FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Jan Václavek

On search complexity of discrete logarithm

Computer Science Institute of Charles University

Supervisor of the master thesis:	Mgr. Pavel Hubáček, Ph.D.
Study programme:	Mathematics
Study branch:	Mathematical Structures

Prague 2021

I would like to thank my supervisor Pavel for his professional guidance, valuable advice and enthusiasm for the topic even in the unfavourable situation given by the current pandemic. I would also like to thank my family, my girlfriend and my friends for their support and care during my studies.

Title: On search complexity of discrete logarithm

Author: Jan Václavek

Institute: Computer Science Institute of Charles University

Supervisor: Mgr. Pavel Hubáček, Ph.D., Computer Science Institute of Charles University

Abstract: In this thesis, we study the discrete logarithm problem in the context of TFNP – the complexity class of search problems with a syntactically guaranteed existence of a solution for all instances. Our main results show that suitable variants of the discrete logarithm problem, which we call Index and DLog, are complete for the classes PPP and PWPP, respectively. Additionally, our reductions provide new structural insights into PWPP by establishing two new PWPP-complete problems. First, the problem Dove, a relaxation of the PPP-complete problem Pigeon. Dove is the first PWPP-complete problem not defined in terms of an explicitly shrinking function. Second, the problem Claw, a total search problem capturing the computational complexity of breaking claw-free permutations. In the context of TFNP, the PWPP-completeness of Claw matches the known intrinsic relationship between collision-resistant hash functions and claw-free permutations established in the cryptographic literature.

Keywords: discrete logarithm problem, TFNP, PPP, PWPP, complexity theory

Contents

Introduction

Computational complexity theory focuses on classifying computational problems according to their inherent computational complexity, which led to the birth of various complexity classes such as P and NP, to mention the most famous ones. In 1989, Megiddo and Papadimitriou [1] introduced the class TFNP, which contains search problems whose decision version lies in NP and which are total, i.e., where each instance is guaranteed to have a solution.

In order to improve our understanding of the seemingly disparate problems in TFNP, Papadimitriou in 1994 [2] suggested to classify the problems in TFNP according to the underlying combinatorial principles ensuring the existence of a solution. His approach proved to be extremely fruitful and it gave rise to various subclasses of TFNP such as PPP and PWPP that cluster many important total search problems from domains such as algorithmic game theory, computational number theory, and combinatorial optimization, to name but a few.

Following the classification rule based on the underlying combinatorial principle, the class PPP formalizes the pigeonhole principle. Concretely, it captures the complexity of finding a collision or a preimage of a zero element in a function from a set to itself and is defined as problems reducible to a problem called Pigeon.

Similarly, the class PWPP formalizes the weak pigeonhole principle, i.e., it captures the complexity of finding a collision in a shrinking function and is defined as problems reducible to a problem called Collision. As its definition suggests, the class PWPP has profound connections to cryptography as finding a collision in a shrinking function on average corresponds to breaking collision-resistant hash functions.

The discrete logarithm problem (DLP), which lies at the foundation of many practical schemes in modern cryptography, seems to naturally fit the TFNP landscape: given a generator g of a cyclic group $(\mathbb{G}, \star)$, we know that a solution x for DLP exists for any target element $t = g^x$ of the group $(\mathbb{G}, \star)$. Nevertheless, despite being a prominent search problem, DLP was not extensively studied in the context of TFNP so far. Only recently, in 2018, Sotiraki, Zampetakis, and Zirdelis [3] presented a total search problem motivated by DLP, which they called discrete logarithm problem over general groups and where the group is represented by a Boolean circuit. Since one cannot efficiently verify that the input instance really represents a valid group, an additional type of a solution in the form of distinct x, y such that $g^x = g^y$ was added to ensure the totality. They showed that the discrete logarithm problem over general groups lies in the complexity class PPP and asked and left open whether it is complete for that class.

In this thesis, we take a closer look at the DLP in the context of the classes PPP and PWPP. First, we observe that the discrete logarithm problem over general groups as defined in [3] is not total, which means it does not even lie in the class TFNP. Hence, we slightly modify the definition from [3] to make the discrete logarithm problem over general groups lie in the class PPP. Moreover, the discrete logarithm problem over general groups allows for remarkably unstructured instances being very far from a valid group, making the connection with DLP rather loose. Therefore, we refer to the modified problem as Index and call the underlying structure groupoid instead of general group.

As our first result, we answer the open problem from [3] in the affirmative as we show that Index is indeed PPP-complete. The corresponding reduction from Pigeon to Index showing that Index is PPP-hard is arguably the most technical part of the thesis. It involves a careful construction of the instance of Index such that the exponentiation g^x in the groupoid emulates the computation of the circuit C given by the instance of Pigeon.

Next, we focus on DLP in the context of the class PWPP. Given that DLP can be used to construct collision-resistant hash functions [4], it seems natural to think about modifications of Index such that the resulting variant lies in the class PWPP. Motivated by the known constructions of collision-resistant hash functions from DLP, which crucially rely on the homomorphic properties of the function g^x, we introduce additional types of a solution in the Index problem to enforce sufficient structure on the groupoid induced by the instance of Index. We refer to this new version of Index as DLog as it is, in our opinion, close enough to the standard DLP in cyclic groups.

We show that DLog indeed lies in the class PWPP. Since DLog is a relaxation of Index obtained by allowing additional types of a solution, it could be the case that DLog lies in PWPP simply because DLog is trivial. Nevertheless, we disprove this surmise as we prove that DLog is in fact PWPP-complete.

Additionally, our reductions showing that DLog is PWPP-complete give rise to two new PWPP-complete problems. The first one, Dove, is a relaxation of the PPP-complete problem Pigeon and, as far as we know, it is the first PWPP-complete problem not defined in terms of an explicitly shrinking function. The second one, Claw, is a search problem capturing the complexity of breaking claw-free pseudopermutations known from cryptographic literature [5].

We conclude by discussing some of the issues that arise when defining total search problems corresponding to actual problems in computational number theory. First, we highlight the distinction between DLog as defined in our thesis and the discrete logarithm problem in any specific group $\mathbb{Z}_p^*$. Second, we note that both our reductions establishing PWPP-hardness of DLog and PPP-hardness of Index result in instances that are far from being a valid group. In other words, the resulting instances do not really correspond to DLP in any group. Finally, we revisit the problem Blichfeldt introduced in [3] and show that it also exhibits a similar phenomenon.

1. Preliminaries

1.1 Notation

We denote by $[m]$ the set $\{0, 1, \dots, m-1\}$. We denote the natural numbers by $\mathbb{N}$, i.e., $\mathbb{N} = \{1, 2, 3, \dots\}$. We denote the non-negative integers by $\mathbb{Z}_0^+$, i.e., $\mathbb{Z}_0^+ = \{0, 1, 2, \dots\}$. For two strings $u, v \in \{0,1\}^*$, $u \,||\, v$ stands for the concatenation of u and v. When it is clear from the context, we omit the operator $||$, e.g., we write $0x$ instead of $0 \,||\, x$. The standard XOR function on binary strings of equal lengths is denoted by $\oplus$. All logarithms $\log()$ use base 2. Moreover, we set $\log(1) = 1$ for the purposes of this thesis in order to capture also the case when the size of the groupoid is one.[1]

Bit composition and decomposition. Throughout the paper, we often make use of bit composition and bit decomposition functions between binary strings of length k and the set $[2^k]$ of non-negative integers less then 2^k. We denote these functions bc^k and bd^k. Concretely, $\mathrm{bc}^k : \{0,1\}^k \to [2^k]$ and $\mathrm{bd}^k : [2^k] \to \{0,1\}^k$. Formally, for $x = x_1 x_2 \dots x_k \in \{0,1\}^k$, we define

$$\mathrm{bc}^k(x) = \sum_{i=0}^{k-1} x_{k-i} 2^i.$$

The function bc^k is bijective and we define the function bd^k as its inverse, i.e., for $a \in [2^k]$, $\mathrm{bd}^k(a)$ computes the unique binary representation of a with leading zeros such that its length is k. When k is known from the context and there is no risk of ambiguity, we omit k and write simply bc and bd to improve readability. Sometimes, we would like to have the output of bd^k without the leading zeros. We denote this modification by bd_0, which is independent of the length k, i.e., $\mathrm{bd}_0 : \mathbb{Z}_0^+ \to \{0,1\}^*$ is the standard function which computes the binary representation without the leading zeros.

1.2 Complexity Theory

We formally define the class FNP and its subclass TFNP, which was introduced in 1989 by Megiddo and Papadimitriou [1]. Consider a binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ such that

- there is a polynomial p such that $(x, y) \in R$ implies $|y| \leq |p(x)|$,
- there is a polynomial-time algorithm that on input x, y determines whether $(x, y) \in R$.

We say that such a relation R is *polynomially balanced polynomial-time recognizable*. The relation R defines the following search problem: given x, find y satisfying $(x, y) \in R$ if such y exists, and reply NO otherwise. We call this x an instance of the given search problem and y a solution to this instance x.

[1]We need $\lceil \log(m) \rceil$ bits to binary represent the set $[m]$.

Throughout the rest of the thesis, we use the notions of a relation and of the underlying search problem interchangeably. Now we can proceed with the definition of the class FNP.

Definition 1 (The class FNP)**.** *The class* FNP *is the class of all search problems defined by a polynomially balanced polynomial-time recognizable relation.*

Furthermore, we say that a polynomially balanced polynomial-time recognizable relation R is *total*, if for every x, there exists y such that $(x, y) \in R$. A search problem defined by a total relation is guaranteed to have a solution for each instance. We say that such a problem is total. Now we can proceed with the definition of the class TFNP, which is a subclass of FNP.

Definition 2 (The class TFNP)**.** *The class* TFNP *is the class of all search problems defined by a polynomially balanced polynomial-time recognizable total relation.*

There are different types of reductions between search problems, e.g., Karp reductions and Cook reductions. In our thesis, we are only interested in Karp reductions, which can be found also under the name *polynomial-time many-one reductions*. Hence, when we mention a reduction, we mean a Karp reduction as defined next. Moreover, we consider only problems from the class TFNP as needed for our thesis.

Definition 3 (Reduction between total search problems)**.** *Let $S, T \subseteq \{0,1\}^* \times \{0,1\}^*$ be total search problems. A reduction from S to T is a pair of polynomial-time computable functions $f, g\colon \{0,1\}^* \to \{0,1\}^*$ such that for all $x, y \in \{0,1\}^*$, if $(f(x), y) \in T$ then $(x, g(y)) \in S$.*

In words, the function f constructs an instance $f(x)$ of the problem T from the original instance x of the problem S and the function g computes a solution $g(y)$ to the original instance x from the solution y to the instance $f(x)$.

In case there exists a reduction from S to T, we say that *S is reducible to T*. In the rest of the thesis, we describe search problems less formally and in a more convenient way as pairs (*Instance*, *Solution*) instead of directly using relations as it is a common practice in complexity theory. The pair (*Instance*, *Solution*) then implicitly determines the underlying relation.

Similarly, in reductions, we typically first describe how the new instance $f(x)$ looks like based on the original instance x, which implicitly determines the function f. Then, we describe how to construct a solution $g(y)$ to the original instance x from the solution y to the new instance $f(x)$, which implicitly determines the function g. We need to make sure that both f, g are polynomial-time computable.

1.2.1 Subclasses of TFNP

In this subsection, we define the subclasses of TFNP called PPP and PWPP, which are relevant to our thesis. The class PPP is defined via a total search problem called Pigeon.

Definition 4 (Pigeon problem and PPP [2])**.**

Instance: A Boolean circuit C with n inputs and n outputs.

Solution: One of the following:
1. *a string $u \in \{0,1\}^n$ such that $\mathsf{C}(u) = 0^n$,*
2. *strings $u, v \in \{0,1\}^n$ such that $u \neq v$ and $\mathsf{C}(u) = \mathsf{C}(v)$.*

The class of all total search problems reducible to Pigeon is called PPP.

We say that a total search problem S is PPP-hard if there is a reduction from Pigeon to S. Moreover, we say that S is PPP-complete if it simultaneously lies in PPP and is PPP-hard.

The class PWPP is defined in similar fashion via a total search problem called Collision.

Definition 5 (Collision problem and PWPP [6])**.**

Instance: A Boolean circuit C with n inputs and m outputs with $m < n$.

Solution: Strings $u, v \in \{0,1\}^n$ such that $u \neq v$ and $\mathsf{C}(u) = \mathsf{C}(v)$.

The class of all total search problems reducible to Collision is called PWPP.

PWPP-hardness and completeness is defined analogously as for PPP, but using the problem Collision instead of Pigeon.

1.3 Groupoid structure

In this section, we describe the structure which we work with and use to define the search problems motivated by DLP. Our starting point is a *general group* as introduced in [3] in the context of the discrete logarithm problem over general groups.

Similarly to [3], given the order $s \in \mathbb{N}$, we consider a representation of a binary operation on $\mathbb{G} = [s]$ by a Boolean circuit $f\colon \{0,1\}^l \times \{0,1\}^l \to \{0,1\}^l$, where $l = \lceil \log(s) \rceil$. Given such a representation (s, f), we define a binary operator $f_{\mathbb{G}} : [s] \times [s] \to [2^l]$ for all $x, y \in [s]$ using the bit decomposition and composition functions as follows:

$$f_{\mathbb{G}}(x, y) = \mathrm{bc}(f(\mathrm{bd}(x), \mathrm{bd}(y))).$$

In words, the binary operator $f_{\mathbb{G}}$ first takes the binary representation of the elements $x, y \in \mathbb{G}$, then evaluates the circuit f on the resulting strings, and maps the value back to $[2^l]$. Observe that we let the possible outputs of $f_{\mathbb{G}}$ to be the whole $[2^l]$ and not just $[s]$ as one could assume. The reason is that we have no guarantee on the range of the outputs of the circuit f and it could happen that the output of $f_{\mathbb{G}}$ is outside of $[s]$.

We denote by $(\mathbb{G}, \star)$ the structure induced by f, where $\star\colon [s] \times [s] \to [s]$ is the binary operation closed on $[s]$ obtained by extending the operator $f_{\mathbb{G}}$ in some fixed way, e.g., by defining for all $x, y \in [s]$,

$$x \star y = \begin{cases} f_{\mathbb{G}}(x, y) & \text{if } f_{\mathbb{G}} \in [s], \\ 1 & \text{otherwise.} \end{cases} \tag{1.1}$$

Algorithm 1 Computation of the x-th power of the generator $g \in [s]$ in 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 the identity $id \in [s]$.

```
1: procedure $\mathcal{I}_{\mathbb{G}}(x)$
2:     $(x_1, \ldots, x_m) \leftarrow \mathrm{bd}_0(x)$
3:     $r \leftarrow \mathrm{bd}(id)$
4:     $g \leftarrow \mathrm{bd}(g)$
5:     for $i$ from 1 to $m$ do
6:         $r \leftarrow f(r, r)$
7:         if $x_i = 1$ then
8:             $r \leftarrow f(g, r)$
9:         end if
10:    end for
11:    return $\mathrm{bc}(r)$
12: end procedure
```

Since the binary operation $\star$ induced on $\mathbb{G}$ by f in this way might not satisfy the group axioms, we decided to refer to $(\mathbb{G}, \star)$ as the induced *groupoid* instead of *general group* adopting the terminology for a set with a binary operation common in universal algebra.

We stress that the second case in the equation 1.1 could be defined arbitrarily and has no impact on the problems defined in the thesis, but only justifies the name *groupoid* as we need that $\star$ is closed. Skipping ahead, in the corresponding computational problems based on the induced groupoid, we make use of the operator $f_{\mathbb{G}}$ instead of directly working with the binary operation $\star$ and one type of a solution would correspond to finding x, y such that $f_{\mathbb{G}}(x, y) \notin [s]$.

1.3.1 Index function

If the induced groupoid $(\mathbb{G}, \star)$ was a cyclic group, then it is natural to consider the indices of the identity element $id \in [s]$ and of a generator $g \in [s]$ of $\mathbb{G}$. Moreover, we could use g to index the elements of the group $(\mathbb{G}, \star)$, e.g, in the order of increasing powers of g, where the corresponding *index function* $\mathcal{I}_{\mathbb{G}}\colon [s] \to [2^l]$ would on input x return simply the x-th power of the generator g. We fix a canonical way of computing the x-th power using the standard square-and-multiply method as defined in Algorithm 1, where $(x_1, x_2, \ldots, x_m) = \mathrm{bd}_0(x)$ is the binary representation of x without the leading zeros with $m \leq l$.

We can observe that, by the definition of $\mathcal{I}_{\mathbb{G}}$, the circuit f is only applied on specific types of inputs during the computation of $\mathcal{I}_{\mathbb{G}}(x)$:

- In each loop, $f(r, r)$ is computed for some $r \in \{0,1\}^l$. In the rest of the thesis, we denote f restricted to this type of inputs by f_0, i.e., $f_0(r) = f(r, r)$. Since the binary group operation is defined by f, f_0 corresponds to squaring in $\mathbb{G}$.

- If the corresponding bit of x is one in the given loop, then also $f(g, r)$ is computed with a fixed $g \in \{0,1\}^l$ and some $r \in \{0,1\}^l$. In the rest of the thesis, we denote f restricted to this type of inputs by f_1, i.e., $f_1(r) = f(g, r)$. This corresponds to multiplication by g in $\mathbb{G}$.

Hence, using the above notation, the computation of $\mathcal{I}_{\mathbb{G}}(x)$ simply corresponds to an iterated composition of functions f_0 and f_1 depending on the binary representation $\mathrm{bd}_0(x)$ of x evaluated on id.

Example. $\mathcal{I}_{\mathbb{G}}(5) = \mathcal{I}_{\mathbb{G}}(\mathrm{bc}(101)) = \mathrm{bc}(f_1 \circ f_0 \circ f_0 \circ f_1 \circ f_0(\mathrm{bd}(id)))$.

We point out that the index function $\mathcal{I}_{\mathbb{G}}$ is well-defined for an arbitrary induced groupoid with arbitrary indices $id, g \in [s]$ and not only for cyclic groups. That is also the reason why we need to consider the outputs of $\mathcal{I}_{\mathbb{G}}$ to be the whole $[2^l]$ as we have no guarantee on the outputs of f similarly as in the definition of $f_{\mathbb{G}}$. Nevertheless, for a general induced groupoid, the index function $\mathcal{I}_{\mathbb{G}}$ is not necessarily a bijection.

2. Index is PPP-complete

In [3], a total search problem associated with DLP was presented. In this chapter, we define Index, a slightly modified variant of the problem from [3], and show that it is PPP-complete. We start with the definition of Index.

Definition 6 (Index problem)**.**

> *Instance: A size parameter $s \in \mathbb{N}$ and a Boolean circuit $f\colon \{0,1\}^{2\lceil \log(s) \rceil} \to \{0,1\}^{\lceil \log(s) \rceil}$ representing a groupoid $(\mathbb{G}, \star)$ and indices $g, id, t \in [s]$.*
>
> *Solution: One of the following:*
> 1. *$x \in [s]$ such that $\mathcal{I}_{\mathbb{G}}(x) = t$,*
> 2. *$x, y \in [s]$ such that $f_{\mathbb{G}}(x, y) \geq s$,*
> 3. *$x, y \in [s]$ such that $x \neq y$ and $\mathcal{I}_{\mathbb{G}}(x) = \mathcal{I}_{\mathbb{G}}(y)$.*

Compared to the problem from [3], we added the second type of a solution to make the problem indeed lie in PPP. Without the second type of a solution, the problem is not even total: Skipping ahead, it follows from the construction in Section 2.2 that there exists a representation (s, f, id, g) of an induced groupoid with $s = 2^n - 1$ for which it holds that $\mathcal{I}_{\mathbb{G}}(x) = x + 1$ for each $x \in [s]$. We can see that on $[s]$, such $\mathcal{I}_{\mathbb{G}}$ is injective and avoids the element 0. Hence, if we set $t = 0$, then the underlying instance (s, f, id, g, t) of Index has no solutions of the first and third type, meaning that the version from [3] is not total.

On a high level, the first type of a solution corresponds to the discrete logarithm of t. Since one cannot efficiently verify that the input instance really represents a valid group, additional types of a solution had to be added in order to guarantee that Index is total. Hence, these remaining types of a solution are witnesses that the instance does not represent a valid group, since for a valid group, these two types of a solution cannot happen. The main result of this chapter establishes PPP-completeness of Index, which we prove in the rest of this chapter.

Theorem 1. *Index is PPP-complete.*

We show that Index lies in PPP in Section 2.1, and we show that Index is PPP-hard in Section 2.2.

2.1 Index lies in PPP

To show that Index lies in PPP, we need to show a reduction from Index to Pigeon. The main idea of our reduction from Index to Pigeon is analogous to the reduction in [3] from their discrete logarithm problem over general groups to Pigeon. Although, we need to handle the additional second type of a solution for Index, which corresponds to $f_{\mathbb{G}}$ outputting an element outside $\mathbb{G}$.

Lemma 2. *Index is reducible to Pigeon.*

Proof. Let (s, f, id, g, t) be an arbitrary instance of Index. Then we know that $\mathcal{I}_{\mathbb{G}} : [s] \to [2^l]$, where $l = \lceil \log(s) \rceil$. We construct a circuit $\mathsf{C} : \{0,1\}^l \to \{0,1\}^l$ as follows:

$$\mathsf{C}(x) = \begin{cases} \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(x)) - t \bmod s) & \text{if } \mathrm{bc}(x) < s, \\ x & \text{otherwise.} \end{cases}$$

The construction is valid since the new circuit C can be constructed in polynomial time with respect to the size of the original instance of Index. We show that any solution to the above Pigeon instance C gives a solution to the original Index instance. There are two possible cases:

1. The solution to C is $x \in \{0,1\}^l$ such that $\mathsf{C}(x) = 0^l$. Then, from the definition of the circuit C, it holds that $\mathsf{C}(x) = \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(x)) - t \bmod s) = 0^l$ and $\mathrm{bc}(x) < s$. Since the function bd is bijective and $\mathrm{bd}(0) = 0^l$, it must hold that $\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(x)) - t \bmod s = 0$. If $\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(x)) \geq s$, then, from the definition of the function $\mathcal{I}_{\mathbb{G}}$, we can find in polynomial time $u, v \in [s]$ such that $f_{\mathbb{G}}(u, v) \geq s$. Hence, these u, v form a solution to the original Index instance, case 2. Otherwise, $\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(x)) < s$, and since $t \in [s]$, i.e., $t < s$, it must be that $\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(x)) = t$. Hence, $\mathrm{bc}(x)$ is a solution to the original Index instance, case 1.

2. The solution to C is a pair $x, y \in \{0,1\}^l$ such that $x \neq y$ and $\mathsf{C}(x) = \mathsf{C}(y)$. Then, from the definition of the circuit C and the bijectivity of bd, it must hold that $\mathrm{bc}(x) < s, \mathrm{bc}(y) < s$ and

   $$\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(x)) - t \equiv \mathcal{I}_{\mathbb{G}}(\mathrm{bc}(y)) - t \pmod{s},$$

   which implies that

   $$\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(x)) \equiv \mathcal{I}_{\mathbb{G}}(\mathrm{bc}(y)) \pmod{s}.$$

   If $\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(x)) \geq s$, then, similarly as in the previous case, we can find $u, v \in [s]$ such that $f_{\mathbb{G}}(u, v) \geq s$, i.e., a solution to the original Index instance, case 2. We proceed analogously if $\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(y)) \geq s$. Otherwise, both $\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(x)), \mathcal{I}_{\mathbb{G}}(\mathrm{bc}(y)) \in [s]$, hence

   $$\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(x)) = \mathcal{I}_{\mathbb{G}}(\mathrm{bc}(y)).$$

   Moreover, from $x \neq y$ and the bijectivity of bc, we get that $\mathrm{bc}(x) \neq \mathrm{bc}(y)$. Hence, the pair $\mathrm{bc}(x), \mathrm{bc}(y)$ is a solution to the original Index instance, case 3.

□

2.2 Index is PPP-hard

In this section, we show that Index is PPP-hard, which is arguably the most technical part of the thesis. First, we provide a high-level overview of the core ideas of our reduction before proceeding with the formal proof.

Reducing Pigeon to Index. Given an instance $\mathsf{C} : \{0,1\}^n \to \{0,1\}^n$ of Pigeon, the main idea is to construct an instance $G = (s, f, id, g, t)$ of Index in which the index function $\mathcal{I}_{\mathbb{G}}$ "emulates" the computation of the circuit C, so that any solution to G provides a solution to the original instance C of Pigeon. In order to achieve this, we exploit the structure of the computation induced by $\mathcal{I}_{\mathbb{G}}$ in terms of evaluations of the circuit f. Specifically, the computation of $\mathcal{I}_{\mathbb{G}}$ gives rise to a tree labeled by the values output by $\mathcal{I}_{\mathbb{G}}$ and structured by two special types of calls to f denoted by f_0 and f_1 (see section 1.3.1 for definition of f_0 and f_1).

Our reduction constructs f inducing $\mathcal{I}_{\mathbb{G}}$ with the computation corresponding to a sufficiently large such tree so that its leaves can represent all the possible inputs for the instance C of Pigeon and the induced index function $\mathcal{I}_{\mathbb{G}}$ can output the corresponding evaluation of C at each leaf. Moreover, for the remaining nodes in the tree, $\mathcal{I}_{\mathbb{G}}$ results in a bijection in order to ensure there are no additional solutions to the Index instance that would be unrelated to the original instance C of Pigeon.

First, we start by describing two constructions of an induced groupoid $(\mathbb{G}, \star)$ which are independent of the instance C of Pigeon but which will serve as a step towards our reduction.

In the first construction, we define f_0 and f_1 and the elements $id, g \in [s]$ such that $\mathcal{I}_{\mathbb{G}}$ is the identity function, i.e., such that $\mathcal{I}_{\mathbb{G}}(a) = a$ for all $a \in [s]$. Our key observation is that for all $a, b \in [s]$ such that either

– $\mathrm{bd}_0(a)$ is a prefix of $\mathrm{bd}_0(b)$

or

– $\mathrm{bd}_0(a) = y0$ and $\mathrm{bd}_0(b) = y1$ for some $y \in \{0,1\}^*$,

the computation of $\mathcal{I}_{\mathbb{G}}(b)$ includes the whole computation of $\mathcal{I}_{\mathbb{G}}(a)$ as a prefix (see Algorithm 1). Specifically, if $\mathrm{bd}_0(a) = y0$ then

$$\mathcal{I}_{\mathbb{G}}(a) = \mathrm{bc}(f_0(\mathrm{bd}(\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(y))))), \tag{2.1}$$

and if $\mathrm{bd}_0(a) = y1$ then

$$\mathcal{I}_{\mathbb{G}}(a) = \mathrm{bc}(f_1(\mathrm{bd}(\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(y0))))). \tag{2.2}$$

The Equation (2.1) and Equation (2.2) are a key observation which allows us to construct the tree capturing the computation of $\mathcal{I}_{\mathbb{G}}$ successively based on the length of $\mathrm{bd}_0(a)$ of any input value a. We set the parameters inducing the groupoid $(\mathbb{G}, \star)$ such that $\mathcal{I}_{\mathbb{G}}(0) = 0$ and then we use Equation (2.1) and Equation (2.2) to define f_0 and f_1 such that $\mathcal{I}_{\mathbb{G}}(a) = a$ for all $a \in [s]$. In Figure 2.1, we illustrate the corresponding tree induced by the computation of $\mathcal{I}_{\mathbb{G}}$ resulting in this construction for a groupoid of order $s = 16$. Solid lines correspond to the application of f_0 and dashed lines to the application of f_1. For each node, the second value is the input a, which in this case equals also the output value $\mathcal{I}_{\mathbb{G}}(a)$, and the first value is $\mathrm{bd}(a)$, i.e., the binary representation of a with the leading zeros.

Then, the second construction is obtained by modifying the first one so that it holds for all $a \in [s]$ that $\mathcal{I}_{\mathbb{G}}(a) = a + b \bmod 2^{\lceil \log(s) \rceil}$ for some fixed $b \in [2^{\lceil \log(s) \rceil}]$.

This can be performed simply by carefully “shifting” the computation of f_0 and f_1 by b.

Finally, we consider the second construction with $s = 2^{n+2}$ and $b = 2^n$ and we make additional adjustments using the instance C of PIGEON such that any solution to the new instance G of INDEX produces a solution to the original instance C. First, we set the target to $t = 0$. Then, we adjust the definition of f_1 such that $\mathcal{I}_{\mathbb{G}}(a) = \mathrm{bc}(\mathsf{C}(h(a)))$ for values a from a suitable set $A_o \subseteq [s]$ of size 2^n and some bijective function h. We set

$$A_o = \{a : \mathrm{bd}^{n+2}(a) = 1y1\}$$

since for these values a, the computation of $\mathcal{I}_{\mathbb{G}}(a)$ is not included in the computation of $\mathcal{I}_{\mathbb{G}}(b)$ for any b, i.e., these values correspond to leaves in the computational tree of $\mathcal{I}_{\mathbb{G}}$. Hence, modifying f_1 such that $\mathcal{I}_{\mathbb{G}}(a) = \mathrm{bc}(\mathsf{C}(h(a)))$ for $a \in A_o$ does not break the relationship $\mathcal{I}_{\mathbb{G}}(a) = a + b$ for any $a \in [s] \setminus A_o$.

Finally, we use another set $A_e \subseteq [s]$ and we modify the definition of f_0 such that $\mathcal{I}_{\mathbb{G}}$ restricted to $[s] \setminus A_o$ is a bijection between $[s] \setminus A_0$ and $[s] \setminus [2^n]$, i.e., it avoids values corresponding to bit compositions of elements in the range of the circuit C. On the other hand, $\mathcal{I}_{\mathbb{G}}$ restricted to A_o produces values in $[2^n]$. Hence, any collision in $\mathcal{I}_{\mathbb{G}}$ corresponds to a collision in C and the discrete logarithm of t corresponds to a preimage of 0^n in C.

We give the formal proof below.

Lemma 3. *PIGEON is reducible to INDEX.*

Proof of Lemma 3. For the purpose of this proof and only for $s = 2^m$, we denote by $\mathcal{I}'_{\mathbb{G}} : \{0,1\}^m \to \{0,1\}^m$ the function

$$\mathcal{I}'_{\mathbb{G}}(u) = \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(u))),$$

where $\mathcal{I}_{\mathbb{G}}$ is induced by a representation (s, f) of a groupoid $(\mathbb{G}, \star)$ and elements $id, g \in [s]$.

First, we describe two concrete constructions of a representation of an induced groupoid independent of any instance of PIGEON, but which we leverage in our reduction. Recall the terminology that $f_0(r) = f(r, r)$ and $f_1(r) = f(g, r)$ for all $r \in \{0,1\}^m$. Additionally, recall that $\mathcal{I}_{\mathbb{G}}$ and $\mathcal{I}'_{\mathbb{G}}$ are fully determined by f_0 and f_1 as discussed in Section 1.3.1.

For the first construction, we show how to define f_0 and f_1 such that for all $v \in \{0,1\}^m$,

$$\mathcal{I}'_{\mathbb{G}}(v) = v, \tag{2.3}$$

which is equivalent to $\mathcal{I}_{\mathbb{G}}(a) = a$ for all $a \in [2^m]$. Moreover, as it will be clear from the construction, it is enough to define f_0 and f_1 only on some subset of its potential inputs. We set $s = 2^m$ and $id = 0$. For all $y \in \{0,1\}^{m-1}$, we define

$$f_0(0y) = y0 \quad \text{and} \quad f_1(y0) = y1.$$

In other words, f_0 shifts the input string by one position to the left and f_1 changes the last bit of the input from 0 to 1. Equivalently, if we interpret functions f_0 and f_1 as functions on the corresponding integers, then f_0 represents multiplying the input by two and f_1 represents adding one to the input. We show that it is enough to define f_0 and f_1 only on the inputs of the above special form to determine the whole computation of functions $\mathcal{I}_{\mathbb{G}}$ and $\mathcal{I}'_{\mathbb{G}}$.

We prove that Equation (2.3) holds for all $v \in \{0,1\}^m$ by induction on the length of $\mathrm{bd}_0(\mathrm{bc}(v))$, i.e., on the length of v without the leading zeros. For $v = 0^m$, it holds that

$$\mathcal{I}'_{\mathbb{G}}(v) = f_0(\mathrm{bd}(id)) = f_0(\mathrm{bd}(0)) = f_0(0^m) = 0^m = v,$$

so Equation (2.3) holds. We first show the inductive step for all v of the form $v = v'0$ and then for all v of the form $v = v'1$. For $v = v'0$, we have that

$$\mathcal{I}'_{\mathbb{G}}(v) = f_0(\mathcal{I}'_{\mathbb{G}}(0v')) = f_0(0v') = v'0 = v,$$

where the first equality follows from the definition of $\mathcal{I}'_{\mathbb{G}}$, the second one from the inductive hypothesis since the length of $\mathrm{bd}_0(\mathrm{bc}(0v'))$ is smaller than the length of $\mathrm{bd}_0(\mathrm{bc}(v'0))$, and the third one from the definition of f_0. Hence, Equation (2.3) holds. Similarly, for $v = v'1$, we have that

$$\mathcal{I}'_{\mathbb{G}}(v) = f_1(\mathcal{I}'_{\mathbb{G}}(v'0)) = f_1(v'0) = v'1 = v,$$

where the first equality follows from the definition of $\mathcal{I}'_{\mathbb{G}}$, the second one from the inductive hypothesis since the case for $v'0$ was already proved, and the third one from the definition of f_1. Thus, for all $v \in \{0,1\}^m$, Equation (2.3) holds.

Figure 2.1 illustrates the tree corresponding to the computation of $\mathcal{I}_{\mathbb{G}}$ induced by the above construction of (s, f, id, g) for $s = [16]$. Solid lines correspond to applications of f_0 and dashed lines to applications of f_1. For each node, the second value is the input a to $\mathcal{I}_{\mathbb{G}}$, which in this case equals also the output $\mathcal{I}_{\mathbb{G}}(a)$, and the first value is $\mathrm{bd}(a)$, i.e., the binary representation of a with the leading zeros.

Now, we show how to adjust the above construction to define f'_0 and f'_1 such that for a given fixed $w \in \{0,1\}^m$ and for all $v \in \{0,1\}^m$,

$$\mathcal{I}'_{\mathbb{G}}(v) = v + w, \tag{2.4}$$

where $\mathcal{I}'_{\mathbb{G}}$ is now determined by f'_0 and f'_1, and by $v + w$ we denote $\mathrm{bd}(\mathrm{bc}(v) + \mathrm{bc}(w) \bmod 2^m)$, i.e., the standard addition with the potential carry being ignored. Observe that Equation (2.4) is equivalent to $\mathcal{I}_{\mathbb{G}}(a) = a + b \bmod 2^m$ for a fixed $b = \mathrm{bc}(w) \in [2^m]$ and all $a \in [2^m]$. We implement this property by shifting the whole computation by w. To do so, we first change the identity element to $id = \mathrm{bc}(w)$. In the computation of f'_0 and f'_1, we first subtract w then apply the original f_0 or f_1 to the result, and, finally shift it back by adding w.

Formally, we define for all $r \in \{0,1\}^m$,

$$f'_0(r) = f_0(r - w) + w \quad \text{and} \quad f'_1(r) = f_1(r - w) + w,$$

where $r - w$ is defined in the same manner as the addition, i.e., $r - w = \mathrm{bd}(\mathrm{bc}(r) - \mathrm{bc}(w) \bmod 2^m)$. We show that for all $v \in \{0,1\}^m$, Equation (2.4) holds by induction on the length of $\mathrm{bd}_0(\mathrm{bc}(v))$ similarly as for Equation (2.3). For $v = 0^m$, we have that

$$\mathcal{I}'_{\mathbb{G}}(v) = f'_0(\mathrm{bd}(id))) = f'_0(w) = f_0(w - w) + w = f_0(0^m) + w = 0^m + w = w,$$

so Equation (2.4) holds. We first show the inductive step for all v such that $v = v'0$ and then for all v such that $v = v'1$. For $v = v'0$, we have that

$$\begin{aligned}\mathcal{I}'_{\mathbb{G}}(v) &= f'_0(\mathcal{I}'_{\mathbb{G}}(0v')) = f'_0(0v' + w) = f_0(0v' + w - w) + w \\ &= f_0(0v') + w = v'0 + w = v + w,\end{aligned}$$

where the first equality comes from the definition of $\mathcal{I}'_{\mathbb{G}}$, the second one from the inductive hypothesis, the third one from the definition of f'_0, and the fifth one from the definition of f_0. Hence, Equation (2.4) holds. Similarly, for $v = v'1$, we have that

$$\begin{aligned}\mathcal{I}'_{\mathbb{G}}(v) &= f'_1(\mathcal{I}'_{\mathbb{G}}(v'0)) = f'_1(v'0 + w) = f_1(v'0 + w - w) + w \\ &= f_1(v'0) + w = v'1 + w = v + w,\end{aligned}$$

for analogous reasons as before. This concludes the proof that Equation (2.4) holds for all $v \in \{0,1\}^m$.

We can now proceed with the reduction from Pigeon to Index. Let $\mathsf{C} : \{0,1\}^n \to \{0,1\}^n$ be an arbitrary instance of Pigeon. We construct an instance $G = (s, f, id, g, t)$ of Index such that any solution to G gives a solution to the original instance C of Pigeon. We utilize the second construction with $m = n+2$. The rest of the proof is the only place in the thesis, where the function bd^k is used for two values of k, concretely for $n+2$ and n. We shorten bd^{n+2} as bd, whereas we use the full name bd^n for the second case. In the rest of the proof, we denote by $\mathbb{Z}_{even}$ the subset of $\mathbb{Z}$ consisting of even integers and, analogously, by $\mathbb{Z}_{odd}$ we denote the subset of odd integers.

We set $s = 2^{n+2}, g = 2^{n+2} - 1, id = 2^n$ and $t = 0$. The idea is to define f such that

$$\mathcal{I}_{\mathbb{G}}(a) = \begin{cases} a + 2^n & \text{if } a \in [2^{n+1}], \\ 2^{n+1} + \frac{a}{2} & \text{if } a \in [2^{n+1}, \dots, 2^{n+2}-1] \cap \mathbb{Z}_{even} =: A_e, \\ \mathrm{bc}(\mathsf{C}(\mathrm{bd}^n(\frac{a-1}{2} - 2^n))) & \text{if } a \in [2^{n+1}, \dots, 2^{n+2}-1] \cap \mathbb{Z}_{odd} =: A_o. \end{cases} \tag{2.5}$$

For the case $n = 2$, we illustrate the structure of the computation corresponding to $\mathcal{I}_{\mathbb{G}}$ satisfying Equation (2.5) in Figure 2.2. Nodes with a solid edge correspond to the set $[2^{n+1}] = [8]$, nodes with a dashed edge correspond to A_e and nodes with a dotted edge correspond to A_o. The label of each node equals the value $\mathcal{I}_{\mathbb{G}}(a)$, where a is the second value of the node at the same position in the tree in Figure 2.1.

Suppose that we can define the circuit f such that the induced index function $\mathcal{I}_{\mathbb{G}}$ satisfies Equation (2.5). Then $[s] = [2^{n+2}] = A_o \mathbin{\dot\cup} ([2^{n+1}] \mathbin{\dot\cup} A_e)$, where $\dot\cup$ denotes the disjoint union operation. By Equation (2.5), we get that $\mathcal{I}_{\mathbb{G}}$ restricted to $[2^{n+1}] \mathbin{\dot\cup} A_e$ is a bijection between $[2^{n+1}] \mathbin{\dot\cup} A_e$ and $[2^n, \dots, 2^{n+2}-1]$. Moreover, $\mathcal{I}_{\mathbb{G}}$ restricted to A_o outputs only values in $[2^n]$. Hence, any collision in $\mathcal{I}_{\mathbb{G}}$ or any preimage of $t = 0$ under $\mathcal{I}_{\mathbb{G}}$ can happen only for values from A_o. Furthermore, for values from A_o, the output of the function $\mathcal{I}_{\mathbb{G}}$ corresponds to the output of the circuit C, so any collision in $\mathcal{I}_{\mathbb{G}}$ or any preimage of t under $\mathcal{I}_{\mathbb{G}}$ give us a solution to the original instance C of Pigeon. Formally, the oracle solving the above instance of Index returns one of the following:

1. $u \in [s]$ such that $\mathcal{I}_{\mathbb{G}}(u) = t = 0$. From the above discussion, we know it must be the case that $u \in A_o$. Hence, from Equation (2.5), we get that

$$\begin{aligned}0^n &= \mathrm{bd}^n(0) = \mathrm{bd}^n(t) = \mathrm{bd}^n(\mathcal{I}_{\mathbb{G}}(u)) = \mathrm{bd}^n(\mathrm{bc}(\mathsf{C}(\mathrm{bd}^n(\tfrac{u-1}{2} - 2^n))))\\ &= \mathsf{C}(\mathrm{bd}^n(\tfrac{u-1}{2} - 2^n)),\end{aligned}$$

and $\mathrm{bd}^n(\frac{u-1}{2} - 2^n)$ is a solution to the original instance C of Pigeon, case 1.

2. $u, v \in [s]$ such that $f_{\mathbb{G}}(u, v) \geq s$. Since $s = 2^{n+2}$, this case cannot happen.

3. $u, v \in [s]$ such that $u \neq v$ and $\mathcal{I}_{\mathbb{G}}(u) = \mathcal{I}_{\mathbb{G}}(v)$. Similarly as for the first case, it must hold that $u, v \in A_o$. Hence, from Equation (2.5), we get that

$$\mathsf{C}(\mathrm{bd}^n(\tfrac{u-1}{2} - 2^n)) = \mathsf{C}(\mathrm{bd}^n(\tfrac{v-1}{2} - 2^n)).$$

From the fact that $u \neq v$ and from the definition of the set A_o, it follows that $\mathrm{bd}^n(\frac{u-1}{2} - 2^n) \neq \mathrm{bd}^n(\frac{v-1}{2} - 2^n)$, hence the pair $\mathrm{bd}^n(\frac{u-1}{2} - 2^n), \mathrm{bd}^n(\frac{v-1}{2} - 2^n)$ forms a non-trivial collision for C, i.e., a solution to the original instance C of Pigeon, case 2.

It remains to define the circuit f such that the induced index function $\mathcal{I}_{\mathbb{G}}$ satisfies Equation (2.5). Here, we make use of the second construction with $m = n + 2$ defined and analysed above, where we set $w = \mathrm{bd}(2^n)$, i.e., for which it holds that $\mathcal{I}_{\mathbb{G}}(a) = a + 2^n \bmod 2^{n+2}$ and that $\mathcal{I}'_{\mathbb{G}}(v) = v + w$. It remains to adjust the definition of the corresponding f'_0 and f'_1 such that we get the desired output for values from A_e and A_o. To this end, we set

$$f(u, v) = \begin{cases} 11v' & \text{if } u = v \neq \mathrm{bd}(g) \text{ and } v - w = 01v',\\ f'_0(v) & \text{if } u = v \neq \mathrm{bd}(g) \text{ and } v - w \neq 01v',\\ \mathsf{C}(\mathrm{bd}^n(\mathrm{bc}(v) - 2^n - 2^{n+1})) & \text{if } u = \mathrm{bd}(g) \text{ and } \mathrm{bc}(v) \in B,\\ f'_1(v) & \text{if } u = \mathrm{bd}(g) \text{ and } \mathrm{bc}(v) \notin B, \end{cases} \tag{2.6}$$

where $B = \{2^n + 2^{n+1}, \ldots, 2^{n+2} - 1\}$. Note that f can be defined on the remaining inputs arbitrarily since they are not used in the computation of $\mathcal{I}_{\mathbb{G}}$.

Note that for $a \in [2^{n+1}]$, only cases 2 and 4 from the definition of f in Equation (2.6) are used in the computation of $\mathcal{I}_{\mathbb{G}}(a)$. Since these cases 2 and 4 coincide with the previous construction, we have that

$$\mathcal{I}_{\mathbb{G}}(a) = a + 2^n \bmod 2^{n+2} = a + 2^n$$

for all $a \in [2^{n+1}]$, which corresponds to the first case in Equation (2.5).

For $a \in A_e$, it holds that $\mathrm{bd}_0(a)$ is of the form $\mathrm{bd}_0(a) = \mathrm{bd}(a) = 1v'0$. Hence, we get that

$$\mathcal{I}_{\mathbb{G}}(a) = \mathrm{bc}(f(\mathcal{I}'_{\mathbb{G}}(01v'), \mathcal{I}'_{\mathbb{G}}(01v'))) = \mathrm{bc}(f(01v' + w, 01v' + w)) = \mathrm{bc}(11v'),$$

where the first equality comes from the definition of $\mathcal{I}_{\mathbb{G}}$, the second one from the previous construction and the last one from the definition of f. Furthermore, we have that $\mathrm{bc}(11v') = 2^{n+1} + 2^n + \mathrm{bc}(v') = 2^{n+1} + \mathrm{bc}(01v') = 2^{n+1} + \frac{a}{2}$, which proves the second case in Equation (2.5).

For $a \in A_o$, it holds that $\mathrm{bd}_0(a)$ is of the form $\mathrm{bd}_0(a) = \mathrm{bd}(a) = 1v'1$. Hence, we get that

$$\mathcal{I}_{\mathbb{G}}(a) = \mathrm{bc}(f(\mathrm{bd}(g), \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(1v'0))))). \tag{2.7}$$

Moreover, it holds that $\mathrm{bc}(\mathrm{bd}(\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(1v'0)))) = \mathcal{I}_{\mathbb{G}}(\mathrm{bc}(1v'0)) = 2^{n+1} + \frac{\mathrm{bc}(1v'0)}{2} \in [2^n + 2^{n+1}, 2^{n+2} - 1] = B$ from the already proved second part of Equation (2.5) since $\mathrm{bc}(1v'0) \in A_e$. Hence, the third case from the definition of f in Equation (2.6) applies to Equation (2.7) and we get that

$$\begin{aligned}
\mathcal{I}_{\mathbb{G}}(a) &= \mathrm{bc}(\mathsf{C}(\mathrm{bd}^n(\mathrm{bc}(\mathrm{bd}(\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(1v'0)))) - 2^n - 2^{n+1}))) \\
&= \mathrm{bc}(\mathsf{C}(\mathrm{bd}^n(\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(1v'0)) - 2^n - 2^{n+1}))) \\
&= \mathrm{bc}(\mathsf{C}(\mathrm{bd}^n(\tfrac{\mathrm{bc}(1v'0)}{2} + 2^{n+1} - 2^n - 2^{n+1}))) \\
&= \mathrm{bc}(\mathsf{C}(\mathrm{bd}^n(\tfrac{\mathrm{bc}(1v'0)}{2} - 2^n))) \\
&= \mathrm{bc}(\mathsf{C}(\mathrm{bd}^n(\tfrac{a-1}{2} - 2^n))),
\end{aligned}$$

which proves the last case in Equation (2.5) and concludes the whole proof. □

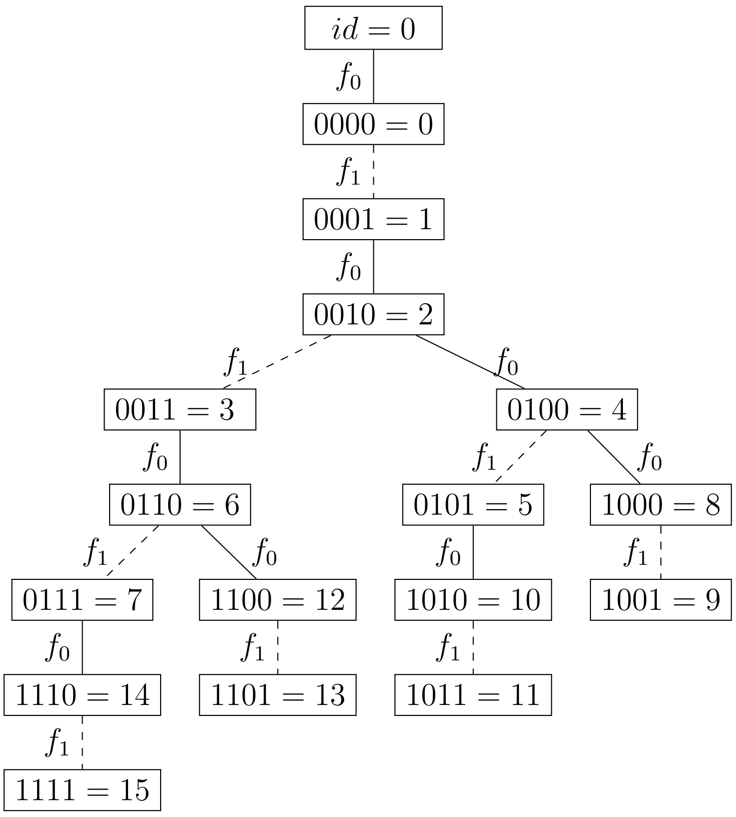

Figure 2.1: Tree induced by the computation of $\mathcal{I}_{\mathbb{G}}$, $\mathcal{I}_{\mathbb{G}}(a) = a$

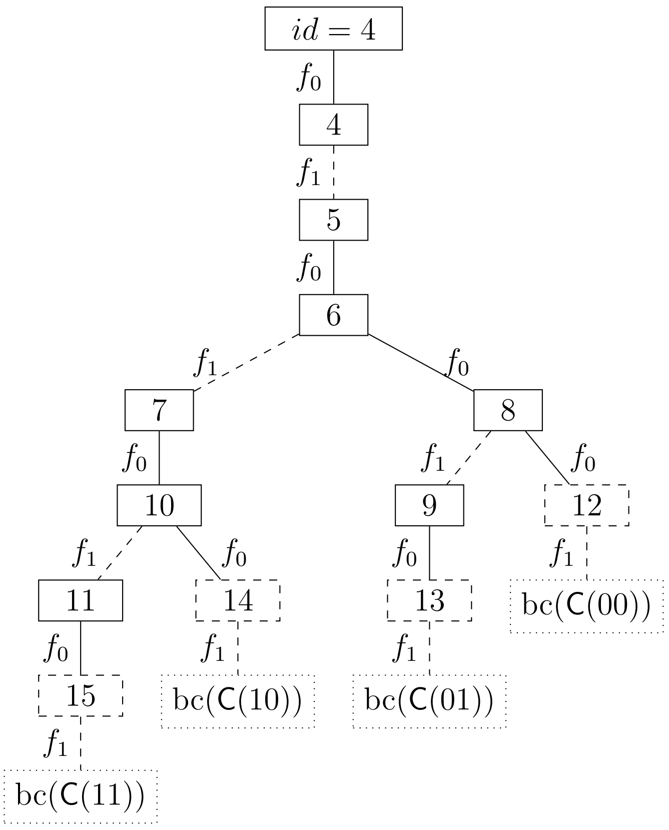

Figure 2.2: Tree induced by the computation of $\mathcal{I}_{\mathbb{G}}$, C incorporated

3. DLog is PWPP-complete

In this chapter, we define DLOG, another total search problem associated with DLP, and we show that it is PWPP-complete. Our reductions give rise to two additional new PWPP-complete problems called DOVE and CLAW, which we discuss further in Section 3.3. We start with the definition of DLOG.

Definition 7 (DLOG problem).

INSTANCE: A size parameter $s \in \mathbb{N}$ and a Boolean circuit $f\colon \{0,1\}^{2\lceil \log(s) \rceil} \to \{0,1\}^{\lceil \log(s) \rceil}$ representing a groupoid $(\mathbb{G}, \star)$ and indices $g, id, t \in [s]$.

SOLUTION: One of the following:

1. *$x \in [s]$ such that $\mathcal{I}_{\mathbb{G}}(x) = t$,*
2. *$x, y \in [s]$ such that $f_{\mathbb{G}}(x, y) \geq s$,*
3. *$x, y \in [s]$ such that $x \neq y$ and $\mathcal{I}_{\mathbb{G}}(x) = \mathcal{I}_{\mathbb{G}}(y)$,*
4. *$x, y \in [s]$ such that $x \neq y$ and $f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(x)) = f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(y))$,*
5. *$x, y \in [s]$ such that $\mathcal{I}_{\mathbb{G}}(x) = f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(y))$ and $\mathcal{I}_{\mathbb{G}}(x - y \bmod s) \neq t$.*

It is immediate that DLOG is a relaxation of INDEX due to the two additional types of a solution. These last two types of a solution make DLOG to lie in the class PWPP and are motivated by the construction of collision-resistant hash functions from DLP [4].

Motivation for DLog. The two additional types of a solution in DLOG are motivated by the construction of collision-resistant hash functions from DLP [4], which, on a high-level, corresponds to showing that DLP lies in PWPP. Let g, t be elements of some group and $x, y \in \mathbb{Z}$. In the construction from [4], the following properties of groups are used:

(a) $(t \cdot g^x = t \cdot g^y) \implies (g^x = g^y)$,

(b) $(g^x = t \cdot g^y) \implies (g^{x-y} = t)$.

The fourth type of a solution in DLOG is a witness that the property (a) does not hold in the induced groupoid and, similarly, the fifth type of a solution in DLOG is a witness that the property (b) does not hold in the induced groupoid. Hence, these two additional types of a solution in DLOG allow us to do the manipulations described by (a) and (b) - either these manipulations are valid, or we found a solution to the instance of DLOG.

The main result of this chapter is the following theorem that we prove in the rest of this chapter.

Theorem 4. *DLOG is PWPP-complete.*

Specifically, we show that DLOG lies in PWPP in Section 3.1 and we establish that DLOG is PWPP-hard in Section 3.2.

3.1 DLog lies in PWPP

In this section, we show that DLog lies in the class PWPP. Our reduction from DLog to Collision is inspired by the standard construction of collision-resistant hash functions from DLP by Damgård [4]. The construction in [4] goes through an intermediate object of *claw-free permutations*, which are sufficient for collision-resistant hashing. A family of claw-free permutations is an efficiently sampleable family of pairs of permutations such that given a "random" pair h_0, h_1 of permutations from the family, it is computationally infeasible to find a *claw* for these two permutations, i.e., inputs u and v such that $h_0(u) = h_1(v)$. We formalize the corresponding total search problem, which we call Claw, below.

Definition 8 (Claw problem)**.**

Instance: *Two Boolean circuits h_0, h_1 with n inputs and n outputs.*

Solution: *One of the following:*
1. *strings $u, v \in \{0,1\}^n$ such that $h_0(u) = h_1(v)$,*
2. *strings $u, v \in \{0,1\}^n$ such that $u \neq v$ and $h_0(u) = h_0(v)$,*
3. *strings $u, v \in \{0,1\}^n$ such that $u \neq v$ and $h_1(u) = h_1(v)$.*

The first type of a solution in Definition 8 corresponds to finding a claw for the pair of functions h_0 and h_1. As we cannot efficiently certify that both h_0 and h_1 are permutations, we introduce the second and third type of a solution which witness that one of these functions is not bijective. In other words, the second and third type of a solution ensure the totality of Claw.

Similarly to [4], our high-level approach is to first give a reduction from DLog to Claw and then a reduction from Claw to Collision. Although, we cannot simply employ his analysis since we have no guarantee that 1) the groupoid induced by an arbitrary DLog instance is a cyclic group and 2) that an arbitrary instance of Claw corresponds to a pair of permutations. It turns out that the second issue is not crucial. It was observed by Russell [5] that the notion of claw-free *pseudopermutations* is sufficient for collision-resistant hashing. Our definition of Claw corresponds exactly to the worst-case version of breaking claw-free pseudopermutations as defined by [5]. As for the first issue, we manage to provide a formal reduction from DLog to General-Claw, a variant of Claw defined below.

Definition 9 (General-Claw problem)**.**

Instance: *Two Boolean circuits h_0, h_1 with n inputs and n outputs and $s \in \mathbb{Z}$ such that $1 \leq s < 2^n$.*

Solution: *One of the following:*
1. *strings $u, v \in \{0,1\}^n$ such that $\mathrm{bc}(u) < s$, $\mathrm{bc}(v) < s$ and $h_0(u) = h_1(v)$,*
2. *strings $u, v \in \{0,1\}^n$ such that $u \neq v$ and $h_0(u) = h_0(v)$,*
3. *strings $u, v \in \{0,1\}^n$ such that $u \neq v$ and $h_1(u) = h_1(v)$,*
4. *a string $u \in \{0,1\}^n$ such that $\mathrm{bc}(u) < s$ and $\mathrm{bc}(h_0(u)) \geq s$,*
5. *a string $u \in \{0,1\}^n$ such that $\mathrm{bc}(u) < s$ and $\mathrm{bc}(h_1(u)) \geq s$.*

The motivation for the additional two types of a solution in General-Claw compared with Claw is that the possible solutions to an instance of DLog are not from the whole domain $[2^n]$, but they must lie in $[s]$. Additionally, note that the last two types of a solution in our definition of DLog (Definition 7) are used in the reduction from DLog to General-Claw to substitute the necessary group axioms for the construction from [4].

Below, we give the formal reduction from DLog to General-Claw.

Lemma 5. *DLog is reducible to General-Claw.*

Proof. We start with an arbitrary instance $G = (s, f, id, g, t)$ of DLog. Let $n = \lceil \log(s) \rceil$. We define $h_0 : \{0,1\}^n \to \{0,1\}^n$ and $h_1 : \{0,1\}^n \to \{0,1\}^n$ as follows:

$$h_0(u) = \begin{cases} \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(u))) & \text{if } \mathrm{bc}(u) < s, \\ u & \text{otherwise,} \end{cases}$$

and

$$h_1(u) = \begin{cases} f(\mathrm{bd}(t), \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(u))) & \text{if } \mathrm{bc}(u) < s, \\ u & \text{otherwise,} \end{cases}$$

where $u \in \{0,1\}^n$. We show that any solution to this instance (h_0, h_1, s) of General-Claw gives a solution to the above instance G of DLog. Five cases can occur:

1. The solution is $u, v \in \{0,1\}^n$ such that $\mathrm{bc}(u) < s$, $\mathrm{bc}(v) < s$ and $h_0(u) = h_1(v)$. Then, for $x = \mathrm{bc}(u), y = \mathrm{bc}(v)$, it holds that $x, y \in [s]$, so

$$h_0(u) = \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(u))) = \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(x))$$

   and

$$h_1(v) = f(\mathrm{bd}(t), \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(v))) = \mathrm{bd}(f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(y))).$$

   Putting these equalities together, we get that

$$\mathrm{bd}(\mathcal{I}_{\mathbb{G}}(x)) = \mathrm{bd}(f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(y))),$$

   and hence

$$\mathcal{I}_{\mathbb{G}}(x) = f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(y)).$$

   If $\mathcal{I}_{\mathbb{G}}(x - y \bmod s) = t$, then $x - y \bmod s \in [s]$ is the discrete logarithm of t, i.e., a solution to the original instance G of DLog, case 1. Otherwise, the pair x, y is a solution to the original instance G of DLog, case 5.

2. The solution is $u, v \in \{0,1\}^n$ such that $u \neq v$ and $h_0(u) = h_0(v)$. Let $x = \mathrm{bc}(u)$, $y = \mathrm{bc}(v)$. If $x \geq s$, then from the definition of h_0 and the fact that $x \neq y$, we get that $y < s$ and

$$u = h_0(u) = h_0(v) = \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(y)),$$

   so

$$x = \mathrm{bc}(u) = \mathcal{I}_{\mathbb{G}}(y)$$

with $x \geq s$ and $y \in [s]$. It means that after some step in the computation of $\mathcal{I}_{\mathbb{G}}(y)$, it holds that $\mathrm{bc}(r) \geq s$. We consider the first such step. Since all steps correspond to applying the circuit f, we have that

$$r = f(r', r'')$$

for some r', r'' such that $\mathrm{bc}(r'), \mathrm{bc}(r'') \in [s]$. This rewrites to

$$s \leq \mathrm{bc}(r) = f_{\mathbb{G}}(\mathrm{bc}(r'), \mathrm{bc}(r'')).$$

Hence, $\mathrm{bc}(r'), \mathrm{bc}(r'')$ is a solution to the original instance G of DLog, case 2. We proceed analogously if $y \geq s$. Now assume that $x, y \in [s]$. Then we get that

$$\mathcal{I}_{\mathbb{G}}(x) = \mathcal{I}_{\mathbb{G}}(y)$$

and since $x \neq y$, we found a solution to the original instance G of DLog, case 3.

3. The solution is $u, v \in \{0,1\}^n$ such that $u \neq v$ and $h_1(u) = h_1(v)$. Let $x = \mathrm{bc}(u)$, $y = \mathrm{bc}(v)$. If $x \geq s$, then from the definition of h_1 and the fact that $x \neq y$, we get that $y < s$ and

$$u = h_1(u) = h_1(v) = f(\mathrm{bd}(t), \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(y))),$$

so

$$x = \mathrm{bc}(u) = \mathrm{bc}(f(\mathrm{bd}(t), \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(y)))) = f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(y))$$

with $x \geq s$ and $y \in [s]$. If $\mathcal{I}_{\mathbb{G}}(y) \geq s$, then we proceed as above in the previous case. If $\mathcal{I}_{\mathbb{G}}(y) \in [s]$, then $t, \mathcal{I}_{\mathbb{G}}(y)$ is a solution to the original instance G of DLog, case 2. We proceed analogously if $y \geq s$. Now assume that $x, y \in [s]$. Then we get that

$$f(\mathrm{bd}(t), \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(x))) = f(\mathrm{bd}(t), \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(y))),$$

so

$$f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(x)) = f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(y))$$

and since $x \neq y$, we found a solution to the original instance G of DLog, case 4.

4. The solution is $u \in \{0,1\}^n$ such that $\mathrm{bc}(u) < s$ and $\mathrm{bc}(h_0(u)) \geq s$. Let $x = \mathrm{bc}(u)$. Then we have that

$$s \leq \mathrm{bc}(h_0(u)) = \mathcal{I}_{\mathbb{G}}(x)$$

with $x \in [s]$. Now we can proceed as in analogous situations in cases 2 and 3 above.

5. The solution is $u \in \{0,1\}^n$ such that $\mathrm{bc}(u) < s$ and $\mathrm{bc}(h_1(u)) \geq s$. Let $x = \mathrm{bc}(u)$. Then we have that

$$s \leq \mathrm{bc}(h_1(u)) = \mathrm{bc}(f(\mathrm{bd}(t), \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(u)))) = f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(x))$$

with $x \in [s]$. Now we can proceed as in the same situation in case 3 above.

□

Now, we give the formal reduction from General-Claw to Collision.

Lemma 6. *General-Claw is reducible to Collision.*

Proof. We start with an arbitrary instance (h_0, h_1, s) of General-Claw, where $h_0, h_1 : \{0,1\}^n \to \{0,1\}^n$. We define a circuit $\mathsf{C} : \{0,1\}^{n+1} \to \{0,1\}^n$ as follows:

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

where $x = (x_0, x_1, \ldots, x_n)$. The construction is valid since the circuit C can be constructed in polynomial time in the size of the given instance (h_0, h_1, s) of General-Claw. Now we show that any solution to this instance C of Collision gives a solution to the original instance (h_0, h_1, s) of General-Claw. There is only one type of a solution for Collision, so assume that $\mathsf{C}(x) = \mathsf{C}(y)$ for $x \neq y$, where $x = (x_0, x_1, \ldots, x_n)$ and $y = (y_0, y_1, \ldots, y_n)$. If it holds that

$$\mathrm{bc}(h_{x_i} \circ \cdots \circ h_{x_n}(0^n)) \geq s$$

for some $0 \leq i \leq n$, then consider the largest such i. We emphasize that we can check this in polynomial time. We have that

$$\mathrm{bc}(h_{x_i} \circ h_{x_{i+1}} \cdots \circ h_{x_n}(0^n)) \geq s$$

and

$$\mathrm{bc}(h_{x_{i+1}} \circ \cdots \circ h_{x_n}(0^n)) < s.$$

Then, for $u = h_{x_{i+1}} \circ \cdots \circ h_{x_n}(0^n)$, it holds that $\mathrm{bc}(u) < s$ and $\mathrm{bc}(h_{x_i}(u)) \geq s$. So, u forms a solution to the original instance (h_0, h_1, s) of General-Claw, case 4 or 5 based on the bit x_i. We proceed analogously if

$$\mathrm{bc}(h_{y_i} \circ \cdots \circ h_{y_n}(0^n)) \geq s$$

for some $0 \leq i \leq n$. For the rest of the proof, we can assume that

$$\mathrm{bc}(h_{x_i} \circ \cdots \circ h_{x_n}(0^n)) < s$$

and

$$\mathrm{bc}(h_{y_i} \circ \cdots \circ h_{y_n}(0^n)) < s$$

for all $0 \leq i \leq n$. Since $x \neq y$, there is some i such that $x_i \neq y_i$. If

$$h_{x_i} \circ \cdots \circ h_{x_n}(0^n) = h_{y_i} \circ \cdots \circ h_{y_n}(0^n),$$

then the pair $u = h_{x_{i+1}} \circ \cdots \circ h_{x_n}(0^n)$, $v = h_{y_{i+1}} \circ \cdots \circ h_{y_n}(0^n)$ satisfies $\mathrm{bc}(u) < s, \mathrm{bc}(v) < s$ and $h_{x_i}(u) = h_{y_i}(v)$ with $x_i \neq y_i$, hence the pair u, v forms a solution to the original instance (h_0, h_1, s) of General-Claw, case 1. Otherwise, if

$$h_{x_i} \circ \cdots \circ h_{x_n}(0^n) \neq h_{y_i} \circ \cdots \circ h_{y_n}(0^n),$$

then there must be some $j < i$, such that

$$h_{x_j} \circ \cdots \circ h_{x_n}(0^n) = h_{y_j} \circ \cdots \circ h_{y_n}(0^n),$$

and we consider the largest such j. Then, it holds that

$$h_{x_{j+1}} \circ \cdots \circ h_{x_n}(0^n) \neq h_{y_{j+1}} \circ \cdots \circ h_{y_n}(0^n).$$

The pair $u = h_{x_{j+1}} \circ \cdots \circ h_{x_n}(0^n)$, $v = h_{y_{j+1}} \circ \cdots \circ h_{y_n}(0^n)$ satisfies $u \neq v$, $\mathrm{bc}(u) < s$, $\mathrm{bc}(v) < s$ and $h_{x_j}(u) = h_{y_j}(v)$. So, the pair u, v forms a solution to the original instance (h_0, h_1, s) of General-Claw, case 1, 2, or 3 based on the bits x_j and y_j. □

The above Lemma 5 and Lemma 6 imply that DLog lies in PWPP and we conclude this section with the corresponding corollary.

Corollary. DLog lies in PWPP.

3.2 DLog is PWPP-hard

In this section, we prove that DLog is PWPP-hard, which, together with the previous section, shows that DLog is PWPP-complete.

Our reduction from Collision to DLog goes through an intermediate problem we call Dove, which is a variant of the PPP-complete problem Pigeon with additional types of a solution. We start with the definition of Dove.

Definition 10 (Dove problem)**.**

Instance: *A Boolean circuit* C *with* n *inputs and* n *outputs.*

Solution: *One of the following:*

1. *a string* $u \in \{0,1\}^n$ *such that* $\mathsf{C}(u) = 0^n$*,*
2. *a string* $u \in \{0,1\}^n$ *such that* $\mathsf{C}(u) = 0^{n-1}1$,
3. *strings* $u, v \in \{0,1\}^n$ *such that* $u \neq v$ *and* $\mathsf{C}(u) = \mathsf{C}(v)$*,*
4. *strings* $u, v \in \{0,1\}^n$ *such that* $\mathsf{C}(u) = C(v) \oplus 0^{n-1}1$.

It is immediate that Dove is a relaxation of Pigeon (cf. Definition 4) with two additional new types of a solution – the cases 2 and 4 in the above definition. Similarly to case 1, case 2 corresponds to a preimage of a fixed element in the range. Case 4 corresponds to a pair of strings such that their images under C differ only on the last bit.

Permutations for which it is computationally infeasible to find inputs with evaluations differing only on a prescribed index appeared in the work of Zheng, Matsumoto, and Imai [7] under the term *distinction-intractable* permutations. In [7], they showed that distinction-intractable permutations are sufficient for collision-resistant hashing. Note that we employ distinction-intractability in a different way than [7]. In particular, their construction of collision-resistant hash from distinction-intractable permutations could be leveraged towards a reduction from Dove to Collision (proving Dove is contained in PWPP) – whereas we use Dove as an intermediate problem when reducing from Collision to DLog (proving PWPP-hardness of DLog).

First, we provide a high-level idea of the reduction from Dove to DLog, which also motivates the definition of Dove.

Reducing Dove to DLog. Let $\mathsf{C} : \{0,1\}^n \to \{0,1\}^n$ be an arbitrary instance of Dove. Our goal is to construct an instance $G = (s, f, id, g, t)$ of DLog such that any solution to G provides a solution to the original instance C of Dove.

The key step in the construction of G is a suitable choice of the circuit f since it defines both $\mathcal{I}_{\mathbb{G}}$ and $f_{\mathbb{G}}$. The main issue with the reduction is the third type of a solution in DLog, i.e., defining f such that we can find a solution to the Dove instance C from any non-trivial collision $\mathcal{I}_{\mathbb{G}}(x) = \mathcal{I}_{\mathbb{G}}(y)$.

As we know, the computation of $\mathcal{I}_{\mathbb{G}}(x)$ simply corresponds to an iterated composition of the functions f_0 and f_1 depending on the binary representation $\mathrm{bd}_0(x)$ of x and evaluated on id. The straightforward option would be to set $f_0(r) = f_1(r) = \mathsf{C}(r)$ for all $r \in \{0,1\}^n$. Unfortunately, such an approach fails since for all distinct $u, v \in [s]$ such that the number of zeros plus twice the number of ones in $\mathrm{bd}_0(u)$ is same as in $\mathrm{bd}_0(v)$, there would be easy to find non-trivial collisions $\mathcal{I}_{\mathbb{G}}(x) = \mathcal{I}_{\mathbb{G}}(y)$, which do not provide any useful information about the circuit C.

Example. $\mathcal{I}_{\mathbb{G}}(5) = \mathcal{I}_{\mathbb{G}}(\mathrm{bc}(101)) = \mathrm{bc}(f_1 \circ f_0 \circ f_0 \circ f_1 \circ f_0(\mathrm{bd}(id))) = \mathsf{C}^5(\mathrm{bd}(id)) = \mathrm{bc}(f_0 \circ f_0 \circ f_0 \circ f_1 \circ f_0(\mathrm{bd}(id))) = \mathcal{I}_{\mathbb{G}}(\mathrm{bc}(1000)) = \mathcal{I}_{\mathbb{G}}(8)$.

Hence, we define f_0 and f_1 such that $f_0 \neq f_1$. On a high level, we set $f_0(r) = \mathsf{C}(r)$ and $f_1(r) = C(h(r))$ for some function $h\colon \{0,1\}^n \to \{0,1\}^n$ that is not the identity as in the flawed attempt above. Then, except for some special cases, a non-trivial collision $\mathcal{I}_{\mathbb{G}}(x) = \mathcal{I}_{\mathbb{G}}(y)$ corresponds to

$$\mathsf{C}(\mathsf{C}(u)) = \mathsf{C}(h(\mathsf{C}(v)))$$

for some $u, v \in \{0,1\}^n$, which are not necessarily distinct. If $\mathsf{C}(u) \neq h(\mathsf{C}(v))$ then $\mathsf{C}(u), h(\mathsf{C}(v))$ forms a non-trivial collision for C. Otherwise, we found a pair u, v such that $\mathsf{C}(u) = h(\mathsf{C}(v))$, which, for the choice $h(y) = y \oplus 0^{n-1}1$, translates into

$$\mathsf{C}(u) = \mathsf{C}(v) \oplus 0^{n-1}1, \tag{3.1}$$

i.e., a pair of inputs breaking distinction-intractability of C, and corresponds to the last type of a solution in Dove. Finally, the second type of a solution (together also with the first one and the third one) in Dove captures the special case when there is no pair u, v such that $\mathsf{C}(\mathsf{C}(u)) = \mathsf{C}(h(\mathsf{C}(v)))$ and also the other types of a solution in DLog.

We give the formal reduction from Dove to DLog below.

Lemma 7. *Dove is reducible to DLog.*

Proof. Let $\mathsf{C}\colon \{0,1\}^n \to \{0,1\}^n$ be an arbitrary instance of Dove. We construct the corresponding instance $G = (s, f, id, g, t)$ of DLog. Set $s = 2^n, g = 0, id = 1, t = 1$ and define the circuit $f : \{0,1\}^{2n} \to \{0,1\}^n$ as follows:

$$f(x,y) = \begin{cases} \mathsf{C}(x) & \text{if } x = y, \\ \mathsf{C}(y \oplus 0^{n-1}1) & \text{if } x = \mathrm{bd}(g) \text{ and } y \neq \mathrm{bd}(g), \\ x \oplus y & \text{otherwise,} \end{cases}$$

where $x, y \in \{0,1\}^n$. Then the groupoid representation (s, f) with the indices g, id, t form an instance of DLog. We emphasize that we can access all intermediate results in the computation of $\mathcal{I}_{\mathbb{G}}$ since the whole computation is performed in polynomial time in the size of the input instance C.

Now we show that any solution to this DLOG instance gives a solution to the original DOVE instance C. Five cases can occur:

1. The solution is $x \in [s]$ such that $\mathcal{I}_{\mathbb{G}}(x) = t$. For our DLOG instance, $t = 1$, hence $\mathcal{I}_{\mathbb{G}}(x) = 1$. From the definition of the function $\mathcal{I}_{\mathbb{G}}$, it holds that $\mathrm{bd}(\mathcal{I}_{\mathbb{G}}(x)) = f(r, r) = \mathsf{C}(r)$ or $\mathrm{bd}(\mathcal{I}_{\mathbb{G}}(x)) = f(g, r) = \mathsf{C}(r \oplus 0^{n-1}1)$ for some $r \in \{0, 1\}^n$. Putting these equalities together, we get that
$$0^{n-1}1 = \mathrm{bd}(1) = \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(x)) = \mathsf{C}(y),$$
where $y = r$ or $y = r \oplus 0^{n-1}1$. So this $y \in \{0, 1\}^n$ is a preimage of $0^{n-1}1$ in C, i.e., it is a solution to the original DOVE instance C, case 2.

2. The solution is a pair $x, y \in [s]$ such that $f_{\mathbb{G}}(x, y) \geq s$. But since $s = 2^n$ and $f_{\mathbb{G}} : [s] \times [s] \to [2^n]$, this case cannot happen.

3. The solution is a pair $x, y \in [s]$ such that $x \neq y$ and $\mathcal{I}_{\mathbb{G}}(x) = \mathcal{I}_{\mathbb{G}}(y)$. First, we can assume that in the computation of $\mathcal{I}_{\mathbb{G}}(x)$ and $\mathcal{I}_{\mathbb{G}}(y), r \neq \mathrm{bd}(g)$ for all iterations. If that was the case, then for the first such occurrence it holds that
$$0^n = \mathrm{bd}(0) = \mathrm{bd}(g) = r = \begin{cases} f(r', r') = \mathsf{C}(r'), \\ f(g, r') = \mathsf{C}(r' \oplus 0^{n-1}1). \end{cases}$$
In both cases, we found a preimage of 0^n in C, i.e., a solution to the original DOVE instance C, case 1.

   Further, let $(x_k, \dots, x_0) = \mathrm{bd}_0(x)$ and $(y_l, \dots, y_0) = \mathrm{bd}_0(y)$ be the binary representations of x and y, respectively, where x_0 and y_0 are the least significant bits and $k, l < n$. We use the following notation: by r_{z_i} we denote the value of the variable r in the computation of $\mathcal{I}_{\mathbb{G}}(z)$ after the loop corresponding to the bit z_i. Since $x \neq y$, their binary representations are distinct as well.

   Hence, there are three possible cases:

   (a) There is some i such that $x_i \neq y_i$. Let j denote the smallest such i. Without loss of generality, assume that $x_j = 0$ and $y_j = 1$. Hence, it holds that $r_{x_j} = \mathsf{C}(a)$ and $r_{y_j} = \mathsf{C}(\mathsf{C}(b) \oplus 0^{n-1}1)$ for some $a, b \in \{0, 1\}^n$.

      i. If $j = 0$, then it holds that $\mathrm{bc}(r_{x_j}) = \mathcal{I}_{\mathbb{G}}(x) = \mathcal{I}_{\mathbb{G}}(y) = \mathrm{bc}(r_{y_j})$, which means that
$$\mathsf{C}(a) = \mathsf{C}(\mathsf{C}(b) \oplus 0^{n-1}1). \tag{3.2}$$
      If $x = 0$, then $a = \mathrm{bd}(id) = \mathrm{bd}(1) = 0^{n-1}1$, so
$$\mathsf{C}(0^{n-1}1) = \mathsf{C}(\mathsf{C}(b) \oplus 0^{n-1}1).$$
      Now, either $0^{n-1}1 \neq \mathsf{C}(b) \oplus 0^{n-1}1$, which means a non-trivial collision in C, i.e., a solution to the original instance C, case 3, or $0^{n-1}1 = \mathsf{C}(b) \oplus 0^{n-1}1$, which implies $0^n = \mathsf{C}(b)$ and b is a preimage of 0^n in C, i.e., a solution to the original instance C, case 1.

If $x \neq 0$, then $j < k$ and $a = \mathsf{C}(c)$ for some $c \in \{0,1\}^n$. Substituting to the above equality 3.2, we get that

$$\mathsf{C}(\mathsf{C}(c)) = \mathsf{C}(\mathsf{C}(b) \oplus 0^{n-1}1).$$

Now, either $\mathsf{C}(c) \neq \mathsf{C}(b) \oplus 0^{n-1}1$, which means a non-trivial collision in C, i.e., a solution to the original instance C, case 3, or $\mathsf{C}(c) = \mathsf{C}(b) \oplus 0^{n-1}1$, so the pair b, c forms a solution to the original instance C, case 4.

ii. Now suppose that $j \neq 0$. If $r_{x_j} = r_{y_j}$, then the proof can be reduced to the previous case $j = 0$. Otherwise, since j is the smallest index where x_j and y_j differ, we know that $(x_{j-1}, \ldots, x_0) = (y_{j-1}, \ldots, y_0)$ and, hence, the computation of $\mathcal{I}_{\mathbb{G}}(x)$ and of $\mathcal{I}_{\mathbb{G}}(y)$ uses exactly same steps starting from r_{x_j}, r_{y_j}. Since $r_{x_j} \neq r_{y_j}$ and $\mathcal{I}_{\mathbb{G}}(x) = \mathcal{I}_{\mathbb{G}}(y)$, there must be a collision after some step. Since all these steps correspond to applying the circuit C, we can find a non-trivial collision in C, i.e., a solution to the original instance C, case 3.

(b) It holds that $(x_l, \ldots, x_0) = (y_l, \ldots, y_0)$ and $k > l$. We know that $r_{x_{l+1}} = \mathsf{C}(a)$ for some a. In the computation of $\mathcal{I}_{\mathbb{G}}$, the variable r is initialized to $\mathrm{bd}(id)$ at the beginning. Since $(x_l, \ldots, x_0) = (y_l, \ldots, y_0)$, the computation of $\mathcal{I}_{\mathbb{G}}(x)$ starting from $r_{x_{l+1}}$ uses the same steps as the whole computation of $\mathcal{I}_{\mathbb{G}}(y)$, which starts from $r = \mathrm{bd}(id) = \mathrm{bd}(1) = 0^{n-1}1$. If $0^{n-1}1 = r_{x_{l+1}} = \mathsf{C}(a)$, then a is a preimage of $0^{n-1}1$ in C, i.e., a solution to the original instance C, case 2. If $0^{n-1}1 \neq r_{x_{l+1}}$, then there must be a non-trivial collision after some step since $\mathcal{I}_{\mathbb{G}}(x) = \mathcal{I}_{\mathbb{G}}(y)$. Since all these steps correspond to applying the circuit C, we can find a non-trivial collision in C, i.e., a solution to the original instance C, case 3.

(c) It holds that $(x_k, \ldots, x_0) = (y_k, \ldots, y_0)$ and $k < l$. Then the proof proceeds analogously as for the case $k > l$ only with the roles of x and y switched.

4. The solution is a pair $x, y \in [s]$ such that $x \neq y$ and $f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(x)) = f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(y))$. If $t = \mathcal{I}_{\mathbb{G}}(x)$ or $t = \mathcal{I}_{\mathbb{G}}(y)$, we can proceed as in the case 1. above. Otherwise, since $t \neq g$, it holds that $f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(x)) = \mathrm{bc}(\mathrm{bd}(t) \oplus \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(x)))$ and that $f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(y)) = \mathrm{bc}(\mathrm{bd}(t) \oplus \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(y)))$. By combining these equalities, we obtain that

$$\mathrm{bc}(\mathrm{bd}(t) \oplus \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(x))) = \mathrm{bc}(\mathrm{bd}(t) \oplus \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(y))),$$

which implies that

$$\mathcal{I}_{\mathbb{G}}(x) = \mathcal{I}_{\mathbb{G}}(y),$$

and since $x \neq y$, we proceed as in the case 3. above.

5. The solution is a pair $x, y \in [s]$ such that

$$\mathcal{I}_{\mathbb{G}}(x) = f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(y)) \tag{3.3}$$

and $\mathcal{I}_{\mathbb{G}}(x - y \bmod s) \neq t$. If $t = \mathcal{I}_{\mathbb{G}}(y)$, then we can proceed as in the case 1. above. Otherwise, since $t \neq g$, we have that

$$f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(y)) = \mathrm{bc}(\mathrm{bd}(t) \oplus \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(y))). \tag{3.4}$$

By combining equations 3.3 and 3.4, we get that

$$\mathcal{I}_{\mathbb{G}}(x) = \mathrm{bc}(\mathrm{bd}(t) \oplus \mathrm{bd}(\mathcal{I}_{\mathbb{G}}(y))).$$

Moreover, we know that $\mathcal{I}_{\mathbb{G}}(x) = \mathrm{bc}(\mathsf{C}(r))$ for some $r \in \{0,1\}^n$ and that $\mathcal{I}_{\mathbb{G}}(y) = \mathrm{bc}(\mathsf{C}(r'))$ for some $r' \in \{0,1\}^n$. Substituting to the previous relationship and using the fact the bc and bd are bijections inverse to each other, we get that

$$\mathsf{C}(r) = \mathrm{bd}(t) \oplus \mathsf{C}(r') = \mathrm{bd}(1) \oplus \mathsf{C}(r') = 0^{n-1}1 \oplus \mathsf{C}(r').$$

Hence, the pair r, r' forms a solution to the original instance C, case 4.

□

In the next lemma, we show that, by introducing additional types of a solution into the definition of PIGEON, we do not make the corresponding search problem too easy – DOVE is at least as hard as any problem in PWPP.

Lemma 8. *COLLISION is reducible to DOVE.*

Proof. We start with an arbitrary instance $\mathsf{C} : \{0,1\}^n \to \{0,1\}^m$ with $m < n$ of COLLISION. Moreover, we can assume that $m = n - 1$ because otherwise we can pad the output with zeros, which preserves the collisions. We construct a circuit $\mathsf{V} : \{0,1\}^{2n} \to \{0,1\}^{2n}$, considered as an instance of DOVE, as follows:

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

where $x_i \in \{0,1\}$. The construction is valid since the new circuit V can be constructed in polynomial time with respect to the size of C. Now we show that any solution to the above instance V of DOVE gives a solution to the original COLLISION instance C. Four cases can occur:

1. The solution to V is $(x_1, \ldots, x_{2n}) \in \{0,1\}^{2n}$ such that $\mathsf{V}(x_1, \ldots, x_{2n}) = 0^{2n}$. From the definition of the circuit V, the last bit of the output is always 1, hence this case cannot happen.

2. The solution to V is $(x_1, \ldots, x_{2n}) \in \{0,1\}^{2n}$ such that $\mathsf{V}(x_1, \ldots, x_{2n}) = 0^{n-1}1 = (0, 0, \ldots, 0, 1)$. From the definition of the circuit V, the next-to-last bit of the output is always 1, hence this case cannot happen.

3. The solution to V is $x = (x_1, \ldots, x_{2n}), y = (y_1, \ldots, y_{2n}) \in \{0,1\}^{2n}$ such that $x \neq y$ and $\mathsf{V}(x) = \mathsf{V}(y)$. From the definition of the circuit V, it holds that

   $$\mathsf{C}(x_1, \ldots, x_n) = \mathsf{C}(y_1, \ldots, y_n)$$

   and

   $$\mathsf{C}(x_{n+1}, \ldots, x_{2n}) = \mathsf{C}(y_{n+1}, \ldots, y_{2n}).$$

   Moreover, we know that, since $x \neq y$, either $(x_1, \ldots, x_n) \neq (y_1, \ldots, y_n)$ or $(x_{n+1}, \ldots, x_{2n}) \neq (y_{n+1}, \ldots, y_{2n})$. In both cases, we found a non-trivial collision in C, i.e., a solution to the original instance C.

4. The solution to V is $x, y \in \{0,1\}^{2n}$ such that $\mathsf{V}(x) = \mathsf{V}(y) \oplus 0^{n-1}1$, i.e., their evaluations differ only on the last bit. From the definition of the circuit V, the last bit of the output is always 1, hence this case cannot happen.

□

The above Lemma 7 and Lemma 8 imply that DLog is PWPP-hard and we conclude this section with the corresponding corollary.

Corollary. DLog is PWPP-hard.

3.3 New characterizations of PWPP

Our results in Section 3.1 and Section 3.2 establish two new PWPP-complete problems besides DLog.

Dove. The chain of reductions in Chapter 3 shows, in particular, that Dove is PWPP-complete. As a relaxation of Pigeon, it is the first PWPP-complete problem not defined in terms of an explicitly shrinking function. Nevertheless, it is equivalent to Collision and, thus, it inherently captures some notion of compression. Given its different structure than Collision, we were able to leverage it in our proof of PWPP-hardness of DLog, and it might prove useful in attempts at proving PWPP-hardness of other problems.

Claw. In [5], Russell showed that a weakening of claw-free permutations is sufficient for collision-resistant hashing. Specifically, he leveraged claw-free *pseudopermutations*, i.e., functions for which it is also computationally infeasible to find a witness refuting their bijectivity. Our definition of Claw ensures totality by an identical existential argument – a pair of functions with identical domain and range either has a claw or some of the functions is not bijective.

Claw trivially reduces to the PWPP-complete problem General-Claw and, thus, it is contained in PWPP. Below, we also show a reduction from Collision to Claw establishing that it is PWPP-hard.

Lemma 9. *Collision is reducible to Claw.*

Proof. We start with an arbitrary instance $\mathsf{C} : \{0,1\}^n \to \{0,1\}^m$ of Collision with $m < n$. Without loss of generality, we can suppose that $m = n - 1$ since otherwise we can pad the output with zeros, which preserves collisions. We construct an instance of Claw as follows:

$$h_0(x) = \mathsf{C}(x)0$$

and

$$h_1(x) = \mathsf{C}(x)1.$$

We show that any solution to this instance (h_0, h_1) of Claw gives a solution to the original instance C of Collision. Three cases can occur:

1. $u, v \in \{0,1\}^n$ such that $h_0(u) = h_1(v)$. Since the last bit of $h_0(u)$ is zero and the last bit of $h_1(v)$ is one, this case cannot happen.

2. $u, v \in \{0,1\}^n$ such that $u \neq v$ and $h_0(u) = h_0(v)$. From the definition of h_0, we get that $\mathsf{C}(u)0 = h_0(u) = h_0(v) = \mathsf{C}(v)0$, which implies that $\mathsf{C}(u) = \mathsf{C}(v)$. Hence, the pair u, v forms a solution to the original Collision instance C.

3. $u, v \in \{0,1\}^n$ such that $u \neq v$ and $h_1(u) = h_1(v)$. We can proceed analogously as in the previous case to show that the pair u, v forms a solution to the original Collision instance C.

$\square$

4. Ensuring the totality of search problems in number theory

In this section, we discuss some of the issues that arise when defining total search problems corresponding to actual problems in computational number theory.

4.1 DLP

In this section, we present a formalization of the discrete logarithm problem in $\mathbb{Z}_p^*$, i.e., the multiplicative group of integers modulo a prime p. Our goal is to highlight the distinction between the general DLog as defined in Definition 7 and the discrete logarithm problem in any specific group $\mathbb{Z}_p^*$. In particular, we argue that the latter is unlikely to be PWPP-complete. We start with the definition of DLog$_p$.

Definition 11 (DLog$_p$).

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

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

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

The second condition in the previous definition ensures that g is a generator of $\mathbb{Z}_p^*$ due to the Lagrange theorem. If g was not a generator, then there would be a $q \in \{1, \dots, p-2\}$ such that $g^q = 1$. We consider the smallest such q. Then $(g, g^2, g^3, \dots, g^q)$ forms a subgroup of $\mathbb{Z}_p^*$, and, from the Lagrange theorem, we get that $q = p_1^{l_1} \cdot \dots \cdot p_n^{l_n}$ such that at least one $l_i < k_i$. Then $g^{(p-1)/p_i}$ is a power of g^q, implying that $g^{(p-1)/p_i} = 1$, which would be a contradiction with the second condition in Definition 11. Hence, g is a generator of $\mathbb{Z}_p^*$.

Our first observation is a straightforward upper bound for DLog$_p$ that follows by showing its inclusion in PWPP.

Lemma 10. *DLog$_p$ is reducible to DLog.*

Proof. Given an instance $(p, p_1, \dots, p_n, k_1, \dots, k_n, g, y)$ of DLog$_p$, we first fix a representation of $\mathbb{Z}_p^*$ by $[p-1]$, i.e, we represent an element $a \in \mathbb{Z}_p^*$ by $a - 1$. Then, we construct the natural instance (s, f, id, g', t) of DLog, where

- $s = p - 1$,
- f implements multiplication in $\mathbb{Z}_p^*$ w.r.t. the fixed representation of $\mathbb{Z}_p^*$,
- id is the representation of $1 \in \mathbb{Z}_p^*$ as an element of $[s]$, i.e, $id = 0$,
- g' is the representation of the given generator g of $\mathbb{Z}_p^*$ as an element of $[s]$,
- t is the representation of $y \in \mathbb{Z}_p^*$ as an element of $[s]$.

Now, we show that any solution to this instance (s, f, id, g', t) of DLog gives a solution to the original instance of DLog$_p$. There are five types of a solution in DLog:

1. $a \in [s]$ such that $\mathcal{I}_{\mathbb{G}}(a) = t$. Since f corresponds to a valid group operation, it holds that $\mathcal{I}_{\mathbb{G}}(a) = g^a$. Hence, $g^a = t = y$ and $a \in [s] = \{0, \ldots, p-2\}$ is a solution to the original instance of DLog$_p$.

2. $a, b \in [s]$ such that $f_{\mathbb{G}}(a, b) \geq s$. Since f corresponds to a valid group operation, this case cannot happen.

3. $a, b \in [s]$ such that $a \neq b$ and $\mathcal{I}_{\mathbb{G}}(a) = \mathcal{I}_{\mathbb{G}}(b)$. Since f corresponds to a valid group operation, we get that $g^a = \mathcal{I}_{\mathbb{G}}(a) = \mathcal{I}_{\mathbb{G}}(b) = g^b$. Suppose without loss of generality that $a > b$. Then, the previous relationship implies that $g^{a-b} = 1$, where $a - b \neq 0$ and $a - b < p - 1$. This would be a contradiction with the fact the g is a generator of $\mathbb{Z}_p^*$. Hence, this case cannot happen.

4. $a, b \in [s]$ such that $a \neq b$ and $f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(a)) = f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(b))$. Since f corresponds to a valid group operation, the previous equality implies that $y \cdot g^a = y \cdot g^b$. By cancelling y, we get that $g^a = g^b$ with $a \neq b$ and $a, b \in [s]$. For the same reason as in the previous case, this case cannot happen.

5. $a, b \in [s]$ such that $\mathcal{I}_{\mathbb{G}}(a) = f_{\mathbb{G}}(t, \mathcal{I}_{\mathbb{G}}(b))$ and $\mathcal{I}_{\mathbb{G}}(a - b \bmod s) \neq t$. Since f corresponds to a valid group operation, we get that $g^a = y \cdot g^b$ and $g^{a-b} = g^{a-b \bmod s} \neq y$, which is impossible. Hence, this case cannot happen.

□

Note that the proof of Lemma 10 shows that the index function defined by taking the respective powers of g is a bijection and any instance of DLog$_p$ has a *unique* solution. Thus, there is a stronger upper bound on the complexity of DLog$_p$ in terms of containment in the class TFUP, i.e., the subclass of TFNP of total search problems with syntactically guaranteed unique solution for every instance.

Corollary. DLog$_p \in$ TFUP.

Even though the class TFUP was not extensively studied, the existence of a reduction of an arbitrary instance of Collision to a search problem with a unique solution for all instances seems implausible. Thus, we conjecture that DLog$_p$ cannot be PWPP-complete.

4.2 Blichfeldt

Both our reductions establishing PWPP-hardness of DLog and PPP-hardness of Index result in instances that induce groupoids unlikely to satisfy the group axioms. In other words, the resulting instances do not really correspond to DLP in any group. It is natural to ask whether this property is common to other PWPP and PPP hardness results. In this section, we revisit the problem Blichfeldt introduced in [3] and show that it also exhibits a similar phenomenon.

Below, we use the natural extension of the bit composition and decomposition functions when applied to vectors. We start with the definition of Blichfeldt.

Definition 12 (Blichfeldt).

Instance: An n-dimensional basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$, $s \in \mathbb{N}$ and a Boolean circuit V with $k = \lceil \log(s) \rceil$ binary inputs and l outputs defining a set of vectors $S \subseteq \mathbb{Z}^n$ as $S = \{\mathrm{bc}\left(\mathsf{V}(\mathrm{bd}\left(i\right))\right), i \in [s]\}$.

Solution: If $s < \det(\mathcal{L}(\mathbf{B}))$, then the vector 0^n. Otherwise, one of the following:

1. *strings $u, v \in \{0,1\}^n$ such that $u \neq v$ and $\mathsf{V}(u) = \mathsf{V}(v)$,*
2. *a vector x such that $x \in S \cap \mathcal{L}(\mathbf{B})$,*
3. *vectors $x \neq y$ such that $x, y \in S$ and $x - y \in \mathcal{L}(\mathbf{B})$.*

In their work, [3] showed that Blichfeldt is PPP-hard by a reduction from Pigeon that relies on some non-trivial properties of q-ary lattices. We show that this is unnecessary and give a more direct reduction that exploits the circuit V in the definition of Blichfeldt. One particularly interesting property of our reduction is that it completely bypasses the solutions in Blichfeldt corresponding to the Blichfeldt's theorem. Specifically, all the instances produced by our reduction are defined w.r.t. the same basis $\mathbf{B}$.

Lemma 11. *Blichfeldt is PPP-hard.*

Proof. We show a reduction from Pigeon to Blichfeldt. We start with an arbitrary instance $\mathsf{C} : \{0,1\}^n \to \{0,1\}^n$ of Pigeon. If $\mathsf{C}(0^n) = 0^n$, then we output 0^n as a solution to this instance C without invoking the Blichfeldt oracle. Otherwise, we construct an instance of Blichfeldt as follows:

- We define $\mathbf{B} = 2 \cdot I_n$, i.e., the $n \times n$ diagonal matrix with 2's on its diagonal and 0's elsewhere.
- We set $s = 2^n$.
- We define the circuit $\mathsf{V} : \{0,1\}^n \to \{0,1\}^n$ as follows:

$$\mathsf{V}(x) = \begin{cases} \mathsf{C}(x) & \text{if } \mathsf{C}(x) \neq 0^n, \\ \mathsf{C}(0^n) & \text{otherwise.} \end{cases}$$

Note that for the set S corresponding to this instance of Blichfeldt, bc maps any binary string $x = (x_1, \ldots, x_n)$ output by V to an identical vector $(x_1, \ldots, x_n)^T$ in $\mathbb{Z}^n$. In particular, all coordinates are either 0 or 1 for any vector from the set S defined by s and V from the above instance of Blichfeldt.

We now show that any solution to the above Blichfeldt instance gives a solution to the original Pigeon instance C. First, notice that $\det(\mathbf{B}) = 2^n = s$, so a solution satisfies one of the cases $1, 2$, or 3 from Definition 12:

1. The solution is $u, v \in \{0,1\}^n$ such that $u \neq v$ and $\mathsf{V}(u) = \mathsf{V}(v)$. If $\mathsf{C}(u) = 0^n$, then u is a solution to the original Pigeon instance C, case 1. Similarly, if $\mathsf{C}(v) = 0^n$, then v is a solution to the original Pigeon instance C, case 1. Otherwise, it holds that $\mathsf{C}(u) \neq 0^n \neq \mathsf{C}(v)$. Hence, from the definition of V, we get that

$$\mathsf{C}(u) = \mathsf{V}(u) = \mathsf{V}(v) = \mathsf{C}(v),$$

and the pair u, v is a solution to the original Pigeon instance C, case 2.

2. The solution is a vector x such that $x \in S \cap \mathcal{L}(\mathbf{B})$. From the definition of the set S, it holds that $x \in \{0,1\}^n$. Moreover, from the definition of V and the fact that $\mathsf{C}(0^n) \neq 0^n$, we get that $0^n \notin S$, but 0^n is the only vector in $\{0,1\}^n \cap \mathcal{L}(\mathbf{B})$. Hence, this case cannot happen.

3. The solution is a pair of vectors x, y such that $x \neq y$, $x, y \in S$ and $x - y \in \mathcal{L}(\mathbf{B})$. But we know that all vectors in S have coefficients in $\{0,1\}$, so all coefficients of $x - y$ would lie in $\{-1,0,1\}$, but the only such vector also contained in $\mathcal{L}(\mathbf{B})$ is 0^n, which would imply $x = y$. Hence, this case cannot happen.

$\square$

Conclusion

In our work, we focused on the discrete logarithm problem in the context of classes PPP and PWPP as suggested in [3] in 2018. Concretely, motivated by [3] and the known construction of collision-resistant hash functions from [4], we introduced two computational problems called Index and DLog.

In the second chapter, we answered the open problem from [3] as we showed that Index is PPP-complete. The reduction from Index to Pigeon showing that Index lies in PPP was the more straightforward part, whereas the reduction from Pigeon to Index showing PPP-hardness of Index was arguably the most technical part of the thesis. We had to carefully define the circuit f determining the instance of Index such that the index function $\mathcal{I}_{\mathbb{G}}$ "emulates" the computation of the circuit C of the Pigeon instance.

In the third chapter, we showed that DLog, which is a relaxation of Index, is PWPP-complete. The reduction from DLog to Collision showing that DLog lies in PWPP was inspired by the construction of collision-resistant hash functions from the discrete logarithm problem via claw-free permutations from [4]. Here, we had to use the homomorphic properties of the induced groupoid ensured by the additional types of a solution in DLog. On the other hand, the reduction from Collision to DLog showing PWPP-hardness of DLog is completely new and goes through an intermediate problem we call Dove.

Additionally, the reductions showing PWPP-completeness of DLog provide new structural insights into PWPP by establishing two new PWPP-complete problems. First, the problem Dove, a relaxation of the PPP-complete problem Pigeon. Dove is the first PWPP-complete problem not defined in terms of an explicitly shrinking function. Second, the problem Claw, a total search problem capturing the computational complexity of breaking claw-free permutations. In the context of TFNP, the PWPP-completeness of Claw matches the known intrinsic relationship between collision-resistant hash functions and claw-free permutations established in the cryptographic literature.

In the last chapter, we focused on the discrete logarithm problem in $\mathbb{Z}_p^*$ called DLog$_p$ and the problem motivated by the Blichfeldt's theorem called Blichfeldt [3] in the context of TFNP. We showed that DLog$_p$ lies in PWPP by presenting a reduction from DLog$_p$ to DLog and we gave evidence that DLog$_p$ is not PWPP-hard. We also revisited the proof of PPP-hardness of Blichfeldt from [3]. We provided a more straightforward reduction from Pigeon to Blichfeldt which does not rely on any non-trivial properties of q-ary lattices and which completely bypasses the solutions in Blichfeldt corresponding to the underlying Blichfeldt's theorem.

Bibliography

[1] Nimrod Megiddo and Christos H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theor. Comput. Sci.*, 81(2):317–324, 1991.

[2] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.

[3] Katerina Sotiraki, Manolis Zampetakis, and Giorgos Zirdelis. PPP-completeness with connections to cryptography. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 148–158. IEEE Computer Society, 2018.

[4] Ivan Damgård. Collision free hash functions and public key signature schemes. In David Chaum and Wyn L. Price, editors, *Advances in Cryptology - EUROCRYPT '87, Workshop on the Theory and Application of of Cryptographic Techniques, Amsterdam, The Netherlands, April 13-15, 1987, Proceedings*, volume 304 of *Lecture Notes in Computer Science*, pages 203–216. Springer, 1987.

[5] Alexander Russell. Necessary and sufficient condtions for collision-free hashing. *J. Cryptol.*, 8(2):87–100, 1995.

[6] Emil Jeřábek. Integer factoring and modular square roots. *J. Comput. Syst. Sci.*, 82(2):380–394, 2016.

[7] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. Duality between two cryptographic primitives. In Shojiro Sakata, editor, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, 8th International Symposium, AAECC-8, Tokyo, Japan, August 20-24, 1990, Proceedings*, volume 508 of *Lecture Notes in Computer Science*, pages 379–390. Springer, 1990.