A Generalized Birthday Problem

David Wagner

U.C. Berkeley

daw@cs.berkeley.edu

The Birthday Paradox

I'm throwing a party of unbounded size,
on a planet with 2^n days in a year.

Problem: How much work will it take to find some pair of party-goers whose birthdays XOR to zero?

The Birthday Paradox

I'm throwing a party of unbounded size,
on a planet with 2^n days in a year.

Problem: How much work will it take to find some pair of party-goers whose birthdays XOR to zero?

Complexity: $\Theta(2^{n/2})$ [Yuv79].

- Need $\Omega(2^{n/2})$ attendees before some such pair is likely to exist.
- Once such a pair exists, it can be found explicitly in $O(2^{n/2})$ time by sorting or hashing.

The Generalized Birthday Paradox

Problem: How much work will it take to find some set of k party-goers whose birthdays XOR to zero?

The Generalized Birthday Paradox

Problem: How much work will it take to find some set of k party-goers whose birthdays XOR to zero?

Complexity: ???.

The Generalized Birthday Paradox

Problem: How much work will it take to find some set of k party-goers whose birthdays XOR to zero?

Complexity: ???.

- Need $\Omega(2^{n/k})$ attendees before some such k-set is likely to exist.

The Generalized Birthday Paradox

Problem: How much work will it take to find some set of k party-goers whose birthdays XOR to zero?

Complexity: ???.

- Need $\Omega(2^{n/k})$ attendees before some such k-set is likely to exist.
- However: the best algorithm previously known for explicitly finding such a k-set needs $O(2^{n/2})$ time.

The Generalized Birthday Paradox

Problem: How much work will it take to find some set of k party-goers whose birthdays XOR to zero?

Complexity: ???.

- Need $\Omega(2^{n/k})$ attendees before some such k-set is likely to exist.
- However: the best algorithm previously known for explicitly finding such a k-set needs $O(2^{n/2})$ time.

In this talk: There is an algorithm for $k = 4$ with complexity $O(2^{n/3})$.

Organization

- The k-sum problem
- A new tree-based algorithm for the k-sum problem
- Applications to cryptanalysis
  - e.g., a subexponential attack on (Okamoto-)Schnorr blind signatures in elliptic curve groups

The k-sum Problem

Parameters: k; a set S of size $|S| \approx 2^n$; and a binary operator $+$ on S.

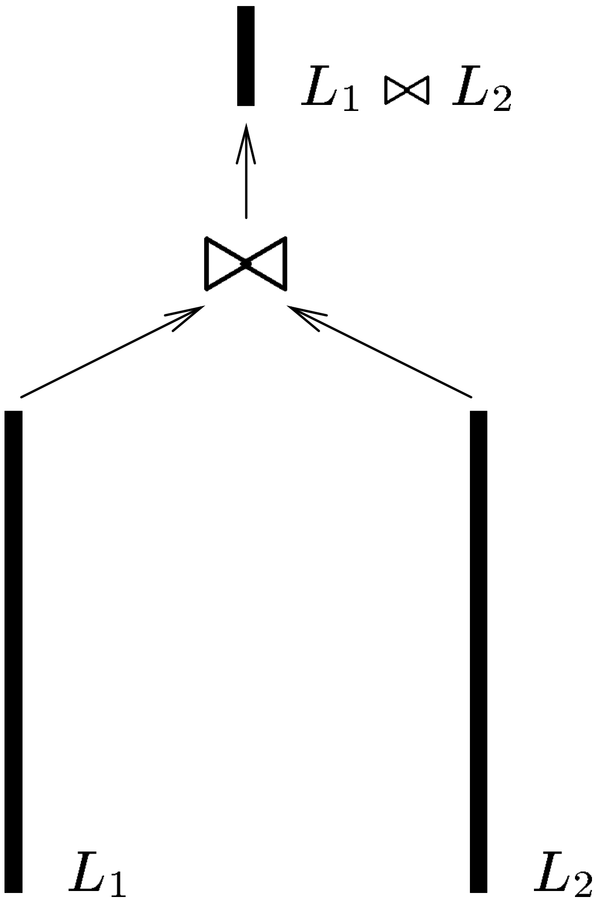
Input: Lists $L_1, \ldots, L_k \subseteq S$.

Desired output: Find $x_1, \ldots, x_k$ satisfying

$$x_1 + \cdots + x_k = 0 \quad \text{and} \quad x_i \in L_i.$$

Note: I'll assume the lists can be extended to be as large as desired.

Review: The Standard Birthday Algorithm



To solve the 2-sum problem:

1. Compute $L' \stackrel{\text{def}}{=} L_1 \bowtie L_2$.
2. Output any $\langle x_1, x_2 \rangle$ from L'.

To compute $L_1 \bowtie L_2$:

1. Sort L_1.
2. Sort L_2.
3. Scan L_1, L_2, looking for matches between them.

Defn: $L_1 \bowtie L_2 \stackrel{\text{def}}{=} \{\langle x_1, x_2 \rangle \in L_1 \times L_2 : x_1 = x_2\}$.

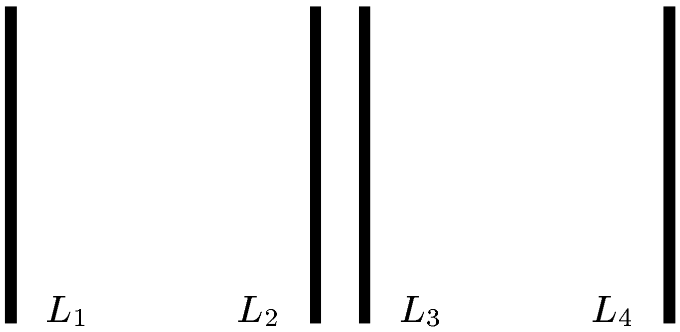
New: The 4-tree Algorithm

To solve the 4-sum problem:

1. Set $\ell \stackrel{\text{def}}{=} n/3$.
2. Compute $L' \stackrel{\text{def}}{=} (L_1 \bowtie_\ell L_2)$ $\bowtie (L_3 \bowtie_\ell L_4)$.
3. Output any element of L'.

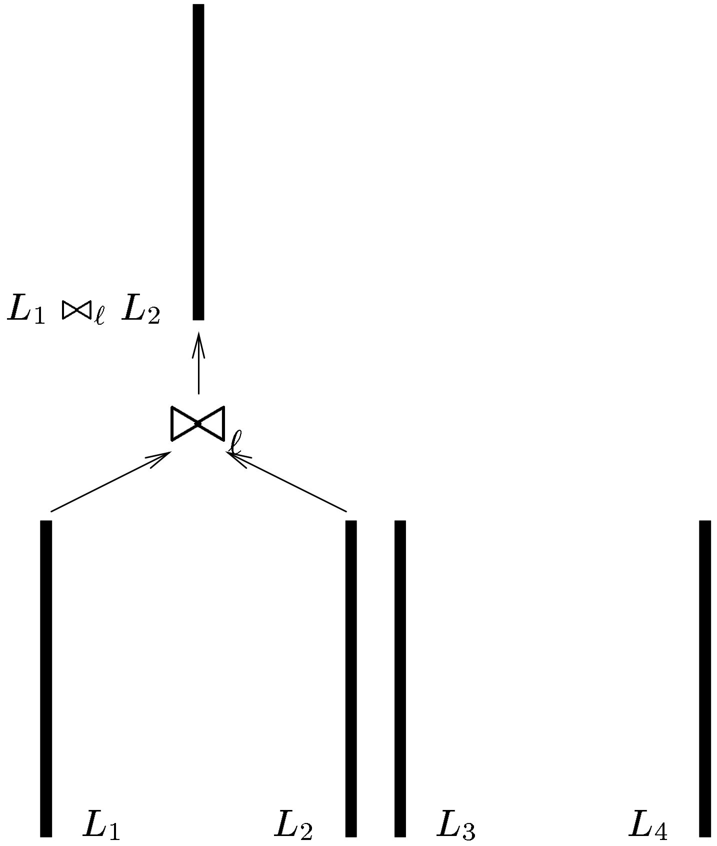
To compute $L_1 \bowtie_\ell L_2$:

1. Sort L_1 by its low ℓ bits.
2. Sort L_2 by its low ℓ bits.
3. Scan L_1, L_2, looking for matches in their low ℓ bits.



Defn: $L_1 \bowtie_\ell L_2 \stackrel{\text{def}}{=} \{\langle x_1, x_2 \rangle \in L_1 \times L_2 : \text{low}_\ell(x_1) = \text{low}_\ell(x_2)\}$.

New: The 4-tree Algorithm



To solve the 4-sum problem:

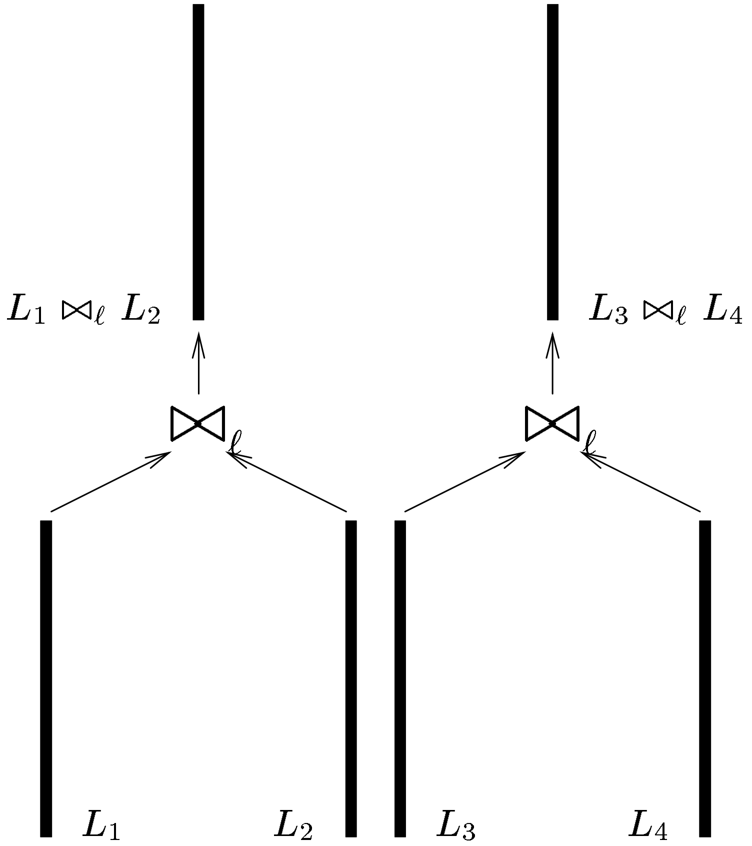
1. Set $\ell \stackrel{\text{def}}{=} n/3$.
2. Compute $L' \stackrel{\text{def}}{=} (L_1 \bowtie_\ell L_2) \bowtie (L_3 \bowtie_\ell L_4)$.
3. Output any element of L'.

To compute $L_1 \bowtie_\ell L_2$:

1. Sort L_1 by its low ℓ bits.
2. Sort L_2 by its low ℓ bits.
3. Scan L_1, L_2, looking for matches in their low ℓ bits.

Defn: $L_1 \bowtie_\ell L_2 \stackrel{\text{def}}{=} \{\langle x_1, x_2 \rangle \in L_1 \times L_2 : \text{low}_\ell(x_1) = \text{low}_\ell(x_2)\}$.

New: The 4-tree Algorithm



To solve the 4-sum problem:

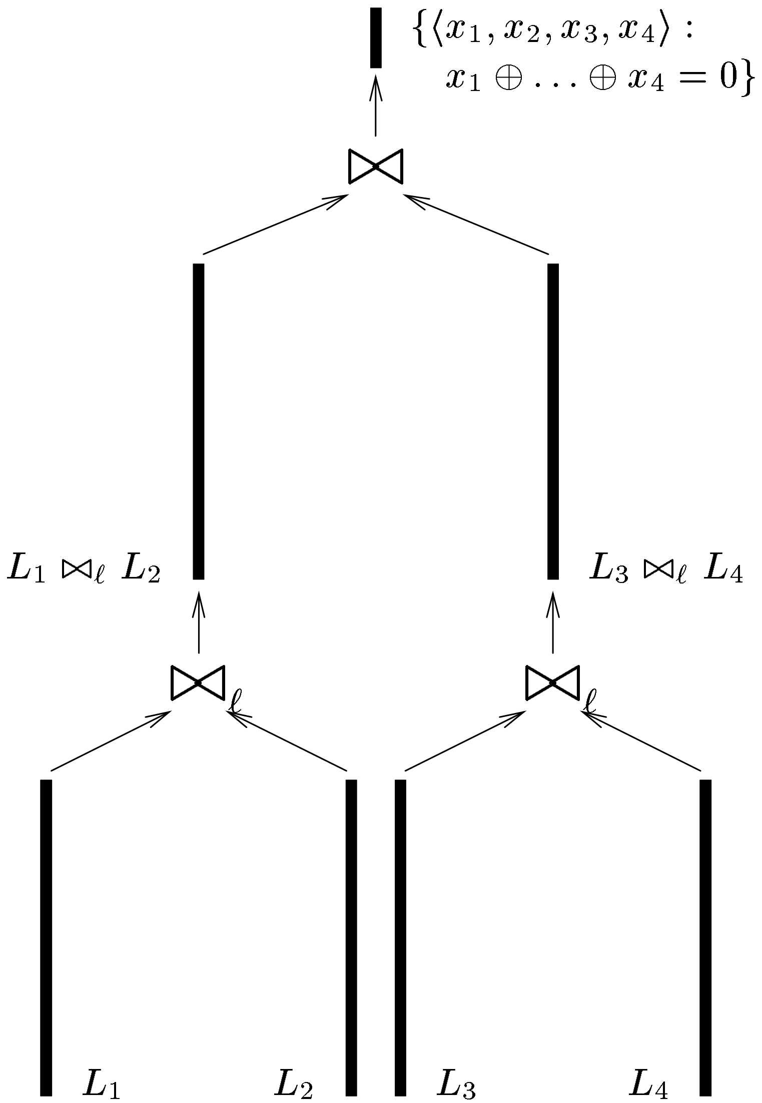
1. Set $\ell \stackrel{\text{def}}{=} n/3$.
2. Compute $L' \stackrel{\text{def}}{=} (L_1 \bowtie_\ell L_2) \bowtie (L_3 \bowtie_\ell L_4)$.
3. Output any element of L'.

To compute $L_1 \bowtie_\ell L_2$:

1. Sort L_1 by its low ℓ bits.
2. Sort L_2 by its low ℓ bits.
3. Scan L_1, L_2, looking for matches in their low ℓ bits.

Defn: $L_1 \bowtie_\ell L_2 \stackrel{\text{def}}{=} \{\langle x_1, x_2 \rangle \in L_1 \times L_2 : \text{low}_\ell(x_1) = \text{low}_\ell(x_2)\}$.

New: The 4-tree Algorithm



To solve the 4-sum problem:

1. Set $\ell \stackrel{\text{def}}{=} n/3$.
2. Compute $L' \stackrel{\text{def}}{=} (L_1 \bowtie_\ell L_2) \bowtie (L_3 \bowtie_\ell L_4)$.
3. Output any element of L'.

To compute $L_1 \bowtie_\ell L_2$:

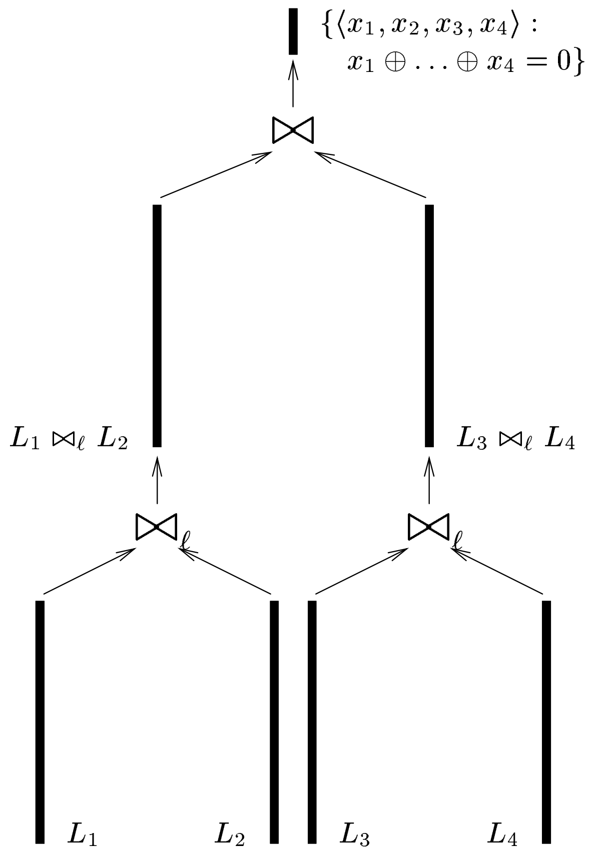
1. Sort L_1 by its low ℓ bits.
2. Sort L_2 by its low ℓ bits.
3. Scan L_1, L_2, looking for matches in their low ℓ bits.

Defn: $L_1 \bowtie_\ell L_2 \stackrel{\text{def}}{=} \{\langle x_1, x_2\rangle \in L_1 \times L_2 : \text{low}_\ell(x_1) = \text{low}_\ell(x_2)\}$.

Analysis of the 4-tree Algorithm

Note: if $\text{low}_\ell(x_1) = \text{low}_\ell(x_2)$ and $\text{low}_\ell(x_3) = \text{low}_\ell(x_4)$, then $\Pr[x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0] = 1/2^{n-\ell} = 2^{-2n/3}$.

Also, $|L_1 \bowtie_\ell L_2| \approx 2^{n/3}$, so there are about $2^{2n/3}$ quadruples (x_1, x_2, x_3, x_4) to consider. Hence, there's probably a quadruple that XORs to zero.



Conclusion: the 4-tree algorithm solves the 4-sum problem for XOR in $O(2^{n/3})$ time and space.

Summary of Algorithms

- For XOR and modular addition:
  - The 4-sum problem can be solved in $O(2^{n/3})$ time.
  - The k-sum problem can be solved in $O(k2^{n/(1+\lfloor \lg k \rfloor)})$ time.
  - If k is unrestricted, there is a $O(2^{2\sqrt{n}})$ time algorithm

    $\Rightarrow$ There's a subexponential time algorithm.

    $\Rightarrow$ If we want 80-bit security, we need $n \geq 1600$.
- For groups where the discrete log is hard:
  - The k-sum problem is hard.
- For more, see

  `http://www.cs.berkeley.edu/~daw/papers/genbday.html`

Cryptanalysis of AdHash

AdHash: For $M = \langle M_1, \ldots, M_k \rangle$, define

$$\mathrm{AdHash}(M) \stackrel{\mathrm{def}}{=} h(1, M_1) + \cdots + h(k, M_k) \bmod m.$$

(Often $\lg m \approx 128$ or 256 in practical instantiations.)

Cryptanalysis of AdHash

AdHash: For $M = \langle M_1, \ldots, M_k \rangle$, define
AdHash$(M) \stackrel{\text{def}}{=} h(1, M_1) + \cdots + h(k, M_k) \bmod m$.
(Often $\lg m \approx 128$ or 256 in practical instantiations.)

The attack: Let L_i contain values of the form $h(i, m_i)$, where m_i varies. To find a pre-image of digest c, use the k-tree algorithm to find $x_1, \ldots, x_k$ solving

$$x_1 + \cdots + x_k \equiv c \pmod{m} \quad \text{and} \quad x_i \in L_i.$$

Cryptanalysis of AdHash

AdHash: For $M = \langle M_1, \ldots, M_k \rangle$, define
AdHash$(M) \stackrel{\text{def}}{=} h(1, M_1) + \cdots + h(k, M_k) \bmod m$.
(Often $\lg m \approx 128$ or 256 in practical instantiations.)

The attack: Let L_i contain values of the form $h(i, m_i)$, where m_i varies. To find a pre-image of digest c, use the k-tree algorithm to find $x_1, \ldots, x_k$ solving

$$x_1 + \cdots + x_k \equiv c \pmod{m} \quad \text{and} \quad x_i \in L_i.$$

The workfactor: Inverts AdHash in $O(2^{2\sqrt{\lg m}})$ time.
Hence for security we need $m \gg 2^{1600}$.

The ROS problem

Goal: Find a $k \times k$ matrix M over $GF(q)$ of the form

$$M = \begin{bmatrix} M_{1,1} & M_{1,2} & \cdots & M_{1,k-1} & H(M_{1,1},\ldots,M_{1,k-1}) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ M_{k,1} & M_{k,2} & \cdots & M_{k,k-1} & H(M_{k,1},\ldots,M_{k,k-1}) \end{bmatrix}$$

so that there is some $v = (v_1, \ldots, v_k)^T$ with $Mv = 0$ and $v_k = -1$. (H is a cryptographic hash function.)

Motivation: Schnorr notes that any algorithm to solve the ROS problem gives a one-more forgery attack on Schnorr and Schnorr-Okamoto blind signatures using $k-1$ parallel interactions with the signer [Sch01].

Cryptanalysis of Schnorr Blind Signatures

How to solve the ROS problem: For $k = 4$, consider

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

This is of the desired form if

$$\frac{F(w_1,0,0)}{w_1} + \frac{F(0,w_2,0)}{w_2} + \frac{F(0,0,w_3)}{w_3} - \frac{F(w_4,w_4,w_4)}{w_4} \equiv 0 \pmod{q}.$$

It's a 4-sum problem over $GF(q)$! Use the 4-tree algorithm.

Cryptanalysis of Schnorr Blind Signatures

How to solve the ROS problem: For $k = 4$, consider

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

This is of the desired form if

$$\frac{F(w_1,0,0)}{w_1} + \frac{F(0,w_2,0)}{w_2} + \frac{F(0,0,w_3)}{w_3} - \frac{F(w_4,w_4,w_4)}{w_4} \equiv 0 \pmod{q}.$$

It's a 4-sum problem over $GF(q)$! Use the 4-tree algorithm.

The attack: Breaks Schnorr blind signatures using 3 parallel interactions and $\Theta(q^{1/3})$ work. More generally, can break with $O(2^{2\sqrt{\lg q}})$ interactions, time, and space $\Rightarrow$ *subexponential attack*.

Concluding Thoughts

- The k-sum problem is a powerful tool for cryptanalysis that generalizes birthday attacks.
  - It yields subexponential-time attacks on AdHash, (Okamoto-)Schnorr blind signatures in elliptic curve groups, and other schemes.
  - “If there is a birthday attack, maybe there is a k-sums attack.”
- 4-sums can be solved in cube-root time.
  - Open problem: Are there better algorithms for k-sums?