J. Cryptology (1996) 9: 199–216

Journal of
CRYPTOLOGY
© 1996 International Association for Cryptologic Research

Efficient Cryptographic Schemes Provably as Secure as Subset Sum*

Russell Impagliazzo
Department of Computer Science, University of California at San Diego,
La Jolla, CA 92093, U.S.A.
russell@cs.ucsd.edu

Moni Naor**
Department of Applied Mathematics and Computer Science,
Weizmann Institute, Rehovot 76100, Israel
naor@wisdom.weizmann.ac.il

Communicated by Steven Rudich

Received 27 July 1993 and revised 1 June 1995

Abstract. We show very efficient constructions for a pseudorandom generator and for a universal one-way hash function based on the intractability of the subset-sum problem for certain dimensions. (Pseudorandom generators can be used for private-key encryption and universal one-way hash functions for signature schemes.) The increase in efficiency in our construction is due to the fact that many bits can be generated/hashed with one application of the assumed one-way function.

All of our constructions can be implemented in NC using an optimal number of processors.

Key words. Pseudorandom generators, Knapsack, Digital signatures, Bit commitment, Parallelism.

1. Introduction

Many cryptosystems are based on the intractability of number-theoretic problems such as factoring and discrete logarithm. Both of these problems have a long history of attempted solutions as well as many convenient features. However, it is desirable to consider protocols based on other types of problems. First, the protocols that are based on factoring and discrete log tend to be rather inefficient. This is one reason why these protocols are

*Part of this work was done while both authors were at the University of California at Berkeley and part when the second author was at the IBM Almaden Research Center. Research supported by NSF Grant CCR 88-13632. A preliminary version of this paper appeared in *Proc. 30th Symposium on Foundations of Computer Science*, 1989.

**Incumbent of the Morris and Rose Goldman Career Development Chair. Research supported by an Alon Fellowship and a grant from the Israel Science Foundation administered by the Israeli Academy of Sciences.

not extensively used in practice, although some implementations of the simplest of these protocols have been made. Secondly, there are no (provably secure) parallelizations of these protocols. Finally, although considered unlikely, it is possible that feasible algorithms for these number-theoretic problems exist. In this case, it is good to have other protocols to fall back on.

One alternative that has been considered since the early days of public-key cryptography is the subset-sum problem (a.k.a. the knapsack problem). The subset-sum problem is: given n numbers, each l bits long, and a target sum T, find a subset of the numbers whose sum is T. Many schemes were suggested that use this problem as the basis for public-key encryption. However, none of these schemes have been proven to be as secure as subset sum, and, in fact, most of them have been broken. See [9] and [41] for a survey. The first to suggest using subset sum were Merkle and Hellman [36], and the only method for using subset sum in a public-key protocol that has not been broken is Chor and Rivest's [11].

The approach taken here is different in two ways. We are less ambitious, and are not attempting to construct a public-key cryptosystem. Many important tasks in cryptography do not require the full power of public-key cryptography. These tasks include: private-key encryption, pseudorandom generation, zero knowledge protocols, identification schemes, and digital signatures. These tasks are known to be implementable based on any one-way function [24], [25], [38], [40], [22], [42]. However, the theoretical constructions suggested are extremely inefficient. We give very efficient constructions for primitives such as **pseudorandom generators** and **universal one-way hash** functions (defined in [40]) which can be used to implement the above tasks. To break our protocols is provably as hard as solving the subset-sum problem for certain dimensions.

Our constructions are extremely simple: at each step, it is only necessary to add n numbers of length $O(n)$. Thus, each step can clearly be implemented in any "reasonable" model of parallel computation with optimal speed-up. Since the steps can also be implemented in parallel, we have optimal parallel implementation of the primitives. This is the first method that can be implemented in NC which is provably secure[1] for any of these tasks (except identification in which the on-line part can be computed in NC using the Fiat–Shamir [13] method). In fact, we show that, in the case of pseudorandom generators, expanding the seed by a small amount can even be done in AC^0.

A major source of inefficiency in the general constructions mentioned above is the fact that the one-way function on which they are based must be evaluated at least as many times as there are bits in the "product" (i.e., the number of pseudorandom bits produced in pseudorandom generators or the number of bits hashed in one-way hashing). We give the first provably secure constructions where the number of times the one-way function is applied is substantially smaller than the number of bits processed.

The next subsection defines precisely the subset-sum problem and the assumptions we make concerning its intractability. Section 1.2 surveys the known results about the complexity of solving subset-sum problems.

In Section 2 we prove that for dimensions n, ℓ with $\ell > n$ for which the subset-sum

[1] Here, "provably secure" means that breaking the primitive in question is provably as hard as inverting a well-studied function which is widely assumed to be one-way. Some parallel constructions have been given without this property, e.g., [37].

problem is hard it is also pseudorandom. In Section 3 we show that for dimensions n, ℓ with $\ell < n$ for which the subset sum problem is hard it is also a **universal one-way hash function**.

In Section 4 we consider a generalization of our results about universal one-way hash functions to other groups (other than addition modulo powers of 2). We show how to relate the security of such functions to conventional number-theoretic problems. We show for instance that the subset product mod N function is as secure a universal one-way hash function as factoring N. Our results for pseudorandom generators also generalize, but this is not as interesting in that we are not able to show any connections to widely accepted number-theoretic conjectures. Hash functions constructed using this method will not be particularly efficient, but will be implementable in parallel.

A primitive for which there is no known construction under a general assumption (e.g., existence of one-way or trapdoor permutations) is bit commitment versus a computationally unlimited receiver. This primitive constructed under number-theoretic assumptions was used by Brassard *et al.* [7] for minimum disclosure proofs. In Section 5 we show how to construct this primitive based on subset sum. Recently, Naor *et al.* [39] have shown how to construct such a bit commitment from any one-way permutation; however, their construction requires many rounds of interaction.

Finally, in Section 6 we show that the assumption that the subset-sum problem is hard yields a pseudorandom generator which can be implemented by a polynomial-size constant-depth circuit, i.e., in AC^0.

1.1. *The Subset-Sum Problem*

The subset-sum problem of dimensions n and ℓ is: given n numbers, $\vec{a} = (a_1, a_2, \ldots, a_n)$, each l bits long, and a number T, find a subset $S \subset \{1, \ldots, n\}$ such that $\sum_{i\in S} a_i = T \bmod 2^{\ell}$. We consider families of subset problems where each family is determined by a function $\ell(n)$ and for size n the length of the n numbers is $\ell(n)$. We can view this problem as that of inverting the following function:

$$f(\vec{a}, S) = \vec{a} \sum_{i\in S} a_i \bmod 2^{l(n)},$$

i.e., the function which concatenates $\vec{a}$ with the sum of the a_i's for $i \in S$. We usually think of $a_1, a_2, \ldots, a_n$ as a fixed parameter (like the modulus in RSA), and view f as mapping an n bit string S to an $l(n)$ bit string. In this case we write $f_{\vec{a}}(S)$ for $f(\vec{a}, S)$. Also, we often identify the subset $S \subset \{1, \ldots, n\}$ with its incidence vector $s \in \{0, 1\}^n$ and write $f_{\vec{a}}(s)$.

We now formally define what we mean by hard:

Definition 1.1. Let $\{f_n\}$ be a sequence of functions indexed by n such that $f_n\colon D_n \mapsto R_n$. We say that $\{f_n\}$ is **one-way** if:

- $f_n(x)$ is polynomial-time computable.
- There is no polynomial-time algorithm I that can invert f on a random input: for every $c > 0$ and for every probabilistic polynomial-time algorithm I (that attempts to invert f), the probability that $f(I(f(x))) = f(x)$ is at most n^{-c} for an input $x \in D_n$ chosen uniformly at random, for all but finitely many n.

We say that the subset-sum problem for length $l(n)$ is **hard** if the corresponding sequence of functions f is one-way.

The character of the subset-sum problem is quite different when $l(n) > n$ and when $l(n) < n$. In the first case the function is almost 1–1 and expands the length of the input. In the second case the function is many to one, almost onto, and contracts the input. The applications we give for these two cases are consequently quite different. We use the first case for pseudorandom generation, and the second case for one-way hashing. We can quantify "almost onto" and "almost 1–1" as follows:

The following definition of Santha and Vazirani [43] describes distributions that are close enough to uniform for our purposes.

Definition 1.2 [43]. Let D be a probability distribution on $\{0, 1\}^m$. We say D is **quasi-random** within ε, if $\forall X \subset \{0, 1\}^m$ we have that $|\mathrm{Pr}_D[X] - |X|/2^m| < \varepsilon$, where $\mathrm{Pr}_D[X]$ is the probability that an element chosen according to D is in X.

Proposition 1.1.

1. *Let $l(n) = cn$ for $c > 1$. Let $\vec{a} = (a_1, a_2, \ldots, a_n)$ and S both be chosen uniformly at random. Except with probability exponentially small, there is no $S' \neq S$ such that $f_{\vec{a}}(S) = f_{\vec{a}}(S')$.*
2. *Let $l(n) \leq cn$ for $c < 1$. For all but an exponentially small fraction of $\vec{a} = (a_1, a_2, \ldots, a_n)$, the distribution given by $f_{\vec{a}}(S)$ for a randomly chosen S is quasi-random within an exponentially small amount.*

Proof. The main idea for obtaining both parts of the proposition is to view the different $f_{\vec{a}}$'s, ranging over all choices of $\vec{a}$, as a pairwise independent family of hash functions from $\{0, 1\}^n$ to $\{0, 1\}^{l(n)}$. Let $S, S' \in \{0, 1\}^n$ be such that $S \neq S'$ and $S, S' \neq 0^n$, and consider $\vec{a}$ being selected uniformly at random. Now, if $S_i \neq 0$, then $f_{\vec{a}}(S) = a_i + \sum_{j \neq i} S_j a_j$ is uniformly distributed, since a_i is independent of the other a_j. Similarly for S'. Also, assume $S_i = 1$ and $S'_i = 0$. Then fixing all the a_j for $j \neq i$ determines $f_{\vec{a}}(S')$ but leaves $f_{\vec{a}}(S)$ completely undetermined. Hence, $f_{\vec{a}}(S')$ and $f_{\vec{a}}(S)$ are independent uniformly distributed variables, for every such pair S and S'.

Hence, if $l(n) \geq cn, c > 1$, we have

$$\mathrm{Prob}[\exists S' \neq S, f_{\vec{a}}(S) = f_{\vec{a}}(S')] \leq \sum_{S' \neq S} \mathrm{Prob}[f_{\vec{a}}(S') = f_{\vec{a}}(S)] \leq 2^n 2^{-l(n)} \leq 2^{-(c-1)n}.$$

If $l(n) \leq cn, c < 1$, we can apply the leftover hash lemma of [25] and [26] to see that the expected distinguishability of $f_{\vec{a}}(S)$ and a random $y \in \{0, 1\}^{l(n)}$ is at most $2^{-(n-l(n))/2} \leq 2^{-((1-c)/2)n}$. We then apply Markov's inequality to get the result claimed in part 2 of the proposition. □

From the above, it is not difficult to see that the most-secure choice of parameters for subset sum is when $l(n) = n$. If the subset S is, with high probability, uniquely determined by the sum for $l(n)$ bits and an inversion algorithm exists for this length, then to invert the function given some $l'(n) > l(n)$ bits, all but the least-significant $l(n)$ bits of $\vec{a}$ and T can simply be ignored. On the other hand, if, for $l(n)$ bits, the sum is almost uniformly distributed, then to invert the function for $l'(n) < l(n)$ bits, simply append $l(n) - l'(n)$ random bits to each of the a_i's and to T, and use whatever method inverts the function for $l(n)$ bit strings. This is summarized in the following proposition:

Proposition 1.2.

1. *Let $n \le l(n) \le l'(n)$. If subset sum is hard for $l'(n)$, then it is also hard for $l(n)$.*
2. *Let $n \ge l(n) \ge l'(n)$. If subset sum is hard for $l'(n)$, then it is also hard for $l(n)$.*
3. *If subset sum is hard for $l(n)$, and $c > 0$, then subset sum is hard for $l(n) + c \log n$ and $l(n) - c \log n$.*

In the applications to come we assume that subset sum is hard for $l(n)$ of the form $c \cdot n$ for some constant c. When $l(n)$ is sufficiently larger than n the function f is almost 1–1; when $l(n)$ is sufficiently smaller than n the function f is almost onto, and almost all outputs occur roughly the same number of times.

Efficiency of implementation. How efficient is it to use this one-way function $f_{\vec{a}}$? A first impression is that large key length might make use of this function impractical. As detailed in the next section, to withstand current attacks we must have n and $l(n)$ on the order of 100, so to describe $\vec{a}$ will require on the order of 10,000 bits. However, this is actually misleading in that for all applications described here, $\vec{a}$ can be chosen once and for all by the protocol designer and publicly announced. Thus, individual keys will be of length $O(n + l(n))$ instead of $O(nl(n))$.

All that is required to compute $f_{\vec{a}}$ is the normal addition of n numbers of length $O(n)$, which requires approximately the same time as a single naive multiplication. Since there are very efficient and highly parallelizable implementations of iterated addition, it is highly suitable for either hardware or software implementations. As mentioned before, this function is in the class NC^1 and all of the constructions in this paper can be implemented in NC.

1.2. *The Complexity of Subset Sum*

Subset sum is one of the original problems that Karp [28] proved to be NP-hard, (i.e., the corresponding decision problem is NP-complete). However, although this probably means that no feasible worst-case algorithm exists for this problem, it says little about the hardness of a random instance. Many NP-complete problems are known to have polynomial average-case algorithms. The subset problem under the assumption that the inputs are chosen uniformly at random has been investigated in a number of papers [8], [12], [14], [15], [27], [32]. For the case $l(n) > n^2$, Lagarias and Odlyzko [32] and Brickell [8] have shown a feasible algorithm which solves this problem for almost all instances of these dimensions.

The Lagarias–Odlyzko [32] and Brickell [8] algorithms mentioned above transform the subset-sum problem into a shortest vector in a lattice problem. A shortest vector in a lattice problem is: given vectors $V_1, V_2, \ldots, V_n \in \mathcal{R}^m$ find the shortest vector (Euclidean norm) V in $\{\sum_{i=1}^{n} z_i V_i | z_i \in \mathcal{Z}\}$. The n numbers $a_1, a_2, \ldots, a_n$ and the target sum T determine the lattice and every solution to the subset-sum problem corresponds to a vector in the lattice.

What they showed is that if $\ell(n) > 1.5472 \cdot n$, then with high probability (over the choices of the subset problem of these dimensions) the vector corresponding to the solution to the subset sum problem is the shortest in the lattice. This was improved very recently by Coster *et al.* [12] and Joux and Stern [27] who showed a different transformation that has the property that the vector corresponding to the solution is the shortest whenever $\ell(n) > 1.0629 \cdot n$. (Reference [12] also contains some evidence showing the limitation of this method.)

The above-mentioned papers suggest as a second stage finding these shortest vectors using the lattice base reduction algorithm of Lenstra *et al.* [33] (or some modification of it, see [44]). This algorithm is not guaranteed to find a shortest vector but one that is at most 2^n times the shortest. In order to get that with high probability the shortest vector is much shorter than the other vectors, one has to require $\ell(n) > n^2$. However, the hardness of the shortest-vector problem is uncertain: the problem is not known to be NP-hard and it is conceivable that a polynomial-time algorithm exists. To our knowledge, there is no generally believed conjecture that it is hard.

Thus, to be on the safe side we must assume that there is an efficient algorithm for the shortest-vector problem and chose the dimension $\ell(n)$ to be smaller than $1.0629 \cdot n$.

As for the other case, when $l(n)$ is very small, say $O(\log n)$, then the subset-sum problem can be solved via dynamic programming. (For very dense problems, $\ell(n) < \frac{1}{2}\log n$, there are more efficient algorithms, as was recently shown by Galil and Margalit [17].) For the length function $\ell(n) = n$, the best known attack takes time $O(2^{n/2})$ and space $O(2^{n/4})$ due to Schroeppel and Shamir [46].

The recent papers [44] and [45] report extensive experimental work on the subset-sum problem and several variants. They manage to solve problems of size up to $n = 74$ using a moderate amount of computation.

From the discussion above it is clear that the subset sum of the dimensions we require is receiving considerable attention. However, we think that more cryptanalytic effort should be devoted to it before we can conclude that it is safe to assume that it hard for some specific ℓ and n and use it in any actual implementation.

2. An Efficient Pseudorandom Generator

A pseudorandom generator is a way to obtain many random looking bits from a short truly random seed. Formally, a function $G: \{0, 1\}^n \mapsto \{0, 1\}^{l(n)}$ is a pseudorandom generator if every algorithm A that tries to distinguish between outputs of G and truly random sequences has a negligible probability of success (i.e., for each $d > 0$ except for finitely many n's, $|\Pr[A(G(x)) = 1] - \Pr[A(y) = 1]| < n^{-c}$ where $x \in \{0, 1\}^n$ and $y \in \{0, 1\}^{l(n)}$ are uniformly chosen.)

Pseudorandom generators have many applications in cryptography. These include: private-key encryption [21], [18], [35], bit commitment [38] (the strong committer variant, which allows zero knowledge proofs [22]), and succinct secret sharing [31].

Cryptographically strong pseudorandom generators were defined by Blum and Micali [5], who constructed a pseudorandom generator based on discrete-log. Blum *et al.* [4] constructed a pseudorandom generator based on quadratic residuosity. Yao [47] showed that the definition given above is equivalent to that of Blum and Micali, and gave a general construction that can be based on any one-way permutation. An essential part in these and other constructions is a hard bit for a one-way function f, i.e., a function $b\colon \{0,1\}^n \mapsto \{0,1\}$ such that given $f(x)$ no polynomial-time machine can guess $b(x)$ with probability greater than $1/2 + 1/\text{poly}(n)$. Goldreich and Levin [20] have shown that for any one-way function f the inner product with a random vector is a hard bit. More precisely, if $x, r \in \{0,1\}^n$ let $r \cdot x$ be the inner product (mod 2) of x and r, i.e., $r \cdot x = 1$ if the number of bit positions i such that $r_i = x_i = 1$ is odd, where r_i and x_i are the ith bits of r and x, respectively. Then the Goldreich–Levin theorem is:

Theorem 2.1 [20]. *Let f be a one-way function. For any polynomial-time algorithm A and any polynomial p, for all but finitely many n's,* $\Pr[A(f(x), r) = r\cdot x] < 1/2 + 1/p(n)$ *where the probability is taken over uniformly chosen $x, r \in \{0,1\}^n$.*

We apply this theorem in a novel way. Functions that hide bits were used to generate pseudorandom sequences as follows: on a seed x, output $b(x)$ and apply f to x; repeat this process with $f(x)$ playing the role of x. This yields only one bit per application of the one-way function and is sequential. Some improvements have been suggested, like extending the one hidden bit to $O(\log n)$ bits, but this is only a slight improvement in performance. Our construction obtains $O(n)$ bits per application of the one-way function (which is the subset-sum function). As was pointed out by Micali and Schnorr [37], the technique of [18] can be used to generate pseudorandom sequences of any polynomial length in parallel, once a way of generating in parallel $n + O(n)$ bits from n bits is given. Since subset sum is amenable to parallelism, this yields a parallel method for generating pseudorandom sequences.

Suppose that the subset-sum problem with $l(n) = (1+c)n$ is hard, for $c > 0$. We show that it is also a pseudorandom generator. This is in contrast with previous results where the one-wayness of a function was used to construct a different pseudorandom generator. (For instance, using the first part of Proposition 1.1, that the subset-sum function is 1–1 with high probability for $l(n) = (1+c)n$, and using the method of Goldreich *et al.* [19] a pseudorandom generator can be constructed. This generator will output one bit for each computation of the subset sum.) In our case the one-way function and the pseudorandom generator are the same. The loss of efficiency which occurs in the conversion process in previous constructions is no problem in ours.

Theorem 2.2. *Let $l(n) = (1+c)n$ for $c > 0$. If the subset-sum function for length $l(n)$ is one-way, then it is also a pseudorandom generator.*

Proof. Although this theorem holds for subset sum as defined above, for ease of exposition we prove it first for subset sum where the addition is performed modulo a prime p and all numbers are chosen at random from $\{0, \ldots, p-1\}$ and later explain how to obtain the case where the addition is modulo a composite, in particular 2^{ℓ}.

Assume that f is not pseudorandom. We will show that we can use the distinguisher to predict, given r and $f_{\bar{a}}(s)$, the inner product of r and s with probability at least $1/2 + 1/\text{poly}(n)$. From the Goldreich–Levin theorem (Theorem 2.1) this contradicts the assumed one-wayness of f. In this particular case the inner product bit has a special interpretation. We think of s and r as describing subsets of the a_i's. The inner product of s and r is 1 if the intersection of these subsets is odd. Our strategy is to use the distinguisher to predict the size of this intersection, and thus to predict its parity.

The distinguisher gets as input $a_1, a_2, \ldots, a_n \in \{0, p-1\}$ and a supposed sum of a subset $T \in \{0, p-1\}$. Without loss of generality, we assume that the distinguisher outputs 1 with probability at least $1/2 + 1/p(n)$ when T is indeed the sum of a random subset and that it outputs 1 with probability almost exactly $1/2$ when T is chosen uniformly at random from $\{0, p-1\}$. On input $f_{\bar{a}}(s), r$ our predictor does the following:

1. Chooses a random $k \in \{0, \ldots, n\}$ (this is our guess as to the size of the intersection).
2. Chooses a random $x \in \{0, \ldots, p-1\}$.
3. Lets $b_i = a_i + x$ if $r_i = 1$, and $b_i = a_i$ otherwise.
4. Feeds the distinguisher with the b_i's and $f_{\bar{a}}(s) + kx (\bmod p)$.
5. If the distinguisher outputs 1, then outputs the parity of k, otherwise outputs the negation of the parity.

We show below that the above algorithm predicts the inner product with probability at least $1/2 + 1/np(n)$. Suppose that at the first step we have guessed k correctly, i.e., k is the size of the intersection of r and s. In this case our prediction for the inner product is correct if the distinguisher outputs 1. Claim 2.1 says that this happens with probability at least $1/2 + 1/p(n)$. In the other cases Claim 2.2 says that the input to the distinguisher is totally random and hence the probability that our algorithm predicts the inner product is 1/2. Since the first case happens with probability $1/n$, altogether the probability of correctly predicting the inner product is at least $1/2 + 1/np(n)$. We now prove these two claims.

Claim 2.1. *The conditional probability that our algorithm predicts the inner product correctly, given that k is the size of the intersection of r and s, is at least* $1/2 + 1/p(n)$.

Proof. The distribution the distinguisher sees in this case can be generated as follows:

- Pick a random $s, r \in \{0, 1\}^n$ and let k be the size of their intersection.
- Pick random $a_1, a_2, \ldots, a_n \in \{0, \ldots, p-1\}$ and let $T = \sum_{i \in s} a_i$.
- Pick x randomly and let $b_i = a_i + x$ if $r_i = 1$ and $b_i = a_i$ otherwise. Let $T' = T + kx$.
- Output $b_1, b_2, \ldots, b_n, T'$.

We claim that this distribution is exactly that of $a_1, a_2, \ldots, a_n, T$. This is true since each b_i is independent and uniformly distributed and since

$$\sum_{i \in s} b_i = \sum_{i \in s} a_i + \sum_{i \in r \cap s} x = T + kx = T'.$$

Both the b_i's and the a_i's are independent and uniform and T and T' are determined by their values in the same way. By assumption, the distinguisher outputs 1 with probability at least $1/2 + 1/p(n)$ on this distribution. As we have noted, in this case our algorithm predicts the inner product precisely when the distinguisher outputs 1. □

Claim 2.2. *The conditional probability that the algorithm predicts the inner product bit, given that k is not the size of the intersection of r and s, is* 1/2.

Proof. Fix any $s, r \in \{0,1\}^n$ and fix k to be different from $k' = |r \cap s|$. The distribution the distinguisher sees in this case can be generated as follows: Pick random $a_1, a_2, \ldots, a_n \in \{0, \ldots, p-1\}$. Let $T = \sum_{i \in s} a_i$. Pick $x \in \{0, \ldots, p-1\}$ randomly. Let $b_i = a_i + x$ if $r_i = 1$ and $b_i = a_i$ otherwise. Let $T' = T + kx$ and let $T'' = T + k'x$. Output $b_1, b_2, \ldots, b_n, T'$. We claim that this distribution is exactly that of $a_1, a_2, \ldots, a_n, U$ where U is chosen uniformly at random from $\{0, \ldots, p-1\}$. As before, $a_1, a_2, \ldots, a_n, T$ and $b_1, b_2, \ldots, b_n, T''$ are identically distributed. We also know that $b_1, b_2, \ldots, b_n, T''$ is independent of x. We have that $T' = T'' + (k-k')x$ and since we are working in a field and $k - k' \neq 0$, T' is uniformly distributed and independent of $b_1, b_2, \ldots, b_n$.

From our assumption on the distinguisher, it outputs 1 on this distribution with probability 1/2. Since this is true for every s, r, and k, the probability of correctly predicting is 1/2 in this case. □

We now turn to the case where the arithmetic is done modulo $2^{l(n)}$. The proof is similar, however, additional complications arise because the multiplication is no longer over a field. The proof of Claim 2.1 remains unchanged. In Claim 2.2 we should consider what happens when $k - k'$ is *not* relatively prime to $2^{\ell(n)}$. Let $g = \mathrm{GCD}(2^{\ell(n)}, k - k') \leq n$ and let $y = (k - k')x \bmod 2^{\ell(n)}$. We know that $g | y$ and that y/g is distributed uniformly in $\{0, \ldots, 2^{\ell(n)}/g - 1\}$. On the other hand, since g is small (much smaller than $\ell(n)$), by part (2) of Proposition 1.1, $T'' \bmod g$ is quasi-random over $\{0, \ldots, g-1\}$. Note that given T'', $b_1, b_2, \ldots, b_n$ and $k - k'$ the conditional distribution of x is uniform in $\{0, \ldots, 2^{\ell(n)} - 1\}$ and therefore T'' is independent of y. Hence we have that

$$T'' + y = g(\lfloor T''/g \rfloor + y/g \bmod 2^{\ell}/g) + (T'' \bmod g)$$

is quasi-random over $\{0, \ldots, 2^{\ell(n)} - 1\}$. This shows that Claim 2.2 holds for arithmetic modulo $2^{l(n)}$ and concludes the proof of the theorem. □

Once we have a way of obtaining $(1+c)n$ pseudorandom bits from a seed of n bits, we can use the method of Goldreich *et al.* [18] to extend an n bit seed to an arbitrary polynomial length pseudorandom sequence. (Once $a_1, a_2, \ldots, a_n$ are chosen at random they can be fixed for all applications. Thus they do not count for the length of the seed.) We start with a seed $s \in \{0,1\}^n$ and compute $f_{\bar{a}}(s)$. The last cn of this number are output and the rest are used as the new seed. This is repeated until we have as many bits as we want.

To generate a sequence of length m we need $O(m/n)$ iterations. Each iteration requires $O(n)$ additions of number which are $O(n)$ bits long. In particular, to generate a sequence of length $2n$ requires only $O(1)$ iterations. Goldreich *et al.* [18] also show how a parallel pseudorandom number generator doubling the number of bits can be used to implement in parallel a pseudorandom generator for any length. The number of parallel iterations in their construction is $\log(m/n)$.

3. A Family of Universal One-Way Hash Functions

A family of universal one-way hash functions is a collection F of functions $f: \{0, 1\}^n \mapsto \{0, 1\}^m$ with the property that for any element $x \in \{0, 1\}^n$ if f is chosen at random from the collection F, then it is hard to find an element $y \neq x$ such that $f(y) = f(x)$ (although many such y's exist[2]). We say that such a y *collides* with x. Such functions were introduced by Naor and Yung [40], who showed how they can be applied to various solve authentication problems, most notably signature schemes and public fingerprints for files. Naor and Yung showed how such a family can be constructed given any 1–1 one-way function. However, in a dual way to the case of the pseudorandom generation, the construction achieves compression of one bit per application of the one-way function.

We suggest using the subset sum function in the case $l(n) < n$ as a one-way hash function. If $l(n) = (1 - c)n$ for $c > 0$, then $a_1, a_2, \ldots, a_n \in \{0, 1\}^{l(n)}$ define a function $f_{\bar{a}}: \{0, 1\}^n \mapsto \{0, 1\}^{l(n)}$. We claim that the collection of all such functions defines a family of universal one-hash functions, assuming subset sum is hard for this length.

Naor and Yung have shown that the composition of several families of universal one-way hash functions is also a universal one-way hash. Therefore, if we can construct a family that achieves compression of cn bits we can construct from it a family that achieves any polynomial compression, i.e., for any N, polynomial in n, we can hash a string of N bits to one which is n bits long. This can be achieved by $O(\log n)$ compositions, and therefore can be implemented in parallel poly-log time (NC). The number of addition operations required to hash a string from N bits to n bits is linear in N.

Theorem 3.1. *Let $l(n) = (1 - c)n$ for $c > 0$. If the subset-sum function for length $l(n)$ is one-way, then it is also a family of universal one-way hash functions.*

Proof. We have to show that, for any $s_1 \in \{0, 1\}^n$, if $a_1, a_2, \ldots, a_n \in \{0, 1\}^{l(n)}$ are chosen at random, then finding $s_1 \neq s_2$ such that $f_{\bar{a}}(s_1) = f_{\bar{a}}(s_2)$ it as hard as inverting the subset sum function.

Suppose not, i.e., an algorithm exists that with nonnegligible probability succeeds in finding such an s_2, then we will show how to use it to invert f with nonnegligible probability.

Given $a_1, a_2, \ldots, a_n \in \{0, 1\}^{l(n)}$ and a target sum T, we construct an input to the collision-finding algorithm as follows:

1. Let the collision-finding algorithm select a (nonempty) $s_1 \in \{0, 1\}^n$.
2. Compute $T' = \sum_{i \in s_1} a_i$. Choose a random j such that $j \in s_1$ and define $a'_j = a_j - T' + T$.
3. Give the instance $a_1, a_2, \ldots, a'_j, \ldots, a_n$ and s_1 to the algorithm that finds collisions. The algorithm attempts to find s_2 such that $f_{(a_1, a_2, \ldots, a'_j, \ldots, a_n)}(s_2) = T'$.

If the algorithm returns s_2 that collides with s_1 and $j \notin s_2$, then s_2 is a solution to our original problem, since swapping a_j and a'_j does not affect the sum over s_2. If we could argue that the distribution that the collision-finding algorithm sees is similar to its usual

[2] Note that we have not required that $m < n$, but the hash function is pretty useless otherwise.

one, then the probability that some colliding s_2 will be found is nonnegligible. Note that we can assume that $s_1 \not\subset s_2$, since otherwise we can use the collision-finding algorithm for solving the subset problem on $T = 0$ (which is at least as hard as the general problem). Furthermore, since j was chosen at random, the probability that $j \notin s_2$ is at least $1/n$. (Actually it is closer to $1/2$, however, $1/n$ is good enough for our purposes.) Hence we have a 1/poly chance of breaking the subset sum problem.

The reason that the distribution the collision-finding algorithm sees is very close to its regular one is that a_i is distributed uniformly and independently for all $i \neq j$. By part 2 of Proposition 1.1 we know that T is distributed quasi-randomly given $a_1, a_2, \ldots, a_n$. Therefore a'_j is quasi-random, and s_1 looks like a random set whose sum is T. □

The following corollary is useful in the next section.

Corollary 3.1. *Given $a_1, a_2, \ldots, a_{n/2} \in \{0, 1\}^{l(n)}$ and $a'_1, a'_2, \ldots, a'_{n/2} \in \{0, 1\}^{l(n)}$ chosen at random, finding two subsets of the a_i's and a'_i's that sum to the same value is as hard as inverting subset sum on length $l(n)$.*

As with pseudorandom generators, the system designer can chose $\vec{a}$ once and fix it. To generate from it a universal one-way hash function chose only a random $r \in \{0, 1\}^n$. The function will be $f_r(s) = f_{\vec{a}}(r \oplus s)$. This yields a succinct representation for the one-way hash function, and thus can lead to efficient provably secure digital signature schemes. Therefore in order to represent a hash function from N bits to ℓ bits using as the building block subset sum of n number of length ℓ we need altogether $n \cdot \ell + n \cdot \log_{n/\ell} N$ bits. Furthermore, the amount of storage required in order to compute the hash function is only $n \log_{n/\ell} N$ bits.

4. Efficient Universal One-Way Hash Functions with Security Based on Group-Theoretic Problems

The proof of Theorem 3.1 in the previous section did not use very much of the structure of addition in showing the equivalence between subset sum as a one-way function and subset sum as a universal one-way hash function. In fact, the analogous result holds for subset G-product (see exact definition below) for any group G whose elements can be coded as strings of a certain length, so that it is possible to multiply and invert elements in polynomial time, and to sample from G at random in polynomial time. In this section we show that inverting the subset G-product function for $n > \log(|G|)$ is as hard as inverting any onto homomorphism to G (from any other group with the listed properties). In particular, for any prime p, and for $n > \log p$, subset product mod p is as difficult to invert as it is to take discrete logarithms mod p, and for N a Blum integer, subset product mod N is as hard as factoring N. Using generalizations of the results of last section, we then obtain efficiently parallelizable universal one-way hash functions based either on discrete log or factoring Blum integers.

The results of the first section are more specific, but can also be generalized to include subset product modulo any integer N as long as n is relatively prime to $\Phi(N)$. (In general, we need $y^n \in_U G_x$ for $y \in_U G_x$.) However, we are not able to connect the security of this one-way function with that of any group-theoretic problem. Thus, it is still an open problem to construct an NC-implementable pseudorandom generator (producing arbitrarily many output bits) based on discrete log or factoring. (However, Kharitonov [30] constructed a pseudorandom generator based on factoring that has a fixed sequential preprocessing phase and can then be expanded to polynomial length in NC.)

Definition 4.1. Let $G = \{G_x, x \in A\}, A \subseteq Z$, be a family of finite groups indexed by integers, whose elements are also integers. We say that G is **polynomial-time computable** if, given x, the following operations can be performed in probabilistic polynomial-time:

1. Compute the identity of G_x.
2. Given $y, z \in G_x$ compute $y \cdot z$, where $\cdot$ is the group operation.
3. Given $y \in G_x$ compute y^{-1}, the multiplicative inverse of y in G_x.
4. Select at random $y \in_U G_x$.

Definition 4.2. Let $G = \{G_x, x \in A\}$ be a polynomial-time computable family of groups. Then the **subset G-product** function with parameters n, x takes as input an n-tuple of elements $y_1, \dots, y_n$ of G_x and an n-bit string S and outputs the product in G_x, in order of appearance in the n-tuple, of those elements y_i with $S_i = 1$. As before, we view $n, x, \vec{y}$ as being fixed, and so consider the subset G-product function as a function from $\{0, 1\}^n \to G_x$, $f_{\vec{y}}(S) = f_{n,x,\vec{y}}(S)$.

Proposition 4.1. *Let $n > c \log(|G_x|)$ for $c > 1$. Then, for all but an exponentially small fraction of the choices of $\vec{y} \in_U (G_x)^n$, the induced distribution $f_{\vec{y}}(S)$ for $S \in_U \{0, 1\}^n$ is statistically indistinguishable within an exponentially small amount from the uniform distribution on G_x.*

Proof. The proof is analogous to Proposition 1.1. (One technical point is that one needs the leftover hash lemma of [26], which works under the weaker assumption that the hash functions in question are *universal*$_2$ in the sense of [10], rather than pairwise independent.) □

Theorem 4.1. *Let $G = \{G_x | x \in A\}$ and $H = \{H_y | y \in B\}$ be polynomial-time computable families of groups, let $g\colon B \to A$ be a polynomial-time computable function, and let $h = \{h_y | y \in B\}$ be a family of polynomial-time computable onto homomorphisms from H_y to $G_{g(y)}$. Then any algorithm to invert the subset G-product function with parameters $x = g(y), n > c \log(|G_X|), c > 1$ with nonnegligible probability of success on random inputs, yields an algorithm to invert h_y.*

Proof. Let A be the algorithm inverting the subset G-product function. Let $z \in_U G_x$. To find $h_y^{-1}(z)$, we randomly generate $b_1, \dots, b_{n-1} \in_U H_y$ and compute $r_i = h_y(b_i)$ for $i = 1, \dots, n-1$, and let $r_n = z$. We then pick a random permutation π and let $r_i' = r_{\pi(i)}$.

Let $T = h_y(t)$, $t \in_U H_y$. We then run A on input $\vec{r}'$ and T. If it outputs a solution S so that $S(\pi(n)) = 1$, we have $(r_1')^{S(1)} \cdots (r'_{\pi(n)-1})^{S(\pi(n)-1)} z (r'_{\pi n+1})^{S(\pi(n)+1)} \cdots (r_n')^{S(n)} = T$. Since we know an h_y^{-1} for all the r_i', $i \neq \pi(n)$, and for T, we can then compute an h_y^{-1} for z. Now, since $\vec{r}'$ is uniformly distributed, by Proposition 4.1, the distribution on T (a random group element) given $\vec{r}'$ is indistinguishable from the output of the subset G-product function with the same parameters. Thus, by assumption, A finds some solution S with nonnegligible probability. If A does so, then since $S = 0^n$ only when $T = 1$, and since π is independent of $\vec{r}'$, $S(\pi(n)) = 1$ with probability at least $1/n$. Hence, our inverting algorithm succeeds with nonnegligible probability. □

Corollary 4.1. *Subset product in a finite field F_p for $n > c \log p$, $c > 1$, is as secure as discrete log in F_p.*

Proof. Let g be a generator for F_p and let $H_{p-1,g}$ be the additive group mod $p - 1$. Then $h_{p-1,g}(y) = g^y$ is an onto group homomorphism. □

Corollary 4.2. *Let N be an integer and let QR_N be the subgroup of quadratic residues* mod N. *Then, for $n > c \log N$, $c > 1$, the subset product in QR_N function is as secure as factoring N.*

Proposition 4.2. *Let $N = pq$ be a product of two primes. Then, for $n > c \log N$, $c > 1$, the subset product* mod N *function is as secure as factoring N.*

Proof. We reduce to the subset product in QR_N function. Let A be any algorithm inverting subset product mod N function on random outputs. Let a and b be such that $a \in QR_p$, $a \notin QR_q$ and $b \notin QR_p$, $b \in QR_q$. (a and b can be chosen at random, and will have the desired properties with a constant probability.) Given an instance $x_1, \ldots, x_n$, T of subset product in QR_N, we reduce to an instance of subset product mod N as follows. We let $r_i, s_i \in \{-1, 0, 1\}$ so that each is -1 with probability $1/4$, 0 with probability $1/2$, and 1 with probability $1/4$ and let $y_i = a^{r_i} b^{s_i} x_i$. We pick $r, s \in \{-1, 0, 1\}$ similarly and let $T' = a^r b^s T$. All the y_i's and T' are uniformly distributed mod N, and so by Proposition 4.1 are exponentially indistinguishable from a random output of the subset product mod N function. Thus, A finds a subset S with $T' = \Pi_{i \in S} y_i$.

We claim that with probability at least $\Omega(1/n)$, $T = \Pi_{i \in S} x_i$. To simplify the discussion, let $y_{n+1} = (T')^{-1}$, $r_{n+1} = -r$, $s_{n+1} = -s$, $x_{n+1} = T^{-1}$, and $S' = S \cup \{n + 1\}$. Then we have $\Pi_{i \in S'} y_i = 1$, $y_i = a^{r_i} b^{s_i} x_i$, and we want $\Pi_{i \in S'} x_i = 1$. Let $R_1 = \{i \in S' | r_i \in \{-1, 1\}\} = \{i \in S' | y_i \notin QR_p\}$ and $R_2 = \{i \in S' | s_i \in \{-1, 1\}\} = \{i \in S' | y_i \notin QR_q\}$. Then given $\vec{y}$ (which determines the distribution on S and hence on R_1, R_2), for each $i \in R_1$ we have that r_i is uniformly and independently distributed in $\{-1, 1\}$, and similarly for each $i \in R_2$ and s_i. Hence, for any $\vec{y}$ and $S = A(\vec{y})$, there is an $\Omega(1/n^{1/2})$ probability that $\sum_{i \in S'} r_i = 0$ and a similar and independent probability that $\sum_{i \in S'} s_i = 0$. Therefore the probability that both events occur is $\Omega(1/n)$. In this case, $1 = \Pi_{i \in S'} y_i = a^0 b^0 \Pi_{i \in S'} x_i$, so we have solved the instance of QR_N subset product. □

Finally, we note:

Theorem 4.2. *Let $G = \{G_x | x \in A\}$ be a family of polynomial-time computable groups. Then if $n > c \log(|G_x|)$ for $c > 1$, and the subset G-product function with parameters x and n is one-way, it is also a universal one-way hash function.*

Proof. Analogous to Theorem 3.1. □

5. Bit Commitment Versus a Strong Receiver

Bit commitment is a basic protocol which is useful and essential in many cryptographic applications, such as coin flipping by telephone [3], zero-knowledge and minimum disclosure proofs [22], [7], and identification schemes [13].

Naor [38] has shown how to implement bit commitment given any pseudorandom generator. His scheme suffices for the all applications above, except minimum disclosure. Furthermore, his scheme enables commitment to n bits at the price of generating a pseudorandom sequence of length $O(n)$. In our context it implies that if subset sum with $l(n) = (1 + c)n$ is one-way, then commitment to n bits can be done with n additions.

As mentioned, minimum disclosure requires a special kind of bit commitment, one which is secure versus a computationally unlimited receiver. In [7] it is shown how to implement this kind of bit commitment based on factoring and discrete log. Naor *et al.* [39] show a general but highly interactive protocol, which can be based on *any* one-way permutation.

Bit commitment is a way for a committer Alice to send a receiver Bob a locked box containing a bit. Only Alice has a key to the box, which she can send to Bob at a later stage. Bob can be sure that the contents of the box are fixed, i.e., not tampered with between the time he received it and the time it was opened. Alice can be sure that Bob has no idea what the value of the bit in the box is. Many times during a protocol, the box is never opened, and Alice wishes that it never be opened in any time in the future. In most bit commitment schemes, Bob can open the box, if he has superpolynomial time. In a strong receiver bit commitment protocol, Alice should have the confidence that the box cannot be opened even by a Bob with an unlimited computational ability.

We give a construction which is based on the assumption that subset sum with $l(n) = cn$, where $c < \frac{1}{2}$, is one-way. Unlike the other constructions in this paper, we do not claim that it is particularly efficient. A more practical version is possible if we assume that a trusted third party, such as the protocol designer, fixes some parameters at random. (The trusted party need not be available during the execution of the protocol!)

We give the protocol assuming that the trusted third party has chosen $a_1, a_2, \ldots, a_{n/2} \in \{0, 1\}^{l(n)}$ and $a'_1, a'_2, \ldots, a'_{n/2} \in \{0, 1\}^{l(n)}$. (If there is no such third party, then the a_i's and a'_i's can be chosen via coin flipping over the telephone [3] which can be implemented using a bit commitment scheme versus a strong *committer*, as in [38]. This commitment can also be based on the hardness of subset sum.)

The commit protocol to a bit b:

1. Alice chooses $s \in_R \{0, 1\}^{n/2}$.
2. If $b = 0$, then Alice sends Bob $T = \sum_{i \in s} a_i$.
3. If $b = 1$, then Alice sends Bob $T = \sum_{i \in s} a'_i$.

To reveal, she sends s and b. Bob verifies that the commit protocol was obeyed.

Alice's unconditional security is based on part 2 of Proposition 1.2: the distributions on T given that $b = 0$ or 1 are both quasi-random (with high probability) and thus indistinguishable even to a strong Bob. Bob's security is based on the fact proved in Section 3 that if an $l(n)$ subset sum function is one-way, then given two sets of $n/2$ numbers, it is hard to find two subsets that sum to the same value.

6. Subset Sum with Constant-Depth Circuits

One of the few lower bounds known in complexity theory concern the class of polynomial-sized constant-depth circuits (AC^0). It is known that the parity function cannot be computed in AC^0 and hence summing many numbers cannot be done in AC^0 [1], [16] (exponential lower bounds on the size of a constant-depth circuit to compute these functions are given in [48] and [23]). However, we show that random instances of the subset-sum problem can be generated in AC^0. If subset sum is secure for any length $l(n)$, then our construction gives a one-way function and a pseudorandom generator computable in AC^0. This is the first example of such functions. In contrast, Linial *et al.* [34] have shown that no pseudorandom *function* generator, as defined by Goldreich *et al.* [18], can be implemented in AC^0 (more precisely, no pseudorandom function secure against an adversary operating in $n^{\mathrm{polylog}(n)}$ time can be computable in AC^0). We should also note that our results have been applied by Kharitonov [29] to show the hardness of learning problems.

The basic idea is to generalize a method suggested by Babai [2] and Boppana and Lagarias [6]. Instead of first generating a random $\vec{a}$ and then computing a sum, which cannot be done in AC^0, we generate the sum, the subset, and $\vec{a}$ simultaneously. Successive members of the subset should cancel each other, so that the biggest and smallest members of the subset are the only indices that affect the sum.

For any length $l(n)$, we can generate uniformly at random $a_1, a_2, \ldots, a_n \in \{0, 1\}^{l(n)}$, $S \in \{0, 1\}^n$, and $T = \sum_{i \in S} a_i \bmod 2^{l(n)}$. The generator is the following:

```
Input: $n + n \cdot \ell(n)$ bits.
  1. Generate $S \in \{0, 1\}^n$ and $a'_1, a'_2, \ldots, a'_n \in \{0, 1\}^{l(n)}$ from the input bits.
  2. Find $i_{\min} = \min_{i \in S} i$ and $i_{\max} = \max_{i \in S} i$.
  3. For every $i \in S$ such that $i > i_{\min}$ evaluate $j_i = \max\{j < i \mid j \in S\}$.
  4. For each $1 \le i \le n$ set $a_i$ by the following rules:
       • If $i \notin S$, then set $a_i = a'_i$.
       • If $i \in S$ and $i > i_{\min}$, then set $a_i = a'_i - a'_{j_i}$.
       • If $i = i_{\min}$, then set $a_i = a'_i$.
  5. Compute $T = \sum_{i \in S} a_i = \sum_{i \in S} a'_i - a'_{j_i} = a_{i_{\max}} - a_{i_{\min}} \bmod 2^{l(n)}$.
Output: $T, a_1, a_2, \ldots a_n$ ($(n + 1) \cdot \ell(n)$ bits).
```

Every step in the above procedure can be computed in AC^0: steps 2 and 3 require only an AND of at most n inputs and steps 4 and 5 are can be done in AC^0 since addition and subtraction are in AC^0.

Claim 6.1. *If the inputs $S \in \{0, 1\}^n$ and $a'_1, a'_2, \ldots, a'_n \in \{0, 1\}^{l(n)}$ to the above procedure are random, then the output $T, a_1, a_2, \ldots a_n$ is distributed as a sum of a subset of numbers all chosen uniformly at random*

Thus, if subset sum is secure for any $l(n)$, the procedure above defines a one-way function which is computable in AC^0. (The function maps $S \in \{0, 1\}^n$ and $a'_1, a'_2, \ldots, a'_n$ to T and $a_1, a_2, \ldots, a_n$.) This is the first example of such a function that we know of. As we have mentioned in Section 1, if subset sum is secure for any $l(n)$, then it is secure for $n + \log n$. Therefore, by Theorem 2.2 the procedure above with $\ell(n) = n + \log n$ defines a pseudorandom generator that expands the input by $\log n$ bits. We do not know if one can construct in AC^0 a pseudorandom generator based on subset sum that expands the input by any polynomial.

7. Conclusions

We have shown a number of very efficient implementation of cryptographic schemes which are secure if subset sum is hard to invert for certain length parameters. More cryptanalytic effort should be devoted to understanding the difficulty of subset sum for these parameters before our constructions should be used. We hope that our results will encourage such an effort. Our results also shed some light on the theoretical question of whether it is possible to have meaningful cryptography in very low-level complexity classes.

Acknowledgments

We thank Manuel Blum for suggesting that we look at subset sum, Ernie Brickell for helpful advise about the state of the art of solving low-density subset problems, and Mike Luby and Ronitt Rubinfeld for interesting discussions. We thank the anonymous referees for their diligent reading and useful suggestions.

Note Added in Proof. A recent result of M. Ajtai ("Generating Hard Instances of Lattice Problems", to appear, *Proc. 28th Symposium on Theory of Computing*, 1996) implies that if the subset sum problem is easy on the average, then the shortest vector in a lattice problem is easy in the worst case.

References

[1] M. Ajtai, Σ_1^1 formulas of finite structures, *Ann. Pure Appl. Logic*, vol. 24, 1983, pp. 1–48.

[2] L. Babai, Random oracles separate PSPACE from the polynomial-time hierarchy, *Inform. Proc. Lett.*, vol. 26, 1987, pp. 51–53.

[3] M. Blum, Coin flipping by telephone, *Proc. 24th IEEE Compcon*, 1982, pp. 133–137.

[4] L. Blum, M. Blum, and M. Shub, A simple unpredictable pseudo-random number generator, *Siam J. Comput.*, Vol. 15, 1986, pp. 364–383.

[5] M. Blum and S. Micali, How to generate cryptographically strong sequences of pseudo-random bits, *SIAM J. Comput.*, vol. 13, 1984, pp. 850–864.

[6] R. B. Boppana and J. C. Lagarias, One-way functions and circuit complexity, in *Proc. Structure in Complexity Theory*, Springer-Verlag, New York, 1986, pp. 51–65.

[7] G. Brassard, D. Chaum, and C. Crepeau, Minimum disclosure proofs of knowledge, *J. Comput. System Sci.*, vol. 37, 1988, pp. 156–189.

[8] E. F. Brickell, Solving low density knapsacks, *Proc. Crypto* '83, pp. 25–37.

[9] E. F. Brickell and A. M. Odlyzko, Cryptanalysis: a survey of recent results, *Proc. IEEE*, vol. 76, 1988, pp. 578–593.

[10] L. Carter and M. Wegman, Universal classes of hash functions, *J. Comput. System Sci.*, vol. 18, 1979, pp. 143–154.

[11] B. Chor and R. L. Rivest, A knapsack type public key crypto-system based on arithmetic in finite fields, *IEEE Trans. Informa. Theory*, vol. 34, 1988, pp. 901–909.

[12] M. J. Coster, B. A. LaMacchia, A. M. Odlyzko, and C. P. Schnorr, An improved low-density subset sum algorithm, *Proc. Advances in Cryptology—Eurocrypt* '91, Springer-Verlag, Berlin, 1991, pp. 54–67.

[13] A. Fiat and A. Shamir, How to prove yourself, *Proc. Advances in Cryptology—Crypto* '86, Springer-Verlag, Berlin, 1987, pp. 641–654.

[14] A. M. Frieze, On the Lagarias–Odlyzko algorithm for the subset sum problem, *SIAM J. Comput.*, vol. 15, 1986, pp. 536–539.

[15] M. Furst and R. Kannan, Succinct certificates for almost all subset sum problems, *SIAM J. Comput.*, vol. 18, 1989, pp. 550–558.

[16] M. Furst, J. Saxe, and M. Sipser, Parity circuits and the polynomial time hierarchy, *Proc. 22nd Symposium on Foundations of Computer Science*, 1981, pp. 260–270.

[17] Z. Galil and O. Margalit, An almost linear time algorithm for the dense subset sum problem, *SIAM J. Comput.*, vol. 20, 1991, pp. 1157–1189.

[18] O. Goldreich, S. Goldwasser, and S. Micali, How to construct random functions, *J. Assoc. Comput. Mach.*, vol. 33, 1986, pp. 792–807.

[19] O. Goldreich, H. Krawczyk, and M. Luby, On the existence of pseudorandom generators, *Proc. 29th Symposium on the Foundation of Computer Science*, 1988, pp. 12–24.

[20] O. Goldreich and L. Levin, A hard-core predicate for all one-way functions, *Proc. 21st Symposium on the Theory of Computing*, 1989, pp. 25–32.

[21] S. Goldwasser and S. Micali, Probabilistic encryption, *J. Comput. System Sci.*, vol. 28, 1984, pp. 270–299.

[22] O. Goldreich, S. Micali, and A. Wigderson, Proofs that yield nothing but their validity and a methodology of cryptographic protocol design, *Proc. 27th Symposium on Foundations of Computer Science*, 1986, pp. 174–187.

[23] J. Hastad, Improved lower bounds for small depth circuits, *Proc. 18th Symposium on Theory of Computing*, 1986, pp. 6–20.

[24] J. Hastad, Pseudo-random generators under uniform assumptions, *Proc. 19th Symposium on Theory of Computing*, 1990, pp. 395–404.

[25] R. Impagliazzo, L. Levin, and M. Luby, Pseudo-random generation from one-way functions, *Proc. 21st Symposium on Theory of Computing*, 1989, pp. 12–24.

[26] R. Impagliazzo and D. Zuckerman, Recycling random bits, *Proc. 30th Symposium on Foundations of Computer Science*, 1989, pp. 248–253.

[27] A. Joux and J. Stern, Improving the critical complexity of the Lagarias–Odlyzko attack against subset sum problems, *Proc. 8th International Conference on Fundamentals of Computation Theory*, Springer-Verlag, Berlin, 1991, pp. 258–264.

[28] R. M. Karp, Reducibility among combinatorial problems, in *Complexity of Computer Computation*, R. E. Miller and J. W. Thatcher, eds., Plenum, New York, 1972.

[29] M. Kharitonov, Cryptographic lower bounds for learnability of Boolean functions on the uniform distribution, *Proc. 5th COLT*, Morgan Kaufman, Los Altos, CA, 1992, pp. 29–36.

[30] M. Kharitonov, Cryptographic hardness of distribution-specific learning, *Proc. 25th Symposium on Theory of Computing*, 1993, pp. 372–381.

[31] H. Krawczyk, Secret sharing made short, *Proc. Advances in Cryptology—Crypto* '93, Springer-Verlag, Berlin, 1994, pp. 136–146.

[32] J. C. Lagarias and A. M. Odlyzko, Solving low density subset sum problems, *J. Assoc. Comput. Mach.*, vol. 32, 1985, pp. 229–246.

[33] A. K. Lenstra, H. W. Lenstra, and L. Lovász, Factoring polynomials with rational coefficients, *Math. Ann.*, vol. 261, 1982, pp. 515–534.

[34] N. Linial, Y. Mansour, and N. Nisan, Constant depth circuits, Fourier transform, and learnability, *J. Assoc. Comput. Mach.*, vol. 40, 1993, pp. 607–620.

[35] M. Luby and C. Rackoff, How to construct a pseudo-random permutation from a pseudo-random function, *SIAM J. Çomput.*, vol. 17, 1988, pp. 373–386.

[36] R. C. Merkle and M. Hellman, Hiding information and signature in trapdoor knapsack, *IEEE Trans. Inform. Theory*, vol. 24, 1978, pp. 525–530.

[37] S. Micali and C. P. Schnorr, Efficient, perfect polynomial random number generators, *J. Cryptology*, vol. 3, 1991, pp. 157–172.

[38] M. Naor, Bit commitment using pseudo-randomness, *J. Cryptology*, vol. 4, 1991, pp. 151–158.

[39] M. Naor, R. Ostrovsky, R. Venkatesan, and M. Yung, Perfect zero-knowledge arguments for NP can be based on general complexity assumptions, *Proc. Advances in Cryptology—Crypto* '92, Springer-Verlag, Berlin, 1993, pp. 196–214.

[40] M. Naor and M. Yung, Universal one-way hash functions and their cryptographic applications, *Proc. 21st Symposium on the Theory of Computing*, 1989, pp. 33–43.

[41] A. M. Odlyzko, The rise and fall of knapsack cryptosystems, in *Cryptology and Computational Number Theory*, C. Pomerance ed., Proc. Sympos. Appl. Math, vol. 42, 1990, American Mathematical Society, Providence, RI, pp. 75–88.

[42] J. Rompel, One-way functions are necessary and sufficient for secure signatures, *Proc. 22nd Symposium on Theory of Computing*, 1990, pp. 387–394.

[43] M. Santha and U. V. Vazirani, Generating quasi-random sequences from slightly-random sources, *Proc. 25th Symposium on the Theory of Computing*, 1984, pp. 434–440.

[44] C. P. Schnorr and M. Euchner, Lattice base reduction: improved practical algorithms for solving subset sum problems, *Math. Programming*, vol. 66, 1994, pp. 181–199.

[45] C. P. Schnorr and H. H. Hörner, Attacking the Chor–Rivest cryptosystem by improved lattice reduction, Manuscript, 1994.

[46] R. Schroeppel and A. Shamir, A $T \cdot S^2 = O(2^n)$ time/space tradeoff for certain NP-complete problems, *Proc. 20th Symposium on Foundations of Computer Science*, 1979, pp. 328–336.

[47] A. C. Yao, Theory and applications of trapdoor functions, *Proc. 23rd Symposium on Foundations of Computer Science*, 1982, pp. 80–91.

[48] A. C. Yao, Separating the polynomial time hierarchy by oracles, *Proc. 26th Symposium on Foundations of Computer Science*, 1985, pp. 1–10.