A Fully-Constructive Discrete-Logarithm Preprocessing Algorithm with an Optimal Time-Space Tradeoff

Lior Rotem*† Gil Segev*

Abstract

Identifying the concrete hardness of the discrete logarithm problem is crucial for instantiating a vast range of cryptographic schemes. Towards this goal, Corrigan-Gibbs and Kogan (EUROCRYPT '18) extended the generic-group model for capturing "preprocessing" algorithms, offering a tradeoff between the space S required for storing their preprocessing information, the time T required for their online phase, and their success probability. Corrigan-Gibbs and Kogan proved an upper bound of $\widetilde{O}(ST^2/N)$ on the success probability of any such algorithm, where N is the prime order of the group, matching the known preprocessing algorithms.

However, the known algorithms assume the availability of truly random hash functions, without taking into account the space required for storing them as part of the preprocessing information, and the time required for evaluating them in essentially each and every step of the online phase. This led Corrigan-Gibbs and Kogan to pose the open problem of designing a discrete-logarithm preprocessing algorithm that is fully constructive in the sense that it relies on explicit hash functions whose description lengths and evaluation times are taken into account in the algorithm's space-time tradeoff.

We present a fully constructive discrete-logarithm preprocessing algorithm with an asymptotically optimal space-time tradeoff (i.e., with success probability $\widetilde{\Omega}(ST^2/N)$). In addition, we obtain an algorithm that settles the corresponding tradeoff for the computational Diffie-Hellman problem. Our approach is based on derandomization techniques that provide rather weak independence guarantees. On the one hand, we show that such guarantees can be realized in our setting with only a minor efficiency overhead. On the other hand, exploiting such weak guarantees requires a more subtle and in-depth analysis of the underlying combinatorial structure compared to that of the known preprocessing algorithms and their analyses.

*School of Computer Science and Engineering, Hebrew University of Jerusalem, Jerusalem 91904, Israel. Email: {lior.rotem,segev}@cs.huji.ac.il. Supported by the European Union's Horizon 2020 Framework Program (H2020) via an ERC Grant (Grant No. 714253).

†Supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

1 Introduction

Identifying the concrete hardness of the discrete logarithm problem in prime-order groups is crucial for instantiating a vast range of cryptographic schemes. Shoup's seminal work [Sho97] introduced the generic-group model, capturing all computations that do not exploit any specific property of the representation of the underlying group, and provided a tight bound on the hardness of the discrete logarithm problem in this model. Specifically, Shoup proved that any generic-group algorithm that runs in time at most T (thus, in particular, performs at most T group operations), computes the discrete logarithm of a uniformly-distributed group element with probability $O(T^2/N)$, where N is the prime order of the group.[1]

Although generic-group algorithms seem somewhat restricted, the generic hardness of the discrete logarithm problem may nevertheless be used for setting concrete security parameters in any group in which non-generic discrete logarithm algorithms do not seem to outperform the known generic ones (most notably, in popular elliptic-curve groups [KMV00, FST10]). However, as recently observed by Corrigan-Gibbs and Kogan [CK18], the bound established by Shoup and Maurer does not apply to "preprocessing" algorithms, as introduced by Hellman in the context of the function inversion problem [Hel80, FN99, DTT10]. For the discrete logarithm problem, a preprocessing algorithm may first preprocess the group in an offline phase. Then, in an online phase, the algorithm receives a group element $h \in \mathbb{G}$, and may use the preprocessing information to compute its discrete logarithm. The efficiency of such algorithms is measured via the tradeoff between the space S required for storing their preprocessing information, the time T required for their online phase, and their success probability.

Motivated by elegant preprocessing algorithms for the discrete logarithm problem due to Lee, Cheon, and Hong [LCH11] and by Bernstein and Lange [BL13], Corrigan-Gibbs and Kogan extended the generic-group model to capture preprocessing algorithms, and proved an upper bound on the success probability of any such algorithm in computing discrete logarithms. Specifically, for essentially any S and T, they proved that any preprocessing algorithm computes the discrete logarithm of a uniformly-distributed group element with probability $\widetilde{O}(ST^2/N)$. Alternatively, denoting by ϵ the success probability of such algorithms, they proved the lower bound $ST^2 = \widetilde{\Omega}(\epsilon N)$ on the required space and time resources.

The tradeoff established by Corrigan-Gibbs and Kogan matches the performance provided by the algorithms of Lee et al. and of Bernstein and Lange. However, these algorithms assume the availability of truly random hash functions, without taking into account the space required for storing them as part of the preprocessing information, and the time required for evaluating them in essentially each and every step of the online phase.[2] A standard approach in the design and analysis of algorithms for eliminating this assumption is to rely on derandomization techniques based on k-wise independent hash functions, guaranteeing that their outputs are independently and uniformly distributed when restricted to any set of most k inputs. Unfortunately, as we discuss in Section 1.3, for the level k of independence that seems required for the probabilistic analysis of the above two algorithms, explicit constructions of k-wise independent hash functions inherently result in a significant increase either in the space required for storing them as part of the preprocessing

[1] An alternative generic-group model was later introduced by Maurer [Mau05], admitting the same tight bound on the hardness of the discrete logarithm problem.

[2] The lower bound of Corrigan-Gibbs and Kogan allows the offline and online phases to share an arbitrary-long common random string which is not accounted for in the space required for storing the preprocessing information. Thus, on the one hand, their lower bound applies even to algorithms that assume the availability of truly random hash functions. On the other hand, however, when taking the required storage into account, a comparable yet more direct solution is to just store the discrete logarithms of all group elements.

information or in the time required for evaluating them in the online phase [Sie04].

This state of affairs has led Corrigan-Gibbs and Kogan to pose the open problem of designing a preprocessing algorithm for computing discrete logarithms that is fully constructive in the sense that it relies on explicit hash functions whose description lengths and evaluation times are taken into account in the algorithm's space-time tradeoff. That is:

Is there an explicit (i.e., fully constructive) *preprocessing algorithm for computing discrete logarithms that matches the* $ST^2 = \widetilde{\Theta}(\epsilon N)$ *tradeoff?*

This question is in fact relevant not only to the discrete logarithm problem, but also to various other problems in prime-order groups for which the known preprocessing algorithms assume the availability of truly random functions without taking into account the space required for storing them and the time required for evaluating them. These include, in particular, the computational Diffie-Hellman problem, for which Corrigan-Gibbs and Kogan proved a similar $ST^2 = \widetilde{\Omega}(\epsilon N)$ lower bound.

1.1 Existing Approaches

The seminal work of Fiat and Naor [FN99] considered the function-inversion variant of this problem, given that Hellman's preprocessing function inversion algorithm [Hel80] assumed the availability of truly random hash functions in a similar manner. Fiat and Naor presented an explicit algorithm that relies on concrete hash functions, and were able to match the tradeoff established by Hellman. The algorithm of Fiat and Naor can be seen as an explicit preprocessing algorithm for computing discrete logarithms. However, the bound that it achieves when applied to the discrete logarithm problem is $S^2T = \Theta(\epsilon N^2)$, far from matching the $ST^2 = \widetilde{\Theta}(\epsilon N)$ tradeoff. It is worth stressing that this gap stems from the fact that the setting considered by Fiat and Naor is much more general: Their algorithm can invert *any* function, and thus cannot exploit the algebraic structure of the underlying group in the discrete logarithm problem. Nevertheless, one could hope that relying on the same ideas of Fiat and Naor, one could make the algorithms of Lee, Cheon, and Hong [LCH11] and of Bernstein and Lange [BL13] explicit. Unfortunately, as we discuss in Section 1.3, the standard amortization technique that enabled Fiat and Naor to rely on concrete hash functions while still matching Hellman's tradeoff does not seem applicable for the known preprocessing algorithms for the discrete logarithm problem.

The recent work of Maurer, Portman, and Zhu [MPZ20] considered the task of replacing the random function assumed by Lee, Cheon, and Hong [LCH11] and of Bernstein and Lange [BL13] by an explicit k-wise independent hash function for a suitable choice of k. However, the focus of their work is different, as they consider idealized models which only account for the number of oracle queries that the online algorithm issues. As a result, their analysis does not take into account the *time* required for evaluating the k-wise independent hash function, and they only consider the space required for representing it. As we discuss in Section 1.3, instantiating their approach in the standard model results in a space-time tradeoff that is far from optimal.

Finally, it should be noted that preprocessing algorithms that assume the availability of truly random hash functions can be viewed as algorithms within the random-oracle model [BR93].[3] When instantiating the random oracle with cryptographic hash functions, most applications rely on the standard assumption that such functions are "sufficiently random" with respect to a polynomial (or, say, moderately super-polynomial) number of queries. However, the known preprocessing algorithms

[3]See also [Unr07, DGK17, CDG18], and the references therein, for the related line of work on bounding the usefulness of auxiliary inputs in the random-oracle model.

issue a nearly exponential number of queries (e.g., $N^{1/3}$ queries), and this holds even when considering only the queries issued in their online phase. From the theoretical perspective, this requires a substantially stronger assumption regarding the heuristic security of cryptographic hash functions. Moreover, from the more practical perspective of setting concrete security parameters against preprocessing attacks, this unnecessarily ties the assumed concrete security of discrete logarithm problem (and of additional related problems) to that of cryptographic hash functions.

1.2 Our Contributions

In this work we resolve the above-stated question by presenting an explicit (i.e., fully constructive) discrete logarithm preprocessing algorithm that is asymptotically optimal in terms of its space-time tradeoff. In fact, our algorithm does not explicitly settle the space-time tradeoff only for the discrete logarithm problem, but also yields an explicit algorithm that settles the corresponding tradeoff for the computational Diffie-Hellman problem (CDH).

Within the unit-cost RAM model, which is the standard model for analyzing the efficiency of explicit data structures and algorithms (see Section 2), we prove the following theorem:

Theorem 1.1 (informal)**.** *For any integers S and T such that $S-T=\Omega(S)$, there exists an explicit algorithm $A=(A_0,A_1)$ such that for any cyclic group $(\mathbb{G},N,g)$ it holds that*

$$\Pr_{x\leftarrow\mathbb{Z}_N}\left[A_1\left(A_0(\mathbb{G},N,g),g^x\right)=x\right]=\widetilde{\Omega}\left(\frac{S\cdot T^2}{N}\right),$$

where the offline algorithm A_0 outputs S bits of preprocessing information, and the online algorithm A_1 runs in time T.

Note that our algorithm A consists of a pair (A_0,A_1) of algorithms: The offline algorithm A_0 takes as input the description $(\mathbb{G},N,g)$ of a cyclic group of order N that is generated by $g\in\mathbb{G}$ and produces the preprocessing information, and the online algorithm A_1 takes as input a uniformly-distributed group element $g^x\in\mathbb{G}$ (together with the preprocessing information) and tries to compute its discrete logarithm $x\in\mathbb{Z}_N$ with respect to the given generator g. Similarly to the previously-known algorithms [LCH11, BL13], our algorithm does not rely on any specific property of the representation of the underlying group, and can be formally presented within Shoup's generic-group model [Sho97].

In addition, note that we require the parameters S and T to satisfy $S-T=\Omega(S)$ (i.e., we require that $(1-\alpha)S\geq T$ for some constant $\alpha>0$), and this results from the additional space overhead we incur for explicitly storing descriptions of hash functions.[4] This is a somewhat natural restriction given the nature of preprocessing algorithms (relying on preprocessing information in order to reduce the online running time), which captures in particular the choice of $S=\Theta(N^{1/3})$ and $T=\Theta(N^{1/3})$ that balances the space and time resources of the algorithm, as well as any other choice of $S=\Theta(N^{1-2\beta})$ and $T=\Theta(N^{\beta})$ for $\beta\geq 1/3$.

Finally, note that any preprocessing algorithm for the discrete logarithm problem directly yields a preprocessing algorithm for the computational Diffie-Hellman (CDH) problem with the same space-time tradeoff. Thus, given that Corrigan-Gibbs and Kogan [CK18] additionally proved the lower bound $ST^2=\widetilde{\Omega}(\epsilon N)$ for the CDH problem with preprocessing, the following corollary of Theorem 1.1 provides an explicit preprocessing algorithm that asymptotically matches the optimal tradeoff for the CDH problem as well:

[4] An interesting technical question is whether the requirement $S-T=\Omega(S)$ can be avoided while still matching the $ST^2=\widetilde{\Theta}(\epsilon N)$ tradeoff with an explicit algorithm.

Corollary 1.2 (informal)**.** *For any integers S and T such that $S-T=\Omega(S)$, there exists an explicit algorithm $A=(A_0,A_1)$ such that for any cyclic group $(\mathbb{G},N,g)$ it holds that*

$$\Pr_{x,y\leftarrow\mathbb{Z}_N}\left[A_1\left(A_0(\mathbb{G},N,g),g^x,g^y\right)=g^{xy}\right]=\widetilde{\Omega}\left(\frac{S\cdot T^2}{N}\right),$$

where the offline algorithm A_0 outputs S bits of preprocessing information, and the online algorithm A_1 runs in time T.

1.3 Overview of Our Approach

Our starting point: Preprocessing with a truly random function. The starting point for our explicit preprocessing algorithms is the approach which underlies the preprocessing algorithms of Lee, Cheon, and Hong [LCH11] and Bernstein and Lange [BL13]. This approach relies on the existence of a truly random hash function $f:\mathbb{G}\to\mathbb{Z}_N$, shared between the preprocessing algorithm A_0 and the online algorithm A_1. This function defines a random walk on the elements of $\mathbb{G}$ via the step function $h\to h\cdot g^{f(h)}$.

The preprocessing algorithm A_0 preforms S such random walks, starting from S uniformly-random group elements $g^{\alpha_1},\ldots,g^{\alpha_S}$, and taking T steps in each walk. It then stored the end point h_i of each walk, together with its discrete logarithm β_i with respect to g. Note that A_0 can indeed compute β_i, since $\beta_i=\alpha_i+\sum_{j=1}^{T}f^{(j)}(g^{\alpha_i})$, where $f^{(1)}(\cdot)=f(\cdot)$ and $f^{(j)}(\cdot)=f(f^{(j-1)}(\cdot))$ for $j\geq 2$. The endpoints $h_1,\ldots,h_S$ and their respective discrete logarithms $\beta_1,\ldots,\beta_S$ are passed as the state to the online algorithm A_1.

The online algorithm A_1 receives as input the above state and a challenge group element $h=g^x$, and its goal is to compute x. To this end, it performs a random walk of length at most $2T$, starting from the challenge h. The hope is that eventually, this walk will "hit" one of the stored endpoints $h_1,\ldots,h_S$. Say that the online walk hits h_i after ℓ steps. In this case, we know that $g^{x+\sum_{j=1}^{\ell}f^{(j)}(h)}=h_i=g^{\beta_i}$. Since g is a generator of the group, this implies that $x=\beta_i-\sum_{j=1}^{\ell}f^{(j)}(h)$. Since β_i and h are both known to A_1, it can compute and output x.

The description length of the state passed from A_0 to A_1 is roughly S (assuming that the function f does not need to be stored as part of the state). The computation executed by A_1 involves roughly $2T$ invocations of H, and T exponentiations in the group, resulting in a running time of roughly T (when ignoring the time required for evaluating f).[5]

We now sketch the analysis for bounding the success probability of these algorithms (see also [CK18] and the references therein). First, observe that if the online random walk collides with at least one of the precomputed paths within the first T steps, then it will inevitably hit a precomputed endpoint and A_1 will successfully output the discrete logarithm of the challenge element h. Hence, it is sufficient to bound the probability that such a collision occurs. This bound follows from two simple probabilistic arguments. The first argument shows that with constant probability, the expected number of distinct group elements "touched" by the precomputed paths is at least $\Omega(ST)$. The second argument shows that when the precomputed paths touch at least $\Omega(ST)$ group elements, the collision probability that we wish to bound is at least $\Omega(ST^2/N)$, since each new group element in the online walk hits any of the elements touched by the precomputed elements with probability $\Omega(ST/N)$. Both of these probabilities are taken also over the choice of the truly random function f, and both arguments inherently rely on the assumption that f is sampled uniformly at random from the set of all functions from the group $\mathbb{G}$ to $\mathbb{Z}_N$.

[5] Both the state size and the running time ignore $\log N$ factors.

Accounting for the description length of the hash function. As previously mentioned, the above attack attains the optimal tradeoff between the preprocessed state size, the online running time and the success probability only when we do not take into count the description length of the truly random hash function f as part of the state size. When we do account for the description length of the function f, the state size blows up to more than $N \cdot \log N$ bits. This clearly renders the entire approach useless, since with this many bits, the preprocessing algorithm can simply pass to the online algorithm the discrete logarithm of every group element.

A standard way of reducing the representation size of such functions while still enjoying certain independence guarantees, is by sampling them from a family of k-wise independent hash functions for a suitable choice of k. Observe that f is applied by A_0 and A_1 to $O(ST)$ group elements. Hence, instead of using a truly uniform function, one can use a hash function sampled from a family of $O(ST)$-wise independent functions while keeping the analysis intact and without damaging the success probability. Alas, this is still far from satisfactory, since the space for storing a function from such families and its evaluation time will yield a tradeoff which is far from optimal. For example, instantiating such functions via randomly-sampled polynomials of degree $O(ST)$ will result in a state of size $S' = O(ST)$ and online running time of $T' = O(ST^2)$. In other words, the success probability is only $\Omega(T'/N)$, smaller by a factor of roughly $S'T'$ from the lower bound of Corrigan-Gibbs and Kogan. This non-optimality seems to be inherent when instantiating f using full-fledged k-wise independence: A lower bound by Siegel [Sie04] shows that any construction reducing the evaluation time of k-wise independent functions below $\Omega(k)$ entails a polynomial increase in the space required for representing each function, thus once again leading to a non-optimal tradeoff.

Relying on $O(T)$-wise independence via a local analysis. As a first step, we present a more nuanced analysis for the success probability of the above algorithms, which enables us to reduce the level of independence required for the family of hash functions from $O(ST)$ to just $O(T)$. Note that indeed, due to its global nature, the analysis presented above for a truly random hash function breaks down when replacing it with a function sampled from an $O(T)$-wise independent family. In particular, we can no longer argue that the random walks in the preprocessing stage cover $\Omega(ST)$ distinct group element with high enough probability. The key observation is that such a global argument is unnecessary. What we are really interested in is the probability that the online walk collides with *one* of the walks from the preprocessing stage. Indeed, this probability can be sufficiently large even if the fraction of group elements covered by the preprocessed walks is very small. Intuitively, this is even likely: The more skewed the distribution over endpoints of T-step random walks is, the greater the probability is for a collision.[6]

Our refined analysis relies on a local argument that only considers the application of the hash function f on $O(T)$ group elements at a time. Therefore, it holds even when f is sampled from a family of $O(T)$-wise independent functions. First, we prove that the probability that the online walk collides with any specific precomputed walk within T steps is at least $\Omega(T^2/N)$. Then, we prove that the probability that it collides with any *two* specific precomputed walks within T steps is at most $O(T^4/N^2)$. Since these arguments only consider at most $2T$ and $3T$ distinct group elements, respectively, we are able to prove them relying on H being sampled from a $3T$-wise independent family. Finally, we use the inclusion-exclusion principle to bound the probability that the online walk hits a precomputed walk with T steps by $\Omega(ST^2/N)$.

[6] As an extreme example, consider a function f^* that maps every group element $h \in \mathbb{G}$ to an integer $\alpha \in \mathbb{Z}_N$ such that $h \cdot g^\alpha = g$. If this function is used, the preprocessed walks cover at most $S+1$ group elements, but the online walk collides with all of them with probability 1. Of course, the function f^* essentially computes the discrete logarithm of every group element without the random-walks-based algorithms. We use it here merely to exemplify the above point.

The remaining gap: The evaluation time of k-wise independent functions. Sampling the hash function f from a family of $O(T)$-wise independent functions still does not suffice to match the lower bound of Corrigan-Gibbs and Kogan. Consider again using a randomly-sampled polynomial of degree $O(T)$. In this case, the size of the state passed from A_0 to A_1 is indeed reduced to $S' = O(S + T)$, simplifying to $S' = O(S)$ in the natural case in which $S - T = \Omega(S)$. However, the running time of A_1 is much greater than T. Each evaluation of a degree $O(T)$ polynomial takes time at least $\Omega(T)$, and at worst, A_1 makes $2T$ such evaluations. This results in a running time of at least $T' = \Omega(T^2)$. In other words, the success probability is only $\Omega(S'T'/N)$, a factor of roughly T' away from the lower bound of Corrigan-Gibbs and Kogan. The lower bound of Siegel [Sie04] again suggests that other instantiations of $O(T)$-wise independent families will also result in far-from-optimal tradeoffs.

The unsuitability of Fiat-Naor's derandomization. The work of Fiat and Naor [FN99] undertakes a similar endeavor to ours, presenting explicit preprocessing algorithms for the *function inversion problem*. They too start from (a modification of) previously-known algorithms that assume the existence of truly random functions shared between the preprocessing and the online algorithms [Hel80] and encounter a similar problem to the one described above: Trying to instantiate the random functions naively by choosing them independently from a k-wise independent family (for a suitable value of k) results in a sub-optimal running time. Their solution to this problem is to choose these functions in a pairwise independent manner, instead of choosing them completely independently. Concretely, each function in their construction is a random polynomial of degree $k - 1$, but the coefficients of the different polynomials are sampled using pairwise independent functions. Fiat and Naor show that this change does not hurt the success probability of the attack too much, while at the same time, it enables a valuable speed-up in the online running time by evaluating these polynomials concurrently using the Fast Fourier Transform. Such an approach does not seem to fit our setting, where we are need to derandomize only a *single* truly random function. Moreover, attempts to modify the algorithms of Lee, Cheon, and Hong [LCH11] and Bernstein and Lange [BL13] to use several different random functions (to instantiate them in a correlated manner like Fiat and Naor did) seem to yield sub-optimal tradeoffs.

Our Solution: Reducing running time via weaker independence. To reduce the overhead in the online running time caused by the T hash evaluations, we prove that the $O(T)$-wise independent family can be replaced with a function family that offers weaker independence guarantees. Concretely, we sample our hash function f from function families put forth by Pagh and Pagh [PP08] (following Siegel [Sie04]). Functions within these families can be evaluated in constant time in the standard unit-cost RAM model (as described in Section 2), effectively eliminating the overhead in running time relative to using full-fledged $O(T)$-wise independent families. The process of sampling a function from these families can be thought of as occurring in two steps: First, a function family $\mathcal{F}$, parameterized by a parameter $k \in \mathbb{N}$, is drawn from a collection of families. Then, a function f is sampled from $\mathcal{F}$. Simplifying, the independence guarantee is that for any *specific* set $\mathcal{S}$ of k elements in the domain, $\mathcal{F}$ is "fully" independent with respect to $\mathcal{S}$ with high probability. This differs from the standard notion of k-wise independence, which requires a randomly chosen function to satisfy this property for *any* set $\mathcal{S}$ of size k.

We would like to argue that the above analysis, for a function f sampled from an $O(T)$-wise independent family, carries over to the case where f is sampled from a family with this weaker independence guarantee (where the parameter k is set to be $O(T)$). The problem, though, is that according to the original analysis of Pagh and Pagh, the family $\mathcal{F}$ is guaranteed to be k-wise inde-

pendent with high probability only with respect to subsets which are fixed before $\mathcal{F}$ is chosen. It immediately follows that $\mathcal{F}$ is k-wise independent with high probability also with respect to subsets which are sampled from a distribution which is independent of the choice of $\mathcal{F}$. In our case, however, the analysis of the discrete log algorithms requires $\mathcal{F}$ to be fully independent with respect to random subsets that do depend on the choice $\mathcal{F}$. Concretely, we want the function f, sampled from $\mathcal{F}$, to behave like a random function on the union of two or three T-step random walks induced by f. Fortunately, a lemma proved by Berman, Haitner, Komargodski, and Naor [BHK+19] in a different context implies that $\mathcal{F}$ remains essentially fully independent with sufficiently high probability on such adaptively-chosen subsets as well. Overcoming various additional technical difficulties, this enables us to rely on explicit hash functions whose description lengths and evaluation times are taken into account in the algorithm's asymptotically optimal space-time tradeoff.

2 Preliminaries

In this section we present the basic notions and standard cryptographic primitives that are used in this work. For an integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, \ldots, n\}$. For a distribution X we denote by $x \leftarrow X$ the process of sampling a value x from the distribution X. Similarly, for a set $\mathcal{X}$ we denote by $x \leftarrow \mathcal{X}$ the process of sampling a value x from the uniform distribution over $\mathcal{X}$.

The computational model. We consider the unit-cost RAM model which has been the subject of much research, and is the standard model for analyzing the efficiency of explicit data structures and algorithms in terms of the running time of their operations (see, for example, [Mil99, HMP01, DP08, PP08] and the references therein). In this model, any operation in a rather minimal instruction set can be executed in constant time on w-bit operands, where $w = O(\log u)$, and all elements are taken from a universe of size u. In our case, u may be any polynomial in the order N of the underlying group $\mathbb{G}$ in which we wish to compute discrete logarithms, and thus $w = O(\log N)$. We consider the standard instruction set for the unit-cost RAM model, which includes integer addition, subtraction, bit-wise Boolean operations, left and right bit shifts, and integer multiplication (we emphasize that the integers considered in the analysis of our algorithms will all be in the range $0, \ldots, N-1$).

Additionally, for an underlying cyclic group $\mathbb{G}$ of order p we denote by t_{mult} and t_{exp} the running times of computing the group operation and the group exponentiation, respectively. Within the unit-cost RAM model we then state the running time of our algorithms as functions of t_{mult} and t_{exp}. Assuming that both operations can be implemented in time polynomial in $\log N$, this translates into (at most) a multiplicative lower-order factor of $\mathsf{poly}(\log N)$.

Uniform hashing. A function family $\mathcal{H}$ is said to be uniform over a set $\mathcal{S}$ of elements in its domain, if a uniformly-sampled function $h \leftarrow \mathcal{H}$ is indistinguishable from a truly random function when evaluated on $\mathcal{S}$. This is formally captured via the following definition:

Definition 2.1. Let $\mathcal{X}$ and $\mathcal{Y}$ be sets and let $\mathcal{S} = \{x_1, \ldots, x_k\} \subseteq \mathcal{X}$ be a subset of size k. We say that a function family $\mathcal{H}$ mapping $\mathcal{X}$ to $\mathcal{Y}$ is *uniform over* $\mathcal{S}$ if for every tuple $(y_1, \ldots, y_k) \in \mathcal{Y}^k$ it holds that

$$\Pr_{h \leftarrow \mathcal{H}}[\forall i \in [k] \ : \ h(x_i) = y_i] = \frac{1}{|\mathcal{Y}|^k}.$$

We say that $\mathcal{H}$ is *k-wise independent* if it is uniform over all subsets of $\mathcal{X}$ of size at most k.

Functions families which are k-wise independent have repeatedly proven to be useful for the design and analysis of data structures in general, and for cryptographic preprocessing attacks in

particular [FN99]. Alas, all known constructions of such families provide functions which take time at least k to evaluate; this will, unfortunately, prove to be too costly for our attack. However, Pagh and Pagh [PP08], following Siegel [Sie04], constructed families of functions which can be evaluated in constant time, offering weaker guarantees than full-fledged k-wise independence, but such that still suffice for our needs. Concretely, they consider a randomized algorithm which generates a random family $\mathcal{H}$ of functions, such that for any predetermined set $\mathcal{S}$ of at most k elements in the domain, the family $\mathcal{H}$ is uniform over $\mathcal{S}$ with high probability.

Theorem 2.2 ([PP08] – simplified)**.** *Let $\mathcal{X}$ and $\mathcal{Y}$ be sets. Then, there exists an algorithm* HashGen *that on input any integer $k \in \mathbb{N}$ and any constant $c > 0$, outputs a description of a function family $\mathcal{H}$ mapping $\mathcal{X}$ to $\mathcal{Y}$ such the following hold:*

1. *For every set $\mathcal{S} \subseteq \mathcal{X}$ of size at most k it holds that*

$$\Pr_{\mathcal{H} \leftarrow \mathsf{HashGen}(k,c)} [\mathcal{H} \text{ is uniform over } \mathcal{S}] \geq 1 - \frac{1}{k^c}.$$

2. *Every function in $\mathcal{H}$ can be represented using at most $2k \cdot \log |\mathcal{Y}| + O(k + \log\log |\mathcal{X}|)$ bits, and evaluated on any input in constant time within the unit-cost RAM model.*

We note that the construction of Pagh and Pagh was later improved in various ways (see, for example, [DW03, DR09, ADW14]), but the parameters it offers already suffice for our needs. In addition, note that Theorem 2.2 guarantees only that $\mathcal{H}$ sampled by HashGen is uniform with high probability over sets of elements which are a-priori fixed, and do not depend on $\mathcal{H}$. Looking ahead, we will want to reason about the output distribution of $\mathcal{H}$ on sets of elements that *do depend* on $\mathcal{H}$. To this end, we will rely on a lemma by Berman et al. [BHK+19] who proved that $\mathcal{H}$ sampled by HashGen is uniform with high probability also on sets of elements which are chosen by an unbounded adversary which queries a random function in $\mathcal{H}$ at most k times.

Lemma 2.3 ([BHK+19] – simplified)**.** *Let $\mathcal{X}$ and $\mathcal{Y}$ be sets, let k be an integer, let $\mathcal{F}_k$ be a k-wise independent family of functions $f : \mathcal{X} \to \mathcal{Y}$, and let* HashGen *be the algorithm guaranteed by Theorem 2.2 producing families of functions $f : \mathcal{X} \to \mathcal{Y}$. Then, for any k-query algorithm D and constant $c > 0$ it holds that*

$$\left| \Pr_{\substack{\mathcal{H} \leftarrow \mathsf{HashGen}(k,c) \\ f \leftarrow \mathcal{H}}} \left[D^f() = 1 \right] - \Pr_{f \leftarrow \mathcal{F}_k} \left[D^f() = 1 \right] \right| \leq O\left(\frac{1}{k^c} \right).$$

3 Our Discrete-Logarithm Preprocessing Algorithm

In this section we present our preprocessing algorithm for computing discrete logarithms in a cyclic group $\mathbb{G}$ of order N relative to a generator $g \in \mathbb{G}$. For simplicity, throughout the section we fix the group $\mathbb{G}$, the order N and the generator g, and note that these can in fact be provided as inputs to our algorithm. Our algorithm A consists of a pair (A_0, A_1) of algorithms, where A_0 and A_1 are the preprocessing algorithm and the on-line algorithm, respectively. Additionally, our algorithm is parameterized by integers $\ell, s \in \mathbb{N}$ and by a constant $c > 0$, and uses as a building block the algorithm HashGen described in Section 2 for producing families of hash functions $f : \mathbb{G} \to \mathbb{Z}_N$.

The preprocessing algorithm A_0

Input: A description $(\mathbb{G}, N, g)$ of a cyclic group $\mathbb{G}$ of order N that is generated by $g \in \mathbb{G}$, integers ℓ and s, and a constant $c > 0$.

1. Sample $\mathcal{H} \leftarrow \mathsf{HashGen}(3\ell, c)$ and $f \leftarrow \mathcal{H}$.
2. For each $i = 1, \ldots, s$:
   (a) Sample $x_{i,1} \leftarrow \mathbb{Z}_N$ and compute $g_{i,1} = g^{x_{i,1}}$.
   (b) For each $j = 2, \ldots, \ell$ compute $x_{i,j} = x_{i,j-1} + f(g_{i,j-1})$ and $g_{i,j} = g^{x_{i,j}}$.
   (c) Set $y_i = x_{i,\ell}$ and $g_i = g_{i,\ell}$.
3. Output $\mathsf{st} = \left(f, \{(g_i, y_i)\}_{i \in [s]}\right)$.

The online algorithm A_1

Input: A description $(\mathbb{G}, N, g)$ of a cyclic group, a group element $h \in \mathbb{G}$, and a state $\mathsf{st} = \left(f, \{(g_i, y_i)\}_{i \in [s]}\right)$ produced by A_0.

1. If $h = g_i$ for some $i \in [s]$, then output y_i and terminate.
2. Set $h_1 = h$ and $\Delta_1 = 0$, and for each $i = 2, \ldots, 2\ell$:
   (a) Compute $\delta_i = f(h_{i-1})$ and $\Delta_i = \Delta_{i-1} + \delta_i$.
   (b) Compute $h_i = h_{i-1} \cdot g^{\delta_i}$.
   (c) If $h_i = g_j$ for some $j \in [s]$, then output $x = y_j - \Delta_i$ and terminate.
3. Output $\perp$.

Note that the description of the online algorithm A_1 includes two non-trivial lookup operations in Steps 1 and 2c for the elements h and h_i, respectively. For avoiding a noticeable overhead in the running time of A_1, these lookup operations can be implemented by having the preprocessing algorithm A_0 store the pairs $\{(g_i, y_i)\}_{i \in [s]}$ within an explicit data structure that supports efficient lookup operations and uses linear space (i.e., $O(s)$ space). In the unit-cost RAM model, existing such data structures range, for example, from the most basic solution of a sorted list that supports lookup operations in time $O(\log s)$, to more advanced solutions such as cuckoo hashing that supports lookup operations in constant time [PR04].

The following theorem states our bounds on the amount of space required for storing the state produced by the preprocessing algorithm A_0, on the running time of the online algorithm A_1, and on the success probability of A_1 in computing the discrete logarithm $\mathsf{dlog}_g(h)$ of a uniformly-distributed group element h.

Theorem 3.1. *Let $\mathbb{G}$ be a cyclic group of order N that is generated by $g \in \mathbb{G}$. Let s and ℓ be any integers such that $s \cdot \ell^2 \leq N/64$, and let $c > 0$ be any constant. Then,*

$$\Pr\left[A_1\left(\mathbb{G}, N, g, h, \mathsf{st}\right) = \mathsf{dlog}_g(h)\right] \geq \frac{1}{8} \cdot \frac{s \cdot \ell^2}{N} - O\left(\frac{1}{(3\ell)^c}\right),$$

where $\mathsf{st} \leftarrow A_0(\mathbb{G}, p, g)$ and $h \leftarrow \mathbb{G}$. In addition, A_0 outputs $O((\ell + s) \cdot \log N)$ bits, and A_1 runs in time $O(\ell \cdot t_{\mathsf{exp}})$ in the unit-cost RAM model, where t_{exp} denotes the running time of exponentiation in the group $\mathbb{G}$.

Assuming that exponentiation in the group $\mathbb{G}$ can be implemented in time $t_{\mathsf{exp}} = \mathsf{poly}(\log N)$, the following corollary captures the specific setting of $s = \ell = O(N^{1/3})$:

Corollary 3.2. *When setting $s = \ell = O(N^{1/3})$, the preprocessing algorithm A_0 outputs $S = \widetilde{O}(N^{1/3})$ bits, and the online algorithm A_1 runs in time $T = \widetilde{O}(N^{1/3})$ in the unit-cost RAM model and succeeds with a constant probability.*

Proof of Theorem 3.1. First, note that the state $\mathsf{st} = \left(f, \{(g_i, y_i)\}_{i\in[s]}\right)$ produced by A_0 consists of the description of a hash function f sampled from the family $\mathcal{H}$ produced by $\mathsf{HashGen}(3\ell, c)$, and of s pairs (g_i, y_i) where $g_i \in \mathbb{G}$ and $y_i \in \mathbb{Z}_N$ for each $i \in [s]$. By Theorem 2.2 the description of f is of length at most $2 \cdot 3\ell \cdot O(\log N) + O(\ell + \log\log N)$ bits, and additionally each pair (g_i, y_i) can be represented using $O(\log N)$ bits. Therefore, A_0 outputs $O((\ell + s) \cdot \log N)$ bits.

Second, note that A_1's running time is dominated by that of Step 2, which is repeated for at most 2ℓ iterations. Each such iteration consists of an evaluation of the hash function f (which by Theorem 2.2 takes constant time in the unit-cost RAM model), a group multiplication, a group exponentiation, and an additional constant number of constant-time operations. Therefore, overall A_1 runs in time $O(\ell \cdot t_{\mathsf{exp}})$.

In the remainder of this proof, we analyze the success probability of our algorithm. For any function $f : \mathbb{G} \to \mathbb{Z}_N$, define the function $\widehat{f} : \mathbb{G} \to \mathbb{G}$ by $\widehat{f}(h) = h \cdot g^{f(h)}$.

Claim 3.3. *Let $\mathcal{H}$ be a family of functions $f : \mathbb{G} \to \mathbb{Z}_N$ and let $\mathcal{G}_{\mathcal{H}} = \left\{\widehat{f}\right\}_{f\in\mathcal{H}}$. Then, for any integer $k \in \mathbb{N}$, if $\mathcal{H}$ is k-wise independent then $\mathcal{G}_{\mathcal{H}}$ is k-wise independent.*

Proof. Assume that $\mathcal{H}$ is k-wise independent, and let $h_1, \ldots, h_k \in \mathbb{G}$ be distinct group elements. Then, for every k group elements $u_1, \ldots, u_k \in \mathbb{G}$ it holds that

$$\begin{aligned}
\Pr_{\widehat{f}\leftarrow\mathcal{G}_{\mathcal{H}}}\left[\forall i \in [k] \ : \ \widehat{f}(h_i) = u_i\right] &= \Pr_{f\leftarrow\mathcal{H}}\left[\forall i \in [k] \ : \ \widehat{f}(h_i) = u_i\right] \\
&= \Pr_{f\leftarrow\mathcal{H}}\left[\forall i \in [k] \ : \ h_i \cdot g^{f(h_i)} = u_i\right] \\
&= \Pr_{f\leftarrow\mathcal{H}}\left[\forall i \in [k] \ : f(h_i) = \mathsf{dlog}_g(u_i) - \mathsf{dlog}_g(h_i)\right] \\
&= \left(\frac{1}{N}\right)^k \qquad (3.1)
\end{aligned}$$

where for a group element $u \in \mathbb{G}$, $\mathsf{dlog}_g(u)$ is the unique $\mathbb{Z}_N$ element x such that $g^x = u$, and Eq. (3.1) follows from the k-wise independence of $\mathcal{H}$. ∎

For a group element $u \in \mathbb{G}$, a function f and an integer $k \in \mathbb{N}$, denote

$$C_{u,f,k} = \left(\widehat{f}^{(j)}(u)\right)_{j\in\{0,\ldots,k-1\}},$$

where $\widehat{f}^{(j)}(u) = \widehat{f}(\widehat{f}^{(j-1)}(u))$ and $\widehat{f}^{(0)}(u) = u$. That is, $C_{u,f,k}$ is the ordered multi-set of all group elements visited by a $(k-1)$-step walk in $\mathbb{G}$, which starts from u and progresses according to the function $\widehat{f}$. For $k = 0$ we use the convention that $C_{u,f,0}$ is the empty set. We define the following random variables:

- Let F denote the random variable corresponding to the hash function f chosen by A_0 in Step 1 by sampling $\mathcal{H} \leftarrow \mathsf{HashGen}(3\ell, c)$ and $f \leftarrow \mathcal{H}$.

- For each $i \in [s]$ let $G_{i,1}$ denote the random variable corresponding to the group element $g_{i,1} \in \mathbb{G}$ sampled uniformly by A_1 in Step 2a.
- Let H be the random variable corresponding to the uniformly-distributed group element $h \in \mathbb{G}$ that is given as input to A_1.

Note that using this notation, for each $i \in [s]$ it holds that $C_{G_{i,1},F,\ell}$ is a random variable corresponding to the multi-set of group elements computed by A_0 in each iteration of Step 2. Similarly, $C_{H,F,2\ell}$ is a random variable corresponding to the multi-set of group elements computed by A_1 in Step 2.

Observe that if $C_{H,F,\ell}$ (which contains the first ℓ elements computed by A_1) intersects $C_{G_{j,1},F,\ell}$ for some $j \in [s]$, then A_1 successfully outputs X for which $g^X = H$. This is the case since $C_{H,F,\ell} \cap C_{G_{j,1},F,\ell} \neq \emptyset$ implies that $G_j \in C_{H,F,2\ell}$, where G_j is included in the state st along with the corresponding exponent Y_j such that $G_j = g^{Y_j}$. Moreover, if G_j is the ith element computed by A_1, then the integer Δ_i computed by A_1 satisfies $G_j = H \cdot g^{\Delta_i}$. Therefore, we obtain

$$g^{Y_j} = G_j = H \cdot g^{\Delta_i}$$

which implies that

$$H = g^{Y_j - \Delta_i},$$

and that the output $X = Y_j - \Delta_i$ of A_1 is indeed the discrete logarithm of H with respect to g. Hence, in the remainder of the proof we will focus on bounding the probability that $C_{H,F,\ell} \cap C_{G_{j,1},F,\ell} \neq \emptyset$ for some $j \in [s]$. By the inclusion-exclusion principle, it holds that

$$\begin{aligned}
&\Pr\left[\bigvee_{j\in[s]} C_{H,F,\ell} \cap C_{G_{j,1},F,\ell} \neq \emptyset\right] \\
&\quad \geq \sum_{1\leq j\leq s} \Pr\left[C_{H,F,\ell} \cap C_{G_{j,1},F,\ell} \neq \emptyset\right] - \sum_{1\leq i<j\leq s} \Pr\left[\begin{array}{c} C_{H,F,\ell} \cap C_{G_{i,1},F,\ell} \neq \emptyset \\ \wedge\, C_{H,F,\ell} \cap C_{G_{j,1},F,\ell} \neq \emptyset \end{array}\right]. \qquad (3.2)
\end{aligned}$$

We now bound each of the sums in Eq. (3.2) separately, in Claims 3.4 and 3.5 below.

Claim 3.4. *For every $j \in [s]$ it holds that*

$$\Pr\left[C_{H,F,\ell} \cap C_{G_{j,1},F,\ell} \neq \emptyset\right] > \frac{1}{2\sqrt{e}} \cdot \frac{\ell^2}{N} - O\left(\frac{1}{(3\ell)^c}\right).$$

Proof. Let $j \in [s]$, and let F^* denote a random variable describing a function from $\mathbb{G}$ to $\mathbb{Z}_N$ distributed uniformly in a 3ℓ-wise independent family $\mathcal{F}$. We will prove that

$$\Pr\left[C_{H,F^*,\ell} \cap C_{G_{j,1},F^*,\ell} \neq \emptyset\right] > \frac{1}{2\sqrt{e}} \cdot \frac{\ell^2}{N},$$

and the claim then follows immediately from Lemma 2.3. By total probability,

$$\begin{aligned}
\Pr\left[C_{H,F^*,\ell} \cap C_{G_{j,1},F^*,\ell} \neq \emptyset\right] \geq\ &\Pr\left[C_{H,F^*,\ell} \cap C_{G_{j,1},F^*,\ell} \neq \emptyset \;\middle|\; |C_{H,F^*,\ell}| = \ell \wedge |C_{G_{j,1},F^*,\ell}| = \ell\right] \\
&\times \Pr\left[|C_{H,F^*,\ell}| = \ell \wedge |C_{G_{j,1},F^*,\ell}| = \ell\right].
\end{aligned}$$

Moreover, it holds that

$$\begin{aligned}
\Pr\left[|C_{H,F^*,\ell}| = \ell \wedge |C_{G_{j,1},F^*,\ell}| = \ell\right] &= 1 - \Pr\left[|C_{H,F^*,\ell}| < \ell \vee |C_{G_{j,1},F^*,\ell}| < \ell\right] \\
&\geq 1 - 2 \cdot \Pr\left[|C_{H,F^*,\ell}| < \ell\right] \qquad (3.3)
\end{aligned}$$

where Eq. (3.3) follows from the union bound and the fact that the random variables $|C_{H,F^*,\ell}|$ and $|C_{G_{j,1},F^*,\ell}|$ are identically distributed. We now turn to bound $\Pr\left[|C_{H,F^*,\ell}| < \ell\right]$. By total probability

$$\Pr\left[|C_{H,F^*,\ell}| < \ell\right] = \sum_{h\in\mathbb{G}} \Pr\left[H = h\right] \cdot \Pr\left[|C_{h,F^*,\ell}| < \ell\right]. \tag{3.4}$$

Since for every $h \in \mathbb{G}$, the events $\{|C_{h,F^*,i-1}| = i-1 \wedge |C_{h,F^*,i}| < i\}_{i\in[\ell]}$ form a partition of the event $|C_{h,F^*,\ell}| < \ell$, for every $h \in \mathbb{G}$ it holds that

$$\begin{aligned}
\Pr\left[|C_{h,F^*,\ell}| < \ell\right] &= \sum_{i=1}^{\ell} \Pr\left[|C_{h,F^*,i-1}| = i-1 \wedge |C_{h,F^*,i}| < i\right] \\
&\leq \sum_{i=1}^{\ell} \Pr\left[|C_{h,F^*,i-1}| = i-1 \mid |C_{h,F^*,i}| < i\right] \\
&= \sum_{i=1}^{\ell} \sum_{\substack{h_2,\ldots,h_{i-1}\in\mathbb{G}: \\ \forall 1\leq k<t\leq i-1,\ h_k\neq h_t}} \Pr\begin{bmatrix}\forall k \in [i-2] : \widehat{F^*}(h_k) = h_{k+1} \\ \wedge\ \widehat{F^*}(h_{i-1}) \in \{h_1,\ldots,h_{i-1}\}\end{bmatrix} \\
&= \sum_{i=1}^{\ell} \sum_{\substack{h_2,\ldots,h_{i-1}\in\mathbb{G}: \\ \forall 1\leq k<t\leq i-1,\ h_k\neq h_t}} \sum_{m\in[i-1]} \Pr\begin{bmatrix}\forall k \in [i-2] : \widehat{F^*}(h_k) = h_{k+1} \\ \wedge\ \widehat{F^*}(h_{i-1}) = h_m\end{bmatrix},
\end{aligned}$$

where $h_1 = h$ and $\widehat{F^*}$ is the function define by $\widehat{F^*}(u) = u \cdot g^{F^*(u)}$. By the fact that $\mathcal{G}_{\mathcal{F}}$ is 3ℓ-wise independent, then it is in particular ℓ-wise independent, and thus the following holds: For every $i \in [\ell]$, for every $h_1,\ldots,h_{i-1} \in \mathbb{G}$ and for every $m \in [i-1]$, the fraction of functions $\widehat{f}$ in $\mathcal{G}_{\mathcal{F}}$ which satisfy $\widehat{f}(h_k) = h_{k+1}$ for all $k \in [i-2]$ and $\widehat{f}(h_{i-1}) = h_m$ is $N^{-(i-1)}$. Hence, we obtain that for every $h \in \mathbb{G}$,

$$\begin{aligned}
\Pr\left[|C_{h,F^*,\ell}| < \ell\right] &\leq \sum_{i=1}^{\ell} \sum_{\substack{h_2,\ldots,h_{i-1}\in\mathbb{G}: \\ \forall 1\leq k<t\leq i-1,\ h_k\neq h_t}} (i-1)\cdot N^{-(i-1)} \\
&< \sum_{i=1}^{\ell} \frac{i-1}{N} \\
&\leq \frac{\ell^2}{N}.
\end{aligned}$$

Together with Eq. (3.3) and (3.4), this implies that

$$\begin{aligned}
\Pr\left[|C_{H,F^*,\ell}| = \ell \wedge |C_{G_{j,1},F^*,\ell}| = \ell\right] &\geq 1 - 2\sum_{h\in\mathbb{G}} \Pr\left[H = h\right] \cdot \frac{\ell^2}{N} \\
&= 1 - \frac{2\ell^2}{N}.
\end{aligned} \tag{3.5}$$

For each $i \in [\ell]$, let E_i denote the event in which

$$\left(C_{H,F^*,i-1} \cap C_{G_{j,1},F^*,\ell} = \emptyset\right) \wedge \left(C_{H,F^*,i} \cap C_{G_{j,1},F^*,\ell} \neq \emptyset\right).$$

That is, E_i is the event in which the ith element in $C_{H,F^*,\ell}$ is the first element in $C_{H,F^*,\ell}$ that also appears in $C_{G_{j,1},F^*,\ell}$. Then, the 3ℓ-wise independence of $\mathcal{F}$ implies in particular its 2ℓ-wise independence, and thus for each $i \in [\ell]$ it holds that

$$\begin{aligned}\Pr\left[\mathsf{E}_i \;\middle|\; |C_{H,F^*,\ell}| = \ell \wedge |C_{G_{j,1},F^*,\ell}| = \ell\right] &= \left(\prod_{k=1}^{i}\left(1 - \frac{\ell+k-1}{N}\right)\right) \cdot \frac{\ell}{N} \\ &\geq \left(1 - \frac{2\ell}{N}\right)^i \cdot \frac{\ell}{N},\end{aligned}$$

and since $1 - x \leq e^{-2x}$ for all $x \in [0, 1/2]$, the fact that $\ell \leq N/4$ implies that

$$\Pr\left[\mathsf{E}_i \;\middle|\; |C_{H,F^*,\ell}| = \ell \wedge |C_{G_{j,1},F^*,\ell}| = \ell\right] \geq e^{-2\ell i/N} \cdot \frac{\ell}{N}.$$

It thus follows that

$$\begin{aligned}&\Pr\left[C_{H,F^*,\ell} \cap C_{G_{j,1},F^*,\ell} \neq \emptyset \;\middle|\; |C_{H,F^*,\ell}| = \ell \wedge |C_{G_{j,1},F^*,\ell}| = \ell\right] \\ &\quad= \sum_{i=1}^{\ell} \Pr\left[\mathsf{E}_i \;\middle|\; |C_{H,F^*,\ell}| = \ell \wedge |C_{G_{j,1},H,\ell}| = \ell\right] \\ &\quad\geq \frac{\ell}{N} \cdot \sum_{i=1}^{\ell} e^{-2\ell i/N} \\ &\quad\geq \frac{\ell}{N} \cdot \ell \cdot e^{-2\ell^2/N}. \qquad (3.6)\end{aligned}$$

Taken together, Eq. (3.5) and Eq. (3.6) imply that

$$\Pr\left[C_{H,F^*,\ell} \cap C_{G_{j,1},F^*,\ell} \neq \emptyset\right] > \left(1 - \frac{2\ell^2}{N}\right) \cdot \frac{\ell^2}{N} \cdot e^{-2\ell^2/N},$$

and since $\ell \leq \sqrt{N}/2$, we obtain

$$\Pr\left[C_{H,F^*,\ell} \cap C_{G_{j,1},F^*,\ell} \neq \emptyset\right] > \frac{1}{2\sqrt{e}} \cdot \frac{\ell^2}{N}.$$

By Lemma 2.3, this implies that

$$\Pr\left[C_{H,F,\ell} \cap C_{G_{j,1},F,\ell} \neq \emptyset\right] > \frac{1}{2\sqrt{e}} \cdot \frac{\ell^2}{N} - O\left(\frac{1}{(3\ell)^c}\right).$$

■

Claim 3.5. *For every $1 \leq i < j \leq s$ it holds that*

$$\Pr\left[C_{H,F,\ell} \cap C_{G_{i,1},F,\ell} \neq \emptyset \wedge C_{H,F,\ell} \cap C_{G_{j,1},F,\ell} \neq \emptyset\right] \leq \frac{8\ell^4}{N^2} + O\left(\frac{1}{(3\ell)^c}\right).$$

Proof. Let i, j such that $1 \leq i < j \leq s$, and as before let F^* denote a random variable describing a function from $\mathbb{G}$ to $\mathbb{Z}_N$ distributed uniformly in a 3ℓ-wise independent family $\mathcal{F}$. We will prove that

$$\Pr\left[C_{H,F^*,\ell} \cap C_{G_{i,1},F^*,\ell} \neq \emptyset \wedge C_{H,F^*,\ell} \cap C_{G_{j,1},F^*,\ell} \neq \emptyset\right] \leq \frac{8\ell^4}{N^2},$$

and the claim then follows immediately from Lemma 2.3. Since the event $C_{H,F^*,\ell} \cap C_{G_{j,1},F^*,\ell} \neq \emptyset$ is contained in the event $C_{G_{j,1},F^*,\ell} \cap \left(C_{H,F^*,\ell} \cup C_{G_{i,1},F^*,\ell}\right) \neq \emptyset$, it holds that

$$\begin{aligned}
&\Pr\left[C_{H,F^*,\ell} \cap C_{G_{i,1},F^*,\ell} \neq \emptyset \wedge C_{H,F^*,\ell} \cap C_{G_{j,1},F^*,\ell} \neq \emptyset\right] \\
&\qquad \leq \Pr\left[C_{H,F^*,\ell} \cap C_{G_{i,1},F^*,\ell} \neq \emptyset \wedge C_{G_{j,1},F^*,\ell} \cap \left(C_{H,F^*,\ell} \cup C_{G_{i,1},F^*,\ell}\right) \neq \emptyset\right] \\
&\qquad \leq \Pr\left[C_{H,F^*,\ell} \cap C_{G_{i,1},F^*,\ell} \neq \emptyset\right] \qquad (3.7) \\
&\qquad\quad \cdot \Pr\left[C_{G_{j,1},F^*,\ell} \cap \left(C_{H,F^*,\ell} \cup C_{G_{i,1},F^*,\ell}\right) \neq \emptyset \,\middle|\, C_{H,F^*,\ell} \cap C_{G_{i,1},F^*,\ell} \neq \emptyset\right] \qquad (3.8)
\end{aligned}$$

For upper bounding Eq. 3.7, note that $|C_{H,F^*,\ell}| \leq \ell$ and $|C_{G_{i,1},F^*,\ell}| \leq \ell$, and therefore the 3ℓ-wise independence of $\mathcal{F}$ (which implies, in particular, 2ℓ-wise independence) guarantees that

$$\Pr\left[C_{H,F^*,\ell} \cap C_{G_{i,1},F^*,\ell} \neq \emptyset\right] \leq \left(1 - \left(1 - \frac{\ell}{N}\right)^{\ell}\right).$$

Similarly, for upper bounding Eq. 3.8, note that $|C_{G_{j,1},F^*,\ell}| \leq \ell$ and $|C_{H,F^*,\ell} \cup C_{G_{i,1},F^*,\ell}| \leq 2\ell$, and therefore the 3ℓ-wise independence of $\mathcal{F}$ guarantees that

$$\Pr\left[C_{G_{j,1},F^*,\ell} \cap \left(C_{H,F^*,\ell} \cup C_{G_{i,1},F^*,\ell}\right) \neq \emptyset \,\middle|\, C_{H,F^*,\ell} \cap C_{G_{i,1},F^*,\ell} \neq \emptyset\right] \leq \left(1 - \left(1 - \frac{2\ell}{N}\right)^{\ell}\right).$$

Since $1 - (1-x)^y \leq 2xy$ for all $x, y \in \mathbb{N}$ such that $x \leq 1/2$, and since $\ell \leq p/4$, these imply that

$$\Pr\left[C_{H,F^*,\ell} \cap C_{G_{i,1},F^*,\ell} \neq \emptyset \wedge C_{H,F^*,\ell} \cap C_{G_{j,1},F^*,\ell} \neq \emptyset\right] \leq \frac{2\ell^2}{N} \cdot \frac{4\ell^2}{N} = \frac{8\ell^4}{N^2},$$

and from Lemma 2.3 we obtain that

$$\Pr\left[C_{H,F,\ell} \cap C_{G_{i,1},F,\ell} \neq \emptyset \wedge C_{H,F,\ell} \cap C_{G_{j,1},F,\ell} \neq \emptyset\right] \leq \frac{8\ell^4}{N^2} + O\left(\frac{1}{(3\ell)^c}\right).$$

■

From Claims 3.4 and 3.5 and from Eq. (3.2), we obtain that

$$\begin{aligned}
\Pr\left[A_1\left(\mathbb{G}, N, g, h, \mathsf{st}\right) = \mathsf{dlog}_g(h)\right] &> \sum_{1 \leq j \leq s} \frac{1}{2\sqrt{e}} \cdot \frac{\ell^2}{N} - \sum_{1 \leq i < j \leq s} \frac{8\ell^4}{N^2} - O\left(\frac{1}{(3\ell)^c}\right) \\
&> \frac{1}{2\sqrt{e}} \cdot \frac{s \cdot \ell^2}{N} - 8 \cdot \frac{s^2 \cdot \ell^4}{N^2} - O\left(\frac{1}{(3\ell)^c}\right) \\
&> \frac{1}{4} \cdot \frac{s \cdot \ell^2}{N} - 8 \cdot \frac{s^2 \cdot \ell^4}{N^2} - O\left(\frac{1}{(3\ell)^c}\right).
\end{aligned}$$

Since $s \cdot \ell^2 \leq N/64$, this implies that

$$8 \cdot \frac{s^2 \cdot \ell^4}{N^2} \leq \frac{1}{8} \cdot \frac{s \cdot \ell^2}{N}$$

and hence

$$\Pr\left[A_1\left(\mathbb{G}, N, g, h, \mathsf{st}\right) = \mathsf{dlog}_g(h)\right] > \frac{1}{8} \cdot \frac{s \cdot \ell^2}{N} - O\left(\frac{1}{(3\ell)^c}\right).$$

This concludes the proof of Theorem 3.1. ■

References

[ADW14] M. Aumüller, M. Dietzfelbinger, and P. Woelfel. Explicit and efficient hash families suffice for cuckoo hashing with a stash. *Algorithmica*, 70(3):428–456, 2014.

[BHK+19] I. Berman, I. Haitner, I. Komargodski, and M. Naor. Hardness-preserving reductions via cuckoo hashing. *Journal of Cryptology*, 32(2):361–392, 2019.

[BL13] D. J. Bernstein and T. Lange. Non-uniform cracks in the concrete: The power of free precomputation. In *Advances in Cryptology – ASIACRYPT '13*, pages 321–340, 2013.

[BR93] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[CDG18] S. Coretti, Y. Dodis, and S. Guo. Non-uniform bounds in the random-permutation, ideal-cipher, and generic-group models. In *Advances in Cryptology – CRYPTO '18*, Lecture Notes in Computer Science, pages 693–721, 2018.

[CK18] H. Corrigan-Gibbs and D. Kogan. The discrete-logarithm problem with preprocessing. In *Advances in Cryptology – EUROCRYPT '18*, pages 415–447, 2018.

[DGK17] Y. Dodis, S. Guo, and J. Katz. Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In *Advances in Cryptology – EUROCRYPT '17*, volume 10211, pages 473–495, 2017.

[DP08] M. Dietzfelbinger and R. Pagh. Succinct data structures for retrieval and approximate membership. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, pages 385–396, 2008.

[DR09] M. Dietzfelbinger and M. Rink. Applications of a splitting trick. *ICALP 2009: Automata, Languages and Programming*, pages 354–365, 2009.

[DTT10] A. De, L. Trevisan, and M. Tulsiani. Time space tradeoffs for attacks against one-way functions and PRGs. In *Advances in Cryptology – CRYPTO '10*, pages 649–665, 2010.

[DW03] M. Dietzfelbinger and P. Woelfel. Almost random graphs with simple hash functions. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 629–638, 2003.

[FN99] A. Fiat and M. Naor. Rigorous time/space trade-offs for inverting functions. *SIAM Journal on Computing*, 29(3):709–803, 1999.

[FST10] D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, 2010.

[Hel80] M. E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transaction on Information Theory*, 26(4):401–406, 1980.

[HMP01] T. Hagerup, P. B. Miltersen, and R. Pagh. Deterministic dictionaries. *Journal of Algorithms*, 41(1):69–85, 2001.

[KMV00] N. Koblitz, A. Menezes, and S. A. Vanstone. The state of elliptic curve cryptography. *Designs, Codes and Cryptography*, 19(2/3):173–193, 2000.

[LCH11] H. T. Lee, J. H. Cheon, and J. Hong. Accelerating ID-based encryption based on trapdoor DL using pre-computation. Cryptology ePrint Archive, Report 2011/187, 2011.

[Mau05] U. Maurer. Abstract models of computation in cryptography. In *Proceedings of the 10th IMA International Conference on Cryptography and Coding*, pages 1–12, 2005.

[Mil99] P. B. Miltersen. Cell probe complexity - a survey. In *Proceedings of the 19th Conference on the Foundations of Software Technology and Theoretical Computer Science, Advances in Data Structures Workshop*, 1999.

[MPZ20] U. Maurer, C. Portmann, and J. Zhu. Unifying generic group models. Cryptology ePrint Archive, Report 2020/996, 2020.

[PP08] A. Pagh and R. Pagh. Uniform hashing in constant time and optimal space. *SIAM Journal on Computing*, 38(1):85–96, 2008.

[PR04] R. Pagh and F. F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[Sho97] V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology – EUROCRYPT '97*, pages 256–266, 1997.

[Sie04] A. Siegel. On universal classes of extremely random constant-time hash functions. *SIAM Journal on Computing*, 33(3):505–543, 2004.

[Unr07] D. Unruh. Random oracles and auxiliary input. In *Advances in Cryptology – CRYPTO '07*, pages 205–223, 2007.