Revisiting and Improving Algorithms for the 3XOR Problem

Charles Bouillaguet\textsuperscript{1} Claire Delaplace\textsuperscript{1,2} Pierre-Alain Fouque\textsuperscript{2}

\textsuperscript{2} University of Rennes 1, IRISA, France

\textsuperscript{1} University of Lille, CRIStAL, France

FSE 2018, Bruges
7th of March
3XOR Problem

Problem

Given three lists $A$, $B$, and $C$ of uniformly random elements of $\{0, 1\}^n$, find $(a, b, c) \in A \times B \times C$, such that $a \oplus b \oplus c = 0$.

- Difficult case of Generalised Birthday Problem
- Application in cryptanalysis of some authenticated encryption scheme
- Lists formed by querying oracles $\Rightarrow$ can be as big as we want
- $|A| \cdot |B| \cdot |C| \geq 2^n \Rightarrow$ solution w.h.p.
1 Background

2 Our New Algorithm

3 Adaptation of BDP Algorithm for the 3SUM problem
A Naive Quadratic Algorithm

Idea

- Create all $v = a \oplus b$
- Check if $v$ is in $C$

Time complexity:
$$O(|A| \cdot |B| + |C|)$$

Space:
$$O(|A| + |B| + |C|)$$

$|A| = |B| = 2^{n/3} \Rightarrow$ Time: $O(2^{2n/3})$, Space: $O(2^{n/3})$

$|A| = |B| = 2^{n/4}, |C| = 2^{n/2} \Rightarrow$ Time: $O(2^{n/2})$, Space: $O(2^{n/2})$

Time/Space tradeoff: Well studied in the past (e.g. [Wagner02], [Bernstein07]).
A Naive Quadratic Algorithm

Idea

- Create all $v = a \oplus b$
- Check if $v$ is in $C$

- **Time complexity:** $\mathcal{O}(|A| \cdot |B| + |C|)$
- **Space:** $\mathcal{O}(|A| + |B| + |C|)$

| $|A| = |B| = |C| = 2^{n/3}$ | Time: $\mathcal{O}(2^{2n/3})$, Space: $\mathcal{O}(2^{n/3})$ |
|-----------------------------|--------------------------------------------------|
| $|A| = |B| = 2^{n/4}, |C| = 2^{n/2}$ | Time: $\mathcal{O}(2^{n/2})$, Space: $\mathcal{O}(2^{n/2})$ |

Time/Space tradeoff: Well studied in the past (e.g. [Wagner02], [Bernstein07]).
A Naive Quadratic Algorithm

**Idea**

- Create all \( v = a \oplus b \)
- Check if \( v \) is in \( C \)

- Time complexity: \( \mathcal{O}(|A| \cdot |B| + |C|) \)
- Space: \( \mathcal{O}(|A| + |B| + |C|) \)
- \( |A| = |B| = |C| = 2^{n/3} \Rightarrow \) Time: \( \mathcal{O}(2^{2n/3}) \), Space: \( \mathcal{O}(2^{n/3}) \)
- \( |A| = |B| = 2^{n/4}, \ |C| = 2^{n/2} \Rightarrow \) Time: \( \mathcal{O}(2^{n/2}) \), Space: \( \mathcal{O}(2^{n/2}) \)

Time/Space tradeoff: Well studied in the past (e.g. [Wagner02], [Bernstein07]).
Wagner and its descendants

\[
\frac{n}{2} \sqrt{n/2} \approx \ell / \ln(\ell)
\]

**Description**

- Number of queries: increased up to \(\sim 2^{n/2}\)
- Elements of C start by \(p\)
Wagner and its descendants

Description

- Number of queries: increased up to $\sim 2^{n/2}$
- Elements of $C$ start by $p$
- For all $a, b$ s.t. $a \oplus b = (p|\ast)$
  - search $a \oplus b$ in $C$
Wagner and its descendants

Number of queries: increased up to $\sim 2^{n/2}$

Elements of $C$ start by $p$

For all $a, b$ s.t.

\[ a \oplus b = (p|*) \]

- search $a \oplus b$ in $C$

[Wagner02]: $2^{n/2}$ queries allowed

$|C| = 1.$

Time/Space $\mathcal{O}(2^{n/2})$
Wagner and its descendants

[NS14]: $2^\ell \sim \frac{2^{n/2}}{\sqrt{(n/2)/\ln(n/2)}}$ queries allowed

$p$: Most frequent prefix in $C$

Time/Space $\mathcal{O}\left(\frac{2^{n/2}}{\sqrt{n/\ln(n)}}\right)$

Description

- Number of queries: increased up to $\sim 2^{n/2}$
- Elements of $C$ start by $p$
- For all $a, b$ s.t. $a \oplus b = (p|\star)$
  - search $a \oplus b$ in $C$
Wagner and its descendants

Description

- Number of queries: increased up to $\sim 2^{n/2}$
- Elements of $C$ start by $p$
- For all $a, b$ s.t.
  \[ a \oplus b = (p|\ast) \]
  - search $a \oplus b$ in $C$

[Joux09]: $2^{n/2}/\sqrt{n/2}$ queries allowed

$|C| = n/2$, Basis change to force $p = 0$

Time/Space $O \left(2^{n/2}/\sqrt{n}\right)$
Discussion

Joux’s Algorithm best time complexity but...

\[ |A| = |B| = |C| = 2^{n/3} \]: about \(2^{64}\) operations

But only 206 GB of data

\[ \Rightarrow \text{Practical} \]

\[ \Rightarrow \text{Keep the lists small!} \]
Discussion

Joux’s Algorithm best time complexity but...

96-bit 3XOR

- Joux Algorithm: about $2^{48}$ operations
- But about 1 PB of data $\implies$ Impractical
Joux’s Algorithm best time complexity but...

96-bit 3XOR

- Joux Algorithm: about $2^{48}$ operations
- But about 1 PB of data $\implies$ Impractical
- Quad algorithm: with $|A| = |B| = |C| = 2^{n/3}$: about $2^{64}$ operations
- But only 206 GB of data $\implies$ Practical

Keep the lists small!
Joux’s Algorithm best time complexity but...

96-bit 3XOR

- Joux Algorithm: about $2^{48}$ operations
- But about 1 PB of data $\implies$ Impractical
- Quad algorithm: with $|A| = |B| = |C| = 2^{n/3}$: about $2^{64}$ operations
- But only 206 GB of data $\implies$ Practical

$\implies$ Keep the lists small!
The Clamping Trick [Berstein07]

- **Idea:** Increase the number of queries to reduce the storage

\[ k \geq \frac{n}{3} \]

Discard vectors that do not start with \( \ell \) zeroes

Let \( n' = n - \ell \)

⇒ 3 lists \( A, B, C \) of \( 2^k - \ell = 2^{n'}/3 \) of \( n' \)-bits vectors

Solve the 3XOR problem over \( A, B, C \) with \(|A| \cdot |B| \cdot |C| = 2^{n'/2} \)

\( \ell = n/4, \quad n' = 3n/4 \)

Stored data: \( O(2^{n'/4}) \) words

Time Complexity: \( O(2^{n'/2}) \) with Quadratic Algorithm
The Clamping Trick [Berstein07]

- **Idea:** Increase the number of queries to reduce the storage
  - $2^k$ queries, $k \geq n/3$
  - $\ell$ s.t. $(n - \ell)/3 = k - \ell$
  - Discard vectors that do not start with $\ell$ zeroes
The Clamping Trick [Berstein07]

- **Idea:** Increase the number of queries to reduce the storage
- $2^k$ queries, $k \geq n/3$
- $\ell$ s.t. $(n - \ell)/3 = k - \ell$
- Discard vectors that do not start with $\ell$ zeroes
- Let $n' = n - \ell$
- $\Rightarrow$ 3 lists $A, B, C$ of $2^{k-\ell} = 2^{n'}/3$ of $n'$-bits vectors
- Solve the 3XOR problem over $A, B, C$ with $|A| \cdot |B| \cdot |C| = 2^{n'}$
The Clamping Trick [Berstein07]

- **Idea:** Increase the number of queries to reduce the storage
- \( 2^k \) queries, \( k \geq n/3 \)
- \( \ell \) s.t. \( \frac{n - \ell}{3} = k - \ell \)
- Discard vectors that do not start with \( \ell \) zeroes
- Let \( n' = n - \ell \)
- \( \Rightarrow \) 3 lists \( A, B, C \) of \( 2^{k-\ell} = 2^{n'/3} \) of \( n' \)-bits vectors
- Solve the 3XOR problem over \( A, B, C \) with \( |A| \cdot |B| \cdot |C| = 2^{n'} \)

\( 2^{n/2} \) Queries

- \( \ell = n/4, \ n' = 3n/4 \)
- **Stored data:** \( \mathcal{O}(2^{n/4}) \) words
- **Time Complexity:** \( \mathcal{O}(2^{n/2}) \) with Quadratic Algorithm
Our Work: A Generalization of Joux Algorithm

Generalization to any size of input lists
Our Work: A Generalization of Joux Algorithm

Generalization to any size of input lists

- Pick $n - k$ arbitrary entries in $C$ (the first ones)
Our Work: A Generalization of Joux Algorithm

Generalization to any size of input lists

- Pick $n - k$ arbitrary entries in $C$ (the first ones)
- Apply Joux’s Algorithm
Our Work: A Generalization of Joux Algorithm

Generalization to any size of input lists
- Pick $n - k$ arbitrary entries in $C$ (the first ones)
- Apply Joux’s Algorithm ($O(|A| + |B|)$)
Our Work: A Generalization of Joux Algorithm

Generalization to any size of input lists

- Pick $n - k$ arbitrary entries in $C$ (the first ones)
- Apply Joux’s Algorithm ($O(|A| + |B|)$)
- Re-iterate with $n - k$ other rows...
Our Work: A Generalization of Joux Algorithm

Generalization to any size of input lists

- Pick $n - k$ arbitrary entries in $C$ (the first ones)
- Apply Joux’s Algorithm ($O(|A| + |B|)$)
- Re-iterate with $n - k$ other rows...
Our Work: A Generalization of Joux Algorithm

Generalization to any size of input lists

- Pick $n - k$ arbitrary entries in $C$ (the first ones)
- Apply Joux’s Algorithm ($\mathcal{O}(|A| + |B|)$)
- Re-iterate with $n - k$ other rows...
- ... until all $C$ has been watched ($\approx \frac{|C|}{n-k}$ iterations)
Our Work: A Generalization of Joux Algorithm

Generalization to any size of input lists
- Pick $n - k$ arbitrary entries in $C$ (the first ones)
- Apply Joux’s Algorithm ($O(|A| + |B|)$)
- Re-iterate with $n - k$ other rows...
- ... until all $C$ has been watched ($\approx \frac{|C|}{n-k}$ iterations)

$$k = \log_2(\min(|A|, |B|))$$
$$\text{Time: } O \left( (|A| + |B|) \cdot \frac{|C|}{n} \right)$$
Our Work: A Generalization of Joux Algorithm

Generalization to any size of input lists

- Pick \( n - k \) arbitrary entries in \( C \) (the first ones)
- Apply Joux’s Algorithm \( O(|A| + |B|) \)
- Re-iterate with \( n - k \) other rows...
- ... until all \( C \) has been watched \( \approx \frac{|C|}{n-k} \) iterations

\[
|A| = |B| = |C| = 2^{n/3}; \; k = n/3, \; \text{Time: } O\left(\frac{2^{2n/3}}{n}\right)
\]
A Concrete Example

A 96-bit 3XOR

- Require $3 \cdot 2^{48}$ queries
A Concrete Example

A 96-bit 3XOR

- Require $3 \cdot 2^{48}$ queries
- Perform the clamping on 24 bits
A Concrete Example

A 96-bit 3XOR

- Require $3 \cdot 2^{48}$ queries
- Perform the clamping on 24 bits
- Process the lists on the first 64 bits of each entries (Find all solutions)
A Concrete Example

A 96-bit 3XOR

- Require $3 \cdot 2^{48}$ queries
- Perform the clamping on 24 bits
- Process the lists on the first 64 bits of each entries (Find all solutions)
- Test them on the remaining 8 bits (about 256 tests)
A Concrete Example

A 96-bit 3XOR

- Require $3 \cdot 2^{48}$ queries
- Perform the clamping on 24 bits
- Process the lists on the first 64 bits of each entries (Find all solutions)
- Test them on the remaining 8 bits (about 256 tests)
**Experimentations**

- 3XOR of 96 bits of SHA-256
- Tests performed on a Haswell Core i5 CPU

### Timing

<table>
<thead>
<tr>
<th></th>
<th>Quadratic</th>
<th>Our Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU hours</td>
<td>340</td>
<td>105</td>
</tr>
<tr>
<td>Data</td>
<td>576 MB</td>
<td>576 MB</td>
</tr>
</tbody>
</table>

Creation of the lists: × 100 slower than processing them!
Experimentations

- 3XOR of 96 bits of SHA-256
- Tests performed on a Haswell Core i5 CPU

### Timing

<table>
<thead>
<tr>
<th></th>
<th>Quadratic</th>
<th>Our Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU hours</td>
<td>340</td>
<td>105</td>
</tr>
<tr>
<td>Data</td>
<td>576 MB</td>
<td>576 MB</td>
</tr>
</tbody>
</table>

Creation of the lists: $\times 100$ slower than processing them!
In a Nutshell

This Algorithm...

- can be applied to any size of input list
- has a $\times n$ speed-up compared to the Quadratic Algorithm
- is about 3 times faster, in practice ($n = 96$)
- is faster than [NS14] with the same amount of data, in theory
- is the same than [Joux09] with the same amount of data
In a Nutshell

This Algorithm...

- can be applied to any size of input list
- has a $\times n$ speed-up compared to the Quadratic Algorithm
- is about 3 times faster, in practice ($n = 96$)
- is faster than [NS14] with the same amount of data, in theory
- is the same than [Joux09] with the same amount of data

Possible improvements

Find basis changes that increase the size of the sublists
- We propose two ways of doing this
- Only a constant time improvement in theory
A 3XOR Adaptation of [BDP05]

- Originally designed for the 3SUM Problem over \((\mathbb{Z}, +)\)
- We transposed it for the 3XOR Problem
A 3XOR Adaptation of [BDP05]

- Originally designed for the 3SUM Problem over $(\mathbb{Z}, +)$
- We transposed it for the 3XOR Problem

- Dispatch entries into buckets (according to the first $k$ bits)
- $A^u$: Bucket of elements of $A$ starting by $u$
- For each triplet $(A^u, B^v, C^{u\oplus v})$ perform constant time preliminary test
  - Test $s$-bit partial collision with a hash table
A 3XOR Adaptation of [BDP05]

- Originally designed for the 3SUM Problem over \((\mathbb{Z}, +)\)
- We transposed it for the 3XOR Problem

- Dispatch entries into buckets (according to the first \(k\) bits)
- \(A^u\): Bucket of elements of \(A\) starting by \(u\)
- For each triplet \((A^u, B^v, C^{u \oplus v})\) perform constant time preliminary test
  - Test \(s\)-bit partial collision with a hash table
- If the test fail: no solution for sure
- If the test succeed: there may be a solution
  - Solve the small instance
Preliminary Test

Instance \((A^u, B^v, C^{u \oplus v})\)

\[
A^u : \quad i \leftarrow \begin{array}{c} i_A \quad i_B \quad i_C \\
\end{array} \quad a_1 \quad a_m \\
\]

\[
T[i] = 1 \iff \exists j, k, \ell \text{ s.t. } a_j \oplus b_k \oplus c_\ell = 0
\]

\[
T[i] = 0 \Rightarrow \text{No solution in } (A^u, B^v, C^{u \oplus v})
\]
Discussion

BDP In Theory

When $n$ grows up to infinity, only one triplet passes the test $\implies$ complexity of the algorithm:

\[
\text{Time: } \mathcal{O}\left(\frac{2^{2n/3} \log^2(n)}{n^2}\right), \quad \text{Space: } \mathcal{O}\left(2^{n/3}\right)
\]
Discussion

**BDP In Theory**

When $n$ grows up to infinity, only one triplet passes the test

$\implies$ complexity of the algorithm:

\[
\text{Time: } \mathcal{O}\left(\frac{2^{2n/3} \log^2(n)}{n^2}\right), \quad \text{Space: } \mathcal{O}\left(2^{n/3}\right)
\]

**BDP In Practice**

$n = 96$, **machine words:** 64 bits

Expected size of a bucket: $m = 0.14$

$\implies$ Completely impractical
Conclusion

This work

- Discusses issues arising from the 3XOR problem
- Propose a new practical algorithm for the 3XOR problem, that is
  - \( n \times \) faster than the quadratic algorithm in theory
  - \( 3 \times \) faster than the quadratic algorithm in practice
- Propose an adaptation of \[BDP05\] algorithm that is
  - asymptotically faster than other algorithms
  - Totally impractical

What's Next?

Compute a 128-bit 3XOR on SHA-256
Expect to have the lists in about 2 years (using one Antminer S7)
Conclusion

This work

- Discusses issues arising from the 3XOR problem
- Propose a new practical algorithm for the 3XOR problem, that is
  - $n \times$ faster than the quadratic algorithm in theory
  - $3 \times$ faster than the quadratic algorithm in practice
- Propose an adaptation of [BDP05] algorithm that is
  - asymptotically faster than other algorithms
  - Totally impractical

What’s Next?

- Compute a 128-bit 3XOR on SHA-256
- Expect to have the lists in about 2 years (using one Antminer S7)
Code available here:
https://github.com/cbouilla/3XOR

Thank you for your time!