On Search Complexity of Discrete Logarithm

Shouqiao Wang[1], Hugo Bucquet[2]

December, 2022

[1]Columbia University, New York. shwang27@gsb.columbia.edu
[2]Columbia University, New York. hb2559@columbia.edu

What is discrete logarithm?

- Given set $S = \{1, 2, 3, 4, 5, 6\}$ and the binary operator $*$, we define $a * b = c$, if $c \equiv a * b \mod 7$.
- Suppose $g = 3$, we have

x	0	1	2	3	4	5
g^x	1	3	2	6	4	5

- Then, given g, we can define the discrete logarithm

x	1	2	3	4	5	6
$\log x$	0	2	1	4	5	3

Group and cyclic group

Definition: A group G is a non-empty set with a binary operation $*$ that satisfies

- Clsoed: For $\forall a, b \in G$, $a * b \in G$
- Associativity: For $\forall a, b, c \in G$, $(a * b) * c = a * (b * c)$
- Identity: $\exists e \in G$ s.t. for $\forall a \in G$, $e * a = a * e = a$
- Inverse: For $\forall a \in G$, $\exists b \in G$ s.t. $a * b = b * a = e$

Definition: G is a finite cyclic group if

- G is a group
- There exists generator $g \in G$, s.t. $G = \{e, g^1, \cdots, g^{n-1}\}$

Note that $\mathbb{Z}_p^* = \{1, 2, \cdots, p-1\}$ is always a finite cyclic group, if p is a prime number.
From previous example, 3 is a generator of $\mathbb{Z}_7^*$.

A more general example

Given a cyclic group G, a generator g and a target $t \in G$, how to search $x \in \mathbb{Z}$ s.t. $t = g^x$?

A more general example

Given a cyclic group G, a generator g and a target $t \in G$, how to search $x \in \mathbb{Z}$ s.t. $t = g^x$?

- Is this search problem in FNP? How to use polynomial time algorithm to calculate g^x?

A more general example

Given a cyclic group G, a generator g and a target $t \in G$, how to search $x \in \mathbb{Z}$ s.t. $t = g^x$?

- Is this search problem in FNP? How to use polynomial time algorithm to calculate g^x?
- Is this search problem Total? How can we verify G is indeed a cyclic group or g is indeed a generator?

Why FNP?

Given a cyclic group G, a generator g and a target $t \in G$, how to search $x \in \mathbb{Z}$ s.t. $t = g^x$?

- Is this search problem in FNP?

Why FNP?

Given a cyclic group G, a generator g and a target $t \in G$, how to search $x \in \mathbb{Z}$ s.t. $t = g^x$?

- Is this search problem in FNP?

Naive algorithm takes exponential time to calculate g^x.

Why FNP?

Given a cyclic group G, a generator g and a target $t \in G$, how to search $x \in \mathbb{Z}$ s.t. $t = g^x$?

- Is this search problem in FNP?

Naive algorithm takes exponential time to calculate g^x.
Thanks to the "repeated squaring" algorithm, it is in FNP!

Algorithm 1 Computation of the x-th power of the generator $g \in [s]$ of a groupoid $(\mathbb{G}, \star)$ of size $s \in \mathbb{N}$ induced by $f\colon \{0,1\}^{2\lceil \log(s) \rceil} \to \{0,1\}^{\lceil \log(s) \rceil}$ with identity $id \in [s]$.

procedure $\mathcal{I}_{\mathbb{G}}(x)$
    $(x_m, \ldots, x_1) \leftarrow \mathrm{bd}_0(x)$
    $r \leftarrow \mathrm{bd}(id)$
    $g \leftarrow \mathrm{bd}(g)$
    for i **from** m **to** 1 **do**
        $r \leftarrow f(r, r)$
        if $x_i = 1$ **then**
            $r \leftarrow f(g, r)$
        end if
    end for
    return $\mathrm{bc}(r)$
end procedure

$(x_m, \cdots, x_1)$ is the binary expression of x, $f(\cdot,\cdot)$ stands for the binary operator $*$

e.g. How to calculate g^{10}?

Why total?

Given a cyclic group G, a generator g and a target $t \in G$, how to search $x \in \mathbb{Z}$ s.t. $t = g^x$?

- Is this search problem Total?

Why total?

Given a cyclic group G, a generator g and a target $t \in G$, how to search $x \in \mathbb{Z}$ s.t. $t = g^x$?

- Is this search problem Total?

Imagine we are dealing with Z_p^* and a corresponding generator g. It is easy for us to prove Z_p^* is a cyclic group, but how can we verify g is a true generator?

DLOG_p

Definition: The search problem DLOG_p is defined via the following relation of instances and solutions:
Instance: Distinct primes $p, p_1, \cdots, p_n \in \mathbb{N}$, numbers $k_1, \cdots, k_n \in \mathbb{N}$, and $g, y \in \mathbb{Z}_p^*$ such that

1. $p - 1 = \prod_{i=1}^{n} p_i^{k_i}$
2. $g^{(p-1)/p_i} \neq 1$ for all $i \in \{1, \cdots, n\}$

Solution: An $x \in \{0, \cdots, p-2\}$ such that $g^x = y$

DLOG_p

Definition: The search problem DLOG_p is defined via the following relation of instances and solutions:

Instance: Distinct primes $p, p_1, \cdots, p_n \in \mathbb{N}$, numbers $k_1, \cdots, k_n \in \mathbb{N}$, and $g, y \in \mathbb{Z}_p^*$ such that

1. $p - 1 = \prod_{i=1}^{n} p_i^{k_i}$
2. $g^{(p-1)/p_i} \neq 1$ for all $i \in \{1, \cdots, n\}$

Solution: An $x \in \{0, \cdots, p-2\}$ such that $g^x = y$

Additionally providing the factorization of $p - 1$, we can verify whether g is indeed a generator. Why? [See next slide]

DLOG_p

Definition: The search problem DLOG_p is defined via the following relation of instances and solutions:

Instance: Distinct primes $p, p_1, \cdots, p_n \in \mathbb{N}$, numbers $k_1, \cdots, k_n \in \mathbb{N}$, and $g, y \in \mathbb{Z}_p^*$ such that

1. $p - 1 = \prod_{i=1}^{n} p_i^{k_i}$
2. $g^{(p-1)/p_i} \neq 1$ for all $i \in \{1, \cdots, n\}$

Solution: An $x \in \{0, \cdots, p-2\}$ such that $g^x = y$

Additionally providing the factorization of $p - 1$, we can verify whether g is indeed a generator. Why? [See next slide]

Theorem: $\mathrm{DLOG}_p \in$TFUP, i.e., subclass of TFNP with unique solution for every instance.

Proof of generator for $\mathbb{Z}_p^*$

Statement: For primes $p, p_1, \cdots, p_n \in \mathbb{N}$, if

1. $p - 1 = \prod_{i=1}^{n} p_i^{k_i}$
2. $g^{(p-1)/p_i} \neq 1$ for all $i \in \{1, \cdots, n\}$

then g is a generator for $\mathbb{Z}_p^*$.

Lemma 1: For $\forall g \in \{1, \cdots, p-1\}$, $g^{p-1} = 1$.
This is the celebrated Fermat's little theorem.
Lemma 2: For $a, b \in \mathbb{Z}^+$, if $g^a = g^b = 1$, then $g^d = 1$, where $d = gcd(a, b)$.
This can be proved by the Bézout's lemma.
Proof: If g is not a generator, then $\exists k < p - 1$ s.t. $g^k = 1$

- Suppose $d = gcd(k, p-1)$, we have $g^d = 1$
- $\exists$ prime p_i s.t. $p_i | \frac{p-1}{d}$, we suppose $p - 1 = tdp_i$
- Then $g^{(p-1)/p_i} = (g^d)^t = 1$. Contradiction!

What questions do we care about?

Intuition: The discrete logarithm problem is very important in cryptography. We also care about discrete logarithm for other groups, not only for Z_p^*.

What questions do we care about?

Intuition: The discrete logarithm problem is very important in cryptography. We also care about discrete logarithm for other groups, not only for Z_p^*.

Question: How can we form a TFNP problem just using a set G and a binary operator $f(\cdot, \cdot)$, without assuming any properties of G and $f(\cdot, \cdot)$ in advance?

What questions do we care about?

Intuition: The discrete logarithm problem is very important in cryptography. We also care about discrete logarithm for other groups, not only for Z_p^*.

Question: How can we form a TFNP problem just using a set G and a binary operator $f(\cdot,\cdot)$, without assuming any properties of G and $f(\cdot,\cdot)$ in advance?

Main Issue: Even for a simple cyclic group Z_p^*, if we only provide the set G and the binary operator $*$, we do not know how to verify a given generator g efficiently.

What questions do we care about?

Intuition: The discrete logarithm problem is very important in cryptography. We also care about discrete logarithm for other groups, not only for Z_p^*.

Question: How can we form a TFNP problem just using a set G and a binary operator $f(\cdot,\cdot)$, without assuming any properties of G and $f(\cdot,\cdot)$ in advance?

Main Issue: Even for a simple cyclic group Z_p^*, if we only provide the set G and the binary operator $*$, we do not know how to verify a given generator g efficiently.

Rough Idea: We can add more types of solutions to make the search problem total.

Overview

This paper [1] introduces two search problems for a general set G with a general binary operator $f(\cdot,\cdot)$ — INDEX and DLOG

For INDEX and DLOG problems, we are given

- Set $G = [s] = \{0, 1, 2, \cdots, s-1\}$, where $s \geq 2$
- Boolean circuit $f : \{0,1\}^{\log(s)} \times \{0,1\}^{\log(s)} \to \{0,1\}^{\log(s)}$
- Element $id \in [s]$, $g \in [s]$ and $t \in [s]$

Solution for INDEX:

- An element $x \in G$, s.t. $g^x = t$
- Some violation showing (G, f) cannot represents a group
- Some violation showing g is not a generator

Solution for DLOG contains all the cases of INDEX's solution, but adding two addional violations showing (G, f) cannot represents a group

Overview

Theorem: INDEX is PPP-complete

- PIGEON $\leq$ INDEX $\leq$ PIGEON

Theorem: DLOG is PWPP-complete

- Step 1: DLOG is PWPP-hard
  - WEAK-PIGEON $\leq$ DOVE $\leq$ DLOG
- Step 2: DLOG lies in PWPP
  - DLOG $\leq$ GEN-CLAW $\leq$ WEAK-PIGEON

WEAK-PIGEON $\leq$ DOVE $\leq$ DLOG $\leq$ GEN-CLAW $\leq$ WEAK-PIGEON
They are all PWPP-complete

Theorem: $\text{DLOG}_p \leq$ DLOG $=$ PWPP-complete
Conjecture: DLOG_p cannot be PWPP-complete

Definition of INDEX

We do not assume any property of the following variables

- Set $G = [s] = \{0, 1, 2, \cdots, s-1\}$, where $s \geq 2$
- Boolean circuit $f : \{0,1\}^l \times \{0,1\}^l$, where $l = \lceil \log(s) \rceil$
- Element $id \in [s]$, $g \in [s]$ and $t \in [s]$

Definition: The search problem INDEX is defined via the following relation

Instance: A tuple (s, f, id, g, t)

Solution: One of the following:

1. $x \in [s]$, s.t. $I_G(x) = t$
2. $x, y \in [s]$, s.t. $f(x, y) \geq s$
3. $x, y \in [s]$ $x \neq y$, s.t. $I_G(x) = I_G(y)$

Definition of INDEX

We do not assume any property of the following variables

- Set $G = [s] = \{0, 1, 2, \cdots, s-1\}$, where $s \geq 2$
- Boolean circuit $f : \{0,1\}^l \times \{0,1\}^l$, where $l = \lceil \log(s) \rceil$
- Element $id \in [s]$, $g \in [s]$ and $t \in [s]$

Definition: The search problem INDEX is defined via the following relation

Instance: A tuple (s, f, id, g, t)

Solution: One of the following:

1. $x \in [s]$, s.t. $I_G(x) = t$
2. $x, y \in [s]$, s.t. $f(x, y) \geq s$
3. $x, y \in [s]$ $x \neq y$, s.t. $I_G(x) = I_G(y)$

How to interpret these cases? Why total?

What is $I_G(x)$

Algorithm 1 Computation of the x-th power of the generator $g \in [s]$ of a groupoid $(\mathbb{G}, \star)$ of size $s \in \mathbb{N}$ induced by $f\colon \{0,1\}^{2\lceil \log(s) \rceil} \to \{0,1\}^{\lceil \log(s) \rceil}$ with identity $id \in [s]$.

procedure $\mathcal{I}_{\mathbb{G}}(x)$
    $(x_m, \ldots, x_1) \leftarrow \mathrm{bd}_0(x)$
    $r \leftarrow \mathrm{bd}(id)$
    $g \leftarrow \mathrm{bd}(g)$
    for i **from** m **to** 1 **do**
        $r \leftarrow f(r, r)$
        if $x_i = 1$ **then**
            $r \leftarrow f(g, r)$
        end if
    end for
    return $\mathrm{bc}(r)$
end procedure

To better understand $I_G(x)$, we show some properties of it:

- Suppose $f_0(r) = f(r,r)$ and $f_1(r) = f(g,r)$, the function $I_G(x)$ can correspond to an iterated composition of f_0 and f_1 evaluated on *id*
- e.g., $I_G(3) = I_G((101)_2) = f_1 \circ f_0 \circ f_0 \circ f_1 \circ f_0(id)$

We also use "repeated squaring" algorithm to define $I_G(x)$, but use f_0 and f_1 for squaring and multiplication by generator respectively.

INDEX lies in PPP

Theorem: INDEX $\leq$ PIGEON

INDEX lies in PPP

Theorem: INDEX $\leq$ PIGEON

Let $G = (s, f, id, g, t)$ be an arbitrary instance of INDEX. We construct a circuit $C : \{0,1\}^l \to \{0,1\}^l$ s.t. any solution to the PIGEON instance C gives a solution to INDEX instance G.

We construct C as follows

$$C(x) = \begin{cases} I_G(x) - t \mod s & \text{if } x \leq s, \\ x & \text{otherwise.} \end{cases}$$

INDEX lies in PPP

Theorem: INDEX $\leq$ PIGEON

Let $G = (s, f, id, g, t)$ be an arbitrary instance of INDEX. We construct a circuit $C : \{0,1\}^l \to \{0,1\}^l$ s.t. any solution to the PIGEON instance C gives a solution to INDEX instance G.

We construct C as follows

$$C(x) = \begin{cases} I_G(x) - t \mod s & \text{if } x \leq s, \\ x & \text{otherwise.} \end{cases}$$

Why it works?

- Given a preimage of 0, we can either find a solution of type 1, i.e., $x \in [s]$ s.t. $I_G(x) = t$, or $I_G(x) > s$, which gives a solution of type 2
- Given a collision, we can either find a solution of type 3, or a solution of type 2 (either $I_G(x) > s$ or $I_G(y) > s$)

INDEX is PPP-Hard

Theorem: PIGEON $\leq$ INDEX
(Too complicated, skip)

Definition of DLOG

Definition: The search problem DLOG is defined via the following relation of instances and solutions
Instance: A tuple (s, f, id, g, t)
Solution: One of the following:

1. $x \in [s]$, s.t. $I_G(x) = t$
2. $x, y \in [s]$, s.t. $f(x, y) \geq s$
3. $x, y \in [s]$ $x \neq y$, s.t. $I_G(x) = I_G(y)$
4. $x, y \in [s]$ $x \neq y$, s.t. $f(t, I_G(x)) = f(t, I_G(y))$
5. $x, y \in [s]$, s.t. $I_G(x) = f(t, I_G(y))$, $I_G(x - y \mod s) \neq t$

Properties of DLOG:

- Type 1,2,3 of solutions are sufficient to guarantee totality
- Type 4,5 of solutions make DLOG to lie in the class of PWPP and are crucial for correctness of the reduction from DLOG to WEAK-PIGEON

Alternative violation in DLOG

The last type of solution

$$x, y \in [s], s.t.\ I_G(x) = f(t, I_G(y)), I_G(x - y \mod s) \neq t$$

implies the violation of associativity. Why?

Alternative violation in DLOG

The last type of solution

$$x, y \in [s], s.t.\ I_G(x) = f(t, I_G(y)), I_G(x - y \mod s) \neq t$$

implies the violation of associativity. Why?

One could think about changing it to finding

$$x, y, z \in [s], s.t.\ f(x, f(y, z)) \neq f(f(x, y), z)$$

However, our proof of PWPP-hardness would fail for such alternative version of DLOG.

Open question: Does there exist an alternative version of DLOG, which is PWPP-complete?

Definition of DOVE

Definition: The search problem DOVE is defined via the following relation

Instance: A Boolean circuit $C : \{0,1\}^n \to \{0,1\}^n$

Solution: One of the following

1. $u \in \{0,1\}^n$, s.t. $C(u) = 0$
2. $u \in \{0,1\}^n$, s.t. $C(u) = 1$
3. $u, v \in \{0,1\}^n$ $u \neq v$, s.t. $C(u) = C(v)$
4. $u, v \in \{0,1\}^n$ $u \neq v$, s.t. $C(u) = C(v) \oplus 1$

Definition of DOVE

Definition: The search problem DOVE is defined via the following relation
Instance: A Boolean circuit $C : \{0,1\}^n \rightarrow \{0,1\}^n$
Solution: One of the following

1. $u \in \{0,1\}^n$, s.t. $C(u) = 0$
2. $u \in \{0,1\}^n$, s.t. $C(u) = 1$
3. $u, v \in \{0,1\}^n$ $u \neq v$, s.t. $C(u) = C(v)$
4. $u, v \in \{0,1\}^n$ $u \neq v$, s.t. $C(u) = C(v) \oplus 1$

Why total?

- Type 1 and 3 can guarantee the totality

DOVE(=PWPP-Complete)$\leq$PIGEON.

DOVE is reducible to DLOG

Theroem: DOVE$\leq$DLOG

DOVE is reducible to DLOG

Theroem: DOVE$\leq$DLOG

Let $C : \{0,1\}^n \to \{0,1\}^n$ be an arbitrary instance of DOVE. We construct an instance $G = (s, f, id, g, t)$ of DLOG s.t. any solution to G provides a solution to C.

DOVE is reducible to DLOG

Theroem: DOVE$\leq$DLOG

Let $C : \{0,1\}^n \to \{0,1\}^n$ be an arbitrary instance of DOVE. We construct an instance $G = (s, f, id, g, t)$ of DLOG s.t. any solution to G provides a solution to C.

We construct (s, f, id, g, t) as follows

- $s = 2^n, g = 0, id = 1, t = 1$
- The binary operator

$$f(x,y) = \begin{cases} C(x) & \text{if } x = y, \\ C(y \oplus 1) & \text{if } x = g \text{ and } y \neq g, \\ x \oplus y & \text{otherwise.} \end{cases}$$

Show any solution to this DLOG gives a solution to the instance C of DOVE. (Too complicated, skip)

WEAK-PIGEON is reducible to DOVE

Theorem: WEAK-PIGEON$\leq$DOVE

WEAK-PIGEON is reducible to DOVE

Theorem: WEAK-PIGEON$\leq$DOVE

For any arbitrary instance $C : \{0,1\}^n \to \{0,1\}^{n-1}$, we construct a circuit $V : \{0,1\}^{2n} \to \{0,1\}^{2n}$

$$V(x_1, \cdots, x_{2n}) = (C(x_1, \cdots, x_n), C(x_{n+1}, \cdots, x_{2n}), 1, 1)$$

Why any solution to V of DOVE gives a solution to the instance C of WEAK-PIGEON?

- We can never find a solution of type 1, 2 or 4, since the last two digits of $V(x_1, \cdots, x_{2n})$ are both 1
- Given a solution of type 3, which is a collision, we can get $C(x_{[1:n]}) = C(y_{[1:n]})$ and $C(x_{[n+1:2n]}) = C(y_{[n+1:2n]})$
- There is either $x_{[1:n]} \neq y_{[1:n]}$ or $x_{[n+1:2n]} \neq y_{[n+1:2n]}$, which is a collision for WEAK-PIGEON

Definition of CLAW

Definition: The search problem CLAW is defined via the following relation

Instance: A pair of Boolean circuits $h_0, h_1 : \{0,1\}^n \to \{0,1\}^n$

Solution: One of the following

1. $u, v \in \{0,1\}^n$, s.t. $h_0(u) = h_1(v)$
2. $u, v \in \{0,1\}^n$ $u \neq v$, s.t. $h_0(u) = h_0(v)$
3. $u, v \in \{0,1\}^n$ $u \neq v$, s.t. $h_1(u) = h_1(v)$

Definition of CLAW

Definition: The search problem CLAW is defined via the following relation
Instance: A pair of Boolean circuits $h_0, h_1 : \{0,1\}^n \to \{0,1\}^n$
Solution: One of the following

1. $u, v \in \{0,1\}^n$, s.t. $h_0(u) = h_1(v)$
2. $u, v \in \{0,1\}^n$ $u \neq v$, s.t. $h_0(u) = h_0(v)$
3. $u, v \in \{0,1\}^n$ $u \neq v$, s.t. $h_1(u) = h_1(v)$

Can we use CLAW as an intermediate problem to prove DLOG$\in$PWPP, i.e., prove DLOG$\leq$CLAW$\leq$WEAK-PIGEON?
It is very hard for us to prove DLOG$\leq$CLAW.
We choose GEN-CLAW as the intermediate problem!

Definition of GEN-CLAW

Definition: The search problem GEN-CLAW is defined via the following relation
Instance: A pair of Boolean circuits $h_0, h_1 : \{0,1\}^n \to \{0,1\}^n$ and $s \in \mathbb{Z}^+$, s.t. $1 \leq s < 2^n$
Solution: One of the following

1. $u, v \in \{0,1\}^n$, s.t. $u < s$ $v < s$ $h_0(u) = h_1(v)$
2. $u, v \in \{0,1\}^n$ $u \neq v$, s.t. $h_0(u) = h_0(v)$
3. $u, v \in \{0,1\}^n$ $u \neq v$, s.t. $h_1(u) = h_1(v)$
4. $u \in \{0,1\}^n$, s.t. $u < s$ $h_0(u) \geq s$
5. $u \in \{0,1\}^n$, s.t. $u < s$ $h_1(u) \geq s$

Definition of GEN-CLAW

Definition: The search problem GEN-CLAW is defined via the following relation
Instance: A pair of Boolean circuits $h_0, h_1 : \{0,1\}^n \to \{0,1\}^n$ and $s \in \mathbb{Z}^+$, s.t. $1 \leq s < 2^n$
Solution: One of the following

1. $u, v \in \{0,1\}^n$, s.t. $u < s$ $v < s$ $h_0(u) = h_1(v)$
2. $u, v \in \{0,1\}^n$ $u \neq v$, s.t. $h_0(u) = h_0(v)$
3. $u, v \in \{0,1\}^n$ $u \neq v$, s.t. $h_1(u) = h_1(v)$
4. $u \in \{0,1\}^n$, s.t. $u < s$ $h_0(u) \geq s$
5. $u \in \{0,1\}^n$, s.t. $u < s$ $h_1(u) \geq s$

Why proving DLOG$\leq$GEN-CLAW is easier?
The possible solutions to an instance of DLOG may not from $[2^n]$ but must lie in $[s]$.
DLOG is more related to GEN-CLAW!

DLOG is reducible to GEN-CLAW

Theorem: DLOG$\leq$GEN-CLAW

DLOG is reducible to GEN-CLAW

Theorem: DLOG$\leq$GEN-CLAW

For any arbitrary $G = (s, g, id, f, t)$ of DLOG, let $n = \lceil \log(s) \rceil$. We construct

- $h_0 : \{0,1\}^n \to \{0,1\}^n$

$$h_0(u) = \begin{cases} I_G(u) & \text{if } u < s, \\ u & \text{otherwise,} \end{cases}$$

- $h_1 : \{0,1\}^n \to \{0,1\}^n$

$$h_1(u) = \begin{cases} f(t, I_G(u)) & \text{if } u < s, \\ u & \text{otherwise.} \end{cases}$$

Show any solution to this instance of GEN-CLAW gives a solution to instance G of DLOG. (skip)

GEN-CLAW is reducible to WEAK-PIGEON

Theorem: GEN-CLAW$\leq$WEAK-PIGEON

GEN-CLAW is reducible to WEAK-PIGEON

Theorem: GEN-CLAW$\leq$WEAK-PIGEON

For any arbitrary instance (h_0, h_1, s) of GEN-CLAW, we construct a circuit $C : \{0,1\}^{n+1} \to \{0,1\}^n$ as follows

$$C(x) = h_{x_0} \circ h_{x_1} \circ \cdots \circ h_{x_n}(0),$$

where $x = (x_0, x_1, \cdots, x_n)_2$.

Show any solution to this instance C of WEAK-PIGEON gives a solution to instance (h_0, h_1, s) of GEN-CLAW. (skip)

CLAW is PWPP-complete

Theorem: WEAK-PIGEON$\leq$CLAW

CLAW is PWPP-complete

Theorem: WEAK-PIGEON$\leq$CLAW

For any instance of WEAK-PIGEON given by an arbitrary circuit $C : \{0,1\}^n \to \{0,1\}^{n-1}$, we construct an instance of CLAW as follows.

$$h_0(x) = C(x)0$$

and

$$h_1(x) = C(x)1$$

Why it works?

- It is impossible to find a solution of type 1
- Either a solution of type 2 or type 3 implies a collision for the WEAK-PIGEON problem

CLAW is PWPP-complete

Theorem: WEAK-PIGEON$\leq$CLAW

For any instance of WEAK-PIGEON given by an arbitrary circuit $C : \{0,1\}^n \to \{0,1\}^{n-1}$, we construct an instance of CLAW as follows.

$$h_0(x) = C(x)0$$

and

$$h_1(x) = C(x)1$$

Why it works?

- It is impossible to find a solution of type 1
- Either a solution of type 2 or type 3 implies a collision for the WEAK-PIGEON problem

It is obvious that CLAW$\leq$GEN-CLAW, and we already have GEN-CLAW is PWPP-complete. Hence, CLAW lies in PWPP.

Key Takeaways

Theorem: INDEX is PPP-complete
PIGEON $\leq$ INDEX $\leq$ PIGEON

Theorem: DLOG is PWPP-complete
WEAK-PIGEON $\leq$ DOVE $\leq$ DLOG $\leq$ GEN-CLAW $\leq$ WEAK-PIGEON
DLOG, DOVE, GEN-CLAW, CLAW are all PWPP-complete.

Theorem: $\text{DLOG}_p \leq \text{DLOG} = \text{PWPP-complete}$
Conjecture: DLOG_p cannot be PWPP-complete

Open Question: Does there exist an alternative version of DLOG, which is PWPP-complete?

Pavel Hubáček and Jan Václavek.
On search complexity of discrete logarithm.
arXiv preprint arXiv:2107.02617, 2021.