Public Randomness in Cryptography*

Amir Herzberg[1] and Michael Luby[2]

[1] I.B.M. T.J. Watson, Yorktown Heights, NY 10598
[2] International Computer Science Institute, U.C. Berkeley, Berkeley, California 94704

Abstract. The main contribution of this paper is the introduction of a formal notion of public randomness in the context of cryptography. We show how this notion affects the definition of the security of a cryptographic primitive and the definition of how much security is preserved when one cryptographic primitive is reduced to another. Previous works considered the public random bits as a part of the input, and security was parameterized in terms of the total length of the input. We parameterize security solely in terms of the length of the private input, and treat the public random bits as a separate resource. This separation allows us to independently address the important issues of how much security is preserved by a reduction and how many public random bits are used in the reduction.

To exemplify these new definitions, we present reductions from weak one-way permutations to one-way permutations with strong security preserving properties that are simpler than previously known reductions.

1 Introduction

Over the years, randomness has proved to be a powerful algorithmic resource, i.e. randomized algorithms that are simpler, or more efficient, or both, than any known deterministic algorithm have been developed for a variety of problems. Randomness has also proved to be a powerful resource in the construction of cryptographic primitives based on other primitives, e.g., the randomized reductions from weak one-way functions to one-way functions and the reductions from one-way functions to pseudo-random generators. The source of randomness used in these reductions is typically *public*, in the sense that the random bits are accessible to all parties enacting the primitive and to any adversary trying to break the primitive. However, up till now, the distinction between the private part of the input and the public random bits has been blurred.

The main contributions of this paper are to formally introduce the notion of public randomness, to introduce appropriate generalizations of the definitions of

* Research supported in part by National Science Foundation operating grant CCR-9016468 and grant No. 89-00312 from the United States-Israel Binational Science Foundation (BSF)

E.F. Brickell (Ed.): Advances in Cryptology - CRYPTO '92, LNCS 740, pp. 421-432, 1993.

cryptographic primitives that use public randomness and, perhaps most importantly, to modify the definition of what it means to reduce one cryptographic primitive to another by allowing public randomness to be used in the reduction. In terms of generalizing the definition of cryptographic primitives to include public randomness, the main advantage is that the security of a primitive can now be parameterized, as it should be, solely in terms of the length of the private part of the input, and not at all in terms of the public random bits. In terms of reductions, the main advantage is that we can now separately consider the two issues of how much security is preserved by the reduction and how much public randomness is used in the reduction.

As particular examples of how a primitive that uses public randomness can be defined, we extend the definitions of one-way functions and pseudo-random generators to include public random bits. Generalizations along the same lines for many other cryptographic primitives can be made, including those related to public key cryptography.

As particular examples of how the new definitions of reductions using public randomness work, we provide reductions that use public randomness from weak one-way permutations to one-way permutations. Following [1], our prime concern is the security preserving properties of the reduction, i.e., how much of the security of the weak one-way permutation is transferred to the one-way permutation. However, unlike [1], we consider the security as a function solely of the length of the private input, which does not include the public random bits. We show reductions that preserve security in a very strong sense, which is stronger than that of the reduction due to [1] (under the new definitions). We begin with a very simple reduction (much simpler than that found in [1]), which uses a large number of public random bits. Through a sequence of increasingly intricate reductions, we converge on a reduction that is a slight modification of the reduction due to [1]. Both the reduction of [1] and our improvement use only a linear number of public random bits.

Another simple reduction from a weak one-way permutation to a one-way permutation was developed recently and independently by Phillips [2]. Phillips showed that his reduction preserves security somewhat better than the reduction of [1], when considering the randomness as a part of the input. However, our new definitions of security preserving reductions with public randomness reveal that Phillips' reduction actually preserves security as well as our reductions, i.e. much better than [1]. Phillips' reduction uses more public random bits ($O(n \log(n))$) than our best reduction.

A full development and details of this work can be found in [3].

2 Definitions

2.1 Basic Notation

If S is a set then $\sharp S$ is the number of elements in S. Let x and y be bit strings. We let $\|x\|$ denote the length of x. We let $\langle x, y \rangle$ denote the sequence of two strings

x followed by y, and when appropriate we also view this as the concatenation of x and y. When $\langle x, y \rangle$ is the input to a function f, we write this as $f(x, y)$. We let x_i denote the i^{th} bit of x. Let $x \in \{0,1\}^n$ and let $S \subset \{1, \ldots, n\}$. We let x_S denote the subsequence of bits in x indexed by S, e.g. $x_{\{1,\ldots,i\}}$ denotes the first i bits of x, $x_{\{i+1,\ldots,n\}}$ denotes all but the first i bits of x, and thus $x = \langle x_{\{1,\ldots,i\}}, x_{\{i+1,\ldots,n\}} \rangle$.

If x and y are bit strings, each of length l, then $x \oplus y$ is the vector sum mod 2 (i.e. bit wise parity) of x and y, i.e. $(x \oplus y)_i = (x_i + y_i) \bmod 2$.

An $m \times n$ bit matrix x is indicated by $x \in \{0,1\}^{m \times n}$. We write $x_{i,j}$ to refer to the (i, j) in x. We can also view x as a sequence $x = \langle x_1, \ldots, x_m \rangle$ of m strings, each of length n, where x_i is the i^{th} row of the matrix, or as a string $x \in \{0,1\}^{mn}$ of length mn, which is the concatenation of the rows of the matrix.

If a is a number, then $|a|$ is the absolute value of a, $\lceil a \rceil$ is the smallest integer greater than or equal to a, $\log(a)$ is the logarithm base two of a. If a number is an input to or an output of an algorithm, the assumption is that it is presented in binary notation.

In general, we use capital letters to denote random variables and random events. When S is a set we use the notation $X \in_U S$ to mean that X is a random variable uniformly distributed in S, and $x \in_U S$ indicates that x is a fixed element of S chosen uniformly.

2.2 Public Randomness

A source of random bits is *public* for a primitive if it can be read by all parties enacting the primitive and by any adversary trying to break the primitive. The public random string is always chosen uniformly. We use ";" to keep the public random string separated from other strings in a list of strings, e.g. if y is the value of the public random string and x is the input to some function f, then we write $f(x; y)$ to indicate the evaluation of f on input x with respect to y (note that the value of f depends both on the input x and on the public random string y), and we write $\langle f(x; y); y \rangle$ to indicate the pair of strings $f(x; y)$ and y.

Although the public random bits are known to an adversary, it turns out that these bits often plays a crucial role in ensuring that the primitive is secure.

2.3 Security

The security of a primitive quantifies how secure the primitive is against attacks by an adversary trying to break the primitive. The important question to consider is "What does the security measure?" Intuitively, the security of a primitive is a measure of the minimal computational resources needed by any adversary to break the primitive. There are two natural computational resources we consider; the maximal total time T that the adversary runs and the success probability δ of the adversary. Both T and δ are stated with respect to a given input instance to the adversary, and their definitions are primitive dependent.

A trivial strategy to increase the success probability δ is to run the adversary again. This doubles the running time, but also almost doubles the success

probability (especially if it is low). This motivates us to simplify the analysis by comparing only the ratios between the success probabilities and the running times of different adversaries. An additional simplification is to consider the ratio between the success probability $\delta(n)$ and the maximal running time $T(n)$, both over all private inputs of length n. Without much loss in generality, we hereafter assume that an adversary A always runs for the same amount of time $T(n)$ on all inputs parameterized by n.

Definition (achievement ratio): The achievement ratio of an adversary A for a primitive f is defined as $\frac{\delta(n)}{T(n)}$, where $T(n)$ is the running time of A and $\delta(n)$ is the success probability of A for f on private inputs of length n.

Definition (breaking adversary and security): An adversary A is $R_f(n)$-breaking for a primitive f if the achievement ratio $\frac{\delta(n)}{T(n)}$ of A for f satisfies $\frac{\delta(n)}{T(n)} \geq R_f(n)$ for infinitely many $n \in \mathcal{N}$. The primitive f is $(1 - R_f(n))$-secure if there is no $R_f(n)$-breaking adversary for f.

Intuitively, 0-secure means totally insecure, whereas 1-secure means totally secure. We would like the primitive to be harder to break than it is to use. For example, suppose f is a $(1 - R_f(n))$-secure one-way function, where for all constants c, $R_f(n) < \frac{1}{n^c}$ for sufficiently large n. Then f can be computed in polynomial time, whereas a polynomial time algorithm can only invert f with inverse polynomial probability for finitely many values of $n \in \mathcal{N}$.

Allowing the security of a primitive to be parameterized is important because different implementations of primitives may achieve different levels of security, which may offer different tradeoffs between efficiency and security. We note that inverse polynomial security (e.g. $R_f(n) = \frac{1}{n^{100}}$) means that the primitive may be broken by a polynomial adversary, so we expect that many applications would require higher security. For example, it may be that a particular function f is a $(1 - R_f(n))$-secure one-way function, where $R_f(n) = \frac{1}{2^{\log^2 n}}$, or even better with $R_f(n) = \frac{1}{2^{\sqrt{n}}}$. The statement that f is secure with respect to either of these bounds is quantifiably stronger than the statement that it is secure with respect to an inverse polynomial bound. On the other hand, for any function computable in time $T(n)$ there is an inverting adversary that runs in $T(n) \cdot 2^n$ time, and thus there is no $(1 - \frac{1}{T(n) \cdot 2^n})$-secure one-way function.

2.4 Primitives with Public Randomness

Definition (standard function): A function $f(x; y)$ is called a *standard function* with *length relationship* $\|x\| = n$, $\|y\| = l(n)$, $\|f(x; y)\| = m(n)$ if

- $f(x; y)$ is computable in polynomial time.
- If $\|x\| = n$ then $\|y\| = l(n)$ and $\|f(x; y)\| = m(n)$, where both $l(n)$ and $m(n)$ are polynomial in n.

We now give the definitions of primitives using public randomness.

Definition (one-way function with public random bits): Let $f(x; y)$ be a standard function with length relationship $\|x\| = n$, $\|y\| = l(n)$, $\|f(x; y)\| =$

$m(n)$. Let $X \in_{\mathcal{U}} \{0,1\}^n$ and $Y \in_{\mathcal{U}} \{0,1\}^{l(n)}$. The *success probability* of adversary A for f is

$$\delta(n) = \Pr_{X,Y}[f(A(f(X;Y);Y);Y) = f(X;Y)].$$

The *running time* $T(n)$ of adversary A for f is the maximum over all $x \in \{0,1\}^n$ and $y \in \{0,1\}^{l(n)}$ of the running time of A on input $\langle f(x;y);y\rangle$. Then, f is a $(1-R_f(n))$-secure one-way function if there is no $R_f(n)$-breaking adversary for f.

Definition (one-way permutation with public random bits): Let $f(x;y)$ be a standard function with length relationship $\|x\| = n, \|y\| = l(n), \|f(x;y)\| = m(n)$. Then, f is a $(1-R_f(n))$-secure one-way permutation if f is a $(1-R_f(n))$-secure one-way function and $m(n) = n$ and x is uniquely determined by $f(x;y)$ and y.

Definition (pseudo-random generator with public random bits): Let $g(x;y)$ be a standard function with length relationship $\|x\| = n$, $\|y\| = l(n)$, $g(x) = m(n)$, where $m(n) > n$. The *stretching parameter* of $g(x;y)$ is $m(n) - n$. Let $X \in_{\mathcal{U}} \{0,1\}^n$, $Y \in_{\mathcal{U}} \{0,1\}^{l(n)}$ and $Z \in_{\mathcal{U}} \{0,1\}^{m(n)}$. The *success probability* (distinguishing probability) of adversary A for g is

$$\delta(n) = \Pr_{X,Y}[A(g(X;Y);Y) = 1] - \Pr_{Z,Y}[A(Z;Y) = 1].$$

The *running time* $T(n)$ of adversary A for g is the maximum over all $z \in \{0,1\}^{m(n)}$ and $y \in \{0,1\}^{l(n)}$ of the running time of A on input $\langle z;y\rangle$. Then, g is a $(1-R_g(n))$-secure pseudo-random generator if there is no $R_g(n)$-breaking adversary for g.

Example : To exemplify the difference between the traditional definition of a one-way function and the definition introduced here with public randomness, consider the subset sum problem. A one-way function based on the difficulty of this problem can be defined in two ways; without public random bits and with public random bits. Let $b \in \{0,1\}^n$ and let $a \in \{0,1\}^{n\times n}$. In the first definition without public random bits the function is

$$f(a,b) = \langle a, \sum_{i=1}^{n} b_i \cdot a_i \rangle.$$

The security is parameterized by the input length $N = n^2 + n$. In the second definition, a is the public random string and the function is defined as

$$f(b;a) = \sum_{i=1}^{n} b_i \cdot a_i.$$

In this case, the security is parameterized by the length of the private input b, which is simply n. Note that in both cases, the actual security of f is based on exactly the same thing, i.e. when a and b are chosen uniformly then given a and $\sum_{i=1}^{n} b_i \cdot a_i$ there is no fast adversary that can find on average a $b' \in \{0,1\}^n$ such that $\sum_{i=1}^{n} b'_i \cdot a_i = \sum_{i=1}^{n} b_i \cdot a_i$. The only difference is how the security is

parameterized. Intuitively, security should be parameterized in terms of what is hidden from the adversary, and not in terms of the overall amount of randomness available to the function. The first definition parameterizes the security in terms of the overall amount of randomness available to the function, i.e. security is parameterized in terms of the length of b *plus* the length of a. The parameter of security in the second definition is the length of b, where b is what is really secret.

Intuitively, a weak one-way function f is a function such that it is hard to find an inverse of $f(x)$ for some significant but perhaps not very large fraction of $x \in \{0,1\}^n$ (the 'hard set'). (In contrast, for a one-way function it is hard to find an inverse of $f(x)$ for all but an insignificant fraction of the $x \in \{0,1\}^n$.) We only give the traditional definition (not using public randomness); the definition using public randomness is straightforward.

Definition (weak one-way function): Let $f(x)$ be a standard function with length relationship $\|x\| = n$, $\|f(x)\| = l(n)$. The *weakness parameter* of f is a function $s(n)$ such that $s(n) \geq \frac{1}{n^c}$ for some constant c. The time bound and success probability of an adversary A for f are defined exactly the same way as for a one-way function. An adversary A is $R_f(n)$-breaking for $s(n)$-weak f if there is a subset H_n of $\{0,1\}^n$ of measure at least $s(n)$ such that $R_f(n) \leq \frac{\delta_H(n)}{T_H(n)}$, where $\delta_H(n)$ is the average success probability over $H(n)$ and $T_H(n)$ is the maximal running time over $H(n)$. A function f is a $(1 - R_f(n))$-secure $s(n)$-weak one-way function if there is no $R_f(n)$-breaking adversary for $s(n)$-weak f.

Example : Define $f(x,y) = xy$, where $x, y \in \{2, \ldots, 2^n - 1\}$. The problem of inverting $f(x,y)$ consists of finding $x', y' \in \{2, \ldots, 2^n - 1\}$ such that $x'y' = xy$. Let $X, Y \in_U \{2, \ldots, 2^n - 1\}$ be independent random variables. On average, $f(X,Y)$ is rather easy to invert. For instance, with probability $\frac{3}{4}$, XY is an even number, in which case setting $x' = 2$ and $y' = \frac{XY}{2}$ inverts $f(X,Y)$. However, with probability approximately $1/n^2$ both X and Y are prime n-bit numbers. If there is no adversary that can factor the product of a pair of random n-bit prime numbers in time $\frac{1}{R_f(2n)}$ on average then f is a $(1 - R_f(2n))$-secure $\frac{1}{n^2}$-weak one-way function.

3 Reductions

All of the results presented in this paper involve a reduction ¿from one type of cryptographic primitive to another. In this section, we give a formal definition of reduction. We only define a reduction in the case when both cryptographic primitives are standard functions.

Central to the definition of a reduction is the notion of an oracle Turing machine.

Definition (oracle Turing machine): An *oracle Turing machine* is a randomized Turing machine S whose behavior is not fully specified. The behavior is not fully specified in the sense that S, in the course of its execution, interactively makes calls (hereafter described as oracle calls) to and receives corresponding outputs from an algorithm that is not part of the description of S. We let S^A

denote the fully specified Turing machine described by S using algorithm A to compute the oracle calls.

Note that although the running time of S is not defined, the running time of S^A is defined. Also, if A is a Turing machine then so is S^A.

Let f be a generic instance of the first primitive, where $f(x)$ is a standard function with length relationship $\|x\| = n$ and $\|f(x)\| = l(n)$. Let $X \in_{\mathcal{U}} \{0,1\}^n$. There are two parts to a reduction: (1) an oracle Turing machine P that efficiently converts $f(X)$ into an instance $g(Y)$ of the second primitive, where g is a standard function and Y is the polynomially samplable probability distribution on inputs to g; (2) an oracle Turing machine S that is the guarantee that the security of $f(X)$ is passed on to $g(Y)$. The security guarantee is of the form that if A is a breaking adversary for $g(Y)$ then S^A is a breaking adversary for $f(X)$. More formally,

Definition (reduction): We say that there is a reduction from *primitive_1* to *primitive_2* if there are two oracle Turing machines P and S with the following properties. Given any instance f of *primitive_1*, P^f is an instance g of *primitive_2*. Given any $R_g(n)$-breaking adversary A for g, S^A is a $R_f(n)$-breaking adversary for f.

The reduction guarantees that there is no $R_g(n)$-breaking adversary for g as long as there is no $R_f(n)$-breaking adversary for f. To have the reduction inject as much of the security of f as possible into g, we would like $R_f(n)$ to be as large as possible with respect to $R_g(n)$, e.g., $R_f(n) = R_g(n)$.

To give a rough measure of the amount of security a reduction preserves, we make the following definitions. Note that in all definitions the reduction has an overhead of $\frac{1}{n^c}$. However, typically $R_g(n) \ll \frac{1}{n^c}$ and it is therefore the dominant factor.

Definition (preserving reductions): The reduction from *primitive_1* to *primitive_2* is said to be

- *slightly preserving* if there are constants $\alpha \geq 1$, $\beta \geq 1$ and $c \geq 0$ such that

$$R_f(n) \geq \frac{R_g(n^\alpha)^\beta}{n^c}.$$

- *polynomially preserving* if there are constants $\beta \geq 1$ and $c \geq 0$ such that

$$R_f(n) \geq \frac{R_g(n)^\beta}{n^c}.$$

- *linearly preserving* if there is a constant $c \geq 0$ such that

$$R_f(n) \geq \frac{R_g(n)}{n^c}.$$

For a linearly preserving reduction, $R_f(n)$ is linearly lower bounded by $R_g(n)$, and for a polynomially preserving reduction, $R_f(n)$ is polynomially lower bounded by $R_g(n)$ (in both cases there is also a polynomial in n factor). On the other hand, for a slightly preserving reduction the lower bound on $R_f(n)$ can be much

weaker than any polynomial factor in $R_g(n)$. For this reason, a linearly preserving reduction is more desirable than a polynomially preserving reduction which in turn is more desirable than a slightly preserving reduction.

Consider a reduction from a one-way function f to a pseudo-random generator g and suppose we want the reduction to guarantee that g is $(1-R_g(n))$-secure. The difference between these types of guarantees isn't so important when $R_g(n)$ is not too small, e.g., if $R_g(n)$ is inverse polynomial in n then all types guarantee that $R_f(n)$ is inverse polynomial in n, and thus g is $(1-R_g(n))$-secure if there is no polynomial time adversary that can invert f with inverse polynomial probability. However, the difference between these types of guarantees increases dramatically as $R_g(n)$ goes to zero at a faster rate, which is expected in most applications. To see the dramatic differences between the strengths of the reductions, consider the case when $R_g(n) = 2^{-n^{1/2}}$ and $\alpha = \beta = 2$ and $c = 0$. For a linearly preserving reduction, g is $(1-R_g(n))$-secure if there is no $R_f(n) = 2^{-n^{1/2}}$-breaking adversary for f. For a polynomially preserving reduction, g is $(1-R_g(n))$-secure if there is no $R_f(n) = 2^{-2n^{1/2}}$-breaking adversary for f. For a slightly preserving reduction, g is $(1-R_g(n))$-secure if there is no $R_f(n) = 2^{-2n}$-breaking adversary for f. Note that in this case $R_f(n)$ is the $2n^{1/2}$ power of $R_g(n)$, which is not at all polynomial in $R_g(n)$. In fact, for trivial reasons there is a 2^{-2n}-breaking adversary for f, and thus the slightly preserving reduction does not guarantee that g is $(1-2^{-n^{1/2}})$-secure no matter how secure f is.

Because of the tremendous superiority of a linearly preserving over a polynomially preserving reduction over a slightly preserving reduction, it is important to design the strongest reduction possible. Some of the most important work (both theoretically and practically) left to be done is to find stronger preserving reductions between cryptographic primitives than are currently known, e.g. the strongest reductions known from a one-way function to a pseudo-random generator and from a weak one-way function to a one-way function (in the general case) are only slightly preserving.

It turns out that the primary quantity that determines the strength of the reduction is the ratio $\frac{n}{n'}$, where n is the length of the private part of the input for g and n' is the length of the private part of the input for calls to f when computing g. The bigger this ratio the more the loss in security. The best case is when these two lengths are equal or nearly equal. The reason for this is that typically the achievement ratio for S^A is either linear or polynomial in the achievement ratio $R_g(n)$ for A, and S^A breaks one of the calls to f on inputs of length n', and thus $R_f(n')$ is either linear or polynomial in $R_g(n)$. For example, if $n' = n$ and $R_f(n') = R_g(n)$ then the reduction is linearly preserving. Slightly weaker, if $n' = \epsilon n$ for some constant $\epsilon > 0$ and $R_f(n') = R_g(n)^\beta$ for some constant $\beta > 1$ then the reduction is polynomially preserving. This can be seen as follows. Even in the worst case, when $R_g(n) = \frac{1}{2^n}$, it is easy to verify that $R_g(n) = R_g(n'/\epsilon) \leq R_g(n')^{1/\epsilon}$. Thus, $R_f(n') \leq R_g(n')^{\beta/\epsilon}$. If n' is substantially smaller than n (but still polynomial in n), then the reduction is typically only slightly preserving.

4 The Reductions

We describe several linearly preserving reductions from a weak one-way permutation to a one-way permutation. All of the reductions work only for functions that are permutations.[3] In [4], Yao describes a reduction from a general weak owf to a one-way function, but the reduction is only slightly preserving. A good research problem is to design a linearly preserving (or even polynomially preserving) reduction without any restriction on the weak one-way function.

In all the reductions, we assume that the weak one-way function doesn't use a public random string. Only minor modifications need be made to handle the case when the weak one-way function uses public randomness.

All of the reductions share a common approach, and each reduction builds on the ideas developed in previous reductions. For completeness, we first describe a general reduction from a weak one-way function to a one-way function.

Reduction 1 [Yao] : Let $f(x)$ be a $s(n)$-weak one-way function, where $x \in \{0,1\}^n$. Let $N = \frac{n}{s(n)}$, let $y \in \{0,1\}^{N \times n}$ and define the one-way function

$$g(y) = (f(y_1), \ldots, f(y_N)).$$

Theorem 1 [Yao] : Reduction 1 is a slightly preserving reduction from a $s(n)$-weak one-way function f to one-way function g. More precisely, there is an oracle algorithm S such that if A is an $R_g(nN)$-breaking adversary for $g(y)$ then S^A is a $R_f(n)$-breaking adversary for $s(n)$-weak $f(x)$, where $R_f(n) = \frac{R_g(nN)}{nN}$.

Note that $s(n)$ must be at least inverse polynomial in n for the reduction to be even slightly preserving. This is because it is necessary for n to be a polynomial fraction of N, and $N = \frac{n}{s(n)}$.

4.1 A simple linearly preserving reduction

An important observation about Reduction 1 is that g doesn't use any public random bits beyond what is used by f. The reason the reduction is only slightly preserving is that g partitions its private input into many small strings and uses each of these strings as a private input to f. This can be thought of as a parallel construction, in the sense that the calls to f are on independent inputs and thus all calls to f can be computed simultaneously. The linearly preserving reduction given here is similar in its basic structure to Reduction 1. The main difference is that instead of partitioning the private input of g into N private inputs of length n for f, the private input to g is a single string $x \in \{0,1\}^n$, and the public random string is used to generate N inputs of length n to f sequentially.

Reduction 2 : Let $f(x)$ be a $s(n)$-weak one-way permutation, where $x \in \{0,1\}^n$. Let $N = \frac{n}{s(n)}$, let $\pi \in \{0,1\}^{N \times n}$ and define the one-way permutation

$$g(x;\pi) = y_{N+1}$$

[3] These reductions can be extended to the important case of regular functions, which is more general than permutations but still not the general case. A function is regular if each point in the range of the function has the same number of preimages.

where $y_1 = x$ and, for all $i = 2, \ldots, N+1$, $y_i = \pi_{i-1} \oplus f(y_{i-1})$.

Theorem 2 : Reduction 2 is a linearly preserving reduction from a $s(n)$-weak one-way permutation f to one-way permutation g. More precisely, there is an oracle algorithm S such that if A is an $R_g(n)$-breaking adversary for $g(x;\pi)$ then S^A is a $R_f(n)$-breaking adversary for $s(n)$-weak $f(x)$, where $R_f(n) = \frac{R_g(n)}{nN}$.

The proof of Theorem 2 is similar in spirit to the proof of the Theorem 1. We only describe the oracle algorithm S. Suppose that A is an adversary with time bound $T(n)$ and success probability $\delta(n)$ for g, and thus the achievement ratio is $\frac{\delta(n)}{T(n)}$. A on input $g(x;\pi)$ and π finds x with probability $\delta(n)$ when $x \in_U \{0,1\}^n$ and $\pi \in_U \{0,1\}^{N \times n}$. The oracle machine described below has the property that S^A inverts f on inputs of length n with probability at least $1 - s(n)$, where the time bound for S^A is $\frac{nNT(n)}{\delta(n)}$. The input to S^A is $f(x)$ where $x \in_U \{0,1\}^n$.

Adversary S^A on input $f(x)$:

Repeat $\frac{nN}{\delta(n)}$ times

Randomly choose $i \in_U \{2, \ldots, N+1\}$.

Randomly choose $\pi \in_U \{0,1\}^{N \times n}$.

Let $y_i = f(x) \oplus \pi_{i-1}$.

Compute $y_{i+1} = \pi_i \oplus f(y_i), \ldots, y_{N+1} = \pi_N \oplus f(y_N)$.

Compute $v_0 = A(y_{N+1};\pi)$.

Compute $v_1 = \pi_0 \oplus f(v_0), \ldots, v_{i-1} = \pi_{i-2} \oplus f(v_{i-2})$.

if $f(v_{i-1}) = f(x)$ then output v_{i-1}.

4.2 A linearly preserving reduction using less randomness

Although Reduction 2 is linearly preserving, it does have the drawback that the length of the public random string is rather large, and even worse this length depends linearly on the weakness parameter $s(n)$ of the weak one-way function. In this subsection, we describe a linearly preserving reduction that uses a much shorter public random string.

The overall structure of the reduction is the same as Reduction 2. The difference is that we use many fewer public random strings of length n in a recursive way to produce the almost random inputs to f. The reduction is in two steps. In the first step we describe a linearly preserving reduction from a $s(n)$-weak one-way permutation f to a $\frac{1}{2}$-weak one-way permutation g. The second step reduces g to a one-way permutation h using the construction given in Reduction 2.

Reduction 3 : Let $f(x)$ be a $s(n)$-weak one-way permutation, where $x \in \{0,1\}^n$. Let $l = \left\lceil \log_{3/2}(2/s(n)) \right\rceil$ and let $N = 2^l$. Let $\pi \in \{0,1\}^{l \times n}$. Define

$$g(x;\pi_1) = f(\pi_1 \oplus f(x)).$$

For all $i = 2, \ldots, l$, recursively define

$$g(x;\pi_{\{1,\ldots,i\}}) = g(\pi_i \oplus g(x;\pi_{\{1,\ldots,i-1\}});\pi_{\{1,\ldots,i-1\}}).$$

Theorem 3 : Reduction 3 is a linearly preserving reduction from a $s(n)$-weak one-way permutation $f(x)$ to $\frac{1}{2}$-weak one-way permutation $g(x;\pi)$. More precisely, there is an oracle algorithm S such that if A is an $R_g(n)$-breaking adversary for $\frac{1}{2}$-weak $g(x;\pi)$ then S^A is a $R_f(n)$-breaking adversary for $s(n)$-weak f, where $R_f(n) = \frac{R_g(n)}{nN}$.

The final step in the reduction is to go from weak one-way permutation g with weakness parameter $\frac{1}{2}$ to a one-way permutation h using Reduction 2, except now g has weakness parameter $\frac{1}{2}$ and uses a public random string of length $m = O(n \log(1/s(n))$. Thus, when using Reduction 2 to go from g to h, we set $N = \log(1/R_g(n)) \leq n$ and partition the public random string into N blocks of length $n+m$. Thus, the overall reduction uses $O(n^2 \log(1/s(n)))$ public random bits, as opposed to $O\left(\frac{n^2}{s(n)}\right)$ for Reduction 2. It is not hard to verify that the overall reduction from f to h is linearly preserving.

4.3 A linearly preserving reduction using expander graphs

The work described in [1] gives a polynomially preserving reduction from a weak one-way permutation to a one-way permutation that uses only a linear amount of public randomness. As briefly described below, their reduction can be modified in minor ways to yield a linearly preserving reduction ¿from a weak one-way permutation f to a one-way permutation h that uses only a linear number of public random bits overall.

As in Reduction 3, the reduction is in two steps: The first step is a linearly preserving reduction from a $s(n)$-weak one-way permutation f to a $\frac{1}{2}$-weak one-way permutation g and the second step reduces g to a one-way permutation h. As in Reduction 3, the first step is recursive and uses $O(\log(s(n))$ independent public random strings, but they are each of constant length instead of length n. The idea is to define a constant degree expander graph with vertex set $\{0,1\}^n$, and then each string is used to select a random edge out of a vertex in the expander graph. The second step is iterative, but uses only an additional $O(n)$ public random bits. These $O(n)$ public random bits are used to define a random walk of length $O(n)$ on a related expander graph.

The overall number of public random bits used in the entire reduction ¿from f to h is only linear. The way [1] describes the reduction, the one-way permutation f is applied to inputs of different lengths (all within a constant multiplicative factor of each other) to yield h. For this reason, as they describe their reduction it is only polynomially preserving, even with respect to the new definitions. Minor modifications to their reduction yields an alternative reduction where all inputs to f are of the same length as the private input to h. It can be shown that the alternative reduction with respect to the new definitions is linearly preserving.

Acknowledgements

We wish to thank Oded Goldreich, Hugo M. Krawczyk and Rafail Ostrovsky for their comments.

References

1. Goldreich, O., Impagliazzo, R., Levin, L., Venketesan, R., Zuckerman, D., "Security Preserving Amplification of Hardness", Proceedings of the 31st IEEE Symposium on Foundations of Computer Science, pp. 318-326, 1990.
2. Phillips, S. J., "Security Preserving Hardness Amplification Using PRGs for Bounded Space - Preliminary Report", unpublished manuscript, July 1992.
3. Luby, M., "Pseudorandomness and Applications", monograph in progress.
4. Yao, A., "Theory and applications of trapdoor functions", Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, pp. 80-91, 1982.