On the Distributed Discrete Logarithm Problem with Preprocessing

Pavel Hubáček
Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic

Ľubica Jančová
Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic

Veronika Králová
Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic

Abstract

Protocols solving the Distributed Discrete Logarithm (DDLog) problem are a core component of many recent constructions of group-based homomorphic secret sharing schemes. On a high-level, these protocols enable two parties to transform multiplicative shares of a secret into additive share *locally without any communication.* Due to their important applications, various generic optimized DDLog protocols were proposed in the literature, culminating in the asymptotically optimal generic protocol of Dinur, Keller, and Klein (J. Cryptol. 2020) solving DDLog in time T with error probability $O(W/T^2)$ when the magnitude of the secret is bounded by W.

Given that DDLog is solved repeatedly with respect to a fixed group in its applications, a natural approach for improving the efficiency of DDLog protocols could be via leveraging some precomputed group-specific advice. To understand the limitations of this approach, we revisit the distributed discrete logarithm problem in the preprocessing model and study the possible time-space trade-offs for DDLog in the generic group model. As our main result, we show that, in a group of size N, any generic DDLog protocol for secrets of magnitude W with parties running in time T using precomputed group-specific advice of size S has success probability $\varepsilon = O\left(T^2/W + \max\{S, \log W\} \cdot T^2/N\right)$. Thus, assuming $N \geq W \log W$, we get a lower bound $ST^2 = \Omega(\varepsilon N)$ on the time-space trade-off for DDLog protocols using *large advice* of size $S = \Omega(N/W)$. Interestingly, for DDLog protocols using *small advice* of size $S = O(N/W)$, we get a lower bound $T^2 = \Omega(\varepsilon W)$ on the running time, which, in the constant-error regime, asymptotically matches the running time of the DDLog protocol *without any advice* of Dinur et al. (J. Cryptol. 2020). In other words, we show that generic DDLog protocols achieving constant success probability do not benefit from any advice of size $S = O(N/W)$ in the online phase of the DDLog problem.

2012 ACM Subject Classification Security and privacy → Information-theoretic techniques

Keywords and phrases Distributed discrete logarithm problem, preprocessing, generic group model

Digital Object Identifier 10.4230/LIPIcs.ITC.2022.6

Related Version *Full Version*: `https://eprint.iacr.org/2022/521`

Funding 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 are thankful to Siyao Guo for clarifications regarding the known techniques for establishing lower bounds for problems in the generic group model with preprocessing. We also wish to thank the anonymous ITC 2022 reviewers for helpful detailed comments on our results and their presentation.

1 Introduction

The *Distributed* Discrete Logarithm problem (DDLog) serves as an abstraction for a non-interactive multiplicative-to-additive share conversion procedure central to many recent constructions of homomorphic secret sharing schemes [4, 2, 11]. On a high-level, during a homomorphic evaluation of a restricted multiplication straight-line program using a homomorphic secret sharing scheme, two parties hold encryptions of the secret inputs under some homomorphic public-key encryption scheme (e.g., ElGamal encryption in [4] or Paillier encryption in [11]), as well as the additive shares of the inputs. After performing the multiplication operation "multiply an input value by a memory value", the parties naturally obtain multiplicative shares of the result. Though, additive shares are needed for further evaluation, and this is where the DDLog problem plays its crucial role.

Distributed discrete logarithm problem for prime-order groups

Let $(\mathbb{G}, \cdot)$ be a group in the multiplicative notation. Two parties P_0 and P_1 are holding $h_0 \in \mathbb{G}$ and $h_1 \in \mathbb{G}$ (respectively) such that $\frac{h_1}{h_0} = \omega^x$ for some $\omega \in \mathbb{G}$ and secret $x \in \mathbb{Z}$. *Without communicating*, the parties need to convert their *multiplicative* shares h_0 and h_1 to *additive* shares $a_0 \in \mathbb{Z}$ and $a_1 \in \mathbb{Z}$ such that $a_1 - a_0 = x$. For example, in the case of [4], the underlying group is a prime-order group and the element ω is its generator. Next, we consider the DDLog problem in this case in more detail.

Let $(\mathbb{G}, \cdot)$ be a cyclic group of prime-order N with a generator g. A natural parameter of the DDLog problem is the magnitude of the secret x. In the distributed discrete logarithm problem in $\mathbb{G}$ *on interval of size* $W \in \mathbb{N}$ such that $W \leq N$, the parties P_0 and P_1 get as input g^b and g^{b+x} respectively, where $x \leftarrow \{x \in \mathbb{Z} \cap [-W/2, W/2]\}$ and $b \leftarrow \mathbb{Z}_N$. The representation of the group $\mathbb{G}$ and the generator g is known to both the parties. At the end of their local executions, the parties output elements $P_0(g^b), P_1(g^{b+x}) \in \mathbb{Z}_N$ respectively and they succeed in solving the distributed discrete logarithm problem instance if $P_1(g^{b+x}) - P_0(g^b) = x$.

[4] gave a protocol solving the DDLog problem achieving error probability at most ε with time complexity $O(W\varepsilon^{-1} \log(\varepsilon^{-1}))$ group operations. Using a pseudorandom function shared among the parties, their protocol selects a set of "special" group elements and each of the parties searches through T points closest to her input, in sense of multiplication by the group generator g. When a party finds a "special" point, the output of the DDLog procedure on her input is the number of steps (i.e., multiplications by g) she made until finding the "special" point. It is clear that if both parties find the same "special" point, $P_0(g^b) - P_1(g^{b+x}) = x$ holds. Therefore, the success probability of the above DDLog protocol is exactly the probability of the parties synchronizing on the same "special" point. In the application towards a construction of a homomorphic secret sharing scheme, the error probability introduced by the DDLog protocol propagates throughout the whole homomorphic evaluation process and, subsequently, significantly affects its efficiency.

The DDLog problem in prime-order groups was further explored in [7]. First, the authors proposed a more sophisticated DDLog protocol resulting in error probability $O(W/T^2)$ in T group operations. Their procedure consists of iterating random walks with carefully chosen parameters of maximal step-length and number of steps in each stage in a way that they continuously reduces the probability of the two parties not synchronizing on their path.

Second, the authors also analyzed the limitations of protocols solving the DDLog problem. By a reduction of the *Discrete Logarithm in Interval* (DLI) problem to the DDLog problem, they proved that the error probability of DDLog protocols running in time T is $\Omega(W/T^2)$ in specific families of groups where the DLI problem is hard. Additionally, [7] also analyzed the

DDLog problem in the generic group model [12, 9], i.e., in a model inhibiting the attacker from exploiting a particular structure and properties of the underlying group. Specifically, they showed that the error probability of any generic DDLog protocol is $\Omega(W/T^2)$.

Due to the matching negative results, the DDLog protocol can be considered as optimal. The error probability $\Omega(W/T^2)$ also implies a non-negligible correctness error in the HSS construction from [4].

DDLog as a synchronization problem

Note that, at its core, the *distributed* DLog problem is different than the standard DLog. In order to succeed, the parties can "merely" synchronize on some special element with high probability. In particular, they certainly do not need to find the discrete logarithms of their respective inputs to synchronize.[1] One could thus attempt to construct more efficient DDLog protocols by leveraging some precomputed group-specific advice that, on one hand, would allow to synchronize with high probability faster while, on the other hand, would not compromise the security of the DLog problem in the underlying group. This approach is particularly well motivated, for example, from the perspective of trying to amortize the cost of the many DDLog instances generated during a homomorphic evaluation in a homomorphic secret sharing scheme.

1.1 Our Results

In this work, we focus on the distributed discrete logarithm problem for prime-order groups in the preprocessing model and examine to what extent can preprocessing help to solve the distributed discrete logarithm problem. In other words, we allow an *offline phase* of the protocol for precomputing a bit-string of length S before receiving the challenge pair (g^b, g^{b+x}) without any restriction on the time of its computation. Then, the precomputed *advice* bit-string is passed as an additional input to the two parties running in the *online phase*, i.e., after receiving the respective challenges. Regarding the time complexity of the protocol, we are interested in the performance of the online phase. To examine whether the additional *advice* computed during the offline phase can help to reduce the error probability in the DDLog problem, we study this question in the generic group model.

▶ **Theorem 1** (main – informal). *Let $N, W \in \mathbb{N}$, such that $N > W$, N be a prime. Then any generic protocol solving DDLog in a group of order N in time T using precomputed advice of size S has success probability*

$$\varepsilon = O\left(\frac{T^2}{W} + \frac{\max\{S, \log W\} \cdot T^2}{N}\right),$$

where W denotes the length of the interval in the DDLog problem. Furthermore, if $N \geq W \cdot \log W$ then

$$\varepsilon = O\left(\frac{T^2}{W} + \frac{S \cdot T^2}{N}\right).$$

Our main result summarized above is an upper bound on the success probability of protocols leveraging preprocessing. Assuming $N \geq W \log W$, our bound for a "large" preprocessing advice of size $S = \Omega(N/W)$ translates into a bound on time-space tradeoff

[1] Importantly, standard DLog is hard in the groups employed in the current applications of DDLog.

in the DDLog problem $ST^2 = \Omega(\varepsilon N)$. Note that this bound matches the lower bounds for standard discrete logarithm problem with preprocessing given by [5] and [6]. As the DDLog problem naturally reduces to two instances of the classical discrete logarithm problem, the DLog algorithms matching these lower bounds give an optimal algorithm for DDLog in this regime of parameters.

For any "small" preprocessing advice of size $S = O(N/W)$, our bound translates to a lower bound on the time complexity of the DDLog problem $T^2 = \Omega(\varepsilon W)$. Interestingly, for DDLog protocols with constant success probability in this regime of parameters, our lower bound on time-complexity of protocols *with preprocessing* is matched by the time complexity of the protocol *with no preprocessing* running in time $T^2 = O(\frac{W}{1-\varepsilon})$ from [7]. Hence, if we want to achieve a constant success probability in the DDLog problem, then allowing the attacker to precompute a preprocessing advice of size $S = O(N/W)$ does not asymptotically help to save computation time in the online phase of the DDLog problem.

Finally, let us consider the regime of parameters relevant to the application of DDLog in homomorphic secret sharing, i.e., the original motivation for DDLog. There, we expect W to be a polynomial function in the security parameter, i.e., $W = O(\text{polylog}(N))$. In this case, our assumption $N \geq W \log W$ is fulfilled. Then, in order to leverage the time-space trade-off matching our lower bound, we would need to allow advice of size $S = \Omega(N/\log N)$, which is larger than, e.g., $S = \sqrt[3]{N}$. Moreover, advice of this size allows to calculate discrete logarithms in polynomial time, which would ultimately break the security of the HSS scheme.

Non-generic DDLog protocols

It is important to stress that neither our results nor the results of Dinur et al. [7] rule out *non-generic* DDLog protocols exploiting particular structure of some groups in which DLI is not a difficult problem, or where the order of the underlying group is not prime. For example, this is the case of the DDLog protocol and a subsequent construction of homomorphic secret sharing in [11] from Paillier encryption scheme. Specifically, [11] exploits the fact that computing discrete logarithms is easy in a particular subgroup of the Paillier group, which allowed them to construct an efficient protocol with *perfect correctness*. Nevertheless, the problem of designing efficient generic DDLog protocols is of interest besides the applications to homomorphic secret sharing schemes. For example in the recent work of Boyle et al. [3], the DDLog protocol of Dinur et al. [7] was exploited towards constructions of a new flavour of locality-preserving hash function.

1.2 Our Techniques

On a high-level, our approach is similar to works that studied the *standard* discrete logarithm problem with preprocessing in the generic group model [6, 5]. Though, as explained above, there are significant differences to the standard DLog problem. In particular, the online adversary is distributed in our context and, thus, we need to introduce the corresponding formalism. Additionally, the DDLog problem is parameterized by the "magnitude" bound W corresponding to the width of the interval used for sampling the secret. We strive to derive quantitative bounds on the power of generic preprocessing adversaries that depend *both* on W and the order of the group N, which corresponds to analyzing the power of algorithms exploiting the knowledge of the parameter W.

Similarly to [5], we use the auxiliary-input generic group model in order to analyze DDLog protocols with preprocessing. Given that proving lower bounds in the auxiliary-input model directly is notoriously challenging, we first prove a lower bound in the more amenable bit-fixing model, which captures the presampling technique of Unruh [13]. Then, in order to translate our lower bound from the bit-fixing model into the auxiliary-input model, we leverage an analogue of theorem from [5] relating the security of primitives in the bit-fixing and auxiliary-input model in the presence of distributed online attackers.[2]

First, we notice that the proof of the correspondence between bit-fixing and auxiliary-input models in the presence of *distributed online attackers* follows via minor adjustments from the analogous correspondence for *standard online attackers* in [5]. We give the formal statement (Theorem 17) and a proof sketch highlighting the important points in Section 3.3.

The most technical part of our work is the upper bound on the success probability of a distributed online attacker for DDLog in the bit-fixing generic group model (BF-GGM). The high-level approach for proving an upper bound on the success probability in BF-GGM is to consider an alternative experiment in which the adversary clearly has limited probability of succeeding and then bounding the power of the adversary to distinguish such alternative experiment from the real security experiment. In more detail, we can create a list of formal polynomials capturing the relations among the group elements in terms of the encoding of the challenge g^x induced by the queries of the adversary to the group oracle. In the context of standard DLog, these are univariate polynomials in $\mathbb{Z}_N[X]$. The core of the proof is then analysing the probability of a collision event when substituting an actual value x into the polynomials – such a collision event corresponds to an inconsistent answer in the alternative experiment. In the context of distributed DLog, it is natural to consider bivariate polynomials in $\mathbb{Z}_N[X, B]$ that capture relations in terms of the encodings of the DDLog challenge (g^x, g^{x+b}). Though, the most significant technical difference is induced by the distributed nature of the online attacker. Specifically, we need to additionally handle a collision event among the queries of the two distributed parts of the online attackers.

2 Preliminaries

Below, we review the Schwartz-Zippel lemma (Proposition 2) and the standard Boole's inequality, also known as the union bound (Proposition 3).

▶ **Proposition 2** (Schwartz-Zippel). *Let $\mathbb{F}$ be a field. Let $f \in \mathbb{F}[X_1, \dots, X_k]$ be a non-zero polynomial of total degree $d \geq 0$. Let S be a finite subset of $\mathbb{F}$ and let $x_1, \dots, x_k$ be chosen independently uniformly at random from S. Then it holds that*

$$\Pr[f(x_1, \dots, x_k) = 0] \leq \frac{d}{|S|}.$$

▶ **Proposition 3** (Union Bound). *Let $n \in \mathbb{N}$ and consider events $E_1, \dots, E_n$, then*

$$\Pr\left[\bigcup_{i=1}^{n} E_i\right] \leq \sum_{i=1}^{n} \Pr\left[E_i\right].$$

[2] We note that one could possibly obtain similar results to ours by adapting the more recent technique from [8] leveraging an alternative characterization of the bit-fixing model.

3 DDLog with Preprocessing in the Generic Group Model

3.1 The Generic Group Model

In this section, we describe a variant of the generic group model that we consider in this work, and we define the DDLog problem in this generic group model. We use a variant of the generic group model [12, 9] similar to the one in [5] with adjustments accommodating distributed attackers.

Let $\mathbb{G}$ be a cyclic group of prime-order N with a generator g. The generic group model allows to capture the power of algorithms that do not leverage additional knowledge about the structure of the group or the specific representation of group elements. In more detail, the structure of $\mathbb{G}$ is random in the sense that the access of the adversary to the group elements is given by a random injective function $\sigma : \mathbb{Z}_N \to [M]$ for some natural number $M \geq N$. Thus, the elements of $\mathbb{G}$ are represented by the elements from $\mathrm{Im}(\sigma)$, i.e., for $a \in \mathbb{Z}_N$, the representation of the group element g^a is $\sigma(a)$. We call the mapping σ an *encoding function*.

▶ **Definition 4** (Encoding function)**.** *Let $N, M \in \mathbb{N}$ be natural numbers such that $M \geq N$. Consider $\mathbb{Z}_N$, the additive group of integers modulo N, and the set $[M] = \{1, \dots, M\}$. An* encoding function of $\mathbb{Z}_N$ on $[M]$ *is an injective mapping $\sigma \colon \mathbb{Z}_N \to [M]$. We denote $\mathcal{I}_{N,M}$ the set of all encoding functions of $\mathbb{Z}_N$ on $[M]$ and $\mathcal{Y}_\sigma$ the image of σ.*[3]

In order to perform group operations in the generic group model, the adversary $\mathcal{A}$ is granted access to a group-operation oracle $\mathcal{O}$, which chooses a random encoding function $\sigma \leftarrow \mathcal{I}_{N,M}$ at the beginning of the experiment and then answers adversary's queries of the following types:

Forward query: Any query $a \in \mathbb{Z}_N$ is answered by $\sigma(a) \in [M]$.

Group-operation query: Any query $(s_1, s_2) \in [M] \times [M]$ is answered by $\sigma(x_1 + x_2)$, where x_1, x_2 are elements of $\mathbb{Z}_N$ such that $\sigma(x_1) = s_1$ and $\sigma(x_2) = s_2$. If any of s_1, s_2 is not in $\mathcal{Y}_\sigma$, the query is answered by $\perp$.

Inverse query: Any query $s \in [M]$ is answered by $\sigma(-x)$, where $x \in \mathbb{Z}_N$ is the preimage of s, i.e., $\sigma(x) = s$. If $s \notin \mathcal{Y}_\sigma$, the query is answered by $\perp$.

In the generic group model, we measure the time complexity of the attacker by the number of oracle queries performed during its execution.

▶ Remark 5 (Explicit inverse queries). Unlike previous works studying the standard DLog with preprocessing [6, 5] and the DDLog without preprocessing [7] in the generic group model, we allow the adversary to make *explicit* inverse queries. This choice is important for our results in order to achieve quantitatively better bounds. Note that an inverse query can be implemented using $O(\log N)$ group-operation queries. Thus, a lower bound in a variant of the generic group model *without* explicit inverse queries yields an equivalent lower bound in our model up to a multiplicative logarithmic factor in N. However, in order to derive meaningful bounds for the DDLog problem, we cannot afford to neglect logarithmic factors in N and, therefore, we analyse the performance of protocols that can make explicit inverse queries. We note that other works previously considered explicit inverse queries in GGM. See, for example, Neven, Smart, and Warinschi [10] or Blocki and Lee [1].

[3] We omit the subscript σ in $\mathcal{Y}_\sigma$ when the encoding is clear from the context.

$\mathsf{Exp}_{\mathcal{A}}^{G^{\mathcal{O}}}(\lambda)$:

1. On input 1^λ, the challenger $\mathcal{C}$ generates a secret x from a space of secrets X and a challenge (c_0, c_1) from the challenge space $\mathcal{CH}_x$. It forwards $(1^\lambda, c_0)$ to $\mathcal{A}^{(0)}$ and $(1^\lambda, c_1)$ to $\mathcal{A}^{(1)}$.
2. For $i = 0, 1$:
   a. On input $(1^\lambda, c_i)$, the attacker $\mathcal{A}^{(i)}$ makes queries to oracle $\mathcal{O}$ and receives answers from the oracle.
   b. The attacker $\mathcal{A}^{(i)}$ chooses a guess $x_i \in X$ and sends it to $\mathcal{C}$.
3. The output of the experiment is 1 if $x_1 - x_0 = x$ and 0 otherwise.

Figure 1 Distributed unpredictability experiment.

Unpredictability application

The distributed discrete logarithm problem can also be seen as a problem of guessing secret information given a challenge from some challenge space dependent on the secret. In particular, the attacker tries to guess x given the challenge of the form g^b, g^{b+x}. Nevertheless, the unpredictability application does not provide a good representation for this problem, as it carries several differences compared to it. Especially, the attacker in the distributed discrete logarithm problem is composed of two algorithms who cannot communicate and each of these algorithms only gets to see half of the challenge, moreover, "to solve" the DDLog problem, is to get the secret x secret-shared between these two algorithms, not known explicitly by any of them. We call this special type of attacker a *distributed attacker* and we represent similar problems by *distributed unpredictability application.*

▶ **Definition 6** (Distributed attacker). *A* distributed attacker $(\mathcal{A}^{(0)}, \mathcal{A}^{(1)})$ *is a pair of independent PPT $\mathcal{A}^{(0)}$ and $\mathcal{A}^{(1)}$, who cannot communicate. We say the distributed attacker runs in the time T if each of $\mathcal{A}^{(0)}$ and $\mathcal{A}^{(1)}$ does not make more than T oracle queries during its execution.*

Below, we define a distributed unpredictability application, which captures the problem of guessing some secret information x given a challenge from a challenge space $\mathcal{CH}_x$ in the generic group model.

▶ **Definition 7** (Distributed unpredictability application). *Let $\lambda \in \mathbb{N}$ be a security parameter. A* distributed unpredictability application *G in the oracle $\mathcal{O}$-model is defined by a space of secrets X, a set of challenge spaces $\{\mathcal{CH}_x \mid x \in X\}$, where the elements of $\mathcal{CH}_x$ are of form (c_0, c_1), a PPT challenger $\mathcal{C}$ with oracle access and an oracle $\mathcal{O}$. We define the advantage of the distributed attacker $\mathcal{A} = (\mathcal{A}^{(0)}, \mathcal{A}^{(1)})$ on G, denoted $\mathsf{Adv}_{\mathcal{A}}^{G^{\mathcal{O}}}(\lambda)$, as the probability of success of $\mathcal{A}$ in the distributed unpredictability experiment $\mathsf{Exp}_{\mathcal{A}}^{G^{\mathcal{O}}}(1^\lambda)$ defined in Figure 1, i.e.,*

$$\mathsf{Adv}_{\mathcal{A}}^{G^{\mathcal{O}}}(\lambda) = \Pr\left[\mathsf{Exp}_{\mathcal{A}}^{G^{\mathcal{O}}}(\lambda) = 1\right].$$

Now, we define the distributed discrete logarithm problem in the generic group model.

▶ **Definition 8** (Distributed discrete logarithm application). *Let $\lambda \in \mathbb{N}$ be a security parameter, let $N, W \in \mathbb{N}$, $N > W$ and $\mathcal{O}$ be a group operation oracle. The (N, W)-*distributed discrete logarithm application *$G_{\mathsf{DDLog}}^{\mathcal{O}}(N, W)$ is a distributed unpredictability application in $\mathcal{O}$-model, where the space of secrets is $X = \{x \mid x \in \mathbb{Z} \cap [-W/2, W/2]\}$, the challenge space for $x \in X$*

is defined as $\mathcal{CH}_x = \{(\sigma(b), \sigma(b+x)) \mid \sigma \in \mathcal{I}_{N,M}, b \in \mathbb{Z}_N\}$*, and the challenger* C^{DDLog} *is a PPT that samples* $b \leftarrow \mathbb{Z}_N$ *and* x *from the set of integers in the interval* $[-W/2, W/2]$ *uniformly at random. Then* C^{DDLog} *makes two forward queries* $b, b+x$ *to the oracle and passes* $(1^\lambda, \sigma(b))$ *as a challenge to* $\mathcal{A}^{(0)}$ *and* $(1^\lambda, \sigma(b+x))$ *as a challenge to* $\mathcal{A}^{(1)}$*.*

3.2 Preprocessing in the Generic Group Model

In this section, we introduce the definitional framework of the generic group model for the algorithms with preprocessing. The framework is adapted from [5] with some small adjustments in order to capture the distributed attacker. First, we define the preprocessing oracle.

▶ **Definition 9** (Preprocessing oracle). *The* preprocessing oracle $\mathcal{O}$ *is an oracle formed by a pair of oracles* $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$*, where* $\mathcal{O}_{\mathsf{pre}}$ *can only be queried during the offline phase of an experiment, i.e., before the challenge is generated and* $\mathcal{O}_{\mathsf{main}}$ *can only be queried in the online phase of an experiment.*

The attacker in the preprocessing model is composed of two algorithms, one of them running in the offline phase and having access to $\mathcal{O}_{\mathsf{pre}}$, the other one running in the online phase and having an access to $\mathcal{O}_{\mathsf{main}}$.

▶ **Definition 10** ((S,T)-attacker, [5, Definition 1]). *Let* $S, T \in \mathbb{N}$ *and* $\mathcal{O} = (\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$ *be a preprocessing oracle. An* (S,T)-attacker $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ in the $\mathcal{O}$-model *consist of two probabilistic algorithms:*

1. *A* preprocessing algorithm $\mathcal{A}_0$*, which is computationally unbounded and which interacts with* $\mathcal{O}_{\mathsf{pre}}$ *and outputs a bit-string of length at most* S *bits.*
2. *An* online algorithm $\mathcal{A}_1$*, which takes as input an* S*-bit output of* $\mathcal{A}_0$ *and a challenge from the challenger, then makes at most* T *queries to* $\mathcal{O}_{\mathsf{main}}$*, and outputs a guess.*

Now, we define the notions of *unpredictability application* and *distributed unpredictability application* in the preprocessing model. Definitions 12 and 13 correspond to Definitions 6 and 7 extended to the preprocessing model. The (S,T)*-attacker* (Definition 10) and the *unpredictability application with preprocessing* (Definition 11) correspond to the model used by [5], whereas the *distributed* (S,T)*-attacker* (Definition 12) and the *distributed unpredictability application with preprocessing* (Definition 7) allow us to model the DDLog problem with preprocessing in the generic group model.

The following definition formalizes the problem of guessing secret information x given a challenge from a related challenge space $\mathcal{CH}_x$, while we allow the attacker to precompute advice string before receiving the challenge.

▶ **Definition 11** (Unpredictability application with preprocessing). *Let* $\lambda \in \mathbb{N}$ *be a security parameter. An* unpredictability application with preprocessing G *in the oracle* $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$*-model is defined by a space of secrets* X*, a set of challenge spaces* $\{\mathcal{CH}_x \mid x \in X\}$*, a PPT challenger* $\mathcal{C}$ *with oracle access to* $\mathcal{O}_{\mathsf{main}}$ *and a preprocessing oracle* $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$*. We define the advantage of* $\mathcal{A}$ *on* G*, denoted* $\mathsf{Adv}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda)$*, as the probability of success of* $\mathcal{A}$ *in the unpredictability experiment with preprocessing* $\mathsf{Exp}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda)$ *defined in Figure 2, i.e.,*

$$\mathsf{Adv}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda) = \Pr\left[\mathsf{Exp}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda) = 1\right].$$

We say that an unpredictability application with preprocessing G *is* (S,T,ε)*-secure in the* $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$*-model if, for every* (S,T)*-attacker* $\mathcal{A}$ *and every* $\lambda \in \mathbb{N}$*, it holds that*

$$\mathsf{Adv}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda) \le \varepsilon(\lambda).$$

$\mathsf{Exp}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda)$:

1. The attacker $\mathcal{A}_0$ makes queries to oracle $\mathcal{O}_{\mathsf{pre}}$ and receives answers from the oracle.
2. At the end of its execution, $\mathcal{A}_0$ outputs an advice bit-string adv of maximal length S bits and forwards adv to the online phase attacker $\mathcal{A}_1$.
3. The challenger $\mathcal{C}$ generates a secret x from the space X and a challenge c from the challenge space $\mathcal{CH}_x$. It forwards $(1^\lambda, c)$ to $\mathcal{A}_1$.
4. On input $(1^\lambda, \mathsf{adv}, c)$, the attacker $\mathcal{A}_1$ makes queries to the oracle $\mathcal{O}_{\mathsf{main}}$ and receives answers from the oracle.
5. The attacker $\mathcal{A}_1$ chooses a guess $x' \in X$ and sends it to $\mathcal{C}$.
6. The output of the experiment is 1 if $x = x'$ and 0 otherwise.

Figure 2 Unpredictability experiment with preprocessing.

As the online attacker in the DDLog problem is not a single algorithm, yet an attacker composed of two algorithms who cannot communicate, we need to introduce a definition of such attacker in the preprocessing model.

▶ **Definition 12** (Distributed (S,T)-attacker)**.** *Let $S, T \in \mathbb{N}$ and $\mathcal{O} = (\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$ be a preprocessing oracle. A* distributed (S,T)-attacker $\mathcal{A} = (\mathcal{A}_0, (\mathcal{A}_1^{(0)}, \mathcal{A}_1^{(1)}))$ in the $\mathcal{O}$-model *consist of three probabilistic algorithms:*

1. *A preprocessing algorithm $\mathcal{A}_0$, which is computationally unbounded and which interacts with $\mathcal{O}_{\mathsf{pre}}$ and outputs a bit-string of length S bits.*
2. *A pair of online algorithms $(\mathcal{A}_1^{(0)}, \mathcal{A}_1^{(1)})$, which cannot communicate and each of which takes as input an S-bit output of $\mathcal{A}_0$ and a challenge, then makes at most T queries to $\mathcal{O}_{\mathsf{main}}$, and outputs a guess.*

Now, we define the *distributed unpredictability application with preprocessing*, which formalizes the problem of the distributed (S,T)-attacker's online algorithms guessing additive shares of secret information x, given a challenge from the related challenge space $\mathcal{CH}_x$.

▶ **Definition 13** (Distributed unpredictability application with preprocessing)**.** *Let $\lambda \in \mathbb{N}$ be a security parameter. A* distributed unpredictability application with preprocessing *G in the oracle $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$-model is defined by a space of secrets X, a set of challenge spaces $\{\mathcal{CH}_x \mid x \in X\}$, where the elements of $\mathcal{CH}_x$ are of the form (c_0, c_1), a PPT challenger $\mathcal{C}$ with oracle access to $\mathcal{O}_{\mathsf{main}}$ and a preprocessing oracle $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$. We define the advantage of a distributed (S,T)-attacker $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1^{(0)}, \mathcal{A}_1^{(1)})$ in G, denoted $\mathsf{Adv}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda)$, as the probability of success of $\mathcal{A}$ in the distributed unpredictability experiment with preprocessing $\mathsf{DistExp}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda)$ defined in Figure 3, i.e.,*

$$\mathsf{Adv}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda) = \Pr\left[\mathsf{DistExp}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda) = 1\right].$$

We say a distributed unpredictability application with preprocessing G is (S,T,ε)-secure in the $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$-model if, for every distributed (S,T)-attacker $\mathcal{A}$ and every $\lambda \in \mathbb{N}$, it holds that $\mathsf{Adv}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda) \le \varepsilon(\lambda)$.

Now, we define the DDLog problem with preprocessing in the generic group model.

$\mathsf{DistExp}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda)$:

1. The attacker $\mathcal{A}_0$ makes queries to oracle $\mathcal{O}_{\mathsf{pre}}$ and receives answers from the oracle.
2. At the end of its execution $\mathcal{A}_0$ outputs an advice bit-string adv of maximal length S bits and forwards adv to the online phase attackers $\mathcal{A}_1^{(0)}, \mathcal{A}_1^{(1)}$.
3. The challenger $\mathcal{C}$ generates a secret x from the space X and a challenge (c_0, c_1) from the challenge space $\mathcal{CH}_x$. It forwards $(1^\lambda, c_0)$ to $\mathcal{A}_1^{(0)}$ and $(1^\lambda, c_1)$ to $\mathcal{A}_1^{(1)}$.
4. For $i = 0, 1$:
   a. On input $(\mathsf{adv}, 1^\lambda, c_i)$ the attacker $\mathcal{A}^{(i)}$ makes at most T queries to oracle $\mathcal{O}_{\mathsf{main}}$ and receives answers from the oracle.
   b. The attacker $\mathcal{A}^{(i)}$ chooses a guess $x_i \in X$ and sends it to $\mathcal{C}$.
5. The output of the experiment is 1 if $x_1 - x_0 = x$ and 0 otherwise.

Figure 3 Distributed unpredictability experiment with preprocessing.

▶ **Definition 14** (Distributed discrete logarithm application with preprocessing). *Let $\lambda \in \mathbb{N}$ be a security parameter, let $N, W \in \mathbb{N}$, $N > W$. The* (N, W)-distributed discrete logarithm application with preprocessing $G_{\mathsf{DDLog}}^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}(N, W)$ *is a distributed unpredictability application with preprocessing in $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$-model, where $\mathcal{O}_{\mathsf{pre}}$ is an oracle that samples $\sigma \leftarrow \mathcal{I}_{N,M}$ at the beginning of the experiment and $\mathcal{O}_{\mathsf{main}}$ is a group operation oracle for the encoding function σ, the space of secrets is $X = \mathbb{Z} \cap [-W/2, W/2]$, the challenge space for $x \in X$ is defined as $\mathcal{CH}_x = \{(\sigma(b), \sigma(b + x)) \mid b \in \mathbb{Z}_N\}$. The challenger C^{DDLog} is a PPT algorithm that samples $x \leftarrow X$ and $b \leftarrow \mathbb{Z}_N$. Then C^{DDLog} makes two forward queries $b, b + x$ to the oracle $\mathcal{O}_{\mathsf{main}}$ and passes $(1^\lambda, \sigma(b))$ as a challenge to $\mathcal{A}_1^{(0)}$ and $(1^\lambda, \sigma(b + x))$ as a challenge to $\mathcal{A}_1^{(1)}$.*

3.3 The Auxiliary-Input and Bit-Fixing Models

Following the technique of [5], we define two different preprocessing oracles: *Auxiliary-input generic group oracle* and *Bit-fixing generic group oracle*. The auxiliary-input generic group oracle allows us to model the preprocessing experiment. Nevertheless, it seems difficult to perform an analysis of complexity directly in this model, while the bit-fixing generic group oracle offers a model that is easier to analyse. [5, Theorem 1] proved a relation between an attackers' success probabilities in these two models. We state this result in Proposition 16.

The auxiliary-input generic group oracle allows modelling the preprocessing experiments for generic groups, in the sense that the interface $\mathcal{O}_{\mathsf{pre}}$ allows the offline attacker to see the entire group structure, i.e., the mapping σ. Then, $\mathcal{A}_0$ can choose a bit-string of maximal length S and pass it to the online phase attacker $\mathcal{A}_1$ as an additional input.

On the other hand, the bit-fixing generic group oracle allows the offline attacker to fix P points $(a, s) \in \mathbb{Z}_N \times \mathcal{Y}$ and the mapping σ is chosen afterwards, in the way that it respects these fixed points.

▶ **Definition 15.** *We define:*
Auxiliary-input generic group oracle $\mathsf{AI\text{-}GG}(N, M)$ *is a pair $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$, where:*
- $\mathcal{O}_{\mathsf{pre}}$*: Samples $\sigma \leftarrow \mathcal{I}_{N,M}$ and outputs σ.*
- $\mathcal{O}_{\mathsf{main}}$*: Answers forward queries, group-operation queries, and inverse queries using σ sampled by $\mathcal{O}_{\mathsf{pre}}$.*

Bit-fixing generic group oracle $\mathsf{BF\text{-}GG}(P, N, M)$ *is a pair* $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$*, where:*

- $\mathcal{O}_{\mathsf{pre}}$*: Samples* $\mathcal{Y} \subset [M]$ *of size* N *uniformly at random, takes as input at most* $P \in \mathbb{N}$ *pairs of the form* (a, s)*,* $a \in \mathbb{Z}_N$*,* $s \in \mathcal{Y}$ *with no collisions, and samples* σ *uniformly at random from the subset of* $\mathcal{I}_{N,M}$ *containing all the injections consistent with the sampled image* $\mathcal{Y}$ *and the given fixed points.*
- $\mathcal{O}_{\mathsf{main}}$*: Answers forward queries, group-operation queries, and inverse queries using* σ *sampled by* $\mathcal{O}_{\mathsf{pre}}$*.*

▶ **Proposition 16** ([5, Theorem 1])**.** *Let* $P, N, M \in \mathbb{N}$*,* $N \geq 16$ *and* $\gamma > 0$*. Consider a* (S, T, ε')*-secure unpredictability application with preprocessing* G *in the* $\mathsf{BF\text{-}GG}(P, N, M)$*-model. If* $P \geq 6(S + \log \gamma^{-1}) \cdot T_G^{\mathsf{comb}}$*, then* G *is* (S, T, ε)*-secure in the* $\mathsf{AI\text{-}GG}(N, M)$*-model for* $\varepsilon \leq 2\varepsilon' + \gamma$*. Where* T_G^{comb} *denotes the combined number of queries of the attacker and the challenger.*

Proof. For the proof we refer to [5, Appendix A]. ◀

Importantly, Proposition 16 was formulated in [5] only for (S, T, ε)-secure unpredictability applications with preprocessing, while not considering *distributed* applications. However, we are interested in proving upper bounds on the success probability of an attacker in the *distributed* discrete logarithm application with preprocessing, which is a distributed unpredictability application with preprocessing. If we applied Proposition 16 to the DDLog problem directly, we would be forced to represent the distributed attacker as a non-distributed attacker in the $\mathsf{BF\text{-}GG}(P, N, M)$-model, i.e., an attacker with an online algorithm that gets both challenges and performs up to $2T$ oracle queries. Then, we would apply the theorem on this *stronger* attacker and we would obtain the bounds in the $\mathsf{AI\text{-}GG}(N, M)$-model. Thus, the clear disadvantage of the distributed attacker of having the challenge given to two separate algorithms that are not allowed to communicate cannot be exploited in the $\mathsf{BF\text{-}GG}(P, N, M)$-model before applying the Proposition 16. Overall, this approach leads to loose bounds on the success probability of the distributed attacker. Nevertheless, our central observation is that the theorem holds in the same way for a distributed attacker.

We also remark that the generic group model used in [5] does not allow the attacker to perform inverse queries. This approach is justified by the fact that the authors apply Proposition 16 to derive bounds precise up to polylogarithmic factors in N. The inverse of an element x in a group of order N is equal to x^{N-1}. Therefore, applying the standard square-and-multiply algorithm, we can simulate the inverse operation using $O(\log N)$ group-operations. Thus, the analysis in a version of the generic group model without the inverse queries translates to the result precise up to polylogarithmic factors in N in a model where these queries are allowed. However, we seek to get bounds for the DDLog problem without neglecting logarithmic factors in N, and, therefore, we explicitly include the inverse query in our generic group model. We note that Proposition 16 holds also in our version of the generic group model. We explain the siginificant differences in more detail in the proof sketch of Theorem 17 below.

▶ **Theorem 17.** *Let* $P, N, M \in \mathbb{N}$*,* $N \geq 16$ *and* $\gamma > 0$*. Consider a* (S, T, ε')*-secure distributed unpredictability application with preprocessing* G *in the* $\mathsf{BF\text{-}GG}(P, N, M)$*-model. If* $P \geq 6(S + \log \gamma^{-1}) \cdot T_G^{\mathsf{comb}}$*, then* G *is* (S, T, ε)*-secure in the* $\mathsf{AI\text{-}GG}(N, M)$*-model for* $\varepsilon \leq 2\varepsilon' + \gamma$*. Where* T_G^{comb} *denotes the combined number of queries of the attacker and the challenger.*

Proof sketch. The proof follows from the proof of Proposition 16 stated in [5, Appendix A] replacing the (S, T)-attacker with preprocessing by a distributed (S, T)-attacker with preprocessing.

In order to prove [5, Theorem 1], the authors first prove closeness of two distributions of encoding functions, in the sense that they bound the probability that a distinguisher which is allowed to make T forward (query $a \in \mathbb{Z}_N$ is answered by $\sigma(a)$) and backward (query $l \in [M]$ is answered by $\sigma^{-1}(l)$) oracle queries succeeds to guess from which distribution the encoding function σ of $\mathbb{Z}_N$ on $[M]$ was chosen. In fact, this part of their proof is general and can be applied in the same manner in the setting with distributed attackers.

Then, to prove Proposition 16, they construct an (S, T)-attacker $\mathcal{A}' = (\mathcal{A}'_0, \mathcal{A}'_1)$ for the $\mathsf{BF\text{-}GG}(P, N, M)$ oracle model from an (S, T)-attacker $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ for the $\mathsf{AI\text{-}GG}(N, M)$ oracle model. Their $\mathcal{A}'_1$ is defined as $\mathcal{A}_1$ and their $\mathcal{A}'_0$ first simulates $\mathcal{A}_0$ to get the advice string and, based on it, it fixes at most P points in the encoding function. Thus, $\mathcal{A}'_0$ forces the oracle to choose the encoding function from a convenient distribution. Then, they let the distinguisher $\mathcal{D}$ for the encoding function internally run the online algorithm $\mathcal{A}_1$ on the advice string calculated by $\mathcal{A}_0$ and, subsequently, the challenger $\mathcal{C}$ for the unpredictability experiment with preprocessing. The output of the distinguisher is defined as the output of the unpredictability experiment with preprocessing resulting from the interaction of $\mathcal{A}_1$ and $\mathcal{C}$. The probability of the distinguisher outputting 1 corresponds either to the probability of success of $\mathcal{A}$ in the $\mathsf{AI\text{-}GG}(N, M)$ oracle model or to the probability of success of $\mathcal{A}'$ in the $\mathsf{BF\text{-}GG}(N, M)$ oracle model, depending on which distribution was σ chosen from. Due to the bound on the distinguishing probability between the two aforementioned distributions, they get a relation between the success probabilities of $\mathcal{A}$ in the $\mathsf{AI\text{-}GG}(N, M)$-model and $\mathcal{A}'$ in the $\mathsf{BF\text{-}GG}(P, N, M)$-model stated in the theorem, which concludes their proof.

If we replace the attacker $\mathcal{A}$ by a distributed attacker and define $\mathcal{A}'$ in the same way as in the original proof, $\mathcal{A}'$ will also be a distributed attacker (as the online algorithms of $\mathcal{A}$ and $\mathcal{A}'$ are defined to be the same) and we can perform the proof in the very same fashion as in [5] and get the same results for the distributed attacker.

The proof of Proposition 16 can also be adapted to the version of generic group model which allows inverse queries. When the distinguisher $\mathcal{D}$ simulates $\mathcal{A}_1$ and $\mathcal{C}$, it must provide the answers to their oracle queries. In their proof, [5] only dealt with the forward query, which $\mathcal{D}$ passes as a forward query to its oracle and passes the answer back, and the group-operation query, which is a query of the form $(a_1, a_2) \in [M]^2$, answered by $\sigma(\sigma^{-1}(a_1) + \sigma^{-1}(a_2))$. The group-operation query is simulated by $\mathcal{D}$ by performing two backward queries $\sigma^{-1}(a_1), \sigma^{-1}(a_2)$ and one forward query $\sigma(\sigma^{-1}(a_1) + \sigma^{-1}(a_2))$ to its oracle. In our case, we have to deal also with the inverse query, which is a query of the form $a \in [M]$, answered by the element corresponding to the inverse of a. This query can be simulated as one backward query $\sigma^{-1}(a)$ and one forward query $\sigma(-\sigma^{-1}(a))$. As in the original proof, our distinguisher makes at most $3T^{\mathsf{comb}}$ queries, where T^{comb} is the combined number of queries of $\mathcal{A}_1$ and $\mathcal{C}$. The rest of the proof is the same, and the same results follow. ◀

4 Lower Bounds for DDLog with Preprocessing in the Generic Group Model

Now, we present our main theorem giving an upper bound on the success probability of a distributed attacker with preprocessing in the DDLog problem. Our theorem is based on [5, Theorem 10], which examines the discrete logarithm problem in the preprocessing model. The structure of our proof is similar to the one of [5, Theorem 10]. However, if we simply adjusted the proof of [5, Theorem 10] to our case, we would get the bound for the success probability of an attacker $\varepsilon = O\left(\frac{ST^2 + T^2 \cdot \log(W)}{W}\right)$.

Nevertheless, the distributed attacker allows us to make a more careful analysis and obtain quantitatively better results. As the distributed attacker is a *weaker* attacker than its non-distributed representation, we can obtain a tighter bound in the $\mathsf{BF\text{-}GG}(P, N, M)$ oracle model. Then, we translate this bound to the $\mathsf{AI\text{-}GG}(N, M)$ oracle model using Theorem 17.

▶ **Theorem 18.** *Let $N, W \in \mathbb{N}$, $N > W$, N be a prime. The (N, W)-distributed discrete logarithm application with preprocessing $G^{\mathsf{AI\text{-}GG}(N,M)}_{\mathsf{DDLog}}(N, W)$ is (S, T, ε)-secure in the $\mathsf{AI\text{-}GG}(N, M)$-model for any*

$$\varepsilon = O\left(\frac{T^2}{W} + \frac{\max\{S, \log W\} \cdot T^2}{N}\right),$$

where W denotes the length of the interval in the DDLog problem. Furthermore, if $N \geq W \cdot \log(W)$ the theorem holds for

$$\varepsilon = O\left(\frac{T^2}{W} + \frac{S \cdot T^2}{N}\right).$$

The proof of Theorem 18 is given in Appendix A.

References

1 Jeremiah Blocki and Seunghoon Lee. On the multi-user security of short Schnorr signatures. *IACR Cryptol. ePrint Arch.*, page 1105, 2019. URL: `https://eprint.iacr.org/2019/1105`.

2 Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic secret sharing: Optimizations and applications. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2105–2122. ACM, 2017. `doi:10.1145/3133956.3134107`.

3 Elette Boyle, Itai Dinur, Niv Gilboa, Yuval Ishai, Nathan Keller, and Ohad Klein. Locality-preserving hashing for shifts with connections to cryptography. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 27:1–27:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. `doi:10.4230/LIPIcs.ITCS.2022.27`.

4 Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 509–539. Springer, 2016. `doi:10.1007/978-3-662-53018-4_19`.

5 Sandro Coretti, Yevgeniy Dodis, and Siyao Guo. Non-uniform bounds in the random-permutation, ideal-cipher, and generic-group models. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 693–721. Springer, 2018. `doi:10.1007/978-3-319-96884-1_23`.

6 Henry Corrigan-Gibbs and Dmitry Kogan. The discrete-logarithm problem with preprocessing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 415–447. Springer, 2018. `doi:10.1007/978-3-319-78375-8_14`.

7 Itai Dinur, Nathan Keller, and Ohad Klein. An optimal distributed discrete log protocol with applications to homomorphic secret sharing. *J. Cryptol.*, 33(3):824–873, 2020. `doi:10.1007/s00145-019-09330-2`.

8 Siyao Guo, Qian Li, Qipeng Liu, and Jiapeng Zhang. Unifying presampling via concentration bounds. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part I*, volume 13042 of *Lecture Notes in Computer Science*, pages 177–208. Springer, 2021. doi:10.1007/978-3-030-90459-3_7.

9 Ueli M. Maurer. Abstract models of computation in cryptography. In Nigel P. Smart, editor, *Cryptography and Coding, 10th IMA International Conference, Cirencester, UK, December 19-21, 2005, Proceedings*, volume 3796 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005. doi:10.1007/11586821_1.

10 Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. Hash function requirements for Schnorr signatures. *J. Math. Cryptol.*, 3(1):69–87, 2009. doi:10.1515/JMC.2009.004.

11 Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of Paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 678–708. Springer, 2021. doi:10.1007/978-3-030-77870-5_24.

12 Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997. doi:10.1007/3-540-69053-0_18.

13 Dominique Unruh. Random oracles and auxiliary input. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 205–223. Springer, 2007. doi:10.1007/978-3-540-74143-5_12.

A Proof of Theorem 18

We start with a short overview of the proof. In order to bound the advantage of a distributed attacker in the $\mathsf{AI\text{-}GG}(N, M)$-model, we analyse its advantage in the $\mathsf{BF\text{-}GG}(P, N, M)$-model and we use Theorem 17 to translate this bound to the $\mathsf{AI\text{-}GG}(N, M)$-model. To bound the attacker's advantage in the $\mathsf{BF\text{-}GG}(P, N, M)$-model, we construct an alternative experiment, where we answer the attacker's queries with randomly chosen elements, while we keep answers consistent in between themselves maintaining a table of already seen elements. The secrets $x \in \mathbb{Z} \cap [-W/2, W/2]$ and $b \in \mathbb{Z}_N$ are chosen uniformly at random from the respective sets at the end of the experiment, after the attacker outputs its guess. Then, we express the probability of success of the distributed attacker in this alternative experiment and we bound the probability that the distribution of query responses given in the alternative experiment differs from the one that would be given in an honest execution (Lemmas 20 and 21). This gives us the bound on the attacker's advantage in the $\mathsf{BF\text{-}GG}(P, N, M)$-model.

Now, we proceed with the formal proof of the theorem. First, consider the interaction of $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1^{(0)}, \mathcal{A}_1^{(1)})$ with C^{DDLog} in the $\mathsf{BF\text{-}GG}(P, N, M)$-model, during which the $\mathcal{O}_{\mathsf{pre}}$ outputs $\mathcal{Y}$, the image of σ. In the rest of the proof, we condition on the choice of $\mathcal{Y}$. Next, we define an *alternative experiment*, during which we construct a table of pairs $(v(X), s)$, where $s \in \mathcal{Y}$ and $v(X) \in \mathbb{Z}_N[X, B]$ is a formal polynomial corresponding to the preimage of s under σ. The experiment and, correspondingly, the construction of the table proceed as follows:

1. $\mathcal{A}_0$ fixes σ in at most P points (a, s), $a \in \mathbb{Z}_N$, $s \in \mathcal{Y}$. We add each such point to the table as a pair (a, s), where a is a constant polynomial.
2. To create the challenge $s_b \in \mathcal{Y}$ for $\mathcal{A}_1^{(0)}$, C^{DDLog} chooses s_b from all unused values in $\mathcal{Y}$ uniformly at random. The pair (B, s_b) is stored in the table.
3. The execution of $\mathcal{A}_1^{(0)}$:
   a. Upon a forward query $q \in \mathbb{Z}_N$ the table is checked for the occurrence of $q \in \mathbb{Z}_N[X, B]$ as a constant polynomial. If such occurrence is found, we respond by the corresponding $s_q \in \mathcal{Y}$ that occurs in the table in a pair with q. Otherwise, we sample s_q uniformly at random from all the unused values in $\mathcal{Y}$ and we store the pair (q, s_q) in the table.
   b. To a group-operation query (s_1, s_2) we respond by $\perp$ if s_1 or s_2 is not in $\mathcal{Y}$. Otherwise, if s_1 is not in the table, we sample a_1 uniformly at random from all unused values in $\mathbb{Z}_N$ and we store the pair (a_1, s_1) in the table. The same applies for s_2. Afterwards, both s_1, s_2 are already stored in the table in pairs with some polynomials $u_1, u_2 \in \mathbb{Z}_N[B]$. We check the table for an occurrence of the polynomial $u_1 + u_2$. If there is a record $(u_1 + u_2, s_3)$ for some $s_3 \in \mathcal{Y}$ in the table, we respond by s_3 to the query. Otherwise we sample s_3 uniformly at random from all unused values in $\mathcal{Y}$, we store $(u_1 + u_2, s_3)$ in the table, and we respond by s_3 to the query.
   c. To an inverse query s_1, we respond by $\perp$ if $s_1 \notin \mathcal{Y}$. If $s_1 \in \mathcal{Y}$ and s_1 is not in the table, we sample u uniformly at random from all unused values in $\mathbb{Z}_N$ and we append the pair (u, s_1) to the table. Now, s_1 is in the table in pair with some polynomial $u \in \mathbb{Z}_N[B]$. We check the table for an occurrence of $(N-1) \cdot u$, if such entry $((N-1) \cdot u, s_2)$ is found for some $s_2 \in \mathcal{Y}$, we answer the query by s_2. Otherwise, we sample s_2 uniformly at random from all unused labels from $\mathcal{Y}$, answer the query by s_2, and we add the pair $((N-1) \cdot u, s_2)$ to the table.
   d. At the end of its execution, $\mathcal{A}_1^{(0)}$ outputs x_0.
4. To create the challenge s_{b+x} for $\mathcal{A}_1^{(1)}$, C^{DDLog} chooses s_{b+x} from all unused values in $\mathcal{Y}$ uniformly at random. The pair $(B + X, s_{b+x})$ is stored in the table.
5. The execution of $\mathcal{A}_1^{(1)}$ is handled in the same way as the execution of $\mathcal{A}_1^{(0)}$. Except if
   - $\mathcal{A}_1^{(1)}$ queries a group-operation query (s_1, s_2) such that either $(\alpha \cdot B + \beta, s_1)$, or $(\alpha \cdot B + \beta, s_2)$, for some $\alpha, \beta \in \mathbb{Z}_N$, $\alpha \neq 0$ is already in the table, or
   - $\mathcal{A}_1^{(1)}$ queries an inverse query s_1 such that $(\alpha \cdot B + \beta, s_1)$ for some $\alpha, \beta \in \mathbb{Z}_N$, $\alpha \neq 0$ is already in the table.

   We denote the event when either one of the above bullet points happens as F. If the event F occurs, we answer the query by $\perp$ and we do not append anything to the table. At the end of its execution, $\mathcal{A}_1^{(1)}$ outputs x_1.
6. C^{DDLog} chooses x uniformly at random from all integers in $[-W/2, W/2]$ and b uniformly at random from $\mathbb{Z}_N$. C^{DDLog} outputs 1 if and only if $x_1 - x_0 = x$.

Note that all of the polynomials in the table at the end of the execution are distinct, as we always first check for an occurrence of a polynomial before adding it to the table. We define a collision event, denoted E, as the event when, after a substitution of the values x, b chosen by C^{DDLog} for the variables X, B (respectively) in the polynomials in the table, there exist two entries (a, s) and (a, s') in the table such that $s \neq s'$. Note that the event E corresponds to a discrepancy in the query responses to the attacker. In other words, there is no encoding function σ such that all of our query responses would be correct because we associated the image of a with two distinct elements s, s'. If the event E does not occur and the event F does not occur either then there exists an encoding function σ compatible with all of our query responses. Furthermore, as we always choose the elements uniformly at random from the appropriate sets, also the distribution of responses in the alternative experiment is identical to the distribution of responses in the real unpredictability experiment with preprocessing.

Thus, the distribution of answers seen by $\mathcal{A}$ in the alternative experiment differs from the one in the honest experiment only if at least one of the events E, F occurs. We bound the probability that the execution differs from the honest execution by bounding the probability $\Pr[E \cup F] \leq \Pr[E \mid \neg F] + \Pr[F]$.

Below, we introduce a lemma that characterizes the structure of the contents of the table at the end of the alternative experiment.

▶ **Lemma 19.** *At most $2T + 2$ non-constant polynomials are in the table at the end of the execution of the alternative experiment. Moreover,*

1. *at most $T + 1$ of those are of the form $\alpha B + \beta$, for some $\alpha, \beta \in \mathbb{Z}_N$, $\alpha \neq 0$ (we say "polynomials of type 1") and they were added either as the challenge for $\mathcal{A}_1^{(0)}$ in the step 2 or as a polynomial corresponding to a group-operation query response during the execution of $\mathcal{A}_1^{(0)}$ in the step 3 b, or as a polynomial in pair with a response to an inverse query in the step 3 c. Furthermore, the value $s \in \mathcal{Y}$ in pair with such polynomial in the table is never returned as an answer to a query made by $\mathcal{A}_1^{(1)}$. Also,*
2. *at most $T + 1$ of non-constant polynomials in the table are of the form $\alpha(X + B) + \beta$, for some $\alpha, \beta \in \mathbb{Z}_N$, $\alpha \neq 0$ (we say "polynomials of type 2") and they were added either as the challenge for $\mathcal{A}_1^{(1)}$ in the step 4 or as a polynomial corresponding to a group-operation query response or inverse query response during the execution of $\mathcal{A}_1^{(1)}$ in step 5. Furthermore, the value $s \in \mathcal{Y}$ in pair with such polynomial in the table is never returned as an answer to a query made by $\mathcal{A}_1^{(0)}$.*

There are no non-constant polynomials in the table of forms different than the polynomials of type 1 and 2.

Proof of Lemma 19. First, notice that a non-constant polynomial can only be introduced in the table in a pair with a challenge s_b, s_{b+x}, or in a pair with a response to a group-operation or inverse query, where at least one of the queried elements s_1, s_2 or the queried element s_1 (for the inverse query) has already been in the table in pair with a non-constant polynomial. As there are two challenges and at most $2T$ queries performed by the distributed attacker, there is at most $2T + 2$ non-constant polynomials in the table at the end of the execution.

Next, we show that the second part of the lemma holds. Notice that the first polynomial with a non-zero coefficient next to the variable X is added to the table with the challenge s_{b+x} in the step 4. Therefore, the polynomials with a non-zero coefficient next to the variable X can be added to the table as a result of a group-operation query or an inverse query only after this moment. Since there are at most T queries performed after this moment, there cannot be more than $T + 1$ polynomials with a non-zero coefficient next to the variable X, and, thus, not more than $T + 1$ polynomials of type 2. Furthermore, when a new polynomial is added to the table, we sample its pair value s from the unused values in $\mathcal{Y}$. As the polynomials of type 2 appear in the table only after the end of execution of $\mathcal{A}_1^{(0)}$, none of the values s in pairs with such polynomials could have been returned as an answer to a query made by $\mathcal{A}_1^{(0)}$.

Next, we prove that, at the end of the alternative experiment, there are no non-constant polynomials of forms different than polynomials of type 1 and 2 in the table.

Recall that non-constant polynomials are only being added to the table in the challenge pair, during a group-operation query as a sum of two polynomials already present in the table, where at least one of them is non-constant, or during an inverse query as an $(N - 1)$

multiple of a polynomial in pair with the queried element if it is non-constant. Therefore, it is obvious that all polynomials in the table are at most of degree one in both B and X. Now, it is enough to show there is no polynomial of the form $\alpha_1 B + \alpha_2 X + \beta$, where $\alpha_1, \alpha_2, \beta \in \mathbb{Z}_N$, $\alpha_2 \neq 0$, $\alpha_1 \neq \alpha_2$ that we refer to as a "polynomial of type 3". We already know no polynomial of this form has been added to the table before the step 4 of the experiment because all of the polynomials until this step are of degree zero in the variable X. The polynomial added with the challenge in step 4 is $X + B$, which is a polynomial of type 2. Therefore, it is enough to look at the polynomials added to the table during step 5. We prove by induction that no non-constant polynomials of type different than type 2 are added to the table during step 5. Suppose no non-constant polynomials different than type 2 were added to the table during step 5 before the i-th query in step 5. For $i = 1$, the assumption holds trivially. We prove the inductive step for $i + 1$ next. By the inductive hypothesis, it follows that there are no polynomials different from the constant polynomials, polynomials of type 1, and polynomials of type 2 before the i-th query of step 5 in the table. In case i-th query is a forward query, either a constant polynomial or no polynomial is added to the table. In case the i-th query is a group operation query (s_1, s_2), the following cases can occur:

1. At least one of the pair s_1, s_2 is not in $\mathcal{Y}$. Then $\perp$ is returned and no polynomials are added to the table.
2. Both s_1, s_2 lie in $\mathcal{Y}$, and either none of them is in the table or one of them is in the table in pair with a constant polynomial and the other one is not in the table or both are in the table in pair with a constant polynomial. Then, only constant polynomials are added to the table.
3. Both s_1, s_2 lie in $\mathcal{Y}$, one of them is in the table in pair with a polynomial of type 2, and the other one is not in the table, is in the table in pair with a constant polynomial, or is in the table in pair with a polynomial of type 2. Then, the response will be in the table in pair with a polynomial of type 2 or a constant polynomial.
4. Both s_1, s_2 lie in $\mathcal{Y}$ and one of them is in the table in pair with a polynomial of type 1. Then, the query is answered by $\perp$ and no polynomials are added to the table.

In case the i-th query is an inverse query s_1, the following cases can occur:

1. The element s_1 does not lie in $\mathcal{Y}$. Then, $\perp$ is returned and no polynomials are added to the table.
2. The element s_1 lies in $\mathcal{Y}$ and s_1 is not in the table, or it is in the table in a pair with a constant polynomial. Then, only constant polynomials are added to the table.
3. The element s_1 lies in $\mathcal{Y}$ and s_1 is in the table in a pair with a polynomial of type 2. Then, the response is in the table in pair with a polynomial of type 2.
4. The element s_1 lies in $\mathcal{Y}$ and s_1 is in the table in pair with a polynomial of type 1. Then, the query is answered by $\perp$ and no polynomials are added to the table.

By the above exhaustive case-analysis, no non-constant polynomial of type different than type 2 was added during the i-th query. Therefore, no non-constant polynomial of type different than type 2 was added during step 5. Thus, no non-constant polynomials of forms different than the polynomials of type 1 and 2 are in the table at the end of the execution of the alternative experiment.

Part 1 of the lemma follows from the fact that no non-constant polynomial of type different than type 2 is being added to the table during the execution of $\mathcal{A}_1^{(1)}$. More precisely, the polynomials of type 1 can only be introduced in the table in a pair with the challenge s_b or in a pair with a response to a group-operation query or an inverse query by $\mathcal{A}_1^{(0)}$. Since $\mathcal{A}_1^{(0)}$ makes at most T queries, at most $T + 1$ polynomials of type 1 are in the table at the end of the execution. By the analysis of the possible group-operation and inverse query responses during step 5, an s in a pair with a polynomial of type 1 is never returned as an answer to a query made by $\mathcal{A}_1^{(1)}$. This concludes the proof of Lemma 19. ◀

The following lemma bounds the probability $\Pr[E \mid \neg F]$ (Due to space restrictions, the proof of the lemma appears in the full version).

▶ **Lemma 20.** *The probability* $\Pr[E \mid \neg F]$ *can by bounded as follows*

$$\Pr[E \mid \neg F] \leq \frac{(T+1)^2}{W} + \frac{(2T+2)\cdot(P+6T+1)}{N}.$$

In the following lemma, we bound the probability $\Pr[F]$ (Due to space restrictions, the proof of the lemma appears in the full version).

▶ **Lemma 21.** *The probability* $\Pr[F]$ *can be bounded as follows*

$$\Pr[F] \leq \frac{2\cdot T\cdot(T+1)}{N-(P+5T)}.$$

By the above Lemmas 20 and 21, we get

$$\Pr[E\cup F] \leq \frac{(T+1)^2}{W} + \frac{(2T+2)\cdot(P+6T+1)}{N} + \frac{2\cdot T\cdot(T+1)}{N-(P+5T)}.$$

Since x is chosen at the end of the experiment uniformly at random from the integer values in the interval $[-W/2, W/2]$ in the alternative experiment, the success probability of any $\mathcal{A}$ in the alternative experiment is at most $1/W$. By the union bound, we can bound the success probability ε' of $\mathcal{A}$ in the standard experiment as

$$\varepsilon' \leq \frac{(T+1)^2+1}{W} + \frac{(2T+2)\cdot(P+6T+1)}{N} + \frac{2\cdot T\cdot(T+1)}{N-(P+5T)}.$$

In the rest of the proof, we assume that $N \geq 16$ (required by Proposition 16). This can be done without loss of generality since we are proving an asymptotic bound. Now, we apply Theorem 17 in order to bound the attacker's success probability ε in the $\mathsf{AI\text{-}GG}(N, M)$-model. It holds that $T^{\mathsf{comb}}_{G^{\mathsf{DDLog}}} = 2T+2$, and we set $\gamma := 1/W$ and $P := 6(S+\log(W))\cdot(2T+2)$. By Theorem 17, we get that

$$\begin{aligned}\varepsilon &\leq 2\cdot\varepsilon' + \gamma\\ &\leq \frac{2\cdot(T+1)^2+3}{W} + \frac{2\cdot(2T+2)\cdot(P+6T+1)}{N} + \frac{4\cdot T\cdot(T+1)}{N-(P+5T)}.\end{aligned}$$

In the rest of the proof, we assume $T \geq 72$. Note that, after proving our result for the attackers with $T \geq 72$, we can bound the advantage of an attacker making less queries as follows. Suppose an attacker $\mathcal{A}$ makes $0 < T < 72$ queries during its execution. Note that any attacker $\mathcal{B}$ making $\tilde{T} := 72\cdot T$ queries can simulate $\mathcal{A}$. However, by our result, we can bound the advantage of $\mathcal{B}$ as $\epsilon_{\mathcal{B}} = O\left(\frac{\tilde{T}^2}{W} + \frac{\max\{S, \log W\}\cdot\tilde{T}^2}{N}\right)$. Therefore, the advantage $\epsilon_{\mathcal{A}}$ of $\mathcal{A}$ is also bounded by this expression and, as $\tilde{T} = 72\cdot T$, it also holds that $\epsilon_{\mathcal{A}} = O\left(\frac{T^2}{W} + \frac{\max\{S, \log W\}\cdot T^2}{N}\right)$.

Furthermore, assume that $N \geq 72 \cdot \max\{S, \log(W), 1\} \cdot T$. Otherwise if $N < 72 \cdot \max\{S, \log(W), 1\} \cdot T \leq \max\{S, \log(W), 1\} \cdot T^2$, then $\frac{\max\{S, \log(W), 1\} \cdot T^2}{N} > 1$ and the theorem's bound is looser than $\varepsilon = O(1)$, which holds trivially. Thus, we get

$$\begin{aligned} N - (P + 5T) &= N - (6(S + \log(W)) \cdot (2T + 2) + 5T) \\ &\geq N - (12 \cdot \max\{S, \log(W), 1\} \cdot (2T + 2) + 5T) \\ &\geq N - 36 \cdot \max\{S, \log(W), 1\} \cdot T \\ &\geq N/2. \end{aligned}$$

Consequently, for the sum of the second and the third term, we have

$$\begin{aligned} &\frac{2 \cdot (2T + 2) \cdot (P + 1 + 6T)}{N} + \frac{4 \cdot T \cdot (T + 1)}{N - (P + 5T)} \\ &\leq \frac{2 \cdot (2T + 2) \cdot (P + 1 + 6T)}{N} + \frac{8 \cdot T \cdot (T + 1)}{N} \\ &= \frac{4 \cdot (T + 1)(P + 1 + 8T)}{N} \\ &= \frac{4 \cdot (T + 1)(6 \cdot (S + \log W)(2T + 2) + 1 + 8T)}{N} \\ &= \frac{48 \cdot (T + 1)^2 \cdot (S + \log W) + (4T + 4) \cdot (1 + 8T)}{N} \\ &\leq \frac{96 \cdot \max\{S, \log(W)\} \cdot (T + 1)^2 + (4T + 4) \cdot (1 + 8T)}{N} \\ &\leq \frac{192 \cdot \max\{S, \log(W), 1\} \cdot T^2}{N}, \end{aligned}$$

where the last inequality holds as we assume $T \geq 72$. Specifically, for all $T \geq 72$, it holds that

$$96T^2 \geq 96 \cdot (2T + 1) + (4T + 4) \cdot (1 + 8T) = 32T^2 + 228T + 100,$$

which we use to bound the sub-quadratic terms in T in the nominator.

Therefore, it holds that

$$\varepsilon \leq \frac{3T^2}{W} + \frac{192 \cdot \max\{S, \log(W), 1\} \cdot T^2}{N} = O\left(\frac{T^2}{W} + \frac{\max\{S, \log(W)\} \cdot T^2}{N}\right).$$

Furthermore, if $N \geq (W \cdot \log(W))$ then we get that

$$\begin{aligned} \varepsilon \leq& \frac{3T^2}{W} + \frac{192 \cdot \max\{S, 1\} \cdot T^2}{N} + \frac{192 \cdot \log(W) \cdot T^2}{N} \\ \leq& \frac{3T^2}{W} + \frac{192 \cdot \max\{S, 1\} \cdot T^2}{N} + \frac{192 \cdot T^2}{W} = O\left(\frac{T^2}{W} + \frac{S \cdot T^2}{N}\right). \end{aligned}$$

The above bounds establish Theorem 18.