On Search Complexity of Discrete Logarithm

Pavel Hubáček
Charles University, Prague, Czech Republic

Jan Václavek
Charles University, Prague, Czech Republic

Abstract

In this work, we study the discrete logarithm problem in the context of TFNP – the complexity class of search problems with a syntactically guaranteed existence of solutions for all instances. Our main results establish that suitable variants of the discrete logarithm problem are complete for the complexity class PPP, respectively PWPP, i.e., the subclasses of TFNP capturing total search problems with a solution guaranteed by the pigeonhole principle, respectively the weak pigeonhole principle. Besides answering an open problem from the recent work of Sotiraki, Zampetakis, and Zirdelis (FOCS'18), our completeness results for PPP and PWPP have implications for the recent line of work proving conditional lower bounds for problems in TFNP under cryptographic assumptions. In particular, they highlight that any attempt at basing average-case hardness in subclasses of TFNP (other than PWPP and PPP) on the average-case hardness of the discrete logarithm problem must exploit its structural properties beyond what is necessary for constructions of collision-resistant hash functions.

Additionally, our reductions provide new structural insights into the class 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.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases discrete logarithm, total search problems, completeness, TFNP, PPP, PWPP

Digital Object Identifier 10.4230/LIPIcs.MFCS.2021.60

Related Version *Full Version*: `https://arxiv.org/pdf/2107.02617`

Funding *Pavel Hubáček*: Supported by the Grant Agency of the Czech Republic under the grant agreement no. 19-27871X and by the Charles University projects PRIMUS/17/SCI/9 and UNCE/SCI/004.

Acknowledgements We wish to thank the anonymous reviewers for their helpful suggestions on the presentation of our results. The first author is grateful to Chethan Kamath for multiple enlightening discussions about TFNP and for suggesting Dove as a name for a total search problem contained in the complexity class PPP.

1 Introduction

The discrete logarithm problem (DLP) and, in particular, its conjectured average-case hardness lies at the foundation of many practical schemes in modern cryptography. To day, no significant progress towards a generic efficient algorithm solving DLP has been made (see, e.g., the survey by Joux, Odlyzko, and Pierrot [17] and the references therein).

One of the distinctive properties of DLP is its *totality*, i.e., given a generator g of a cyclic group $(\mathbb{G}, \star)$, we know that a solution x for DLP exists for any target element $t = g^x$ in the group. Thus, the perceived hardness of DLP does not stem from the uncertainty whether a solution exists but pertains to the search problem itself. In this respect, DLP is not unique – there are various total search problems with unresolved computational complexity in many domains such as algorithmic game theory, computation number theory, and combinatorial optimization, to name but a few. More generally, the complexity of all total search problems is captured by the complexity class TFNP.

In order to improve our understanding of the seemingly disparate problems in TFNP, Papadimitriou [20] suggested to classify total search problems based on syntactic arguments ensuring the existence of a solution. His approach proved to be extremely fruitful and it gave rise to various subclasses of TFNP that cluster many important total search problems. For example,

PPAD: formalizes parity arguments on directed graphs and captures, e.g., the complexity of computing Nash equilibria in bimatrix games [7, 3].

PPA: formalizes parity arguments on *undirected* graphs and captures, e.g., the complexity of Necklace splitting [9].

PPP: formalizes the pigeonhole principle and captures, e.g., the complexity of solving problems related to integer lattices [22].

PWPP: formalizes the *weak* pigeonhole principle and captures, e.g., the complexity of breaking collision-resistant hash functions and solving problems related to integer lattices [22].

DLP and TFNP

DLP seems to naturally fit the TFNP landscape. Though, a closer look reveals a subtle issue regarding its totality stemming from the need to certify that the given element g is indeed a generator of the considered group $(\mathbb{G}, \star)$ or, alternatively, that the target element t lies in the subgroup of $(\mathbb{G}, \star)$ generated by g. If the order $s = |\mathbb{G}|$ of the group $(\mathbb{G}, \star)$ is known then there are two natural approaches. The straightforward approach would be to simply allow additional solutions in the form of distinct $x, y \in [s] = \{0, \ldots, s-1\}$ such that $g^x = g^y$. By the pigeonhole principle, either $t = g^x$ for some $x \in [s]$ or there exists such a non-trivial collision $x, y \in [s]$. The other approach would be to leverage the Lagrange theorem that guarantees that the order of any subgroup must divide the order of the group itself. If we make the factorization of the order s of the group a part of the instance then it can be efficiently tested whether g is indeed a generator.

Despite being a prominent total search problem, DLP was not extensively studied in the context of TFNP so far. Only recently, Sotiraki, Zampetakis, and Zirdelis [22] presented a total search problem motivated by DLP. They showed that it lies in the complexity class PPP and asked whether it is complete for the complexity class PPP.

1.1 Our Results

In this work, we study formalizations of DLP as a total search problem and prove new completeness results for the classes PPP and PWPP.

Our starting point is the discrete logarithm problem in "general groups" suggested by Sotiraki et al. [22]. Given the order $s \in \mathbb{Z}$, $s > 1$, we denote by $\mathbb{G} = [s] = \{0, \ldots, s-1\}$ the canonical representation of a set with s elements. Any efficiently computable binary operation on $\mathbb{G}$ can be represented by a Boolean circuit $f\colon \{0,1\}^l \times \{0,1\}^l \to \{0,1\}^l$ that evaluates the operation on binary strings of length $l = \lceil \log(s) \rceil$ representing the elements of $\mathbb{G}$. Specifically, the corresponding binary operation $\star$ on $\mathbb{G}$ can be computed by first taking the binary representation of the elements $x, y \in \mathbb{G}$, evaluating f on the resulting strings, and mapping the value back to $\mathbb{G}$. Note that the binary operation $\star$ induced on $\mathbb{G}$ by f in this way might not satisfy the group axioms and, thus, we refer to $(\mathbb{G}, \star)$ as the induced *groupoid* adopting the terminology for a set with a binary operation common in universal algebra.

Assuming that $(\mathbb{G}, \star)$ is a cyclic group, we might be provided with the representations of the identity element $id \in \mathbb{G}$ and a generator $g \in \mathbb{G}$, which, in particular, enable us to efficiently access the group elements via an indexing function $\mathcal{I}_{\mathbb{G}} \colon [s] \to \mathbb{G}$ computed as the corresponding powers of g (e.g. via repeated squaring). An instance of a general DLP is then given by a representation (s, f) inducing a groupoid $(\mathbb{G}, \star)$ together with the identity element $id \in \mathbb{G}$, a generator $g \in \mathbb{G}$, and a target $t \in \mathbb{G}$; a solution for the instance (s, f, id, g, t) is either an index $x \in [s]$ such that $\mathcal{I}_{\mathbb{G}}(x) = t$ or a pair of distinct indices $x, y \in [s]$ such that $\mathcal{I}_{\mathbb{G}}(x) = \mathcal{I}_{\mathbb{G}}(y)$. Note that the solutions corresponding to non-trivial collisions in $\mathcal{I}_{\mathbb{G}}$ ensure totality of the instance irrespective of whether the induced groupoid $(\mathbb{G}, \star)$ satisfies the group axioms – the indexing function $\mathcal{I}_{\mathbb{G}}$ either has a collision or it is a bijection and must have a preimage for any t.

The general DLP as defined above can clearly solve DLP in specific groups with efficient representation such as any multiplicative group $\mathbb{Z}_p^*$ of integers modulo a prime p, which are common in cryptographic applications. On the other hand, it allows for remarkably unstructured instances and the connection to DLP is rather loose – as we noted above, the general groupoid $(\mathbb{G}, \star)$ induced by the instance might not be a group, let alone cyclic. Therefore, we refer to this search problem as Index (see Definition 15 in Section 4 for the formal definition).

A priori, the exact computational complexity of Index is unclear. Sotiraki et al. [22] showed that it lies in the class PPP by giving a reduction to the PPP-complete problem Pigeon, where one is asked to find a preimage of the 0^n string or a non-trivial collision for a function from $\{0,1\}^n$ to $\{0,1\}^n$ computed by a Boolean circuit given as an input. No other upper or lower bound on Index was shown in [22]. Given that DLP can be used to construct collision-resistant hash functions [6], it seems natural to ask whether Index lies also in the class PWPP, a subclass of PPP defined by the canonical problem Collision, where one is asked to find a collision in a shrinking function computed by a Boolean circuit given as an input.

However, a closer look at the known constructions of collision-resistant hash functions from DLP reveals that they crucially rely on the homomorphic properties of the function $g^x = \mathcal{I}_{\mathbb{G}}(x)$. Given that $(\mathbb{G}, \star)$ induced by an arbitrary instance of Index does not necessarily posses the structure of a cyclic group, the induced indexing function $\mathcal{I}_{\mathbb{G}}$ is not guaranteed to have any homomorphic properties and it seems unlikely that Index could be reduced to any PWPP-complete problem such as Collision. In Section 4, we establish that the above intuition is indeed correct since our Theorem 1 shows that Index is PPP-complete:

▶ **Theorem 1.** *Index is PPP-complete.*

On the other hand, we show that, by introducing additional types of solutions in the Index problem, we can enforce sufficient structure on the induced groupoid $(\mathbb{G}, \star)$ that allows for a reduction to the PWPP-complete problem Collision. First, we add a solution type witnessing that the coset of t is not the whole $\mathbb{G}$, i.e., that $\{t \star a \mid a \in \mathbb{G}\} \neq \mathbb{G}$, which cannot be the case in a group. Specifically, a solution is also any pair of distinct $x, y \in [s]$ such that $t \star \mathcal{I}_{\mathbb{G}}(x) = t \star \mathcal{I}_{\mathbb{G}}(y)$. Second, we add a solution enforcing some form of homomorphism in $\mathcal{I}_{\mathbb{G}}$ with respect to t. Specifically, a solution is also any pair of $x, y \in [s]$ such that $\mathcal{I}_{\mathbb{G}}(x) = t \star \mathcal{I}_{\mathbb{G}}(y)$ and $\mathcal{I}_{\mathbb{G}}(x - y \mod s) \neq t$. The second type of a solution is motivated by the classical construction of a collision-resistant hash function from DLP by Damgård [6]. Notice that if there are no solutions of the second type then any pair x, y such that $\mathcal{I}_{\mathbb{G}}(x) = t \star \mathcal{I}_{\mathbb{G}}(y)$ gives rise to the preimage of t under $\mathcal{I}_{\mathbb{G}}$ by simply computing $x - y \mod s$. We refer to the version of Index with the additional two types of solutions as DLog (see Definition 5 in Section 3 for the formal definition), as it is in our opinion close enough to the standard DLP in cyclic groups.

Since DLOG is a relaxation of INDEX obtained by allowing additional types of solutions, it could be the case that we managed to reduce DLOG to COLLISION simply because DLOG is trivial. Note that this is not the case since DLOG is at least as hard as DLP in any cyclic group with an efficient representation, where DLP would naturally give rise to an instance of DLOG with a unique solution corresponding to the solution for the DLP. In Section 3, we establish that DLOG is at least as hard as the problem of finding a non-trivial collision in a shrinking function by proving Theorem 2 that shows that DLOG is PWPP-complete:

▶ **Theorem 2.** *DLOG is* PWPP*-complete.*

Implications for cryptographic lower bounds for subclasses of TFNP

It was shown already by Papadimitriou [20] that cryptographic hardness might serve as basis for arguing the existence of average-case hardness in subclasses of TFNP. A recent line of work attempts to show such cryptographic lower bounds for subclasses of TFNP under increasingly more plausible cryptographic hardness assumptions [16, 2, 10, 13, 11, 18, 4, 5, 8, 1, 19, 15]. However, it remains an open problem whether DLP can give rise to average-case hardness in subclasses of TFNP other than PWPP and PPP. Our results highlight that any attempt at basing average-case hardness in subclasses of TFNP (other than PWPP and PPP) on the average-case hardness of the discrete logarithm problem must exploit its structural properties beyond what is necessary for constructions of collision-resistant hash functions.

Witnessing totality of number theoretic problems

In the full version [12], we discuss some of the issues that arise when defining total search problems corresponding to actual problems in computational number theory. First, we highlight some crucial distinctions between the general DLOG as defined in Definition 5 and the discrete logarithm problem in multiplicative groups $\mathbb{Z}_p^*$. In particular, we argue that the latter is unlikely to be PWPP-complete.

Second, we clarify the extent to which our reductions exploit the expressiveness allowed by the representations of instances of DLOG and INDEX. In particular, both the reduction from COLLISION to DLOG and from PIGEON to INDEX output instances that induce groupoids unlikely to satisfy group axioms and, therefore, do not really correspond to DLP. Additionally, we revisit the problem BLICHFELDT introduced in [22] and show that it also exhibits a similar phenomenon in the context of computational problems on integer lattices.

Alternative characterizations of PWPP

Our PWPP-completeness result for DLOG is established via a series of reductions between multiple intermediate problems, which are thus also PWPP-complete. We believe this characterization will prove useful in establishing further PWPP-completeness results. These new PWPP-complete problems are defined in Section 3 and an additional discussion is provided in Section 5.

2 Preliminaries

We denote by $[m]$ the set $\{0, 1, \dots, m-1\}$, by $\mathbb{Z}^+$ the set $\{1, 2, 3, \dots\}$ of positive integers, and by $\mathbb{Z}_0^+$ the set $\{0, 1, 2, \dots\}$ of non-negative integers. 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 $\oplus$.

Bit composition and decomposition

Throughout the paper, we often make use of the 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 zeroes such that its length is k. When clear from the context, we omit k and write simply bc and bd to improve readability. At places, we work with the output of bd^k without the leading zeroes. We denote by $\mathrm{bd}_0 : \mathbb{Z}_0^+ \to \{0,1\}^*$ the standard function which computes the binary representation without the leading zeroes.

TFNP and some of its subclasses

A *total* NP *search problem* is a relation $S \subseteq \{0,1\}^* \times \{0,1\}^*$ such that: 1) the decision problem whether $(x,y) \in S$ is computable in polynomial-time in $|x| + |y|$, and 2) there exists a polynomial q such that for all $x \in \{0,1\}^*$, there exists a $y \in \{0,1\}^*$ such that $(x,y) \in S$ and $|y| \leq q(|x|)$. The class of all total NP search problems is denoted by TFNP.

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 case there exists a reduction from S to T, we say that *S is reducible to T*. The above corresponds to so-called polynomial-time many-one (or Karp) reductions among decision problems in the context of search problems. In the rest of the paper, we consider only such reductions.

▶ **Definition 3** (Pigeon problem and PPP [20]).
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 $C(u) = 0^n$,*
2. *distinct strings $u, v \in \{0,1\}^n$ such that $C(u) = C(v)$.*

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

▶ **Definition 4** (Collision problem and PWPP [16]).
Instance: A Boolean circuit C with n inputs and m outputs with $m < n$.
Solution: Distinct strings $u, v \in \{0,1\}^n$ such that $C(u) = C(v)$.

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

3 DLog is PWPP-complete

In this section, we define DLog, a total search problem associated to DLP and show that it is PWPP-complete. Our reductions give rise to additional new PWPP-complete problems Dove and Claw, which we discuss further in Section 5.

Similarly to Sotiraki et al. [22], we represent a binary operation on $\mathbb{G} = [s] = \{0, \dots, s-1\}$ 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 $x, y \in [s]$ as $f_{\mathbb{G}}(x,y) = \mathrm{bc}(f(\mathrm{bd}(x), \mathrm{bd}(y)))$. We denote by $(\mathbb{G}, \star)$ the groupoid 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 $x \star y = 1$ for all $x, y \in [s]$ such that $f_{\mathbb{G}}(x,y) \notin [s]$.

Algorithm 1 Computation of the x-th power of the generator $g \in [s]$ of a groupoid $(\mathbb{G}, \star)$ of size $s \in \mathbb{Z}_0^+$ induced by $f\colon \{0,1\}^{2\lceil\log(s)\rceil} \to \{0,1\}^{\lceil\log(s)\rceil}$ with identity $id \in [s]$.

```
1:  procedure $\mathcal{I}_{\mathbb{G}}(x)$
2:      $(x_m, \ldots, x_1) \leftarrow \mathrm{bd}_0(x)$
3:      $r \leftarrow \mathrm{bd}(id)$
4:      $g \leftarrow \mathrm{bd}(g)$
5:      for $i$ from $m$ to $1$ 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
```

If the induced groupoid $(\mathbb{G}, \star)$ was a cyclic group then we could find the indices of the identity element $id \in [s]$ and a generator $g \in [s]$. 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, and the corresponding *indexing 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. The algorithm first computes $(x_m, x_{m-1}, \ldots, x_1) = \mathrm{bd}_0(x)$, i.e., the binary representation of the exponent x without the leading zeroes for some $m \leq l$, and it then proceeds with the square-and-multiply method using the circuit f. As explained above, f implements the binary group operation. Hence, $f(r, r)$ corresponds to squaring the intermediate value r and $f(g, r)$ corresponds to multiplication of the intermediate value r by the generator g.

With the above notation in place, we can give the formal definition of DLOG.

▶ **Definition 5** (DLOG problem).

INSTANCE: A size parameter $s \in \mathbb{Z}^+$ such that $s \geq 2$ 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 $id, g, t \in [s]$.

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 $\mathcal{I}_{\mathbb{G}}(x - y \bmod s) \neq t$.*

The first type of a solution in DLOG corresponds to the discrete logarithm of t. Since we cannot efficiently verify that the input instance represents a group with the purported generator g, additional types of a solution had to be added in order to guarantee that DLOG is total. Note that any solution of these additional types witnesses that the instance does not induce a group, since for a valid group these types cannot happen. Nevertheless, the first three types of a solution are sufficient to guarantee the totality of DLOG. The last two types of a solution make DLOG to lie in the class PWPP and are crucial for correctness of our reduction from DLOG to COLLISION presented in Section 3.2. In Section 5, we provide further discussion of DLOG and some possible alternative definitions.

In Section 3.1, we show that DLOG is PWPP-hard. In Section 3.2, we show that DLOG lies in PWPP. Therefore, we prove Theorem 2.

▶ **Theorem 2.** *DLog is* PWPP*-complete.*

3.1 DLog is PWPP-hard

To show that DLog is PWPP-hard, we reduce to it from the PWPP-complete problem Collision (see Definition 4). Given an instance $\mathsf{C}\colon \{0,1\}^n \to \{0,1\}^{n-1}$ of Collision, our reduction to DLog defines a representation (s, f) of a groupoid $(\mathbb{G}, \star)$ and the elements id, g, and t such that we are able to extract some useful information about C from any non-trivial collision $\mathcal{I}_{\mathbb{G}}(x) = \mathcal{I}_{\mathbb{G}}(y)$ in the indexing function $\mathcal{I}_{\mathbb{G}}$ computed by Algorithm 1. The main obstacle that we need to circumvent is that, even though the computation performed by $\mathcal{I}_{\mathbb{G}}$ employs the circuit f representing the binary operation in the groupoid, it has a very restricted form. In particular, we need to somehow define f using C so that there are no collisions in $\mathcal{I}_{\mathbb{G}}$ unrelated to solutions of the instance of Collision. To sidestep some of the potential issues when handling an arbitrary instance of Collision, we reduce to DLog from an intermediate problem we call Dove.

▶ **Definition 6** (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. *distinct strings $u, v \in \{0,1\}^n$ such that $\mathsf{C}(u) = \mathsf{C}(v)$,*
4. *distinct strings $u, v \in \{0,1\}^n$ such that $\mathsf{C}(u) = \mathsf{C}(v) \oplus 0^{n-1}1$.*

It is immediate that Dove is a relaxation of Pigeon (cf. Definition 3) 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 [23] under the term *distinction-intractable* permutations. Zheng et al. showed that distinction-intractability is sufficient for collision-resistant hashing. Note that we employ distinction-intractability in a different way than [23]. 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) – we use Dove as an intermediate problem when reducing from Collision to DLog (proving PWPP-hardness of DLog). In the overview of the reduction from Dove to DLog below, we explain why distinction-intractability seems as a natural choice for our 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 $f_{\mathbb{G}}$. Our initial observation is that, by the definition of $\mathcal{I}_{\mathbb{G}}$ (Algorithm 1), the circuit f is only applied on specific types of inputs during the computation of $\mathcal{I}_{\mathbb{G}}(x)$. Specifically:

- In each loop, $f(r, r)$ is computed for some $r \in \{0,1\}^*$. We denote f restricted to this type of inputs by f_0, i.e., $f_0(r) = f(r, r)$.
- If the corresponding bit of x is one then $f(g, r)$ is computed with fixed $g \in \{0,1\}^*$ and some $r \in \{0,1\}^*$. We denote f restricted to this type of inputs by f_1, i.e., $f_1(r) = f(g, r)$.

Hence, using the above notation, 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 of x evaluated on id (e.g., $\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(101)) = f_1 \circ f_0 \circ f_0 \circ f_1 \circ f_0(\mathrm{bd}(id))$). Exploiting the observed structure of the computation of $\mathcal{I}_{\mathbb{G}}$, our approach is to define f_0 and f_1 (i.e., the corresponding part of f) using the circuit C so that we can extract some useful information about C from any non-trivial collision $\mathcal{I}_{\mathbb{G}}(x) = \mathcal{I}_{\mathbb{G}}(y)$ (i.e., from a solution to DLog, case 3).

The straightforward option is 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 $u, v \in \{0,1\}^n$ with Hamming weight l, there would be an easy to find non-trivial collision $x = \mathrm{bc}(u)$ and $y = \mathrm{bc}(v)$ of the form $\mathcal{I}_{\mathbb{G}}(x) = \mathrm{bc}(\mathsf{C}^{n+l}(id)) = \mathcal{I}_{\mathbb{G}}(y)$, which might not provide any useful information about the circuit C. 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 case, a non-trivial collision $\mathcal{I}_{\mathbb{G}}(x) = \mathcal{I}_{\mathbb{G}}(y)$ corresponds to the identity $\mathsf{C}(\mathsf{C}(u)) = \mathsf{C}(h(\mathsf{C}(v)))$ for some $u, v \in \{0,1\}^n$, which are not necessarily distinct. In particular, if $\mathsf{C}(u) \neq h(\mathsf{C}(v))$ then the pair of strings $\mathsf{C}(u), h(\mathsf{C}(v))$ forms a non-trivial collision for C. Otherwise, we found a pair u, v such that $\mathsf{C}(u) = h(\mathsf{C}(v))$ that, for the choice $h(y) = y \oplus 0^{n-1}1$, translates into $\mathsf{C}(u) = \mathsf{C}(v) \oplus 0^{n-1}1$, i.e., a pair of inputs breaking distinction-intractability of C, and corresponds to the fourth type of a solution in Dove. Finally, the second type of a solution in Dove captures the special case when there is no pair u, v such that $\mathsf{C}(\mathsf{C}(u)) = \mathsf{C}(h(\mathsf{C}(v)))$. The formal reduction from Dove to DLog establishing Lemma 7 below is given in the full version [12].

▶ **Lemma 7.** *Dove is reducible to DLog.*

PWPP-hardness of Dove. Next, we show that, by introducing additional types of solutions 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. Our reduction from Collision to Dove is rather syntactic and natural. In particular, it results in instances of Dove with only one type of solutions corresponding to collisions of the original instance of Collision. For the formal proof, see the full version [12].

▶ **Lemma 8.** *Collision is reducible to Dove.*

Lemma 7 and Lemma 8 imply PWPP-hardness of DLog as stated in the corollary below.

▶ **Corollary 9.** *DLog is PWPP-hard.*

3.2 DLog Lies in PWPP

In order to establish that DLog lies in PWPP, we build on the existing cryptographic literature on constructions of collision-resistant hash functions from the discrete logarithm problem. Specifically, we mimic the classical approach by Damgård [6] to first construct a family of *claw-free permutations* based on DLP and then define a collision-resistant hash using the family of claw-free permutations.[1] Recall that a family of claw-free permutations is an efficiently sampleable family of pairs of permutations such that given a “random” pair

[1] In principle, it might be possible to adapt any alternative known construction of collision-resistant hash from DLP such as the one of Ishai, Kushilevitz, and Ostrovsky [14], which goes through the intermediate object of *homomorphic one-way commitments*. However, this would necessitate not only the corresponding changes in the definition of DLog but also an alternative proof of its PWPP-hardness.

h_0 and h_1 of permutations from the family, it is computationally infeasible to find a *claw* for the 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 10** (CLAW problem).
INSTANCE: Two Boolean circuits h_0, h_1 with n inputs and n outputs.
SOLUTION: One of the following:

1. *two strings $u, v \in \{0,1\}^n$ such that $h_0(u) = h_1(v)$,*
2. *two distinct strings $u, v \in \{0,1\}^n$ such that $h_0(u) = h_0(v)$,*
3. *two distinct strings $u, v \in \{0,1\}^n$ such that $h_1(u) = h_1(v)$.*

The first type of a solution in CLAW 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 solutions which witness that one of the functions is not bijective. In other words, the second and third type of solution ensure the totality of CLAW.

Similarly to [6], our high-level approach when reducing from DLOG to COLLISION is to first reduce from DLOG to CLAW and then 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 [21] 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 [21]. As for the first issue, we manage to provide a formal reduction from DLOG to GENERAL-CLAW, a variant of CLAW defined below.

▶ **Definition 11** (GENERAL-CLAW problem).
INSTANCE: Two Boolean circuits h_0, h_1 with n inputs and n outputs and $s \in \mathbb{Z}^+$ such that $1 \le s < 2^n$.
SOLUTION: One of the following:

1. *two 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. *two distinct strings $u, v \in \{0,1\}^n$ such that $h_0(u) = h_0(v)$,*
3. *two distinct strings $u, v \in \{0,1\}^n$ such that $h_1(u) = h_1(v)$,*
4. *a string $u \in \{0,1\}^n$ such that $\mathrm{bc}(u) < s$ and $\mathrm{bc}(h_0(u)) \ge s$,*
5. *a string $u \in \{0,1\}^n$ such that $\mathrm{bc}(u) < s$ and $\mathrm{bc}(h_1(u)) \ge s$.*

The main issue that necessitates the introduction of additional types of a solution in the definition of GENERAL-CLAW (compared to CLAW) is that the possible solutions to an instance of DLOG are not from the whole domain $[2^n]$ but they must lie in $[s]$.

The formal reductions proving Lemma 12 and Lemma 13 below are presented in the full version [12]. The two lemmata establish Corollary 14, which concludes the proof of Theorem 2.

▶ **Lemma 12.** *DLOG is reducible to GENERAL-CLAW.*

▶ **Lemma 13.** *GENERAL-CLAW is reducible to COLLISION.*

▶ **Corollary 14.** *DLOG is contained in* PWPP.

4 Index is PPP-complete

In this section, we study the complexity of a more restricted version of DLog that we call Index. In the definition of Index, we use the notation from Section 3 introduced for the definition of DLog. In particular, the function $\mathcal{I}_{\mathbb{G}}$ is the same as defined in Algorithm 1.

▶ **Definition 15** (Index problem).
Instance: *A size parameter $s \in \mathbb{Z}^+$ such that $s \geq 2$ 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 $x \neq y$ and $f_{\mathbb{G}}(x, y) \geq s$,*
3. *$x, y \in [s]$, such that $x \neq y$ and $\mathcal{I}_{\mathbb{G}}(x) = \mathcal{I}_{\mathbb{G}}(y)$.*

It is immediate that DLog is a relaxation of Index due to the additional types of solutions. In Section 4.1, we show that Index is PPP-hard. In Section 4.2, we show that Index lies in PPP. Therefore, we prove PPP-completeness of Index.

▶ **Theorem 1.** *Index is* PPP*-complete.*

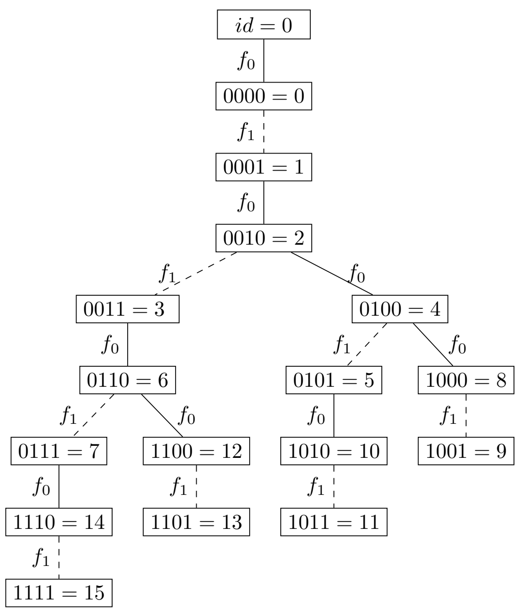
4.1 Index is PPP-hard

The formal reduction from the PPP-complete problem Pigeon to Index is arguably the most technical. Given an instance $\mathsf{C}\colon \{0,1\}^n \to \{0,1\}^n$ of Pigeon, the main idea is to define an instance $G = (s, f, id, g, t)$ of Index such that the induced indexing function $\mathcal{I}_{\mathbb{G}}$ carefully "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 representing the binary operation in the groupoid $(\mathbb{G}, \star)$. 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 the two special types of calls to f (i.e., squaring the intermediate value or multiplying it by the generator). 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 indexing function $\mathcal{I}_{\mathbb{G}}$ outputs the corresponding evaluation of C at each leaf. Moreover, for the remaining nodes in the tree, $\mathcal{I}_{\mathbb{G}}$ results in a bijection to ensure there are no additional solutions of the constructed instance of Index that would be unrelated to the original instance of Pigeon. Below, we provide additional details of the ideas behind the formal reduction given in the full version [12].

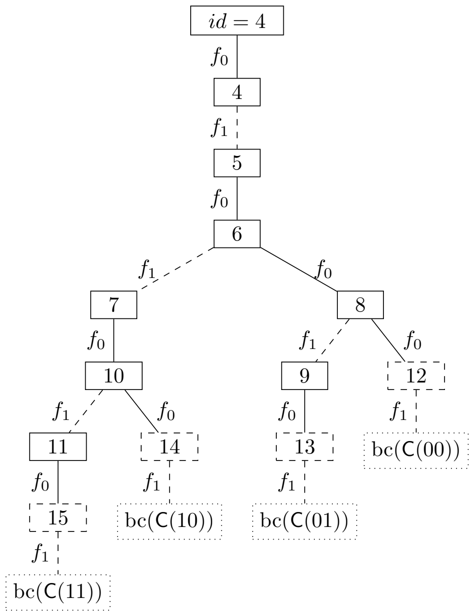
Similarly to the reduction from Dove to DLog, the key step in our construction of G is a suitable choice of the circuit f since it determines the function $\mathcal{I}_{\mathbb{G}}$. Recall the notation for f_0 and f_1 introduced in the reduction from Dove to DLog, i.e., $f_0(r) = f(r, r)$ and $f_1(r) = f(g, r)$. We start by describing a construction of an induced groupoid $(\mathbb{G}, \star)$ independent of the instance C of Pigeon but which serves as a natural step towards our reduction.

Constructing bijective $\mathcal{I}_{\mathbb{G}}$

Our initial goal in the first construction is to 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 $a \in [s]$. To this end, our key observation is that, for many pairs of inputs $a, b \in [s]$, the computation of $\mathcal{I}_{\mathbb{G}}(b)$ includes the whole computation of $\mathcal{I}_{\mathbb{G}}(a)$ as a prefix (see Algorithm 1), e.g., for all $a, b \in [s]$ such that



(a) $\mathcal{I}_{\mathbb{G}}(a) = a$.



(b) C incorporated.

Figure 1 Trees induced by the computation of $\mathcal{I}_{\mathbb{G}}$.

- either $\mathrm{bd}_0(a)$ is a prefix of $\mathrm{bd}_0(b)$
- or $\mathrm{bd}_0(a) = y||0$ and $\mathrm{bd}_0(b) = y||1$ for some $y \in \{0,1\}^*$.

Specifically, if $\mathrm{bd}_0(a) = y||0$ then $\mathcal{I}_{\mathbb{G}}(a) = \mathrm{bc}(f_0(\mathrm{bd}(\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(y)))))$, and if $\mathrm{bd}_0(a) = y||1$ then $\mathcal{I}_{\mathbb{G}}(a) = \mathrm{bc}(f_1(\mathrm{bd}(\mathcal{I}_{\mathbb{G}}(\mathrm{bc}(y||0)))))$.

Thus, we can capture the whole computation of $\mathcal{I}_{\mathbb{G}}$ on all the possible inputs from $\mathbb{G}$ via a tree representing the successive calls to f_0 and f_1 based on the bit decomposition $\mathrm{bd}_0(a)$ of the input a without the leading zeroes. In Figure 1a, we give a tree induced by the computation of $\mathcal{I}_{\mathbb{G}}$ in a groupoid of order $s = 16$ with $id = 0$. Solid lines correspond to the application of f_0 and dotted lines to application of f_1. Except for the root labeled by the identity element id, each node of the tree corresponds to the point at which $\mathcal{I}_{\mathbb{G}}$ terminates on the corresponding input $a \in [s]$, where the second value in the label of the node is the input a and the first value is $\mathrm{bd}(a)$, i.e., the binary representation of a with the leading zeroes.

Note that Figure 1a actually suggests which functions f_0 and f_1 induce $\mathcal{I}_{\mathbb{G}}$ such that $\mathcal{I}_{\mathbb{G}}(a) = a$ for all $a \in [s]$. In particular, Algorithm 1 initializes the computation of $\mathcal{I}_{\mathbb{G}}$ with $r = \mathrm{bd}(id) = \mathrm{bd}(0) = 0^n$ and, thus, the desired traversal of the computation tree is achieved for all inputs $a \in [s]$ by 1) f_0 that performs a cyclic shift of the input r to the left and 2) f_1 that flips the last bit of the input r.

Similarly, the above observation allows to construct f'_0 and f'_1 such that for all $a \in [s]$ that $\mathcal{I}_{\mathbb{G}}(a) = a + b \mod s$ for some fixed $b \in [s]$, which can be performed simply by setting $id = b$ and consistently "shifting" the intermediate value r by the bit decomposition of the fixed value b before and after application of the above functions f_0 and f_1.

Incorporating the Pigeon instance

The issue which makes it non-trivial to reduce from PIGEON to INDEX is that the functions f_0 and f_1 inducing the groupoid $(\mathbb{G}, \star)$ are oblivious to the actual progress of the computation performed by $\mathcal{I}_{\mathbb{G}}$. The above discussion shows that we have some level of control over the computation of $\mathcal{I}_{\mathbb{G}}$. However, it is a priori unclear how to meaningfully incorporate the Pigeon instance C into the above construction achieving that $\mathcal{I}_{\mathbb{G}}(a) = a$ for all $a \in [s]$. For example, we cannot simply allow f_0 or f_1 to output $\mathsf{C}(r)$ while at some internal node in the computation tree of $\mathcal{I}_{\mathbb{G}}$ as this would completely break the global structure of $\mathcal{I}_{\mathbb{G}}$ on the node and all its children and, in particular, could induce collisions in $\mathcal{I}_{\mathbb{G}}$ unrelated to the collisions in C. However, we can postpone the application of C to the leaves of the tree since, for all inputs a corresponding to a leaf in the tree, the computation of $\mathcal{I}_{\mathbb{G}}(a)$ is not a part of the computation for $\mathcal{I}_{\mathbb{G}}(b)$ for another input b.

Given that we are restricted to the leaves of the computation tree when embedding the computation of C into $\mathcal{I}_{\mathbb{G}}$, we must work with a big enough tree in order to have as many leaves as the 2^n possible inputs of the circuit $\mathsf{C}\colon \{0,1\}^n \to \{0,1\}^n$. In other words, the instance of Index must correspond to a groupoid of order s strictly larger than n. Note that for $s = 2^k$, the leaves of the tree correspond exactly to the inputs for $\mathcal{I}_{\mathbb{G}}$ from the set

$$A_o = \{a \in [2^k] \mid \exists y \in \{0,1\}^{k-2}\colon \mathrm{bd}(a) = 1||y||1\},$$

i.e., the set of *odd* integers between 2^{k-1} and 2^k, which has size 2^{k-2}. Thus, in our construction, we set $s = 2^{n+2}$ to ensure that there are 2^n leaves that can represent the domain of C.

Our goal is to define $\mathcal{I}_{\mathbb{G}}$ so that its restriction to the internal nodes of the tree (non-leaves) is a bijection between $[2^{n+2}] \setminus A_o$ and $[2^{n+2}] \setminus [2^n]$. In other words, when evaluated on any internal node of the tree, $\mathcal{I}_{\mathbb{G}}$ avoids the values in $[2^n]$ corresponding to bit composition of the elements in the range of C. If we manage to induce such $\mathcal{I}_{\mathbb{G}}$ then there are no non-trivial collisions in $\mathcal{I}_{\mathbb{G}}$ involving the internal nodes – the restrictions of $\mathcal{I}_{\mathbb{G}}$ to A_o and to its complement $[2^{n+2}] \setminus A_o$ would have disjoint images and, by the bijective property of the restriction to the internal nodes of the tree, any collision in $\mathcal{I}_{\mathbb{G}}$ would be induced by a collision in C. Our construction achieves this goal by starting with f_0 and f_1 inducing $\mathcal{I}_{\mathbb{G}}$ such that, for all $a \in [2^{n+2}]$, it holds that $\mathcal{I}_{\mathbb{G}}(a) = a + 2^n \mod 2^{n+2}$, which we already explained above.

Note that the image of the restriction of $\mathcal{I}_{\mathbb{G}}$ to the set

$$A_e = \{a \in [2^{n+2}] \mid \exists y \in \{0,1\}^n\colon \mathrm{bd}(a) = 1||y||0\},$$

i.e., the set of *even* integers between 2^{n+1} and 2^{n+2}, has non-empty intersection with integers in $[2^n]$ corresponding to the range of C. Nevertheless, it is possible to locally alter the behaviour of f_0 andf_1 on A_e so that $\mathcal{I}_{\mathbb{G}}$ does not map to $[2^n]$ when evaluated on A_e. Then, we adjust the definition of f_0 and f_1 such that for all inputs $a \in A_o$ corresponding to a leaf of the tree, $\mathcal{I}_{\mathbb{G}}(a) = \mathrm{bc}(C(h(a)))$ for some bijection h between A_o and $\{0,1\}^n$ (a natural choice is simply the function that drops the first and the last bit from the binary decomposition $\mathrm{bd}(a)$ of a). Finally, we set the target in the resulting instance of Index to $t = 0$ to ensure that the preimage of t under $\mathcal{I}_{\mathbb{G}}$ corresponds exactly to a preimage of 0^n under C.

In Figure 1b, we illustrate the computation tree of $\mathcal{I}_{\mathbb{G}}$ corresponding to an instance of Index produced by our reduction on input $\mathsf{C}\colon \{0,1\}^n \to \{0,1\}^n$ for $n = 2$. Accordingly, $\mathbb{G}$ is of size $s = 2^{n+2} = 16$ and its elements are represented by the nodes of the tree. When compared with the tree in Figure 1a, the label of each node in Figure 1b equals the value $\mathcal{I}_{\mathbb{G}}(a)$, where a is the second value in the label of the node at the same position in the tree in Figure 1a. Nodes belonging to $[2^s] \setminus A_e \cup A_o, A_e$, and A_o are highlighted by differing styles of edges. Specifically, the labels of nodes with a solid edge correspond to evaluations of the inputs from $[2^{n+1}] = [8]$, the labels of nodes with a dashed edge correspond to evaluations of the inputs from A_e, and the labels of nodes with a dotted edge correspond to the evaluations of the inputs from A_o. Since the image of $\mathsf{bc} \circ \mathsf{C}$ is $[2^n] = [4]$, it is straightforward to verify that any collision in $\mathcal{I}_{\mathbb{G}}$ depicted in Figure 1b must correspond to a collision in C and that any preimage of $t = 0$ under $\mathcal{I}_{\mathbb{G}}$ corresponds directly to a preimage of 0^n under C.

The formal reduction establishing Lemma 16 is given in the full version [12].

▶ **Lemma 16.** *Pigeon is reducible to Index.*

4.2 Index Lies in PPP

The main idea of our reduction from Index to Pigeon is analogous to the reduction in [22] from their discrete logarithm problem in "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 $\mathbb{G}$. The formal reduction proving Lemma 17 is given in the full version [12]. Together, Lemma 16 and Lemma 17 establish Theorem 1.

▶ **Lemma 17.** *Index is reducible to Pigeon.*

5 New Characterizations of PWPP

Our results in Section 3.1 and Section 3.2 establish new PWPP-complete problems DLog, Dove, and Claw. Below, we provide additional discussion of these new PWPP-complete problems.

DLog

Alternative types of violations. Since the last type of a solution in DLog implies that the associative property does not hold for the elements $t, \mathcal{I}_{\mathbb{G}}(x)$, and $\mathcal{I}_{\mathbb{G}}(y)$, one could think about changing the last type of a solution to finding $x, y, z \in [s]$ such that $f_{\mathbb{G}}(x, f_{\mathbb{G}}(y, z)) \neq f_{\mathbb{G}}(f_{\mathbb{G}}(x, y), z)$ to capture violations of the associative property directly. However, our proof of PWPP-hardness would fail for such alternative version of DLog and we do not see an alternative way of reducing to it from the PWPP-complete problem Collision. In more detail, any reduction from Collision to DLog must somehow embed the instance C of Collision in the circuit f in the constructed instance of DLog. However, a refutation of the associative property of the form $f(x, f(y, z)) \neq f(f(x, y), z)$ for some x, y, and z might simply correspond to a trivial statement $C(u) \neq C(v)$ for some $u \neq v$, which is unrelated to any non-trivial collision in C.

Explicit $\mathcal{I}_{\mathbb{G}}$. A natural question about our definition of DLog is whether its computational complexity changes if the instance additionally contains an explicit circuit computing the indexing function $\mathcal{I}_{\mathbb{G}}$. First, the indexing function $\mathcal{I}_{\mathbb{G}}$ could then be independent of the group operation f and, thus, the reduction from Collision to such variant of DLog would become trivial by defining the indexing function $\mathcal{I}_{\mathbb{G}}$ directly via the Collision instance C. On the other hand, the core ideas of the reduction from DLog to Collision would remain mostly unchanged as it would have to capture also $\mathcal{I}_{\mathbb{G}}$ computed by Algorithm 1. Nevertheless, we believe that our version of DLog with an implicit $\mathcal{I}_{\mathbb{G}}$ computed by the standard square-and-multiply algorithm strikes the right balance in terms of modeling an interesting problem. The fact that it is more structured than the alternative with an explicit $\mathcal{I}_{\mathbb{G}}$ makes it significantly less artificial and relevant to the discrete logarithm problem, which is manifested especially in the non-trivial reduction from DLog to Collision in Section 3.2.

Dove

The chain of reductions in Section 3 shows, in particular, that Dove (Definition 6) is PWPP-complete. The most significant property of Dove compared to the known PWPP-complete problems (Pigeon or the weak constrained-SIS problem defined by Sotiraki et al. [22]) is that it is 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 other attempt at proving PWPP-hardness of other problems. We emphasize that all four types of a solution in Dove are exploited towards our reduction from Dove to DLog and we are not aware of a more direct approach of reducing Collision to DLog that avoids Dove as an intermediate problem.

Claw

Russel [21] 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 we can efficiently witness that one of the functions is not surjective.

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

▶ **Lemma 18.** *Collision is reducible to Claw.*

Proof of Lemma 18. 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 zeroes, which preserves the 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 instance C of Collision.
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 instance C of Collision. ◀

References

1 Nir Bitansky and Idan Gerichter. On the cryptographic hardness of local search. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, pages 6:1–6:29, 2020. doi:10.4230/LIPIcs.ITCS.2020.6.

2 Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a Nash equilibrium. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 1480–1498, 2015. doi:10.1109/FOCS.2015.94.

3 Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player Nash equilibria. *J. ACM*, 56(3), 2009. doi:10.1145/1516512.1516516.

4 Arka Rai Choudhuri, Pavel Hubáček, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. Finding a Nash equilibrium is no easier than breaking Fiat-Shamir. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 1103–1114. ACM, 2019. doi:10.1145/3313276.3316400.

5 Arka Rai Choudhuri, Pavel Hubáček, Chethan Kamath, Krzysztof Pietrzak, Alon Rosen, and Guy N. Rothblum. PPAD-hardness via iterated squaring modulo a composite. *IACR Cryptology ePrint Archive*, 2019:667, 2019. URL: https://eprint.iacr.org/2019/667.

6 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. doi:10.1007/3-540-39118-5_19.

7 Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. *SIAM J. Comput.*, 39(1):195–259, 2009. doi:10.1137/070699652.

8 Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In *Advances in Cryptology – EUROCRYPT 2020 – 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 125–154, 2020. doi:10.1007/978-3-030-45727-3_5.

9 Aris Filos-Ratsikas and Paul W. Goldberg. The complexity of splitting necklaces and bisecting ham sandwiches. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 638–649, 2019. doi:10.1145/3313276.3316334.

10 Sanjam Garg, Omkant Pandey, and Akshayaram Srinivasan. Revisiting the cryptographic hardness of finding a Nash equilibrium. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 579–604, 2016. doi:10.1007/978-3-662-53008-5_20.

11 Pavel Hubáček, Moni Naor, and Eylon Yogev. The journey from NP to TFNP hardness. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, pages 60:1–60:21, 2017. doi:10.4230/LIPIcs.ITCS.2017.60.

12 Pavel Hubáček and Jan Václavek. On search complexity of discrete logarithm. *CoRR*, abs/2107.02617, 2021. arXiv:2107.02617.

13 Pavel Hubáček and Eylon Yogev. Hardness of continuous local search: Query complexity and cryptographic lower bounds. *SIAM J. Comput.*, 49(6):1128–1172, 2020. doi:10.1137/17M1118014.

14 Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Sufficient conditions for collision-resistant hashing. In Joe Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 445–456. Springer, 2005. doi:10.1007/978-3-540-30576-7_24.

15 Ruta Jawale, Yael Tauman Kalai, Dakshita Khurana, and Rachel Zhang. Snargs for bounded depth computations and PPAD hardness from sub-exponential LWE. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 708–721. ACM, 2021. doi:10.1145/3406325.3451055.

16 Emil Jeřábek. Integer factoring and modular square roots. *J. Comput. Syst. Sci.*, 82(2):380–394, 2016. doi:10.1016/j.jcss.2015.08.001.

17 Antoine Joux, Andrew M. Odlyzko, and Cécile Pierrot. The past, evolving present, and future of the discrete logarithm. In Çetin Kaya Koç, editor, *Open Problems in Mathematics and Computational Science*, pages 5–36. Springer, 2014. doi:10.1007/978-3-319-10683-0_2.

18 Ilan Komargodski and Gil Segev. From Minicrypt to Obfustopia via private-key functional encryption. *J. Cryptol.*, 33(2):406–458, 2020. doi:10.1007/s00145-019-09327-x.

19 Alex Lombardi and Vinod Vaikuntanathan. Fiat-Shamir for repeated squaring with applications to PPAD-hardness and VDFs. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 632–651. Springer, 2020. doi:10.1007/978-3-030-56877-1_22.

20 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. doi:10.1016/S0022-0000(05)80063-7.

21 Alexander Russell. Necessary and sufficient condtions for collision-free hashing. *J. Cryptol.*, 8(2):87–100, 1995. doi:10.1007/BF00190757.

22 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. doi:10.1109/FOCS.2018.00023.

23 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. doi:10.1007/3-540-54195-0_66.