A preliminary version of this paper appears in *Advances in Cryptology – Crypto 98 Proceedings*, Lecture Notes in Computer Science Vol. 1462, H. Krawczyk ed., Springer-Verlag, 1998.

Many-to-one Trapdoor Functions and their Relation to Public-key Cryptosystems

M. Bellare* S. Halevi† A. Sahai‡ S. Vadhan§

June 1998

Abstract

The heart of the task of building public key cryptosystems is viewed as that of "making trapdoors;" in fact, public key cryptosystems and trapdoor functions are often discussed as synonymous. How accurate is this view? In this paper we endeavor to get a better understanding of the nature of "trapdoorness" and its relation to public key cryptosystems, by broadening the scope of the investigation: we look at general trapdoor functions; that is, functions that are not necessarily injective (ie., one-to-one). Our first result is somewhat surprising: we show that non-injective trapdoor functions (with super-polynomial pre-image size) can be constructed from any one-way function (and hence it is unlikely that they suffice for public key encryption). On the other hand, we show that trapdoor functions with polynomial pre-image size are sufficient for public key encryption. Together, these two results indicate that the pre-image size is a fundamental parameter of trapdoor functions. We then turn our attention to the converse, asking what kinds of trapdoor functions can be constructed from public key cryptosystems. We take a first step by showing that in the random-oracle model one can construct injective trapdoor functions from any public key cryptosystem.

Keywords: One-way functions, trapdoor functions, public key cryptosystems, relations amongst primitives.

*Dept. of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-Mail: mihir@cs.ucsd.edu. URL: http://www-cse.ucsd.edu/users/mihir. Supported in part by NSF CAREER Award CCR-9624439 and a 1996 Packard Foundation Fellowship in Science and Engineering.

† IBM T. J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA. E-Mail: shaih@watson.ibm.com.

‡ MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, USA. E-Mail: amits@theory.lcs.mit.edu. Supported by a DOD/NDSEG Graduate Fellowship and partially by DARPA grant DABT-96-C-0018.

§ MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, MA 02139, USA. E-Mail: salil@math.mit.edu. URL: http://www-math.mit.edu/~salil. Supported by a DOD/NDSEG Graduate Fellowship and partially by DARPA grant DABT-96-C-0018.

Contents

1 Introduction

A major dividing line in the realm of cryptographic primitives is that between "one-way" and "trapdoor" primitives. The former effectively means the primitives of private key cryptography, while the latter are typically viewed as tied to public key cryptosystems. Indeed, the understanding is that the problem of building public key cryptosystems is the problem of "making trapdoors."

Is it really? It is well known that injective (ie. one-to-one) trapdoor functions suffice for public key cryptography [Ya, GoMi]. We ask: is the converse true as well, or can public key cryptosystems exist under a weaker assumption? We take a closer look at the notion of a trapdoor, in particular from the point of view of how it relates to semantically secure encryption schemes, and discover some curious things. Amongst these are that "trapdoor one-way functions" are not necessarily hard to build, and their relation to public key encryption is more subtle than it might seem.

1.1 Background

The main notions discussed and related in this paper are one-way functions [DiHe], trapdoor (one-way) functions [DiHe], semantically secure encryption schemes [GoMi], and unapproximable trapdoor predicates [GoMi].

Roughly, a "one-way function" means a family of functions where each particular function is easy to compute, but most are hard to invert; trapdoor functions are the same with the additional feature that associated to each particular function is some "trapdoor" information, possession of which permits easy inversion. (See Section 2 for formal definitions.)

In the study of one-way functions, it is well appreciated that the functions need not be injective: careful distinctions are made between "(general) one-way functions", "injective one-way functions," and "one-way permutations." In principle, the distinction applies equally well to trapdoor one-way functions. (In the non-injective case, knowledge of the trapdoor permits recovery of *some* pre-image of any given range point [DiHe].) However, all attention in the literature has focused on injective trapdoor functions, perhaps out of the sense that this is what is necessary for constructing encryption schemes: the injectivity of the trapdoor function guarantees the unique decryptability of the encryption scheme.

This paper investigates general (ie. not necessarily injective) trapdoor one-way functions and how they relate to other primitives. Our goal is to understand exactly what kinds of trapdoor one-way functions are necessary and sufficient for building semantically secure public key encryption schemes; in particular, is injectivity actually necessary?

Among non-injective trapdoor functions, we make a further distinction based on "the amount of non-injectivity", measured by pre-image size. A (trapdoor, one-way) function is said to have pre-image size $Q(k)$ (where k is the security parameter) if the number of pre-images of any range point is at most $Q(k)$. We show that pre-image size is a crucial parameter with regard to building public-key cryptosystems out of a trapdoor function.

Rather than directly working with public-key cryptosystems, it will be more convenient to work with a more basic primitive called an unapproximable trapdoor predicate. Unapproximable trapdoor predicates are equivalent to semantically secure public key schemes for encrypting a single bit, and these in turn are equivalent to general semantically secure cryptosystems [GoMi].

1.2 Results

We have three main results. They are displayed in Figure 1 together with known relations. We now discuss them.

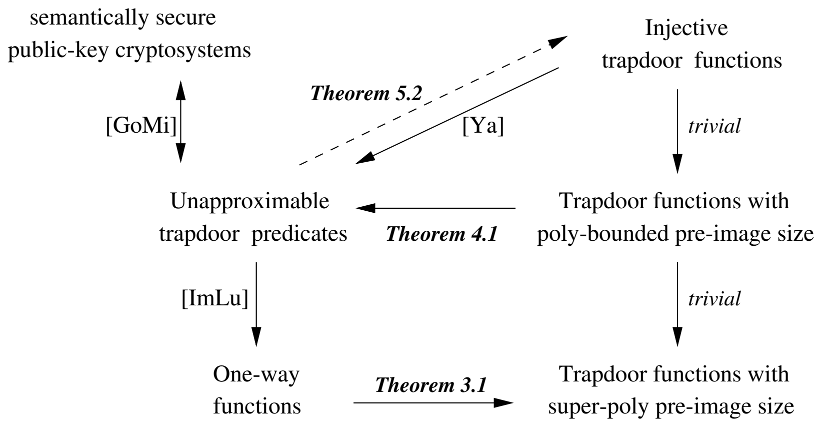

Figure 1: **Illustrating our results:** *Solid lines are standard implications; the dotted line is an implication in the random oracle model.*

One-way functions imply trapdoor functions. Our first result, given in Theorem 3.1, may seem surprising at first glance: we show that one-way functions imply trapdoor functions. We present a general construction which, given an arbitrary one-way function, yields a trapdoor (non-injective) one-way function.

Put in other words, we show that trapdoor functions are not necessarily hard to build; it is the combination of trapdoorness with "structural" properties like injectivity that may be hard to achieve. Thus the "curtain" between one-way and trapdoor primitives is not quite as opaque as it may seem.

What does this mean for public key cryptography? Impagliazzo and Rudich [ImRu] show that it would be very hard, or unlikely, to get a proof that one-way functions (even if injective) imply public key cryptosystems. Hence, our result shows that it is unlikely that any known technique can be used to construct public key encryption schemes from generic, non-injective, trapdoor functions. As one might guess given [ImRu], our construction does not preserve injectivity, so even if the starting one-way function is injective, the resulting trapdoor one-way function is not.

Trapdoor functions with poly pre-image size yield cryptosystems. In light of the above, one might still imagine that injectivity of the trapdoor functions is required to obtain public key encryption. Still, we ask whether the injectivity condition can be relaxed somewhat. Specifically, the trapdoor one-way functions which we construct from one-way functions have super-polynomial pre-image size. This leads us to ask about trapdoor functions with polynomially bounded pre-image size.

Our second result, Theorem 4.1, shows that trapdoor functions with polynomially bounded pre-image size suffice to construct unapproximable trapdoor predicates, and hence yield public key cryptosystems. This belies the impression that injectivity of the trapdoor function is a necessary feature to directly build a public key cryptosystem from it, and also suggests that the super-polynomial pre-image size in the construction of Theorem 3.1 is necessary.

From trapdoor predicates to trapdoor functions. We then turn to the other side of the coin and ask what kinds of trapdoor functions must necessarily exist to have a public key cryptosystem. Since unapproximable trapdoor predicates and semantically secure public key cryptosystems are equivalent [GoMi] we consider the question of whether unapproximable trapdoor predicates imply injective trapdoor functions.

In fact whether or not semantically secure public key cryptosystems imply injective trapdoor functions is not only an open question, but seems a hard one. (In particular, a positive answer would imply injective trapdoor functions based on the Diffie-Hellman assumption, a long standing open problem.) In order to get some insight and possible approaches to it, we consider it in a random oracle model (cf. [ImRu, BeRo]). Theorem 5.2 says that here the answer is affirmative: given an arbitrary secure public key cryptosystem, we present a function that has access to an oracle H, and prove the function is injective, trapdoor, and one-way when H is random.

The construction of Theorem 5.2 is quite simple, and the natural next question is whether the random oracle H can be replaced by some constructible cryptographic primitive. We show that this may be difficult, by showing that a cryptographically strong pseudorandom bit generator [BlMi, Ya], which seems like a natural choice for this construction, does not suffice. (See Appendix A). The next step may be to follow the approach initiated by Canetti [Ca]: find an appropriate cryptographic notion which, if satisfied by H, would suffice for the correctness of the construction, and then try to implement H via a small family of functions. However, one should keep in mind that replacement of a random oracle by a suitable constructible function is not always possible [CGH]. Thus, our last result should be interpreted with care.

1.3 Discussion and implications

Theorems 3.1 and 4.1 indicate that pre-image size is a crucial parameter when considering the power of trapdoor functions, particularly with respect to constructing public-key cryptosystems. The significance and interpretation of Theorem 5.2, however, requires a bit more discussion.

At first glance, it may seem that public key cryptosystems "obviously imply" injective trapdoor functions. After all, a public key cryptosystem permits unique decryptability; doesn't this mean the encryption algorithm is injective? No, because, as per [GoMi], it is a *probabilistic* algorithm, and thus not a function. To make it a function, you must consider it a function of two arguments, the message and the coins, and then it may no longer be injective, because two coin sequences could give rise to the same ciphertext for a given message. Moreover, it may no longer have a (full) trapdoor, since it may not be possible to recover the randomness from the ciphertext. (Public key cryptosystems in the Diffie and Hellman sense [DiHe] imply injective trapdoor one-way functions as the authors remark, but that's because encryption there is deterministic. It is now understood that secure encryption must be probabilistic [GoMi].)

Theorem 5.2 has several corollaries. (Caveat: All in the random oracle model). First, by applying a transformation of [BeRo], it follows that we can construct non-malleable and chosen-ciphertext secure encryption schemes based on the Ajtai-Dwork cryptosystem [AjDw]. Second, combining Theorems 5.2 and 4.1, the existence of trapdoor functions with polynomially bounded pre-image size implies the existence of injective trapdoor functions. (With high probability over the choice of oracle. See Remark 5.11.) Third, if the Decisional Diffie-Hellman problem is hard (this means the El Gamal [ElG] cryptosystem is semantically secure) then there exists an injective trapdoor function.

Note that in the random oracle model, it is trivial to construct (almost) injective one-way functions: a random oracle mapping, say, n bits to $3n$ bits, is itself an injective one-way function except with probability 2^{-n} over the choice of the oracle. However, random oracles do not directly or naturally give rise to trapdoors [ImRu]. Thus, it is interesting to note that our construction in Theorem 5.2 uses the oracle to "amplify" a trapdoor property: we convert the weak trapdoor property of a cryptosystem (in which one can only recover the message) to a strong one (in which one can recover both the message and the randomness used).

Another interpretation of Theorem 5.2 is as a demonstration that there exists a model in which semantically secure encryption implies injective trapdoor functions, and hence it may be hard to prove a separation result, in the style of [ImRu], between injective trapdoor functions and probabilistic encryption schemes.

2 Definitions

We present definitions for one-way functions, trapdoor functions, and unapproximable trapdoor predicates.

PRELIMINARIES. If S is any probability distribution then $x \leftarrow S$ denotes the operation of selecting an element uniformly at random according to S, and $[S]$ is the support of S, namely the set of all points having non-zero probability under S. If S is a set we view it as imbued with the uniform distribution and write $x \leftarrow S$. If A is a probabilistic algorithm or function then $A(x, y, \cdots ; R)$ denotes the output of A on inputs $x, y, \ldots$ and coins R, while $A(x, y, \ldots)$ is the probability distribution assigning to each string the probability, over R, that it is output. For deterministic algorithms or functions A, we write $z{:=}A(x, y, \ldots)$ to mean that the output of $A(x, y, \ldots)$ is assigned to z. The notation $\Pr[\, E \,:\, R_1 \,;\, R_2 \,;\, \ldots \,;\, R_k \,]$ refers to the probability of event E after the random processes $R_1, \ldots, R_k$ are performed in order. If x and y are strings we write their concatenation as $x \| y$ or just xy. "Polynomial time" means time polynomial in the security parameter k, PPT stands for "probabilistic, polynomial time", and "efficient" means computable in polynomial time or PPT.

2.1 One-way and trapdoor function families

We first define families of functions, then say what it means for them to be one-way or trapdoor.

FAMILIES OF FUNCTIONS. A *family of functions* is a collection $F = \{F_k\}_{k \in \mathbb{N}}$ where each F_k is probability distribution over a set of functions. Each $f \in [F_k]$ has an associated domain $\mathrm{Dom}(f)$ and range $\mathrm{Range}(f)$. We require three properties of the family:

- *Can generate:* The operation $f \leftarrow F_k$ can be efficiently implemented, meaning there is a PPT *generation* algorithm $F\text{-}Gen$ that on input 1^k outputs a "description" of a function f distributed according to F_k. This algorithm might also output some auxiliary information *aux* associated to this function (this is in order to later model trapdoors).
- *Can sample:* $\mathrm{Dom}(f)$ is efficiently samplable, meaning there is a PPT algorithm $F\text{-}Smp$ that given $f \in [F_k]$ returns a uniformly distributed element of $\mathrm{Dom}(f)$.
- *Can evaluate:* f is efficiently computable, meaning there is a polynomial time evaluation algorithm $F\text{-}Eval$ that given $f \in F_k$ and $x \in \mathrm{Dom}(f)$ returns $f(x)$.

For an element $y \in \mathrm{Range}(f)$ we denote the set of *pre-images* of y under f by

$$f^{-1}(y) \;=\; \{\, x \in \mathrm{Dom}(f) \;:\; f(x) = y \,\} \,.$$

We say that F is *injective* if f is injective (ie. one-to-one) for every $f \in [F_k]$. If in addition $\mathrm{Dom}(f) = \mathrm{Range}(f)$ then we say that F is a family of *permutations.* We measure the amount of "non-injectivity" by looking at the maximum pre-image size. Specifically we say that F has *pre-image size* bounded by $Q(k)$ if $|f^{-1}(y)| \le Q(k)$ for all $f \in [F_k]$, all $y \in \mathrm{Range}(f)$ and all $k \in \mathbb{N}$. We say that F has *polynomially bounded pre-image size* if there is a polynomial $Q(k)$ which bounds the pre-image size of F.

ONE-WAYNESS. Let F be a family of functions as above. The *inverting probability* of an algorithm $I(\cdot, \cdot)$ with respect to F is a function of the security parameter k, defined as $\mathrm{InvProb}_F(I, k) \stackrel{\mathrm{def}}{=}$

$$\Pr\left[\, x' \in f^{-1}(y) \;:\; f \leftarrow F_k \,;\, x \leftarrow \mathrm{Dom}(f) \,;\, y \leftarrow f(x) \,;\, x' \leftarrow I(f, y) \,\right] \,.$$

F is *one-way* if $\mathrm{InvProb}_F(I, k)$ is negligible for any PPT algorithm I.

TRAPDOORNESS. A family of functions is said to be trapdoor if it is possible, while generating an instance f, to simultaneously generate as auxiliary output "trapdoor information" tp, knowledge of which permits inversion of f. Formally, a family of functions F is *trapdoor* if F-Gen outputs pairs (f, tp) where f is the "description" of a function as in any family of functions and tp is auxiliary *trapdoor information*. We require that there exists a probabilistic polynomial time algorithm F-Inv such that for all k, all $(f, tp) \in [F\text{-}Gen(1^k)]$, and all points $y \in \mathrm{Range}(f)$, the algorithm $F\text{-}Inv(f, tp, y)$ outputs an element of $f^{-1}(y)$ with probability 1. A family of trapdoor functions is said to be one-way if it is also a family of one-way functions.

Example 2.1 [The Rabin family of trapdoor functions] The *Rabin family* [Rab] is a good example in our setting since it consists of non-injective trapdoor functions. It is captured as follows. We set $Rab = \{ Rab_N : N \in Rab\text{-}Keys \}$, where Rab-$Keys$ is the set of all numbers which are the product of two primes. We let $\mathrm{Dom}(Rab_N) = Z_N^*$ and let $\mathrm{Range}(Rab_N)$ be the set of quadratic residues in Z_N^*. The function is defined by $Rab_N(x) = x^2 \bmod N$ for all $x \in Z_N^*$. The generator Rab-Gen uses its coins r to pick two primes p, q of roughly equal length, and outputs as the description of the function instance the modulus $N = pq$, and as the trapdoor the primes p, q themselves. The sampler, given N, outputs a uniformly distributed point in Z_N^*. The evaluation algorithm, given N and $x \in Z_N^*$, outputs $x^2 \bmod N$. The inversion algorithm Rab-Inv, given N, (p, q), and a quadratic-residue $y \bmod N$, computes the square roots of y modulo both p and q (which can be done in polynomial time [Be, AMM]) and then uses the Chinese Remainder Theorem to obtain a square root of y modulo N. Notice that the Rabin function family is *not* injective, as every quadratic residue mod N has four square roots. (Traditionally, this function is used as the basis of a public key cryptosystem by first modifying it to be injective.)

Remark 2.2 It is well known that one can define one-way functions either in terms of function families (as above), or in terms of a single function, and the two are equivalent. However, for trapdoor functions, one must talk of families. To maintain consistency, we use the family view of one-way functions as well.

2.2 Trapdoor Predicate Families

We define unapproximable trapdoor predicate families [GoMi]. Recall that such a family is equivalent to a semantically secure public-key encryption scheme for a single bit [GoMi].

A *predicate* in our context means a *probabilistic* function with domain $\{0, 1\}$, meaning a predicate p takes a bit b and flips coins r to generate some output $y = p(b; r)$. In a *trapdoor predicate family* $P = \{P_k\}_{k \in \mathbb{N}}$, each P_k is a probability distribution over a set of predicates, meaning each $p \in [P_k]$ is a predicate as above. We require:

- *Can generate:* There is a generation algorithm P-Gen which on input 1^k outputs (p, tp) where p is distributed randomly according to P_k and tp is trapdoor information associated to p. In particular the operation $p \leftarrow P_k$ can be efficiently implemented.
- *Can evaluate:* There is a PPT algorithm P-$Eval$ that given p and $b \in \{0, 1\}$ flips coins to output y distributed according to $p(b)$.

We say P has *decryption error* $\delta(k)$ if there is a PPT algorithm P-Inv who, with knowledge of the trapdoor, fails to decrypt only with this probability, namely

$$\mathrm{DecErr}_P(P\text{-}Inv, k) \stackrel{\mathrm{def}}{=} \\ \Pr\left[\, b' \neq b \,:\, p \leftarrow P_k \,;\, b \leftarrow \{0,1\} \,;\, y \leftarrow p(b) \,;\, b' \leftarrow P\text{-}Inv(p, tp, y) \,\right] \qquad (1)$$

is at most $\delta(k)$. If we say nothing it is to be assumed that the decryption error is zero, but sometimes we want to discuss families with non-zero (and even large) decryption error.

UNAPPROXIMABILITY. Let P be a family of trapdoor predicates as above. The *predicting advantage* of an algorithm $I(\cdot,\cdot)$ with respect to P is a function of the security parameter k, defined as $\mathrm{PredAdv}_P(I,k) \stackrel{\mathrm{def}}{=}$

$$\Pr\left[\, b' = b \;:\; p \leftarrow P_k \;;\; b \leftarrow \{0,1\} \;;\; y \leftarrow p(b) \;;\; b' \leftarrow I(p,y) \,\right] - \frac{1}{2}\,.$$

We say that P is *unapproximable* if $\mathrm{PredAdv}_P(I,k)$ is negligible for any PPT algorithm I.

3 From one-way functions to trapdoor functions

In this section we establish the following result:

Theorem 3.1 *Suppose there exists a family of one-way functions. Then there exists a family of trapdoor, one-way functions.*

This is proved by taking an arbitrary family F of one-way functions and "embedding" a trapdoor to get a family G of trapdoor functions. The rest of this section is devoted to the proof.

3.1 Proof of Theorem 3.1

Given a family $F = \{F_k\}_{k\in\mathbb{N}}$ of one-way functions we show how to construct a family $G = \{G_k\}_{k\in\mathbb{N}}$ of trapdoor one-way functions.

Let us first sketch the idea. Given $f \in F_k$ we want to construct g which "mimics" f but somehow embeds a trapdoor. The idea is that the trapdoor is a particular point α in the domain of f. Function g will usually just evaluate f, except if it detects that its input contains the trapdoor; in that case it will do something trivial, making g easy to invert given knowledge of the trapdoor. (This will not happen often in normal execution because it is unlikely that a randomly chosen input contains the trapdoor.) But how exactly can g "detect" the trapdoor? The first idea would be to include α in the description of g so that it can check whether its input contains the trapdoor, but then g would no longer be one-way. So instead the description of g will include $\beta = f(\alpha)$, an image of the trapdoor under the original function f, and g will run f on a candidate trapdoor to see whether the result matches β. (Note that we do not in fact necessarily detect the real trapdoor α; the trivial action is taken whenever some pre-image of β under f is detected. But that turns out to be OK.)

In the actual construction, g has three inputs, y, x, v, where v plays the role of the "normal" input to f; x plays the role of the candidate trapdoor; and y is the "trivial" answer returned in case the trapdoor is detected. We now formally specify the construction and sketch a prof that it is correct.

A particular function $g \in [G_k]$ will be described by a pair (f, β) where $f \in [F_k]$ and $\beta \in \mathrm{Range}(f)$. It is defined on inputs y, x, v by

$$g(y,x,v) = \begin{cases} y & \text{if } f(x) = \beta \\ f(v) & \text{otherwise.} \end{cases} \tag{2}$$

Here $x, v \in \mathrm{Dom}(f)$, and we draw y from some samplable superset S_f of $\mathrm{Range}(f)$. (To be specific, we set S_f to the set of all strings of length at most $p(k)$ where $p(k)$ is a polynomial that bounds the lengths of all strings in $\mathrm{Range}(f)$.) So the domain of g is $\mathrm{Dom}(g) = S_f \times \mathrm{Dom}(f) \times \mathrm{Dom}(f)$.

We now give an intuitive explanation of why G is one-way and trapdoor. First note that for any z it is the case that (z, α, α) is a preimage of z under g, so knowing α enables one to invert in a trivial manner, hence G is trapdoor. For one-wayness, notice that if $g(y,x,v)=z$ then either $f(v)=z$ or $f(x)=\beta$. Thus, producing an element of $g^{-1}(z)$ requires inverting f at either z or β, both of which are hard by the one-wayness of F. We now proceed with a more formal proof that G satisfies the definition of a family of one-way trapdoor functions.

The generator $G\text{-}Gen$ takes input 1^k and lets $f \leftarrow F_k$; $\alpha \leftarrow \mathrm{Dom}(f)$; $\beta := f(\alpha)$. It outputs (g,α) where the function g is as defined above, and $\alpha = tp$ is the trapdoor.

The operations of $G\text{-}Gen$ can be performed in PPT given that F is a family of functions as per our definition. Notice that it is possible to sample uniformly from $\mathrm{Dom}(g) = S_f \times \mathrm{Dom}(f) \times \mathrm{Dom}(f)$ in PPT (as required to be a family of functions) because each of the three constituent sets has this property. (It is to make this true that we used S_f rather than $\mathrm{Range}(f)$ in the first component). Finally it is clear that $g(y,x,v)$ can be evaluated in $\mathrm{poly}(k)$ time since this is assumed true for f. We need to check two things; that G is a trapdoor family, meaning possession of α permits inversion, and that G retains the one-wayness of F.

Claim 3.2 *The family G is trapdoor.*

Proof: We show that knowing the trapdoor information α allows one to invert g. Formally, we define the inverter as follows. Let w be any point in $\mathrm{Range}(g)$ where g is described by (f,β). Then set

$$G\text{-}Inv(g,\alpha,w) \;=\; (w,\alpha,\alpha)\ .$$

To see that this works, first notice that (w,α,α) is indeed in $\mathrm{Dom}(g)$ because w, being in $\mathrm{Range}(g)$, is also in S_f, and α is in $\mathrm{Dom}(f)$. Now apply Equation (2) to compute $g(w,\alpha,\alpha)$: since $f(\alpha)=\beta$ we get $g(w,\alpha,\alpha)=w$. Thus, (w,α,α) is indeed a pre-image of w under g, as desired. ▮

It remains to show that G is one-way. Intuitively, this is true because producing an element of $g^{-1}(w)$ requires producing either an element of $f^{-1}(w)$ or an element of $f^{-1}(\beta)$, both of which are hard by the one-wayness of F. The following proof formalizes this intuition.

Claim 3.3 *The family G is one-way.*

Proof: Let I be any polynomial time inverting algorithm. We want to show that $s(k) = \mathrm{InvProb}_G(I,k)$ is a negligible function of k. Let Experiment 1 be that underlying the definition of $\mathrm{InvProb}_G(I,k)$ as in Section 2.1; it is given by:

$$g \leftarrow G_k\ ;\ (y,x,v) \leftarrow \mathrm{Dom}(g)\ ;\ w \leftarrow g(y,x,v)\ ;\ (y',x',v') \leftarrow I(g,w)\ .$$

Let $\Pr_1[\cdot]$ denote the probability under this experiment. Note that for any function g as above, if $I(g,w)$ succeeds and produces (y',x',v') such that $g(y',x',v')=w$, then it must be that either $f(x')=\beta$ or $f(v')=w$. Accordingly let

$$s_1(k) \;=\; \Pr_1[\,f(x')=\beta\,] \qquad \text{and} \qquad s_2(k) \;=\; \Pr_1[\,f(v')=w\,]\ .$$

Then we have $s(k) \le s_1(k)+s_2(k)$, so it suffices to prove that both $s_1(k)$ and $s_2(k)$ are negligible.

Subclaim 1: $s_1(k)$ is negligible.

We will construct an inverter I_1 for F which succeeds with probability $s_1(k)$. Then, by the one-wayness of F, it follows that $s_1(k)$ is negligible. I_1 is constructed as follows:

Inverter $I_1(f,z)$ // *Wants to compute* $f^{-1}(z)$
  $\beta{:=}z$, $g{:=}(f,\beta)$ // *Let* z *play the role of* β *for* g
  $(y,x,v) \leftarrow \mathrm{Dom}(g)$ // *Sample from the domain of* g

$w := g(y, x, v)$ // *Evaluate* g
$(y', x', v') \leftarrow I(g, w)$ // *Apply the inversion algorithm for* g *to* w
Output x'

Note that the input w fed to I by I_1 in the above is distributed exactly as in Experiment 1 when we also consider the initial choices of $f \leftarrow F_k$ and $z \leftarrow f(u)$ for uniformly selected $u \leftarrow \mathrm{Dom}(f)$. Thus, we know that I will produce x' such that $f(x') = \beta$ with probability at least $s_1(k)$, and $\mathrm{InvProb}_F(I_1, k) = s_1(k)$.

Subclaim 2: $s_2(k)$ is negligible.

Again, we want to construct an inverter I_2 to invert f at a given point z. We pick β as it would be chosen by G_k, and then would like to let z play the role of w, meaning let $(y', x', v') \leftarrow I(g, z)$ and, assuming $f(v') = z$, output v'. If we just do this, however, the input z to I as provided by I_2 is not distributed in the same way as the w of Experiment 1, because in Equation (2) we sometimes don't output $f(v)$, specifically in the case that $f(x) = \beta$. To compensate for this, we run a "simulation" of this part of the process of generating a domain point for g, by picking x and doing the test for $f(x) = \beta$. It if succeeds we abort, else we do what we really want, namely apply I to z.

Inverter $I_2(f, z)$ // *Wants to compute* $f^{-1}(z)$
$\alpha \leftarrow \mathrm{Dom}(f)$ // *Sample from the domain of* f
$\beta{:=}f(\alpha)$, $g{:=}(f, \beta)$
$x \leftarrow \mathrm{Dom}(f)$
If $f(x) = \beta$ **then abort** // *If you got a pre-image of* β *then quit*
Else
$(y', x', v') \leftarrow I(g, z)$
Output v'

For the analysis, first note that in Experiment 1,

$$s_2(k) \;=\; \Pr_1\left[\, f(v') = w \,\right] \leq \Pr_1\left[\, f(v') = w \;\;\&\;\; f(x) \neq \beta \,\right] + \Pr_1\left[\, f(x) = \beta \,\right].$$

The first term on the right equals $\mathrm{InvProb}_F(I_2, k)$, which is negligible by the one-wayness of F. The second term on the right is also negligible, since otherwise one can invert F by the following simple algorithm I_3: given f, z, it picks x at random from the domain of f and returns x. ▮

Remark 3.4 One can verify that the trapdoor functions g produced in the above construction are *regular* (ie. the size of $g^{-1}(y)$ is the same for all $y \in \mathrm{Range}(g)$) if the original one-way functions f are regular. Thus, adding regularity as a requirement is *not* likely to suffice for making public-key cryptosystems.

4 From trapdoor functions to cryptosystems

Theorem 3.1 coupled with [ImRu] says that it is unlikely that general trapdoor functions will yield semantically secure public-key cryptosystems. However, in our construction of Section 3.1 the resulting trapdoor function was "very non-injective" in the sense that the pre-image size was exponential in the security parameter. So, we next ask, what is the power of trapdoor function families with polynomially bounded pre-image size? We show a positive result:

Theorem 4.1 *If there exist trapdoor one-way function families with polynomially bounded pre-image size, then there exists a family of unapproximable trapdoor predicates with exponentially small decryption error.*

Theorem 4.1 extends the well-known result of [Ya, GoMi] that injective trapdoor functions yield semantically secure public-key cryptosystems, by showing that the injectivity requirement can be relaxed. Coupled with [ImRu] this also implies that it is unlikely that the analogue of Theorem 3.1 can be shown for trapdoor functions with polynomially bounded pre-image sizes.

4.1 Proof of Theorem 4.1

Let $F = \{F_k\}_{k \in \mathbb{N}}$ be a family of trapdoor one-way functions with pre-image size bounded by a polynomial Q. The construction is in two steps. We first build an unapproximable family of trapdoor predicates P with decryption error $1/2 - 1/\operatorname{poly}(k)$, and then reduce the decryption error by repetition to get the family claimed in the theorem.

The first step uses the Goldreich-Levin inner-product construction [GoLe]. This construction says that if f is a one-way function, one can securely encrypt a bit b via the triple $(f(x), r, \sigma)$ where $\sigma = b \oplus (x \odot r)$ with r a random string, $x \in \operatorname{Dom}(f)$, and $\odot$ denoting the inner-product mod 2. Now, if f is an *injective* trapdoor function, then with the trapdoor information, one can recover b from $f(x)$, r, and σ by finding x and computing $b = \sigma \oplus (x \odot r)$. If instead f has polynomial-size pre-images, the "correct" x will only be recovered with an inverse polynomial probability. However, we will show that the rest of the time, the success probability is exactly 50%. This gives a noticeable $(\frac{1}{2} + \frac{1}{\operatorname{poly}(k)})$ bias towards the right value of b. Now, this slight bias needs to be amplified, which is done by repeating the construction many times in parallel and having the decryptor take the majority of its guesses to the bit in the different coordinates. A full description and proof follow.

We may assume wlog that there is a polynomial $l(k)$ such that $\operatorname{Range}(f) \subseteq \{0,1\}^{l(k)}$ for all $f \in [F_k]$ and all $k \in \mathbb{N}$. We now describe how to use the Goldreich-Levin inner-product construction [GoLe] to build $P = \{P_k\}_{k \in \mathbb{N}}$. We associate to any $f \in [F_k]$ a predicate p defined as follows:

Predicate $p(b)$ // *Takes input a bit b*
  $x \leftarrow \operatorname{Dom}(f)$ // *Choose x at random from the domain of f*
  $r \leftarrow \{0,1\}^{l(k)}$ // *Choose a random $l(k)$-bit string*
  $\sigma := b \oplus (x \odot r)$ // *XOR b with the GL bit*
  Output $(f(x), r, \sigma)$

Here $\oplus$ denotes XOR (ie. addition mod 2) and $\odot$ denotes the inner-product mod 2. The generator algorithm for P will choose $(f, tp) \leftarrow F\text{-}Gen(1^k)$ and then output (p, tp) with p defined as above. Notice that p is computable in PPT if f is.

The inversion algorithm $P\text{-}Inv$ is given p, the trapdoor tp, and a triple (y, r, σ). It first runs the inversion algorithm $F\text{-}Inv$ of F on inputs f, tp, y to obtain x', and then outputs the bit $b' = \sigma \oplus (x' \odot r)$. It is clear that the inversion algorithm is not always successful, but in the next claim we prove that it is successful appreciably more often than random guessing.

Claim 4.2 *P is an unapproximable trapdoor predicate family, with decryption error at most $(1/2) - 1/[2Q(k)]$.*

Proof: We know that F is one-way. Thus, the inner product is a hardcore bit for F [GoLe]. This implies that P is unapproximable. It is left to show that the decryption error of P is as claimed, namely that $\operatorname{DecErr}_P(P\text{-}Inv, k)$ (as defined in Equation (1)) is at most $(1/2) - 1/[2Q(k)]$.

Fix f, tp, b, let x, r be chosen at random as by $p(b)$, let $y = f(x)$, let $\sigma = b \oplus (x \odot r)$, let $x' \leftarrow F\text{-}Inv(f, tp, y)$, and let $b' = \sigma \oplus (x' \odot r)$. Notice that if $x' = x$ then $b' = b$, but if $x' \neq x$ then the random choice of r guarantees that $b' = b$ with probability at most $1/2$. (Because F-*Inv*, who generates x', gets no information about r.) The chance that $x = x'$ is at least $1/Q(k)$ (because F-*Inv* gets no information about x other than that $f(x) = y$) so

$$\mathrm{DecErr}_P(P\text{-}Inv, k) \ \leq \ \left(1 - \frac{1}{Q(k)}\right) \cdot \frac{1}{2}$$

as desired. ▮

Now, we can iterate the construction $q(k) \stackrel{\text{def}}{=} \Theta(kQ(k)^2)$ times independently and decrypt via a majority vote to reduce the decryption error to e^{-k}. In more detail, our final predicate family $P^q = \{P^q_k\}_{k \in \mathbb{N}}$ is like this. An instance $p^q \in [P^q_k]$ is still described by a function $f \in [F_k]$ and defined as $p^q(b) = p(b) \| \cdots \| p(b)$, meaning it consists of $q(k)$ repetitions of the original algorithm p on independent coins. The inversion algorithm P^q-*Inv* is given the trapdoor tp and a sequence of triples

$$(y_1, r_1, \sigma_1) \| \cdots \| (y_{q(k)}, r_{q(k)}, \sigma_{q(k)}) \ .$$

For $i = 1, \ldots, q(k)$ it lets $b'_i = P\text{-}Inv(p, tp, (y_i, r_i, \sigma_i))$. It outputs b' which is 1 if the majority of the values $b'_1, \ldots, b'_{q(k)}$ are 1, and 0 otherwise. Chernoff bounds show that $\mathrm{DecErr}_{P^q}(P^q\text{-}Inv, k) \leq e^{-k}$. Furthermore standard "hybrid" arguments [GoMi, Ya] show that P^q inherits the unapproximability of P. This concludes the proof of Theorem 4.1.

Remark 4.3 Notice that Theorem 4.1 holds even if the family F only satisfies a very weak trapdoor property — namely, that F-*Inv* produces an element of $f^{-1}(y)$ with probability at least $1/p(k)$ for some polynomial p. Essentially the same proof will show that P-*Inv* can guess b correctly with probability at least $1/2 + 1/[2Q(k)p(k)]$.

5 From cryptosystems to trapdoor functions

In this section we investigate the relation between semantically secure public key cryptosystems and injective trapdoor functions. It is known that the existence of unapproximable trapdoor predicates is equivalent to the existence of semantically secure public-key encryption [GoMi]. It is also known that injective trapdoor one-way functions can be used to construct unapproximable trapdoor predicates [Ya] (see also [GoLe]). In this section, we ask whether the converse is true:

Question 5.1 Can unapproximable trapdoor predicates be used to construct injective trapdoor one-way functions?

Note the importance of the injectiveness condition in Question 5.1. We already know that non-injective trapdoor functions can be constructed from trapdoor predicates (whether the latter are injective or not) because trapdoor predicates imply one-way functions [ImLu] which in turn imply trapdoor functions by Theorem 3.1.

We suggest a construction which requires an additional "random looking" function G and prove that the scheme is secure when G is implemented as a random oracle (to which the adversary also has access). Hence, *IF* it is possible to implement using one-way functions a function G with "sufficiently strong randomness properties" to maintain the security of this scheme, then Question 5.1 would have a positive answer (as one-way functions can be constructed from unapproximable trapdoor predicates [ImLu]).

The key difference between trapdoor functions and trapdoor predicates is that predicates are *probabilistic*, in that their evaluation is a probabilistic process. Hence, our construction is essentially a de-randomization process.

Suppose we have a family P of unapproximable trapdoor predicates, and we want to construct a family F of injective one-way trapdoor functions from P. A first approach would be to take an instance p of P and construct an instance f of F as

$$f(b_1 b_2 \cdots b_k \| r_1 \| \cdots \| r_k) = p(b_1; r_1) \| \cdots \| p(b_k; r_k),$$

where k is the security parameter. Standard direct product arguments [Ya] imply that F constructed in this manner is one-way. However, F may fail to be trapdoor; the trapdoor information associated with p only allows one to recover $b_1, \ldots, b_k$, but not $r_1, \ldots, r_k$.

Our approach to fixing this construction is to instead have $r_1, \ldots, r_k$ determined by applying some "random-looking" function G to $b_1, \ldots, b_k$:

$$f(b_1 b_2 \cdots b_k) = p(b_1; r_1) \| \cdots \| p(b_k; r_k), \qquad \text{where } r_1 \| \cdots \| r_k = G(b_1 \cdots b_k).$$

Since G must be length-increasing, an obvious choice for G is a pseudo-random generator. A somewhat circular intuitive argument can be made for the security of this construction: If one does not know $b_1, \ldots, b_k$, then $r_1, \ldots, r_k$ "look random," and if $r_1, \ldots, r_k$ "look random," then it should be hard to recover $b_1, \ldots, b_k$ by the unapproximability of P. In Appendix A, we show that this argument is in fact false, in that there is a choice of an unapproximable trapdoor predicate P and a pseudorandom generator G for which the resulting scheme is insecure.

However, it is still possible that there are choices of functions G that make the above secure. Below we show that the scheme is secure when G is implemented as a truly random function, ie. a random oracle (to which the adversary also has access). Intuitively, having access to the oracle does not help the adversary recover $b_1 \cdots b_k$ for the following reason: the values of the oracle are irrelevant except at $b_1 \cdots b_k$, as they are just random strings that have nothing to do with $b_1 \cdots b_k$ or $f(b_1 \cdots b_k)$. The adversary's behavior is independent of the value of the oracle at $b_1 \cdots b_k$ unless the adversary queries the oracle at $b_1 \cdots b_k$. On the other hand, if the adversary queries the oracle at $b_1 \cdots b_k$, it must already "know" $b_1 \cdots b_k$. Specifically, if the adversary queries the oracle at $b_1 \cdots b_k$ with non-negligible probability then it can invert f with non-negligible probability without making the oracle call, by outputting the query. We now proceed with a more formal description of the random oracle model and our result.

THE RANDOM ORACLE MODEL. In any cryptographic scheme which operates in the random oracle model, all parties are given (in addition to their usual resources) the ability to make oracle queries [BeRo]. It is postulated that all oracle queries, independent of the party which makes them, are answered by a single function, denoted $\mathcal{O}$, which is uniformly selected among all possible functions (where the set of possible functions is determined by the security parameter).

The definitions of families of functions and predicates are adapted to the random oracle model in a straightforward manner: We associate some fixed polynomial Q with each family of functions or predicates, such that on security parameter k all the algorithms in the above definitions are given oracle access to a function $\mathcal{O} : \{0,1\}^* \to \{0,1\}^{Q(k)}$. The probabilities in these definitions are then taken over the randomness of these algorithms and also over the choice of $\mathcal{O}$ uniformly at random among all such functions.

Theorem 5.2 *If there exists a family of unapproximable trapdoor predicates, then there exists a family of injective trapdoor one-way functions in the random oracle model.*

Remark 5.3 Theorem 5.2 still holds even if the hypothesis is weakened to only require the existence of a family of unapproximable trapdoor predicates *in the random oracle model.* To see that this hypothesis is weaker, note that a family of unapproximable trapdoor predicates (in the standard, non-oracle model) remains unapproximable in the random oracle model, as the oracle only provides randomness which the adversary can generate on its own. To prove the result with the weaker assumption, we modify the construction below by dividing the random oracle into two independent parts. One part is used for any oracle calls the predicate generation and evaluation algorithms make and the other part is used as the oracle $\mathcal{O}$ below.

See Sections 1.2 and 1.3 for a discussion of the interpretation of such a result. We now proceed to the proof.

5.1 Proof of Theorem 5.2

Let $P = \{P_k\}_{k \in \mathbb{N}}$ be a family of unapproximable trapdoor predicates. Let $q(k)$ be a polynomial upper bound on the number of random bits used by any $p \in P_k$. When used with security parameter k, we view the oracle as a function $\mathcal{O} : \{0,1\}^* \to \{0,1\}^{kq(k)}$.

We define a family $F = \{F_k\}_{k \in \mathbb{N}}$ of trapdoor functions in the random oracle model as follows: We associate to any $p \in [P_k]$ the function f defined on input $b_1 \dots b_k \in \{0,1\}^k$ by

$$f(b_1 \cdots b_k) = p(b_1; r_1) \| \cdots \| p(b_k; r_k),$$

where

$$r_1 \| \cdots \| r_k = \mathcal{O}(b_1 \cdots b_k),\ r_i \in \{0,1\}^{q(k)}\ .$$

The generator $F\text{-}Gen$ takes input 1^k, runs $(p, tp) \leftarrow P\text{-}Gen(1^k)$ and outputs (f, tp) where f is as defined above. It is clear that f can be evaluated in polynomial time using the evaluator $P\text{-}Eval$ for p.

Notice that f can be inverted given the trapdoor information. Given f, tp, and $y_1 \| \cdots \| y_k = f(b_1 \dots b_k)$, inverter $F\text{-}Inv$ computes $b_i = P\text{-}Inv(p, tp, y_i)$ for $i = 1, \dots, k$, and outputs $b_1 \dots b_k$. Furthermore, f is injective because P has zero decryption error: in this inversion process, $P\text{-}Inv$ correctly returns b_i, so we correctly recover the full input. It remains to show that F is one-way.

Claim 5.4 *F is one-way.*

We prove this claim by describing several probabilistic experiments, modifying the role of the oracle with each experiment. The first arises from the definition of a family of one-way functions in the random oracle model. Let A be any PPT, let k be any positive integer, and let $q = q(k)$.

Experiment 5.5

(1) Choose a random oracle $\mathcal{O} : \{0,1\}^* \to \{0,1\}^{kq(k)}$.
(2) Choose $p \leftarrow P_k$
(3) Select $b_1, \dots, b_k$ uniformly and independently from $\{0,1\}$.
(4) Let $r_1 \| \cdots \| r_k = \mathcal{O}(b_1 \cdots b_k)$, where $|r_i| = q(k)$ for each i.
(5) Let $x = p(b_1; r_1) \| \cdots \| p(b_k; r_k)$.
(6) Compute $z \leftarrow A^{\mathcal{O}}(1^k, p, x)$.

We need to prove the following:

Claim 5.6 *For every PPT A, the probability that $z = b_1 \cdots b_k$ in Experiment 5.5 is a negligible function of k.*

To prove Claim 5.6, we first analyze what happens when the r_i's are chosen independently of the oracle, as in the following experiment: Let A be any PPT, let k be any positive integer, and let $q = q(k)$.

Experiment 5.7
(1)–(3) As in Experiment 5.5.
(4) Select $r_1, \ldots, r_k$ uniformly and independently from $\{0,1\}^q$.
(5)–(6) As in Experiment 5.5.

Claim 5.8 *For every PPT A, the probability that $z = b_1 \cdots b_k$ in Experiment 5.7 is a negligible function of k.*

Claim 5.8 follows from standard direct product arguments [Ya, GNW]. Specifically, Claim 5.8 is a special case of the uniform complexity version of the Concatenation Lemma in [GNW, Lemma 10].

Claim 5.9 *For every PPT A, the probability that $\mathcal{O}$ is queried at point $b_1 \cdots b_k$ during the execution of $A^{\mathcal{O}}(1^k, p, x)$ in Step 6 of Experiment 5.7 is a negligible function of k.*

Proof: Suppose that the probability that $\mathcal{O}$ is queried at point $b_1 \cdots b_k$ was greater that $1/s(k)$ for infinitely many k, where s is a polynomial. Then we could obtain a PPT A' that violates Claim 5.8 as follows. Let $t(k)$ be a polynomial bound on the running time of A. A' does the following on input $(1^k, p, x)$:

(1) Select i uniformly from $\{1, \ldots, t(k)\}$.
(2) Simulate A on input $(1^k, p, x)$, with the following changes:
  (1) Replace the oracle responses with strings randomly selected on-line, with the condition that multiple queries at the same point give the same answer.
  (2) Halt the simulation at the i'th oracle query and let w be this query.
(3) Output w.

Then A', when used in Experiment 5.7, outputs $b_1 \cdots b_k$ with probability greater that $1/(s(k)t(k))$ for infinitely many k, which contradicts Claim 5.8. ▮

In order to deduce Claim 5.6 from Claims 5.8 and 5.9, we give an equivalent reformulation of Experiment 5.5: Let A be any PPT, let k be any positive integer, and let $q = q(k)$.

Experiment 5.10
(1)–(3) As in Experiment 5.5.
(4) Select $r_1, \ldots, r_k$ uniformly and independently from $\{0,1\}^q$.
(5) Let $x = p(b_1; r_1) \| \cdots \| p(b_k; r_k)$.
(6) Modify $\mathcal{O}$ at location $b_1 \cdots b_k$ to have value $r_1 \| \cdots \| r_k$.
(7) Compute $z \leftarrow A^{\mathcal{O}}(1^k, p, x)$.

We now argue that Experiment 5.10 is equivalent to Experiment 5.5. In Experiment 5.5, $r_1, \ldots, r_k$ are uniformly and independently distributed in $\{0,1\}^q$ and after Step 5 of Experiment 5.5 the only information about the oracle that has been used is that $r_1 \| \cdots \| r_k = \mathcal{O}(b_1 \cdots b_k)$. Thus, the final distribution on all random variables are identical in the two experiments and it suffices to prove Claim 5.6 for Experiment 5.10 rather than Experiment 5.5.

Proof: Let E be the event that $z = b_1 \cdots b_k$ in Experiment 5.10. Let F be the event that $\mathcal{O}$ is queried at point $b_1 \cdots b_k$ during the execution of $A^{\mathcal{O}}(p, x)$ in Step 7 of Experiment 5.10. To show that E occurs with negligible probability, it suffices to argue that both F and $E \wedge \overline{F}$ occur with negligible probability.

First we show that F occurs with negligible probability. Notice that whether or not $A^{\mathcal{O}}$ queries $\mathcal{O}$ at $b_1 \cdots b_k$ in Experiment 5.10 will not change if Step 6 is removed. This is because its behavior cannot be affected by the change in $\mathcal{O}(b_1 \cdots b_k)$ until it has already queried that position of the oracle. If Step 6 is removed from Experiment 5.10, we obtain Experiment 5.7. Hence, the probability of F is negligible by Claim 5.9.

Similarly, the probability that $[z = b_1 \cdots b_k$ and $A^{\mathcal{O}}$ never queries the oracle at $b_1 \cdots b_k]$ will not change if Step 6 is removed. Thus, the probability of $E \cap \overline{F}$ is bounded above by the probability that $z = b_1 \cdots b_k$ in Experiment 5.7, which is negligible by Claim 5.8. ▮

Remark 5.11 If the family of unapproximable trapdoor predicates we start with has negligible decryption error, then the family of trapdoor functions we construct will in general also have negligible decryption error and may fail to be injective with some small probability.

By first reducing the decryption error of the predicate family to $\exp(-\Omega(k^3))$ as in the proof of Theorem 4.1 and then using the oracle to derandomize the inversion algorithm, one can produce an *injective* family that has *zero* decryption error with probability $1 - 2^{-k}$ (where the probability is just taken over the choice of the oracle).

Acknowledgments

The starting point of this research was a question posed to us by Shafi Goldwasser, namely whether trapdoor permutations could be built from the assumptions underlying the Ajtai-Dwork cryptosystem.

Thanks to Oded Goldreich, Shafi Goldwasser, and the members of the Crypto 98 program committee for their comments on the paper.

References

[AjDw] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case / average-case equivalence. *Proceedings of the* 29th *Annual Symposium on the Theory of Computing*, ACM, 1997.

[AMM] Adleman, Manders and Miller. On taking roots in finite fields. *Proceedings of the* 18th *Symposium on Foundations of Computer Science*, IEEE, 1977.

[BHSV] M. Bellare, S. Halevi, A. Sahai, and S. Vadhan. Many-to-one trapdoor functions and their relation to public-key cryptosystems. *Advances in Cryptology – Crypto* 98 *Proceedings*, Lecture Notes in Computer Science Vol. 1462, H. Krawczyk ed., Springer-Verlag, 1998.

[BeRo] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. *Proceedings of the First Annual Conference on Computer and Communications Security*, ACM, 1993.

[Be] E. Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation*, Vol. 24, 1970, pp. 713–735.

[BlMi] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits, *SIAM Journal on Computing*, Vol. 13, No. 4, 850-864, November 1984.

[Ca] R. Canetti. Towards realizing random oracles: Hash functions that hide all partial information. *Advances in Cryptology – Crypto* 97 *Proceedings*, Lecture Notes in Computer Science Vol. 1294, B. Kaliski ed., Springer-Verlag, 1997.

[CGH] R. Canetti, O. Goldreich and S. Halevi. The random oracle model, revisited. *Proceedings of the* 30th *Annual Symposium on the Theory of Computing*, ACM, 1998.

[DiHe] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Info. Theory*, Vol. IT-22, No. 6, November 1976, pp. 644–654.

[DDN] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. *Proceedings of the* 23rd *Annual Symposium on the Theory of Computing*, ACM, 1991.

[ElG] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, Vol. 31, 1985, pp. 469–472.

[GoLe] O. Goldreich and L. Levin. A hard predicate for all one-way functions. *Proceedings of the* 21st *Annual Symposium on the Theory of Computing*, ACM, 1989.

[GoMi] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, Vol. 28, April 1984, pp. 270–299.

[GNW] O. Goldreich, N. Nisan, and A. Wigderson. On Yao's XOR Lemma. *Electronic Colloquium on Computational Complexity*, TR95-050. March 1995. `http://www.eccc.uni-trier.de/eccc/`

[HILL] J. Håstad, R. Impagliazzo, L. Levin and M. Luby. Construction of a pseudo-random generator from any one-way function. Manuscript. Earlier versions in STOC 89 and STOC 90.

[ImLu] R. Impagliazzo and M. Luby. One-way Functions are Essential for Complexity-Based Cryptography. *Proceedings of the* 30th *Symposium on Foundations of Computer Science*, IEEE, 1989.

[ImRu] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. *Proceedings of the* 21st *Annual Symposium on the Theory of Computing*, ACM, 1989.

[NaYu] M. Naor and M. Yung. Public-Key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. *Proceedings of the* 22nd *Annual Symposium on the Theory of Computing*, ACM, 1990.

[Rab] M. Rabin. Digitalized Signatures and Public Key Functions as Intractable as Factoring. *MIT/LCS/TR-212,* 1979.

[Ya] A. Yao. Theory and applications of trapdoor functions. *Proceedings of the* 23rd *Symposium on Foundations of Computer Science*, IEEE, 1982.

A Using a PRG in the construction of Section 5

Below we show that implementing the oracle of the construction of Section 5 by a pseudorandom generator may result in a function family which is not one-way. Oded Goldreich has suggested a simpler counterexample that works with a slight generalization of our construction from Section 5, which we describe at the end. We start by recalling the definition of a pseudorandom generator.

Definition A.1 [Pseudorandom number generators, [BlMi, Ya]] Let G be a (deterministic) polynomial time algorithm which given a *seed* $x \in \{0,1\}^k$ outputs a string of length $l(k) > k$, for any $k \in \mathbb{N}$. We say G is a pseudorandom number generator if for every PPT algorithm A, the function of k given by

$$\left|\Pr\left[\, A(G(x)) = 1 \;:\; x \leftarrow \{0,1\}^k \,\right] - \Pr\left[\, A(y) = 1 \;:\; y \leftarrow \{0,1\}^{l(k)} \,\right]\right|$$

is negligible.

Below we show that (if there exist pseudorandom number generators and unapproximable trapdoor predicate families, then) there exists a pseudorandom number generator G' and an unapproximable trapdoor predicate family P' for which applying the construction of Section 5 (with G' replacing the random oracle) results in a function family which is not one-way.

In the construction below, let $P = \{P_k\}_{k \in \mathbb{N}}$ be any unapproximable trapdoor predicate family and let $q'(k)$ be a polynomial upper bound on the number of random bits used by $p \in [P_k]$. Below we often use the shorthand $q = q'(k) + 1$. Also, let G be any pseudorandom generator with length function $l(k) = 2kq$. We start by modifying P and G to get a new trapdoor family P' and a new pseudorandom generator G'.

THE MODIFIED GENERATOR. The new generator takes as input a $2k$ bit seed $s' = b_1 \cdots b_{2k}$, and will output a string of length $2kq$, as follows:

(1) Let $s = (b_1 \oplus b_2) \| (b_3 \oplus b_4) \| \cdots \| (b_{2k-1} \oplus b_{2k})$. This is a k bit string which will be viewed as a seed for G.

(2) Compute $G(s)$ and divide this $2kq$ bit string into $2k$ blocks each of length q, namely $G(s) = r_1 \| r_2 \| \cdots \| r_{2k}$.

(3) Denoting the j-th bit of r_i by $r_{i,j}$, set

$$r'_i = \begin{cases} b_{i+1} r_{i,2} \cdots r_{i,q} & \text{if } i \text{ is odd} \\ r_i & \text{if } i \text{ is even} \end{cases}$$

Namely, replace the first bits of $r_1, r_3, \ldots, r_{2k-1}$ with the bits $b_2, b_4, \ldots, b_{2k}$ respectively.

(4) Output $r'_1 \| r'_2 \| \cdots \| r'_{2k}$.

Claim A.2 *If G is a pseudorandom generator, then so is G'.*

Proof: The intuition is that the bits $b_2, b_4, \ldots, b_{2k}$ are independent of the string $G(s)$. A formal proof can be given that if there was a way to break G' then there would be a way to break G. Details omitted. ▮

THE MODIFIED PREDICATE FAMILY. Next we define a modified predicate family $P' = \{P'_k\}_{k \in \mathbb{N}}$. We associate to any $p \in [P_k]$ a predicate $p' \in [P'_k]$ which takes input $b \in \{0,1\}$ and coins $s_1 \| s$, where s_1 is a bit and $|s| = q'(k)$, and is defined by

$$p'(b; s_1 \| s) \; = \; (s_1 \oplus b) \| p(b; s) \; .$$

The distribution on P'_k is that given by selecting $p \leftarrow P_k$ and setting p' to the above. The generator P'-*Gen* on input 1^k outputs (p', tp) where $(p, tp) \leftarrow P\text{-}Gen(1^k)$. The inversion algorithm P'-*Inv* is given $p', tp, c' \| y$ where c' is a bit. It lets $b' \leftarrow P\text{-}Inv(p, tp, y)$ and outputs b'. Thus P' has the same decryption error as P.

Claim A.3 *If P is an unapproximable trapdoor predicate family, then so is P'.*

Proof: The intuition is that $s_1 \oplus b$ is independent of both b and $p(b; s)$. A formal proof can be given that the ability to predict P' implies the ability to predict P. Details omitted. ▮

THE CONSTRUCTION FAILS. Now apply the construction of Section 5 to P' using G' as the randomizing function. The resulting function family $F = \{F_k\}_{k \in \mathbb{N}}$ associates to any $p' \in [P'_k]$ a function $f \in [F_k]$ defined like this: for any $b_1, \ldots, b_{2k} \in \{0,1\}$ set

$$f(b_1 b_2 \cdots b_{2k}) \; = \; p'(b_1; r'_1) \| p'(b_2; r'_2) \| \cdots \| p'(b_{2k}; r'_{2k}) \; ,$$

where $r'_1 \| r'_2 \cdots \| r'_{2k} = G'(b_1 b_2 \cdots b_{2k})$. The distribution on F_k is that given by selecting $p' \leftarrow P'_k$ and outputting f as defined above.

Claim A.4 *F is not one-way.*

Proof: We show that there exists an inversion algorithm I which given p and $w = F_p(b_1 \cdots b_{2k})$, recovers $b_1 \cdots b_{2k}$ with probability one. Letting $r'_1 \| r'_2 \cdots \| r'_{2k} = G'(b_1 b_2 \cdots b_{2k})$, recall that by definition we have

$$\begin{aligned} f(b_1 \cdots b_{2k}) &= p'(b_1; r'_1) \| p'(b_2; r'_2) \| \cdots p'(b_{2k}; r'_{2k}) \\ &= (b_1 \oplus r'_{1,1}) \ \| \ p(b_1; r'_{1,2} \cdots r'_{1,q}) \| \\ &\quad\ (b_2 \oplus r'_{2,1}) \ \| \ p(b_2; r'_{2,2} \cdots r'_{2,q}) \| \\ &\qquad\qquad \vdots \\ &\quad\ (b_{2k} \oplus r'_{2k,1}) \| \ p(b_{2k}; r'_{2k,2} \cdots r'_{2k,q}) \end{aligned}$$

and therefore the first bit of the i'th block in $p'(b_1 \cdots b_{2k})$ is always $b_i \oplus r'_{i,1}$. However, by the construction of G' we know that for odd i we have $r'_{i,1} = b_{i+1}$. Hence, for odd i, the first bit in the i'th block is simply $b_i \oplus b_{i+1}$.

Denote now $s = (b_1 \oplus b_2) \| (b_3 \oplus b_4) \| \cdots \| (b_{2k-1} \oplus b_{2k})$. Then s can be recovered from $p'(b_1 \cdots b_{2k})$ simply by concatenating the first bits of all the odd blocks. We can now compute $G(s)$, and chop it up as before as $G(s) = r_1 \| r_2 \| \cdots \| r_{2k}$. Again by construction of G' we know that for all i, $r'_{i,2} \cdots r'_{i,q} = r_{i,2} \cdots r_{i,q}$. Finally, for all i we can evaluate both $p(0; r'_{i,2} \cdots r'_{i,q})$ and $p(1; r'_{i,2} \cdots r'_{i,q})$ and compare them to the value of $p(b_i; r'_{i,2} \cdots r'_{i,q})$ which we know, thus recovering b_i. ▮

An alternative counterexample. Our construction from Section 5 can be viewed directly in terms of a semantically secure encryption scheme [GoMi] as follows: Let $\{E_e\}$ be a semantically secure encryption scheme, where e varies over the possible public encryption keys. We write $E_e(m, r)$ for the encryption of message m using randomness r. Then we can define a family $F = \{f_e\}$ of trapdoor functions in the random oracle model as follows:

$$f_e(m) = E_e(m, \mathcal{O}(m)).$$

The construction we present in Section 5 is the special case when the encryption scheme is constructed from a trapdoor predicate family via the Goldwasser-Micali construction [GoMi]:

$$E_p(b_1 \cdots b_k, r_1 \| \cdots \| r_k) = p(b_1; r_1) \| \cdots \| p(b_k; r_k)$$

By a proof similar to the one in Section 5, one can show that even for an arbitrary secure encryption scheme $\{E_e\}$, $\{f_e\}$ is a family of one-way, injective trapdoor functions in the random oracle model. However, a simpler construction than the one above can be used to show that F is not necessarily one-way when the random oracle is replaced with a pseudorandom generator. This construction is due to Oded Goldreich. Let $\{E_e\}$ be a secure encryption scheme as above and let G be a pseudorandom generator. Then define

$$E'_e(m, r) = \begin{cases} m & \text{if } G(m) = r \\ E_e(m, r) & \text{otherwise} \end{cases}$$

$\{E'_e\}$ is still a semantically secure encryption scheme, as $G(m)$ will equal r with exponentially small probability. However, $\{f'_e\}$ constructed as above using $\{E'_e\}$ and G will not be one-way:

$$f'_e(m) = E'_e(m, G(m)) = m.$$