *FOCS 1982*, pp. 80–91, 1982.

A The PRP/PRF Switching Lemma

The lemma. The natural and conventional assumption to make about a blockcipher is that it behaves as a pseudorandom permutation (PRP). However, it usually turns out to be easier to analyze the security of a blockcipher-based construction assuming the blockcipher is secure as a pseudorandom function (PRF). The gap is then bridged (meaning, a result about the security of the construct assuming the blockcipher is a PRP is obtained) using the following lemma. In what follows, we denote by $A^P \Rightarrow 1$ the event that adversary A, equipped with an oracle P, outputs

the bit 1. Let $\mathrm{Perm}(n)$ be the set of all permutations on $\{0,1\}^n$ and let $\mathrm{Func}(n)$ be the set of all functions from $\{0,1\}^n$ to $\{0,1\}^n$. We assume below that π is randomly sampled from $\mathrm{Perm}(n)$ and ρ is randomly sampled from $\mathrm{Func}(n)$.

Lemma 8 [PRP/PRF Switching Lemma] *Let $n \geq 1$ be an integer. Let A be an adversary that asks at most q oracle queries. Then*

$$|\Pr[\,A^{\pi} \Rightarrow 1\,] - \Pr[\,A^{\rho} \Rightarrow 1\,]| \;\leq\; \frac{q(q-1)}{2^{n+1}}\;. \quad \blacksquare$$

In this section we point to some subtleties in the "standard" proof of this widely used result, as given for example in [5, 28], showing in particular that one of the claims made in these proofs is incorrect. We then show how to prove the lemma in a simple and correct way 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 proof. The standard analysis proceeds as follows. Let Coll ("collision") be the event that A, interacting with oracle $\rho \xleftarrow{\$} \mathrm{Func}(n)$, asks distinct queries X and X' that return the same answer. Let Dist ("distinct") be the complementary event. Now

$$\Pr[A^{\pi} \Rightarrow 1] \;=\; \Pr[A^{\rho} \Rightarrow 1 \mid \textsc{Dist}] \tag{18}$$

since a random permutation is the same as a random function in which everything one obtains from distinct queries is distinct. Letting x be this common value and $y = \Pr[A^{\rho} \Rightarrow 1 \mid \textsc{Coll}]$ we have

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

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

Where is the error in the simple proof above? It's at equation (18): it needn't be the case that $\Pr[A^{\pi} \Rightarrow 1] = \Pr[A^{\rho} \Rightarrow 1 \mid \textsc{Dist}]$, 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 $\Pr[A^{\rho} \Rightarrow 1 \mid \textsc{Dist}]$. Let $n = 1$ and consider the following adversary A with oracle $P\colon \{0,1\} \to \{0,1\}$:

procedure Adversary A
  if $P(0) = 0$ then return 1
    else if $P(1) = 1$ then return 1 else return 0

We claim that

$$\Pr[A^{\pi} \Rightarrow 1] = 1/2 \qquad \text{and} \qquad \Pr[A^{\rho} \Rightarrow 1 \mid \textsc{Dist}] = 2/3\;.$$

The first equation is true because there are two possibilities for $(\pi(0), \pi(1))$, namely $(0,1), (1,0)$, and A returns 1 for one of them, namely $(0,1)$. On the other hand, there are four possibilities for $(\rho(0), \rho(1))$, namely $(0,0), (0,1), (1,0), (1,1)$. The event $A^{\rho} \Rightarrow 1 \wedge \textsc{Dist}$ is true for two of them, namely $(0,0), (0,1)$, while the event Dist is true for three of them, namely $(0,0), (0,1), (1,0)$. Thus $\Pr[A^{\rho} \Rightarrow 1 \wedge \textsc{Dist}]/\Pr[\textsc{Dist}] = 2/3$.

Notice that the number of oracle queries made by the adversary of this counterexample varies, being either one or two, depending on the reply it receives to its first query. This turns out to be crucial in making equation (18) fail, in that if A always makes exactly q oracle queries (regardless of A's coins and the answers returned to its queries) then equation (18) is true. (This was pointed out by Kohno, and his argument is re-produced in Appendix B.) Since one can always first modify

procedure $P(X)$		Game S_0
100	$Y \xleftarrow{\$} \{0,1\}^n$	$\boxed{\text{Game } S_1}$
101	if $Y \in \text{image}(\pi)$ then $bad \leftarrow \mathsf{true}$, $\boxed{Y \xleftarrow{\$} \overline{\text{image}}(\pi)}$	
102	return $\pi[X] \leftarrow Y$	

Figure 6: Games used in the proof of the Switching Lemma. Game S_1 includes the boxed statement and S_0 doesn't.

A to make exactly q queries without altering $\Pr[A^\rho \Rightarrow 1]$ or $\Pr[A^\pi \Rightarrow 1]$, we would be loath to say that the proofs in [5, 28] are incorrect. But the authors make claim (18), 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 (18) 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 us use them to give a correct proof.

Game-based proof. Assume without loss of generality (since A's oracle is deterministic) 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 as interacting with a random permutation oracle $\pi \xleftarrow{\$} \text{Perm}(n)$, think of it as interacting with the Game S_1 shown in Figure 6. Instead of thinking of A as interacting with a random function oracle $\rho \xleftarrow{\$} \text{Func}(n)$, think of A as interacting with the game S_0 shown in the same figure. Game S_0 is game S_1 without the boxed statement. By convention, the boolean variable *bad* is initialized to false while the array π begins everywhere undefined. The games make available to A an oracle which has a formal name, in this case P. Adversary A can query this oracle with a string $X \in \{0,1\}^n$, in which case the code following the procedure $P(X)$ line is executed and the value in the return statement is provided to A as the response to its oracle query. As the game runs, we fill-in values of $\pi[X]$ with n-bit strings. At any point in time, we let $\text{image}(\pi)$ be the set of all n-bit strings Y such that $\pi[X] = Y$ for some X. Let $\overline{\text{image}}(\pi)$ be the complement of this set relative to $\{0,1\}^n$. Let $A^S \Rightarrow 1$ denote the event that A outputs 1 in game $S \in \{S_0, S_1\}$.

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 S_0? Whatever query X it asks, the game returns a random n-bit string Y. So game S_0 perfectly simulates a random function $\rho \xleftarrow{\$} \text{Func}(n)$ (remember that the adversary isn't allowed to repeat a query) and $\Pr[A^\rho \Rightarrow 1] = \Pr[A^{S_0} \Rightarrow 1]$. Similarly, if we're in game S_1, 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 $\Pr[A^\pi \Rightarrow 1] = \Pr[A^{S_1} \Rightarrow 1]$. We complete the proof via the following chain of inequalities, the first of which we have just justified:

$$\begin{aligned} |\Pr[A^\pi \Rightarrow 1] - \Pr[A^\rho \Rightarrow 1]| &= |\Pr[A^{S_1} \Rightarrow 1] - \Pr[A^{S_0} \Rightarrow 1]| \\ &\le \Pr[A^{S_0} \text{ sets } bad] \qquad (19) \\ &\le q(q-1)/2^{n+1}\,. \qquad (20) \end{aligned}$$

Above, "A^{S_0} sets *bad*" refers to the event that the flag *bad* is set to true in the execution of A with game S_0. We justify (19) by appealing to the fundamental lemma of game playing (Lemma 1), which 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.) We justify (20) by observing that, by the union bound, the probability that a Y will ever be in image(π) at line 101 is at most $(1 + 2 + \cdots + (q-1))/2^n = q(q-1)/2^{n+1}$. This completes the proof.

Counter-example revisited. It is instructive to see how the adversary A of the counter-example above fares in the game-playing proof. A computation shows that

$$\Pr[A^{S_0} \Rightarrow 1] = 3/4\,, \qquad \Pr[A^{S_1} \Rightarrow 1] = 1/2\,, \qquad \text{and} \qquad \Pr[A^{S_0} \text{ sets } bad\,] = 1/4\,.$$

So none of the equalities or inequalities that arose in the game-playing proof are violated. Another interesting thing to note is that

$$\Pr\left[\, A^{S_0} \Rightarrow 1 \mid A^{S_0} \text{ does not set } bad \,\right] \;=\; 1 \;=\; \Pr[A^{S_1} \Rightarrow 1]\,,$$

This tells us that the event that *bad* is not set to true in the game-playing proof does *not* correspond to the event Dist in the standard proof, although it appears to capture similar intuition. In the game-playing proof we are considering two experiments with a single underlying probability space, while equation (18) equates probabilities taken in two different spaces, something that is harder to do accurately.

B Fixing the PRP/PRF Switching Lemma Without Games

Let adversary A and other notation be as in Section A,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 (18) 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 (18) 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. (Adversary 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 (18). The bug we highlighted in Section Athus amounts to having claimed (18) 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 (18) 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

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

Let $\Pr_{\text{rand}}[\,\cdot\,]$ denote the probability in the experiment where $\rho \xleftarrow{\$} \text{Func}(n)$. Then

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

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

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

C An Example Programming Language for Games

Formalizing the underlying programming language. Games, as well as adversaries, are programs written in some programming language. In this section we describe a suitable programming language for specifying games, denoted $\mathcal{L}$. See Figure 7 for the context-free grammar $\mathcal{G}$ for $\mathcal{L}$. As usual, not every program generated by this grammar is valid (eg, identifiers mustn't be keywords, two procedures can't have the same name, and so forth). The start symbol for a game is *game* and that for an adversary is *adversary*. We regard a game (and an adversary) as being specified by its parse tree and therefore ignore the fact that $\mathcal{G}$ is ambiguous. If one wants to regard games as textual strings instead of parse trees then ambiguity can easily be dealt with by bracketing *if* and *for* statements, adding parenthesis to expressions, and making extra productions to account for precedence and grouping rules.

Structure of $\mathcal{L}$. Our programming language is intentionally simple. Games employ only static, global variables. A game consists of a sequence of procedures (the order of which is irrelevant). There are three kinds of procedures in games: an initialization procedure, a finalization procedure, and oracle procedures. The first two are distinguished by the keyword Initialize or Finalize, which is used as though it were the procedure name. The adversary is a single procedure, one which uses the keyword Adversary as though it were the procedure name.

The language $\mathcal{L}$ is strongly typed. The types of expressions are integer, boolean, string, set, and array. $\triangleright$ integer: A value of this type is a point in the set $Z = \{\cdots, -2, -1, 0, 1, 2, \cdots\}$, or else $\mathsf{undefined}$. $\triangleright$ boolean: A value of this type is either true or false, or else $\mathsf{undefined}$. $\triangleright$ string: A value of this type is a finite string over the binary alphabet $\Sigma = \{\mathbf{0}, \mathbf{1}\}$, or else $\mathsf{undefined}$. $\triangleright$ set: A value of this type is a finite set of strings, or else $\mathsf{undefined}$. $\triangleright$ array: A value of this type is an associative array from and to strings; formally, an array A is a map $A\colon \{0,1\}^* \to \{0,1\}^* \cup \{\mathsf{undefined}\}$. At any given time, there will be a finite number of strings X for which $A[X] \neq \mathsf{undefined}$. An array can alternatively be regarded as a partial function from strings to strings. An array cannot have the value of $\mathsf{undefined}$, but can be everywhere $\mathsf{undefined}$: $A[X] = \mathsf{undefined}$ for all $X \in \{0,1\}^*$. We assert that $A[\mathsf{undefined}] = \mathsf{undefined}$.

We do not bother to declare variables, but each variable and each expression must have a well-defined type, this type inferable from the program. Demanding that each variable has a statically-inferable type rules out programs with statements like $x \leftarrow x$ or $x \leftarrow \mathsf{undefined}$ where x occurs in no other context to make manifest its type. The possible types for variables are *integer*,

game	$\longrightarrow$	ε | *procedure game*
procedure	$\longrightarrow$	*initialization* | *oracle* | *finalization*
initialization	$\longrightarrow$	**procedure Initialize** *arguments compound*
oracle	$\longrightarrow$	**procedure** *identifier arguments compound*
finalization	$\longrightarrow$	**procedure Finalize** *arguments compound*
adversary	$\longrightarrow$	**procedure Adversary** *arguments compound*
arguments	$\longrightarrow$	ε | (*arglist*)
arglist	$\longrightarrow$	*identifier* | *identifier*, *arglist*
compound	$\longrightarrow$	*simple* | *simple*, *compound*
simple	$\longrightarrow$	*empty* | *assign* | *random* | *if* | *for* | *return*
empty	$\longrightarrow$	ε
assign	$\longrightarrow$	*lvalue* $\leftarrow$ *exp*
random	$\longrightarrow$	*lvalue* $\xleftarrow{\$}$ *set*
if	$\longrightarrow$	if *exp* then *compound* | if *exp* then *compound* else *compound*
for	$\longrightarrow$	for *str* $\in$ *set* do *compound* | for *identifier* $\leftarrow$ *int* to *int* do *compound*
return	$\longrightarrow$	return *exp*
lvalue	$\longrightarrow$	*identifier* | *identifier*[str]
exp	$\longrightarrow$	*bool* | *int* | *str* | *set* | *array* | *call*
bool	$\longrightarrow$	false | true | *exp* = *exp* | *exp* $\neq$ *exp* | *bool* and *bool* | *bool* or *bool* | not *bool* |
		int $<$ *int* | *int* $\leq$ *int* | *int* $>$ *int* | *int* $\geq$ *int* | *str* $\in$ *set* | *str* $\notin$ *set* | *exp* |
		identifier | undefined
int	$\longrightarrow$	*digits* | *int* + *int* | *int* $-$ *int* | *int* $\cdot$ *int* | *int* / *int* | |*set*| | |*str*| |
		identifier | undefined
str	$\longrightarrow$	ε | *bits* | *str* $\|$ *str* | *str*[*int* $\rightarrow$ *int*] | encode(*list*) | *identifier* | *identifier*[*str*] | undefined
set	$\longrightarrow$	$\emptyset$ | { *strlist* } | *set* $\cup$ *set* | *set* $\cap$ *set* | *set* $\setminus$ *set* | *set* $\circ$ *set* | *set* ^ *int* |
		domain(*identifier*) | image(*identifier*) | *identifier* | undefined
array	$\longrightarrow$	*identifier*
call	$\longrightarrow$	*identifier arguments*
list	$\longrightarrow$	*exp* | *exp list*
strlist	$\longrightarrow$	*str* | *str*, *strlist*
bits	$\longrightarrow$	0 | 1 | 0 *bits* | 1 *bits*
digits	$\longrightarrow$	*digit* | *digit digits*
digit	$\longrightarrow$	**0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**
identifier	$\longrightarrow$	*letter characters*
characters	$\longrightarrow$	ε | *letter characters* | *digit characters*
letter	$\longrightarrow$	a | b | $\cdots$ | z | A | B | $\cdots$ | Z

Figure 7: CFG for the sample game-programming-language $\mathcal{L}$. A game G is a program in this language starting from the "game" production. An adversary is also a program in this language, but starting from the "adversary" nonterminal.

boolean, *string*, *set*, or *array*. These mean the same as they did for expressions except that a boolean variable has the semantics of a flag: once true a boolean variable remains true, even if it is assigned false or undefined.

We provide traditional operators like addition on integers, concatenation of strings, and union of sets. Observe that no operator can create an infinite set (eg, we do not provide for Kleene-closure). For an array A we support operators domain(A) and image(A) that return $\{x \in \{0,1\}^* \colon A[x] \neq \mathsf{undefined}]\}$ and $\{A[x] \colon x \in \{0,1\}^*\}$, respectively. We provide an operator encode($\cdots$) that takes a list of values, of any type, and creates a string in such a way that encode(L) $\neq$ encode(L') when $L \neq L'$. We assume lazy evaluation of *and* and *or*, so false and undefined = false, while true or undefined = true.

Each procedure is a sequence of statements. The types of statements supported in $\mathcal{L}$ are as follows. $\triangleright$ The *empty statement* does nothing. $\triangleright$ The *assignment statement* is of the form $x \leftarrow e$ where the left-hand-side must be either a variable or an array reference $A[s]$ for an expression s of type string. In the first case the expression e must have the same type as the variable x, and in the second case it must be a string. The semantics is to evaluate the expression e and then modify the store by assigning this value to x. $\triangleright$ For the *random-assignment statement* $x \xleftarrow{\$} S$, we require x to be a string variable or an array reference $A[s]$ for an array A and string expression s. The right-hand-side S must be a set. The statement uniformly selects a random element from the set S and assigns that value to x. If $S = \emptyset$ or $S = \mathsf{undefined}$ then the result of the random-assignment statement is to set x to $\mathsf{undefined}$. Random-assignment statements are the only source of randomness in programs. An *if* statement comes in two forms, if e then S, and if e then S else S'. The expression e may have any type. If it is not a boolean then $\mathsf{undefined}$ is treated as false and any other value is treated as true (so "if $A[B[s]]$ then S" is legal, and it tests if both $B[s]$ and $A[B[s]]$ are both defined). $\triangleright$ A *for* statement comes in two forms: for $x \in A$ do S, and for $i \leftarrow a$ to b do S. In the first case x must be a string variable and A must be a set. The expression A is evaluated once and the value remembered. The statement that follows is then executed $|A|$ times, with x taking on the values in A in lexicographic order. If A is empty or $\mathsf{undefined}$ then S is not executed. For the form for $i \leftarrow a$ to b do S, the variable i must be an integer variable and expressions a and b are evaluated once, their values being saved. The statement is then evaluated for successive integers between a and b. If $a < b$ or either is $\mathsf{undefined}$ then S is not executed. $\triangleright$ The *return* x statement returns the value x to the caller. All procedures (including Initialize and Finalize) return values in this way. If control flows to the end of a routine without encountering a return statement then $\mathsf{undefined}$ is returned. It is understood that identifiers must not be keywords and that oracles must have unique names. The procedure invocation *call* may not be used in a game; it is provided for the benefit of an adversary, who may call out to named oracles.

An adversary is syntactically identical to an oracle except that the oracle is named "Adversary" and a new expression is allowed, the *call* expression.

We refer the reader to Section 2 for a discussion of initialization and the subsequent execution of a game with an adversary. The description of an execution there is admittedly informal. A rich theory in computer science exists to formalize the execution of programs, and our programming language is simple enough to present no particular difficulties in pinning down an operational semantics.

We observe that programs in our programming language must terminate. This is because we gave the language no *goto*, *while*, or the like; the only way to do repetition is via the *for* statement, which iterates over a finite set. We have never seen it to be an issue if a game programming language is not Turing-complete. It is, intuitively, more of a concern if adversaries are written in a language that is not Turing-complete (and one might wish to add a statement like a *while* statement to address that concern). But note that once a game is fixed and a bound is set on the total number of oracle queries and their lengths, the optimal adversary for this game (optimal in terms of setting a flag *bad*, maximizing the chance of the game outcome is 1, etc.) will be computable, and it will be computable by a program over $\mathcal{L}$. So at least when adversary computational complexity is not at issue, there is no loss of generality to assume that adversaries are programs over $\mathcal{L}$.

Enhancing the language. The language $\mathcal{L}$ is a bit too restrictive to make for a convenient description of some games; it is conveniently supplemented with a bit of "syntactic sugar" (and beyond). (1) We use indentation to indicate grouping for compound statements. (2) We often omit commas as statement separators and use line breaks instead. (3) We allow arrays to be indexed by things other than strings. In such a case the meaning is to apply the encode$(\cdot)$ operator to the

argument: $A[i,x]$ is short for $A[\text{encode}(i,x)]$. (4) We allow use subscripts, superscripts, and Greek characters in identifiers. (5) We sometimes use a subscript in place of an array argument, as in $E_K[X]$ for $E[K,X]$. (6) We write $\overline{\text{domain}}(A)$ and $\overline{\text{image}}(A)$ for $U - \text{domain}(A)$ and $U - \text{image}(A)$ for an understood and specified set U, such as $U = \{0,1\}^n$. (7) Sometimes we write a shorthand that actually would be implemented with a fragment of code. For example, when π is an array and one writes $\pi^{-1}[Y]$, this would have to be re-coded to fit into the syntax of $\mathcal{L}$. If one knows that $\text{domain}(\pi) \subseteq \{0,1\}^n$ then $\pi^{-1}[Y]$ could be re-coded as a *for* loop that runs over $\{0,1\}^n$ looking for the first (and presumably only) point X such that $\pi[X] = Y$; $\pi^{-1}[Y]$ is then that value.

In general, we view a game as being written in the language $\mathcal{L}$ as long as it is obvious how to transform the program so as to be literally in $\mathcal{L}$. If one were using some sort of automated tool for verifying game-based proofs then it would be necessary to make sure that each game was *literally* in the target programming language. But when giving a human-readable proof, incorporating whatever constructs or conventions work to simplify the games one is writing is not only permissible, it is nearly essential: a few well-chosen and problem-specific conventions can make for much more readable code, facilitating more easily verified proofs.

D 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 and widely applicable methods. We begin with some useful terminology.

D.1 Game chains

The power of the game-playing technique stems, in large part, from our ability to incrementally rewrite games, constructing chains of them. Using the Fundamental Lemma, you first arrange that the analysis you want to carry out amounts to bounding $\epsilon = \Pr[G_1^A \text{ sets } bad\,]$ for some first game G_1 and some adversary A. (In fact, a game chain may be used also for this first phase, before we apply the Fundamental Lemma; triple encryption and OAEP are such examples.) You want to bound ϵ as a function of the resources expended by A. To this end, you modify the game G_1, one step at a time, constructing a chain of games $G_1 \longrightarrow G_2 \longrightarrow G_3 \longrightarrow \cdots \longrightarrow G_n$. Game G_1 is the *initial* game and game G_n is the *terminal* game. Game G_1 is run using adversary A; other games may run with different adversaries.

Consider a transition $G^A \to H^B$. Let $p_G = \Pr[G^A \text{ sets } bad\,]$ and let $p_H = \Pr[H^B \text{ sets } bad\,]$. We want to bound p_G in terms of p_H. Sometimes we show that $p_G \le p_H$. In this case, the transformation is *safe*. A common case of this is when $p_G = p_H$, in which case the transformation is *conservative*. 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, the transformation is *lossy*. When a chain of safe and lossy transformations is performed, a bound for *bad* getting set in the initial game is obtained from a bound for *bad* getting set in the terminal game by adding and/or multiplying the loss terms in the natural way. We use the terms safe, conservative, or lossy to apply to pairs of games even in the absence of a specific adversary: the statement is then understood to apply to all adversaries with understood resource bounds.

D.2 Basic techniques

We start off with game-transformation methods that do not require an extensive discussion. The methods can be quite powerful just the same.

Resampling idiom. Let T be a finite, nonempty set and let $S \subseteq T$. Then the code fragment $x \xleftarrow{\$} S$ can be replaced by the equivalent code fragment $x \xleftarrow{\$} T$, if $x \notin S$ then $x \xleftarrow{\$} S$. We call this motif *resampling*. It is a basic "idiom" employed in games, often with *bad* getting set when resampling is required, replacing $x \xleftarrow{\$} S$ by $x \xleftarrow{\$} T$, if $x \notin S$ then $bad \leftarrow \mathsf{true}$, $x \xleftarrow{\$} S$. The code-replacement just specified is clearly safe (but not conservative). Our proof for the PRP/PRF Switching Lemma, our proof for triple-encryption, and our proof for the CBC MAC all employ the resampling idiom.

Swapping dependent and independent variables. Instead of choosing a random value $X \xleftarrow{\$} \{0,1\}^n$ and then defining $Y \leftarrow X \oplus C$, one can choose $Y \xleftarrow{\$} \{0,1\}^n$ and define $X \leftarrow Y \oplus C$. This can be generalized in natural ways. Swapping dependent and independent variables is invariably a conservative change. This technique is used in our proof of OAEP.

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 employ those 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 procedure Finalize making random choices that had been made earlier. Permissible code motion is conservative. Examples of code motion are given in the proof of the CBC MAC.

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. Marking-instead-of-recording is used in our proof of the CBC MAC.

Derandomizing a variable. Suppose a game G chooses a variable $X \xleftarrow{\$} \mathcal{X}$ and never re-defines it. We may eliminate the random-assignment statement $X \xleftarrow{\$} \mathcal{X}$ and replace all uses of X by a fixed constant $c \in X$, obtaining a new game G_c. Given an adversary A, let H be G_c for the lexicographically first $c \in \mathcal{X}$ that maximizes $\Pr[G_c^A \text{ sets } bad\,]$. We say that game H has been obtained by derandomizing the variable X. It is easy to see that $\Pr[G^A \text{ sets } bad\,] \leq \Pr[H^A \text{ sets } bad\,]$; that is, derandomizing a variable is a safe transformation. One can also de-randomize when elements of an array are chosen at random, replacing code for $i \xleftarrow{\$} 1$ to q do $A[i] \xleftarrow{\$} \mathcal{X}$ by references to constants. Our analysis of the triple can be viewed as using this method in passing from game C_2 to C_T (but see Section D.3 for a fuller discussion).

Poisoned points. Sometimes a game will place a value in an array that it probably will not use; that is, $A[x]$ is defined as some value c but it is unlikely that the value of $A[x]$ will actually be needed. In this case, one can think of "poisoning" array position x of A, dropping c into $A[x]$ but setting a flag *bad* if $A[x]$ is ever accessed. Next one can imagine storing a value b different from c in at $A[x]$, and since any access to $A[x]$ will set *bad*, it doesn't matter that a "wrong" value was placed in $A[x]$: it will not increase the probability that *bad* gets set. We call this the "poisoned-point technique." It is useful because it allows one to make a change in what is stored in an array (setting $A[x]$ to b instead of c) and pay for corrupting the game only later, and only if $A[x]$ is actually accessed. We use the poisoned-point technique in our proof of triple encryption when passing from game G_4 to game G_5.

Unplayable games. We point out that 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 most settings it is perfectly fine to use unplayable games, although none are used within this paper.

D.3 Coin fixing

It is often much easier to carry out an analysis if one can assume that adversaries decide on their queries in advance; the fact that adversaries can choose their queries adaptively is one of the major factors complicating many analyses. We now describe the *coin-fixing technique* to eliminate or reduce adversarial adaptivity. You can't *always* apply this method to remove adaptivity; our purpose is to given an example of a sufficient condition when you can. The coin-fixing technique stems from a classical method in complexity theory to eliminate coins, hardwiring them in, as in the proof that BPP $\subseteq$ P/poly [2].

Fix a game G and an enumeration of its oracles, $P_1, \ldots, P_m$ for some $m \geq 0$. We would like to eliminate oracle P_1, constructing a new game H with oracles $P_2, \ldots, P_m$. If G has only one oracle, P_1, then H will be left with no oracles: it will be a non-interactive game, a game whose execution does not depend on an adversary. A *resource vector* is a sequence of numbers $R = (q_1, \ldots, q_m) \in \mathbb{N}^m$. For such a vector $R = (q_1, \ldots, q_m)$ let $\mathcal{A}_R$ denote the set of all adversaries that ask exactly q_i queries to oracle P_i. Let $\mathrm{tail}(R) = (q_2, \ldots, q_m)$ be R stripped of its first component. When speaking of a game G where each oracle P_i has an understood domain D_i, the adversary class $\mathcal{A}_R$ is further assumed to include only adversaries that ask queries that are within the domain of each oracle.

Now suppose that oracle P_1 is of the form:

$$\textbf{procedure } P_1(X) \quad i \leftarrow i+1, \quad X[i] \leftarrow X, \quad Y[i] \xleftarrow{\$} D_{X[1],\ldots,X[i],Y[1],\ldots,Y[i-1]}, \quad S, \quad \text{return } Y[i]$$

where i is some integer variable and X and Y are distinct array variables and i, $X[\cdot]$, and $Y[\cdot]$ appear in game G as l-values (that is, on the left-hand-side of an assignment operator or random-assignment operator) only in the statements shown. Statement S is empty if $m > 1$ and an arbitrary compound statement if $m = 1$. By $D_{X[1],\ldots,X[i],Y[1],\ldots,Y[i-1]}$ we mean a finite set that depends only on $X[1], \ldots, X[i], Y[1], \ldots, Y[i-1]$. To fall within our code-based framework the set $D_{X[1],\ldots,X[i],Y[1],\ldots,Y[i-1]}$ must be specified in code; what we are asserting that this code computes the associated set and is without side-effects. A useful special cases is $Y[i] \xleftarrow{\$} \{0,1\}^n$ for some constant n. When P_1 has the form we have just described we say that it is *eliminable* and has parameters i, X, Y, S.

With all notation as above, let $\mathtt{X}, \mathtt{Y} \in (\{0,1\}^*)^{q_1}$. We say that $(\mathtt{X}, \mathtt{Y})$ is *possible* for $(G, \mathcal{A}_R)$ if there exists an adversary $A \in \mathcal{A}_R$ for which there is nonzero probability that, in the interaction between A and G, the former asks queries $\mathtt{X}_1, \ldots, \mathtt{X}_{q_1}$ and receives in response $\mathtt{Y}_1, \ldots, \mathtt{Y}_{q_1}$. A *query set* for $(G, \mathcal{A}_R)$ is a finite set of points $\mathcal{Q}$ that includes all possible $(\mathtt{X}, \mathtt{Y})$ for $(G, \mathcal{A}_R)$.

For $\mathtt{X}, \mathtt{Y} \in (\{0,1\}^*)^{q_1}$ define the game $G_{\mathtt{X},\mathtt{Y}}$ to be identical to G except for the following: (1) Eliminate the procedure for oracle P_1. (2) Replace every expression of the form $X[e]$ by $\mathtt{X}[e]$. (no $X[e]$ appears as an l-value). (3) Replace every expression of the form $Y[e]$ by $\mathtt{Y}[e]$ (no $Y[e]$ appears as an l-value). (4) At the beginning of procedure Finalize, add the statement: for $i \leftarrow 1$ to q_1 do S.

We eliminate P_1 by appealing to the following.

Proposition 9 *Let G be a game with oracles $P_1, \ldots, P_m$. Let $R = (q_1, \ldots, q_m) \in \mathbb{N}^m$ and $R' =$*

Game G_1	Game G_2
procedure Initialize 100 $\pi \xleftarrow{\$} \mathrm{Perm}(n)$ **procedure $\mathbf{\Pi}(X)$** 110 return $\pi[X]$	**procedure $\mathbf{\Pi}(X)$** 210 return $\pi[X] \xleftarrow{\$} \overline{\mathrm{image}}(\pi)$
Game H_1	**Game H_2**
procedure Initialize 100 $(\pi_0, \pi_1) \xleftarrow{\$} \mathrm{Perm}_2^{\neq}(S)$ **procedure $\mathbf{\Pi}_0(X)$** 110 return $\pi_0[X]$ **procedure $\mathbf{\Pi}_1(X)$** 120 return $\pi_1[X]$	**procedure $\mathbf{\Pi}_0(X)$** 210 return $\pi_0[X] \xleftarrow{\$} \overline{\mathrm{image}}(\pi_0) - \{\pi_1[X]\}$ **procedure $\mathbf{\Pi}_1(X)$** 220 return $\pi_1[X] \xleftarrow{\$} \overline{\mathrm{image}}(\pi_1) - \{\pi_0[X]\}$

Game H_3

procedure $\mathbf{\Pi}_0(X)$
310 if $X \in \overline{\mathrm{domain}}(\pi_0)$ then $(\pi_0[X], \pi_1[X]) \xleftarrow{\$} \{\, (Y_1, Y_2) \in \overline{\mathrm{image}}(\pi_0) \times \overline{\mathrm{image}}(\pi_1) \;:\; Y_1 \neq Y_2 \,\}$
311 return $\pi_0[X]$

procedure $\mathbf{\Pi}_1(X)$
320 if $X \in \overline{\mathrm{domain}}(\pi_1)$ then $(\pi_0[X], \pi_1[X]) \xleftarrow{\$} \{\, (Y_1, Y_2) \in \overline{\mathrm{image}}(\pi_0) \times \overline{\mathrm{image}}(\pi_1) \;:\; Y_1 \neq Y_2 \,\}$
321 return $\pi_1[X]$

Figure 8: Games referred to in discussion of lazy sampling.

tail(R). *Let $\mathcal{Q}$ be a query set. Then*

$$\max_{A \in \mathcal{A}_R} \{\Pr\left[\, G^A \text{ sets } \mathit{bad} \,\right]\} \;\leq\; \max_{B \in \mathcal{A}_{R'},\, (\mathtt{X},\mathtt{Y}) \in \mathcal{Q}} \{\Pr\left[\, G^B_{\mathtt{X},\mathtt{Y}} \text{ sets } \mathit{bad} \,\right]\} \qquad \blacksquare$$

D.4 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*. Whether a lazy sampling method works or not is sometimes clear but other times subtle. In this section we look at some examples and then provide a sufficient condition for lazy sampling to work.

Example 1. As a simple but frequently used example, consider a game that provides the adversary an oracle $\mathbf{\Pi}(X)$ that is a permutation on $\{0,1\}^n$. One implementation of the game is game G_1 of Figure 8 which simply chooses π at random from $\mathrm{Perm}(n)$ during Initialize and then, when asked a $\mathbf{\Pi}(X)$ query, answers $\pi[X]$. The alternative, lazy, method for implementing π is shown in game G_2. This game maintains a partial permutation π from n bits to n bits that is initially everywhere undefined. When asked a query X not yet in the domain of π, the game chooses a value Y randomly from the co-range of π, defines $\pi[X] \leftarrow Y$, and returns Y.

You can think of the current partial function π as imposing the "constraint" that $\pi[X] \notin \mathrm{image}(\pi)$ on our choice of $\pi[X]$. We choose $\pi[X]$ at random from all points respecting the constraint. In this case, it seems obvious that the two ways to simulate a random permutation are equivalent. (Recall we have defined this to mean that $\mathbf{Adv}(G_1^A, G_2^A) = 0$ for any adversary A.) 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. Let $\mathrm{Perm}_2^{\neq}(S)$ denote the set of all (π_0, π_1) such that π_0, π_1 are permutations on the set S that satisfy $\pi_0[X] \neq \pi_1[X]$ for all $X \in S$. Game H_1 of Figure 8 provides the adversary with oracles for π_0 and π_1 chosen at random from this set. Game H_2 presents a lazy sampling

alternative. Here, π_0, π_1 are (according to our conventions) initially everywhere undefined. When the adversary queries $\mathbf{\Pi}_i(X)$, the reply $\pi_i[X]$ is chosen at random from the set of points that are not yet in the range of π_i and also different from $\pi_{1-i}[X]$. This, as above, can be thought of as imposing the "constraint" that $\pi_i[X] \notin \text{image}(\pi_i) \cup \{\pi_{1-i}[X]\}$ on our choice of $\pi[X]$, and is thus quite natural.

We will assume as usual that the adversary does not repeat an oracle query. Now we ask whether games H_1, H_2 are equivalent, meaning whether our lazy sampling method "works." Curiously, the answer is no. To see this, let $S = \{1, 2, 3\}$, and consider the adversary A who queries $\mathbf{\Pi}_0(1)$, $\mathbf{\Pi}_0(2)$, $\mathbf{\Pi}_0(3)$, $\mathbf{\Pi}_1(1)$, $\mathbf{\Pi}_1(2)$. It outputs 1 if and only if the answer sequence received back is $1, 2, 3, 2, 1$. Then $\Pr[A^{H_1} \Rightarrow 1] = 0$ because the probability that A gets back the answer sequence $1, 2, 3, 2, 1$ when run with H_1 is zero. This is the case because there is no pair $(\pi_0, \pi_1) \in \text{Perm}_2^{\neq}(S)$ satisfying $\pi_0[1] = 1, \pi_0[2] = 2, \pi_0[3] = 3, \pi_1[1] = 2, \pi_1[2] = 1$. (It leaves no choice for $\pi_1[3]$, since the latter is not allowed to equal any of $\pi_1[1], \pi_1[2], \pi_0[3]$.) However, $\Pr[A^{H_2} \Rightarrow 1] = 1/24$ because the probability that A gets back the answer sequence $1, 2, 3, 2, 1$ when run with H_2 is $1/24$. So $\mathbf{Adv}(H_2^A, H_1^A) = 1/24 \neq 0$, meaning the games are not equivalent. That is, in this case, at least, the "natural" way of performing lazy sampling did not work.

There could, however, be some different lazy sampling method that did work in this case. We present a possibility via game H_3 of Figure 8. We leave as an exercise for the reader to determine whether or not H_3 is equivalent to H_1 and prove the answer correct.

An aid to justifying lazy sampling methods. We would like some way to "test" a candidate lazy sampling method and determine whether or not it works. At least we would like a sufficient condition which, if met by the method, allows us to conclude it works. Here we put forth a result in this vein. Basically we say that we can restrict our attention to adversaries that make all possible oracle queries. Let us now make this more precise.

Let I be a game with an oracle P. We say that I *implements* P *consistently* if for any adversary A, if A has made a query $P(X)$ and got back a value Y, then, if A again makes query $P(X)$, the response will be Y. (This does *not* mean that there are no random choices involved in implementing P, but rather that the value of this oracle at some point, once defined, is not changed.) When all oracles provided by a game are implemented consistently, we can without loss of generality restrict attention to adversaries that never repeat an oracle query.

We say that games G, H are *compatible* if they provide access to oracles of the same names, meaning there is some $n \geq 0$ and some $P_1, \ldots, P_n$ such that both games provide these oracles and no others, and also both games implement all oracles consistently. (The oracles might be implemented differently in the two games. Think of G as implementing them via an eager sampling method and H via a lazy sampling one.)

Let G, H be compatible games providing oracles $P_1, \ldots, P_n$ for some $n \geq 1$. We assume there is a finite domain D_i associated to P_i and that all adversaries considered must pick their queries to P_i from D_i for $1 \leq i \leq n$. (Such domains are always implicit in games.) We say that an adversary A is *exhaustive* if it queries all oracles at all possible points. That is, for any game I compatible with games G, H, for each i and each $X \in D_i$, there is some point in the execution of A with I at which A makes query $P_i(X)$. Note there are many possible exhaustive adversaries, for we do not impose requirements on the order in which they make their queries, or even on whether they are adaptive or not.

Lemma 10 *Let G, H be compatible games. Then G and H are equivalent if and only if $\mathbf{Adv}(G^A, H^A) = 0$ for all exhaustive adversaries A.* ▌

Proof: If the games are equivalent then by definition $\mathbf{Adv}(G^A, H^A) = 0$ for any adversary A, so in

particular this is true when A is exhaustive. Conversely, let B be any adversary, and associate to it an exhaustive adversary A that works as follows. Adversary A runs B, answering B's oracle queries via its own oracles. When B halts with some output b, adversary A continues, making all oracle queries not made by B. (Since the domains of the oracles are finite, A will terminate.) Finally it outputs b and halts. Then $\mathbf{Adv}(G^B, H^B) = \mathbf{Adv}(G^A, H^A)$. But the latter is by assumption zero since A is exhaustive, and thus $\mathbf{Adv}(G^B, H^B) = 0$ as well. Since B was arbitrary, this shows the games are equivalent. ▮

We remark that we effectively used the principle of the above lemma in justifying the lazy sampling method of G_4 in the proof of Lemma 11, when we assumed that A makes all $6N$ oracle queries.

E Proofs for Triple-Encryption

E.1 Proof of Lemma 5

It will be easy to prove Lemma 5 given the following lemma:

Lemma 11 *Let A be an adversary asking at most q oracle queries. Then*

$$\left|\mathbf{Adv}(G^A, H^A)\right| \;\leq\; 2.5\, q^2/2^n \;. \qquad ▮$$

Let us now complete the proof of Lemma 5 given Lemma 11 and then return to prove the latter.

Proof of Lemma 5: We construct an adversary A that has the same oracles as B. Adversary A runs B, answering B's oracle queries via its own oracles. It also keeps track of the number of oracle queries that B makes. If this number hits h, it stops and outputs 1. Else, it outputs whatever B outputs. Then we note that

$$\begin{aligned} \Pr[H^B \Rightarrow 1] &\leq \Pr[H^A \Rightarrow 1] \\ \Pr[G^A \Rightarrow 1] &\leq \Pr[G^B \Rightarrow 1] + \delta \;. \end{aligned}$$

Thus we have

$$\begin{aligned} \mathbf{Adv}(H^B, G^B) &= \Pr[H^B \Rightarrow 1] - \Pr[G^B \Rightarrow 1] \\ &\leq \Pr[H^A \Rightarrow 1] - \big(\Pr[G^A \Rightarrow 1] - \delta\big) \\ &= \mathbf{Adv}(H^A, G^A) + \delta \;. \end{aligned}$$

Since A always makes at most h oracle queries, we conclude by applying Lemma 11 to A with $q = h$. ▮

Proof of Lemma 11: We assume that the adversary A never repeats a query, never asks a query $\mathbf{\Pi^{-1}}(i, Y)$ having asked some $\mathbf{\Pi}(i, X)$ that returned Y, and never asks a query $\mathbf{\Pi}(i, X)$ having asked some $\mathbf{\Pi^{-1}}(i, Y)$ that returned X. We say that the adversary *extends a two-chain* if it asks a query $\mathbf{\Pi}(i+2, R)$ when the graph $\mathcal{G}$ already has a path $P \xrightarrow{i} Q \xrightarrow{i+1} R$, or if it asks a query $\mathbf{\Pi^{-1}}(i+2, P)$ when the graph $\mathcal{G}$ already has a path $P \xrightarrow{i} Q \xrightarrow{i+1} R$. When we write addition here we mean addition modulo 3. Throughout this section, an adversary is said to be *valid* if it never extends a two-chain (no matter what sequence of oracle-responses it receives).

We begin by noting that to bound A's advantage in distinguishing games G and H we may assume that A is valid. Why? Because for any F making at most q_0 queries there exists a valid A that

procedure Initialize — Game G_1 / $\boxed{\text{Game } G_1'}$

$\pi_0, \pi_1, \pi_2 \xleftarrow{\$} \mathrm{Perm}(n)$, $\boxed{\pi_2 \leftarrow \pi_1^{-1} \circ \pi_0^{-1}}$

procedure $\mathbf{\Pi}(i, X)$

return $\pi_i[X]$

procedure $\mathbf{\Pi^{-1}}(i, Y)$

return $\pi_i^{-1}[Y]$

---

procedure $\mathbf{\Pi}(i, X)$ — Game G_2 / $\boxed{\text{Game } G_2'}$

$Y \xleftarrow{\$} \{0,1\}^n$

if $Y \in \mathrm{image}(\pi_i)$ then $bad \leftarrow \mathsf{true}$,

$\boxed{Y \xleftarrow{\$} \overline{\mathrm{image}}(\pi_i)}$

$\pi[X] \leftarrow Y$

return Y

procedure $\mathbf{\Pi^{-1}}(i, Y)$

$X \xleftarrow{\$} \{0,1\}^n$

if $X \in \mathrm{domain}(\pi_i)$ then $bad \leftarrow \mathsf{true}$,

$\boxed{X \xleftarrow{\$} \overline{\mathrm{domain}}(\pi_i)}$

$\pi[X] \leftarrow Y$

return X

---

procedure $\mathbf{\Pi}(i, X)$ — Game G_3

return $Y \xleftarrow{\$} \{0,1\}^n$

procedure $\mathbf{\Pi^{-1}}(i, Y)$

return $X \xleftarrow{\$} \{0,1\}^n$

---

procedure $\mathbf{\Pi}(i, X)$ — Game G_4

if $\exists\, (+1, i, X, Y) \in \mathcal{C}$ then return Y

$X_i \leftarrow X$

$X_{i+1} \xleftarrow{\$} \{0,1\}^n$, if $X_{i+1} \in S_{i+1}$ then $bad \leftarrow \mathsf{true}$, $X_{i+1} \xleftarrow{\$} \{0,1\}^n \setminus S_{i+1}$

$X_{i+2} \xleftarrow{\$} \{0,1\}^n$, if $X_{i+2} \in S_{i+2}$ then $bad \leftarrow \mathsf{true}$, $X_{i+2} \xleftarrow{\$} \{0,1\}^n \setminus S_{i+2}$

$S_i \leftarrow S_i \cup \{X_i\}$, $S_{i+1} \leftarrow S_{i+1} \cup \{X_{i+1}\}$, $S_{i+2} \leftarrow S_{i+2} \cup \{X_{i+2}\}$

$\mathcal{C} \leftarrow \mathcal{C} \cup \{(+1, i, X_i, X_{i+1}), (-1, i, X_{i+1}, X_i),$

$(+1, i+1, X_{i+1}, X_{i+2}), (-1, i+1, X_{i+2}, X_{i+1}),$

$(+1, i+2, X_{i+2}, X_i), (-1, i+2, X_i, X_{i+2})\}$

return X_{i+1}

procedure $\mathbf{\Pi^{-1}}(i, Y)$

if $\exists\, (-1, i, Y, X) \in \mathcal{C}$ then return X

$X_{i+1} \leftarrow Y$

$X_i \xleftarrow{\$} \{0,1\}^n$, if $X_i \in S_{i+1}$ then $bad \leftarrow \mathsf{true}$, $X_i \xleftarrow{\$} \{0,1\}^n \setminus S_{i+1}$

$X_{i+2} \xleftarrow{\$} \{0,1\}^n$, if $X_{i+2} \in S_{i+2}$ then $bad \leftarrow \mathsf{true}$, $X_{i+2} \xleftarrow{\$} \{0,1\}^n \setminus S_{i+2}$

$S_i \leftarrow S_i \cup \{X_i\}$, $S_{i+1} \leftarrow S_{i+1} \cup \{X_{i+1}\}$, $S_{i+2} \leftarrow S_{i+2} \cup \{X_{i+2}\}$

$\mathcal{C} \leftarrow \mathcal{C} \cup \{(+1, i, X_i, X_{i+1}), (-1, i, X_{i+1}, X_i),$

$(+1, i+1, X_{i+1}, X_{i+2}), (-1, i+1, X_{i+2}, X_{i+1}),$

$(+1, i+2, X_{i+2}, X_i), (-1, i+2, X_i, X_{i+2})\}$

return X_i

Figure 9: Games for bounding the probability of distinguishing (π_0, π_1, π_2) and $(\pi_0, \pi_1, \pi_1^{-1} \circ \pi_0^{-1})$.

makes at most q_0 queries and the advantage of A is at least that of F. A runs F, answering F's oracle queries via its own oracles, but at any point that F would extend a two chain, A simply halts and outputs 1. Assuming A's validity, our task is to show that $|\mathbf{Adv}(A^{G_1}, A^{G_1'})| \leq 2\,q^2/2^n$ where the games G_1, G_1' are shown in Figure 9.

We show that games G_1 and G_1' are close by showing that both are close to game G_3 (defined in the same figure). First, we claim that

$$\left|\mathbf{Adv}(A^{G_1}, A^{G_3})\right| \;\leq\; 0.5\,q^2/N \tag{21}$$

where, here and in the rest of this proof, $N = 2^n$. The argument is analogous to that in the game-based proof of the PRP/PRF Switching Lemma given in Appendix A. Rewrite game G_1 to game G_2' by lazily grow π_0, π_1, π_2, setting the flag *bad* whenever there is a collision. Then modify

game G_2' to *not* re-sample after setting *bad*. This results in game G_2. Now we have

$$\left|\mathbf{Adv}(A^{G_1}, A^{G_3})\right| \;=\; \left|\mathbf{Adv}(A^{G_2'}, A^{G_3})\right| \;=\; \left|\mathbf{Adv}(A^{G_2'}, A^{G_2})\right| \;\leq\; \Pr[A^{G_2} \text{ sets } bad\,]\,.$$

Then note that on the ith query the probability that *bad* will be set in the execution of A with G_2 is at most $(i-1)/N$ since the size of domain(π_j) and image(π_j) will be at most $i-1$ for each $j \in \{0,1,2\}$. So over all q queries, the probability that *bad* ever gets set is at most $0.5q(q-1)/N < 0.5q^2/N$, as desired.

Now we claim that

$$\left|\mathbf{Adv}(A^{G_1'}, A^{G_3})\right| \;\leq\; 2\,q^2/N\,. \tag{22}$$

The lemma now follows from (22), (21), triangle equality (1) and the triangle inequality. It remains to show (22). This is somewhat more tricky.

First rewrite game G_1' as game G_4 (again shown in Figure 9). Let us explain game G_4. Here, again, addition (operations +1 and +2) is understood to be addition modulo 3.

Game G_4 uses a rather more involved form of lazy sampling than is typical. In fact the sampling is not entirely lazy; rather, on each query, not only its answer is chosen, but answers for a few related queries are chosen as well and stored for future use.

The game maintains a set $\mathcal{C}$ of *commitments*. Initially there are no commitments (the set is empty) but every time that a query $\mathbf{\Pi}(i,X)$ or $\mathbf{\Pi^{-1}}(i,Y)$ is asked, one of two things happens: if a commitment has already been made as to how to answer this query, we answer according to that prior commitment; and if a commitment has not yet been made, then we not only answer the particular query asked, but we commit ourselves to answers to five further queries—all of the possible queries in a "triangle" containing the queried point.

In greater detail, a commitment $(d,i,P,Q) \in \mathcal{C}$ has four components. The first, $d \in \{+1,-1\}$, indicates if this is a commitment relevant to answering forward queries $\mathbf{\Pi}$ (when $d=+1$) or backward queries $\mathbf{\Pi^{-1}}$ (when $d=-1$). The second component, $i \in \{0,1,2\}$, indicates for which of the three possible permutations this is a commitment. The next two components indicate the input and the output one should give for this query. As an example, a commitment $(+1,2,X,Y) \in \mathcal{C}$ indicates that a query of $\mathbf{\Pi}(2,X)$ must return Y, while a commitment $(-1,0,Y,X) \in \mathcal{C}$ indicates that a query of $\mathbf{\Pi^{-1}}(0,Y)$ must return X.

We will in effect be lazily growing the permutations π_0, π_1, and π_2, but the partially defined permutations have their values stored in $\mathcal{C}$ instead of explicit partial functions π_0, π_1, and π_2. Sets S_0, S_1, and S_2 keep track of the current domains for π_0, π_1 and π_2, respectively.

We claim that games G_4 and G_1' are adversarially indistinguishable even for an adversary that is not valid and that asks all $6N$ possible queries. In particular, this tells us that $\Pr[A^{G_4} \Rightarrow 1] = \Pr[A^{G_1'} \Rightarrow 1]$. To show this equivalence we claim that whether the queries are answered by game G_4 or by game G_1' the adversary gets the *same* view: any of $(N!)^2$ possible outcomes and each of these outcomes occurs with probability $1/(N!)^2$. In particular, the answers correspond to a random pair of permutations π_0, π_1 and the permutation $\pi_2 = \pi_1^{-1} \circ \pi_0^{-1}$. This is obviously the case when playing game G_1'; we must show it to be the case of game G_4. Note that sets S_0, S_1, S_2, and $\mathcal{C}$ begin with no points in them. Then, as A is run with game G_4, they grow with sizes obeying a particular pattern. Namely, the next step is to 1, 1, 1, and 6 points; then to 2, 2, 2, and 12 points; and so forth, until they have N, N, N, and $6N$ points. (Not every query changes the sizes of these sets, but it either leaves them un-altered or changes them as indicated.)

```
procedure Π(i, X)                                                    Game G_5
510  if ∃ (+1, i, X, Y, b) ∈ C then
511      if b = 1 then bad ← true,  Y ←$ {0,1}^n
512      return Y
513  X_i ← X
514  X_{i+1} ←$ {0,1}^n,  if X_{i+1} ∈ S_{i+1} then bad ← true
515  X_{i+2} ←$ {0,1}^n,  if X_{i+2} ∈ S_{i+2} then bad ← true
516  S_i ← S_i ∪ {X_i},  S_{i+1} ← S_{i+1} ∪ {X_{i+1}},  S_{i+2} ← S_{i+2} ∪ {X_{i+2}}
517  C ← C ∪ {(+1, i+1, X_{i+1}, X_{i+2}, 0), (−1, i+1, X_{i+2}, X_{i+1}, 1),
518              (+1, i+2, X_{i+2}, X_i,     1), (−1, i+2, X_i,     X_{i+2}, 0)}
519  return X_{i+1}
procedure Π^{-1}(i, Y)
520  if ∃ (−1, i, Y, X, b) ∈ C then
521      if b = 1 then bad ← true,  X ←$ {0,1}^n
522      return X
523  X_{i+1} ← Y
524  X_i ←$ {0,1}^n,     if X_i ∈ S_{i+1} then bad ← true
525  X_{i+2} ←$ {0,1}^n, if X_{i+2} ∈ S_{i+2} then bad ← true
526  S_i ← S_i ∪ {X_i},  S_{i+1} ← S_{i+1} ∪ {X_{i+1}},  S_{i+2} ← S_{i+2} ∪ {X_{i+2}}
527  C ← C ∪ {(+1, i+1, X_{i+1}, X_{i+2}, 0), (−1, i+1, X_{i+2}, X_{i+1}, 1),
528              (+1, i+2, X_{i+2}, X_i,     1), (−1, i+2, X_i,     X_{i+2}, 0)}
529  return X_i
```

Figure 10: Game G_5.

The first query to increment $\mathcal{C}$ extends the partial functions (π_0, π_1, π_2) in any of N^2 different ways, each with the same probability; the second query to increment $\mathcal{C}$ extends (π_0, π_1, π_2) in any of $(N-1)^2$ different ways, each with the same probability; and so forth, until we have extended (π_0, π_1, π_2) in any of $(N!)^2$ different ways, each with the same probability. This establishes the claim.

Now let us go back to assuming that the adversary is valid. We make a conservative change to game G_4 to arrive at game G_5, shown in Figure 10. We store in each commitment $C \in \mathcal{C}$ a final bit b to indicate if this particular commitment is "poisoned"—that is, if it encodes a query that we would not expect the adversary to be able to find. If the adversary ever asks a query with a committed answer but with a poisoned bit $b = 1$ then we set *bad* and return a random point. In addition, we drop the first two commitments from each group of six that were added, since the constraint that the adversary does not repeat queries or ask the inverse query to answers it has been given means that these commitments will never be used. Finally we pass to game G_5, dropping the sequels to *bad* getting set at lines 412, 413, 422, and 423. We have that $\Pr[A^{G_5} \text{ sets } bad\,] \geq \Pr[A^{G_4} \text{ sets } bad\,]$.

We have two claims to make about game G_5, once concerning the distribution on its output and one concerning the probability that it *bad* gets set in it. We begin with the former, claiming that game G_5 is adversarially indistinguishable from game G_3. Remember that our adversary is valid: it does not ask queries whose answers are trivially known and it does not ask to extend any 2-chain.

Suppose first that the adversary asks a query whose answer has not been memoized in a commitment from $\mathcal{C}$. Then for a forward query, we choose a uniform value X_{i+1} at line 514 and return it (never changing it) at line 519. Likewise for a backward query, we choose a uniform value X_i at line 524 and return it (never changing it) at line 529. So consider instead a memoized forward query, the query belonging to some necessarily unique set of four points that were added earlier to $\mathcal{C}$. We note

that our restrictions on A's queries mean one can query any of these four points at most once; after that, the adversary would be extending a 2-chain. Now if the the "poisoned bit" b is 1 then we return a uniform random n-bit string as our response. Otherwise this is the first query to any of the four added points, and it is a query with poisoned bit $b = 0$. In this case we will be returning the X_{i+2} value from when the first tuple at line 517 was earlier added to $\mathcal{C}$, or we will be returning the X_{i+2} value from when the first tuple at line 527 was earlier added to $\mathcal{C}$. In either case, the value X_{i+2} was chosen at random (at lines 515 or 525) and the value has not yet been used for anything (it was added to $\mathcal{C}$ but has not yet influenced the run), and so this is a uniform random value that we are returning, independent of all oracle responses so far. The same reasoning holds for backward queries and so we have that $\Pr[A^{G_5} \Rightarrow 1] = \Pr[A^{G_3} \Rightarrow 1]$.

The second thing we must do is bound the probability that *bad* gets set in game G_5. The probability that *bad* ever gets set at any of lines 514, 515, 524, or 525 is at most $2(1+2+\cdots+(q-1))/N \le q^2/N$. The probability that *bad* gets set at either of lines 511 or 521 is likewise at most $2(1+2+\cdots+(q-1))/N$, as no information about the poisoned query has been surfaced to the adversary. Overall we have that $\Pr[A^{G_5} \text{ sets } bad\,] \le 2q^2/N$. Putting everything together we have (22) and are done. ∎

E.2 Proof of Lemma 6

We assume without loss of generality that B is deterministic. For any particular blockcipher $E \in \mathrm{Bloc}(k,n)$ we consider the game in which B is executed with oracles E, E^{-1}, which it queries, adaptively, until it halts. (Note that there is no randomness involved in this game, since E is fixed and B is deterministic). Recall that the notation $X \stackrel{K}{\to} Y$ means that B has either made query $E(K,X)$ and obtained Y as a result, or it has made query $E^{-1}(K,Y)$ and obtained X as a result, for $K \in \{0,1\}^k$ and $X, Y \in \{0,1\}^n$. Now we let

$$\mathsf{Ch}_3^{E,B} \;=\; \Big|\{\, (K_0,K_1,K_2,P) \,:\, \exists\, Q,R,S\,[\,P \stackrel{K_0}{\to} Q \stackrel{K_1}{\to} R \stackrel{K_2}{\to} S\,]\,\}\Big| \;.$$

This is the number of *3-chains* created by B's queries. Here $K_0, K_1, K_2 \in \{0,1\}^k$ are keys, and $P, Q, R, S \in \{0,1\}^n$.

As the notation indicates, $\mathsf{Ch}_3^{E,B}$ is a number that depends on E and B. Regarding it as a random variable over the choice of E we have the following.

Lemma 12 *Let* $\alpha = \max(2e2^{k-n}, 2n+k)$. *Then* $\mathbf{E}[\mathsf{Ch}_3^{E,B}] < 2\alpha \cdot q^2$, *the expectation over* $E \stackrel{\$}{\leftarrow} \mathrm{Bloc}(k,n)$. ∎

We will show how to prove Lemma 6 using the above lemma, and then proceed to prove the lemma. We consider a game T^E that is parameterized by a blockcipher $E \in \mathrm{Bloc}(k,n)$. The adversary B is executed with oracles E, E^{-1} until it halts. Then K_0, K_1, K_2 are chosen at random from $\{0,1\}^k$, and flag *bad* is set if there exist P, Q, R, S such that $P \stackrel{K_0}{\to} Q \stackrel{K_1}{\to} R \stackrel{K_2}{\to} S$. Let

$$p^{E,B} \;=\; \Pr[B^{\mathrm{T}^E} \text{ sets } bad\,]\;,$$

the probability being over the random choices of K_0, K_1, K_2. Then for any $E \in \mathrm{Bloc}(k,n)$ we have

$$\begin{aligned} p^{E,B} \;&=\; \Pr\Big[\,\exists\, P,Q,R,S \,:\, P \stackrel{K_0}{\to} Q \stackrel{K_1}{\to} R \stackrel{K_2}{\to} S\,\Big] \;=\; \frac{|\{\, (K_0,K_1,K_2) \,:\, \exists\, P,Q,R,S \,:\, P \stackrel{K_0}{\to} Q \stackrel{K_1}{\to} R \stackrel{K_2}{\to} S\,\}|}{2^{3k}} \\ &\le\; \frac{\sum_P |\{\, (K_0,K_1,K_2) \,:\, \exists\, Q,R,S \,:\, P \stackrel{K_0}{\to} Q \stackrel{K_1}{\to} R \stackrel{K_2}{\to} S\,\}|}{2^{3k}} \;=\; \frac{\mathsf{Ch}_3^{E,B}}{2^{3k}}\;. \end{aligned}$$

Using this and Lemma 12 we have the following, where $\alpha = \max(2e2^{k-n}, 2n+k)$ and the expectation is over $E \xleftarrow{\$} \mathrm{Bloc}(k,n)$:

$$\Pr[\mathrm{T}_B \text{ sets } bad\,] \;=\; \mathbf{E}[p^{E,B}] \;\le\; \frac{\mathbf{E}[\mathsf{Ch}_3^{E,B}]}{2^{3k}} \;<\; \frac{2\alpha \cdot q^2}{2^{3k}} \;.$$

This concludes the proof of Lemma 6. It remains to prove Lemma 12.

For the proof of Lemma 12 we first need some definitions and another lemma. For $E \in \mathrm{Bloc}(k,n)$ and $Q, R \in \{0,1\}^n$ we let

$$\mathsf{Keys}^E(Q,R) \;=\; |\{\, K \;:\; E(K,Q) = R \,\}| \quad \text{and} \quad \mathsf{Keys}^E \;=\; \max_{Q,R}\{\mathsf{Keys}^E(Q,R)\} \;.$$

The first is the number of keys for which Q maps to R under E, and the second is the maximum value of $\mathsf{Keys}^E(Q,R)$ over all $Q, R \in \{0,1\}^n$. We clarify that no adversary is involved in this definition. Rather Keys^E is simply a number associated to any given blockcipher. Viewing it as a random variable over the choice of blockcipher we have the following.

Lemma 13 *Suppose $\beta \ge 2e2^{k-n}$ and $E \xleftarrow{\$} \mathrm{Bloc}(k,n)$. Then*

$$\Pr\left[\,\mathsf{Keys}^E \ge \beta\,\right] \;<\; 2^{2n+1-\beta} \qquad \blacksquare$$

We postpone the proof of the above, showing first how to exploit it to prove Lemma 12.

Proof of Lemma 12: For any $Q, R \in \{0,1\}^n$ we let

$$\begin{aligned}
\mathsf{Ch}_2^{E,B}(R) &= |\{\, (K_0, K_1, P) \;:\; \exists\, Q\, [P \xrightarrow{K_0} Q \xrightarrow{K_1} R\,] \,\}| \\
\mathsf{Ch}_1^{E,B}(Q) &= |\{\, (K_0, P) \;:\; P \xrightarrow{K_0} Q \,\}| \\
\mathsf{Ch}_0^{E,B}(R) &= |\{\, K_2 \;:\; \exists\, S\, [R \xrightarrow{K_2} S\,] \,\}| \;.
\end{aligned}$$

Then for any $E \in \mathrm{Bloc}(k,n)$ we have

$$\begin{aligned}
\mathsf{Ch}_3^{E,B} &= \sum_R \mathsf{Ch}_2^{E,B}(R) \cdot \mathsf{Ch}_0^{E,B}(R) \;\le\; \sum_R \left(\sum_Q \mathsf{Ch}_1^{E,B}(Q) \cdot \mathsf{Keys}^E(Q,R) \right) \cdot \mathsf{Ch}_0^{E,B}(R) \\
&\le \sum_R \left(\sum_Q \mathsf{Ch}_1^{E,B}(Q) \cdot \mathsf{Keys}^E \right) \cdot \mathsf{Ch}_0^{E,B}(R) \;=\; \mathsf{Keys}^E \cdot \left(\sum_Q \mathsf{Ch}_1^{E,B}(Q) \right) \cdot \left(\sum_R \mathsf{Ch}_0^{E,B}(R) \right) \\
&\le \mathsf{Keys}^E \cdot q \cdot q \;=\; q^2 \cdot \mathsf{Keys}^E \;.
\end{aligned}$$

Using the above and Lemma 13, we have the following, where the probability and expectation are both over $E \xleftarrow{\$} \mathrm{Bloc}(k,n)$:

$$\begin{aligned}
\mathbf{E}[\mathsf{Ch}_3^{E,B}] &= \mathbf{E}\left[\mathsf{Ch}_3^{E,B} \mid \mathsf{Keys}^E < \alpha\right] \cdot \Pr\left[\,\mathsf{Keys}^E < \alpha\,\right] + \mathbf{E}\left[\mathsf{Ch}_3^{E,B} \mid \mathsf{Keys}^E \ge \alpha\right] \cdot \Pr\left[\,\mathsf{Keys}^E \ge \alpha\,\right] \\
&< \mathbf{E}\left[\mathsf{Ch}_3^{E,B} \mid \mathsf{Keys}^E < \alpha\right] + \mathbf{E}\left[\mathsf{Ch}_3^{E,B} \mid \mathsf{Keys}^E \ge \alpha\right] \cdot 2^{2n+1-\alpha} \\
&\le q^2 \cdot \alpha + q^2 \cdot 2^k \cdot 2^{2n+1-\alpha} \;.
\end{aligned}$$

The last inequality above used the fact that Keys^E is always at most 2^k. Since $\alpha = \max(2e2^{k-n}, 2n+k) > 2$ we get

$$\mathbf{E}[\mathsf{Ch}_3^{E,B}] \;<\; q^2\alpha + q^2 \cdot 2 \;<\; 2\alpha \cdot q^2$$

as desired. ∎

Proof of Lemma 13: We claim that for any $Q, R \in \{0,1\}^n$

$$\Pr\left[\,\mathsf{Keys}^E(Q,R) \geq \beta\,\right] \;<\; 2^{1-\beta}\,. \tag{23}$$

We will justify this below. Now using this and the union bound we have

$$\Pr\left[\,\mathsf{Keys}^E \geq \beta\,\right] \;=\; \Pr\left[\,\exists\, Q,R \;:\mathsf{Keys}^E(Q,R) \geq \beta\,\right] \;\leq\; \sum_{Q,R} \Pr\left[\,\mathsf{Keys}^E(Q,R) \geq \beta\,\right] \;<\; 2^{2n}\cdot 2^{1-\beta}$$

as desired. We now prove (23) using a standard occupancy-problem approach [35, Section 3.1]. Let $b = \lceil \beta \rceil$. Then

$$\begin{aligned}
\Pr\left[\,\mathsf{Keys}^E(Q,R) \geq \beta\,\right] \;&=\; \sum_{i=b}^{2^k} \binom{2^k}{i} \cdot \left(\frac{1}{2^n}\right)^i \cdot \left(1-\frac{1}{2^n}\right)^{2^k-i} \;\leq\; \sum_{i=b}^{2^k} \binom{2^k}{i} \cdot \left(\frac{1}{2^n}\right)^i \\
&\leq\; \sum_{i=b}^{2^k} \left(\frac{2^k e}{i}\right)^i \cdot \left(\frac{1}{2^n}\right)^i \;=\; \sum_{i=b}^{2^k} \left(\frac{2^k e}{2^n i}\right)^i \;\leq\; \sum_{i=b}^{2^k} \left(\frac{2^k e}{2^n b}\right)^i
\end{aligned}$$

Let $x = (e/b)2^{k-n}$. The assumption $\beta \geq 2e2^{k-n}$ implies that $x \leq 1/2$. So the above is

$$= \sum_{i=b}^{2^k} x^i \;=\; x^b \cdot \sum_{i=0}^{2^k-b} x^i \;<\; x^b \cdot \sum_{i=0}^{\infty} x^i \;=\; x^b \cdot \frac{1}{1-x} \;\leq\; 2^{-b} \cdot \frac{1}{1-1/2} \;=\; 2^{1-b} \;\leq\; 2^{1-\beta}$$

as desired. ∎

E.3 Proof of Lemma 7

Let X be the random variable that is the number of queries by B that involve keys K_0, K_1, or K_2 in the experiment where we first run B with oracles E, E^{-1} for $E \xleftarrow{\$} \mathrm{Bloc}(k,n)$ and then pick $(K_0, K_1, K_2) \xleftarrow{\$} \mathrm{Distinct}_3^k$. Then

$$\Pr\left[\,B_{S,b}^G \text{ makes} \geq 3cq/K \text{ oracle queries}\,\right] \;=\; \Pr\left[\,X \geq 3cq/K\,\right].$$

This is because the view B sees when it is run by $B_{S,b}$ is exactly that of oracles consisting of a random block cipher and its inverse. Composing one of the permutations with S does not change anything since the composition of a fixed permutation with a random one is a random permutation. Now assume wlog that B always makes exactly q (distinct) oracle queries rather than at most q. Then

$$\begin{aligned}
\mathbf{E}[X] \;&=\; q \cdot \left[1 - \left(1-\frac{1}{K}\right)\left(1-\frac{1}{K-1}\right)\left(1-\frac{1}{K-2}\right)\right] \\
&=\; q \cdot \left[1 - \frac{K-1}{K}\frac{K-2}{K-1}\frac{K-3}{K-2}\right] \;=\; q \cdot \left[1 - \frac{K-3}{K}\right] \;=\; \frac{3q}{K}\,.
\end{aligned}$$

We can conclude via Markov's inequality.

F Elementary Proof for the CBC MAC

In this section we give a new proof for the security of the CBC MAC. Our proof is far simpler than the original one [5] or the recently devised one that offers an improved bound [7].

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 $M_i = M[n(i-1)+1\,..\,ni]$ denote the ith block of M. If $\pi\colon \{0,1\}^n \to \{0,1\}^n$ is a function and $M \in (\{0,1\}^n)^m$ is a string of n-bit blocks then we define $\mathrm{CBC}_\pi(M)$ as the result of running the following algorithm: $C \leftarrow 0^n$, for $i \leftarrow 1$ to m do $C \leftarrow \pi(C \oplus M_i)$; return C. Let $\mathrm{Perm}(n)$ denote the set of all permutations on $\{0,1\}^n$ and let $\mathrm{Func}(mn,n)$ 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_π^m be the restriction of CBC_π to the domain $\{0,1\}^{mn}$. Given an algorithm A that never repeats a query and asks queries that are strings of n-bit blocks, the adversary having oracle-access to a function $F\colon \{0,1\}^{mn} \to \{0,1\}^n$, let

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{cbc}}_{n,m}(A) &= \Pr[\pi \xleftarrow{\$} \mathrm{Perm}(n) :\ A^{\mathrm{CBC}_\pi^m(\cdot)} \Rightarrow 1] - \Pr[\rho \xleftarrow{\$} \mathrm{Func}(mn,n) :\ A^{\rho(\cdot)} \Rightarrow 1] \quad \text{and} \\
\mathbf{Adv}^{\mathrm{cbc}}_{n,m}(q) &= \max\left\{\mathbf{Adv}^{\mathrm{cbc}}_{n,m}(A)\right\}
\end{aligned}$$

where, for the second definition, 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 $q, m \geq 2$. We will show the following using games followed by a simple case analysis.

Theorem 14 [CBC MAC, conventional bound] $\mathbf{Adv}^{\mathrm{cbc}}_{n,m}(q) \leq m^2q^2/2^n$ *for any* $m, q \geq 2$, $n \geq 1$. ∎

Proof: Without loss of generality assume that A is deterministic and asks exactly q queries of m blocks each and that it never repeats a query. Refer to games C_0–C_9 defined in Figures 11 and 12. Let us begin by explaining the conventions used in these games. When M is string of blocks we write M_i for the ith block of M, as before, and we write $M_{1..j} = M[1\,..\,jn]$ for the first j blocks. In writing $\overline{\mathrm{domain}}(\pi)$ the complement is with respect to $\{0,1\}^n$. The value $\mathsf{defined}$ is an arbitrary string (it is never used), for concreteness, 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 $\mathtt{A}$, $\mathtt{B}$, and $\mathtt{C}$ 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. $\triangleright$ Game C_0 is obtained from game C_1 by dropping the assignment statements that immediately follow the setting of *bad*. $\triangleright$ Game C_1 is a realization of $\mathsf{CBC}^m[\mathrm{Perm}(n)]$ and game C_0 is a realization of $\mathrm{Func}(mn,n)$. The games use lazy sampling of a random permutation (as described in Section D.4) and the resampling idiom (as described in Section D.2). Games C_1 and C_0 are designed so that the Fundamental Lemma applies, so the advantage of A in attacking the CBC construction is at most $\Pr[A^{\mathrm{C}_0}$ sets *bad*$]$. $\triangleright$ $C_0 \to C_2$: This is a lossy transition that takes care of *bad* getting set at line 105, which clearly happens with probability at most $(0+1+\cdots+(mq-1))/2^n \leq 0.5\,m^2q^2/2^n$, so $\Pr[A^{\mathrm{C}_0}$ sets *bad*$] \leq \Pr[A^{\mathrm{C}_2}$ sets *bad*$] + 0.5\,m^2q^2/2^n$. $\triangleright$ $C_2 \to C_3$: Next notice that in game C_2 we never actually use the values assigned to π, all that matters is that we record that a value has been placed in the domain of π, and so game C_3 does just that, dropping a fixed value $\mathsf{defined}$ 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 D.2. The change is conservative. $\triangleright$ $C_3 \to C_4$: Now notice that in game C_3 the value returned to the adversary, although dropped into $T[M_1^s \cdots M_m^s]$, 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 C_4. The transition is conservative. $\triangleright$ $C_4 \to C_5$: Changing game C_4 to C_5 is by the de-randomization technique. First de-randomize line 417, letting each Z^s be a constant $\mathtt{Z}^s$. Thus we are not returning random values $Z^1,\ldots,Z^q$ to the adversary, we are returning constants

Initialize Game C_0 / $\boxed{\text{Game } C_1}$

$T[\varepsilon] \leftarrow 0^n$

procedure $F(M)$

$s \leftarrow s+1,\ \ M^s \leftarrow M$

$P \leftarrow \mathsf{Prefix}(M^1,\ldots,M^s),\ p \leftarrow |P|/n,\ \ C \leftarrow T[P]$

for $j \leftarrow p+1$ to m do

  $X \leftarrow C \oplus M_j$

  $C \xleftarrow{\$} \{0,1\}^n$

  if $C \in \mathrm{image}(\pi)$ then $bad \leftarrow \mathsf{true}$, $\boxed{C \leftarrow \overline{\mathrm{image}}(\pi)}$

  if $X \in \mathrm{domain}(\pi)$ then $bad \leftarrow \mathsf{true}$, $\boxed{C \leftarrow \pi[X]}$

  $\pi[X] \leftarrow C$

  $T[M_{1..j}] \leftarrow C$

return C

Initialize Game C_2

$T[\varepsilon] \leftarrow 0^n$

procedure $F(M)$

$s \leftarrow s+1,\ \ M^s \leftarrow M$

$P \leftarrow \mathsf{Prefix}(M^1,\ldots,M^s),\ p \leftarrow |P|/n,\ \ C \leftarrow T[P]$

for $j \leftarrow p+1$ to m do

  $X \leftarrow C \oplus M_j$

  $C \xleftarrow{\$} \{0,1\}^n$

  if $X \in \mathrm{domain}(\pi)$ then $bad \leftarrow \mathsf{true}$

  $\pi[X] \leftarrow C$

  $T[M_{1..j}] \leftarrow C$

return C

Initialize Game C_3

$T[\varepsilon] \leftarrow 0^n,\ \ \mathsf{defined} \leftarrow 0^n$

procedure $F(M)$

$s \leftarrow s+1,\ \ M^s \leftarrow M$

$P \leftarrow \mathsf{Prefix}(M^1,\ldots,M^s),\ p \leftarrow |P|/n,\ \ C \leftarrow T[P]$

for $j \leftarrow p+1$ to m do

  $X \leftarrow C \oplus M_j$

  $C \xleftarrow{\$} \{0,1\}^n$

  if $X \in \mathrm{domain}(\pi)$ then $bad \leftarrow \mathsf{true}$

  $\pi[X] \leftarrow \mathsf{defined}$

  $T[M_{1..j}] \leftarrow C$

return C

Initialize Game C_4

$T[\varepsilon] \leftarrow 0^n,\ \ \mathsf{defined} \leftarrow 0^n$

procedure $F(M)$

$s \leftarrow s+1,\ \ M^s \leftarrow M$

$P \leftarrow \mathsf{Prefix}(M^1,\ldots,M^s),\ p \leftarrow |P|/n,\ \ C \leftarrow T[P]$

for $j \leftarrow p+1$ to m do

  $X \leftarrow C \oplus M_j$

  if $X \in \mathrm{domain}(\pi)$ then $bad \leftarrow \mathsf{true}$

  $\pi[X] \leftarrow \mathsf{defined}$

  $C \leftarrow T[M_{1..j}] \xleftarrow{\$} \{0,1\}^n$

$Z^s \xleftarrow{\$} \{0,1\}^n$

return Z^s

Finalize Game C_5

$T[\varepsilon] \leftarrow 0^n,\ \ \mathsf{defined} \leftarrow 0^n$

for $s \leftarrow 1$ to q do

  $P \leftarrow \mathsf{Prefix}(\mathtt{M}^1,\ldots,\mathtt{M}^s),\ \ p \leftarrow |P|/n,\ \ C \leftarrow T[P]$

  for $j \leftarrow p+1$ to m do

    $X \leftarrow C \oplus \mathtt{M}^s_j$

    if $X \in \mathrm{domain}(\pi)$ then $bad \leftarrow \mathsf{true}$

    $\pi[X] \leftarrow \mathsf{defined}$

    $C \leftarrow T[\mathtt{M}^s_{1..j}] \xleftarrow{\$} \{0,1\}^n$

Finalize Game C_6

$T[\varepsilon] \leftarrow 0^n,\ \ \mathsf{defined} \leftarrow 0^n$

for $s \leftarrow 1$ to q do

  $P \leftarrow \mathsf{Prefix}(\mathtt{M}^1,\ldots,\mathtt{M}^s),\ \ p \leftarrow |P|/n,\ \ C \leftarrow T[P]$

  $X \leftarrow C \oplus \mathtt{M}^s_{p+1}$

  if $X \in \mathrm{domain}(\pi)$ then $bad \leftarrow \mathsf{true}$

  $\pi[X] \leftarrow \mathsf{defined}$

  $C \leftarrow T[\mathtt{M}^s_{1..p+1}] \xleftarrow{\$} \{0,1\}^n$

  for $j \leftarrow p+2$ to m do

    $X \leftarrow C \oplus \mathtt{M}^s_j$

    if $X \in \mathrm{domain}(\pi)$ then $bad \leftarrow \mathsf{true}$

    $\pi[X] \leftarrow \mathsf{defined}$

    $C \leftarrow T[\mathtt{M}^s_{1..j}] \xleftarrow{\$} \{0,1\}^n$

Finalize Game C_7

$T[\varepsilon] \leftarrow 0^n,\ \ \mathsf{defined} \leftarrow 0^n$

for $X \in (\{0,1\}^n)^+$ do $T[X] \xleftarrow{\$} \{0,1\}^n$

for $s \leftarrow 1$ to q do

  $P \leftarrow \mathsf{Prefix}(\mathtt{M}^1,\ldots,\mathtt{M}^s),\ \ p \leftarrow |P|/n$

  if $T[P] \oplus \mathtt{M}^s_{p+1} \in \mathrm{domain}(\pi)$ then $bad \leftarrow \mathsf{true}$

  $\pi[T[P] \oplus \mathtt{M}^s_{p+1}] \leftarrow \mathsf{defined}$

  for $j \leftarrow p+2$ to m do

    if $T[\mathtt{M}^s_{1..j-1}] \oplus \mathtt{M}^s_j \in \mathrm{domain}(\pi)$ then $bad \leftarrow \mathsf{true}$

    $\pi[T[\mathtt{M}^s_{1..j-1}] \oplus \mathtt{M}^s_j] \leftarrow \mathsf{defined}$

Finalize Game C_8

$T[\varepsilon] \leftarrow 0^n,\ \ \mathsf{defined} \leftarrow 0^n$

for $X \in (\{0,1\}^n)^+$ do $T[X] \xleftarrow{\$} \{0,1\}^n$

for $s \leftarrow 1$ to q do

  $\mathtt{P}^s \leftarrow \mathsf{Prefix}(\mathtt{M}^1,\ldots,\mathtt{M}^s),\ \ \mathtt{p}^s \leftarrow |\mathtt{P}^s|/n$

  if $T[\mathtt{P}^s] \oplus \mathtt{M}^s_{\mathtt{p}^s+1} \in \mathrm{domain}(\pi)$ then $bad \leftarrow \mathsf{true}$

  $\pi[T[\mathtt{P}^s] \oplus \mathtt{M}^s_{\mathtt{p}^s+1}] \leftarrow \mathsf{defined}$

  for $j \leftarrow \mathtt{p}^s+1$ to $m-1$ do

    if $T[\mathtt{M}^s_{1..j}] \oplus \mathtt{M}^s_{j+1} \in \mathrm{domain}(\pi)$ then $bad \leftarrow \mathsf{true}$

    $\pi[T[\mathtt{M}^s_{1..j}] \oplus \mathtt{M}^s_{j+1}] \leftarrow \mathsf{defined}$

Figure 11: Games used for the CBC MAC analysis. $\mathsf{Prefix}(M^1,\ldots,M^s)$, M_i, and $M_{1..j}$ are defined in the text.

Finalize	Game C_9
900 $T[\varepsilon] \leftarrow 0^n$	
901 for $X \in (\{0,1\}^n)^+$ do $T[X] \xleftarrow{\$} \{0,1\}^n$	
902 for $s \leftarrow 1$ to q do $\mathtt{P}^s \leftarrow \mathsf{Prefix}(\mathtt{M}^1, \ldots, \mathtt{M}^s)$, $\mathtt{p}^s \leftarrow \lvert\mathtt{P}^s\rvert/n$	
903 $bad \leftarrow$	
904 $T[\mathtt{P}^s] \oplus \mathtt{M}^s_{\mathtt{p}^s+1} = T[\mathtt{P}^r] \oplus \mathtt{M}^r_{\mathtt{p}^r+1}$	for some $1 \le r < s \le q$, or
905 $T[\mathtt{P}^s] \oplus \mathtt{M}^s_{\mathtt{p}^s+1} = T[\mathtt{M}^r_{1..i}] \oplus \mathtt{M}^r_{i+1}$	for some $1 \le r < s \le q$, $\mathtt{p}^r+1 \le i \le m-1$, or
906 $T[\mathtt{M}^s_{1..j}] \oplus \mathtt{M}^s_{j+1} = T[\mathtt{P}^r] \oplus \mathtt{M}^r_{\mathtt{p}^r+1}$	for some $1 \le r \le s \le q$, $\mathtt{p}^s+1 \le j \le m-1$, or
907 $T[\mathtt{M}^s_{1..j}] \oplus \mathtt{M}^s_{j+1} = T[\mathtt{M}^r_{1..i}] \oplus \mathtt{M}^r_{i+1}$	for some $1 \le r \le s \le q$, $\mathtt{p}^r+1 \le i \le m-1$, $\mathtt{p}^s+1 \le j \le m-1$, $i<j$ if $r=s$

Figure 12: Terminal game used in the analysis of the CBC MAC.

$\mathtt{Z}^1, \ldots, \mathtt{Z}^q$ instead. Since adversary A is deterministic there are corresponding queries $\mathtt{M}^1, \ldots, \mathtt{M}^q$ that it will ask. As these are constants, we need no longer consider the adversary as interacting with its oracle; in game C_5 we "hardwire in" the sequence of queries $\mathtt{M}^1, \ldots, \mathtt{M}^q$ (and responses $\mathtt{Z}^1, \ldots, \mathtt{Z}^q$ except that these have no influence on the game) that the adversary A would have asked. The "virtual-queries" $\mathtt{M}^1, \ldots, \mathtt{M}^q$ still have to be valid: each $\mathtt{M}^s$ is an mn-bit string different from all prior ones. We have that $\Pr[A^{C_4} \text{ sets } bad] \le \Pr[C_5 \text{ sets } bad]$. $\triangleright\, C_5 \to C_6$: Game C_6 unrolls the first iteration of the loop at lines 503–507. This transformation is conservative. $\triangleright\, C_6 \to C_7$: Game C_7 is a rewriting of game C_6 that omits mention of the variables C and X, directly using values from the T-table instead, these values now being chosen at the beginning of the game. The change is conservative. $\triangleright C_7 \to C_8$: Game C_8 re-indexes the for loop at line 705–707. The change is conservative. $\triangleright C_8 \to C_9$: Game C_9 restructures the setting of *bad* inside the loop at 802–808 to set *bad* in a single statement. Points were put into the domain of π at lines 805 and 808 and we checked if any of these points coincide with specified other points at lines 804 and 807. Running through the four possible types of collisions gives rise to the tests at lines 904–907. The change is conservative.

At this point we have only to bound $\Pr[C_9 \text{ sets } bad]$. We do this using the sum bound and a case analysis. Fix a line number among 904, 905, 906, and 907, and fix a corresponding r, i, s, j as specified on that line. We will show that the probability that the indicated equation holds is at most 2^{-n} and so, adding the loss incurred in transitioning from From this we can conclude that $\Pr[C_9 \text{ sets } bad] \le 0.5m^2q^2$ because the total number of equations under consideration is less than $0.5m^2q^2$. To see that this is the total number of equations note that we can name a candidate equation by choosing a pair of points $(r,i),(q,j) \in [1\,..\,q] \times [0\,..\,m-1]$ and then rejecting this pair if (r,i) does not proceed (s,j) in lexicographic order. Adding in the $0.5m^2q^2$ term we lost in going from game C_0 to game C_2, we will be done. The case analysis follows.

Line 904. We first bound $\Pr[T[\mathtt{P}^r] \oplus \mathtt{M}^r_{\mathtt{p}^r+1} = T[\mathtt{P}^s] \oplus \mathtt{M}^s_{\mathtt{p}^s+1}]$. If $\mathtt{P}^r = \mathtt{P}^s = \varepsilon$ then $\Pr[T[\mathtt{P}^r] \oplus \mathtt{M}^r_{\mathtt{p}^r+1} = T[\mathtt{P}^s] \oplus \mathtt{M}^s_{\mathtt{p}^s+1}] = \Pr[\mathtt{M}^r_1 = \mathtt{M}^s_1] = 0$ because $\mathtt{M}^r$ and $\mathtt{M}^s$, having only ε as a common block prefix, must differ in their first blocks. If $\mathtt{P}^r = \varepsilon$ but $\mathtt{P}^s \ne \varepsilon$ then $\Pr[T[\mathtt{P}^r] \oplus \mathtt{M}^r_{\mathtt{p}^r+1} = T[\mathtt{P}^s] \oplus \mathtt{M}^s_{\mathtt{p}^s+1}] = \Pr[\mathtt{M}^r_1 = T[\mathtt{P}^s] \oplus \mathtt{M}^s_{\mathtt{p}^s+1}] = 2^{-n}$ since the probability expression involves the single random variable $T[\mathtt{P}^s]$ 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 $\Pr[T[\mathtt{P}^r] \oplus \mathtt{M}^r_{\mathtt{p}^r+1} = T[\mathtt{P}^s] \oplus \mathtt{M}^s_{\mathtt{p}^s+1}] = 2^{-n}$ unless $\mathtt{P}^r = \mathtt{P}^s$, so assume that to be the case. Then $\Pr[T[\mathtt{P}^r] \oplus \mathtt{M}^r_{\mathtt{p}^r+1} = T[\mathtt{P}^s] \oplus \mathtt{M}^s_{\mathtt{p}^s+1}] = \Pr[\mathtt{M}^r_{\mathtt{p}^r+1} = \mathtt{M}^s_{\mathtt{p}^s+1}] = 0$ because $\mathtt{P}^r = \mathtt{P}^s$ is the *longest* block prefix that coincides in $\mathtt{M}^r$ and $\mathtt{M}^s$.

Line 905. We want to bound $\Pr[T[\mathtt{P}^s] \oplus \mathtt{M}^s_{\mathtt{p}^s+1} = T[\mathtt{M}^r_{1..i}] \oplus \mathtt{M}^r_{i+1}]$. If $\mathtt{P}^s = \varepsilon$ then $\Pr[T[\mathtt{P}^s] \oplus \mathtt{M}^s_{\mathtt{p}^s+1} = T[\mathtt{M}^r_{1..i}] \oplus \mathtt{M}^r_{i+1}] = \Pr[\mathtt{M}^s_{\mathtt{p}^s+1} = T[\mathtt{M}^r_{1..i}] \oplus \mathtt{M}^r_{i+1}] = 2^{-n}$ because it involves a single random value $T[\mathtt{M}^r_{1..i}]$. So assume that $\mathtt{P}^s \ne \varepsilon$. Then $\Pr[T[\mathtt{P}^s] \oplus \mathtt{M}^s_{\mathtt{p}^s+1} = T[\mathtt{M}^r_{1..i}] \oplus \mathtt{M}^r_{i+1}] = 2^{-n}$ unless $\mathtt{P}^s = \mathtt{M}^r_{1..i}$ 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 $\mathtt{P}^s = \mathtt{M}^r_{1..i}$ means that the longest prefix that $\mathtt{M}^s$ shares with $\mathtt{M}^r$ is $\mathtt{P}^s$ and so $\mathtt{M}^s_{\mathtt{p}^s+1} \neq \mathtt{M}^r_{\mathtt{p}^s+1}$.

Line 906. We must bound $\Pr[T[\mathtt{P}^r] \oplus \mathtt{M}^r_{\mathtt{p}^r+1} = T[\mathtt{M}^s_{1..j}] \oplus \mathtt{M}^s_{j+1}]$. It is at most by 2^{-n} as above.

Line 907. What is $\Pr[T[\mathtt{M}^s_{1..j}] \oplus \mathtt{M}^s_{j+1} = T[\mathtt{M}^r_{1..i}] \oplus \mathtt{M}^r_{i+1}]$. It is 2^{-n} unless $i = j$ and $\mathtt{M}^s_{1..j} = \mathtt{M}^r_{1..i}$. In that case $\mathtt{p}^s \geq j$ and $\mathtt{p}^r \geq i$, contradicting the allowed values for i and j at line 907. This completes the proof. ▮

G A Game-Based Proof for OAEP

Background. We recall the needed background for the asymmetric encryption scheme OAEP [8]. A trapdoor-permutation generator with associated security parameter k is a randomized algorithm $\mathcal{F}$ that 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

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

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 $\mathcal{D}^{G,H}_{f^{-1}}(Y)$ /* $Y \in \{0,1\}^k$ */
$R \xleftarrow{\$} \{0,1\}^\rho$	$X \leftarrow f^{-1}(Y)$
$S \leftarrow G(R) \oplus M,\; T \leftarrow H(S) \oplus R$	$S \leftarrow X[1..k-\rho],\; T \leftarrow X[k-\rho+1..k]$
$Y \leftarrow f(S \,\|\, T)$	$R \leftarrow H(S) \oplus T,\; M \leftarrow G(R) \oplus S$
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 $LR(\cdot,\cdot)$ which on input a pair M_0, M_1 of equal-length messages computes $Y \xleftarrow{\$} \mathcal{E}_f(M_b)$ and returns Y. The output of adversary is a bit b' and $\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{AE}}(A) = 2\Pr[b' = b] - 1$.

Theorem 15 [IND-CPA security of OAEP] *Let $\mathcal{F}$ be a trapdoor permutation generator with associated security parameter k, and let $\rho < k$ be an integer. 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*

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

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: The proof is based on games shown in Figures 13–15. For these games regard $\mathcal{F}$, k, and ρ as fixed; they are hardwired into the games. We have striven to makes steps between adjacent games small at the cost of a somewhat longer game chain.

For the analysis let $p_i = \Pr\left[\, P^A_i \Rightarrow \mathsf{true} \,\right]$ for $0 \leq i \leq 5$ (the Finalize procedure of these games outputs a boolean). ▷ Game P_0 perfectly mimics the game defining the security of $\mathsf{OAEP}^\rho[\mathcal{F}]$.

procedure $LR(M_0, M_1)$ — Game P_0

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

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

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

procedure $LR(M_0, M_1)$ — Game P_1

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

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

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

procedure $LR(M_0, M_1)$ — Game P_2

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

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

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

procedure $LR(M_0, M_1)$ — Game P_3

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

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

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

procedure $LR(M_0, M_1)$ — Game P_4

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

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

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

procedure $LR(M_0, M_1)$ — Game P_5

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

procedure $G(R)$
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 $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

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

Figure 13: Games used in the analysis of OAEP. Procedure **Initialize** is the same in all of these games: $(f, f^{-1}) \xleftarrow{\$} \mathcal{F}(k)$, $b \xleftarrow{\$} \{0,1\}$, return f. Procedure **Finalize**(out) is also the same: return $(out = b)$.

procedure $LR(M_0, M_1)$ — Game Q_0

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 $Y^* \leftarrow f(S^* \parallel T^*)$

procedure $G(R)$
010 if $H[S^*]$ and $R = R^*$ then $bad \leftarrow$ true
011 return $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

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

procedure $LR(M_0, M_1)$ — Game Q_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 $Y^* \leftarrow f(S^* \parallel T^*)$

procedure $G(R)$
110 if $H[S^*]$ and $R = R^*$ then $bad \leftarrow$ true
111 return $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

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

procedure $LR(M_0, M_1)$ — Game Q_2

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

procedure $G(R)$
210 if $H[S^*]$ and $R = R^*$ then $bad \leftarrow$ true
211 return $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

procedure $H(S)$
220 if $S = S^*$ then return $H[S^*] \leftarrow R^* \oplus T^*$
221 return $H[S] \xleftarrow{\$} \{0,1\}^\rho$

procedure $LR(M_0, M_1)$ — Game Q_3

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

procedure $G(R)$
310 if $H[S^*]$ and $R = R^*$ then $bad \leftarrow$ true
311 return $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

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

procedure $LR(M_0, M_1)$ — Game Q_4

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

procedure $G(R)$
410 if $H[S^*]$ and $R = R^*$ then $bad \leftarrow$ true
411 return $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

procedure $H(S)$
420 $H[S] \xleftarrow{\$} \{0,1\}^\rho$
421 if $S = S^*$ then $R^* \leftarrow H[S^*] \oplus T^*$
422 return $H[S]$

procedure $LR(M_0, M_1)$ — Game Q_5

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

procedure $G(R)$
510 if $R = H[S^*] \oplus T^*$ then $bad \leftarrow$ true
511 return $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

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

procedure $LR(M_0, M_1)$ — Game Q_6

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

procedure $G(R)$
610 if $\exists\, S$ s.t. $f(S \parallel T^*) = Y^*$ and $R = H[S] \oplus T^*$ then $bad \leftarrow$ true
611 return $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

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

procedure $LR(M_0, M_1)$ — Game Q_7

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

procedure $G(R)$
710 if $\exists\, S$ s.t. $f(S \parallel H[S] \oplus R) = Y^*$ then $bad \leftarrow$ true
711 return $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

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

Figure 14: Games used in the analysis of OAEP3 continued. Initialize and Finalize are as before.

procedure $LR(M_0, M_1)$ — Game $\hat{Q}_0$

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 $Y^* \leftarrow f(S^* \parallel T^*)$

procedure $G(R)$
010 if $\neg H[S^*]$ and $R = R^*$ then $bad \leftarrow$ true
011 return $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

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

procedure $LR(M_0, M_1)$ — Game $\hat{Q}_1$

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 $Y^* \leftarrow f(S^* \parallel T^*)$

procedure $G(R)$
110 if $\neg H[S^*]$ and $R = R^*$ then $bad \leftarrow$ true
111 return $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

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

procedure $LR(M_0, M_1)$ — Game $\hat{Q}_2$

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

procedure $G(R)$
210 if $R = R^*$ then $bad \leftarrow$ true
211 return $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

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

procedure $LR(M_0, M_1)$ — Procedures of Inverter I

900 return Y^*

procedure $G(R)$
910 if $\exists\, S$ s.t. $f(S \parallel H[S] \oplus R) = Y^*$ then
$bad \leftarrow$ true, $S^* \parallel T^* \leftarrow S \parallel H[S] \oplus R$
911 return $G[R] \xleftarrow{\$} \{0,1\}^{k-\rho}$

procedure $H(S)$
920 return $H[S] \xleftarrow{\$} \{0,1\}^\rho$

Figure 15: **Top:** Games used in the analysis of OAEP, continued. Procedures Initialize and Finalize are as before. **Bottom:** The inverting algorithm I, likewise with an identical Initialize procedure.

Thus

$$0.5 + 0.5\,\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{OAEP}^\rho[\mathcal{F}]}(A) \;=\; p_0 \;=\; p_1 + (p_0 - p_1) \;\le\; p_1 + \Pr[P_0^A \text{ sets } bad\,]\,,$$

the last step by the Fundamental Lemma. Since game P_0 chooses R^*, S^* at random, $\Pr[P_0^A \text{ sets } bad\,] \le q_G/2^\rho + q_H/2^{k-\rho}$. $\triangleright$ Game P_2 differs from game P_1 only in the setting of bad, so $p_1 = p_2$, and using the Fundamental Lemma again we have

$$p_1 \;=\; p_2 \;=\; p_3 + (p_2 - p_3) \;\le\; p_3 + \Pr[P_3^A \text{ sets } bad\,]\,.$$

$\triangleright$ In game P_4 the string GR^* is chosen but not referred to in responding to any oracle queries of the adversary. Thus P_4 is a conservative replacement for P_3, $p_3 = p_4$, and $\Pr[P_3^A \text{ sets } bad\,] = \Pr[P_4^A \text{ sets } bad\,]$. However, the bit b is not used in P_4, and hence $p_4 = 1/2$. In summary

$$p_3 + \Pr[P_3^A \text{ sets } bad\,] \;=\; p_4 + \Pr[P_4^A \text{ sets } bad\,] \;=\; 1/2 + \Pr[P_4^A \text{ sets } bad\,]\,.$$

Putting all this together we have

$$0.5\,\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{OAEP}^\rho[\mathcal{F}]}(A) - \frac{q_G}{2^\rho} - \frac{q_H}{2^{k-\rho}} \;\le\; \Pr[P_4^A \text{ sets } bad\,]\,. \tag{24}$$

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

$\Pr[P_4^A \text{ sets } bad\,] \le \Pr[P_5^A \text{ sets } bad\,] \le \Pr[Q_0^A \text{ sets } bad\,] + \Pr[\hat{Q}_0^A \text{ sets } bad\,] \le \Pr[Q_0^A \text{ sets } bad\,] + q_G/2^\rho$.

Above, we have split the analysis of game P_5 into the analysis of games Q_0 and $\hat{Q}_0$, the former being accomplished by the game chain at Figure 14 and the latter by the game chain of Figure 15. The second term is readily shown to be at most $q_G/2^\rho$ by way of the game chain at Figure 15. Going back to the analysis of game Q_0 in Figure 14, we have a series of conservative changes, giving Q_1, Q_2, Q_3, Q_4, Q_5 leading to

$$\Pr[Q_0^A \text{ sets } bad\,] \;=\; \Pr[Q_5^A \text{ sets } bad\,] \;\leq\; \Pr[Q_6^A \text{ sets } bad\,] \;=\; \Pr[Q_7^A \text{ sets } bad\,]\,.$$

To conclude the proof we design I so that

$$\Pr[Q_7^A \text{ sets } bad\,] \;\leq\; \mathbf{Adv}_{\mathcal{F}}^{\mathrm{owf}}(I)\,. \tag{25}$$

On input f, Y^*, inverter I runs A on input public key f, responding to its oracle queries via the procedures specified in the last box in Figure 15. When A halts, inverter I returns $S^* \parallel T^*$ if this has been defined. By comparison with game Q_7 we see that (25) is true, completing the proof. ∎