Physically Observable Cryptography

Silvio Micali Leonid Reyzin

Abstract

After a quarter century of impetuous development, complexity-theoretic cryptography has succeeded in finding rigorous definitions of security and provably secure schemes. In complexity-theoretic cryptography, however, computation has been "abstracted away": an adversary may attack a cryptographic algorithm essentially only by exchanging messages with it. Consequently, this theory fails to take into account the *physical* nature of actual computation, and cannot protect against *physical* attacks cleverly exploiting the information leakage inherent to the *physical* execution of any cryptographic algorithm. Such "physical observation attacks" *bypass* the impressive barrier of mathematical security erected so far, and successfully *break* mathematically impregnable systems. The great practicality and the inherent availability of physical attacks threaten the very relevance of complexity-theoretic security. Why erect majestic walls if comfortable underpasses will always remain wide open?

Responding to the present crisis requires *extending* the current mathematical models of cryptography to the physical setting. We do so by eliminating the mathematically convenient but physically unrealistic separation between the adversary and cryptographic computations. Specifically,

- We put forward *physically observable cryptography*: a powerful, comprehensive, and precise model for defining and delivering cryptographic security when an adversary has full (and indeed adaptive) access to any information leaked from the physical execution of cryptographic algorithms;
- We show that some of the basic theorems and intuitions of traditional cryptography no longer hold in a physically observable setting; and
- We construct schemes (such as pseudorandom generators and digital signatures) that are provably secure against *all* physical-observation attacks.

1 Introduction

"NON-PHYSICAL" ATTACKS. A *non-physical attack* against a cryptographic algorithm A is one in which the adversary is given some access to (at times even full control over) A's explicit inputs (e.g., messages and plaintexts) and some access to A'S outputs (e.g., ciphertexts and digital signatures). The adversary is also given full knowledge of A —except, of course, for the secret key— but absolutely no "window" into A's internal state during a computation: he may know every single line of A's code, but whether A's execution on a given input results in making more multiplications than additions, in using lots of RAM, or in accessing a given subroutine, remains totally unknown to him. In a non-physical attack, A's execution is essentially a *black box*. Inputs and outputs may be visible, but what occurs within the box cannot be observed at all.

For a long time, due to the lack of cryptographic theory and the consequent naive design of cryptographic algorithms, adversaries had to search no further than non-physical attacks for their devious deeds. (For instance, an adversary could often ask for and obtain the digital signature of a properly chosen message and then forge digital signatures at will.) More recently, however, the sophisticated reduction techniques of complexity-theoretic cryptography have shut the door to such attacks. For instance, if one-way functions exist, fundamental tools such as pseudorandom generation [15] and digital signatures [24, 21] can be implemented so as to be *provably* secure against *all* non-physical attacks.

Unfortunately, other realistic and more powerful attacks exist.

“PHYSICAL-OBSERVATION” ATTACKS. In reality, a cryptographic algorithm A must be run in a *physical* device P, and, quite outside of our control, the laws of Nature have something to say on whether P is reducible to a black box during an execution of A. Indeed, like for other physical processes, a real algorithmic execution generates all kinds of physical *observables*, which may thus fall into the adversary’s hands, and be quite informative at that. For instance, Kocher et al. [17] show that monitoring the electrical power consumed by a smart card running the DES algorithm [22] is enough to retrieve the very secret key! In another example, a series of works [23, 2] show that sometimes the electromagnetic radiation emitted by a computation, even measured from a few yards away with a homemade antenna, could suffice to retrieve a secret key.

PHYSICALLY OBSERVABLE CRYPTOGRAPHY. Typically, physical-observation attacks are soon followed by defensive measures (e.g., [9, 16]), giving us hope that at least *some* functions could be securely computed in our physical world. However, no rigorous theory currently exists that identifies *which* elementary functions need to be secure, and to *what extent*, so that we can construct complex cryptographic systems *provably* robust against *all* physical-observation attacks. This paper puts forward such a theory.

Our theory is not about “shielding” hardware (neither perfectly[1] nor partially[2]) but rather about how to *use partially shielded hardware in a provably secure manner.* That is, we aim at providing rigorous answers to questions of the following *relative* type:

(1) *Given a piece of physical hardware $\mathcal{P}$ that is guaranteed to compute a specific, elementary function $f(x)$ so that only some information $L_{\mathcal{P},f}(x)$ leaks to the outside,*

is it possible to construct

(2) *a physical pseudorandom generator, digital signature scheme, etc., provably secure against all physically-observing adversaries?*

Notice that the possibility of such constructions is far from guaranteed: hardware P is assumed “good” only for computing f, while any computation outside $\mathcal{P}$ (i.e., beyond f) is assumed to be fully observable by the adversary.

Answering such questions is important even with the current, incomplete knowledge about shielding hardware.[3] In fact, physically observable cryptography may properly *focus* the research in hardware protection by identifying which specific and elementary functions need to be protected and how much.

UNDERSTANDING A NEW WORLD. Physically observable cryptography is a new and fascinating world defying our traditional intuition. For example, such fundamental results as the equivalence of unpredictability and indistinguishability for pseudorandom generators [27] fail to hold. To explore this new world and achieve our goals, we

1. put forward a powerful and precise *model* for physically observable cryptography;
2. *identify* fundamental primitives;
3. *define* what secure reductions should mean; and
4. *exhibit* the first such reductions to build some desirable tools.

The model and the notions are center stage. The reductions, at least in this paper, are merely the best way to understand our new world by working in it.

BEYOND PHYSICALLY OBSERVABLE CRYPTOGRAPHY. Physically observable cryptography captures “the passive half” of a physical adversary. The “active half” consists of an adversary that can *tamper* with the content of a cryptographic device (e.g., flip a few bits in memory, or alter —somehow– the code of the algorithm itself). Attacks (e.g., [4, 8, 6, 5, 25]), defenses (e.g., [23, 20]), and models (e.g., [12]) in the active case are already under investigation, but their full understanding will ultimately depend on a full understanding of the passive case.

[1]Perfectly shielded hardware, so that all computation performed in it leaks nothing to the outside, might be impossible to achieve and is much more than needed.

[2]We are after a computational theory here, and constructing totally or partially shielded hardware is not a task for a computational theorist.

[3]Had complexity-theoretic cryptography waited for a proof of existence of one-way functions, we would be waiting still!

2 Intuition for Physically Observable Computation

Our model for physically observable (PO for short) computation is based on the following (overlapping)

Informal Axioms

1. *Computation, and only computation, leaks information*

   Information may leak whenever bits of data are accessed and computed upon. The leaking information actually depends on the particular operation performed, and, more generally, on the configuration of the currently active part of the computer. However, there is no information leakage in the absence of computation: unaccessed memory content is totally secure.

2. *Same computation leaks different information on different computers*

   Traditionally, we think of algorithms as carrying out computation. However, an algorithm is an abstraction: a set of general instructions, whose physical implementation may vary. In one case, an algorithm may be executed in a physical computer with lead shielding hiding the electromagnetic radiation correlated to the machine's internal state. In another case, the same algorithm may be executed in a computer with a sufficiently powerful inner battery hiding the power utilized at each step of the computation. As a result, the same elementary operation on 2 bits of data may leak different information: e.g., (for all we know) their XOR in one case and their AND in the other.

3. *Information leakage depends on the chosen measurement*

   While much may be observable at any given time, not all of it can be observed simultaneously (either for theoretical or practical reasons), and some may be only observed in a probabilistic sense (due to quantum effects, noise, etc.). The specific information leaked depends on the actual measurement made. Different measurements can be chosen (adaptively and adversarially) at each step of the computation.

4. *Information leakage is local*

   The information that may be leaked by a physically obervable device is the same in any execution with the same input, independent of the computation that takes place before the device is invoked or after it halts. In particular, therefore, *measurable information dissipates:* though an adversary can choose what information to measure at each step of a computation, information not measured is lost. Information leakage depends on the *past* computational history only to the extent that the *current* computational configuration depends on such history.

5. *All leaked information is efficiently computable from the computer's internal configuration.*

   Given an algorithm and its physical implementation, the information leakage is a polynomial-time computable function of (1) the algorithm's internal configuration, (2) the chosen measurement, and possibly (3) some randomness (outside anybody's control).

Remarks

As exepected, the real meaning of our axioms lies in the precise way we use them in our proofs. However, it may be worth to clarify here a few points about the meaningfulness of secure computation.

- *Some form of security for unaccessed memory is mandatory.*

  For instance, if a small amount of information leakage from a stored secret occurs at every unit of time (e.g., if a given bit becomes 51% predictable within a day) then a patient enough adversary will eventually reconstruct the entire secret.

- *Some form of locality for information leakage is mandatory.*

  The hallmark of modern cryptography has been constructing complex systems out of basic components. If the behavior of these components changed depending on the context, then no general principles for modular design can arise.

- *The restriction of a single adversarial measurement per step should not misinterpreted.*

  If two measurements M_1 and M_2 can be "fruitfully" performed one after the other, our model allows the adversary to perform the single measurement $M = (M_1, M_2)$.

- *The polynomial-time computability of leaked information should not be misinterpreted.*

  This efficient computability is quite orthogonal to the debate on whether physical (e.g., quantum) computation could break the polynomial-time barrier. Essentially, our model says that *the most* an adversary may obtain from a measurement is the entire current configuration of the cryptographic machine. And such configuration is computable in time linear in the number of steps executed by the crypto algorithm. For instance, if a computer stores a Hamiltonian graph but not its Hamiltonian tour, then performing a breadth-first search on the graph should not leak its Hamiltonian tour.

  (In any case, polynomial-time computable leakage is reasonable for a polynomial-time adversary. Should an adversary more powerful than polynomial-time be considered, then the power of the leakage function might also be increased "accordingly.")

Of course, we do not know that these axioms are "exactly true", but definitely hope to live in a world that "approximates" them to a sufficient degree: life without cryptography would be rather dull indeed!

3 Models and Goals of Physically Observable Cryptography

3.1 Computational Model

MOTIVATION. Axiom 1 guarantees that unaccessed memory leaks no information. Thus we need a computing device that clearly separates memory that is actively being used from memory that is not. The traditional Turing machine, which accesses its tape sequentially, is not a suitable computational device for the goal at hand: if the reading head is on one end of the tape, and the machine needs to read a value on the other end, it must scan the entire tape, thus accessing every single memory value. We thus must augment the usual Turing machine with random access memory, where each bit can be addressed individually and independently of other bits, and enable the resulting machine to copy bits between this random-access memory and the usual tape where it can work on them. (Such individual random access can be realistic implemented.)

Axiom 4 guarantees that the leakage of a given device is the same, independent of the computation that follows or preceeds it. Thus we need a model that can properly segregate one portion of a computation from another. The traditional notion of computation as carried out by a *single* Turing machine is inadequate for separating computation into multiple independent components, because the configuration of a Turing machine must incorporate (at a minimum) all future computation. To enable the modularity of physically observable cryptography, our model of computation will actually consist of *multiple* machines, each with its own physical protection, that may call each other as subroutines. In order to provide true independence, each machine must "see" its own memory space, independent of other machines (this is commonly known as virtual memory). Thus our multiple machines must be accompanied by a *virtual memory manager* that would provide for parameter passing while ensuring memory independence that is necessary for modularity. (Such virtual memory management too can be realistically implemented.)

FORMALIZATION WITHOUT LOSS OF GENERALITY. Let us now formalize this model computation (without yet specifying how information may leak). A detailed formalization is of course necessary for proofs to be meaningful.

This is particularly true in the case of a new theory, where no strong intuition has yet been developed. However, the particular choice of these details is not crucial. Our theorems are robust enough to hold also for different reasonable instantiations of this model.

ABSTRACT VIRTUAL-MEMORY COMPUTERS. An *abstract virtual-memory computer*, or abstract computer for short, consists of a *collection* of special Turing machines, which invoke each other as subroutines and share a special common memory. We call each member of our collection an *abstract virtual-memory Turing machine* (abstract VTM or simply VTM for short). We write $\mathcal{A} = (A_1, \ldots, A_n)$ to mean that an abstract computer $\mathcal{A}$ consists of abstract VTMs $A_1, \ldots, A_n$, where A_1 is a distinguished VTM: the one invoked first and whose inputs and outputs coincide with those of $\mathcal{A}$.

In addition to the traditional input, output, work and random tapes of a probabilistic Turing machine, a VTM has random access to its own *virtual address space* (VAS): an unbounded array of bits that starts at address 1 and goes on indefinitely.

The salient feature of an abstract virtual memory computer is that, while each VTM "thinks" it has its own individual VAS, in reality all of them, via a proper memory manager, share a single *physical adress space (PAS)*.

VIRTUAL-MEMORY MANAGEMENT. As it is common in modern operating systems, a single *virtual-memory manager* (working in polynomial time) supervises the mapping between individual VASes and the unique PAS. The virtual-memory manager also allows for parameter passing among the different VTMs.

When a VTM is invoked, from its point of view every bit in its VAS is initalized to 0, except for those locations where the caller placed the input. The virtual-memory manager ensures that the VAS of the caller is not modified by the callee, except for the callee's output values (that are mapped back into the caller's VAS).

Virtual-memory management is a well studied subject (outside the scope of cryptography), and we shall refrain from discussing it in detail. The only explicit requirement that we impose onto our virtual-memory manager is that it should only *remap* memory addresses, but never *access* their content. (As we shall discuss in later sections, this requirement is crucial to achieving cryptographic security in the *physical world*, where each memory access may result in a leakage of sensitive information to the adversary.)

ACCESSING VIRTUAL MEMORY. If A is a VTM, then we denote by m_A the content of A's VAS, and, for a positive integer j, we denote by $m_A[j]$ the bit value stored at location j. Every VTM has an additional, special *VAS-access tape*. To read the bit $m_A[j]$, A writes down j on the VAS-access tape, and enters a special state. Once A is in that state, the value $m_A[j]$ appears on the VAS-access tape at the current head position (the mechanics of this are the same as for an oracle query). To write a bit b in location j in its VAS, A writes down (j, b) on the VAS-access tape, and enters another special state, at which point $m_A[j]$ gets set to b.

Note that this setup allows each machine to work almost entirely in VAS, and use its work tape for merely computing addresses and evaluating simple gates.

INPUTS AND OUTPUTS OF A VTM. All VTM inputs and outputs are binary strings always residing in virtual memory. Consider a computation of a VTM A with an input i of length ℓ and an output o of length L. Then, at the start of the computation, the input tape of A contains 1^ℓ, the unary representations of the input length. The input i itself is located in the first ℓ bit positions of A's VAS, which will be read-only to A. At the end of the computation, A's output tape will contain a sequence of L addresses, $b_1, \ldots, b_L$, and o itself will be in A's VAS: $o = m_A[b_1] \ldots m_A[b_L]$. (The reason for input length to be expressed in unary is the preservation of the notion of polynomial running time with respect to the length of the *input tape*.)

CALLING VTMS AS SUBROUTINES. Each abstract VTM in the abstract virtual-memory computer has a unique name and a special *subroutine-call tape*. When a VTM A' makes a subroutine call to a VTM A, A' specifies where A' placed the input bits to A and where A' wants the output bits of A, by writing the corresponding addresses on this tape. The memory manager remaps locations in the VAS of A' to the VAS of A and vice versa. Straightforward details are provided in Appendix B.

3.2 Physical Security Model

PHYSICAL VIRTUAL-MEMORY COMPUTERS. We now formally define what information about the operation of a machine can be learned by the adversary. Note, however, that an abstract virtual-memory computer is an abstract object that may have different physical implementations. To model information leakage of any particular implementation, we introduce a *physical* virtual-memory computer (physical computer for short) and a *physical* virtual-memory Turing machine (physical VTM for short). A physical VTM $\mathcal{P}$ is a pair (L, A), where A is an abstract VTM and L is the *leakage function* described below. If $\mathcal{A} = (A_1, A_2, \ldots, A_n)$ is an abstract computer and $P_i = (L_i, A_i)$, then we call P_i a *physical implementation* of A_i and $\mathcal{P} = (P_1, P_2, \ldots P_n)$ a *physical implementation* of $\mathcal{A}$.

If a physical computer $\mathcal{P}$ is deterministic (or probabilistic, but Las Vegas), then we denote by $f_{\mathcal{P}}(x)$ the function computed by $\mathcal{P}$ on input x.

THE LEAKAGE FUNCTION. The leakage function L of a physical VTM $P = (L, A)$ is a function of three inputs, $L = L(\cdot, \cdot, \cdot)$.

- The first input is the current internal configuration C of A, which incorporates everything that is in principle measurable. More precisely, C is a binary string encoding (in some canonical fashion) the information of all the tapes of A, the locations of all the heads, and the current state (but *not* the contents of its VAS m_A). We require that only the "touched" portions of the tapes be encoded in C, so that the space taken up by C is polynomially related to the space used by T (not counting the VAS space).

- The second input M is the setting of the measuring apparatus, also encoded as a binary string (in essence, a specification of what the adversary chooses to measure).

- The third input R is a sufficiently long random string to model the randmoness of the measurement.

By specifying the setting M of its measuring apparatus, while A is in configuration C, the adversary will receive information $L(C, M, R)$, for a fresh random R (unknown to the adversary).

Because the adversary's computational abilities are restricted to polynomial time, we require $L(C, M, R)$ to be computable in time that is polynomial in the lengths of C and M.

THE ADVERSARY. Adversaries for different cryptographic tasks can be quite different (e.g., compare a signature scheme adversary to a pseudorandom generator distinguisher). However, we will augment all of the them in the same way with the ability to observe computation. We formalize this notion below.

Definition 1. The adversary F *observes* the computation of a physical computer $\mathcal{P} = (P_1, P_2, \ldots, P_n)$, where $P_i = (L_i, A_i)$ if:

1. F is invoked before each step of a physical VTM of $\mathcal{P}$, with configuration of F preserved between invocations.

2. F has a special read-only *name tape* that contains the name of the physical VTM P_i of $\mathcal{P}$ that is currently active.

3. At each invocation, upon peforming some computation, F writes down a string M on a special *observation tape*, and then enters a special state. Then the value $L_i(C, M, R)$, where P_i is the currently active physical VTM and R is a sufficiently long fresh random string unknown to F, appears on the observation tape, and $\mathcal{P}$ takes its next step.

4. This process repeats until $\mathcal{P}$ halts. At this point F is invoked again, with its name tape containing the index 0 indicating that $\mathcal{P}$ halted.

Notice that the above adversary is adaptive: while it cannot go back in time, its choice of what to measure in each step can depend on the results of measurements chosen in the past. Moreover, while at each step the adversary can measure only one quantity, to have a strong security model, we give the adversary all the time it needs to obtain the result of the previous measurement, decide what to measure next, and adjust its measuring apparatus appropriately.

Suppose the adversary F running on input x_F observes a physical computer $\mathcal{P}$ running on input $x_{\mathcal{P}}$, then $\mathcal{P}$ halts and produces output y_F, and then F halts and produces output $y_{\mathcal{P}}$. We denote this by

$$y_{\mathcal{P}} \leftarrow \mathcal{P}(x_{\mathcal{P}}) \rightsquigarrow F(x_F) \rightarrow y_F \,.$$

Note that F sees neither $x_{\mathcal{P}}$ nor $y_{\mathcal{P}}$ (unless it can deduce these values indirectly by observing the computation).

3.3 Assumptions, Reductions, and Goals

In addition to traditional, complexity-theoretic assumptions (e.g., the existence of one-way permutations), physically observable cryptography also has *physical assumptions*. Indeed, the very existence of a machine that "leaks less than complete information" is an assumption about the physical world. Let us be more precise.

Definition 2. A physical VTMs is *trivial* if its leakage function reveals its entire internal configuration[4] and *non-trivial* otherwise.

Fundamental Premise. *The very existence of a non-trivial physical VTM is a physical assumption.*

Just like in traditional cryptography, the goal of physically observable cryptography is to rigorously derive desirable objects from simple (physical and computational) assumptions. As usual, we refer to such rigorous derivations as *reductions*. Reductions are expected to use stated assumptions, *but should not themselves consist of assumptions!*

Definition 3. Let $\mathcal{P}'$ and $\mathcal{P}$ be physical computers. We say that $\mathcal{P}'$ *reduces to* $\mathcal{P}$ (alternatively, $\mathcal{P}$ *implies* $\mathcal{P}'$) if every non-trivial physical VTM of $\mathcal{P}'$ is also a physical VTM of $\mathcal{P}$.

4 Examples of Physically Observable Assumptions and Reductions

Having put forward the rules of physically observable cryptography, we now need to gain some experience in distilling its first assumptions and constructing its first reductions.

We start by quickly recalling basic the notions and facts from traditional cryptography that we use in this paper.

4.1 Traditional Building Blocks

We will assume familiarity with the traditional GMR notation (recalled in our Appendix A).

We also assume familiarity with the notions of one-way function [10] and permutation; of hardcore bits [7], with the fact that all one-way functions have a Goldreich-Levin hardcore bit [13], and with the notion of a *natural* hardcore bit (one that is simply a bit of the input, such as the last bit of the RSA input [3]). (All this traditional material is more thoroughly summarized in Appendix C.)

[4]It suffices, in fact, to reveal only the current state and the characters observed by the reading heads—the adversary can infer the rest by observing the leakage at every step.

4.2 Physically Observable One-Way Functions and Permutations

AVOIDING A LOGICAL TRAP. In traditional cryptography, the *existence* of a one-way function is currently an assumption, while the *definition* of a one-way function does not depend on any assumption. We wish that the same be true for physically observable one-way functions. Unfortunately, the most obvious attempt to defining physically observable one-way functions does not satisfy this requirement. The attempt consists of replacing the Turing machine T in the one-way function definition of Appendix C with a physical computer $\mathcal{P}$ observed by F. Precisely,

Definition Attempt: A *physically observable (PO) one-way functions* is a function $f : \{0,1\}^* \to \{0,1\}^*$ such that there exists a polynomial-time physical computer $\mathcal{P}$ that computes f and, for any polynomial-time adversary F, the following probability is negligible as a function of k:

$$\Pr[x \overset{R}{\leftarrow} \{0,1\}^k \,;\, y \leftarrow \mathcal{P}(x) \leadsto F(1^k) \to \mathit{state} \,;\, z \leftarrow F(\mathit{state}, y) : f(z) = y].$$

Intuitively, physically observable one-way functions should be "harder to come by" than traditional ones: unless no traditional one-way functions exist, we expect that only some of them may also be PO one-way. Recall, however, that *mathematically* a physical computer $\mathcal{P}$ consists

of pairs (L, A), where L is a leakage function and A an abstract VTM, in particular a single Turing machine. Thus, by setting L be the constant function 0, and $A = \{T\}$, where T is the Turing machine computing f, we obtain a non-trivial computer $\mathcal{P} = \{(L, A)\}$ that ensures that f is PO one-way as soon as it is traditionally one-way. The relevant question, however, is not whether such a computer can be mathematically defined, but whether it can be *physically* built. As we have said already, the mere existence of a non-trivial physical computer is in itself an assumption, and *we do not want the definition of a physically observable one-way function to rely on an assumption.* Therefore, we do not define what it means for a *function f* to be physically observable one-way. Rather, we define what it means for a particular *physical computer computing f* to be one-way.

We shall actually introduce, in order of strength, three physically observable counterparts of traditional one-way functions and one-way permutations.

MINIMAL ONE-WAY FUNCTIONS AND PERMUTATIONS. Avoiding the logical trap discussed above, the first way of defining one-way functions/permutations in the physically observable world is to say that $\mathcal{P}$ is a one-way function/permutation if it computes a permutation $f_{\mathcal{P}}$ that is hard to invert despite the leakage from $\mathcal{P}$'s computation. We call such physically observable one-way functions/permutations "minimal" in order to distinguish them from the other two counterparts we are going to discuss later on.

Definition 4. A polynomial-time deterministic physical computer $\mathcal{P}$ is *minimal one-way function* if for any polynomial-time adversary F, the following probability is negligible as a function of k:

$$\Pr[x \overset{R}{\leftarrow} \{0,1\}^k \,;\, y \leftarrow \mathcal{P}(x) \leadsto F(1^k) \to \mathit{state} \,;\, z \leftarrow F(\mathit{state}, y) : f_{\mathcal{P}}(z) = y].$$

Furthermore, if $f_{\mathcal{P}}$ is length-preserving and bijective, we call $\mathcal{P}$ a *minimal one-way permutation.*

DURABLE FUNCTIONS AND PERMUTATIONS. A salient feature of an abstract permutation is that the output is random for a random input. The following definition captures this feature, even in the presence of computational leakage.

Definition 5. A *durable* function (permutation) is a minimal one-way function (permutation) $\mathcal{P}$ such that, for any polynomial-time adversary F, the value $|p_k^P - p_k^R|$ is negligible in k, where

$$\begin{aligned} p_k^P &= \Pr[x \overset{R}{\leftarrow} \{0,1\}^k \,;\, y \leftarrow \mathcal{P}(x) \leadsto F(1^k) \to \mathit{state} : F(\mathit{state}, y) = 1] \\ p_k^R &= \Pr[x \overset{R}{\leftarrow} \{0,1\}^k \,;\, y \leftarrow \mathcal{P}(x) \leadsto F(1^k) \to \mathit{state} \,;\, z \overset{R}{\leftarrow} \{0,1\}^k : F(\mathit{state}, z) = 1] \end{aligned}$$

MAXIMAL ONE-WAY FUNCTIONS AND PERMUTATIONS. We now define physically observable one-way functions that leak nothing at all.

Definition 6. A *maximal* one-way function (permutation) is a minimal one-way function (permutation) $\mathcal{P}$ such that the leakage functions of its component physical VTMs are independent of the input x.

One can also define *statistically maximal* functions and permutations, where for any two inputs x_1 and x_2, the observed leakage from $\mathcal{P}(x_1)$ and $\mathcal{P}(x_2)$ is statistically close; and *computationally maximal* functions and permutations, where for any two inputs x_1 and x_2, what $\mathcal{P}(x_1)$ leaks is indistinguishable from what $\mathcal{P}(x_2)$ leaks. We postpone defining these formally.

4.3 Physically Observable Hardcore Bits

The essence of a traditional hardcore bit B is increasing the available "computational randomness." For instance, if f is a one-way permutation and x a k-bit random value, then $f(x) \circ B(x)$ produces $k+1$ computationally random bits. If we were not careful in defining physically observable hardcore bits, this crucial property would be lost: because the computation of the hardcore bit itself may also leak information. Consider the following

Definition Attempt: Let $\mathcal{P}$ and $\mathcal{B}$ be deterministic physical computers. We say that $\mathcal{B}$ is *PO hardcore* for $\mathcal{P}$ if $\mathcal{B}$ has a one-bit output and, for any polynomial-time adversary F, the value $p_k - 1/2$ is negligible in k, where

$$p_k = \Pr[x \overset{R}{\leftarrow} \{0,1\}^k \;;\; y \leftarrow \mathcal{P}(x) \rightsquigarrow F(1^k) \rightarrow \mathit{state} \;;\; b \leftarrow \mathcal{B}(x) \rightsquigarrow F(\mathit{state}, y) \rightarrow g : b = g]$$

At first glance this may seem like the right definition: it captures the difficulty of predicting $B(x)$ from y, even after observing how y and $B(x)$ are computed. However, this definition fails to address the problem that leakage of $\mathcal{B}(x)$ may reveal other information about x, thus possibly nullifying the protection that $\mathcal{P}$ provides. For example, this definition could be satisfied even if $\mathcal{B}(x)$ leaks y (since the adversary gets y anyway). Therefore, the net result of running $\mathcal{P}$ and then $\mathcal{B}$ on a truly random, k-bit, secret input x is that we have lost all the true unpredictability of the k bits of x, and gained just one computationally random bit: $\mathcal{B}(x)$!

Notice that modifying the above definitional attempt by calling for the existence of a physical computer $\mathcal{PB}$ that "jointly computes $\mathcal{P}$ and $\mathcal{B}$" would just be a second bad definitional attempt, since it leads to the already discussed logical trap of section 4.3.

The correct route is to use the second definitional attempt while avoiding its logical trap. However, this leads to a separate definition of a hardcore bit for each counterpart.

Definition 7. A deterministic physical computer $\mathcal{PB}$ is a *minimal one-way function with a hardcore bit* if there exist an (abstract) function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ and an (abstract) predicate $B : \{0,1\}^* \rightarrow \{0,1\}$ such that $f_{\mathcal{PB}}(x) = (f(x), B(x))$; without the last bit of output, $\mathcal{PB}$ is minimal one-way; and the last bit of output $B(x)$ is unpredictable, i.e., for any polynomial-time adversary F, the value $p_k - 1/2$ is negligible in k, where

$$p_k = \Pr[x \overset{R}{\leftarrow} \{0,1\}^k \;;\; (y,b) \leftarrow \mathcal{PB}(x) \rightsquigarrow F(1^k) \rightarrow \mathit{state} : b = F(\mathit{state}, y)]\,.$$

Definition 8. A minimal one-way function (permutation) with a hardcore bit $\mathcal{PB}$ is a *durable function (permutation) with a hardcore bit* if, when considered without the last output bit $\mathcal{B}$, it is durable.

Definition 9. A minimal one-way function (permutation) with a hardcore bit $\mathcal{PB}$ is a *maximal one-way function (permutation) with a hardcore bit* if the leakage functions of its component physical VTMs are independent of the input x.

4.4 The First Physically Observable Reduction

Reductions in our new environment are substantially more complex than in the traditional setting, and we have chosen a very simple one as our first example. Namely, we prove that minimal one-way permutations compose just like traditional one-way permutations.

Theorem 1. *A minimal one-way permutatation $\mathcal{P}$ implies a minimal one-way permutation $\mathcal{P}'$ such that $f_{\mathcal{P}'}(\cdot) = f_{\mathcal{P}}(f_{\mathcal{P}}(\cdot))$.*

Proof. See Appendix E. □

We wish to emphasize that, though simple, the proof of Theorem 1 illustrates exactly how our axioms for physically observable computation (formalized in our model) play out in our proofs.

5 Physically Observable Pseudorandomness

THE GOAL OF THE SECTION. In this last section, we point out (without proof) that fundamental constructions are indeed achievable in physically observable cryptography. Namely, we point out that physically observable pseudorandom generators are constructable. The choice of pseudorandom generation is quite natural for our first construction, because (1) it is conceptually simple (involving a single machine rather than multiple interrelated machines), and (2) it is easily implementable in traditional cryptography (as long as we are willing to rely on one-way permutations[5]).

More specifically, in this section we achieve two goals:

- We construct physically observable pseudorandom generators; and
- We identify which physically secure assumptions suffice for extending the well-known *iterative generator* of [7]:

iterate a one-way permutation on a random seed, outputting the hardcore bit at each iteration.

(Identifying the minimal physically observable assumption for pseudorandom generation is a much harder problem, best addressed after gaining a firmer grasp of the new field.)

DEFINITIONS. The definitions of PO indistinguishable and PO unpredictable generators are provided for completeness in Appendix D, but do not present particular technical challenges. They are derived from the corresponding traditional notions [7, 27] by allowing the adversary to observe the appropriate computation.

5.1 Statement of Our Results

A DIFFERENT WORLD. Physically observable pseudorandomness is drastically different from traditional one, and requires new intuition. The crucial property of traditional pseudorandomness is the equivalence of unpredictability and indistinguishability [27]. Surprisingly, this equivalence no longer holds in our new world.

Theorem 2. *A PO unpredictable generator is not necessarily PO indistinguishable.*

However, of course, PO indistinguishability still implies PO unpredictability.

A MORE DIFFICULT WORLD. In the physically observable world, pseudorandom generators are actually harder to build: just PO one-way permutations are no longer enough.

Theorem 3. *Using minimal one-way permutations in the iterative PRG construction does not imply a physically observable unpredictable generator.*

[5]It is known that one-way functions suffice for the existence of pseudorandom generators [15]; however, a much simpler construction —the one we wish to emulate first— relies on one-way permutations.

A WORKABLE WORLD. Nevertheless, it is possible to construct both unpredictable and indistinguishable generators from reasonable assumptions. Namely,

Theorem 4. *A durable function implies a PO unpredictable generator (with any polynomial expansion).*

Theorem 5. *A durable function with a hardcore bit implies a PO indistinguishable generator (with any polynomial expansion).*

A NEW ROLE FOR OLDER NOTIONS. Recall that a (traditional) hardcore bit of x is *natural* if it is a bit in some fixed location of x. Even though, in complexity-theoretic cryptography, the difference between a natural hardcore bit and any other hardcore bit seems insignificant, it turns out to be very important in physically observable cryptography.

Theorem 6. *Using a maximal one-way permutation $\mathcal{P}$ in the iterative PRG construction implies only a PO $\underline{\text{unpredictable}}$ generator. But using a maximal one-way permutation $\mathcal{P}$ for which $f_{\mathcal{P}}$ has a (traditional) natural hardcore bit implies a PO $\underline{\text{indistinguishable}}$ generator.*

In traditional cryptography, in light of the Goldreich-Levin construction [13], it seemed that finding natural hardcore bits of one-way functions became a nearly pointless endeavor (from which only minimal efficiency could be realized). However, Theorem 6 changes the state of affairs dramatically: *it provides new impetus for research on natural hardcore bits.* This shows how physically observable cryptography may actually have influence back on its predecessor.

PROOFS. In this extended abstract, only the proof sketches of our "enabling" Theorems 4 and 5 are provided (in Appendices F and G, respectively). The latter proof consists of a hybrid argument, but such arguments are more complex in our physically observable setting (in particular, rather than a traditional single "pass" through n intermediate steps —where the first is pseudorandom and the last is truly random— they now require two passes: from 1 to n and back).

The proofs of the "negative" results are not difficult once the right vein of counterexamples is identified, and will be provided in the full version of the paper.

6 Further Directions

ASSUMED RANDOMNESS VS. GENERATED RANDOMNESS. Our definitions in the physically observable model do not address the origin of the secret input x for a one-way function $\mathcal{P}$: according to the

definitions, nothing about x is observable by F before $\mathcal{P}$ starts running. One may take another view of a one-way function, however: one that includes the generation of a random input x as the first step. While in traditional cryptography this distinction seems unimportant, it is quite crucial in physically observable cryptography: the very generation of a random x may leak information about x. It is conceivable that some applications require a definition that includes the generation of a random x as part of the functionality of $\mathcal{P}$. However, we expect that in many instances it is possible to "hardwire" the secret randomness before the adversary has a chance to observe the machine, and then rely on pseudorandom generation.

SINGLE EVALUATION VS. MULTIPLE EVALUATIONS. Our definitions do not address the issue of computing the same (or similar) function on the same input multiple times. Of course, many traditional encryption and signature schemes do exactly that: the same secret key is input to every decryption and signing operation. What assumptions are needed for such schemes to work in the physically observable model is a question for future research.

SIGNATURE SCHEMES. This work provides a sufficient understanding of physically observable cryptography to see that signatures are possible in the new setting! Indeed, it is not too hard to see that minimal one-way functions imply one-time signature schemes (e.g., using construction of [18]), and durable functions imply even more efficient one-time signature schemes (e.g., using construction of [11]). Merkle trees [19] provide for a way to make multi-time (stateful) signatures out of one-time signatures. Note that, even though the Merkle tree construction requires hashing, no new notion of physically observable hashing is required, because the values being hashed are public, anyway. To avoid large secret storage, at key generation (which can be done away from the prying eyes of the adversary) the secret keys are all generated using a PO indistiguishable generator from a single secret seed x; the public keys are then computed and hashed to arrive at the Merkle tree. The Merkle tree can be stored publicly; the only secret that needs to be stored is the seed, which is updated after each new signature is issued. Further details will be provided in a future paper.

References

[1] *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, Seattle, Washington, 15–17 May 1989.

[2] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM side-channel(s). In *Cryptographic Hardware and Embedded Systems Conference (CHES '02)*, 2002.

[3] W. Alexi, B. Chor, O. Goldreich, and C. Schnorr. RSA and Rabin functions: Certain parts are as hard as the whole. *SIAM Journal on Computing*, 17(2):194–209, April 1988.

[4] Ross Anderson and Markus Kuhn. Tamper resistance — a cautionary note. In *The Second USENIX Workshop on Electronic Commerce*, pages 1–11, November 1996.

[5] Ross Anderson and Markus Kuhn. Low cost attacks on tamper resistant devices. In *Fifth International Security Protocol Workshop*, April 1997.

[6] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In Burton S. Kaliski, Jr., editor, *Advances in Cryptology—CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer-Verlag, 17–21 August 1997.

[7] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13(4):850–863, November 1984.

[8] D. Boneh, R. DeMillo, and R. Lipton. On the importance of checking cryptographic protocols for faults. In Walter Fumy, editor, *Advances in Cryptology—EUROCRYPT 97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 11–15 May 1997.

[9] Suresh Chari, Charanjit Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power analysis attacks. In Wiener [26], pages 398–412.

[10] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[11] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, Winter 1996.

[12] Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Tamper-proof security. Unpublished manuscript, 2003.

[13] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In ACM [1], pages 25–32.

[14] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2001.

[15] J. Håstad, R. Impagliazzo, L.A. Levin, and M. Luby. Construction of pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[16] Joshua Jaffe, Paul Kocher, and Benjamin Jun. United states patent 6,510,518: Balanced cryptographic computational method and apparatus for leak minimizational in smartcards and other cryptosystems, 21 January 2003.

[17] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Wiener [26], pages 388–397.

[18] Leslie Lamport. Constructing digital signatures from a one way function. Technical Report CSL-98, SRI International, October 1979.

[19] Ralph C. Merkle. A certified digital signature. In G. Brassard, editor, *Advances in Cryptology—CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer-Verlag, 1990, 20–24 August 1989.

[20] S. W Moore, R. J. Anderson, P. Cunningham, R. Mullins, and G. Taylor. Improving smartcard security using self-timed circuits. In *Asynch 2002*. IEEE Computer Society Press, 2002.

[21] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In ACM [1], pages 33–43.

[22] FIPS publication 46: Data encryption standard, 1977. Available from `http://www.itl.nist.gov/fipspubs/`.

[23] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In *Smart Card Programming and Security (E-smart 2001) Cannes, France*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210, September 2001.

[24] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the Twenty Second Annual ACM Symposium on Theory of Computing*, pages 387–394, Baltimore, Maryland, 14–16 May 1990.

[25] Sergei Skorobogatov and Ross Anderson. Optical fault induction attacks. In *Cryptographic Hardware and Embedded Systems Conference (CHES '02)*, 2002.

[26] Michael Wiener, editor. *Advances in Cryptology—CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*. Springer-Verlag, 15–19 August 1999.

[27] A. C. Yao. Theory and application of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois, 3–5 November 1982. IEEE.

A Minimal GMR Notation

- *Random assignments.*

  If S is a probability space, then "$x \leftarrow S$" denotes the algorithm which assigns to x an element randomly selected according to S. If F is a finite set, then the notation "$x \leftarrow F$" denotes the algorithm which assigns to x an element selected according to the probability space whose sample space is F and uniform probability distribution on the sample points.

- *Probabilistic experiments.*

  If $p(\cdot, \cdot, \cdots)$ is a predicate, the notation $\Pr[x \leftarrow S; y \leftarrow T; ... : p(x, y, \cdots)]$ denotes the probability that $p(x, y, \cdots)$ will be true after the ordered execution of the algorithms $x \leftarrow S$, $y \leftarrow T, ...$

B Calling VTMs as Subroutines

If A' wants to call A on the ℓ-bit input $i = m_{A'}[a'_1] \dots m_{A'}[a'_\ell]$, and if A returns an L-bit output on an ℓ-bit input, then the VTM A' has to write down on its subroutine-call tape

1. name of A;
2. a sequence of ℓ addresses in its own VAS, $a'_1, \dots, a'_\ell$;
3. a sequence of L distinct addresses in its own VAS, $b'_1, \dots, b'_L$.

Then A' enters a special "call" state and suspends its computation. At this point, the memory manager creates a new VAS for A, ensuring that

- location i in the VAS of A, for $1 \le i \le \ell$, is mapped to the same PAS location as a'_i in the VAS of A', and
- all the other locations in the VAS of A map to blank and unassigned PAS locations. (Namely, in case of nested calls, any VAS location of any machine in the call stack —i.e., A', the caller of A', etc.— must not map to these PAS locations.)

Then the computation of A begins in the "start" state, with a blank work tape and the input tape containing 1^ℓ. When A halts, the memory manager remaps location b'_i, for $1 \le i \le L$, in the VAS of A' to the same PAS location as b_i in the VAS of A. (Recall that b_i appears on the output tape of A, and that all the b'_i are distinct, so the remapping is possible.) The output value of A is taken to be the value $o = m_{A'}[b'_1] \dots m_{A'}[b'_\ell]$, and A' resumes operation.

Note that the input locations a'_i in the caller's VAS do not need to be distinct; nor do the output locations b_i in the callee's VAS. Therefore, it is possible that the memory manager will need to map two or more locations in a VTM's VAS to the same PAS location (indeed, because accessing memory may cause leakage, remapping memory is preferable to copying it). When a VAS location is written to, however, the memory manager ensures that only one PAS location is affected: if the VAS location is mapped to the same physicall address as another VAS location, it gets remapped to a fresh physical address.

C Traditional Building Blocks

- *One-way functions* [10]. A one-way function is a function $f : \{0,1\}^* \to \{0,1\}^*$ such that there exists a polynomial-time Turing machine T that computes f and, for any polynomial-time adversary F, the following probability is negligible as a function of k:
$$\Pr[x \overset{R}{\leftarrow} \{0,1\}^k \ ; \ y \leftarrow T(x) \ ; \ z \leftarrow F(1^k, y) : f(z) = y].$$
- *One-way permutations.* A one-way permutation is a one-way function that is length-preserving and bijective.
- *One-way permutations are composable.* For all n, if f is a one-way permutation, so is f^n.
- *One-way permutations are chainable.* For all $0 \le i < n$ and for all polynomial-time adversary F, the following probability is negligible as a function of k:
$$\Pr[x \overset{R}{\leftarrow} \{0,1\}^k \ ; \ y \leftarrow f^n(x) \ ; \ (i, \mathit{state}) \leftarrow F(y) \ ; \ z \leftarrow F(\mathit{state}, f^{n-i}(x)) : f^{i+1}(z) = y].$$
- *Hardcore Bits* [7]. Let f be a one-way permutation, and B a polynomial-time computable predicate. We say that B is a hardcore bit (for f) if, for any polynomial-time adversary F, the value $|p_k - 1/2|$ is negligible in k, where
$$p_k = \Pr[x \overset{R}{\leftarrow} \{0,1\}^k \ ; \ y \leftarrow f(x) \ ; \ g \leftarrow F(1^k, y) : g = B(x)].$$
The first hardcore bit was exhibited for the discrete-log function [7].

- *All one-way permutations have a hardcore bit* [13]. Let f be a one-way permutation, and let $r_1, \ldots, r_k$ be a sequence of random bits. Then, informally, the randomly chosen predicate B_r is overwhelmingly likely a hardcore bit for f, where B_r is the predicate so defined: for a k-bit string $x = x_1 \cdots x_k$, $B_r(x) = x_1 \times r_1 + \ldots x_k \times r_k$ mod 2.

- *Natural hardcore bits*. We call a hardcore bit B natural if $B(x)$ returns the bit in a fixed location of the bit string x. Some specific one-way permutations possess natural hardcore bits —for instance, the last bit is hardcore for the RSA function [3].

- *Unpredictable pseudorandom generators* [7]. Let p be a polynomially bounded function such that $p(k) > k$ for all positive integers k. Let G be a polynomial-time deterministic algorithm that, on a k-bit input, produces a $p(k)$-bit output. We say that G is an *unpredictable pseudorandom generator with expansion* p if for any polynomial-time adversary F, the value $|p_k - 1/2|$ is negligible in k, where

  $$p_k \quad = \quad \Pr[(i, state) \leftarrow F(1^k) \; ; \; x \overset{R}{\leftarrow} \{0,1\}^k \; ; \; y \leftarrow G(x) : F(state, y_1 \ldots y_i) = y_{i+1}] \, ,$$

  (where y_j denotes the j-th bit of y).

- *Indistinguishable pseudorandom generators* [27]. Unpredictable pseudorandom generators are provably the same as indistinguishable generators, defined as follows. Let G, again, be a polynomial-time deterministic algorithm that, on a k-bit input, produces a $p(k)$-bit output. We say that G is an *indistinguishable pseudorandom generator with expansion* p if for any polynomial-time adversary F, the value $|p_k^G - p_k^R|$ is negligible in k, where

  $$\begin{aligned} p_k^G &= \Pr[x \overset{R}{\leftarrow} \{0,1\}^k \; ; \; y \leftarrow G(x) : F(state, y) = 1] \\ p_k^R &= \Pr[x \overset{R}{\leftarrow} \{0,1\}^k \; ; \; y \leftarrow G(x) \; ; \; z \overset{R}{\leftarrow} \{0,1\}^{p(k)} : F(state, z) = 1] \end{aligned}$$

  Because every unpredictable pseudorandom generator is indistinguishable and vice versa, we refer to them as simply “pseudorandom generators” or “PRGs.”

- *The iterative PRG construction* [7].

  For any one-way permutation f, the following is a pseudorandom generator:

  choose a random secret seed, and iterate f on it, outputting the hardcore bit at each iteration.

D Definitions of Physically Observable Pseudorandom Generators

Let us map both unpredictable and undistinguishable generators to our new setting.

UNPREDICTABILITY. The corresponding physically observable notion replaces “unpredictability of bit $i+1$ from the first i bits” with “unpredictability of bit $i+1$ from the first i bits and the leakage from their computation.”

Definition 10. Let p be a polynomially bounded function such that $p(k) > k$ for all positive integers k. Let $\mathcal{G}$ be a polynomial-time deterministic physical computer that, on a k-bit input, produces $p(k)$-bit output, one bit at a time (i.e., it writes down on the output tape the VAS locations of the output bits in left to right, one a time). Let $\mathcal{G}^i$ denote running $\mathcal{G}$ and aborting it after it outputs the i-th bit. We say that $\mathcal{G}$ is a *PO unpredictable generator with expansion* p if for any polynomial-time adversary F, the value $|p_k - 1/2|$ is negligible in k, where

$$p_k = \Pr[(i, state_1) \leftarrow F(1^k) \; ; \; x \overset{R}{\leftarrow} \{0,1\}^k \; ; \; y_1 y_2 \ldots y_i \leftarrow G^i(x) \rightsquigarrow F(state_1) \rightarrow state_2 : \\ F(state_2, y_1 \ldots y_i) = y_{i+1}] \, ,$$

(where y_j denotes the j-th bit of $y = \mathcal{G}(x)$).

INDISTINGUISHABILITY. The corresponding physically observable notion replaces "indistinguishability" by "*indistinguishability in the presence of leakage.*" That is, a polynomial-time adversary F first observes the computation of a pseudorandom string, and then receives either that same pseudorandom string or a totally independent random string, and has to distinguish between the two cases.

Definition 11. Let p be a polynomially bounded function such that $p(k) > k$ for all positive integers k. We say that a polynomial-time deterministic physical computer $\mathcal{G}$ is a *PO indistinguishable generator with expansion* p if for any polynomial-time adversary F, the value $|p_k^G - p_k^R|$ is negligible in k, where

$$\begin{aligned} p_k^G &= \Pr[x \overset{R}{\leftarrow} \{0,1\}^k \; ; \; y \leftarrow \mathcal{G}(x) \rightsquigarrow F(1^k) \rightarrow \mathit{state} : F(\mathit{state}, y) = 1] \\ p_k^R &= \Pr[x \overset{R}{\leftarrow} \{0,1\}^k \; ; \; y \leftarrow \mathcal{G}(x) \rightsquigarrow F(1^k) \rightarrow \mathit{state} \; ; \; z \overset{R}{\leftarrow} \{0,1\}^{p(k)} : F(\mathit{state}, z) = 1] \end{aligned}$$

E Proof of Theorem 1

Proof of Theorem 1. Let $\mathcal{P} = (P_1, \ldots, P_n)$ be a minimal one-way permutation, where each physical VTM P_i is a pair consisting of a leakage function L_i and an abstract VTM A_i. Intuitively, $\mathcal{P}'$ simply runs $\mathcal{P}$ twice (i.e., it calls twice P_1 which is the entry point to all other P_i of $\mathcal{P}$). Formally this is accomplished by creating a new trivial physical VTM P_0 that twice calls P_1. Define P_0 to be the (new) trivial physical VTM (L_0, A_0), where L_0 is the trivial leakage function (i.e., the one leaking everything) and A_0 is the following abstract VTM:

> On input a k-bit value x in VAS locations $1, 2, \ldots, k$, call $A_1(x)$ as a subroutine specifying that the returned value y_1 be placed in VAS locations $k+1, k+2, \ldots, 2k$.
>
> Then, run A_1 again on input y_1, specifying that the returned value y_2 be placed in VAS locations $2k+1, 2k+2, \ldots, 3k$.
>
> Output y_2 (i.e., place the addresses $2k+1, 2k+2, \ldots, 3k$ on the output tape) and halt.

Consider now the physical computer $\mathcal{P}'$ that has the above specified P_0 as the first machine, together with all the machines of $\mathcal{P}$, that is, $\mathcal{P}' = (P_0, P_1, \ldots, P_n)$. It is clear that $\mathcal{P}'$ is implied by $\mathcal{P}$ and that $\mathcal{P}'$ computes $f_{\mathcal{P}}(f_{\mathcal{P}}(x))$ in polynomial time. Therefore, all that is left to prove is the "one-wayness" of $\mathcal{P}'$: that is, that the adversary will not succeed in finding z such that $f_{\mathcal{P}}(f_{\mathcal{P}}(z)) = y_2$ as described in the experiment of Definition 4. This is done by the following elementary reduction.

Because $f'_{\mathcal{P}} = f^2_{\mathcal{P}}$ is a permutation, finding any inverse z of y_2 means finding the original input x. Suppose there exists an adversary F' that succeeds in finding x after observing the computation of $\mathcal{P}'$ and receiving $y_2 = f_{\mathcal{P}}(f_{\mathcal{P}}(x))$. Then, in the usual style of cryptographic reductions, we derive a contradiction by showing that there exists another adversary F that (using F') succeeds in finding x after observing the computation of $\mathcal{P}$ and receiving $y_1 = f_{\mathcal{P}}(x)$.

$F(1^k)$ "virtually" executes $\mathcal{P}'(x) \rightsquigarrow F'(1^k)$: at at each (virtual) step of $\mathcal{P}'$, F receives the measurement that F' wishes to make, and responds with the appropriately distributed leakage. In so doing, however, F is only entitled to observe $\mathcal{P}(x)$ once.

Recall that F' expects to observe a five-stage computation:

1. P_0 prepares the tapes for the subroutine call to $P_1(x)$
2. P_1 and its subroutines compute $y_1 = f_{\mathcal{P}}(x)$
3. P_0 prepares the tapes for the subroutine call to $P_1(y_1)$
4. P_1 and its subroutines compute $y_2 = f_{\mathcal{P}}(y_1)$
5. P_0 places the address of y_2 on the output tape

During Stage 1, F can very easily answer any measurement made by F'. In fact, (1) because P_0 trivial, any measurement of F' should be answered with the entire configuration of P_0 and, (2) because P_0 just reassigns VAS pointers without reading or handling any secret VAS bits, each of P_0's configurations can be computed by F from 1^k (which is given to F as an input).

After so simulating Stage 1, F starts observing the computation of $\mathcal{P}(x)$. At each step, F is allowed a measurement M, and the measurement it chooses coincides with the one F' wants, thus F can easily forward to F' the obtained result. At the end of Stage 2 F receives $y_1 = f_{\mathcal{P}}(x)$ (which it stores but does not forward to F').

Stage 3 is as easily simulated as Stage 1.

During Stage 4, F "virtually runs" physical computer $\mathcal{P}(y_1)$, that is, it runs the corresponding abstract computer $A(y_1)$. At each step, if A_i is the active machine in configuration C, and F' specifies a measurement M, then F returns the leakage $L_i(C, M, R)$ for a random R.

Upon simulating stage 5 (as easily as Stage 1), F computes $y_2 = f_{\mathcal{P}}(y_1)$, and gives it to F' to receive x.

$\square$

The Axioms in Action

Let us show that all our axioms for physically observable computation are already reflected in the very simple proof of Theorem 1.

- The simulation of Stages 1, 3, and 5 relies on Axiom 1. In fact, F can simulate P_0 only because P_0 does not access the VAS, and unaccessed VAS leaks no information.

- The simulation of Stage 2 relies on Axiom 4. Specifically, we relied on the fact that $\mathcal{P}(x)$ run in "isolation" has the same leakage distribution as $\mathcal{P}(x)$ "introduced" by P_0, and more generally in the "context" of $\mathcal{P}'$.

  Similarly, also the simulation of Stage 4 relies on Axiom 4: the leakage of running $\mathcal{P}$ from scratch on a string y_1 is guaranteed to be the same as the leakage of running $\mathcal{P}$ after y_1 is computed as $\mathcal{P}(x)$.

- The simulation of Stage 4 relies on Axiom 5. In fact F was not observing the real $\mathcal{P}$, but rather was running $\mathcal{P}$ on its own and simulating $\mathcal{P}$'s leakage which therefore had to be polynomial-time computable.

- Axiom 2 is implicitly relied upon. In a sense, Axiom 2 says that the same algorithm can have different leakage distributions, depending on the different physical machines which run it. In particular, therefore, it makes the very existence of a physically observable one-way permutation *plausible*. Trivial machines that leak everything certainly exist, and using them to compute $f(x)$ from x would make it easy to find an inverse of $f(x)$. Thus, if the f's leakage were the same for every machine, PO one-way permutations would not exist, making the entire theory moot.

- Axiom 3 has been incorporated into the model, by giving adversary F' the power of choosing its own measurements at every step of the computation.

F Proof Sketch of Theorem 4

Proof sketch of Theorem 4. Let $\mathcal{P}$ be a durable function. To construct out of $\mathcal{P}$ a PO unpredictable generator $\mathcal{G}$ with expansion p, we will mimic the iterative construction of Blum and Micali [7], combining it with the Goldreich-Levin [13] hardcore bit. For this construction, it is crucial that the bits are output in reverse order, as in [7]: namely, that all computations of $\mathcal{P}$ take place before any Goldreich-Levin bits are computed (because we are not assuming a secure machine for computing Goldreich-Levin bits, and hence the hardcore bit computation will leak everything about its inputs).

Specifically, given a random seed (x_0, r), to output $\ell = p(|x| + |r|)$ bits, $\mathcal{G}$ computes $x_1 = \mathcal{P}(x_0)$, $x_2 = \mathcal{P}(x_1)$, $\ldots$, $x_\ell = \mathcal{P}(x_{\ell-1})$, and outputs $b_1 = x_{\ell-1} \cdot r, b_2 = x_{\ell-2} \cdot r, \ldots, b_\ell = x_0 \cdot r$, where "$\cdot$" denotes the dot product modulo 2 (i.e., the Goldreich-Levin bit). Formally, this is done by constructing a trivial physical VTM to "drive" this process and compute the hardcore bits; we omit the details here, as they are straightforward and similar to the proof of Theorem 1.

To prove that this is indeed unpredictable, consider first a simpler situation. Starting from a random x, compute $\mathcal{P}(x)$, letting the adversary observe the computation. Now provide the adversary with $\mathcal{P}(x)$ and a random r, and have it predict $r \cdot x$. If the adversary is successful with probability significantly better than $1/2$, then it is successful for significantly more than 50% of all possible values for r. Thus, we can run it for multiple different values of r, and reconstruct x using the same techniques as in [13], which would contradict the minimal one-wayness of $\mathcal{P}$. Note that even though we use the adversary to predict $x \cdot r$ for multiple values r, the adversary needs to observe $\mathcal{P}(x)$ only *once*. This is because the observation takes place before r is provided to the adversary, and therefore the choices made by the adversary during the observation are independent of r.

The actual generator, of course, is more complex than the above scenario. To prove that bit b_i is unpredictable, first note that $x_{\ell-i}$ is not computable by the adversary even if the adversary observes the computation until b_{i-1} is output (this can be shown by a properly constructed hybrid argument based on the definition of durable). Also observe that the previous bits, $b_1, \ldots, b_{i-1}$ are all efficiently computable from $x_{\ell-i+1} = \mathcal{P}(x_{\ell-i})$, which the adversary receives anyway when it observes the computation of b_{i-1}. Thus, proving that b_i is unpredictable reduces to the simpler case already proven above. $\square$

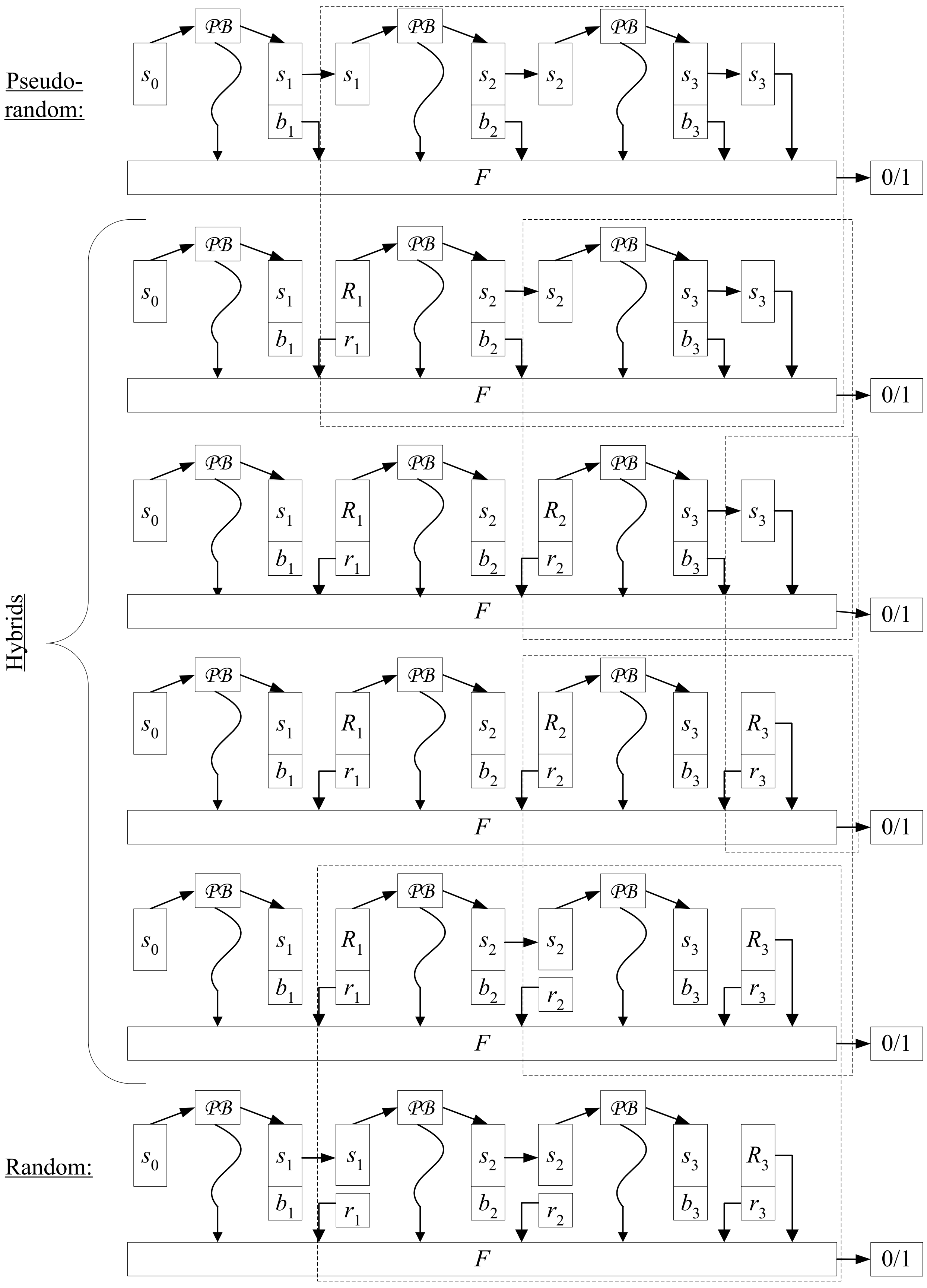
G Proof Sketch of Theorem 5

Proof sketch of Theorem 5. Let $\mathcal{PB}$ be a durable function with a hardcore bit. To consturct out of $\mathcal{PB}$ a PO indistinguishable generator $\mathcal{G}$ with expansion p, we will simply mimic the iterative construction of [7]: to generate $\ell = p(k)$ pseudorandom bits on a k-bit input seed s_0, compute $(s_1, b_1) = \mathcal{PB}(s_0)$ and output b_1; then compute $(s_2, b_2) = \mathcal{PB}(s_1)$ and output b_3, and so on for ℓ times (note that there is no need here to output bits in reverse order). Formally, this is done by constructing a trivial physical VTM to "drive" this process; we omit the details here, as they are straightforward and similar to the proof of Theorem 1.

The proof that the resulting $\mathcal{G}$ is PO indistinguishable is by a hybrid argument, somewhat similar to (but more complex than) the hybrid argument that shows that unpredictability implies indistinguishability for traditional pseudorandom generators ([27]; see [14] for an excellent exposition). We recall the essence of that hybrid argument here to prepare for the more complex hybrid argument in our case. Suppose that the pseudorandom string $b_1 b_2 \ldots b_\ell$ is unpredictable (i.e., b_i cannot be predicted given $b_1 \ldots b_{i-1}$), but can be distinguished from a truly random string $r_1 r_2 \ldots r_\ell$. Then consider the $\ell - 1$ "hybrid" strings, the i-th string h_i being $b_1 b_2 \ldots b_{\ell-i} r_{\ell-i+1} r_{i+2} \ldots r_\ell$ (then $h_0 = b_1 b_2 \ldots b_\ell$ and $h_\ell = r_1 r_2 \ldots r_\ell$). If the i-th string can be distinguished from the $(i+1)$-th, then the bit $b_{\ell-i+1}$ can be distinguished from $r_{\ell-i+1}$ in the presence of $b_1 b_2 \ldots b_{\ell-i}$, i.e., the bit $b_{\ell-i+1}$ can be predicted (which is a contradiction).

In our proof, our hybrids are not just strings. Rather, because we have to also deal with the leakage, our hybrids are *processes*. The pseudorandom process consists of running $\mathcal{PB}(s_0)$ and then giving the adversary $b_1 \ldots b_\ell$ (actually, we can give s_ℓ as well, it will not change the proof, just like in the hybrid argument above). The random process consists of running $\mathcal{PB}(s_0)$ and then giving the adversary random bits $r_1 \ldots r_\ell$ (and a random R_ℓ in place of s_ℓ). There are $2(n-1)$ hybrid processes, depcited in the figure on page 19 and defined as follows.

The i-th hybrid process H_i for $i \leq n$ is the process that runs $\mathcal{PB}(s_0)$ to obtain (s_1, b_1); then replaces (s_1, b_1) with new random (R_1, r_1), outputs r_1 and runs $\mathcal{PB}(R_1)$ to obtain (s_2, b_2); then replaces (s_2, b_2) with new random (R_2, r_2), outputs r_2 and runs $\mathcal{PB}(R_2)$ to obtain (s_3, b_3); it continues in this manner until it replaces (s_i, b_i) with (R_i, r_i), at which point it proceeds properly as $\mathcal{G}$ would. Thus, H_0 is the pseudorandom process. The i-th hybrid process H_i for $i > n$ is the same as the $(2n-1-i)$-th hybrid process, except that it always outputs truly random bits and a random


Pseudo-random:
Hybrids
Random:
$\mathcal{PB}$
s_0
s_1
b_1
R_1
r_1
s_2
b_2
R_2
r_2
s_3
b_3
R_3
r_3
F
0/1

R_l (even where $(2n-1-i)$-th hybrid process would have output a pseudorandom bit b_j and the actual s_l) (see figure on page 19). Thus, H_{2n-1} is the random process from the definition of PO indistinguishability.

If any two consecutive hybrids were distinguishable, then the output of $\mathcal{PB}$ on a random input would be distinguishable from random, by a simple reduction, which we omit here (but depict via large rectangles in the figure on page 19). This implies (by the same argument as in abstract cryptography) that either the s output of $\mathcal{PB}$ is distinguishable from random, or the b output is predictable in the presence of s. This is a contradiction, however, because $\mathcal{PB}$ is a durable function with a hardcore bit. $\square$