A Generalized Birthday Problem*

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 $x_1 \oplus x_2 \oplus \cdots \oplus x_k = 0$.

* This is the full version of an extended abstract published at CRYPTO 2002.

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, how to break certain pseudorandom number generators, how to find partial collisions in KCDSA, how to speed up attacks on GHR's signature scheme, how to break a group signature scheme, how to attack certain instantiations of a method proving disjunctions in zero knowledge, how to decode a symmetric-key cryptosystem based on error-correcting codes, and how to find low-weight parity checks. 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, return-

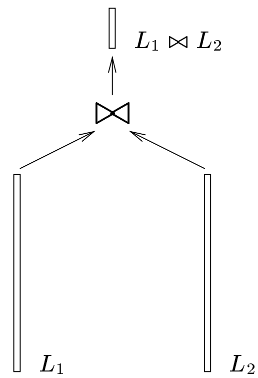

Fig. 1. An abstract representation of the standard algorithm for the (2-list) birthday problem: given two lists L_1, L_2, we use a join operation to find all pairs (x_1, x_2) such that $x_1 = x_2$ and $x_1 \in L_1$ and $x_2 \in L_2$. The thin vertical boxes represent lists, the arrows represent flow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

ing 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: 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 [37].

The birthday problem has many applications. For example, if we want to find a collision for a hash function $h : \{0,1\}^* \rightarrow \{0,1\}^n$, 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 [51].

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

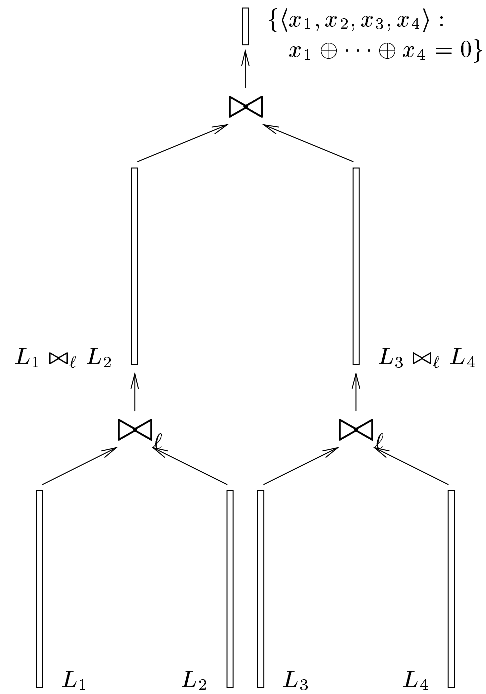

Fig. 2. A pictorial representation of our algorithm for the 4-sum problem.

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 $\mathrm{low}_\ell(x)$ denote the least significant ℓ bits of x, and define the generalized join operator $\bowtie_\ell$ 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 $\ell \stackrel{\text{def}}{=} n/3$ 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 $L'_k = L_k \oplus c \stackrel{\text{def}}{=} \{x_k \oplus c : x_k \in L_k\}$, 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 $c \stackrel{\text{def}}{=} x_{k'+1} \oplus \cdots \oplus x_k$ 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 $L_{i...j}$, 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 $[a, b] \stackrel{\text{def}}{=} \{x \in \mathbb{Z}/m\mathbb{Z} : a \le x \le b\}$ 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.

More generally, the 4-list tree algorithm will apply whenever we have a group $(G, \times)$ with an efficiently computable homomorphism $\rho : G \to G/H$ whose kernel $\ker \rho = H$ forms a normal subgroup of G of size $|H| \approx |G|^{1/3}$. In this case, we define $L^{-1} \stackrel{\text{def}}{=} \{x^{-1} : x \in L\}$ and let $L \bowtie_H L'$ represent the solutions to $x \equiv x' \pmod{H}$ with $x \in L, x' \in L'$. With this notation, we compute $L_{12} \stackrel{\text{def}}{=} L_1 \bowtie_H L_2^{-1}$, $L_{34} \stackrel{\text{def}}{=} L_3 \bowtie_H L_4^{-1}$, and $L_{12} \bowtie L_{34}^{-1}$. Note that the join operator $\bowtie_H$ can be computed efficiently using ρ.

Similarly, the k-list tree algorithm applies to group G whenever we have a chain of $\lg k$ normal subgroups $\{e\} = H_0 \subseteq H_1 \subseteq \cdots \subseteq H_{1+\lg k} = G$ such that $|H_j| \approx |G|^{j/(1+\lg k)}$ along with efficiently computable homomorphisms $\rho_j : G \to G/H_j$ with kernel $\ker \rho_j = H_j$ for each j.

Non-group operations. Interestingly, our algorithm can even be generalized to non-group operations in some special cases. Let V, W be vector spaces with $|W| \ge |V|$, and suppose $f : V \times V \to V$ is a bi-affine map, i.e., for every constant $a \in V$ both $f(a, \cdot)$ and $f(\cdot, a)$ should be affine. (When the latter two maps are both bijective, this is a special case of a $(2,1)$-multipermutation, also known as a latin square [47].) For this case, we can compute the generalized join $L \bowtie_f L'$, which finds all solutions to $f(x,x')=0$, as follows: set $h(x') \stackrel{\text{def}}{=} g_{x'}^{-1}(0)$, where $g_{x'}(x) \stackrel{\text{def}}{=} f(x,x')$, and compute $L \bowtie h(L')$. Note that h can usually be computed efficiently, e.g., in $O((\dim V)^3)$ time using Gaussian elimination when $g_{x'}$ is injective.

This idea can be used, in some (but not all) cases, to find solutions to $f(x_1,\ldots,x_k)=0$ where $f: V^k \to W$ is a multi-affine map. For instance, one tractable case is $f(x_1,x_2,x_3,x_4)=g(x_1,x_2)+h(x_3,x_4)$ where $g,h: V^2 \to W$ are surjective bi-affine maps with $\dim W \gg 1$ and $\dim V \geq \frac{1}{3}\dim W$. In this case, we can find a solution to $f(x_1,\ldots,x_4)=0$ in approximately $O(|W|^{1/3})$ time: pick an affine map $\rho: W \to W'$ with $\dim\ker\rho \approx \frac{1}{3}\dim W$, and then compute $g(L_1 \bowtie_{\rho\circ g} L_2) \bowtie -h(L_3 \bowtie_{\rho\circ h} L_4)$.

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 $\alpha \leq 2^{n/(\lfloor\lg k\rfloor\cdot(1+\lfloor\lg k\rfloor))}$.

Subexponential time algorithms. Note that when $k=2^{\sqrt{n}}$, the k-list tree algorithm runs in $O(2^{2\sqrt{n}})$ time, thus providing a subexponential time algorithm for the k-sum problem for many combining operations.

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.

The case of XOR*: Gaussian elimination.* Bellare, et al., have noted that there is an efficient algorithm for the k-sum problem with XOR as the binary operation, when k is sufficiently large [4, Appendix A]. In particular, if $L_i \subseteq GF(2)^n$ and $k \geq n$, then we can easily solve the equation $x_1 \oplus \cdots \oplus x_k = 0$ for $x_i \in L_i$.

We recall the algorithm here. Assume each list has only two elements, $L_i = \{w_{i,0}, w_{i,1}\}$ (any other elements can be ignored). The main observation is that the equation

$$x_1 \oplus \cdots \oplus x_k = 0 \qquad x_i \in L_i$$

[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.

can be transformed into a linear equation. Applying the substitutions $x_i = b_i \cdot \alpha_i$, $\alpha_i = w_{i,0} \oplus w_{i,1}$, and $\beta = w_{1,0} \oplus \cdots \oplus w_{k,0}$ yields the system

$$(b_1 \cdot \alpha_1) \oplus \cdots \oplus (b_k \cdot \alpha_k) = \beta \qquad b_i \in GF(2)$$

where $\alpha_1, \ldots, \alpha_k, \beta$ are specified constants and we wish to solve for the unknowns $b_1, \ldots, b_k$. The latter system can be written in the form $Mb = \beta$, where M is the $n \times k$ boolean matrix whose i-th row is α_i. When $k \geq n$, the equation $Mb = \beta$ will have a solution for b with high probability, and then b can be recovered using Gaussian elimination. (If it does not have a solution, we can throw away our lists and try again with new elements.) Hence, there is an algorithm for the k-sum problem over $GF(2)^n$ with $O(n^3 + kn)$ complexity when $k \geq n$.

A hybrid algorithm for XOR. When considering the XOR operation, the Gaussian elimination algorithm given above can be combined with our new tree-based algorithm to obtain a hybrid algorithm that is faster than either individual algorithm, for some parameter choices. The algorithm runs in $O((n-k+k')nk'2^{(n-k+k')/(1+\lfloor \lg k' \rfloor)} + n(k-k')^2)$ time, where $0 < k' < k$ is a parameter that may be freely chosen to optimize the running time. For instance, with $n = 600$ and $k = 512$, we can solve a k-sum problem over $GF(2)^n$ in about 2^{41} time by choosing $k' = 16$. Compare to the plain k-tree algorithm, which would need 2^{69} work, or to the Gaussian elimination algorithm, which would have negligible success probability.

The idea is as follows. Choose k' with $0 < k' < k$ and consider the equation

$$x_1 \oplus \cdots \oplus x_{k'} = x_{k'+1} \oplus \cdots \oplus x_k \qquad x_i \in L_i.$$

We assume that lists $L_1, \ldots, L_{k'}$ are very large, and the remaining lists have only two elements (the rest are discarded). Then the right-hand side can be viewed as a linear form:

$$x_1 \oplus \cdots \oplus x_{k'} = Mb \oplus \beta \qquad x_i \in L_i, b \in GF(2)^{k-k'}.$$

Since M is a $n \times (k-k')$ boolean matrix, there is a $(n-k+k') \times n$ matrix H so that $Hy = 0$ if and only if y is in the image of M, i.e., if and only if there exists x so that $Mx = y$. H is sometimes known as the parity check matrix corresponding to M and is defined by the requirement that it have maximal rank subject to $HM = 0$. Now multiplying both sides of the previous equation by H yields the new system

$$Hx_1 \oplus \cdots \oplus Hx_{k'} = H\beta \qquad x_i \in L_i.$$

Applying the substitutions $x'_i = Hx_i$, $L'_i = \{Hx : x \in L_i\}$, and $\beta' = H\beta$ gives

$$x'_1 \oplus \cdots \oplus x'_{k'} = \beta' \qquad x'_i \in L'_i,$$

which can now be solved with the k'-tree algorithm. Finding $x'_1, \ldots, x'_{k'}$ will take $O(k'2^{(n-k+k')/(1+\lfloor \lg k' \rfloor)})$ time. Also, finding H takes $O(n(k-k')^2))$ time if we use Gaussian elimination, and applying the substitutions takes $O((n-k+k')nk'2^{(n-k+k')/(1+\lfloor \lg k' \rfloor)})$ time, which yields the claimed time bound.

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 [32, 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 [43, 44]. 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 [36], or parallel collision search [37], 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 $a^3 + 2b^3 + 3c^3 - 4d^3 = 0$ [5].

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 [8]. 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 [6].

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 [16]. 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 $\alpha = \text{low}_l(x_1 \oplus x_2)$, 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 [30].

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 [7]. 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 [50], and the hardness of this problem has even been proposed as the basis for a human-computer authentication scheme [26].

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 [2].

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 $k \geq n$ [4, 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 [48]. 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.

After the initial publication of our work, we discovered earlier work by Camion and Patarin [10] on the following problem: Given a 128-bit integer b and a set of 256 120-bit integers a_i, find a subset of a_i's that sums to b. They showed how to solve this problem in about 2^{32} operations using what amounts to a 4-tree algorithm, and then they employed this idea to attack a knapsack-based hash function. This represents significant prior art, and we believe Camion and Patarin should receive credit for the idea of using tree-based algorithms to solve k-sum problems faster than square-root time. In this paper, we have extended their work by considering the general case of the k-sum problem for arbitrary groups, by giving the asymptotic complexity of the k-tree algorithm, and by introducing a number of new applications to cryptanalysis.

Summary. We have shown how to solve the k-sum problem (for certain operations) 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 [19].

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 $\Omega(\sqrt{p})$ [35, 46]. 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* [41]. This observation applies to Schnorr blind signatures and Okamoto-Schnorr blind signatures, especially 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 $2^{\sqrt{lg q} - 1}\sqrt{\lg q}$ 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 [21, 22]:

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

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

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

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 [4]:

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

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 [21, 22], and several implementations have used a 128-bit modulus [13, 14, 45]. 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 [23]. PCIHF hashes a padded n-block message x as follows:

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

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

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

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 $[11, 29, 15, 34]$, and the cost of the precomputation for finding parity checks has been identified as a significant barrier in some cases [15]. 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 may be faster on some problem instances than any previously known technique. Let $\mathbb{F} \stackrel{\text{def}}{=} GF(2)[t]/(p(t))$ 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 $m(x) \stackrel{\text{def}}{=} x^{a_1} + \cdots + x^{a_k} + 1$ 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 $[24, 33, 40, 29]$.

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 cube-root-time algorithm [38], 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.

Pseudorandom number generation. A critical component of most cryptographic software is a scheme for generating pseudorandom values, typically by expanding some short seed to many outputs suitable for use as nonces or session keys. Careful implementations typically mix additional entropy into the secret state of the generator, for example by sampling system state, the arrival times of network packets, interactions with the user, and so on [25]. However, this introduces the possibility of chosen-input attacks [31, 25], and we discuss how the k-tree algorithm may be used to give new attacks in this setting.

The BSAFE PRNG works as follows: to mix an entropy input x into the state s_i, we set $s_{i+1} = s_i + H(x) \bmod 2^{128}$ (here H is the MD5 hash); to generate an output z from state s_i, we set $z = H(s_i)$ and update the state via $s_{i+1} = s_i + 1 \bmod 2^{128}$. Kelsey, et al., have noted that if we can find an input x so that $H(x) \equiv -1 \pmod{2^{128}}$, then we cause the generator to repeat an output, and more generally, if we can find x, x' so that $H(x) + H(x') \equiv -1 \pmod{2^{128}}$, the same is possible [31]. However, their attack has complexity 2^{64}, and thus is only barely feasible.

We note that the k-tree algorithm provides an efficient chosen-input attack on the BSAFE PRNG: find $x_1, \ldots, x_k$ so that $H(x_1)+\cdots+H(x_k) \equiv -1 \pmod{2^{128}}$, and use these values as chosen inputs to the PRNG. For $k = 8$, this requires just 2^{35} offline hash computations and 8 chosen inputs to break the PRNG. Time-travel attacks [31], where the PRNG is sent back t steps in time, can also be easily mounted by finding a solution to $H(x_1) + \cdots + H(x_k) \equiv -t \pmod{2^{128}}$. Moreover, in practice only very weak assumptions on our ability to control inputs are necessary: for instance, if for each input x_i the attacker can choose from one of two predictable possibilities, then with 256 chosen inputs the attacker can force the PRNG to repeat. Since many cryptosystems (e.g., DSA) become insecure or leak the key when nonces are repeated, we feel this gives a good reason to avoid the BSAFE generator in favor of some other PRNG with stronger resistance to chosen-input attacks.

An unusual property of KCDSA. KCDSA is a digital signature algorithm proposed as a Korean standard [12]. The signature on a message m is of the form $\langle r, s\rangle$. We show a peculiar property of KCDSA: It is feasible for the signer to find two messages m, m' whose signatures partially collide, i.e., we can efficiently find message m and m' whose signatures $\langle r, s\rangle$ and $\langle r', s'\rangle$ satisfy $s = s'$. We are not aware of any other signature scheme with such a property.

Although this property might at first glance appear to have implications for the non-repudiation properties of KCDSA, there does not seem to be any obvious way to exploit it in an attack. We stress that we are not aware of any way to use this property to break KCDSA, nor does it contradict the security claims of the designers. Nonetheless, since this property appears to be unusual for a signature algorithm, we explain it here to further our understanding of KCDSA, and we leave it as an open problem whether this property is of any practical relevance.

We recall the KCDSA signature algorithm. The public key is $g^x \pmod p$, along with the parameters p, q, g, z. To sign a message m, the signer picks a random nonce $k \in (\mathbb{Z}/q\mathbb{Z})^*$ and uses this to generate a signature $\langle r, s\rangle$ given by

$$r = h(g^k \bmod p), \qquad s = x \cdot f(k, m) \bmod q$$
$$\text{where } f(k, m) \stackrel{\text{def}}{=} k - (h(g^k \bmod p) \oplus h(z||m)) \bmod q.$$

We show how to find k, m, k', m' so that $f(k, m) = f(k', m')$. This will yield a collision for the only part of the signature that depends on the private key x.

Collisions for f can be found using our 4-tree algorithm. First, we find many pairs (k, m) satisfying $k \oplus h(g^k \bmod p) \equiv h(z||m) \pmod{2^{54}}$, using the (2-list) birthday paradox. After $2^{55.25}$ work, we can expect to find about $2^{54.5}$ such pairs. Note that for each such pair we have $k \equiv h(g^k \bmod p) \oplus h(z||m) \pmod{2^{54}}$, and as long as there are no carries, we will also have $f(k, m) \equiv 0 \pmod{2^{54}}$. Thus, we can expect at least half, or about $2^{53.5}$, of our pairs to satisfy $f(k, m) \equiv 0 \pmod{2^{54}}$. Then, in a second phase we try to combine the pairs from first phase to find a quadruple $(k, m), (k', m')$ such that $f(k, m) = f(k', m')$. Note that for any quadruple, the latter holds with probability about 2^{-106} (since we only need the top 106 bits to match; the bottom 54 bits come for free), and with $2^{53.5} \times 2^{53.5}/2 \approx 2^{106}$ pairs we expect to have a good chance of finding one such collision in f. The attack has total complexity about $2^{55.6}$ (much faster than a naive 2^{80} birthday attack), and thereafter the signer can use k, k', m, m' to generate signatures $\langle r, s\rangle$ and $\langle r', s'\rangle$ satisfying $s = s'$.

Improved attacks on the GHR signature scheme. Coron and Naccache introduced attacks on the Gennaro-Halevi-Rabin signature scheme that work by looking for smooth hash digests [17], and they then showed how to speed up the attack by a factor of 4 by exploiting properties of the GHR hash function. We show how to further speed up their attack by another factor of 16 by using k-sums.

The GHR hash function is

$$H(m, r) \stackrel{\text{def}}{=} 1||f_1(m, r_1)||f_2(m, r_2)||f_3(m, r_3)||f_4(m, r_4)||1$$
$$\text{where } f_i(m, r_i) \stackrel{\text{def}}{=} \text{SHA}(m||i||r_i).$$

We suggest to fix a message m, pick an arbitrary 120-bit odd integer n, and search for random nonces $r_1, \ldots, r_4$ so that $H(m, r) \equiv 0 \pmod{n}$; such hash digests all share a common 120-bit divisor n, and the remainder $H(m, r)/n$ will have an increased chance of being smooth. We note that finding such nonces can be cast as a 4-sum problem: write

$$2^{480} f_1(m, r_1) + 2^{320} f_2(m, r_2) + 2^{160} f_3(m, r_3) + f_4(m, r_4) \equiv -2^{640} - 2^{-1} \pmod{n},$$

and let list L_i contain candidates for the i-th term above, i.e., values of the form $2^{160i} \cdot f_i(m, r_i) \pmod{n}$ where r_i ranges over many possibilities. In this way, we can find 2^{60} hash digests having n as a factor with 2^{62} hash computations and similar workfactor by using the 4-tree algorithm. Then the Coron-Naccache algorithm will find a division collision for H with about 2^{62} work [17, Table 1], since we are essentially invoking their algorithm as a black box on random $640 - 120 = 520$ bit integers. We obtain an algorithm using 2^{63} work and 2^{61} space, which is a 16-fold speedup over Coron-Naccache's improved attack. We expect that the space overhead can be drastically reduced using Shamir and Schroeppel's techniques [43, 44].

Group signatures. Back has previously proposed a simple construction for group signatures [3]. Let Alice and Bob have RSA public keys (n_A, e_A) and (n_B, e_B) respectively. In Back's scheme, we accept (w_A, w_B) as a valid signature on message M just if $(w_A^{e_A} \bmod n_A) \oplus (w_B^{e_B} \bmod n_B) = H(M)$. Note that either Alice or Bob can create signatures acceptable in this way on their own, and noone can tell who has signed the message.

There is a natural generalization of this scheme to groups of $k \geq 2$ participants (though Back did not suggest it). Unfortunately, the scheme is insecure if signatures by large groups are allowed. In particular, outsiders can forge a valid signature by solving a k-sum problem: let L_i contain values of the form $x_i = w_i^{e_{A_i}} \bmod n_{A_i}$, where w_i varies over many possibilities; then any solution to $x_1 \oplus \cdots \oplus x_k = H(M)$ will give a signature that appears to have been created by one of $A_1, \ldots, A_k$. Thus, the scheme is totally insecure with $k \geq \max_i \lg n_i$ participants. Simply replacing XOR with modular addition will not fix the problem, either, as there will remain subexponential attacks that are barely feasible for typical modulus sizes. The attack on XOR is what led Rivest, et al., to use a more complicated combining function in their work on ring signatures [39], and our attack on modular addition gives further motivation along these lines.

Zero-knowledge proofs of disjunctions. Let the predicate φ be a disjunction of predicates $\varphi_1, \ldots, \varphi_k$, i.e., $\varphi = \varphi_1 \vee \cdots \vee \varphi_k$. There is a standard trick for constructing a zero-knowledge proof of φ from zero-knowledge proofs for $\varphi_1, \ldots, \varphi_k$. We show that this trick can be endangered by the k-tree algorithm in certain special cases.

We recall the trick. An interactive three-round zero-knowledge proof system takes the form

1. $P \to V : m$
2. $V \to P : c$
3. $P \to V : r$

where P denotes the prover and V the verifier; afterwards, V verifies some validity predicate $v(m, c, r) = 1$. Recall that in a zero-knowledge system, even outsiders can generate a fake transcript that is computationally indistinguishable from a real interaction. The interactive proof of φ then contains k parallel runs of the proof system for φ_i, but linked together cleverly:

1. $P \to V : m_1, \ldots, m_k$
2. $V \to P : c$
3. $P \to V : c_1, \ldots, c_k, r_1, \ldots, r_k$.

Afterwards, V verifies that $v_1(m_1, c_1, r_1) = \cdots = v_k(m_k, c_k, r_k) = 1$ and $c_1 \oplus \cdots \oplus c_k = c$. The prover's strategy for convincing the verifier is as follows. If $\varphi = \varphi_1 \vee \cdots \vee \varphi_k$ is true, then there must be some $t \in \{1, \ldots, k\}$ so that φ_t is true. The prover first uses the simulator to generate a fake transcript (m_i, c_i, r_i) for each $i \neq t$, then begins a legitimate run of the proof system for φ_t to get the first message m_t, and sends $m_1, \ldots, m_k$ to V. When V replies with a challenge c, P computes $c_t = c \oplus c_1 \oplus \cdots \oplus c_{t-1} \oplus c_{t+1} \oplus \cdots \oplus c_k$, hands c_t to the proof system for φ_t to get a response r_t, and transmits $c_1, \ldots, c_k, r_1, \ldots, r_k$ to V. We can recognize a special case of this general technique embedded within Back's group signature scheme, and our attack will be a generalization of the attack against Back's scheme given earlier.

Cramer, et al., have shown that the above trick for disjunctions is sound when the underlying proof system for φ_i is a honest-verifier zero-knowledge proof of knowledge satisfying the special soundness property [18]. The special soundness property requires that any two proof transcripts $(m_i, c_i, r_i), (m_i, c_i', r_i')$ for φ_i with $c_i \neq c_i'$ reveal the underlying witness. Cramer, et al., posed the following open problem: Is special soundness needed? We give evidence that the answer may be "Yes," as schemes that don't possess special soundness may be susceptible to a k-sum attack.

Suppose that the proof system for each φ_i has the following convenient property: given any m_i, there is a simulator that can generate many fake transcripts $(m_i, c_{i,j}, r_{i,j})$ such that $v_i(m_i, c_{i,j}, r_{i,j}) = 1$. Consider the following attack, where we attempt to fool the verifier V into thinking that φ is true when in fact it is false. First, choose and fix $m_1, \ldots, m_k$, and send them to V. Await for V's challenge c. Next let list L_i contain many transcripts (m_i, c_i, r_i) where c_i, r_i vary over many possibilities, with L_i generated by running the simulator mentioned above. Then, we look for solutions to $c_1 \oplus \cdots \oplus c_k = c$ such that $(\cdot, c_i, \cdot) \in L_i$, and we send the corresponding c_i, r_i values to V. Note that the latter problem is an instance of the k-sum problem over XOR and hence has a polynomial-time algorithm once k exceeds the bit-length of c [4].

Hence, we obtain a polynomial-time attack on the soundness of the proof system for φ, whenever the proof systems for φ_i satisfy our convenient property. Of course, any proof of knowledge satisfying special soundness is guaranteed not to satisfy our convenient property. However, if we consider zero-knowledge proofs of knowledge without special soundness, it seems plausible that our convenient property is compatible with the requirement that there exist a knowledge extractor: a knowledge extractor must be able to recover the witness in a chosen-challenge attack, whereas we require only that the witness remain hidden against a known-challenge attack.

The disjunction trick has been used for deniable signatures [27], privacy-sensitive certificates [9], and elsewhere [20]. We believe the k-sum attack gives new insight on the limitations of the disjunction trick[2].

Encryption based on error-correcting codes. In 1990, Hwang and Chen proposed a symmetric-key encryption system based on error-correcting codes [28]. They noted that the system is susceptible to a partial attack based on the birthday paradox. They argued that this attack needs $2^{n/2}$ ciphertexts and 2^n work (here n is the length of codewords), and so they picked parameters to render this attack infeasible. We note that their attack can be sped up with the method of k-sums. Our improved attack needs only $O(n)$ ciphertexts and $O(n^3)$ time, dramatically improving their analysis. We are not aware of any practical systems that use Hwang-Chen encryption; instead, we present the improved cryptanalysis only as an example of how birthday attacks can sometimes be improved using k-sums.

We recall first the Hwang-Chen cryptosystem. The key consists of a $k \times n$ matrix G for a (n, k) binary code with minimum distance t, a $n \times n$ permutation matrix P, and a nonlinear function $f : GF(2^k) \to GF(2^k)$. To encrypt a k-bit message m, we pick a random n-bit vector r and transmit r along with the ciphertext

$$c = f(m \oplus rP^*)G \oplus rP,$$

where rP^* denotes the first k bits of the n-bit quantity rP. The attack works by noting that if we observe some pair of ciphertexts (r_1, c_1) and (r_2, c_2) satisfying $r_1 = r_2$, then $c_1 \oplus c_2 = [f(m_1 \oplus r_1P^*) \oplus f(m_2 \oplus r_2P^*)]G$, hence the n-bit quantity $c_1 \oplus c_2$ is a non-trivial codeword of G if $m_1 \neq m_2$. With $\sqrt{n}2^{(n+1)/2}$ ciphertexts, we expect to find about n pairs like this, and from them we can derive a combinatorially equivalent code $\hat{G}$. Alabbadi has argued that in some cases this may allow to break the cryptosystem [1].

The improved attack follows from a natural generalization of their observation. In particular, if we observe q ciphertexts $(r_1, c_1), \ldots, (r_q, c_q)$ satisfying $r_1 \oplus \cdots \oplus r_q = 0$, then $c_1 \oplus \cdots \oplus c_q$ is a codeword of G. Of course, if we have a supply of 2 candidate ciphertexts $L_i = \{(r_{i,1}, c_{i,1}), (r_{i,2}, c_{i,2})\}$ for each of the q positions, we can phrase this as a q-sum problem with lists of size 2. By the observations of Bellare, et al., this has a solution that can be found efficiently with linear algebra if $q \geq n$ [4]. Hence with $2n$ ciphertexts and $O(n^3)$ work we can find a codeword of G, and with just a bit more we can find n independent codewords (simply take $q = n + \lg n$ and collect $2q = O(n)$ ciphertexts). This reveals a combinatorially equivalent code $\hat{G}$.

Of course, the full power of k-sums was not necessary to find this attack. It is clear that—once discovered—the attack may be described without reference

[2] For instance, our observations show immediately that the proof of Prop. 3.5.2(a) in [9] is lacking: the proof sketch never requires anywhere that the underlying proof system for φ_i satisfies special soundness, and thus proves too much.

to k-sums, instead using only linear algebra. Nonetheless, k-sums were a convenient analytic framework that made it possible for us to find the attack in the first place. Recognizing the relevance of linear algebra arguably requires some insight (and apparently the linear algebra attack previously escaped notice), but checking whether the known birthday attack can be extended to a k-sum attack is often straightforward. In general, a good rule of thumb may be "Whenever there is a birthday-style attack, check whether the method of k-sums applies."

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. [4]) inspires hope for improvements.

Other combining operations. We can ask for other group 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? It may also be natural to consider larger classes of associative binary operations. One might even ask when there is an efficient algorithm to find solutions to $f(x_1, \ldots, x_k) = c$: we have shown that this can be solved when f is linear, but what about the case where f is 1-resilient (i.e., a $(k, 1)$-multipermutation), tree-structured, or linear in each of its arguments, for instance? Progress on these questions might lead to improved attacks on various ciphers.

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 [8, 29].

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 [36], van Oorschot & Wiener's parallel collision search [37]) 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 evidence to the contrary, 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. Finally, I thank Serge Vaudany for pointing me to Camion and Patarin's work [10].

References

1. M.M. Alabbadi, "Security Comments on the Hwang-Chen Algebraic-code Cryptosystem," *ICIS '97*, LNCS 1334, Springer-Verlag, 1997.
2. M. Ajtai, R. Kumar, D. Sivakumar, "A Sieve Algorithm for the Shortest Lattice Vector Problem," *STOC 2001* (Proc. 31st Symp. Theory of Computing), pp.601–610, ACM Press, 2001.
3. A. Back, "Non-transferable signatures," `ietf-open-pgp` mailing list, Sept. 24, 1997. http://www.imc.org/ietf-openpgp/mail-archive/msg00189.html
4. M. Bellare, D. Micciancio, "A New Paradigm for Collision-free Hashing: Incrementality at Reduced Cost," *EUROCRYPT'97*, LNCS 1233, Springer-Verlag, 1997.
5. D. Bernstein, "Enumerating solutions to $p(a) + q(b) = r(c) + s(d)$," *Math. Comp.*, 70(233):389–394, AMS, 2001.
6. D. Bleichenbacher, "On the generation of DSA one-time keys," unpublished manuscript, Feb. 7, 2002.
7. A. Blum, A. Kalai, H. Wasserman, "Noise-Tolerant Learning, the Parity Problem, and the Statistical Query Model," *STOC 2000* (Proc. 32nd Symp. Theory of Computing), ACM Press, 2000.

8. 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.
9. S. Brands, *Rethinking Public-Key Infrastructure and Digital Certificates—Building in Privacy*, Ph.D. thesis, Sept. 4, 1999.
10. P. Camion, J. Patarin, "The Knapsack Hash Function proposed at Crypto'89 can be broken," *EUROCRYPT '91*, LNCS 457, Springer-Verlag, pp.39–53, 1991.
11. 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.
12. C.-H. Lim, P.-J. Lee, "A study on the proposed Korean Digital Signature Algorithm," *ASIACRYPT'98*, LNCS 1514, Springer-Verlag, pp.175–186, 1998.
13. M. Casto, B. Liskov, "Practical Byzantine Fault Tolerance," *Proc. 3rd OSDI* (Operating Systems Design & Implementation), Usenix, Feb. 1999.
14. M. Casto, B. Liskov, "Proactive Recovery in a Byzantine-Fault-Tolerant System," *Proc. 4th OSDI* (Operating Systems Design & Implementation), Usenix, Oct. 2000.
15. V.V. Chepyzhov, T. Johansson, B. Smeets, "A Simple Algorithm for Fast Correlation Attacks on Stream Ciphers," *FSE 2000*, LNCS 1978, Springer-Verlag, pp.181–195, 2001.
16. P. Chose, A. Joux, M. Mitton, "Fast Correlation Attacks: an Algorithmic Point of View," *EUROCRYPT 2002*, LNCS 2332, Springer-Verlag, 2002.
17. J.-S. Coron, D. Naccache, "Security Analysis of the Gennaro-Halevi-Rabin Signature Scheme," *EUROCRYPT 2000*, LNCS 1807, Springer-Verlag pp.91–101.
18. R. Cramer, I. Damgard, B. Schoenmakers, "Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols," *CRYPTO '94*, LNCS 893, Springer-Verlag, pp.174–187, 1995.
19. W. Dai, personal communication, Aug. 1999.
20. A. De Santis, G. Di Crescenzo, G. Persiano, M. Yung, "On Monotone Formula Closure of SZK," *STOC '94* (Proc. 24th Symp. Theory of Computing), pp.454–465, ACM Press, 1994.
21. H. Gobioff, "Security for a High Performance Commodity Storage Subsystem," Ph.D. thesis, CS Dept., Carnegie Mellon Univ., July 1999.
22. 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.
23. 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.
24. J. Golić, "Computation of low-weight parity-check polynomials," *Electronics Letters*, 32(21):1981–1982, 1996.
25. P. Gutmann, "Software Generation of Practically Strong Random Numbers," *Usenix Security '98*, 1998.
26. N.J. Hopper, M. Blum, "Secure Human Identification Protocols," *ASIACRYPT 2001*, LNCS 2248, Springer-Verlag, pp.52–66, 2001.
27. B.A. Huberman, M. Franklin, T. Hogg, "Enhancing Privacy and Trust in Electronic Communications," *ACM Conference on Electronic Commerce*, pp.78–86, 1999.
28. T. Hwang, Y. Chen, "Algebraic-code cryptosystem using random code chaining," *IEEE TENCON'90* (1990 IEEE Region 10 Conference on Computer & Communication Systems), vol. 1, pp.194–196, IEEE, 1990.
29. T. Johansson, F. Jönsson, "Fast Correlation Attacks Through Reconstruction of Linear Polynomials," *CRYPTO 2000*, LNCS 1880, Springer-Verlag, pp.300–315, 2000.

30. 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.
31. J. Kelsey, B. Schneier, D. Wagner, C. Hall. "Cryptanalytic Attacks on Pseudorandom Number Generators," *FSE '98*, LNCS 1372, Springer-Verlag, 1998.
32. D.E. Knuth, *The Art of Computer Programming*, vol 3, Addison-Wesley, 1973.
33. W. Meier, O. Staffelbach. "Fast correlation attacks on certain stream ciphers," *J. Cryptology*, 1(3):159–167, 1989.
34. M.J. Mihaljević, 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.
35. V.I. Nechaev, "Complexity of a determinate algorithm for the discrete logarithm," *Math. Notes*, 55(2):165–172, 1994.
36. 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.
37. P.C. van Oorschot, M.J. Wiener, "Parallel Collision Search with Cryptanalytic Applications," *Journal of Cryptology*, 12(1):1–28, 1999.
38. 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.
39. R.L. Rivest, A. Shamir, Y. Tauman, "How to Leak a Secret," *ASIACRYPT 2001*, LNCS 2248, Springer-Verlag, pp.552–565, 2001.
40. 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).
41. C.P. Schnorr, "Security of Blind Discrete Log Signatures against Interactive Attacks," *ICICS 2001*, LNCS 2229, Springer-Verlag, pp.1–12, 2001.
42. C.P. Schnorr, S. Vaudenay, "Black box cryptanalysis of hash networks based on multipermutations," *EUROCRYPT'94*, LNCS 950, Springer-Verlag, pp.47–57, 1994.
43. R. Schroeppel, A. Shamir, "A $TS^2 = O(2^n)$ Time/Space Tradeoff for Certain NP-Complete Problems," *FOCS '79* (Proc. 20th IEEE Symposium on Foundations of Computer Science), pp. 328–336, 1979.
44. 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.
45. 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.
46. V. Shoup, "Lower Bounds for Discrete Logarithms and Related Problems," *EUROCRYPT'97*, LNCS 1233, Springer-Verlag, pp.256–266, 1997.
47. S. Vaudenay, "On the need for multipermutations: Cryptanalysis of MD4 and SAFER." *FSE'94*, LNCS 1008, Springer-Verlag, pp.286–297, 1994.
48. 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
49. D. Wagner, "A Generalized Birthday Problem," Full version at http://www.cs.berkeley.edu/~daw/papers/genbday.html.
50. K. Yang, "On Learning Correlated Functions Using Statistical Query," *ALT'01* (12th Intl. Conf. Algorithmic Learning Theory), LNAI 2225, Springer-Verlag, 2001.
51. G. Yuval, "How to Swindle Rabin," *Cryptologia*, 3(3):187–189, 1979.