A Generalized Birthday Problem
(extended abstract)

David Wagner

University of California at Berkeley

Abstract. We study a k-dimensional generalization of the birthday problem: given k lists of n-bit values, find some way to choose one element from each list so that the resulting k values XOR to zero. For $k = 2$, this is just the extremely well-known birthday problem, which has a square-root time algorithm with many applications in cryptography. In this paper, we show new algorithms for the case $k > 2$: we show a cube-root time algorithm for the case of $k = 4$ lists, and we give an algorithm with subexponential running time when k is unrestricted.

We also give several applications to cryptanalysis, describing new subexponential algorithms for constructing one-more forgeries for certain blind signature schemes, for breaking certain incremental hash functions, and for finding low-weight parity check equations for fast correlation attacks on stream ciphers. In these applications, our algorithm runs in $O(2^{2\sqrt{n}})$ time for an n-bit modulus, demonstrating that moduli may need to be at least 1600 bits long for security against these new attacks. As an example, we describe the first-known attack with subexponential complexity on Schnorr and Okamoto-Schnorr blind signatures over elliptic curve groups.

1 Introduction

One of the best-known combinatorial tools in cryptology is the birthday problem:

Problem 1. Given two lists L_1, L_2 of elements drawn uniformly and independently at random from $\{0,1\}^n$, find $x_1 \in L_1$ and $x_2 \in L_2$ such that $x_1 \oplus x_2 = 0$.

(Here the $\oplus$ symbol represents the bitwise exclusive-or operation.) The birthday problem is well understood: A solution x_1, x_2 exists with good probability once $|L_1| \times |L_2| \gg 2^n$ holds, and if the list sizes are favorably chosen, the complexity of the optimal algorithm is $\Theta(2^{n/2})$. The birthday problem has numerous applications throughout cryptography and cryptanalysis.

In this paper, we explore a generalization of the birthday problem. The above presentation suggests studying a variant of the birthday problem with an arbitrary number of lists. In this way, we obtain the following k-dimensional analogue, which we call the k-sum problem:

Problem 2. Given k lists $L_1, \ldots, L_k$ of elements drawn uniformly and independently at random from $\{0,1\}^n$, find $x_1 \in L_1, \ldots, x_k \in L_k$ such that $x_1 \oplus x_2 \oplus \cdots \oplus x_k = 0$.

We allow the lists to be extended to any desired length, and so it may aid the intuition to think of each element of each list as being generated by a random (or pseudorandom) oracle R_i, so that the j-th element of L_i is $R_i(j)$. It is easy to see that a solution to the k-sum problem exists with good probability so long as $|L_1| \times \cdots \times |L_k| \gg 2^n$. However, the challenge is to find a solution $x_1, \ldots, x_k$ explicitly and efficiently.

This first half of this paper is devoted to a theoretical study of this problem. First, Section 2 describes a new algorithm, called the k-tree algorithm, that allows us to solve the k-sum problem with lower complexity than previously known to be possible. Our algorithm works only when one can extend the size of the lists freely, i.e., in the special case where there are sufficiently many solutions to the k-sum problem. For example, we show that, for $k = 4$, the k-sum problem can be solved in $O(2^{n/3})$ time using lists of size $O(2^{n/3})$. We also discuss a number of generalizations of the problem, e.g., to operations other than XOR. Next, in Section 3, we study the complexity of the k-sum problem and give several lower bounds. This theoretical study provides a tool for cryptanalysis which we will put to use in the second half of the paper.

The k-sum problem may not appear very natural at first sight, and so it may come as no surprise that, to our knowledge, this problem has not previously been stated or studied in full generality. Nonetheless, we show in the second half of this paper a number of cases where the k-sum problem has applications to cryptanalysis of various systems: in Section 4, we show how to break various blind signature schemes, how to attack several incremental hash functions, and how to find low-weight parity checks. Other examples may be found in the full version of this paper [36]. We do not claim that this is an exhaustive list of possible applications; rather, it is intended to motivate the relevance of this problem to cryptography.

Finally, we conclude in Sections 5 and 6 with several open problems and final remarks.

2 Algorithms

The classic birthday problem. We recall the standard technique for finding solutions to the birthday problem (with 2 lists). We define a join operation $\bowtie$ on lists so that $S \bowtie T$ represents the list of elements common to both S and T. Note that $x_1 \oplus x_2 = 0$ if and only if $x_1 = x_2$. Consequently, a solution to the classic (2-list) birthday problem may be found by simply computing the join $L_1 \bowtie L_2$ of the two input lists L_1, L_2. We represent this algorithm schematically in Figure 1.

The join operation has been well-studied in the literature on database query evaluation, and several efficient methods for computing joins are known. A merge-join sorts the two lists, L_1, L_2, and scans the two sorted lists, returning any matching pairs detected. A hash-join stores one list, say L_1, in a hash table, and then scans through each element of L_2 and checks whether it is present in the hash table. If memory is plentiful, the hash-join is very efficient:

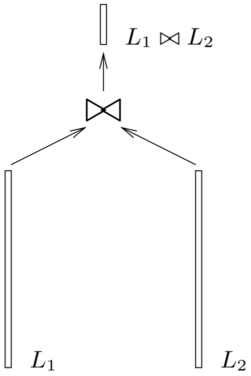

Fig. 1. An abstract representation of the standard algorithm for the (2-list) birthday problem: given two lists L_1, L_2, we use a join operation to find all pairs (x_1, x_2) such that $x_1 = x_2$ and $x_1 \in L_1$ and $x_2 \in L_2$. The thin vertical boxes represent lists, the arrows represent flow of values, and the $\bowtie$ symbol represents a join operator.

it requires $|L_1| + |L_2|$ simple steps of computation and $\min(|L_1|, |L_2|)$ units of storage. A merge-join is slower in principle, running in $O(n \log n)$ time where $n = \max(|L_1|, |L_2|)$, but external sorting methods allow computation of merge-joins even when memory is scarce.

The consequence of these observations is that the birthday problem may be solved with square-root complexity. In particular, if we operate on n-bit values, then the above algorithms will require $O(2^{n/2})$ time and space, if we are free to choose the size of the lists however we like. Techniques for reducing the space complexity of this algorithm are known for some important special cases [25].

The birthday problem has many applications. For example, if we want to find a collision for a hash function $h : \{0,1\}^* \to \{0,1\}^n$, we may define the j-th element of list L_i as $h(i, j)$. Assuming that h behaves like a random function, the lists will contain an inexhaustible supply of values distributed uniformly and independently at random, so the premises of the problem statement will be met. Consequently, we can expect to find a solution to the corresponding birthday problem with $O(2^{n/2})$ work, and any such solution immediately yields a collision for the hash function [38].

The 4-list birthday problem. To extend the above well-known observations, consider next the 4-sum problem. We are given lists $L_1, \ldots, L_4$, and our task is to find values $x_1, \ldots, x_4$ that XOR to zero. (Hereafter $x_i \in L_i$ holds implicitly.) It is easy to see that a solution should exist with good probability if each list is of length at least $2^{n/4}$. Nonetheless, no good algorithm for explicitly finding such a solution was previously known: The most obvious approaches all seem to require $2^{n/2}$ steps of computation.

We develop here a more efficient algorithm for the 4-sum problem. Let $\text{low}_\ell(x)$ denote the least significant ℓ bits of x, and define the generalized join operator

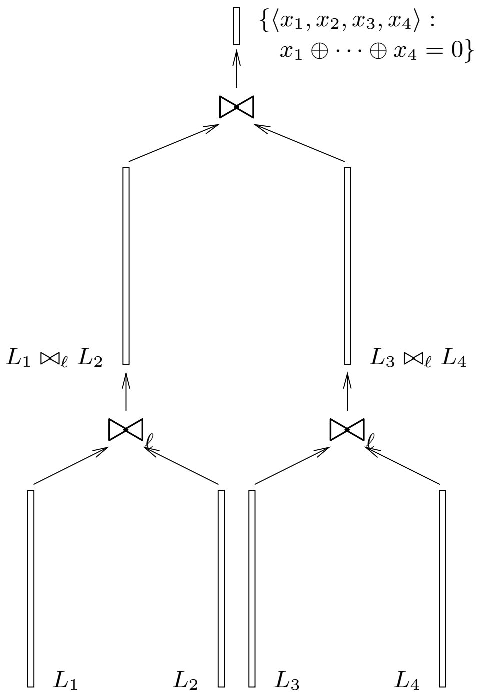

Fig. 2. A pictorial representation of our algorithm for the 4-sum problem.

$\bowtie_\ell$ so that $L_1 \bowtie_\ell L_2$ contains all pairs from $L_1 \times L_2$ that agree in their ℓ least significant bits. We will use the following basic properties of the problem:

Observation 1 *We have $low_\ell(x_i \oplus x_j) = 0$ if and only if $low_\ell(x_i) = low_\ell(x_j)$.*

Observation 2 *Given lists L_i, L_j, we can easily generate all pairs $\langle x_i, x_j \rangle$ satisfying $x_i \in L_i$, $x_j \in L_j$, and $low_\ell(x_i \oplus x_j) = 0$ by using the join operator $\bowtie_\ell$.*

Observation 3 *If $x_1 \oplus x_2 = x_3 \oplus x_4$, then $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$.*

Observation 4 *If $low_\ell(x_1 \oplus x_2) = 0$ and $low_\ell(x_3 \oplus x_4) = 0$, then we necessarily have $low_\ell(x_1 \oplus x_2 \oplus x_3 \oplus x_4) = 0$, and in this case $\Pr[x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0] = 2^\ell / 2^n$.*

These properties suggest a new algorithm for the 4-sum problem. First, we extend the lists $L_1, \ldots, L_4$ until they each contain about 2^ℓ elements, where ℓ is a parameter to be determined below. Then, we apply Observation 2 to generate a large list L_{12} of values $x_1 \oplus x_2$ such that $\mathrm{low}_\ell(x_1 \oplus x_2) = 0$. Similarly, we generate a large list L_{34} of values $x_3 \oplus x_4$ where $\mathrm{low}_\ell(x_3 \oplus x_4) = 0$. Finally, we search for matches between L_{12} and L_{34}. By Observation 3, any such match will satisfy $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$ and hence will yield a solution to the 4-sum problem. See Figure 2 for a visual depiction of this algorithm.

The complexity of this algorithm may be analyzed as follows. We have $\Pr[\mathrm{low}_\ell(x_1 \oplus x_2) = 0] = 1/2^\ell$ when x_1, x_2 are chosen uniformly at random. Thus, by the birthday paradox (or by linearity of expectation),

$$\mathbb{E}[|L_{12}|] = |L_1| \times |L_2|/2^\ell = 2^{2\ell}/2^\ell = 2^\ell.$$

Similarly, L_{34} has expected size 2^ℓ. Moreover, Observation 4 ensures that any pair of elements from $L_{12} \times L_{34}$ yields a match with probability $2^\ell/2^n$. Therefore, a second invocation of the birthday paradox shows that the expected number of elements in common between L_{12} and L_{34} is about $|L_{12}| \times |L_{34}|/2^{n-\ell}$. The latter is at least 1 when $\ell \geq n/3$. Consequently, if we set $\ell \overset{\text{def}}{=} n/3$ as the beginning of the above procedure, we expect to find a solution to the 4-sum problem with non-trivial probability. Since the size of all lists is around $2^{n/3}$, the resulting algorithm can be implemented with $O(2^{n/3})$ time and space.

Extensions. The above algorithm finds only solutions with a special property, namely, $x_1 \oplus x_2$ and $x_3 \oplus x_4$ are zero in their low ℓ bits. However, this restriction was made merely for ease of presentation, and it can be eliminated. To sample randomly from the set of all solutions, pick a random ℓ-bit value α, and look for pairs (x_1, x_2) and (x_3, x_4) whose low ℓ bits XOR to α. In other words, compute $(L_1 \bowtie_\ell (L_2 \oplus \alpha)) \bowtie (L_3 \bowtie_\ell (L_4 \oplus \alpha))$.

Also, the value 0 in $x_1 \oplus \cdots \oplus x_k = 0$ is not essential, and can be replaced by any other constant c without increasing the complexity of the problem. This may be easily seen as follows: if we replace L_k with $L'_k = L_k \oplus c \overset{\text{def}}{=} \{x_k \oplus c : x_k \in L_k\}$, then any solution to $x_1 \oplus \cdots \oplus x_{k-1} \oplus x'_k = 0$ will be a solution to $x_1 \oplus \cdots \oplus x_k = c$ and vice versa. Consequently, we may assume (without loss of generality) that $c = 0$.

As a corollary, when $k > k'$ the complexity of the k-sum problem can be no larger than the complexity of the k'-sum problem. This can be proven using a trivial list-elimination trick. We pick arbitrary values $x_{k'+1}, \ldots, x_k$ from $L_{k'+1}, \ldots, L_k$ and fix this choice. Then, we set $c \overset{\text{def}}{=} x_{k'+1} \oplus \cdots \oplus x_k$ and use a k'-sum algorithm to find a solution to the equation $x_1 \oplus \cdots \oplus x_{k'} = c$. For instance, this shows that we can solve the k-sum problem with complexity at most $O(2^{n/3})$ for all $k \geq 4$.

More interestingly, we can use the above ideas to solve the k-sum problem even faster than cube-root time for larger values of k. We extend the 4-list tree algorithm above as follows. When k is a power of two, we replace the complete binary tree of depth 2 in Figure 2 with a complete binary tree of depth $\lg k$. At internal nodes of height h, we use the join operator $\bowtie_{\ell_h}$ (where $\ell_h = hn/(1 + \lg k)$), except that at the root we use the full join operator $\bowtie$. Each element x of an internal list $L_{i \ldots j}$ contains back-pointers to elements x' and x'' of the two child lists used to form $L_{i \ldots j}$, such that $x = x' \oplus x''$. In this way we will obtain an algorithm for the k-sum problem that requires $O(k \cdot 2^{n/(1+\lg k)})$ time and space and uses lists of size $O(2^{n/(1+\lg k)})$. The complexity of this algorithm improves only slowly as k increases, though, so this does not seem to yield large improvements unless k becomes quite large.

We can also obtain an algorithm for the general k-sum problem when k is not a power of two. We take $k' \stackrel{\text{def}}{=} 2^{\lfloor \lg k \rfloor}$ to be the largest power of two less than k, and we use the list-elimination trick above. However, the results are less satisfying: The algorithm obtained in this way runs essentially no faster for $k = 2^i + j$ than for $k = 2^i$.

Operations other than XOR. The k-sum problem has so far been described over the group $(GF(2)^n, \oplus)$, but it is natural to wonder whether these techniques will apply over other groups as well.

We note first that the tree algorithm above transfers immediately to the additive group $(\mathbb{Z}/2^n\mathbb{Z}, +)$. In particular, we compute $L_{12} \stackrel{\text{def}}{=} L_1 \bowtie_\ell -L_2$, $L_{34} \stackrel{\text{def}}{=} L_3 \bowtie_\ell -L_4$, and finally $L_{12} \bowtie -L_{34}$, where $-L \stackrel{\text{def}}{=} \{-x \bmod 2^n : x \in L\}$. The result will be a set of solutions to the equation $x_1 + \cdots + x_k \equiv 0 \pmod{2^n}$. The reason this works is that $a \equiv b \pmod{2^\ell}$ implies $(a + c \bmod 2^n) \equiv (b + c \bmod 2^n) \pmod{2^\ell}$: the carry bit propagates in only one direction.

We can also apply the tree algorithm to the group $(\mathbb{Z}/m\mathbb{Z}, +)$ where m is arbitrary. Let $[a, b] \stackrel{\text{def}}{=} \{x \in \mathbb{Z}/m\mathbb{Z} : a \leq x \leq b\}$ denote the interval of elements between a and b (wrapping modulo m), and define the join operation $L_1 \bowtie_{[a,b]} L_2$ to represent the solutions to $x_1 + x_2 \in [a, b]$ with $x_i \in L_i$. Then we may solve a 4-sum problem over $\mathbb{Z}/m\mathbb{Z}$ by computing $(L_1 \bowtie_{[a,b]} L_2) \bowtie (L_3 \bowtie_{[a,b]} L_4)$ where $[a, b] = [-m/2^{\ell+1}, m/2^{\ell+1} - 1]$ and $\ell = \frac{1}{3} \lg m$. In general, one can adapt the k-tree algorithm to work in $(\mathbb{Z}/m\mathbb{Z}, +)$ by replacing each $\bowtie_\ell$ operator with $\bowtie_{[-m/2^{\ell+1}, m/2^{\ell+1}-1]}$, and this will let us solve k-sum problems modulo m about as quickly as we can for XOR.

Finding many solutions. In some applications, it may be useful to find many solutions to the k-sum problem. It is not too hard to see[1] that we can find α^3 solutions to the 4-sum problem with about α times the work of finding a single solution, so long as $\alpha \leq 2^{n/6}$. Similarly, we can find $\alpha^{1+\lfloor \lg k \rfloor}$ solutions to the k-sum problem with α times as much work as finding a single solution, as long as $\alpha \leq 2^{n/(\lfloor \lg k \rfloor \cdot (1 + \lfloor \lg k \rfloor))}$.

Reducing the space complexity. As we have described it so far, these algorithms require a lot of memory. Since memory is often more expensive than computing time, this may be a barrier in practice. While we have not extensively studied the memory complexity of the k-sum problem, we note that in some cases a trivial technique can greatly reduce the space complexity of our k-tree algorithm. In particular, when $k \gg 2$, we can evaluate the tree in postfix order, discarding lists when they are no longer needed. In this way, we will need storage for only about $\lg k$ lists. For example, if we take $k = 2^{\sqrt{n}-1}$, then the k-tree algorithm will run in approximately $2^{2\sqrt{n}}$ time and $\sqrt{n}2^{\sqrt{n}}$ space using this optimization, a significant improvement over naive implementations.

[1] Simply use lists $L_1, \ldots, L_4$ of size $\alpha \cdot 2^{n/3}$, and filter on $\ell' = n/3 + \lg \alpha$ bits at the lower level of the tree.

Related work. The idea of using a priority queue to generate pairwise sums $x_1 + x_2$ in sorted order (for $x_1 \in L_1$, $x_2 \in L_2$ with lists L_1, L_2 given as input) first appeared in Knuth, exercise 5.2.3-29, and was credited to W.S. Brown [20, p.158].

Later, Schroeppel and Shamir showed how to generate 4-wise sums $x_1 + \cdots + x_4$ in sorted order using a tree of priority queues [30, 31]. In particular, given 4 lists of integers and a n-bit integer c, they considered how to find all solutions to $x_1 + \cdots + x_4 = c$, and they gave an algorithm running in $\Theta(2^{n/2})$ time and $\Theta(2^{n/4})$ space when the lists are of size $\Theta(2^{n/4})$. In contrast, the problem we consider differs in four ways: we relax the problem to ask only for a single solution rather than all solutions; we allow an arbitrary number of lists; we consider other group operations; and, most importantly, our main goal in this paper is to break the $\Theta(2^{n/2})$ running time barrier. When looking for only a single solution, it is possible to beat Schroeppel and Shamir's algorithm—using Floyd's cycle-finding algorithm, distinguished points cycling algorithms [24], or parallel collision search [25], one can often achieve $\Theta(2^{n/2})$ time and $\Theta(1)$ space—but there was previously no known algorithm with running time substantially better than $2^{n/2}$. Consequently, Schroeppel and Shamir's result is not directly applicable to our problem, but their idea of using tree-based algorithms can be seen as a direct precursor of our k-tree algorithm.

Bernstein has used similar techniques in the context of enumerating solutions in the integers to equations such as $a^3 + 2b^3 + 3c^3 - 4d^3 = 0$ [3].

Boneh, Joux and Nguyen have used Schroeppel and Shamir's algorithm for solving integer knapsacks to reduce the space complexity of their birthday attacks on plain RSA and El Gamal [6]. They also used (a version of) our Theorem 3 to transform a 4-sum problem over $((\mathbb{Z}/p\mathbb{Z})^*, \times)$ to a knapsack (i.e., 4-sum) problem over $(\mathbb{Z}/q\mathbb{Z}, +)$, which allowed them to apply Schroeppel and Shamir's techniques.

Bleichenbacher used similar techniques in his attack on DSA [4].

Chose, Joux, and Mitton have independently discovered a space-efficient algorithm for finding all solutions to $x_1 \oplus \cdots \oplus x_k = 0$ and shown how to use it to speed up search for parity checks for stream cipher cryptanalysis [11]. For $k = 4$, their approach runs in $O(2^{n/2})$ time and $O(2^{n/4})$ space if $|L_1| = \cdots = |L_4| = 2^{n/4}$ and all values are n bits long, and so their scheme is in a similar class as Schroeppel and Shamir's result. Interestingly, the algorithm of Chose, et al., is essentially equivalent to repeatedly running our 4-list tree algorithm once for each possible predicted value of $\alpha = \mathrm{low}_l(x_1 \oplus x_2)$, taking $\ell = n/4$. Thus, their work is complementary to ours: their algorithm does not beat the square-root barrier, but it takes a different point in the tradeoff space, thereby reinforcing the importance of the k-sum problem to cryptography.

Joux and Lercier have used related ideas to reduce the space complexity of a birthday step in point-counting algorithms for elliptic curves [19].

Blum, Kalai, and Wasserman previously have independently discovered something closely related to the k-tree algorithm for XOR in the context of their work on learning theory [5]. In particular, they use the existence of a subexponential algorithm for the k-sum problem when k is unrestricted to find the first known subexponential algorithm for the "learning parity with noise" problem. We note that any improvement in the k-tree algorithm would immediately lead to improved algorithms for learning parity with noise, a problem that has resisted algorithmic progress for many years. Others in learning theory have since used similar ideas [37], and the hardness of this problem has even been proposed as the basis for a human-computer authentication scheme [17].

Ajtai, Kumar, and Sivakumar have used Blum, Kalai, and Wasserman's algorithm as a subroutine to speed up the shortest lattice vector problem from $2^{O(n \log n)}$ to $2^{O(n)}$ time [1].

Bellare, et al., showed that the k-sum problem over $(GF(2)^n, \oplus)$ can be solved in $O(n^3 + kn)$ time using Gaussian elimination when $k \geq n$ [2, Appendix A].

Wagner and Goldberg have shown how to efficiently find solutions to $x_1 = x_2 = \cdots = x_k$ (where $x_i \in L_i$) using parallel collision search [35]. This is an alternative way to generalize the birthday problem to higher dimensions, but the techniques do not seem to carry over to the k-sum problem.

There is also a natural connection between the k-sum problem over $(\mathbb{Z}/m\mathbb{Z}, +)$ and the subset sum problem over $\mathbb{Z}/m\mathbb{Z}$. This suggests that techniques known for the subset sum problem, such as LLL lattice reduction, may be relevant to the k-sum problem. We have not explored this direction, and we leave it to future work.

Summary. We have shown how to solve the k-sum problem (for the XOR operation) in $O(k \cdot 2^{n/(1+\lfloor \lg k \rfloor)})$ time and space, using lists of size $O(2^{n/(1+\lfloor \lg k \rfloor)})$. In particular, for $k = 4$, we can solve the 4-sum problem with complexity $O(2^{n/3})$. If k is unrestricted, we obtain a subexponential algorithm running in $O(2^{2\sqrt{n}})$ time by setting $k = 2^{\sqrt{n}-1}$.

3 Lower bounds

In this section we study how close to optimal the k-tree algorithm is. This section may be safely skipped on first reading.

Information-theoretic bounds. We can easily use information-theoretic arguments to bound the complexity of the k-sum problem as follows.

Theorem 1. *The computational complexity of the k-sum problem is $\Omega(2^{n/k})$.*

Proof. For the k-sum problem to have a solution with constant probability, we need $|L_1| \times \cdots \times |L_k| = \Omega(2^n)$, i.e., $\max_i |L_i| = \Omega(2^{n/k})$. The bound follows easily.

This bound applies to the k-sum problem over all groups.

However, this gives a rather weak bound. There is a considerable gap between the information-theoretic lower bound $\Omega(2^{n/k})$ and the constructive upper bound $O(k \cdot 2^{n/(1+\lfloor \lg k \rfloor)})$ established in the previous section. Therefore, it is natural to wonder whether this gap can be narrowed. In the general case, this seems to be a difficult question, but we show next that the lower bound can be improved in some special cases.

Relation to discrete logs. There are close connections to the discrete log problem, as shown by the following observation from Wei Dai [12].

Theorem 2 (W. Dai). *If the k-sum problem over a cyclic group $G = \langle g \rangle$ can be solved in time t, then the discrete logarithm with respect to g can be found in $O(t)$ time as well.*

Proof. We describe an algorithm for finding the discrete logarithm $\log_g y$ of a group element $y \in G$ using any algorithm for the k-sum problem in G. Each list will contain elements of the form g^w for w chosen uniformly at random. Then any solution to $x_1 \times \cdots \times x_k = y$ with $x_i \in L_i$ will yield a relation of the form $g^w = y$, where $w = w_1 + \cdots + w_k$, and this reveals the discrete log of y with respect to g, as claimed.

This immediately allows us to rule out the possibility of an efficient generic algorithm for the k-sum problem over any group G with order divisible by any large prime. Recall that a generic algorithm is one that uses only the basic group operations (multiplication, inversion, testing for equality) and ignores the representation of elements of G.

Corollary 1. *Every generic algorithm for the k-sum problem in a group G has running time $\Omega(\sqrt{p})$, where p denotes the largest prime factor of the order of G.*

Proof. Any generic algorithm for the discrete log problem in a group of prime order p has complexity $\Omega(\sqrt{p})$ [23, 33]. Now see Theorem 2.

Moreover, Theorem 2 shows that we cannot hope to find a polynomial-time algorithm for the k-sum problem over any group where the discrete log problem is hard. For example, finding a solution to $x_1 \times \cdots \times x_k \equiv 1 \pmod{p}$ is as hard as taking discrete logarithms in $(\mathbb{Z}/p\mathbb{Z})^*$, and thus we cannot expect any especially good algorithm for this problem.

The relationship to the discrete log problem goes both ways:

Theorem 3. *Suppose the discrete log problem in a multiplicative group $G = \langle g \rangle$ of order m can be solved in time t. Suppose moreover that the k-sum problem over $(\mathbb{Z}/m\mathbb{Z}, +)$ with lists of size ℓ can be solved in time t'. Then the k-sum problem over G with lists of size ℓ can be solved in time $t' + k\ell t$.*

Proof. Let $L'_i = \{\log_g x : x \in L_i\}$; then any solution to the k-sum problem over $(\mathbb{Z}/m\mathbb{Z}, +)$ with lists $L'_1, \ldots, L'_k$ yields a solution to the k-sum problem over G with lists $L_1, \ldots, L_k$.

As we have seen earlier, there exists an algorithm for solving the k-sum problem over $(\mathbb{Z}/m\mathbb{Z}, +)$ in time $t' = O(k \cdot m^{1/(1+\lfloor \lg k \rfloor)})$ so long as each list has size at least $\ell \geq t'/k$. As a consequence, there are non-trivial algorithms for solving the k-sum problem in any group where the discrete log problem is easy.

4 Attacks and applications

Blind signatures. Schnorr has recently observed that the security of several discrete-log-based blind signature schemes depends not only on the hardness of the discrete log but also on the hardness of a novel algorithmic problem, called *the ROS problem* [28]. This observation applies to Schnorr blind signatures and Okamoto-Schnorr blind signatures, particular when working over elliptic curve groups and other groups with no known subexponential algorithm for the discrete log.

We recall the ROS problem. Suppose we are working in a group of prime order q. Let $F : \{0,1\}^* \to GF(q)$ represent a cryptographic hash function, e.g., a random oracle. The goal is to find a singular $k \times k$ matrix M over $GF(q)$ satisfying two special conditions. First, the entries of the matrix should satisfy

$$M_{i,k} = F(M_{i,1}, M_{i,2}, \ldots, M_{i,k-1}) \qquad \text{for } i = 1, \ldots, k.$$

Second, there should be a vector in the kernel of M whose last component is non-zero: in other words, there should exist $v = (v_1, \ldots, v_k)^T \in GF(q)^k$ with $Mv = 0$ and $v_k = -1$.

Any algorithm to solve the ROS problem immediately leads to a one-more forgery attack using $k - 1$ parallel interactions with the signer. Previously, the best algorithm known for the ROS problem required $\Theta(q^{1/2})$ time.

We show that the ROS problem can be solved in subexponential time using our k-tree algorithm. To illustrate the idea, we first show how to solve the ROS problem in cube-root time for the case $k = 4$. Consider matrices of the following form:

$$M = \begin{bmatrix} w_1 & 0 & 0 & F(w_1, 0, 0) \\ 0 & w_2 & 0 & F(0, w_2, 0) \\ 0 & 0 & w_3 & F(0, 0, w_3) \\ w_4 & w_4 & w_4 & F(w_4, w_4, w_4) \end{bmatrix},$$

where $w_1, \ldots, w_4$ vary over $GF(q)^*$. We note that M is of the desired form if the unknowns $w_1, \ldots, w_4$ satisfy the equation

$$F(w_1, 0, 0)/w_1 + F(0, w_2, 0)/w_2 +$$

$$+ F(0, 0, w_3)/w_3 - F(w_4, w_4, w_4)/w_4 \equiv 0 \pmod{q}.$$

Thus, this can be viewed as an instance of a 4-sum problem over $GF(q)$: we fill list L_1 with candidates for the first term of the equation above, i.e., with values of the form $F(w_1, 0, 0)/w_1$, and similarly for L_2, L_3, L_4; then we search for a solution to $x_1 + \cdots + x_4 \equiv 0 \pmod{q}$ with $x_i \in L_i$. Applying our 4-list tree algorithm lets us break Schnorr and Okamoto-Schnorr blind signatures over a group of prime order q in $\Theta(q^{1/3})$ time and using 3 parallel interactions with the signer.

Of course, the above attack can be generalized to any number $k > 4$ of lists. As a concrete example, if we consider an elliptic curve group of order $q \approx 2^{160}$, then there is a one-more forgery attack using $k - 1 = 2^9 - 1$ parallel interactions,

2^{25} work, and 2^{12} space. Compare this to the conjectured 2^{80} security level that seems to be usually expected if one assumes that the best attack is to compute the discrete log using a generic algorithm. We see that the k-tree algorithm yields unexpectedly devastating attacks on these blind signature schemes.

In the general case, we obtain a signature forgery attack with subexponential complexity. If we take $k = 2^{\sqrt{\lg q}-1}$, the k-tree algorithm runs in roughly $2^{2\sqrt{\lg q}}$ time, requires $2^{\sqrt{lgq}-1}\sqrt{\lg q}$ space, and uses $2^{\sqrt{\lg q}-1}-1$ parallel interactions with the signer. Consequently, it seems that we need a group of order $q \gg 2^{1600}$ if we wish to enjoy 80-bit security. In other words, the size of the group order in bits must be an order of magnitude larger than one might otherwise expect from the best currently-known algorithms for discrete logs in elliptic curve groups.

NASD incremental hashing. One proposal for network-attached secure disks (NASD) uses the following hash function for integrity purposes [13, 14]:

$$H(x) \overset{\text{def}}{=} \sum_{i=1}^{k} h(i, x_i) \bmod 2^{256}.$$

Here x denotes a padded k-block message, $x = \langle x_1, \ldots, x_k \rangle$. We reduce inverting this hash to a k-sum problem over the additive group $(\mathbb{Z}/2^{256}\mathbb{Z}, +)$.

The inversion attack proceeds as follows. Generate k lists $L_1, \ldots, L_k$, where L_i consists of $y_i = h(i, x_i)$ with x_i ranging over many values chosen at random. Then any solution to $y_1 + \cdots + y_k \equiv c \pmod{2^{256}}$ with $y_i \in L_i$ reveals a pre-image of the digest c. If we take $k = 128$, for example, we can find a 128-block message that hashes to a desired digest using the k-tree algorithm and about 2^{40} work.

The attack can be further improved by exploiting the structure of h, which divides its one-block input x_i into two halves y_i, z_i and then computes

$$h(i, \langle y_i, z_i \rangle) \overset{\text{def}}{=} (\mathrm{SHA}(2i, y_i) \lll 96) \oplus \mathrm{SHA}(2i+1, z_i).$$

Our improved attack proceeds as follows. First, we find values $z_1, \ldots, z_k$ satisfying $\mathrm{SHA}(3, z_1) + \mathrm{SHA}(5, z_2) + \cdots + \mathrm{SHA}(2k+1, z_k) \equiv 0 \pmod{2^{96}}$. If we set $k = 128$, this can be done with about 2^{20} work using the k-tree algorithm. We fix the values $z_1, \ldots, z_k$ obtained this way, and then we search for values $y_1, \ldots, y_k$ such that $h(1, \langle y_1, z_1 \rangle) + \cdots + h(k, \langle y_k, z_k \rangle) \equiv 0 \pmod{2^{256}}$. Due to the structure of h, the left-hand side is guaranteed to be zero modulo 2^{96}, so 96 bits come for free and we have a k-sum problem over only 160 bits. The latter problem can be solved with about 2^{28} work using a second invocation of the k-tree algorithm.

This allows an adversary to find a pre-image with 2^{28} work. Similar techniques can be used to find collisions in about the same complexity. We conclude, therefore, that the NASD hash should be considered thoroughly broken.

AdHash. The NASD hash may be viewed as a special case of a general incremental hashing construction proposed by Bellare, et al., and named AdHash

[2]:

$$H(x) \overset{\text{def}}{=} \sum_{i=1}^{k} h(i, x_i) \bmod m,$$

where the modulus m is public and chosen randomly. However, Bellare, et al., give no concrete suggestions for the size of m, and so it is no surprise that some implementors have used inadequate parameters: for instance, NASD used a 256-bit modulus [13, 14], and several implementations have used a 128-bit modulus [8, 9, 32]. Our first attack on the NASD hash applies to AdHash as well, so we find that AdHash's modulus m must be very large indeed: the asymptotic complexity of the k-sum problem is as low as $O(2^{2\sqrt{\lg m}})$ if we take $k = 2^{\sqrt{\lg m}-1}$, so we obtain an attack on AdHash with complexity $O(2^{2\sqrt{\lg m}})$.

To our knowledge, this appears to be the first subexponential attack on AdHash. As a consequence of this attack, we will need to ensure that $m \gg 2^{1600}$ if we want 80-bit security. The need for such a large modulus may reduce or negate the performance advantages of AdHash.

The PCIHF hash. We next cryptanalyze the PCIHF hash construction, proposed recently for incremental hashing [15]. PCIHF hashes a padded n-block message x as follows:

$$H(x) \overset{\text{def}}{=} \sum_{i=1}^{n-1} \mathrm{SHA}(x_i, x_{i+1}) \bmod 2^{160} + 1.$$

Our attack on AdHash does not apply directly to this scheme, because the blocks cannot be varied independently: changing x_i affects two terms in the above sum. However, it is not too difficult to extend our attack on AdHash to apply to PCIHF as well.

We first show how to compute pre-images. Let us fix every other block of x, say $x_2 = x_4 = x_6 = \cdots = 0$, and vary the remaining blocks of x. Then the hash computation takes the form

$$H(x) = \sum_{j=1}^{\lfloor (n+1)/2 \rfloor} h(x_{2j-1}) \bmod 2^{160} + 1 \quad \text{where } h(w) \overset{\text{def}}{=} \mathrm{SHA}(0, w) + \mathrm{SHA}(w, 0).$$

Now we may apply the AdHash attack to this equation, and if we take $n = 255$ and apply the 128-list tree algorithm, we can find a 255-block preimage of H with about 2^{28} work.

Similarly, it is straightforward to adapt the attack to find collisions for PCIHF. The above ideas can be used to find a pair of 127-block messages that hash to the same digest, after about 2^{28} work. These results demonstrate that PCIHF is highly insecure as proposed. Though the basic idea underlying PCIHF may be sound, it seems that one must choose a much larger modulus, or some other combining operation with better resistance to subset sum attacks.

Low-weight parity checks. Let $p(x)$ be an irreducible polynomial of degree n over $GF(2)$. A number of attacks on stream cipher begin by solving an instance of the following problem:

> *The parity-check problem.* Given an irreducible polynomial $p(x)$, find a multiple $m(x)$ of $p(x)$ so that $m(x)$ has very low weight (say, 3 or 4 or 5) and so that the degree of $m(x)$ is not too large (say, 2^{32} or so).

Here, the weight of a polynomial is defined as the number of non-zero coefficients it has. Recently, there has been increased interest in finding parity-check equations of weight 4 or 5 [7, 18, 10, 22], and the cost of the precomputation for finding parity checks has been identified as a significant barrier in some cases [10]. Efficient solutions for finding low-weight multiples of $p(x)$ provide low-weight parity checks and thereby enable fast correlation attacks on stream ciphers, so stream cipher designers are understandably interested in the complexity of this problem.

We show a new algorithm for the parity-check problem that is faster on some problem instances than any previously known technique. Let $\mathbb{F} \overset{\text{def}}{=} GF(2)[t]/(p(t))$ be the finite field of size 2^n induced by $p(t)$, and let $\oplus$ denote addition in $\mathbb{F}$. We generate k lists $L_1, \ldots, L_k$, each containing values from $\mathbb{F}$ of the form $t^a \bmod p(t) \in \mathbb{F}$ where a ranges over many small integer values. Then any solution of the form $u_1 \oplus \cdots \oplus u_k = 1$ with $u_i \in L_i$, i.e., $u_i = t^{a_i} \bmod p(t) \in \mathbb{F}$, yields a non-trivial low-weight multiple $m(x) \overset{\text{def}}{=} x^{a_1} + \cdots + x^{a_k} + 1$ of $p(x)$: it is a multiple of $p(x)$ since $m(t) = t^{a_1} \oplus \cdots \oplus t^{a_k} \oplus 1 = 0$ in $\mathbb{F}$ and hence $m(x) \equiv 0 \pmod{p(x)}$, it is non-trivial since it is very unlikely to find fully repeated u_i's, and it has weight at most $k+1$. If we ensure that $a \in \{1, 2, \ldots, A\}$ for every a used in any list L_i, then $m(x)$ will also be guaranteed to have degree at most A, so we have a parity check with our desired properties. With our k-tree algorithm, we will typically need to take $A \approx 2^{n/(1+\lg k)}$ to find the first parity check.

Consequently, we obtain an algorithm to find a parity check of weight $k+1$ and degree about $2^{n/(1+\lfloor \lg k \rfloor)}$ after about $k \cdot 2^{n/(1+\lfloor \lg k \rfloor)}$ work. If we wish to obtain many parity checks, about $d^{1/(1+\lfloor \lg k \rfloor)}$ times as much work will suffice to find d parity checks, as long as $d \leq 2^{n/\lfloor \lg k \rfloor}$. This algorithm is an extension of previous techniques which used the (2-list) birthday problem [16, 21, 27, 18].

As a concrete example, if $p(x)$ represents a polynomial of degree 120, we can find a multiple $m(x)$ with degree 2^{40} and weight 5 after about 2^{42} work by using the 4-tree algorithm. Compare this to previous birthday-based techniques, which can find a multiple with degree 2^{30} and weight 5, or a multiple with degree 2^{60} and weight 3, in both cases using 2^{61} work. Thus, our k-tree algorithm runs faster than previous algorithms, but the multiples it finds have higher degrees or larger weights, so where previous techniques for finding parity-checks are computationally feasible, they are likely to be preferable. However, our algorithm may make it feasible to find non-trivial parity checks in some cases that are intractible for the previously known birthday-based methods.

Interestingly, Penzhorn and Kühn also gave a totally different cubic-time algorithm [26], using discrete logarithms in $GF(2^n)$. Their method finds a parity check with weight 4 and degree $2^{n/3}$ in $O((1+\alpha) \cdot 2^{n/3})$ time, where α represents the time to compute a discrete log in $GF(2^n)$. Using batching, they predict α will be a small constant. Also, they can obtain d times as many parity checks with about $d^{1/2}$ times as much work. Hence, when finding only a single parity check, their method improves on our algorithm: it reduces the weight from 5 to 4, while all other parameters remain comparable. However, when finding multiple parity checks, our method may be competitive. Further implementation work may be required to determine which of these algorithms performs better in practice.

5 Open problems

Other values of k. We have shown improved algorithms only for the case where k is a power of two. An open question is whether this restriction can be removed. A case of particular interest might be where $k = 3$: is there any group operation $+$ where we can find solutions to $x_1 + x_2 + x_3 = 0$ more efficiently than a naive square-root birthday search? It would also be nice to have more efficient algorithms for the case where k is large: our techniques provide only very modest improvements as k increases, yet the existence of other approaches (such as the Gaussian elimination trick of Bellare, et al. [2]) inspires hope for improvements.

Other combining operations. We can ask for other operations $+$ where the k-sum problem $x_1 + x_2 + \cdots + x_k = c$ has efficient solutions. For example, for modular addition modulo n, can we find better algorithms using lattice reduction or other methods?

Golden solutions. Suppose there is a single "golden" solution to $x_1 + \cdots + x_k = 0$ that we wish to find, hidden amongst many other useless solutions. How efficiently can we find the golden solution for various group operations? Similarly, how efficiently can we find all solutions to $x_1 + \cdots + x_k = 0$? Better answers would have implications for some attacks [6, 18].

Memory and communication complexity. We have not put much thought into optimizing the memory consumption of our algorithms. However, in practice, N bytes of memory often cost much more than N steps of computation, and so it would be nice to know whether the memory requirements of our k-tree algorithm can be reduced. Another natural question to ask is whether the algorithm can be parallelized effectively without enormous communication complexity. Over the past two decades, researchers have found clever ways (e.g., Pollard's rho, distinguished points [24], van Oorschot & Wiener's parallel collision search [25]) to dramatically reduce the memory and parallel complexity of standard (2-list) birthday algorithms, so improvements are not out of the question. In the meantime, we believe it would be prudent for cryptosystem designers to assume that such algorithmic improvements may be forthcoming: for example, in the absence of other evidence, it appears unwise to rely on the large memory consumption of our algorithms as the primary defense against k-list birthday attacks.

Lower bounds. Finally, since the security of a number of cryptosystems seems to rest on the hardness of the k-sum problem, it would be very helpful to have better lower bounds on the complexity of this problem. As it stands, the existing lower bounds are very weak when $k \gg 2$. Lacking provable lower bounds, we hope the importance of this problem will motivate researchers to search for credible conjectures regarding the true complexity of this problem.

6 Conclusions

We have introduced the k-sum problem, shown new algorithms to solve it more efficiently than previously known to be possible, and discussed several applications to cryptanalysis of various cryptosystems. We hope this will motivate further work on this topic.

Acknowledgements

I thank Wei Dai for Theorem 2. Also, I would like to gratefully acknowledge helpful comments from Dan Bernstein, Avrim Blum, Wei Dai, Shai Halevi, Nick Hopper, Claus Schnorr, Luca Trevisan, and the anonymous reviewers.

References

1. M. Ajtai, R. Kumar, D. Sivakumar, “A Sieve Algorithm for the Shortest Lattice Vector Problem,” *STOC 2001*, pp.601–610, ACM Press, 2001.
2. M. Bellare, D. Micciancio, “A New Paradigm for Collision-free Hashing: Incrementality at Reduced Cost,” *EUROCRYPT’97*, LNCS 1233, Springer-Verlag, 1997.
3. D. Bernstein, “Enumerating solutions to $p(a) + q(b) = r(c) + s(d)$,” *Math. Comp.*, 70(233):389–394, AMS, 2001.
4. D. Bleichenbacher, “On the generation of DSA one-time keys,” unpublished manuscript, Feb. 7, 2002.
5. A. Blum, A. Kalai, H. Wasserman, “Noise-Tolerant Learning, the Parity Problem, and the Statistical Query Model,” *STOC 2000*, ACM Press, 2000.
6. D. Boneh, A. Joux, P.Q. Nguyen, “Why Textbook ElGamal and RSA Encryption are Insecure,” *ASIACRYPT 2000*, LNCS 1976, Springer-Verlag, pp.30–44, 2000.
7. A. Canteaut, M. Trabbia, “Improved Fast Correlation Attacks Using Parity-Check Equations of Weight 4 and 5,” *EUROCRYPT 2000*, LNCS 1807, Springer-Verlag, pp.573–588, 2000.
8. M. Casto, B. Liskov, “Practical Byzantine Fault Tolerance,” *Proc. 3rd OSDI* (Operating Systems Design & Implementation), Usenix, Feb. 1999.
9. M. Casto, B. Liskov, “Proactive Recovery in a Byzantine-Fault-Tolerant System,” *Proc. 4th OSDI* (Operating Systems Design & Implementation), Usenix, Oct. 2000.
10. V.V. Chepyzhov, T. Johansson, B. Smeets, “A Simple Algorithm for Fast Correlation Attacks on Stream Ciphers,” *FSE 2000*, LNCS 1978, Springer-Verlag, 2001.
11. P. Chose, A. Joux, M. Mitton, “Fast Correlation Attacks: an Algorithmic Point of View,” *EUROCRYPT 2002*, LNCS 2332, Springer-Verlag, 2002.
12. W. Dai, personal communication, Aug. 1999.

13. H. Gobioff, “Security for a High Performance Commodity Storage Subsystem,” Ph.D. thesis, CS Dept., Carnegie Mellon Univ., July 1999.
14. H. Gobioff, D. Nagle, G. Gibson, “Embedded Security for Network-Attached Storage,” Tech. report CMU-CS-99-154, CS Dept., Carnegie Mellon Univ., June 1999.
15. B.-M. Goi, M.U. Siddiqi, H.-T. Chuah, “Incremental Hash Function Based on Pair Chaining & Modular Arithmetic Combining,” *INDOCRYPT 2001*, LNCS 2247, Springer-Verlag, pp.50–61, 2001.
16. J. Golić, “Computation of low-weight parity-check polynomials,” *Electronics Letters*, 32(21):1981–1982, 1996.
17. N.J. Hopper, M. Blum, “Secure Human Identification Protocols,” *ASIACRYPT 2001*, LNCS 2248, Springer-Verlag, pp.52–66, 2001.
18. T. Johansson, F. Jönsson, “Fast Correlation Attacks Through Reconstruction of Linear Polynomials,” *CRYPTO 2000*, LNCS 1880, Springer-Verlag, 2000.
19. A. Joux, R. Lercier, “‘Chinese & Match’, an alternative to Atkin’s ‘Match and Sort’ method used in the SEA algorithm,” *Math. Comp.*, 70(234):827–836, AMS, 2001.
20. D.E. Knuth, *The Art of Computer Programming*, vol 3, Addison-Wesley, 1973.
21. W. Meier, O. Staffelbach. “Fast correlation attacks on certain stream ciphers,” *J. Cryptology*, 1(3):159–167, 1989.
22. M.J. Mihalević, M.P.C. Fossorier, H. Imai, “A Low-Complexity and High-Performance Algorithm for the Fast Correlation Attack,” *FSE 2000*, LNCS 1978, Springer-Verlag, pp.196–212, 2001.
23. V.I. Nechaev, “Complexity of a determinate algorithm for the discrete logarithm,” *Math. Notes*, 55(2):165–172, 1994.
24. J.-J. Quisquater, J.-P. Delescaille, “How easy is collision search? Application to DES (Extended summary),” *EUROCRYPT’89*, LNCS 434, Springer-Verlag, pp.429–434, 1990.
25. P.C. van Oorschot, M.J. Wiener, “Parallel Collision Search with Cryptanalytic Applications,” *Journal of Cryptology*, 12(1):1–28, 1999.
26. W.T. Penzhorn, G.J. Kühn, “Computation of Low-Weight Parity Checks for Correlation Attacks on Stream Ciphers,” *Cryptography and Coding*, LNCS 1024, Springer, pp.74–83, 1995.
27. M. Salmasizadeh, J. Golic, E. Dawson, L. Simpson. “A Systematic Procedure for Applying Fast Correlation Attacks to Combiners with Memory,” *SAC’97* (Selected Areas in Cryptography).
28. C.P. Schnorr, “Security of Blind Discrete Log Signatures against Interactive Attacks,” *ICICS 2001*, LNCS 2229, Springer-Verlag, pp.1–12, 2001.
29. C.P. Schnorr, S. Vaudenay, “Black box cryptanalysis of hash networks based on multipermutations,” *EUROCRYPT’94*, LNCS 950, Springer-Verlag, 1994.
30. R. Schroeppel, A. Shamir, “A $TS^2 = O(2^n)$ Time/Space Tradeoff for Certain NP-Complete Problems,” *FOCS ’79*, pp. 328–336, 1979.
31. R. Schroeppel, A. Shamir, “A $T = O(2^{n/2}), S = O(2^{n/4})$ Algorithm for Certain NP-Complete Problems,” *SIAM J. Comput.*, 10(3):456–464, 1981.
32. L. Shrira, B. Yoder, “Trust but Check: Mutable Objects in Untrusted Cooperative Caches,” *Proc. POS8* (Persistent Object Systems), Morgan Kaufmann, pp.29–36, Sept. 1998.
33. V. Shoup, “Lower Bounds for Discrete Logarithms and Related Problems,” *EUROCRYPT’97*, LNCS 1233, Springer-Verlag, pp.256–266, 1997.
34. S. Vaudenay, “On the need for multipermutations: Cryptanalysis of MD4 and SAFER.” *FSE’94*, LNCS 1008, Springer-Verlag, pp.286–297, 1994.

35. D. Wagner, I. Goldberg, “Parallel Collision Search: Making money the old-fashioned way—the NOW as a cash cow,” unpublished report, 1997. http://www.cs.berkeley.edu/~daw/papers/kcoll97.ps
36. D. Wagner, “A Generalized Birthday Problem,” Full version at http://www.cs.berkeley.edu/~daw/papers/genbday.html.
37. K. Yang, “On Learning Correlated Functions Using Statistical Query,” *ALT’01* (12th Intl. Conf. Algorithmic Learning Theory), LNAI 2225, Springer-Verlag, 2001.
38. G. Yuval, “How to Swindle Rabin,” *Cryptologia*, 3(3):187–189, 1979.