The k-sum Problem

Solutions and Applications

Christiane Peters

DTU

Ice Break – June 8, 2013

Talk outline

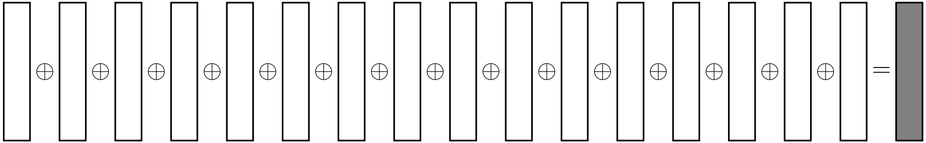
1. Motivation
2. Information-set decoding
3. Linearization
4. Generalized birthday attacks
5. Outlook

1. Motivation
2. Information-set decoding
3. Linearization
4. Generalized birthday attacks
5. Outlook

The k-sum problem

- Given k lists $L_1, \ldots, L_k$ containing bit strings of length n.
- Find elements $x_1 \in L_1, \ldots, x_k \in L_k$:

$$x_1 \oplus \ldots \oplus x_k = 0.$$



- Examples in this talk: $k = 2$, $k = w$, $k = 2w$, $k =$ something related to n, k, w etc.

The k-sum problem is well-studied

Appears in many different fields in cryptanalysis:

- birthday attacks
- meet-in-the-middle attacks on multiple encryption
- multi-collisions
- solving knapsacks
- syndrome decoding
- attacking the learning-parity-with-noise problem (LPN)
- ...

Selected literature:

- Yuval (1978)
- Hellman–Merkle (1981)
- Coppersmith (1985)
- Camion–Patarin (1991)
- Coppersmith (1992)
- van Oorschot–Wiener (1996)
- Micciancio–Bellare (1997)
- Wagner (2002)
- Augot–Finiasz–Sendrier (2003)
- Saarinen (2007, 2009)
- Joux–Lucks (2009)
- Howgrave-Graham–Joux (2010)
- Bernstein–Lange–P.–Schwabe (2011)
- Becker–Coron–Joux (2011)
- Dinur–Dunkelman–Keller–Shamir (2012)

Applications in this talk

Bellare–Micciancio (1997):

- "incrementable" hash function

$$\mathrm{XHASH}(f, m) = \bigoplus_{i=1}^{w} f(m_i)$$

Finiasz et al. (2003, 2007, 2008):

- fast syndrome-based hash function

$$\mathrm{FSB}(H, m) = \bigoplus_{i=1}^{w} H_i[m_i]$$

- Use as compression function in a Merkle–Damgård construction.
- Plus: fast, incrementable, parallelizable,...
- Minus: large matrix of random constants (fix: quasi-cyclic structure).

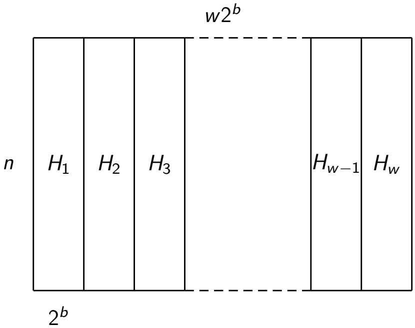
A simple compression function

- Consider inputs of length $w \cdot b$:

$$m = (m_1, m_2, \ldots, m_w),$$

each m_i having b bits.

- Take an $n \times w2^b$ binary (pseudo-)random matrix, consisting of w blocks with 2^b columns each: $H = (H_1, H_2, \ldots, H_w)$.



- Regard the m_i as b-bit indices and define

$$\mathrm{FSB}(H, m) = H_1[m_1] \oplus H_2[m_2] \oplus \ldots \oplus H_w[m_w].$$

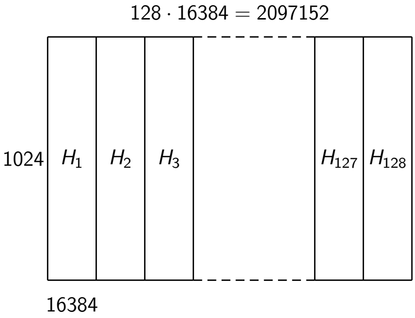
Mini example: compression function

```
sage: n=8; w=4; b=2
sage: set_random_seed(314)
sage: # compression matrix
sage: H=random_matrix(GF(2), n, w*2^b); print H
[1 1 1 0 1 0 1 0 1 1 1 1 0 1 1 0]
[1 1 1 0 1 1 0 0 1 0 1 0 1 1 0 0]
[1 1 0 1 0 1 0 0 0 1 0 0 1 0 1 0]
[0 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0]
[1 0 1 0 0 0 0 1 1 1 0 1 0 0 0 1]
[0 1 0 0 0 0 0 1 1 0 1 0 0 0 1 1]
[1 1 1 0 1 1 1 1 1 0 1 1 0 0 0 0]
[1 1 0 0 1 0 0 1 1 1 1 1 0 0 0 0]
sage: # message m=(m[1],..,m[w]), m[i] in [0,..,2^b-1]
sage: m=random_vector(IntegerModRing(2^b),w); print m
(2, 3, 3, 0)
sage: # hash
sage: x=sum([H.column(i*2^b+m[i]) for i in range(w)]); print x
(0, 0, 1, 0, 0, 0, 0, 1)
```

FSB parameters for 128-bit security

FSB-256:

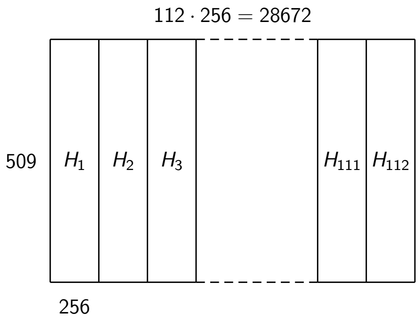
- FSB was a SHA-3 round-1 candidate;
- Parameters: $b = 14$, $w = 128$, $n = 1024$.
- FSB didn't make it to round 2.
- Too slow? No, sloppy security analysis. Parameters not tight. Loss in speed.



(R)FSB parameters for 128-bit security

FSB-256:

- FSB was a SHA-3 round-1 candidate;
- Parameters: $b = 14$, $w = 128$, $n = 1024$.
- FSB didn't make it to round 2.
- Too slow? No, sloppy security analysis. Parameters not tight. Loss in speed.

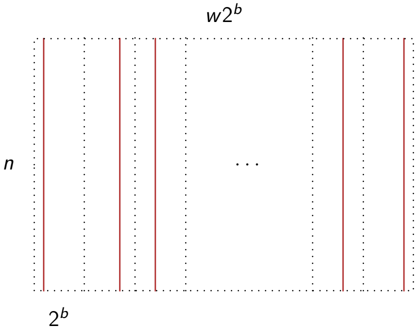


RFSB-509 (really fast syndrome-based):

- RFSB fast version of FSB by Bernstein et al.
- Parameters: $b = 8$, $w = 112$, $n = 509$.
- Fast software implementation by Bernstein and Schwabe in SUPERCOP.

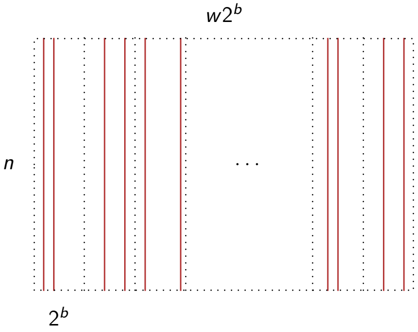
Preimages

- A preimage of $x \in \{0,1\}^n$ is given by w columns, exactly one per block, which add up to x.
- Note the abuse of notation: ultimately we're interested in the indices of those columns, not the columns themselves.
- A preimage here is in fact a pseudo-preimage for the actual hash function.
- In this talk we're only interested in the compression function.



Collisions

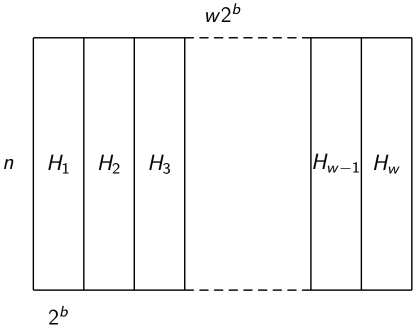
- A collision is given by $2w$ columns, exactly two per block, which add up to 0.
- Again abuse of notation: ultimately we're interested in the column indices.
- Collisions are in fact pseudo-collisions for the actual hash function.
- In this talk we're only interested in the compression function.



Parameters

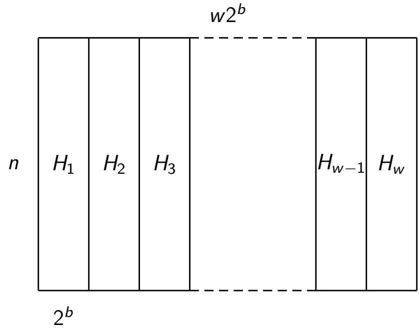
Security obviously depends on b, w, and n.

- Larger n makes it harder to find collisions (but reduces compression factor)
- Smaller w or b makes it harder to find collisions (but reduces compression factor)



Finding collisions and preimages

- Information-set decoding to find *regular* low-weight codewords (Augot–Finiasz–Sendrier, Bernstein–Lange–P.–Schwabe).
- Linearization (Bellare–Micchiancio, Saarinen)
- Generalized birthday attacks (Camion–Patarin, Wagner)



1. Motivation

2. Information-set decoding

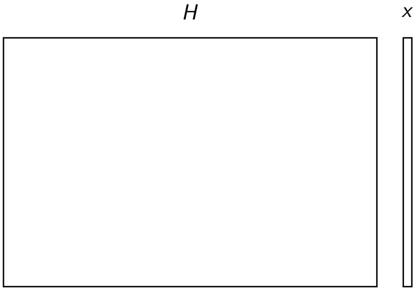
3. Linearization

4. Generalized birthday attacks

5. Outlook

Information-set decoding

Finding a preimage of $x \in \{0,1\}^n$ means finding w columns with xor x.

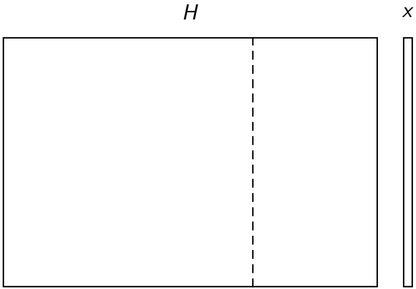


- Forget the block structure of H for a moment.
- "Unstructured w-sum problem"

Information-set decoding

Finding a preimage of $x \in \{0,1\}^n$ means finding w columns with xor x.

- Pick a set of n linearly independent columns.

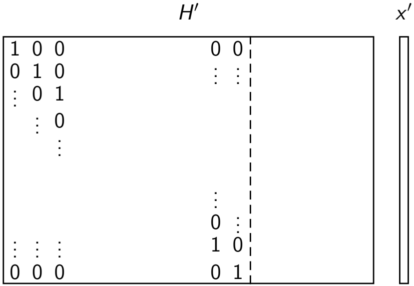


- Forget the block structure of H for a moment.
- "Unstructured w-sum problem"

Information-set decoding

Finding a preimage of $x \in \{0,1\}^n$ means finding w columns with xor x.

- Pick a set of n linearly independent columns.
- Apply elementary row operations to H and x to bring H into a form $H' = [I_n|Q]$ wrt to the selected columns.
- If x' has weight w, it is sum of w columns from the identity submatrix. Done.
- If not start with a fresh set of n columns (iterative algorithm).



- Forget the block structure of H for a moment.
- "Unstructured w-sum problem"

Cost information-set decoding

Very rough cost:

$$\text{Cost}_{\text{Gauss Elim}} / \text{Prob}_{\text{success}}$$

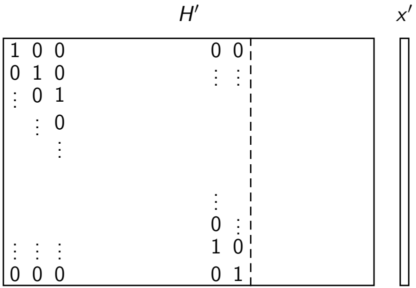
where

$$\text{Prob}_{\text{success}} = \frac{\binom{n}{w}}{\binom{2^b w}{w}} \cdot \frac{\binom{2^b w}{w}}{2^n} = \frac{\binom{n}{w}}{2^n}$$

- E.g., $n = 1024, w = 128, b = 14$: $\text{Prob}_{\text{success}} \approx 2^{-472}$.

Much better algorithms:

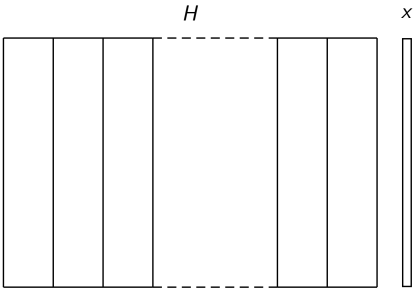
- Stern's collision decoding (birthday paradox), ball-collision decoding etc



- Forget the block structure of H for a moment.
- "Unstructured w-sum problem"

Regular information-set decoding

Finding a preimage of $x \in \{0,1\}^n$ means finding w columns, exactly one per block, with xor x.

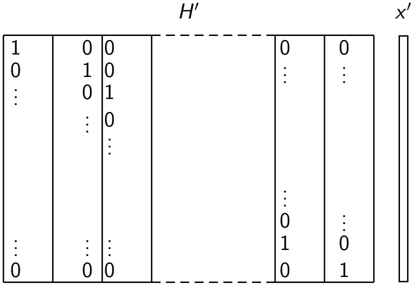


- Don't forget the block structure of H.
- w-sum problem

Regular information-set decoding

Finding a preimage of $x \in \{0,1\}^n$ means finding w columns, exactly one per block, with xor x.

- Pick a set of n linearly independent columns, one per block.
- Apply elementary row operations to H and x to bring H into a form $H' = [\text{“}I_n\text{”}|Q]$ where “I_n” is spread over w blocks.
- If x' has weight w, it is sum of w columns from the identity submatrix. Done.
- If not start with a fresh set of n columns.



- Don't forget the block structure of H.
- w-sum problem

Cost of regular information-set decoding

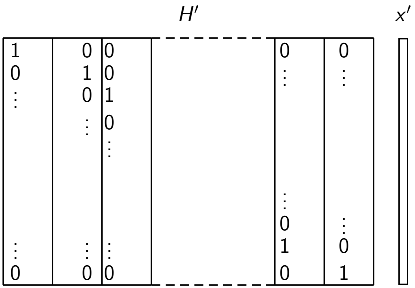
Finding a preimage of $x \in \{0,1\}^n$ means finding w columns, exactly one per block, with xor x.

Augot et al (2003):

- The probability of finding a preimage is roughly

$$\frac{\binom{n}{w}^w}{2^n}$$

- This probability is much smaller than for the classical decoding problem (which is already NP-hard).
- Ratio $w!/w^w$.
- E.g., $n = 1024$, $w = 128$, $b = 14$: $\text{Prob}_{\text{success}} \approx 2^{-640}$.



- Don't forget the block structure of H.
- w-sum problem

Cost of 2-regular information-set decoding

Find collisions, i.e., two columns per block with xor 0.

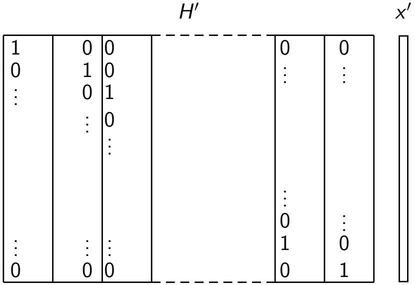
Augot et al (2003):

- The expected number of iterations of the 2-regular syndrome-decoding algorithm is

$$\min\left\{\frac{2^n}{\left(\binom{n/w_0}{2}+1\right)^{w_0}} : w_0 \in \{1,2,\ldots,w\}\right\}.$$

Bernstein et al (2011):

- 2-regular syndrome decoding using birthday paradox.
- Faster, much more complicated.

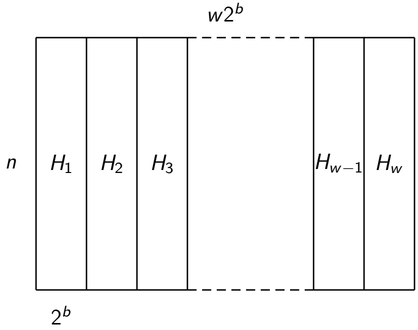


- Don't forget the block structure of H.
- $2w$-sum problem

Preimages through linearization

Given $x \in \{0,1\}^n$ find $(m_1, \ldots, m_w) \in [0, 2^b - 1]^w$ so that

$$x = H_1[m_1] \oplus \ldots \oplus H_w[m_w].$$



- w-sum problem

- Restrict to messages $m_i \in \{0,1\}$.
- Consider the $n \times w$ matrix

$$\Delta = [H_1[0] \oplus H_1[1] | \cdots | H_w[0] \oplus H_w[1]].$$

- Note that

$$\Delta[i] \cdot m_i \oplus H_i[0] = \begin{cases} H_i[0] & \text{if } m_i = 0 \\ H_i[1] & \text{if } m_i = 1 \end{cases}$$

- Hence

$$\Delta \cdot m = x \oplus \bigoplus_{i=1}^{w} H_i[0].$$

- $[n = w]$: if Δ is invertible we can find

$$m = \Delta^{-1}(x \oplus \bigoplus_{i=1}^{w} H_i[0])$$

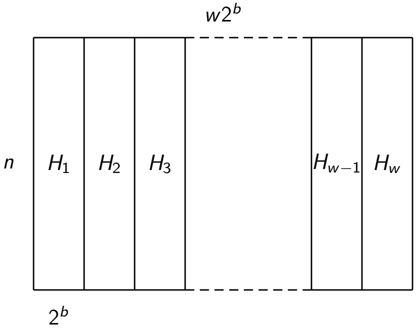
Toy example: linearization

```
sage: n=4; w=4; b=2   # m=(m[1],..,m[w]), m[i] in [0,..,2^b-1]
sage: set_random_seed(0)
sage: H=random_matrix(GF(2), n, w*2^b) # FSB random matrix
sage: print H
[0 1 0 1 1 0 0 0 1 1 0 1 0 0 1 0]
[0 1 1 1 0 1 1 0 0 1 0 0 0 1 1 1]
[0 0 0 1 0 1 0 0 1 0 1 1 1 0 0 1]
[0 1 1 0 0 1 0 1 1 0 0 0 0 0 0 0]
sage: c=sum([H.column(i*2^b) for i in range(w)]); print c
(0, 0, 0, 1)
sage: Delta=column_matrix([H.column(i*2^b)+H.column(i*2^b+1)\
....:                      for i in range(w)]); print Delta
[1 1 0 0]
[1 1 1 1]
[0 1 1 1]
[1 1 1 0]
sage: Delta.det()
1
sage: x=sum([H.column(i*2^b+randrange(2)) for i in range(w)]);
print x
(1, 0, 0, 1)
sage: m=(Delta^(-1)*(x+c)).lift() # lift m[i] to integer value
sage: (x-sum([H.column(i*2^b+m[i]) for i in range(w)])).is_zero()
True
```

Preimages through linearization: try again

Given $x \in \{0,1\}^n$ find $(m_1, \ldots, m_w) \in [0, 2^b - 1]^w$ so that

$$x = H_1[m_1] \oplus \ldots \oplus H_w[m_w].$$



- w-sum problem

- Restrict to $m_i \in \{\alpha_i, \beta_i\}$, $\alpha_i \neq \beta_i$.
- Consider the matrix

$$\Delta = [H_1[\alpha_1] \oplus H_1[\beta_1] | \cdots | H_w[\alpha_w] \oplus H_w[\beta_w]].$$

- Note that

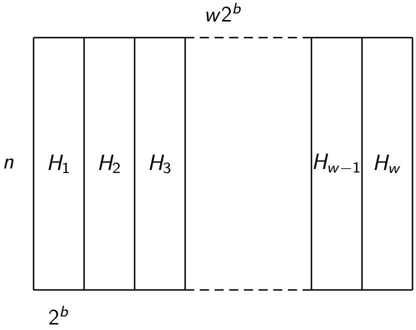
$$\Delta[i] \cdot \gamma_i \oplus H_i[\alpha_i] = \begin{cases} H_i[\alpha_i] & \text{if } \gamma_i = 0 \\ H_i[\beta_i] & \text{if } \gamma_i = 1 \end{cases}$$

- Hence $\Delta \cdot \gamma = x \oplus \bigoplus_{i=1}^{w} H_i[\alpha_i]$.
- $[n = w]$: if Δ is invertible, compute the $\{0,1\}$-vector γ.
- Then
  $x = \bigoplus H_i[m_i] = \bigoplus H_i[\alpha_i + \gamma_i(\beta_i - \alpha_i)]$.

Preimages through linearization $w < n$

Given $x \in \{0,1\}^n$ find $(m_1, \ldots, m_w) \in [0, 2^b - 1]^w$ so that

$$x = H_1[m_1] \oplus \ldots \oplus H_w[m_w].$$



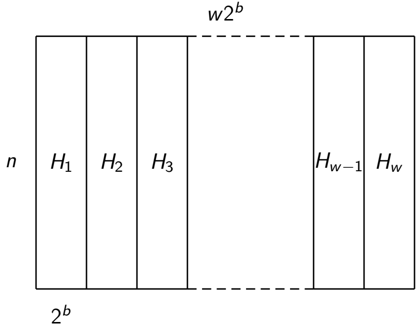
- w-sum problem

- The main obstacle to this attack is that if $w < n$ then rank Δ is at most w (and sometimes less),
- Under suitable randomness assumptions the desired linear relation exists with probability at most $2^w/2^n$.
- The expected number of iterations is therefore at least $2^n/2^w$; e.g., approximately $2^{0.75n}$ if $w \approx n/4$.

Collisions through linearization for $n = 2w$

Find $(m_1, \dots, m_w), (m'_1, \dots, m'_w) \in [0, 2^b - 1]^w$ so that

$$\bigoplus_{i=1}^{w} (H_i[m_i] \oplus H_i[m'_1]) = 0.$$



- $2w$-sum problem

Saarinen (2007):

- Compute two $n \times w$ matrices
  $\Delta = [H_1[\alpha_1] \oplus H_1[\beta_1] | \cdots | H_w[\alpha_w] \oplus H_w[\beta_w]]$
  and
  $\Delta' = [H_1[\alpha'_1] \oplus H_1[\beta'_1] | \cdots | H_w[\alpha'_w] \oplus H_w[\beta'_w]]$.
- Find solutions to γ, γ' to
  $\Delta \cdot \gamma \oplus \bigoplus_{i=1}^{w} H_i[\alpha_i] = \Delta' \cdot \gamma' \oplus \bigoplus_{i=1}^{w} H_i[\alpha'_i]$.
- $n = 2w$: if $(\Delta|\Delta')$ is invertible, we find

$$(\Delta|\Delta')^{-1} \cdot \begin{bmatrix} \gamma \\ \gamma' \end{bmatrix} = \begin{bmatrix} \bigoplus_{i=1}^{w} H_i[\alpha_i] \\ \bigoplus_{i=1}^{w} H_i[\alpha'_i] \end{bmatrix}.$$

- Solution (γ, γ') exists with probability $2^{2w}/2^n$.

Implications

Old FSB parameters: $n = 1024$, $w = 1024$, $b = 8$, i.e., a compression matrix H with $w \cdot 2^b = 262144$ columns.

- Originally claimed to provide 128-bit security against information-set decoding.
- Saarinen found collisions and preimages in under a second on a low-end pc.

Newer parameters:

- Very rough: ensure $w < n/4$ (reduced compression factor).

1. Motivation
2. Information-set decoding
3. Linearization
4. Generalized birthday attacks
5. Outlook

Birthday Paradox

- Given two lists L, L' containing bit strings of length n.
- Find collision $(x, x') \in L \times L'$: $x = x'$

Applications:

- Collisions in hash functions
- Any kind of meet-in-the-middle attack

Can expect to find a collision if $|L|, |L'|$ in $O(2^{n/2})$.

- Cost: $O(2^{n/2})$ time and space.

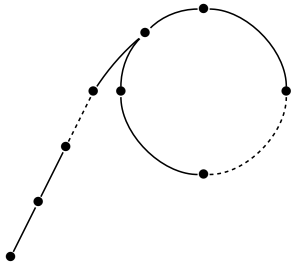
Birthday attack in practice

Straight-forward:

- Sort list L' and then check for each $x \in L$ if $x \in L'$.
- Alternative: use hash tables.

Space-efficient:

- Use Pollard variant (functional graph).



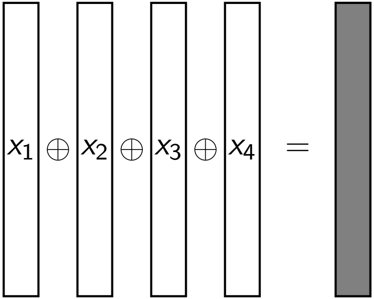
[http://cryptojedi.org/misc/data/pollard.tex]

4-sum problem

Consider 4 lists L_1, L_2, L_3, L_4 containing uniform random n-bit strings.

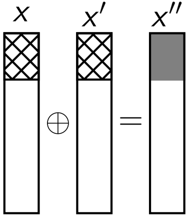
- Goal: find at least one tuple (x_1, x_2, x_3, x_4) with $x_i \in L_i$ such that

$$x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0.$$



Merge operation

- Given two lists L, L' containing bit strings of length n.



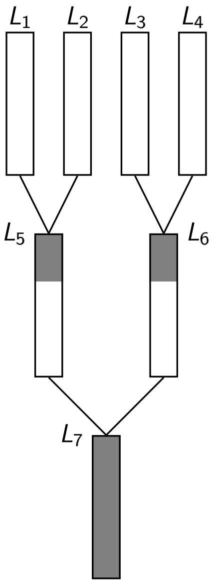
Merge L and L' on ℓ bits:

- For all pairs $(x, x') \in L \times L'$:
- If x and x' are equal on their *left-most ℓ bits* compute $x'' = x \oplus x'$ and store x'' in a new list L''.

- Note that all elements in L'' have their left-most bits set to 0.

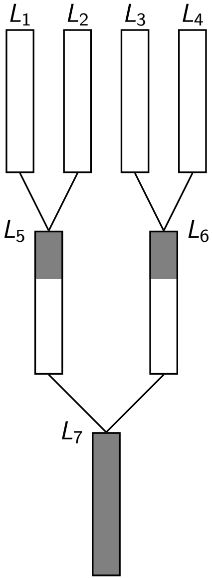
Tree algorithm (1)

1. Generate lists L_i with each $\sim 2^{n/3}$ bit strings of length n.
2. Merge lists L_1 and L_2 on left-most $n/3$ bits.
3. Similarly create a list L_6 by merging the lists L_3 and L_4 on $n/3$ bits.
4. Merge L_5 and L_6 on the remaining $2n/3$ bits.



Tree algorithm (2)

- If $|L_i| \sim 2^{n/3}$ for $i = 1, 2, 3, 4$ then we can expect that the merged lists L_5, L_6 also have $\sim 2^{n/3}$ elements.
- Apply birthday trick to remaining $2n/3$ bits to find collision.



Camion–Patarin (1991), Wagner (2002):

- expect to find one collision in $O(2^{n/3})$ time and space.

Example: 4-sum Problem (Level 1)

- $n = 9$, $|L_i| = 8$ for $i = 1, 2, 3, 4$:
- Find one column per list whose xor is 0.

$$
\left|\begin{array}{cccccccc|cccccccc|cccccccc|cccccccc}
1&1&1&0&0&0&1&1&0&1&0&1&1&0&0&1&0&1&0&1&1&0&0&0&0&1&1&1&1&0&0&1\\
1&0&1&0&1&1&0&0&1&0&1&1&0&1&1&0&1&0&0&0&1&1&1&1&1&1&0&1&0&0&1&0\\
0&1&1&0&1&1&0&1&0&1&1&1&1&1&1&0&1&1&1&0&0&0&0&0&1&1&0&1&0&0&0&1\\
0&0&1&0&0&1&1&0&1&1&1&1&1&0&1&0&1&0&1&1&1&1&0&0&1&1&0&0&1&0&1&0\\
0&0&1&0&0&1&1&1&1&1&0&1&1&0&0&1&1&1&1&1&0&1&0&0&0&1&1&0&0&0&0&0\\
0&0&0&1&0&0&0&1&0&1&0&0&1&1&1&1&1&0&0&1&1&0&0&1&0&1&1&1&0&0&0&1\\
1&0&0&0&1&1&0&1&0&0&0&0&0&1&1&1&0&0&0&1&0&0&1&0&0&1&1&0&0&1&0&0\\
1&1&1&0&1&0&1&1&1&1&1&0&0&1&1&1&0&0&1&1&1&1&1&0&0&0&0&1&1&1&1&1\\
1&1&1&1&1&1&1&0&1&1&1&1&1&0&0&0&1&0&1&1&1&1&1&1&0&1&1&1&0&0&0&0
\end{array}\right|
$$

Example: 4-sum Problem (Level 1)

- $n = 9$, $|L_i| = 8$ for $i = 1, 2, 3, 4$:
- Consider lists L_1 and L_2.

$$
\left|\begin{array}{cccccccc|cccccccc|c}
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & \\
1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & \\
1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 &
\end{array}\right|
$$

Example: 4-sum Problem (Level 1)

- $n = 9$, $|L_i| = 8$ for $i = 1, 2, 3, 4$:
- Consider lists L_1 and L_2 on 3 bits.

1 1 1 0 0 0 1 1	0 1 0 1 1 0 0 1	
1 0 1 0 1 1 0 0	1 0 1 1 0 1 1 0	
0 1 1 0 1 1 0 1	0 1 1 1 1 1 1 0	

Example: 4-sum Problem (Level 1)

- $n = 9$, $|L_i| = 8$ for $i = 1, 2, 3, 4$:
- Look for matches between elements of L_1 and L_2.

$$
\begin{array}{|cccccccc|cccccccc|c|}
1 & \color{red}{1} & 1 & 0 & 0 & 0 & 1 & 1 & 0 & \color{red}{1} & 0 & 1 & 1 & 0 & 0 & 1 & \\
1 & \color{red}{0} & 1 & 0 & 1 & 1 & 0 & 0 & 1 & \color{red}{0} & 1 & 1 & 0 & 1 & 1 & 0 & \\
0 & \color{red}{1} & 1 & 0 & 1 & 1 & 0 & 1 & 0 & \color{red}{1} & 1 & 1 & 1 & 1 & 1 & 0 & \\
 & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & \\
\end{array}
$$

Example: 4-sum Problem (Level 1)

- $n = 9$, $|L_i| = 8$ for $i = 1, 2, 3, 4$:
- Look for matches between elements of L_1 and L_2.

<table>
<tr><td>1 1 1 0 0 0 1 1 | 0 1 0 1 1 0 0 1</td><td></td></tr>
<tr><td>1 0 1 0 1 1 0 0 | 1 0 1 1 0 1 1 0</td><td></td></tr>
<tr><td>0 1 1 0 1 1 0 1 | 0 1 1 1 1 1 1 0</td><td></td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
<tr><td></td><td></td></tr>
</table>

Example: 4-sum Problem (Level 1)

- $n = 9$, $|L_i| = 8$ for $i = 1, 2, 3, 4$:
- Look for matches between elements of L_1 and L_2.

1 1 1 0 0 0 1 1	0 1 0 1 1 0 0 1	
1 0 1 0 1 1 0 0	1 0 1 1 0 1 1 0	
0 1 1 0 1 1 0 1	0 1 1 1 1 1 1 0	

Example: 4-sum Problem (Level 1)

- $n = 9$, $|L_i| = 8$ for $i = 1, 2, 3, 4$:
- Store positions of matching columns in L_1 and L_2 in L_5.

$$
\left|\begin{array}{cccccccc|cccccccc|}
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0
\end{array}\right|
$$

L_5 :[1, 1], [1, 4], [2, 3], [4, 2], [4, 5], [4, 6], [5, 2], [5, 5], [5, 6], [6, 7], [7, 1], [7, 4]

Example: 4-sum Problem (Level 1)

- $n = 9$, $|L_i| = 8$ for $i = 1, 2, 3, 4$:
- Look for matches between elements of L_3 and L_4 on 3 bits.

0 1 0 1 1 0 0 0	0 1 1 1 1 0 0 1
1 0 0 0 1 1 1 1	1 1 0 1 0 0 1 0
1 1 1 0 0 0 0 0	1 1 0 1 0 0 0 1

L_5 :[1, 1], [1, 4], [2, 3], [4, 2], [4, 5], [4, 6], [5, 2], [5, 5], [5, 6], [6, 7], [7, 1], [7, 4]

Example: 4-sum Problem (Level 1)

- $n = 9$, $|L_i| = 8$ for $i = 1, 2, 3, 4$:
- Look for matches between elements of L_3 and L_4 on 3 bits.

	0 1 0 1 1 0 0 0	0 1 1 1 1 0 0 1
	1 0 0 0 1 1 1 1	1 1 0 1 0 0 1 0
	1 1 1 0 0 0 0 0	1 1 0 1 0 0 0 1

L_5 :[1, 1], [1, 4], [2, 3], [4, 2], [4, 5], [4, 6], [5, 2], [5, 5], [5, 6], [6, 7], [7, 1], [7, 4]

Example: 4-sum Problem (Level 1)

- $n = 9$, $|L_i| = 8$ for $i = 1, 2, 3, 4$:
- Look for matches between elements of L_3 and L_4 on 3 bits.

	L_3	L_4
	0 1 0 1 1 0 0 0	0 1 1 1 1 0 0 1
	1 0 0 0 1 1 1 1	1 1 0 1 0 0 1 0
	1 1 1 0 0 0 0 0	1 1 0 1 0 0 0 1

L_5 :$[1, 1], [1, 4], [2, 3], [4, 2], [4, 5], [4, 6], [5, 2], [5, 5], [5, 6], [6, 7], [7, 1], [7, 4]$

Example: 4-sum Problem (Level 1)

- $n = 9$, $|L_i| = 8$ for $i = 1, 2, 3, 4$:
- Store positions of matching columns in L_3 and L_4 in L_6.

0 1 0 1 1 0 0 0	0 1 1 1 1 0 0 1
1 0 0 0 1 1 1 1	1 1 0 1 0 0 1 0
1 1 1 0 0 0 0 0	1 1 0 1 0 0 0 1

L_5 :[1, 1], [1, 4], [2, 3], [4, 2], [4, 5], [4, 6], [5, 2], [5, 5], [5, 6], [6, 7], [7, 1], [7, 4]

L_6 :[0, 0], [1, 7], [3, 2], [3, 4], [5, 6], [6, 6], [7, 6]

Example: 4-sum Problem (after Level 1)

- $n = 9$, $|L_i| = 8$ for $i = 1, 2, 3, 4$:
- Candidate columns after Level 1.

$$
\left|\begin{array}{cccccc|ccccccc|cccccc|ccccc}
1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0
\end{array}\right|
$$

L_5 :[1, 1], [1, 4], [2, 3], [4, 2], [4, 5], [4, 6], [5, 2], [5, 5], [5, 6], [6, 7], [7, 1], [7, 4]

L_6 :[0, 0], [1, 7], [3, 2], [3, 4], [5, 6], [6, 6], [7, 6]

Example: 4-sum Problem (Level 2)

- $n = 9$, $|L_i| = 8$ for $i = 1, 2, 3, 4$:
- Merge: take the xor of those candidate columns.

$$\left|\begin{array}{cccccccccccc|ccccccc}
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&0&1&0&1&0&1&0&1&1&1&0&0&1&0&0&1&1\\
1&1&0&0&0&0&1&1&1&0&0&0&1&1&0&1&1&0&0\\
1&1&0&0&1&1&0&1&1&1&0&0&1&1&0&1&0&0&1\\
0&0&0&1&0&0&1&0&0&1&1&1&0&0&0&1&0&1&0\\
0&1&1&0&0&0&1&1&1&0&0&1&0&1&1&0&0&0&1\\
0&0&0&0&1&1&0&1&1&1&1&1&1&0&0&1&1&1&1
\end{array}\right|$$

L_5 :[1, 1], [1, 4], [2, 3], [4, 2], [4, 5], [4, 6], [5, 2], [5, 5], [5, 6], [6, 7], [7, 1], [7, 4]

L_6 :[0, 0], [1, 7], [3, 2], [3, 4], [5, 6], [6, 6], [7, 6]

Example: 4-sum Problem (Level 2)

- $n = 9$, $|L_i| = 8$ for $i = 1, 2, 3, 4$:
- Ignore first three rows (zero after first round).

$$\left|\begin{array}{cccccccccccc|ccccccc}
 & & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & & \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}\right|$$

L_5 :[1, 1], [1, 4], [2, 3], [4, 2], [4, 5], [4, 6], [5, 2], [5, 5], [5, 6], [6, 7], [7, 1], [7, 4]

L_6 :[0, 0], [1, 7], [3, 2], [3, 4], [5, 6], [6, 6], [7, 6]

Example: 4-sum Problem (Level 2)

- $n = 9$, $|L_i| = 8$ for $i = 1, 2, 3, 4$:
- Look for matches on the remaining 6 bits.

$$
\left|\begin{array}{cccccccccccc|ccccccc}
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & \color{red}{1} & 1 & 0 & 0 & 1 & 0 & 0 & \color{red}{1} & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & \color{red}{0} & 0 & 1 & 1 & 0 & 1 & 1 & \color{red}{0} & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & \color{red}{0} & 0 & 1 & 1 & 0 & 1 & 0 & \color{red}{0} & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & \color{red}{1} & 1 & 0 & 0 & 0 & 1 & 0 & \color{red}{1} & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & \color{red}{0} & 1 & 0 & 1 & 1 & 0 & 0 & \color{red}{0} & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & \color{red}{1} & 1 & 1 & 0 & 0 & 1 & 1 & \color{red}{1} & 1
\end{array}\right|
$$

Notice the square root coming from the birthday paradox.
Lists of size $\sim 2^3$ containing elements of $2 \cdot 3$ (nonzero) bits.

Example: Match

- $n = 9$, $|L_i| = 8$ for $i = 1, 2, 3, 4$:
- Columns indexed by "[7, 1]" in L_5 and "[6, 6]" in L_6 yield a collision.

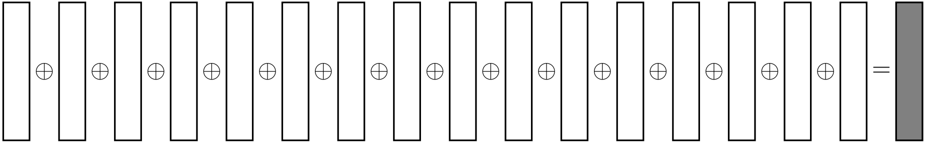
$$
\left|\begin{array}{cccccccc|cccccccc|cccccccc|cccccccc}
1&1&1&0&0&0&1&{\color{red}1} & 0&{\color{red}1}&0&1&1&0&0&1 & 0&1&0&1&1&0&{\color{red}0}&0 & 0&1&1&1&1&0&{\color{red}0}&1 \\
1&0&1&0&1&1&0&{\color{red}0} & 1&{\color{red}0}&1&1&0&1&1&0 & 1&0&0&0&1&1&{\color{red}1}&1 & 1&1&0&1&0&0&{\color{red}1}&0 \\
0&1&1&0&1&1&0&{\color{red}1} & 0&{\color{red}1}&1&1&1&1&1&0 & 1&1&1&0&0&0&{\color{red}0}&0 & 1&1&0&1&0&0&{\color{red}0}&1 \\
0&0&1&0&0&1&1&{\color{red}0} & 1&{\color{red}1}&1&1&1&0&1&0 & 1&0&1&1&1&1&{\color{red}0}&0 & 1&1&0&0&1&0&{\color{red}1}&0 \\
0&0&1&0&0&1&1&{\color{red}1} & 1&{\color{red}1}&0&1&1&0&0&1 & 1&1&1&1&0&1&{\color{red}0}&0 & 0&1&1&0&0&0&{\color{red}0}&0 \\
0&0&0&1&0&0&0&{\color{red}1} & 0&{\color{red}1}&0&0&1&1&1&1 & 1&0&0&1&1&0&{\color{red}0}&1 & 0&1&1&1&0&0&{\color{red}0}&1 \\
1&0&0&0&1&1&0&{\color{red}1} & 0&{\color{red}0}&0&0&0&1&1&1 & 0&0&0&1&0&0&{\color{red}1}&0 & 0&1&1&0&0&1&{\color{red}0}&0 \\
1&1&1&0&1&0&1&{\color{red}1} & 1&{\color{red}1}&1&0&0&1&1&1 & 0&0&1&1&1&1&{\color{red}1}&0 & 0&0&0&1&1&1&{\color{red}1}&1 \\
1&1&1&1&1&1&1&{\color{red}0} & 1&{\color{red}1}&1&1&1&0&0&0 & 1&0&1&1&1&1&{\color{red}1}&1 & 0&1&1&1&0&0&{\color{red}0}&0
\end{array}\right|
$$

Notice the square root coming from the birthday paradox.
Lists of size $\sim 2^3$ containing elements of $2 \cdot 3$ (nonzero) bits.

The k-sum problem

- Given k lists $L_1, \ldots, L_k$ containing bit strings of length n.
- Find elements $x_1 \in L_1, \ldots, x_k \in L_k$:

$$x_1 \oplus \ldots \oplus x_k = 0.$$



We've seen the generalized birthday algorithm for $k = 4$

- Let's move on to bigger k.
- Keep k a power of 2, so the computation can be organized using binary trees.

k-tree algorithm (1)

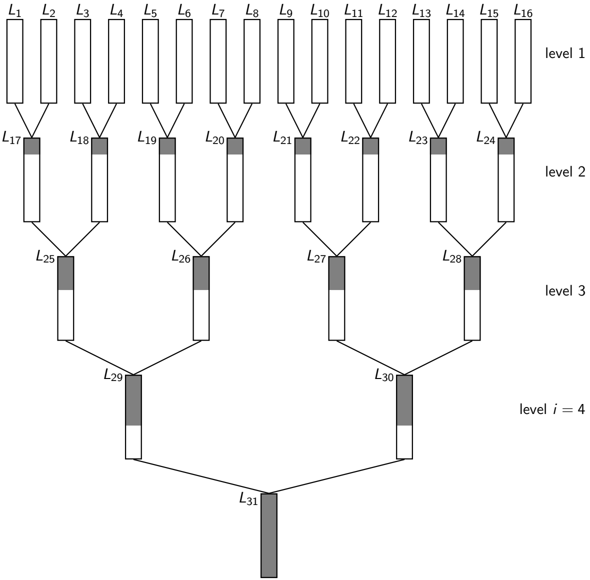
Goal: Find collision among $k = 2^i$ lists.

For $j = 1, \ldots, i-1$:

- Merge lists on level j by comparing elements on left-most $j \cdot n/(i+1)$ bits.

Level $j = i$:

- merge remaining two lists on $2n/(i+1)$ bits.



k-tree algorithm (2)

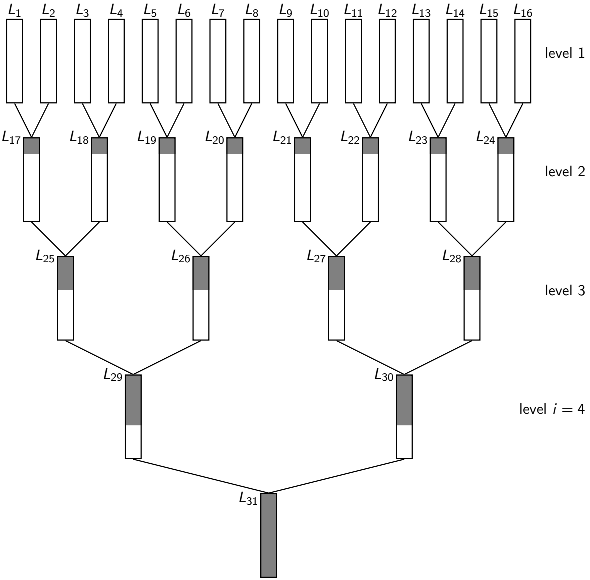
- If $|L_i| \sim 2^{n/(i+1)}$ on level j we expect that the merged lists also have $\sim 2^{n/(i+1)}$ elements.

Level i: list elements coincide on $(i-1)n/(i+1)$ bits.

- Apply birthday trick to remaining $2n/(i+1)$ bits.

Camion–Patarin (1991), Wagner (2002):

- expect to find one collision in $O(k2^{n/(i+1)})$ time and $O(2^{n/(i+1)})$ space.



Finding collisions using GBA

- Find collisions in the FSB compression function $n = 160$, $w = 64$, $b = 8$.
- $2w$-sum problem

Exercises

- Try to determine cost of an collision attack against FSB parameters
  - $n = 288$, $w = 128$, $b = 6$
  - $n = 224$, $w = 96$, $b = 8$

- Each of the $w = 64$ matrix blocks contains $2^b = 256$ columns.
- Build $w = 64$ lists by generating all $\binom{256}{2} \approx 2^{15}$ possible xors of two columns.
- Can we expect a collision on $n = 160$ bits using the generalized birthday attack using these 64 lists?
- No since $n = 160 > (\log_2(w) + 1) \cdot 15$.

Finding collisions using GBA

- Find collisions in the FSB compression function $n = 160, w = 64, b = 8$.
- $2w$-sum problem

Exercises

- Try to determine cost of an collision attack against FSB parameters
  - $n = 288, w = 128, b = 6$
  - $n = 224, w = 96, b = 8$

- Each of the $w = 64$ matrix blocks contains $2^b = 256$ columns.
- Build 32 lists from two blocks by generating all possible $\binom{256}{2}^2 \approx 2^{30}$ possible xors of four columns.
- Can we expect a collision on $n = 160$ bits using the generalized birthday attack using these 32 lists?
- Yes. Expect to find a collision in time $w2^{30} = 2^{36}$ since $n = 160 < \log_2(w) \cdot 30$.

Attack due to Coron and Joux (2004).

1. Motivation
2. Information-set decoding
3. Linearization
4. Generalized birthday attacks
5. Outlook

Todo

Difficult to choose parameters:

- Automated tool taking different approaches for the k-sum problem into account?

Further cryptanalysis needed:

- Asymptotic analysis for different w/n ratios
- Space-efficient variants?

Thanks.