SIAM J. COMPUT.
Vol. 17, No. 2, April 1988

011

HOW TO CONSTRUCT PSEUDORANDOM PERMUTATIONS FROM PSEUDORANDOM FUNCTIONS*

MICHAEL LUBY† AND CHARLES RACKOFF‡

Abstract. We show how to efficiently construct a pseudorandom invertible permutation generator from a pseudorandom function generator. Goldreich, Goldwasser and Micali ["How to construct random functions," Proc. 25th Annual Symposium on Foundations of Computer Science, October 24–26, 1984.] introduce the notion of a pseudorandom function generator and show how to efficiently construct a pseudorandom function generator from a pseudorandom bit generator. We use some of the ideas behind the design of the Data Encryption Standard for our construction. A practical implication of our result is that any pseudorandom bit generator can be used to construct a block private key cryptosystem which is secure against chosen plaintext attack, which is one of the strongest known attacks against a cryptosystem.

Key words. cryptography, pseudorandom, Data Encryption Standard, security

AMS(MOS) subject classifications. 68P25, 68Q15, 68Q25

1. Introduction. The main result of this paper is a method for efficiently constructing a pseudorandom invertible permutation generator from a pseudorandom function generator. The question of whether pseudorandom permutation generators exist was first posed by Goldreich et al. [GGM]. A practical application of our result is that a pseudorandom bit generator can be used to efficiently construct a block private key cryptosystem which is provably secure against chosen plaintext attack, which is one of the strongest possible attacks known against a cryptosystem. We expect that there will be other applications of pseudorandom invertible permutation generators in cryptographic protocols. Before describing in detail our results, we discuss the basic questions which motivated our work and the partial answers we can give to these questions.

The field of cryptography has changed dramatically over the past few years. Several years ago, cryptography was an art more than a science. Cryptosystem design was based on clever ad hoc ideas. The security of the cryptosystem rested solely on the cleverness of the designer; a cryptosystem was deemed secure until it was broken (in many cases the cryptographer only knew it was broken long after the breaker). Certain design rules were recognized as playing a crucial role in improving security. Eventually, the apparent increase in security obtained by using these design rules became part of the folklore, although there were no formal proofs that security was increased. The foundations of cryptography were not yet established, there was no established formal framework for talking about a cryptographic protocol, security, etc. Thus, there was no formal way to either confirm or dispute the folklore.

More recently, researchers have formalized the notions of cryptographic protocols and security and today cryptography is an integral part of computational complexity. Many cryptosystems whose security is based on mathematical assumptions have come out of this research. However, the security of cryptosystems typically used in practice is still based on the old folklore. This paper uses the new cryptographic rigor to formalize and to analyze some of the previously unexamined folklore.

* Received by the editors November 22, 1985; accepted for publication (in revised form) March 2, 1987.

† Department of Computer Science, University of Toronto, Toronto, Canada M5S 1A4. The work of this author was supported by Natural Sciences and Engineering Research Council of Canada grant A8092 and University of Toronto grant 3-370-126-40.

‡ Department of Computer Science, University of Toronto, Toronto, Canada M5S 1A4. The work of this author was supported by Natural Sciences and Engineering Research Council of Canada grant A3611.

Our primary motivation when we started this work was to investigate the soundness of some of the design rules used in the development of the Data Encryption Standard (hereafter called DES). DES, which was designed to be used as a block private key cryptosystem, was developed by IBM for the U.S. National Bureau of Standards, with an undisclosed amount of highly classified kibbitzing by the super-secret National Security Agency. The reader can consult [De] for more information about DES and [Ba] for more information about the National Security Agency and its role in the development of DES.

DES has features which make it very practical to use. However, one of the properties that DES does not have (as far as we know) is that it is provably secure, even assuming some reasonable mathematical hypothesis. DES certainly has several very clever features, but there is no formal justification that these features help achieve security. We abstract and formalize what we think are some of the more interesting design features of DES to see if they can be used to achieve security or if they are inherently flawed.

The construction of a pseudorandom invertible permutation generator from a pseudorandom function generator is based on one of the main design features of DES. We view this as partial justification for the use of this design rule in DES. What our result intuitively says is that if DES used pseudorandom function generators in its construction then it would be secure when used as a block private key cryptosystem. However, this is a weak justification because the function generators used in DES are not at all pseudorandom. Our result is more of a justification of the use of the design rule than it is a statement about the security of the entire DES system.

Section 2 gives terminology used throughout the remainder of the paper. In § 3, we discuss block private key cryptosystems, describe how DES is used as a block private key cryptosystem, make the connection between secure block private key cryptosystems and pseudorandom invertible permutation generators, and present the design rule of DES which we use to construct a pseudorandom invertible permutation generator from a pseudorandom function generator. In § 4, we give formal definitions of pseudorandom bit generators, pseudorandom function generators and pseudorandom permutation generators and state our main results. In § 5, we give the construction for an invertible permutation generator from a function generator and prove that if the function generator is pseudorandom then the invertible permutation generator is pseudorandom. A preliminary version of these results appears in [LR].

2. Terminology. A **string** is a bit string. Let $\{0,1\}^n$ be the set of all 2^n strings of length n. Let a and b be two equal length strings. Define $a \oplus b$ to be the bit-by-bit **exclusive or** of a and b, where the resulting string has the same length as a. Let $a \bullet b$ denote the **concatenation** of the two strings a and b.

Let F^n be the set of all 2^{n2^n} functions mapping $\{0,1\}^n$ into $\{0,1\}^n$. Let f_1 and f_2 be functions in F^n. We use $f_1 \circ f_2$ to denote the function in F^n which is the composition of f_1 and f_2. Let $P^n \subset F^n$ be the set of such functions that are permutations, i.e., they are 1-1 onto functions.

In all cases, a random choice of an object from a set of objects is such that each object is equally likely to be chosen. For example, a random choice of a string from $\{0,1\}^n$ will choose each such string with probability $1/2^n$. As another example, a random choice of a function from F^n will choose each such function with probability $1/2^{n2^n}$.

3. Connections between cryptosystems and invertible permutation generators. Suppose agent A wants to send plaintext to agent B. A wants to do this securely, i.e., in

a way such that any agent L, who is able to read all the information sent from A to B, has no significant idea about the content of the plaintext. A very practical way to achieve this goal is for A and B to use a secure block private key cryptosystem. Both A and B, but not L, have the same randomly chosen private key k. When A sends plaintext M to B, M is partitioned into equal length plaintext blocks. A encrypts each plaintext block into a ciphertext block of the same length using k. B, upon receiving a ciphertext block, decrypts it back into the plaintext block using k.

There are two very attractive features of block private key cryptosystems. First, the total number of bits sent from A to B is the minimum number possible, because the number of encrypted bits sent from A to B is exactly the same as the number of plaintext bits. Second, since the plaintext is encrypted in blocks, if the encryption of one block is lost in transmission then the other blocks can still be decrypted by B, whether or not B knows that a block was lost. Other cryptosystems tend to send many more encrypted bits than there are plaintext bits and/or are not robust to transmission losses.

There is one inherent weakness in any block private key cryptosystem, which is that any listener L can always tell if exactly the same plaintext block is repeated from the encryption. In our definition of a secure block private key cryptosystem, this is essentially the only insecurity of the system. There are encryption systems which avoid this problem, but these systems tend to be less efficient than a block private key cryptosystem, in certain respects, as explained above.

As far as we know, there is no provably secure block private key cryptosystem. One of the main implications of the results given in this paper is that, under the assumption that there is a pseudorandom bit generator, there is a provably secure block private key cryptosystem.

One of the main motivations of this work is to study the security of DES, when it is used as a block private key cryptosystem. In this context, DES works as follows. Both A and B, but not L, have the same 56-bit private key k. Suppose A wants to send plaintext M to B. A first partitions M into plaintext blocks of 64 bits each. A encrypts each plaintext block, using the DES encryption algorithm with key k, into a 64-bit ciphertext block and sends the ciphertext blocks to B. B decrypts the ciphertext blocks, using the DES decryption algorithm with key k, to recover the plaintext blocks. One important property of DES is that, given k, it is easy to encrypt and decrypt. An important implication of this is that, given k, each string of length 64 has a unique encryption and a unique decryption.

The encryption algorithm for DES can be thought of as a family of 2^{56} permutations $h^{64} = \{h_k^{64}: k \in \{0,1\}^{56}\}$, where each permutation is a member of P^{64} indexed by a key k. Similarly, the decryption algorithm for DES can be thought of as a family of 2^{56} permutations $\bar{h}^{64} = \{\bar{h}_k^{64}: k \in \{0,1\}^{56}\}$, where each permutation is a member of P^{64} indexed by a key k. Furthermore, $h_k^{64} \circ \bar{h}_k^{64}$ and $\bar{h}_k^{64} \circ h_k^{64}$ are both the identity permutation. DES has the property that, given k and α, both $h_k^{64}(\alpha)$ and $\bar{h}_k^{64}(\alpha)$ can be computed very efficiently.

One of the strongest attacks known against a cryptosystem is a chosen plaintext attack. We would deem DES secure if it was secure against chosen plaintext attack (which is a stronger notion of security than that given in the first paragraph of this section). Intuitively, in a chosen plaintext attack an agent L, who does not know the key k, is nevertheless able to trick A into sending to B encryptions of plaintext blocks chosen by L. L is allowed to choose a "reasonable" number of plaintext blocks $M_1, \cdots, M_i$, A encrypts these plaintext blocks and L sees the encryptions of these plaintext blocks $h_k^{64}(M_1), \cdots, h_k^{64}(M_i)$. During this process, L interactively chooses

the next plaintext block for A to encrypt based on all previous plaintext blocks and their encryptions. Let M_{i+1} be a plaintext block chosen by A which is different from all plaintext blocks chosen by L during the attack. Intuitively, the cryptosystem is secure against L if L cannot predict M_{i+1} given the information gained from the attack and given the encryption $h_k^{64}(M_{i+1})$ of M_{i+1} (but not M_{i+1}) "significantly" better than if L had not received the information obtained from the attack and from seeing $h_k^{64}(M_{i+1})$. The cryptosystem is secure against chosen plaintext attack if it is secure against all such agents L.

The apparent security of DES when it is used as a block private key cryptosystem rests on the fact that DES seems to pass the black box test, which was informally suggested by Turing [Hod]. The black box test is the following:

> Say that we have two black boxes, one of which computes a fixed randomly chosen function from F^{64} and the other computes h_k^{64} for a fixed randomly chosen k. Then no algorithm which examines the boxes by feeding inputs to them and looking at the outputs can obtain, in a "reasonable" time, any "significant" idea about which box is which.

If DES passes the black box test then it is secure against a chosen plaintext attack when used as a block private key cryptosystem. We do not give a formal definition of security for a block private key cryptosystem. A rigorous definition in a somewhat different setting appears in [Ra].

The black box test, and therefore security, is very informally stated, i.e., the terms "reasonable" and "significant" are not well defined. The tools of mathematics and computer science are designed to analyze asymptotic behavior; to utilize these tools we introduce an asymptotic version of DES and define security in terms of asymptotic security. We introduce a collection of private key block cryptosystems, one for each possible plaintext block length n. Let $h = \{h^n : n \in \mathbb{N}\}$ where, for each n, h^n is the generalization of h^{64} defined for DES, i.e., h^n is used to encrypt plaintext blocks of length n. Similarly, let $\bar{h} = \{\bar{h}^n : n \in \mathbb{N}\}$, where for each n, $\bar{h}^n$ is the generalization of $\bar{h}^{64}$ defined for DES, i.e., $\bar{h}^n$ is used to decrypt plaintext blocks of length n. Thus, both h^n and $\bar{h}^n$ specify for each key k of a given length a permutation, $h_k^n \in P^n$, $\bar{h}_k^n \in P^n$, respectively, where $h_k^n \circ \bar{h}_k^n$ is the identity permutation. We require that, given $\alpha \in \{0, 1\}^n$ and a key k of a given length, both $h_k^n(\alpha)$ and $\bar{h}_k^n(\alpha)$ can be computed in time polynomial in n. In the terminology of § 4, both h and $\bar{h}$ are invertible permutation generators. A very similar notion, a function generator, was first formally defined in [GGM]. We give the definition of a function generator in § 4, but intuitively it is the same as the definition of an invertible permutation generator except that each function specified by n and k is not necessarily 1-1 onto function, i.e., not necessarily a permutation. Thus, an invertible permutation generator is a special case of a function generator.

It is enough that h be pseudorandom for h to be secure against chosen plaintext attack when used as a block private key cryptosystem. The concept of a pseudorandom function generator was introduced in [GGM]. An invertible permutation generator is pseudorandom if it is a pseudorandom function generator. Informally, a pseudorandom function generator is a function generator which passes the black box test for all sufficiently large n, where n replaces all occurrences of 64 in the description of the black box test, "reasonable" time corresponds to time polynomial in n and "significant" corresponds to probability polynomial in $1/n$. We give a formal definition of a pseudorandom function generator in § 4.

Goldreich et al. [GGM] show how to construct a pseudorandom function generator

from a pseudorandom bit generator. Pseudorandom function generators have many applications in cryptography, but, unless they are also invertible permutation generators, they cannot be used directly in a block private key cryptosystem as described above. A natural question to ask is can we construct a pseudorandom invertible permutation generator from a pseudorandom function generator. In § 5, we give an efficient construction of an invertible permutation generator based on a function generator and we prove that if the function generator is pseudorandom then so is the invertible permutation generator. This construction uses some of the ideas behind the design of DES. Below, we describe the design details of DES relevant to our construction together with some comments about DES.

DES begins with f^{32}, where f^{32} specifies for each key k of length 48 a function $f_k^{32} \in F^{32}$. Alternatively, f^{32} can be thought of as an algorithm which, given k and an input α, computes $f_k^{32}(\alpha)$. Let $L, R \in \{0,1\}^{32}$. Define g^{64} such that for each key k of length 48, $g_k^{64}(L \bullet R) = R \bullet [L \oplus f_k^{32}(R)]$. It is easy to see that g_k^{64} is 1-1 onto and easy to invert if k is known. Let h^{64} be g^{64} composed with itself 16 times, so that the key length function for h^{64} is $768 = 48 \cdot 16$. Given a 768-bit key k, h_k^{64} is 1-1 onto and easily invertible. We refer to h^{64} as MDES for modified DES. MDES differs from DES in certain inconsequential ways, but also in one way that might be very important: DES only has a 56-bit key, which is used to generate (in a very simple way) the $16 \cdot 48$-bit key to be used in MDES. It is not clear if this makes DES more or less secure than MDES, but most observers feel that MDES would be much more secure than DES. In any case, no one has yet succeeded (as far as we know) in breaking DES.

The f^{32} used in DES is not by any stretch of the imagination pseudorandom. The creators of DES apparently feel that f^{32} does enough "mixing up" so that, together with the rest of the construction, DES as a whole is secure. Our construction replaces f^{32} with a pseudorandom function generator f. Invertible permutation generator g is constructed from f as described above. The invertible permutation generator h is formed by composing g three times with itself (instead of 16 times as in MDES). We prove in § 5 that h formed in this way is a pseudorandom invertible permutation generator if f is a pseudorandom function generator.

4. Formal definitions and statement of results. In this section we present some necessary formal definitions and state the main results of the paper.

4.1. Pseudorandom bit generators. The original definitions for pseudorandom bit generators are due to Blum and Micali [BM] and are generalized to those given here by Yao [Yao]. A **bit generator** is a collection of functions $f = \{f^n : n \in \mathbb{N}\}$ such that f^n maps $\{0,1\}^n$ into $\{0,1\}^{t(n)}$, where $t(n) \geqq n+1$ (for example, $t(n) = n^3$) and, given n and α, $f^n(\alpha)$ can be computed in time polynomial in n (this implies that $t(n)$ is polynomial in n). Informally, f is pseudorandom if there is no (probabilistic) polynomial in n time algorithm which, for infinitely many n, can significantly distinguish a string $\beta = f^n(\alpha)$, where α is a randomly chosen from $\{0,1\}^n$, from a string randomly chosen from $\{0,1\}^{t(n)}$. Formally, f is **pseudorandom** if there is no distinguishing circuit family for f. A distinguishing circuit family for f is an infinite family of Boolean circuits consisting of and/or/not unbounded fan-out gates (for readers unfamiliar with this model of computation, replace all occurrences of a family of Boolean circuits with a probabilistic polynomial time algorithm and all the results are the same) $C = \{C_{n_1}, C_{n_2}, \cdots\}$, where $n_1 < n_2 \cdots$, such that for some pair of constants s and c, for each n for which there is a circuit C_n:

(1) The size of C_n is less than or equal to n^s. The size of a circuit is the total number of wires plus the number of gates in the circuit.

(2) The input to C_n is a string of length $t(n)$ and the output is a single bit.

(3) Let r_n be the probability that the output bit of C_n is one when the input to C_n is a string randomly chosen from $\{0, 1\}^{t(n)}$. Let p_n be the probability that the output bit of C_n is 1 when a string α is randomly chosen from $\{0, 1\}^n$ and the input is $f^n(\alpha)$. Let $d_n = |p_n - r_n|$ be the distinguishing probability for C_n. Then, $d_n \geqq 1/n^c$.

It is important to note that if f is a pseudorandom bit generator, then f is a pseudorandom bit generator even if we allow distinguishing circuit families for f to be probabilistic. A probabilistic circuit C_n, besides the inputs described above, has an additional number of input bits whose values are randomly chosen. It is not hard to see that there is some way of fixing these additional bits to C_n, forming a deterministic circuit C'_n, such that the distinguishing probability for C'_n is at least as great as the distinguishing probability for C_n, i.e., $d'_n \geqq d_n$. Hence, if f has a distinguishing probabilistic circuit family then f has a distinguishing deterministic circuit family, which implies that if f has no distinguishing deterministic circuit family then f has no distinguishing probabilistic circuit family.

Whether or not pseudorandom bit generators exist is an open question. Blum and Micali [BM] introduce conditions which are sufficient for constructing pseudorandom bit generators. They show how to construct a pseudorandom bit generator based on the assumption that the Discrete Log problem is hard. Yao [Yao] generalizes these conditions and shows how to construct a pseudorandom bit generator based on the assumption that the factoring problem is hard. A series of results has improved the efficiency of the Yao construction [ACGS], [BCS], [GMT], and [VV]. Levin [Le] introduces weaker conditions which are sufficient to construct a pseudorandom bit generator.

4.2. Pseudorandom function generators. The concept of a pseudorandom function generator is defined in [GGM]. They give a construction for a pseudorandom function generator using a pseudorandom bit generator. We give the definition of a pseudorandom function generator in this section. Let $l(n)$ be polynomial in n. A **function generator** with key length function $l(n)$ is a collection $f = \{f^n : n \in \mathbb{N}\}$, where f^n specifies for each key k of length $l(n)$ a function $f_k^n \in F^n$. It is required that, given a key $k \in \{0, 1\}^{l(n)}$, and a string $\alpha \in \{0, 1\}^n$, $f_k^n(\alpha)$ can be computed in time polynomial in n.

Informally, f is pseudorandom if there is no polynomial time in n algorithm which, for infinitely many n, is able to even slightly distinguish whether a function was randomly chosen from f^n or from F^n after seeing polynomial in n input/output pairs of the function, even when the algorithm is allowed to choose the next input based on all previously seen input/output pairs. Formally, f is **pseudorandom** if there is no distinguishing circuit family for f. A distinguishing circuit family for f is an infinite family of circuits $\{C_{n_1}, C_{n_2}, \cdots\}$, where $n_1 < n_2 \cdots$, such that for some pair of constants s and c, for each n for which there is a circuit C_n:

(1) C_n is an acyclic circuit which contains Boolean gates of type **and**, **or** and **not** (these gates are interpreted in the usual way, i.e., the **and** gate computes the "and" of the two inputs). In addition there are constant gates of type **zero** and **one**. Each constant gate has no inputs, and the output is 0 if it is a **zero** gate and 1 if it is a **one** gate. C_n also contains **oracle gates**. Each oracle gate has an input and an output which are both strings of length n. Each oracle gate is to be evaluated using some function from F^n, which for now is left unspecified and is to be thought of as a variable of the circuit. All gates can fan-out their output bits to an unbounded number of other gates. The output of C_n is a single bit. Such a circuit is called an **oracle circuit**.

(2) The size of C_n is less than or equal to n^s. The size of C_n is the total number

of connections between gates, Boolean gates, constant gates and oracle gates.

(3) We let $\Pr[C_n(F^n)]$ be the probability that the output bit of C_n is one when a function is randomly chosen from F^n and used to evaluate the oracle gates. We let $\Pr[C_n(f^n)]$ be the probability that the output bit of C_n is one when a key k of length $l(n)$ is randomly chosen and f_k^n is used to evaluate the oracle gates. The distinguishing probability for C_n, $|\Pr[C_n(F^n)] - \Pr[C_n(f^n)]|$, is greater than or equal to $1/n^c$.

THEOREM [GGM]. *If there is a pseudorandom bit generator then there is a pseudorandom function generator.*

In fact, Goldreich et al. [GGM] give an explicit construction for a function generator f based on a bit generator g and prove that if g is a pseudorandom bit generator then f is a pseudorandom function generator. The converse of this theorem is easily seen to be true.

4.3. Pseudorandom invertible permutation generators and block private key cryptosystems. A **permutation generator** f is a function generator such that each function f_k^n is 1-1 onto. Let $\bar{f} = \{\bar{f}^n : n \in \mathbb{N}\}$, where $\bar{f}^n = \{\bar{f}_k^n : k \in \{0,1\}^{l(n)}\}$, where $\bar{f}_k^n$ is the inverse function of f_k^n. We say f is **invertible** if $\bar{f}$ is also a permutation generator. In this case, $\bar{f}$ is the inverse permutation generator for f (of course, f is the inverse permutation generator for $\bar{f}$ also). We say f is **pseudorandom** if it is pseudorandom as a function generator as defined in the previous section. (The definition of pseudorandom is provably no different if we replace F^n with P^n in part 3 of the definition of an oracle circuit.) In § 5, we give a construction for a pseudorandom invertible permutation generator based on a pseudorandom function generator.

4.4. Super pseudorandom invertible permutation generators. There is even a stronger notion of pseudorandom for invertible permutation generators which is very natural. Let f be an invertible permutation generator. We say that f is a **super pseudorandom** if there is no super distinguishing circuit family for f. A super distinguishing circuit family for f is an infinite family of circuits $C = \{C_{n_1}, C_{n_2}, \cdots\}$ where $n_1 < n_2 \cdots$, where each circuit is an oracle circuit containing two types of oracle gates, **normal** and **inverse**. For some pair of constants s and c, for each n for which there is a circuit C_n:

(1) The size of C_n is less than or equal to n^s.

(2) We let $\Pr[C_n(P^n)]$ be the probability that the output bit of C_n is one when a permutation σ is randomly chosen from P^n and σ and $\bar{\sigma}$ are used to evaluate normal and inverse gates in C_n, respectively, where $\bar{\sigma}$ is the inverse permutation of σ. We let $\Pr[C_n(f^n)]$ be the probability that the output bit of C_n is one when a key k of length $l(n)$ is randomly chosen and f_k^n and $\bar{f}_k^n$ are used to evaluate the normal and inverse oracle gates in C_n, respectively. The distinguishing probability for C_n, $|\Pr[C_n(P^n)] - \Pr[C_n(f^n)]|$, is greater than or equal to $1/n^c$.

We state in § 5 that there is a super pseudorandom invertible permutation generator if there is a pseudorandom function generator. Although every super pseudorandom invertible generator is also a pseudorandom invertible generator, we show in § 5 that the converse is not necessarily true.

Let f be a super pseudorandom invertible permutation generator. We can use f as described above in a block private key cryptosystem. This cryptosystem is secure against chosen plaintext/ciphertext attack, which is an even stronger attack than chosen plaintext attack. In chosen plaintext/ciphertext attack, L can interactively choose plaintext blocks and see their encryptions and choose encryptions and see their corresponding plaintext blocks. Thus, L is allowed to attack the cryptosystem from

"both ends." Let M be a plaintext block which is different than all plaintext blocks seen in the attack. Intuitively, the cryptosystem is secure against chosen plaintext/ciphertext attack if for all agents L, given the information L gained from the attack and given the encryption $f_k^n(M)$ of M (but not M), L cannot predict the value of M any better than if L had no information about M. Symmetrically, let E be the encryption of a plaintext block which is different than all encryptions of plaintext blocks seen in the attack: The cryptosystem is secure against chosen plaintext/ciphertext attack if for all agents L, given the information L gained from the attack and given the decryption $\bar{f}_k^n(E)$ of E (but not E), L cannot predict the value of E any better than if L had no information about E.

4.5. Definition of cryptographic composition. In this section, we formally define the composition of function generators. Let g and h be two function generators, where $l_1(n)$ and $l_2(n)$ are the key length functions for g and h, respectively (we allow the possibility that $g = h$). Then $f = g \circ h$ is a function generator defined as follows.

(1) The key length function for f is $l(n) = l_1(n) + l_2(n)$.

(2) $f_{k_2 \bullet k_1}^n = h_{k_2}^n \circ g_{k_1}^n$, where k_1 is of length $l_1(n)$ and k_2 is of length $l_2(n)$.

Note that if both g and h are (invertible) permutation generators then so is f.

5. Generating permutations from functions. In this section we show how to construct a pseudorandom invertible permutation generator from a pseudorandom function generator (see § 4 for definitions). A natural thing to try is an abstraction of MDES. Let $f = \{f^n\}$ be a pseudorandom function generator where the key length function is $l(n)$. Define a function generator $g = \{g^n\}$ in terms of f as follows. Let k be a string of length $l(n)$, let k' be a string of length $l(n+1)$, let L, R and L' be strings of length n and let R' be a string of length $n+1$. Then

$$g_k^{2n}(L \bullet R) = R \bullet [L \oplus f_k^n(R)],$$

$$g_{k'}^{2n+1}(L' \bullet R') = R' \bullet [L' \oplus \text{first } n \text{ bits of } f_{k'}^{n+1}(R')].$$

Note that g is an invertible permutation generator. The inverse permutation generator for g, $\bar{g}$, is computed as follows. Let α, β and β' be strings of length n and let α' be a string of length $n+1$. Then

$$\bar{g}_k^{2n}(\alpha \bullet \beta) = [\beta \oplus f_k^n(\alpha)] \bullet \alpha,$$

$$\bar{g}_{k'}^{2n+1}(\alpha' \bullet \beta') = [\beta' \oplus \text{first } n \text{ bits of } f_{k'}^{n+1}(\alpha')] \bullet \alpha'.$$

Thus,

$$\bar{g}_k^{2n}(g_k^{2n}(L \bullet R)) = L \bullet R,$$

$$\bar{g}_{k'}^{2n+1}(g_{k'}^{2n+1}(L' \bullet R')) = L' \bullet R'.$$

Let $h = g \circ g \circ g$. Since g is an invertible permutation generator, h is also an invertible permutation generator. Theorem 1 shows that h is pseudorandom if f is pseudorandom. Before proving this theorem, we show that $h = g \circ g$ is not at all pseudorandom, and then we state and prove the main lemma used in the proof of Theorem 1. The proof of the main lemma, which is a combinatorial lemma not based on any unproven assumptions, is the interesting and difficult portion of the proof of the theorem.

We show that $h = g \circ g$ is *not* pseudorandom as follows. Let k_1 and k_2 be keys of length $l(n)$. For all strings L_1, L_2 and R of length n, the $\oplus$ of the first n bits of $h_{k_2 \bullet k_1}^{2n}(L_1 \bullet R)$ and $h_{k_2 \bullet k_1}^{2n}(L_2 \bullet R)$ is equal to $L_1 \oplus L_2$. Thus, a distinguishing circuit C_{2n} for h^{2n} can be described as follows. Choose L_1 and L_2 such that $L_1 \neq L_2$ and let R be

any string of length n, e.g., a string of n zeros. C_{2n} has two oracle gates: the input to the first oracle gate is $L_1 \bullet R$ and the input to the second oracle gate is $L_2 \bullet R$. The output of C_{2n} is 1 iff the $\oplus$ of the first n bits of the outputs from the two oracle gates is equal to $L_1 \oplus L_2$. Thus, the output of C_{2n} is always 1 when the oracle gates are evaluated using a function in h^{2n}. On the other hand, if the oracle gates are evaluated using a function randomly chosen from F^{2n}, the output of C_{2n} is 1 with probability $1/2^n$.

The following definitions are used in the main lemma and in Theorem 1. We define the operator $H: F^n \times F^n \times F^n \to P^{2n}$ as follows. Let f_0, f_1 and f_2 be three functions from F^n. Let L and R be strings of length n. Define

$$\alpha = f_0(R), \quad \beta = f_1(L \oplus \alpha), \quad \gamma = f_2(R \oplus \beta),$$

$$H(f_2, f_1, f_0)(L \bullet R) = [R \oplus \beta] \bullet [L \oplus \alpha \oplus \gamma].$$

H is derived from three applications of the DES design rule described in § 3. The main lemma states that any Boolean circuit with m oracle gates, where each oracle gate has an input and an output which are both strings of length $2n$, can distinguish with probability at most $m^2/2^n$ between the case when the oracle gates are evaluated using a randomly chosen function from F^{2n} and the case when the oracle gates are evaluated using $H(f_2, f_1, f_0)$ and f_2, f_1 and f_0 are randomly chosen from F^n. This result is surprising in the sense that it uses no unproven assumptions and that the number of distinct permutations generated by H is at most 2^{3n2^n}, which is a very small fraction of the $2^{2n2^{2n}}$ functions in F^{2n}. On the other hand, it is not so surprising given that we allow the distinguishing circuit to examine only a very small portion of the domain of the function (there are only m oracle gates).

PROPOSITION. *Let D_n be an oracle circuit with m oracle gates and size s, where the input and output of each oracle gate is a string of length n and where $m \leqq 2^n$. D_n can be easily modified to another circuit D'_n which never repeats an input value to an oracle gate such that*: (1) *For each function $f_0 \in F^n$, the output bit of D'_n when f_0 is used to evaluate the oracle gates is exactly the same as the output bit of D_n when f_0 is used to evaluate the oracle gates*; (2) *The number of oracle gates in D'_n is m*; (3) *The size of D'_n is at most s^3.*

Proof. Omitted but easy. □

MAIN LEMMA. *Let C_{2n} be an oracle circuit with m oracle gates such that no input value is repeated to an oracle gate as in the above proposition. Then,* $|\Pr[C_{2n}(F^{2n})] - \Pr[C_{2n}(H(F^n, F^n, F^n))]| \leqq m^2/2^n$.

Proof. There are really two experiments occurring in different probability spaces in the statement of the Main Lemma. For $\Pr[C_{2n}(F^{2n})]$ the sample space is F^{2n} with the uniform probability distribution, and for $\Pr[C_{2n}(H(F^n, F^n, F^n))]$ the sample space is $F^n \times F^n \times F^n$ with the uniform probability distribution. To be able to analyze the behavior of C_{2n} with respect to these two probability distributions we introduce a new probability space with sample space $\Omega = \{0, 1\}^{3nm}$ and the uniform probability distribution, i.e., for all $\omega \in \Omega$, $\Pr[\omega] = \frac{1}{2}^{3nm}$. We define two random variables A, B on the new probability space with the following properties:

(1) $A: \Omega \to \{0, 1\}$, $B: \Omega \to \{0, 1\}$.

(2) $E[A] = \Pr[C_{2n}(F^{2n})]$. (The left-hand side is with respect to the new probability space, the right-hand side is with respect to the probability space for the original experiment.)

(3) $E[B] = \Pr[C_{2n}(H(F^n, F^n, F^n))]$. (Same remarks as for (2).)

(4) $|E[A] - E[B]| \leqq m^2/2^n$.

Note that $|E[A]-E[B]| = |\sum_{\omega\in\Omega}(A(\omega)-B(\omega))|/2^{3nm}$. Our objective is to define A and B with properties (1), (2) and (3) such that for most sample points A and B take on the same value, so that property (4) is also satisfied. A sample point in Ω is a string $\omega = \omega_1, \cdots, \omega_{3nm}$. For $1 \leqq i \leqq m$ we define the random variables $X_i : \Omega \to \{0,1\}^n$, $Y_i : \Omega \to \{0,1\}^n$, $Z_i : \Omega \to \{0,1\}^n$ as follows:

$$X_i(\omega) = \omega_{(i-1)n+1} \bullet \cdots \bullet \omega_{(i-1)n+n},$$

$$Y_i(\omega) = \omega_{mn+(i-1)n+1} \bullet \cdots \bullet \omega_{mn+(i-1)n+n},$$

$$Z_i(\omega) = \omega_{2mn+(i-1)n+1} \bullet \cdots \bullet \omega_{2mn+(i-1)n+n}.$$

We define the vector $X(\omega) = \langle X_1(\omega), \cdots, X_m(\omega)\rangle$. $Y(\omega)$ and $Z(\omega)$ are defined similarly.

At sample point $\omega \in \Omega$, $B(\omega)$ is defined as the output bit of C_{2n} when the oracle gates of C_{2n} are computed as described below. This description also defines $L(\omega)$ and $R(\omega)$, where L and R are vectors of m random variables corresponding to the left half of the inputs to the oracle gates and the right half of the inputs to the oracle gates, respectively. In addition, $\alpha'(\omega)$, $\beta'(\omega)$ and $\gamma'(\omega)$ are defined, where α', β' and γ' are each vectors of m random variables defined by the following circuit. The ith oracle gate of C_{2n} is computed as follows:

***B*–gate$_i$:**

The input is $L_i(\omega) \bullet R_i(\omega)$,
$l \leftarrow \min\{j : 1 \leqq j \leqq i \text{ and } R_i(\omega) = R_j(\omega)\}$,
$\alpha'_i(\omega) \leftarrow L_i(\omega) \oplus X_l(\omega)$,
$l \leftarrow \min\{j : 1 \leqq j \leqq i \text{ and } \alpha'_i(\omega) = \alpha'_j(\omega)\}$,
$\beta'_i(\omega) \leftarrow R_i(\omega) \oplus Y_l(\omega)$,
$l \leftarrow \min\{j : 1 \leqq j \leqq i \text{ and } \beta'_i(\omega) = \beta'_j(\omega)\}$,
$\gamma'_i(\omega) \leftarrow \alpha'_i(\omega) \oplus Z_l(\omega)$,
The output is $\beta'_i(\omega) \bullet \gamma'_i(\omega)$.

The circuit defining B can be thought of as a circuit with Boolean **and**, **or** and **not** gates and constant **zero** and **one** gates. The input to the entire circuit can be thought of as $X(\omega)$, $Y(\omega)$ and $Z(\omega)$.

We now describe a random variable B' which is exactly the same as B except that it is the output bit of C_{2n} when the ith oracle gate is computed as follows:

***B'*–gate$_i$:**

The input is $L_i(\omega) \bullet R_i(\omega)$,
$l \leftarrow \min\{j : 1 \leqq j \leqq i \text{ and } R_i(\omega) = R_j(\omega)\}$,
$\alpha'_i(\omega) \leftarrow L_i(\omega) \oplus X_l(\omega)$,
$l \leftarrow \min\{j : 1 \leqq j \leqq i \text{ and } \alpha'_i(\omega) = \alpha'_j(\omega)\}$,
$Y'_i(\omega) \leftarrow Y_i(\omega) \oplus R_i(\omega)$,
$\beta'_i(\omega) \leftarrow R_i(\omega) \oplus Y'_l(\omega)$,
$l \leftarrow \min\{j : 1 \leqq j \leqq i \text{ and } \beta'_i(\omega) = \beta'_j(\omega)\}$,
$Z'_i(\omega) \leftarrow Z_i(\omega) \oplus \alpha'_i(\omega)$,
$\gamma'_i(\omega) \leftarrow \alpha'_i(\omega) \oplus Z'_l(\omega)$,
The output is $\beta'_i(\omega) \bullet \gamma'_i(\omega)$.

B and B' are exactly the same except that Y'_i and Z'_i are computed from Y_i and Z_i

and used in place of Y_i and Z_i in the oracle gate computation. The definition of B' also redefines the vectors of random variables L, R, α', β', γ'.

FACT. $E[B'] = E[B]$. This follows because the random variables X_i, Y_i and Z_i are identically and uniformly distributed and because R_i and α'_i do not depend on Y_i and Z_i, respectively. Thus, Y'_i and Z'_i are independently and randomly chosen strings from $\{0,1\}^n$.

We now describe the computation of B' in an equivalent way. This alternative description is used in all further references to B'. B' is the output of C_{2n} when the ith oracle gate in C_{2n} is evaluated as follows:

***B'*–gate$_i$:**

The input is $L_i(\omega) \bullet R_i(\omega)$,
$u_i(\omega) \leftarrow \min\{j : 1 \leqq j \leqq i \text{ and } R_i(\omega) = R_j(\omega)\}$,
$\alpha'_i(\omega) \leftarrow L_i(\omega) \oplus X_{u_i(\omega)}(\omega)$,
$v_i(\omega) \leftarrow \min\{j : 1 \leqq j \leqq i \text{ and } \alpha'_i(\omega) = \alpha'_j(\omega)\}$,
$\beta'_i(\omega) \leftarrow R_i(\omega) \oplus R_{v_i(\omega)}(\omega) \oplus Y_{v_i(\omega)}(\omega)$,
$w_i(\omega) \leftarrow \min\{j : 1 \leqq j \leqq i \text{ and } \beta'_i(\omega) = \beta'_j(\omega)\}$,
$\gamma'_i(\omega) \leftarrow \alpha'_i(\omega) \oplus \alpha'_{w_i(\omega)}(\omega) \oplus Z_{w_i(\omega)}(\omega)$,
The output is $\beta'_i(\omega) \bullet \gamma'_i(\omega)$.

From the above discussion, it is clear that $E[B'] = \Pr[C_{2n}(H(F^n, F^n, F^n))]$. In addition to defining L, R, α', β' and γ' in terms of X, Y and Z, this also defines the vectors u, v and w in terms of X, Y and Z.

Let A be the random variable which is defined to be the output of C_{2n} when the oracle gates are evaluated exactly the same way as in the definition of B' except that the output of the ith oracle gate is $Y_i(\omega) \bullet Z_i(\omega)$. This defines different vectors of random variables L, R, α', β', γ', u, v and w in terms of X, Y and Z then those defined for B'. In the following, the vectors of random variables we are considering are explicitly mentioned as needed. Because A is determined by C_{2n} when the output values from each oracle gate are independently and identically distributed random variables and because C_{2n} never repeats an input value to an oracle gate, $E[A] = \Pr[C_{2n}(F^{2n})]$. Our goal now is to show that $|E[A] - E[B']| \leqq m^2/2^n$.

DEFINITION. $\omega \in \Omega$ is **preserving** if for $1 \leqq i \leqq m$, $v_i(\omega) = i$ and $w_i(\omega) = i$, where v_i and w_i are the random variables defined with respect to A.

CLAIM. For all $\omega \in \Omega$, if ω is preserving then $A(\omega) = B'(\omega)$.

Justification. If for $1 \leqq i \leqq m$, $v_i(\omega) = i$ and $w_i(\omega) = i$, then for $1 \leqq i \leqq m$, $\beta'_i(\omega) = Y_i(\omega)$ and $\gamma'_i(\omega) = Z_i(\omega)$, where all these random variables are those defined with respect to A. Thus, all the outputs of the oracle gates are the same as though they were calculated as in B', and thus all the variables defined with respect to A have exactly the same values as the corresponding variables defined with respect to B'.

For all $\omega \in \Omega$, if ω is preserving then $A(\omega) - B'(\omega) = 0$ and even if ω is not preserving then $|A(\omega) - B'(\omega)| \leqq 1$. Thus, $|E[A] - E[B']| \leqq \Pr[\omega \text{ is not preserving}]$. In the following discussion, all variables are defined with respect to A.

DEFINITION. For all $\omega \in \Omega$, $Y(\omega)$ is **bad** if there is i, j, $1 \leqq j < i \leqq m$, such that $Y_i(\omega) = Y_j(\omega)$. It is not hard to verify that $\Pr[Y \text{ is bad}] \leqq m^2/2^{n+1}$.

DEFINITION. For all $\omega \in \Omega$, $X(\omega)$ is **bad** if there is i, j, $1 \leqq j < i \leqq m$, such that $\alpha'_i(\omega) = \alpha'_j(\omega)$.

CLAIM. $\Pr[X \text{ is bad}] \leqq m^2/2^{n+1}$.

Proof. For $1 \leqq i \leqq m$, let $\tilde{y}_i$ and $\tilde{z}_i$ be strings of length n, let $\tilde{y} = \langle \tilde{y}_1, \cdots, \tilde{y}_m \rangle$ and let $\tilde{z} = \langle \tilde{z}_1, \cdots, \tilde{z}_m \rangle$. Let $\Omega_{\tilde{y},\tilde{z}} = \{\omega \in \Omega : Y(\omega) = \tilde{y} \text{ and } Z(\omega) = \tilde{z}\}$. We show that for each

$\tilde{y}$, $\tilde{z}$, $\Pr[X(\omega) \text{ is bad} | \Omega_{\tilde{y},\tilde{z}}] \leq m^2/2^{n+1}$ which implies that $\Pr[X \text{ is bad}] \leq m^2/2^{n+1}$. Y, Z, L, R and u are constants $\tilde{y}$, $\tilde{z}$, $\tilde{l}$, $\tilde{r}$ and $\tilde{u}$ on $\Omega_{\tilde{y},\tilde{z}}$, respectively. This is by definition for Y and Z, and it is true for L and R because the outputs from all oracle gates are constant on $\Omega_{\tilde{y},\tilde{z}}$, which implies that the inputs to all oracle gates are also constant. u is constant because it depends only on R. On the other hand, X is a vector of m independently and identically distributed random variables on the probability space restricted to $\Omega_{\tilde{y},\tilde{z}}$, and each such variable has uniform distribution on $\{0,1\}^n$. Let *eqn* (i,j) be the equation $\alpha'_j(\omega) = \alpha'_i(\omega)$. On $\Omega_{\tilde{y},\tilde{z}}$, this equation reduces to

$$\tilde{l}_j \oplus X_{\tilde{u}_j}(\omega) = \tilde{l}_i \oplus X_{\tilde{u}_i}(\omega).$$

When $\tilde{u}_j = \tilde{u}_i$, $\Pr[$*eqn* (i,j) is satisfied $|\Omega_{\tilde{y},\tilde{z}}] = 0$ because $\tilde{u}_j = \tilde{u}_i$ implies that $\tilde{r}_j = \tilde{r}_i$ which implies that $\tilde{l}_j \neq \tilde{l}_i$ (because C_{2n} does not repeat an input value to an oracle gate) which implies that *eqn* (i,j) is not satisfied (because *eqn* (i,j) simplifies to $\tilde{l}_j = \tilde{l}_i$ in this case). When $\tilde{u}_j \neq \tilde{u}_i$, $\Pr[$*eqn* (i,j) is satisfied $|\Omega_{\tilde{y},\tilde{z}}] = 1/2^n$ because $\tilde{u}_j \neq \tilde{u}_i$ together with the fact that X is a vector of independent random variables on $\Omega_{\tilde{y},\tilde{z}}$ implies that for fixed $X_{\tilde{u}_j}$, the probability that a randomly chosen $X_{\tilde{u}_i}$ satisfies the equation is $1/2^n$. Thus, summing these probabilities over all $m(m-1)/2$ equations yields $\Pr[X(\omega)$ is bad $|\Omega_{\tilde{y},\tilde{z}}] \leq m^2/2^{n+1}$.

CLAIM. For all $\omega \in \Omega$, $X(\omega)$ is not bad and $Y(\omega)$ is not bad implies that ω is preserving.

Proof. $X(\omega)$ is not bad implies that for $1 \leq j < i \leq m$, $\alpha'_j(\omega) \neq \alpha'_i(\omega)$ which implies that for $1 \leq i \leq m$, $v_i(\omega) = i$ which implies that for $1 \leq i \leq m$, $\beta'_i(\omega) = Y_i(\omega)$. $Y(\omega)$ is not bad and for $1 \leq i \leq m$, $\beta'_i(\omega) = Y_i(\omega)$ implies that for $1 \leq j < i \leq m$, $\beta'_j(\omega) \neq \beta'_i(\omega)$ which implies that for $1 \leq i \leq m$, $w_i(\omega) = i$.

CLAIM. $\Pr[\omega \text{ is not preserving}] \leq \Pr[X \text{ is bad or } Y \text{ is bad}] \leq m^2/2^n$. Thus, $|E[A] - E[B']| \leq m^2/2^n$ and the Main Lemma follows. □

THEOREM 1. *$h = g \circ g \circ g$ is a pseudorandom invertible permutation generator.*

Proof. What must be shown is that h is pseudorandom, i.e., there is no distinguishing circuit family for h. The proof is by contradiction. Assume that there is a distinguishing circuit family $C = \{C_{2n_1}, C_{2n_2}, \cdots\}$ where $n_1 < n_2 \cdots$, for h. We show this implies there is a distinguishing circuit family $D = \{D_{n_i} : i \in \mathbb{N}\}$ for f, contradicting the fact that f is a pseudorandom function generator. In particular, for a fixed n we show that if C_{2n} distinguishes h^{2n} from F^{2n} with probability at least $1/n^c$ then there is a circuit D_n (where D_n is not much bigger than C_{2n}) which distinguishes f^n from F^n with probability at least $1/4n^c$. (The proof for C_{2n+1} is similar). Let us first state the definition of h^{2n} in a different way. If k_0, k_1 and k_2 are strings of length $l(n)$ then

$$h^{2n}_{k_2 \bullet k_1 \bullet k_0} = H(f^n_{k_2}, f^n_{k_1}, f^n_{k_0}).$$

Let p_0^H, p_1^H, p_2^H, p_3^H be $\Pr[C_{2n}(H(f^n, f^n, f^n))]$, $\Pr[C_{2n}(H(f^n, f^n, F^n))]$, $\Pr[C_{2n}(H(f^n, F^n, F^n))]$, $\Pr[C_{2n}(H(F^n, F^n, F^n))]$, respectively. Let $p^R = \Pr[C_{2n}(F^{2n})]$. The main lemma shows that $|p^R - p_3^H| \leq m^2/2^n$, where m is the number of oracle gates in C_{2n}. Since m is less than or equal to n^s for some constant s, $m^2/2^n \leq 1/4n^c$ for sufficiently large n. Since

$$1/n^c \leq |p^R - p_0^H| \leq |p^R - p_3^H| + |p_3^H - p_2^H| + |p_2^H - p_1^H| + |p_1^H - p_0^H|,$$

there is an $i \in \{0,1,2\}$ such that $|p_{i+1}^H - p_i^H| \geq 1/4n^c$. The cases when $i = 0$ and $i = 2$ are similar to the case when $i = 1$ and are omitted. For the case $i = 1$, we use C_{2n} to

construct a probabilistic circuit D_n not much larger than C_{2n} which will distinguish f^n from F^n with probability at least $1/4n^c$. D_n is the same as C_{2n} except that it first randomly chooses $k_2 \in \{0,1\}^{l(n)}$ and $f_0 \in F^n$, and then each oracle gate X in C_{2n} is replaced with a subcircuit which computes $H(f^n_{k_2}, X', f_0)$, where X' is an oracle gate with an input and output which are both strings of length n. The main problem with constructing D_n is how to randomly choose f_0 and F^n. In fact, D_n does not specify f_0 at all (since a complete description would involve $n2^n$ bits). Instead, D_n remembers all inputs and outputs to f_0. When D_n is to compute $f_0(\alpha)$ for some string α of length n, if α is the same as a previous input then the same output as before is given, otherwise α is a new input and then $f_0(\alpha)$ is a randomly chosen string from $\{0,1\}^n$.

When the oracle gates in D_n are evaluated using $f^n_{k_1}$, where k_1 is a randomly chosen key of length $l(n)$, then $p_1^H = \Pr[\text{The output of } D_n \text{ is } 1]$. When the oracle gates in D_n are evaluated using f_1, where f_1 is randomly chosen from F^n, then $p_2^H = \Pr[\text{The output of } D_n \text{ is } 1]$. Since $|p_2^H - p_1^H| \geqq 1/4n^c$, D_n distinguishes F^n from f^n with probability greater than or equal to $1/4n^c$. □

THEOREM 2. *Let g be defined in terms of pseudorandom function generator f as described in the beginning of this section. Then $h = g \circ g \circ g \circ g$ is a super pseudorandom invertible permutation generator.*

Proof. The proof of this theorem is very similar to the proof of Theorem 1 and is omitted. □

It is interesting to note that $h = g \circ g \circ g$ is *not* super pseudorandom, although Theorem 1 shows that it is pseudorandom. This can be seen as follows. Let k_1, k_2 and k_3 be keys of length $l(n)$. A distinguishing circuit C_{2n} for h^{2n} can be described as follows. Let L_1, L_2 and R be strings of length n such that $L_1 \neq L_2$. C_{2n} has two normal oracle gates: the input to the first normal oracle gate is $L_1 \bullet R$ and the input to the second normal oracle gate is $L_2 \bullet R$. Let $S_1 \bullet T_1$ and $S_2 \bullet T_2$ be the outputs of these two normal oracle gates respectively. C_{2n} also has an inverse oracle gate with input $S_2 \bullet [T_2 \oplus L_1 \oplus L_2]$. The output of C_{2n} is 1 if and only if the last n bits of the output from this inverse oracle gate is equal to $R \oplus S_1 \oplus S_2$. It can be verified that the output of C_{2n} is always 1 when the normal oracle gates and inverse oracle gates are computed by first choosing a key k at random and then using h^{2n}_k for the normal oracle gates and $\bar{h}^{2n}_k$ to compute the inverse oracle gates. On the other hand, if the oracle gates are computed using a permutation randomly chosen from P^{2n}, the output of C_{2n} is 1 with probability $1/2^n$.

6. Conclusions. Assuming the existence of a pseudorandom function generator, we have proven the existence of a super pseudorandom invertible permutation generator. This is of interest by itself, but the construction and the proof can be viewed as a partial justification of some of the methodology used in the design of DES.

Another methodology which is important both in DES and in the cryptographic literature is using cryptographic composition of permutation generators to **increase** security. In [LR] we define a **measure** of security for permutation generators, and prove that if one composes two permutation generators which are slightly secure the result is more secure than either one alone.

Acknowledgments. We thank Oded Goldreich for suggesting the notion of super pseudorandom and for finding a problem with an initial construction we had for a pseudorandom invertible permutation generator. We thank Johan Hastad for simplifying the proof of the Main Lemma. We thank Paul Beame for numerous suggestions which kept us on the right track.

REFERENCES

[ACGS] W. ALEXI, B. CHOR, O. GOLDREICH AND C. P. SCHNORR, RSA *and Rabin functions: Certain parts are as hard as the whole*, SIAM J. Comput., 17 (1988), pp. 194–209; preliminary version in Proceedings of the 25th IEEE Symposium on Foundations of Computer Science, New York, 1984, pp. 449–457.

[Ba] J. BAMFORD, *The Puzzle Palace: A Report on America's Most Secret Agency*, Penguin, 1983.

[BCS] M. BEN-OR, B. CHOR AND A. SHAMIR, *On the cryptographic security of single* RSA *bits*, in Proceedings of the 15th ACM Symposium on Theory of Computing, Boston, 1983, pp. 421–430.

[BM] M. BLUM AND S. MICALI, *How to generate cryptographically strong sequences of pseudo-random bits*, in Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, November 3–5, 1982, pp. 112–117; preliminary version in SIAM J. Comput., 13 (1984), 850–886.

[De] D. DENNING, *Cryptography and Data Security*, Addison-Wesley, Reading, MA, January 1983.

[GGM] O. GOLDREICH, S. GOLDWASSER AND S. MICALI, *How to construct random functions*, in Proceedings of the 25th Annual Symposium on Foundations of Computer Science, October 24–26, 1984.

[GMT] S. GOLDWASSER, S. MICALI AND P. TONG, *Why and how to establish a private code on a public network*, in Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, New York, 1982, pp. 134–144.

[H] A. HODGES, *Alan Turing: The Enigma of Intelligence*, Unwin Paperbacks, 1985.

[Le] L. A. LEVIN, *One-way functions and pseudorandom generators*, in Proceedings of the 17th ACM Symposium on Theory of Computing, Providence, RI, 1985, pp. 363–365.

[LR] M. LUBY AND C. RACKOFF, *Pseudorandom permutation generator and cryptographic composition*, Proc. 18th Annual Symposium on Theory of Computing, May 28–30, 1986.

[Ra] C. RACKOFF, *Class Notes on Cryptography*, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, 1985.

[VV] U. V. VAZIRANI AND V. V. VAZIRANI, *Efficient and secure pseudo-random number generation*, in Proceedings of the 25th IEEE Symposium on Foundations of Computer Science, 1984, pp. 458–463.

[Yao] A. C. YAO, *Theory and applications of trapdoor functions*, 23rd Annual Symposium on Foundations of Computer Science, November 3–5, 1982, pp. 80–91.