An Optimal Distributed Discrete Log Protocol with Applications to Homomorphic Secret Sharing*

Itai Dinur[1], Nathan Keller[2], and Ohad Klein[2]

[1] Department of Computer Science, Ben-Gurion University, Israel
[2] Department of Mathematics, Bar-Ilan University, Israel

Abstract. The distributed discrete logarithm (DDL) problem was introduced by Boyle et al. at CRYPTO 2016. A protocol solving this problem was the main tool used in the share conversion procedure of their homomorphic secret sharing (HSS) scheme which allows non-interactive evaluation of branching programs among two parties over shares of secret inputs.

Let g be a generator of a multiplicative group $\mathbb{G}$. Given a random group element g^x and an unknown integer $b \in [-M, M]$ for a small M, two parties A and B (that cannot communicate) successfully solve DDL if $A(g^x) - B(g^{x+b}) = b$. Otherwise, the parties err. In the DDL protocol of Boyle et al., A and B run in time T and have error probability that is roughly linear in M/T. Since it has a significant impact on the HSS scheme's performance, a major open problem raised by Boyle et al. was to reduce the error probability as a function of T.

In this paper we devise a new DDL protocol that substantially reduces the error probability to $O(M \cdot T^{-2})$. Our new protocol improves the asymptotic evaluation time complexity of the HSS scheme by Boyle et al. on branching programs of size S from $O(S^2)$ to $O(S^{3/2})$. We further show that our protocol is optimal up to a constant factor for all relevant cryptographic group families, unless one can solve the discrete logarithm problem in a *short* interval of length R in time $o(\sqrt{R})$.

Our DDL protocol is based on a new type of random walk that is composed of several iterations in which the expected step length gradually increases. We believe that this random walk is of independent interest and will find additional applications.

Keywords: Homomorphic secret sharing, share conversion, fully homomorphic encryption, discrete logarithm, discrete logarithm in a short interval, random walk.

1 Introduction

Homomorphic Secret Sharing Homomorphic secret sharing (HSS) is a practical alternative approach to fully homomorphic encryption (FHE) [13, 19] that

* A preliminary version of the paper was presented at the CRYPTO 2018 conference [10].

provides some of its functionalities. It was introduced by Boyle, Gilboa and Ishai [5] at CRYPTO 2016 and further studied and extended in [4, 6, 7, 11]. The main advantage of HSS over traditional secure multiparty computation protocols [1, 8, 22] is that, similarly to FHE, its communication complexity is smaller than the circuit size of the computed function.

HSS allows homomorphic evaluation to be distributed among two parties who do not interact with each other. A (2-party) HSS scheme randomly splits an input w into a pair of shares (w_0, w_1) such that: (1) each share w_i computationally hides w, and (2) there exists a polynomial-time local evaluation algorithm Eval such that for any program P from a given class (e.g., a boolean circuit or a branching program), the output $P(w)$ can be efficiently reconstructed from $\text{Eval}(w_0, P)$ and $\text{Eval}(w_1, P)$.

The main result of [5] is an HSS scheme for branching programs under the Decisional Diffie-Hellman (DDH) assumption that satisfies $P(w) = \text{Eval}(w_0, P) + \text{Eval}(w_1, P)$. It was later optimized in [4, 6], where the security of the optimized variants relies on other discrete log style assumptions.

Let $\mathbb{G}$ be a multiplicative cyclic group of prime order N in which the discrete log problem is (presumably) hard and let g be a generator of this group. The scheme of [5] allows the parties to locally multiply an encrypted (small) input $w \in \mathbb{Z}$ with an additively secret-shared (small) value $y \in \mathbb{Z}$, such that the result $z = wy$ is shared between the parties. The problem is that at this stage g^z is multiplicatively shared by the parties, so they cannot multiply z with a new encrypted input w'. Perhaps the most innovative idea of [5] allows the parties to convert multiplicative shares of g^z into additive shares of z without any interaction via a share conversion procedure. Once the parties have an additive sharing of z, they can proceed to add it to other additive shares. These operations allow to evaluate restricted multiplication straight-line (RMS) programs which can emulate any branching program of size S using $O(S)$ instructions.

The share conversion procedure of [5] is not perfect in the sense that the parties may err. More specifically, the parties fail to compute correct additive shares of z with some error probability δ that depends on the running time T of the parties and on a small integer M that bounds the intermediate computation values. As share conversion is performed numerous times during the execution of Eval, its total error probability accumulates and becomes roughly $\delta \cdot S$, where S is the number of multiplications performed by the RMS program P. Thus, for the total error probability to be constant one has to set the running time T of the parties in the share conversion procedure such that $\delta \approx 1/S$. Consequently, the running time to error tradeoff has a significant impact on the performance of the HSS scheme.

Since the main motivation behind HSS is to provide a practical alternative to FHE, one of the main open problems posed in [5] was to improve the running time to error tradeoff of the share conversion procedure. Progress on this open problem was made in the followup works [4, 6] which significantly improved the practicality of the HSS scheme. Despite this progress, the asymptotic running time to error tradeoff of the share conversion procedure was not substantially

improved and the running time T in all the schemes grows (roughly) linearly with the inverse error probability $1/\delta$ (or as M/δ in general). Thus, to obtain $\delta \approx 1/S$, one has to set $T \approx S$, and since the total number of multiplications in P is S, the total running time becomes $O(S^2)$.

The Distributed Discrete Log Problem The focus of this paper is on the "distributed discrete log" (DDL) problem which the parties collectively solve in the share conversion procedure. We now describe the DDL problem and abstract away the HSS details for simplicity. The DDL problem involves two parties A and B. The input of A consists of a group element g^x, where x is chosen uniformly at random from $\mathbb{Z}_N$. The input of B consists of g^{x+b}, where $b \in [-M, M]$ is an unknown uniformly chosen integer in the interval (for a small fixed integer parameter M). The algorithms A, B are restricted by a parameter T which bounds the number of group operations they are allowed to compute.[3] After executing its algorithm, each party outputs an integer. The parties successfully solve the DDL instance if $A(g^x) - B(g^{x+b}) = b$. We stress that A and B are not allowed to communicate.[4]

If g^z is multiplicatively shared by A (party 0) and B (party 1), then $g^{z_0} \cdot g^{z_1} = g^z$. In the share conversion procedure party A runs $A(g^{-z_0})$ while party B runs $B(g^{z_1})$. Assuming they correctly solve DDL for $|z| \leq M$, we have $A(g^{-z_0}) - B(g^{z_1}) = z_1 + z_0 = z$, namely, $A(g^{-z_0})$ and $-B(g^{z_1})$ are additive shares of z as required.

It is convenient to view the DDL problem as a synchronization problem: A and B try to agree or *synchronize* on a group element with a known offset in the exponent from their input. If they manage to do so, the parties solve DDL by outputting this offset. For example, if both parties synchronize on g^y, then $A(g^x) = y - x$ while $B(g^{x+b}) = y - (x + b)$, so $A(g^x) - B(g^{x+b}) = b$ as required. In particular, if both A and B can solve the discrete logarithm problem for their input (i.e., compute x and $x + b$, respectively), then they can synchronize on the generator g by outputting $A(g^x) = 1 - x$ and $B(g^{x+b}) = 1 - (x + b)$. Of course, this would violate the security of the HSS scheme, implying that the discrete logarithm problem in $\mathbb{G}$ should be hard and A, B have to find other means to succeed.

[3] In all algorithms presented in this paper, the bulk of computation involves performing group operations, hence this is a reasonable complexity measure. Alternatively, the parameter T may bound the complexity of A, B in some reasonable computational model.

[4] We note that in the applications of [4–6], the distribution of $b \in [-M, M]$ is arbitrary. However (as we show in Lemma 12), our choice to define and analyze DDL for the uniform distribution of $b \in [-M, M]$ is technically justified since the uniform distribution is the hardest for DDL: algorithms for A, B that solve DDL with an error probability δ for the uniform distribution, also solve DDL with an error probability $O(\delta)$ for any distribution of $b \in [-M, M]$.

Our Goals The goal of this paper is to devise algorithms for A and B (i.e., a DDL protocol) that maximize their success probability (taken over the randomness of x, b), or equivalently, minimize their error probability δ given T. Our point of reference is the DDL protocol of [6] (which is a refined version of the original DDL protocol [5]) that achieves a linear tradeoff between the parameter T and error probability δ. More precisely, given that A, B are allowed T group operations, the DDL error probability is roughly M/T. In fact, there are several closely related protocols devised in [4–6] which give similar linear tradeoffs between the parameter T and error probability δ.

Yet another goal of this paper is to better understand the limitations of DDL protocols. More specifically, we aim to prove lower bounds on the error probability of DDL protocols by reducing a well-studied computational problem on groups to DDL. In particular, we are interested in the *discrete log in an interval (DLI)* problem, where the input consists of a group element in a known interval of length R and the goal is to compute its discrete log.

DLI has been the subject of intensive study in cryptanalysis and the best known algorithms for it are adaptations of the classical baby-step giant-step algorithm and the memory-efficient variant of Pollard [17] (see [12, 18] for additional extensions). These algorithms are based on collision finding and have complexity of about $\sqrt{R}$. They are the best known in concrete prime-order group families (in which discrete log is hard) up to large values of the interval R. In particular, for elliptic curve groups, the best known DLI algorithm has complexity of about $\sqrt{R}$ where R is as large as the size of the group N (which gives the standard discrete logarithm problem). For some other groups (such as prime order subgroups of $\mathbb{Z}_p^*$), the best known complexity is about $\sqrt{R}$, where R can be up to subexponential in $\log N$ (as discrete log can be solved in subexponential complexity in these groups [14, 15]). We note that besides its relevance in cryptanalysis, DLI is solved as part of the decryption process of some cryptosystems (notably in the cryptosystem by Boneh, Goh and Nissim [3]).

An alternative approach to establishing error probability lower bounds for DDL is to use the generic group model (GGM), introduced by Shoup [20]. In GGM, an algorithm is not allowed direct access to the bit representation of the group elements, but can only obtain randomized encodings of the elements, available via oracle queries. The generic group model is a standard model for proving computational lower bounds on certain (presumably hard) problems on groups and thus establishing confidence in their hardness. Although the bounds obtained in GGM are relevant to a restricted class of algorithms, it is essentially the only model in which meaningful lower bounds are known for some computational problems on groups (such as discrete log). Moreover, for several problems (such as discrete log computation in some elliptic curve groups), generic algorithms are essentially the best algorithms known. The downside of this alternative proof approach is that it does not directly relate DDL to any hard problem in a group family, but rather establishes a lower bound proof in an abstract model.

Our Contribution The main result of this work is closing the gap for DDL in many concrete group families by presenting upper and lower bounds that are tight (within a constant factor) based on the hardness of DLI in these families. We first develop an improved DDL protocol that is applicable in any group $\mathbb{G}$ and achieves a quadratic tradeoff between the parameter T and the error probability, namely $\delta = O(M/T^2)$. This is a substantial improvement over the linear tradeoff $\delta = O(M/T)$ obtained in [4–6]. Therefore, when executing Eval on an RMS program P with multiplicative complexity S, one can set $T = O(S^{1/2})$ to obtain $\delta = O(1/S)$ and the total running time is reduced from $O(S^2)$ in [4–6] to $O(S^{3/2})$. This result directly improves upon the computational complexity of some of the HSS applications given in [4–6]. For example, in private information retrieval [9] (PIR), a client privately searches a database distributed among several servers for the existence of a document satisfying a predicate P. The 1-round 2-server PIR scheme of [5] supports general searches expressed as branching programs of size S applied to each document. The computational complexity per document in the scheme of Boyle et al. is $O(S^2)$ and our result reduces this complexity to $O(S^{3/2})$.

On the practical side, we fully verified our protocol by extensive experiments. We further consider the optimizations implemented in [4, 6] for DDL, which have a significant impact on its concrete running time. Although applying these optimizations to our DDL protocol is not straightforward, we show how to tweak our protocol such that it can benefit from these optimizations as well. Overall, we believe that our new DDL protocol will allow us to improve upon the most efficient HSS implementation [4] by several orders of magnitude and render it practical for new applications.

Our DDL protocol uses a new type of (pseudo) random walk composed of several iterations. Each one of these iterations resembles Pollard's "kangaroo" random walk algorithm for solving DLI using limited memory [17]. However DDL is different from DLI as the parties cannot communicate and seek to minimize their error probability (rather than make it constant). This leads to a more complex iterative algorithm, where the parties carefully distribute their time complexity T among several random walks iterations. These iterations use increasingly longer step lengths that gradually reduce the error probability towards $O(M/T^2)$.

The new random walk maximizes the probability that parties with close inputs agree (or synchronize) on a common output without communicating. We believe that this random walk is of independent interest and will find additional applications beyond homomorphic secret sharing schemes and cryptography in general.

After presenting our DDL protocol, we focus on lower bounds and show that any DDL protocol for a family of groups must have error probability of $\delta = \Omega(M/T^2)$, unless DLI (with interval of length R) can be solved in time $T' \approx T = o(\sqrt{R})$ in this family. This is currently not achievable for small (polynomial) T in standard cryptographic groups (for which the group-based HSS scheme is deemed to be secure).

Finally, we analyze DDL protocols in the generic group model. In this model, our DDL protocol is adaptive, as the oracle queries of A and B depend on the answers to their previous queries. This stands in contrast to the protocols of [4–6] in GGM, whose oracle queries are fixed in advance (or selected with high probability from a pre-fixed set of size $O(T)$). It is therefore natural to ask whether adaptivity is necessary to obtain optimal DDL protocols in GGM. Interestingly, we prove that the answer is positive. In fact, we show that the linear tradeoff obtained in [4–6] is essentially the best possible for non-adaptive DDL protocols in GGM.

Paper Organization The rest of the paper is organized as follows. We describe preliminaries in Section 2 and present an overview of our new protocol and related work in Section 3. Our new DDL protocol is analyzed in Section 4. We prove lower bounds on the DDL error probability in concrete group families in Section 5 and finally prove lower bounds on non-adaptive algorithms in GGM in Section 6.

2 Preliminaries

In this section we describe the preliminaries required for this work. First we introduce notation that we use throughout the paper and then we present and analyze the DDL algorithm of [5], which will serve as a basis for our algorithms. For sake of completeness, we give a brief description of the group-based homomorphic secret sharing scheme of [5] in Appendix A.

2.1 Notation for the Distributed Discrete Log Problem

Recall that the parties A and B successfully solve the DDL instance if $A(g^x) - B(g^{x+b}) = b$. To simplify our notation, we typically do not explicitly write the parameters $\mathbb{G}, g, N, M, T$ in the description of A, B, although some of them will appear in the analysis. We are interested in the success (or error) probability of A and B, taken over the randomness of x, b (and possibly over the randomness of A, B). We denote by $\mathbf{err}(A, B, x, b, T)$ the error event $A(g^x) - B(g^{x+b}) \neq b$, and by $\Pr_{\mathbf{err}}(A, B, [M_1, M_2], T)$ its probability $\Pr_{x,b}[\mathbf{err}(A, B, x, b, T)]$, where $x \in \mathbb{Z}_N$ and $b \in [M_1, M_2]$ are uniform (typically, we are interested in $M_2 = -M_1 = M$). We also denote by $\mathbf{suc}(A, B, x, b, T)$ the complementary success event $A(g^x) - B(g^{x+b}) = b$.

When both parties perform the same algorithm A, we shorten the notation into $\mathbf{err}(A, x, b, T)$, $\Pr_{\mathbf{err}}(A, [M_1, M_2], T)$, and $\mathbf{suc}(A, x, b, T)$, respectively. If the parameters A, B, x, b, T are apparent from the context, we sometimes use $\mathbf{err}$ and $\mathbf{suc}$ instead of $\mathbf{err}(A, B, x, b, T)$ and $\mathbf{suc}(A, B, x, b, T)$, respectively. As mentioned above, A and B can be randomized algorithms and in this case the success (and error) probabilities are taken over their randomness as well. However, to simplify our notation we will typically not refer to this randomness explicitly.

We note that the DDL problem considered in [4–6] is slightly different, as A, B are allowed to perform up to T group operations in expectation. In this alternative definition, one can construct DDL protocols that are more efficient than ours by a small constant factor, while our lower bounds remain the same (again, up to a constant factor).

In the description and analysis of the DDL algorithms, we make frequent use of group elements of the form g^{x+j}. For sake of simplicity, we denote $g_j := g^{x+j}$. In addition, we usually assume $b \geq 0$, as otherwise we can simply exchange the names of the parties A and B when they use the same algorithm. Finally, we refer to a group operation whose output is h as a query to h.

2.2 The Basic DDL Algorithm

Let $\phi : \mathbb{G} \to [0, N-1]$ be a pseudo-random function (PRF) that maps group elements to integers. Our protocols evaluate ϕ on $O(T)$ group elements for $T \ll N^{1/2}$. We assume throughout the analysis that ϕ behaves as a truly random permutation on the evaluated group elements, and in particular, we do not encounter collisions in ϕ (i.e., for arbitrary $h \neq h'$, $\phi(h) \neq \phi(h')$). Our probabilistic calculations are taken over the choice of ϕ, even though we do not indicate this explicitly for simplicity.[5]

We describe the min-based DDL algorithm of [5] in Algorithm 1 and refer to it as the basic DDL algorithm. The algorithm is executed by both A and B. When applied to $g_0 = g^x$, the algorithm scans the T values $g_0, g_1, \ldots, g_{T-1}$ and chooses the index i_{min} for which $\phi(g_i)$ is minimal. The output of the algorithm is $\text{Basic}_T(g^x) = (i_{min}, g_{min})$. Note that the algorithm depends also on $\mathbb{G}, g$; however, we do not mention them explicitly in the notation. Furthermore, the output g_{min} will only be relevant later, when we use this algorithm as a subprocedure. For the sake of analysis, we slightly abuse notation below and refer to i_{min} as the (only) output of $\text{Basic}_T(g^x)$.

The motivation behind the algorithm is apparent: if party A applies $\text{Basic}_T(g^x)$ and party B applies $\text{Basic}_T(g^{x+b})$, where $b \ll T$, then the lists of values scanned by the two algorithms (i.e., $g_0, g_1, \ldots, g_{T-1}$ and $g_b, g_{b+1}, \ldots, g_{b+T-1}$) contain many common values, and thus, with a high probability the minimum is one of the common values, resulting in success of the algorithm.

2.3 Analysis of the Basic DDL Algorithm

Error probability. The following lemma calculates the error probability of the basic DDL algorithm, as a function of $|b|$ and T.

[5] The function ϕ (and additional pseudo-random functions defined in this paper) can be implemented by a keyed MAC, where the key is pre-distributed to A and B. Thus, our probabilistic calculations should be formally taken over the choice of the key. They should include an error term that accounts for the distinguishing advantage of an efficient adversary (A or B in our case) that attempts to distinguish the PRF from a truly random permutation. However, for an appropriately chosen PRF, the distinguishing advantage is negligible and we ignore it for the sake of simplicity.

Algorithm 1: $\text{Basic}_T(g^x)$

```
1  begin
2      h' ← g^x, i ← 0, min ← ∞;
3      while i < T do
4          y ← φ(h');
5          if y < min then
6              g_min ← h';
7              i_min ← i, min ← y;
8          end
9          h' ← h' · g;
10         i ← i + 1;
11     end
12     Output (i_min, g_min);
13 end
```

Lemma 1. *The error probability of the basic DDL algorithm is*

$$\Pr_x[\mathbf{err}(\text{Basic}_T, x, b, T)] = \Pr[(\text{Basic}_T(g^x) - \text{Basic}_T(g^{x+b}) \neq b)] = \frac{2|b|}{|b| + T}.$$

Proof. We assume $b \geq 0$, as otherwise we exchange the names of A and B. Since both A and B use Algorithm 1, then A computes the function ϕ on $g_0, g_1, \ldots, g_{T-1}$, while B computes this function on $g_b, g_{b+1}, \ldots, g_{b+T-1}$. If the minimum value of ϕ for each party is obtained on an element $g_{min} = g^x \cdot g^{i_{min}}$ which is queried by both, then we have $\text{Basic}_T(g^x) = i_{min}$ and $\text{Basic}_T(g^{x+b}) = i_{min} - b$, implying that $\text{Basic}_T(g^x) - \text{Basic}_T(g^{x+b}) = b$ and the parties are successful. Similarly, they fail when the minimal value of ϕ on the elements $g_0, g_1, \ldots, g_{b+T-1}$ is obtained on an element computed only by one party, namely on one of the $2b$ elements $g^x \cdot g^i$ for $0 \leq i < b$ or $T \leq i < b+T$. Assuming that the output of ϕ on each element is uniform and the outputs are distinct, this occurs with probability $2b/(b+T)$. Hence $\Pr_x[\mathbf{err}(\text{Basic}_T, x, b, T)] = 2|b|/(|b| + T)$, as asserted. ■

The output difference in case of failure. An important quantity that plays a role in our improved protocol is the output difference of the parties in case they fail to synchronize on the same element g_{min} (i.e., their output difference is not b). The following lemma calculates the expectation of this difference, as function of $|b|$ and T.

Lemma 2.

$$\mathbb{E}\left[\left|\text{Basic}_T(g^x) - \text{Basic}_T(g^{x+b}) - b\right| \,\middle|\, \mathbf{err}\right] = \frac{|b| + T}{2}.$$

Proof. We assume $b \geq 0$. As written above, A computes the function ϕ on $g_0, \ldots, g_{T-1}$, and B computes this function on $g_b, \ldots, g_{b+T-1}$. The ordering of

the values $\phi(g_0), \ldots, \phi(g_{b+T-1})$ is uniform, and so the permutation π satisfying $\phi(g_{\pi(0)}) < \phi(g_{\pi(1)}) < \ldots < \phi(g_{\pi(b+T-1)})$ is uniformly random in the permutation group of $\{0, 1, \ldots, b+T-1\}$. The event **err** is equivalent to the event $\pi(0) \notin [b, T-1]$. Without loss of generality let us restrict ourselves to the event $\pi(0) < b$, i.e. A encounters the minimal group element, and B does not; the other possibility $\pi(0) \geq T$ is symmetric with respect to reflection. Clearly, $\pi(0)$, which equals $\mathrm{Basic}_T(g^x)$, is uniformly random in $[0, b-1]$. Moreover, $\mathrm{Basic}_T(g^{x+b}) + b$ is $\pi\left(\min\left\{i \,|\, \pi(i) > b\right\}\right)$ which uniformly distributes in $[b, b+T-1]$. Hence the expected final distance between the parties is $(2b+T+1)/2-(b+1)/2 = (b+T)/2$. ■

3 Overview of our New Protocol and Related Work

3.1 The New DDL Protocol

For the sake of simplicity, we assume in this overview that $M = 1$, hence $|b| \leq 1$. The starting point of our new DDL protocol is Algorithm 1. It makes T queries (i.e., group operations) and fails with probability of roughly $2/T$ according to Lemma 1. Let us assume that we run this algorithm with only $T/2$ queries, which increases the error probability by a factor of 2 to about $4/T$. On the other hand, we still have a budget of $T/2$ queries and we can exploit them to reduce the error probability. Interestingly, simply proceeding to calculate more consecutive group elements is not an optimal way to exploit the remaining budget.

After the first $T/2$ queries, we say that A (or B) is placed at group element g^y if $\phi(g^y)$ is the minimal value in its computed set of size $T/2$. Assume that A and B fail to synchronize on the same group element after the first $T/2$ queries (which occurs with probability of roughly $4/T$). Then, by Lemma 2, A and B are placed at elements which are at distance of about $T/4$, i.e., if A is placed at g^y and B is placed at g^z, then $|y-z| \approx T/4$. Our main idea is to use a somewhat different procedure in order to try to synchronize A and B in case they fail to do so after the first $T/2$ queries, while keeping A and B synchronized if they already are.

The next procedure employed by both A and B is a (pseudo) random walk starting from their initial position, whose step length is uniformly distributed in $[1, L-1]$, where $L \approx \sqrt{T}$. The step length at group element g^y is determined by $\psi_{L-1}(g^y)$, where ψ_{L-1} is a pseudo-random function independent of ϕ that outputs a uniform integer in $[1, L-1]$.[6] Assume that after the first $T/2$ queries, B is placed at distance of about $T/4$ in front of A. Then A will pass B's initial position after about $\sqrt{T}/2$ steps and simple probabilistic analysis shows that A will land on one of B's steps after an additional expected number of about $\sqrt{T}/2$ steps. From this point, the walks coincide for the remaining $T/2 - \sqrt{T}$ steps, and B makes about $\sqrt{T}/2$ extra steps. Similarly to Algorithm 1, each party outputs the offset of the minimal $\phi(g^y)$ value visited during its walk. Since

[6] Our analysis assumes that ψ_{L-1} is a truly random function and our probabilistic calculations are taken over the choice of ψ_{L-1}.

both A and B use the same deterministic algorithm, they remain synchronized if they already are at the beginning of the walks. On the other hand, if they are not initially synchronized, their walks are expected to coincide on $T/2 - \sqrt{T}$ elements, and hence the probability that they remain unsynchronized is roughly $\sqrt{T}/(T/2) = 2 \cdot T^{-1/2}$. Thus, the error probability at this stage is about $4 \cdot T^{-1} \cdot 2 \cdot T^{-1/2} = 8 \cdot T^{-3/2}$, which already significantly improves upon the $2 \cdot T^{-1}$ error probability of Algorithm 1 for large T.

However, we can still do better. For the sake of simplicity, let us completely ignore constant factors in rest of this rough analysis. Note that we may reserve an additional number of $O(T)$ queries to be used in another random walk by shortening the first two random walks, without affecting the failure probability significantly. Hence, assume that the parties fail to synchronize after the random walk (which occurs with probability of about $T^{-3/2}$) and that we still have enough available queries for another random walk with $O(T)$ steps. Since each party covers a distance of about $T^{3/2}$ during its walk, then the expected distance between the parties in case of failure is roughly $T^{3/2}$. We can now perform another random walk with expected step length of $T^{3/4}$ (hence the walks are expected to coincide after about $T^{3/4}$ steps), reducing the error probability to about $T^{-3/2} \cdot (T^{3/4} \cdot T^{-1}) = T^{-7/4}$. This further increases the expected distance between A and B in case of failure to approximately $T^{7/4}$. We continue executing random walk iterations with a carefully chosen step length (distributing a budget of $O(T)$ queries among them). After i random walk iterations, the error probability is reduced to about $T^{-2+2^{-i}}$ (and the expected distance between the parties is roughly $T^{2-2^{-i}}$). Choosing $i \approx \log\log T$ gives an optimal error probability of about $T^{-2+1/\log T} = O(T^{-2})$.

Our new DDL protocol is presented in algorithms 2 and 3. Algorithm 2 describes a single iteration of the random walk, parameterized by (L, T) which determine the maximal step length and the number of steps, respectively.[7] Algorithm 3 describes the full protocol which is composed of application of the basic DDL algorithm (using $t_0 < T$ queries, reserving queries for the subsequent random walks), and then I additional random walks, where the i'th random walk is parameterized by (L_i, t_i) which determine its maximal step length and number of steps. Between each two iterations in Step 6, both parties are moved forward by a large (deterministic) number of steps, in order to guarantee independence between the iterations (the computation time used to perform these calculations is negligible compared to T). We are free to choose the parameters $I, \{L_i, t_i\}$, as long as $\sum_{i=0}^{I} t_i = T$ is satisfied.

The very rough analysis presented above assumes that we have about T queries in each of the $\log\log T$ iterations, whereas we are only allowed T queries overall. Moreover, it does not accurately calculate the error probability and the distance between the parties in case of failure in each iteration. Taking all

[7] We assume that the algorithm uses a table containing the pre-computed values $g, g^2, \ldots, g^{L-1}$. Otherwise, it has to compute $g^{z_{i+1}}$ on-the-fly in Step 10, which results in a multiplicative penalty of $O(\log(T))$ on the number of group operations. Of course, it is also possible to obtain a time-memory tradeoff here.

Algorithm 2: $\mathrm{RandW}_{L,T}(h)$

```
begin
    h' ← h, i ← 0, min ← ∞, d_0 ← 0;
    while i < T do
        y ← φ(h');
        if y < min then
            h_min ← h';
            d_min ← d_i, min ← y;
        end
        z_{i+1} ← ψ_{L-1}(h');
        h' ← h' · g^{z_{i+1}};
        d_{i+1} ← d_i + z_{i+1};
        i ← i + 1;
    end
    Output (d_min, h_min);
end
```

Algorithm 3: $\mathrm{IteratedRandW}_{I,t_0,\{(L_i,t_i)_{i=1}^{I}\}}(h)$

```
begin
    (c_0, h_0) ← Basic_{t_0}(h);
    p_0 ← c_0;
    i ← 1;
    while i ≤ I do
        h'_{i-1} ← h_{i-1} · g^{Σ_{j<i} t_j L_j};
        (c_i, h_i) ← RandW_{L_i,t_i}(h'_{i-1});
        p_i ← p_{i-1} + c_i;
        i ← i + 1;
    end
    Output p_I;
end
```

of these into account in an accurate analysis results in an error probability of $\Omega(\log T \cdot T^{-2})$. Surprisingly, we can still achieve an error probability of $O(T^{-2})$. This is done by a fine tuning of the parameters which distribute the number of queries among the iterations and select the step length of each random walk. In particular, it is not optimal to independently optimize the step length of each iteration and one has to analyze the subtle dependencies between the iterations in order to achieve an error probability of $O(T^{-2})$.

As the fine tuning of the parameters is rather involved, in addition to the theoretical analysis we verified the failure probability by extensive experiments.

3.2 Related Work

The most closely related work to our DDL algorithm is Pollard's "kangaroo" method for solving the discrete logarithm problem in an interval (DLI) using limited memory (see Galbraith et al. [17] and Pollard [12, 18] for further analysis and extensions). The kangaroo method launches two random walks (kangaroos), one from the input $h = g^x$ (where x the unknown discrete log) and one from g^y, where y is a known value in an interval of a fixed size R around x. The algorithm is optimized such that the walks meet at a "distinguished point", which reveals x. The kangaroo method thus resembles a single random walk iteration of our DDL algorithm.

On the other hand, there are fundamental differences between the standard DLI and DDL. These differences result in the iterative structure of our algorithm that differs from Pollard's method. First, in contrast to the DLI problem, in DDL A and B cannot communicate and never know if they succeed to synchronize. Hence, the parties cannot abort the computation at any time. Second, the goal in DDL is to minimize the error probability, whereas achieving a constant error probability (as in standard DLI) is unsatisfactory. To demonstrate the effect of these differences, observe that solving the discrete log problem in an interval of size 3 can be trivially done with probability 1 using 3 group operations. On the other hand, our algorithm for solving DDL for $M = 1$ is much more complicated and achieves an error probability of about T^{-2} using T group operations (which is essentially optimal for many concrete group families).

Yet another difference between DLI and DDL is that in DLI the boundaries of the interval of the input h are known, whereas in DDL the input of each party is completely uniform. The knowledge of the interval boundaries in DLI allows to shift it to the origin (using the self-reducibility property of discrete log) and efficiently use preprocessing (with a limited amount of storage) to speed up the online computation, as shown by Bernstein and Lange [2]. It is not clear how to efficiently exploit preprocessing in DDL.

4 The New Distributed Discrete Log Protocol

In this section we study our new DDL protocol in more detail. In Section 4.1 we focus on a single iteration of our DDL protocol (i.e., a single random walk iteration) and analyze its failure probability and the expected distance between its outputs in case of a failure. In Section 4.2 we analyze the complete protocol. The experimental verification of the protocol is presented in Section 4.3. We also describe some practical considerations regarding the protocol in Appendix D.

4.1 A Single Iteration of our DDL Protocol – the Random Walk DDL Algorithm

Recall that in Algorithm 2, applied with parameters (L, T), both parties perform a random walk of T steps of the form $g^y \to g^{y+a_i}$, where the length a_i of each step

is determined by a (pseudo) random function $\psi_{L-1} : \mathbb{G} \to \{1, 2, \ldots, L-1\}$ which guarantees that the step length is uniformly distributed in the range $[1, L-1]$. Each party then chooses, among the elements of $\mathbb{G}$ visited by its walk, the element h_{min} for which $\phi(h_{min})$ is minimal (as in the basic DDL algorithm).[8]

In Algorithm 3, once the parties synchronize in a given stage (i.e., application of Algorithm 2), they remain synchronized in the subsequent ones as each stage is deterministic. Hence, in each stage, Algorithm 3 tries to minimize the failure probability, given that in the previous stage, the parties failed to synchronize. This error probability depends on the 'initial' distance between the parties, at the start of the stage. Hence, in order to analyze the total error probability of the whole Algorithm 3, we should estimate the error probability of each stage (given the distance at its beginning), together with the (expected) distance between the parties in case of failure to synchronize.

Additional Notation In our analysis we use some auxiliary notation. Without loss of generality, we assume that $b \geq 0$ (namely, B is located at distance b in front of A). We let S_A be the number of steps of A until its walk lands on an element visited by B (i.e., the number of queries made by A strictly before the first element of A that is included in B's path). If this never occurs, we let $S_A = T$. Similarly, we define S_B as the number of steps of B until its walk lands on an element visited by A. Clearly, the walks of A and B coincide for $T - \max(S_A, S_B)$ steps.

We define U_A as the number of steps A performs until it is within reach of a single step from the starting point of B. Namely, $U_A = \min\{i \mid d_i^A > b - L\}$, where d_i^A is the variable d_i in Algorithm 2 applied by A. In addition, we let V_A, V_B denote the numbers of steps performed by A and B, respectively, starting from the point where A is within reach of a single step from the starting point of B, until the walks collide or one of them ends. Furthermore, we denote $V_m = \max\{V_A, V_B\}$. Notice that $S_A = U_A + V_A$ and $S_B = V_B$, and hence

$$\max(S_A, S_B) \leq U_A + V_m. \tag{1}$$

Below, we evaluate the expectations of the random variables U_A, V_m in order to bound the error probability of synchronization based on Algorithm 2.

Finally, while $\mathrm{RandW}_{L,T}(h)$ has two outputs, we slightly abuse notation and refer to d_{min} as the (only) output of $\mathrm{RandW}_{L,T}(h)$ since only this output is relevant for this analysis.

The Failure Probability of Algorithm 2 First, we bound the expected number of steps performed by A until it reaches the starting point of B.

Lemma 3. $\mathbb{E}[U_A] < \frac{2b}{L}$.

[8] We assume in our analysis that during the application of the whole protocol by a single party, each function ϕ, ψ_{L-1} is not evaluated twice on the same input. These constrains are satisfied since $|\mathbb{G}| = N$ is much larger than T (e.g., $|\mathbb{G}| > cT^2$ for a sufficiently large constant c).

Proof. By the definition of U_A, we have $d^A_{U_A} < b$. Consider the martingale $d'_i = d^A_i - iL/2$ (which is indeed a martingale, as $\psi_{L-1}(h')$ computed in the algorithm are independent and have expectation $L/2$). The classical *Doob's martingale theorem* yields

$$0 = d'_0 = \mathbb{E}[d'_{U_A}] = \mathbb{E}[d^A_{U_A}] - L\,\mathbb{E}[U_A]/2.$$

As $d_{U_A} < b$, we deduce $\mathbb{E}[U_A] < 2b/L$. ■

Our next lemma bounds the expectation of $S_A + S_B$, that is, the total number of steps performed by the two walks together before they meet.

Lemma 4. *Suppose the initial distance between the parties, b, satisfies $0 < b < L$. Then $\mathbb{E}[S_A + S_B] \leq L - 1$.*

Proof. For ease of computation, we do not trim S_A and S_B with T. Of course, this can only make the upper bound larger. One easily sees that $\mathbb{E}[S_A + S_B]$ is finite and depends only on b (and the parameter L). Write E_b for this expectation. We have $\mathrm{E}_0 = 0$, and by dividing into cases according to the result of the first step of A, we obtain

$$\mathrm{E}_b = 1 + \frac{1}{L-1}\left(\mathrm{E}_{b-1} + \mathrm{E}_{b-2} + \ldots + \mathrm{E}_1 + \mathrm{E}_0 + \mathrm{E}_1 + \ldots + \mathrm{E}_{L-1-b}\right).$$

A valid solution for this system of linear equations is $\mathrm{E}_b = L-1$ for all $0 < b < L$. This is actually the only solution, since the matrix corresponding to this system is strictly diagonally dominant, and thus is invertible by the *Levy-Desplanques theorem*. Therefore, $\mathbb{E}[S_A + S_B] = L - 1$, independently of b. ■

The next lemma bounds the maximum between the numbers of steps performed by A and B between the time A "almost" reached the starting point of B and the meeting of the walks.

Lemma 5. $\mathbb{E}[V_m] \leq \frac{L-1}{2} + \sqrt{8(L-1)}$.

Proof. The proof consists of several steps.

Step 1. We write $V_m = \max\{V_A, V_B\} = \frac{V_A+V_B}{2} + \frac{|V_A-V_B|}{2}$. Lemma 4 upper bounds the first summand by $(L-1)/2$.

Step 2. We now continue and bound $\mathbb{E}\,|V_A - V_B|$. From Cauchy-Schwarz it suffices to bound $\sqrt{\mathbb{E}[(V_A - V_B)^2]}$. To analyze such terms, we define

$$F_b = \mathbb{E}\left[(V_A - V_B)^2\right].$$

Note that we write F_b in order to emphasize that F_b depends on the initial distance between the parties. As $F_b = F_{-b}$, we may assume $b \geq 0$, i.e. A is behind B. To compute F_b, we let n be the minimal time with $d^A_n \geq b$. Note that n is a random variable and not a constant. We denote $(V'_A, V'_B) = (V_A - n, V_B)$, which corresponds to the walk after letting A perform n steps. We thus have

$$F_b = \mathbb{E}[(n + V'_A - V'_B)^2] = \mathbb{E}[n^2] + \mathbb{E}[(V'_A - V'_B)^2] + 2\,\mathbb{E}[(V'_A - V'_B)n]. \qquad (2)$$

We now wish to understand each of the terms in the RHS (right-hand side) of (2).

Step 2(a). We start with $\mathbb{E}[n^2]$. From Lemma 3, $\mathbb{E}[n] \le 2(b+L-1)/L$. Additionally, consider the martingale

$$S_k = \sum_{i=0}^{k} (d_k^A - d_i^A) - \frac{L}{2}\binom{k+1}{2}.$$

To see this is a martingale, observe that

$$S_{k+1} - S_k = (k+1)(d_{k+1}^A - d_k^A) - (k+1)L/2,$$

which has expectation 0 regardless of the values of $S_0, \ldots, S_k$. Hence, by Doob's martingale theorem, we have $\mathbb{E}[S_n] = 0$, where n is the stopping time defined above. Thus,

$$\frac{L}{2}\,\mathbb{E}\left[\binom{n+1}{2}\right] = \mathbb{E}\left[\sum_{i=0}^{n}(d_n^A - d_i^A)\right] \le \mathbb{E}[n](L+b-2) \le \frac{2(b+L-1)^2}{L}.$$

Therefore,

$$\mathbb{E}[n^2] \le 8\left(\frac{L+b-1}{L}\right)^2.$$

Step 2(b). We now reason about $\mathbb{E}[(V_A' - V_B')^2]$. Notice $V_A' - V_B'$ equals 0 with probability $\ge 1/(L-1)$, since in the last step before $d_n^A \ge b$ we had a probability of $1/(L-1)$ to end up with $d_n^A = b$. Hence, $\mathbb{E}[(V_A' - V_B')^2]$ is a positive weighted sum of some $(F_{b'})$'s, all satisfying $b' \le L-2$, with total weight $\le 1 - 1/(L-1)$.

Step 2(c). Another crucial component is that $\mathbb{E}[(V_A' - V_B')n] \le 0$. Actually, for any fixed η we have $\mathbb{E}\left[V_A' - V_B' \,|\, n = \eta\right] \le 0$, which implies the former assertion. This is intuitive, since once A is ahead of B, on average B will have to perform more steps than A in order to catch up A. Formally, one can use once again a martingale argument and obtain

$$\mathbb{E}[d^A_{V_A'+n} - V_A' L/2] \underbrace{=}_{\text{martingale}} d_n^A \ge b = d_0^B + b \underbrace{=}_{\text{martingale}} \mathbb{E}[d^B_{V_B'} - V_B' L/2] + b.$$

Observing that $d^A_{V_A'+n} = d^B_{V_B'} + b$ we find that $\mathbb{E}[V_A'] \le \mathbb{E}[V_B']$ as desired.

Step 3. So far, we obtained some inequalities of the form

$$F_b \le 8\left(\frac{L+b-1}{L}\right)^2 + (1 - 1/(L-1))\,\mathop{\mathbb{E}}_{b'}[F_{b'}],$$

where the involved b' are $\le L-2$. Letting $m = \max_{b' \le L-2}\{F_{b'}\}$, we deduce $m \le 32 + (1-1/(L-1))m$, and thus, $m \le 32(L-1)$.

Step 4. Putting all together, we obtain

$$V_m \leq \frac{L-1+\sqrt{F_b}}{2} \leq \frac{L-1}{2} + \sqrt{8(L-1)}, \tag{3}$$

as asserted. ■

Now we are ready to estimate the failure probability of Algorithm 2.

Lemma 6. *Let $R = 2b/L + L/2 + \sqrt{8L}$ for $0 < b < L$. The error probability of the random walk DDL algorithm satisfies*

$$\begin{aligned}\Pr_x[\mathbf{err}(\mathrm{RandW}, x, b, T)] &= \Pr[(\mathrm{RandW}_{L,T}(g^x) - \mathrm{RandW}_{L,T}(g^{x+b}) \neq b)] \\ &\leq \frac{2R}{T+R}.\end{aligned}$$

Proof. The walks of A and B coincide for $T - \max(S_A, S_B)$ steps. Notice that we have,

$$\mathbb{E}[\max(S_A, S_B)] \leq \mathbb{E}[U_A + V_m] \leq \frac{L}{2} + \frac{2b}{L} + \sqrt{8L} = R, \tag{4}$$

where the first inequality uses (1) and the second inequality uses Lemmas 3 and 5. Similarly to the basic DDL algorithm (Lemma 1), the error probability (assuming that the output of ϕ on each element is uniformly random) is

$$\begin{aligned}\Pr_x[\mathbf{err}(\mathrm{RandW}, x, b, T)] &= \mathbb{E}[2\max(S_A, S_B)/(T + \max(S_A, S_B))] \\ &\leq \frac{\mathbb{E}[2\max(S_A, S_B)]}{T + \mathbb{E}[\max(S_A, S_B)]} \leq \frac{2R}{T+R},\end{aligned}$$

where the first inequality is Jensen's inequality applied to the increasing concave function $x \mapsto 2x/(T+x)$ in the domain $x > 0$, and the second inequality uses the monotonicity of the function $x \mapsto 2x/(T+x)$ and Equation (4). ■

The Output Difference in Case of Failure Similarly to Lemma 2 which bounded the expected difference of outputs in case of failure for the basic DDL algorithm, we bound the analogous quantity for Algorithm 2. In order to achieve this result, we need the following corollary of the classical *Azuma's martingale inequality.*

Theorem 1 (Azuma's inequality). *Let $X_0, X_1, \ldots, X_n$ be a martingale with $|X_i - X_{i-1}| \leq V$. Then for any $t \geq 0$,*

$$\Pr\left[|X_n - X_0| \geq V \cdot t\sqrt{n}\right] \leq 2\exp(-t^2/2).$$

Lemma 7. *Let $X_1, \ldots, X_n$ be independent random variables with $|X_i - \mathbb{E}[X_i]| \leq V$ and let $\mathcal{E}$ be an event. Then*

$$\mathbb{E}\left[\max_{k=1}^{n}\left|\sum_{i\leq k}(X_i - \mathbb{E}[X_i])\right| \middle| \mathcal{E}\right] \leq V\sqrt{8n\log(2/\Pr[\mathcal{E}])}.$$

To understand the intuition behind the lemma, consider the sum of independent $\{1,-1\}$ random variables $\{X_i\}_{i=1}^n$. By Chernoff's inequality, $\Pr[|\sum X_i| > t\sqrt{n}] < e^{-t^2/2}$. However, if we condition on the event $\mathcal{E} = \{\sum X_i = n\}$, then we have $\Pr[|\sum X_i| > t\sqrt{n}|\mathcal{E}] = 1$ for all $t < \sqrt{n}$. On the other hand, the probability of the event $\mathcal{E}$ is extremely small. We claim that if one is allowed to condition only on events with not-so-small probability, then the expectation of the maximum between the sums $|\sum_{i=1}^k X_i|$ is not much larger than $O(\sqrt{n})$ (which follows from Chernoff's inequality in the unconditioned case), and in particular, $\Pr[|\sum X_i| > t\sqrt{n}|\mathcal{E}]$ is negligible for any "large" t. Our lemma is the martingale version of this intuition.

Proof (of Lemma 7). Let $t \geq 0$. Consider the martingale $Y_0, Y_1, \ldots, Y_n$ defined by $Y_0 = 0$ and

$$Y_k = Y_{k-1} + \begin{cases} X_k - \mathbb{E}[X_k], & \text{if } |Y_{k-1}| < t \\ 0, & \text{otherwise} \end{cases}.$$

Write $M = \max_{k=1}^n \left|\sum_{i\leq k}(X_i - \mathbb{E}[X_i])\right|$. Notice $|Y_n| \geq t$ if and only if $M \geq t$. Using Azuma's inequality for the martingale $\{Y_k\}_k$, we deduce

$$\Pr[M \geq t] \leq 2\exp(-t^2/(2nV^2)).$$

Using Fubini's theorem and a partition into "small" and "large" values of t, we obtain that for any $r > 0$,

$$\begin{aligned}
\mathbb{E}[M \mid \mathcal{E}]\Pr[\mathcal{E}] &= \int_0^\infty \Pr[(M > t) \wedge \mathcal{E}]\,\mathrm{dt} \\
&\leq \int_0^\infty \min\left\{\Pr[\mathcal{E}], 2\exp(-t^2/(2nV^2))\right\}\mathrm{dt} \\
&\leq r\Pr[\mathcal{E}] + 2\int_r^\infty \exp(-t^2/(2nV^2))\mathrm{dt} \\
&\leq r\Pr[\mathcal{E}] + \frac{2V^2 n \cdot \exp(-r^2/(2nV^2))}{r}.
\end{aligned}$$

Choosing $r = V\sqrt{2n\log(2/\Pr[\mathcal{E}])}$, we obtain

$$\mathbb{E}[M \mid \mathcal{E}]\Pr[\mathcal{E}] \leq 2r\Pr[\mathcal{E}] = \Pr[\mathcal{E}] \cdot 2V\sqrt{2n\log(2/\Pr[\mathcal{E}])}.$$

Dividing both sides by $\Pr[\mathcal{E}]$ gives the result. ■

Finally, we show that either the error probability is "very small" (and so there is no need to continue the random walk iterations), or we can bound the distance between the outputs in case of failure.

Lemma 8. *If* $\Pr_{\mathbf{err}}(\mathrm{RandW}_{L,T}, [1,1], T) \geq \epsilon$*, then for* $0 < b < L$ *and* $h_1 \in \mathbb{G}$*,*

$$\begin{aligned}
&\mathbb{E}\left[\left|\mathrm{RandW}_{L,T}(h_1) - \mathrm{RandW}_{L,T}(h_1 \cdot g^b) - b\right| \,\middle|\, \mathbf{err}\right] \\
&\qquad \leq b + \frac{TL}{4} + L\sqrt{32T\log(2/\epsilon)}.
\end{aligned} \tag{5}$$

Proof. The jumping patterns of the parties ($\psi_{L-1}(h')$) are independent of their choices of minimums h_{min} among the group elements they queried. Hence, similarly to Lemma 2, we note that the event **err** happens if and only if the minimal element in the union of the elements queried by A and B, was queried only by one of them. Since A and B query the same number of elements, when conditioning on **err**, both A and B have probability $1/2$ to query this minimal element. From now on we rename A, B so that A is the one who queries the minimal element (in case of a tie we randomly rename them). Notice that since b measures how much A is behind B, we might have changed b's sign. We observe that, even when conditioning on **err**, d^B_{min} uniformly distributes in $d^B_0, \ldots, d^B_{T-1}$, since, as mentioned, **err** is independent of the relative order of the non-minimum elements queried by the parties (and A queried the minimum element).

Let P_B be the point in B'th route which is closest to d^A_{min}, i.e. we choose $P_B \in \{d^B_0, \ldots, d^B_{T-1}\}$ which minimizes $|d^A_{min} - P_B - b|$. Notice the lemma asks to bound the expected value of $|d^A_{min} - d^B_{min} - b|$, given **err**. We simply bound

$$|d^A_{min} - d^B_{min} - b| \leq |d^A_{min} - P_B - b| + |P_B - d^B_{min}|, \tag{6}$$

and analyze the values on the right hand side of (6). First,

$$|d^A_{min} - P_B - b| \leq \max\{|b|, L/2, |d^A_{T-1} - d^B_{T-1} - b|\}. \tag{7}$$

We can verify (7) by case analysis:

- If $d^A_{min} \leq b$, we may take $P_B = d^B_0 = 0$.
- If $d^A_{min} \geq d^B_{T-1} + b$, we may take $P_B = d^B_0 = 0$.
- Otherwise, $d^B_0 + b \leq d^A_{min} \leq d^B_{T-1} + b$; since the jumps of B are $< L$ (i.e. $d^B_{i+1} - d^B_i < L$), we may find $P_B \in \{d^B_0, \ldots, d^B_{T-1}\}$ with $|d^A_{min} - P_B - b| < L/2$.

Triangle inequality applied to (7) implies

$$\begin{aligned} |d^A_{min} - P_B - b| &\leq |b| + L/2 + |d^A_{T-1} - d^B_{T-1}| \\ &\leq |b| + L/2 + |d^A_{T-1} - (T-1)L/2| + |d^B_{T-1} - (T-1)L/2|. \end{aligned}$$

Using Lemma 7 to bound the right hand side of the previous inequality, we obtain

$$\mathbb{E}\left[|d^A_{min} - P_B - b| \,\middle|\, \mathbf{err}\right] \leq |b| + L/2 + L\sqrt{8T\log(2/\epsilon)}. \tag{8}$$

We now wish to bound $\mathbb{E}\left[|P_B - d^B_{min}| \,\middle|\, \mathbf{err}\right]$. Let j satisfy $d^B_j = P_B$. As d^B_{min} uniformly distributes in $\{d^B_0, \ldots, d^B_{T-1}\}$ (even given **err**), we deduce

$$\begin{aligned} \mathbb{E}\left[|P_B - d^B_{min}| \,\middle|\, \mathbf{err}\right] &\leq \mathbb{E}\left[\mathop{\mathbb{E}}_{i\sim U(0,T-1)}[|P_B - d^B_i|] \,\middle|\, \mathbf{err}\right] \\ &\leq \mathbb{E}\left[\mathop{\mathbb{E}}_{i\sim U(0,T-1)}[|i-j|L/2 + |d^B_i - iL/2| + |d^B_j - jL/2|] \,\middle|\, \mathbf{err}\right] \\ &\leq (T-2)L/4 + L\sqrt{8T\log(2/\epsilon)}, \end{aligned} \tag{9}$$

where the last inequality uses Lemma 7 again. Combining (8) and (9) implies the required (5). ∎

4.2 The Iterated Random Walk DDL Algorithm

As described in Section 3, our full DDL protocol (i.e., Algorithm 3) runs iteratively several stages of Algorithm 2. It depends on a set of parameters: I, which is the number of iterations in the algorithm (on top of the basic DLL algorithm), $(t_i)_{i=0}^{I}$, which represent the number of queries in each of the $I+1$ iterations, and $(L_i)_{i=1}^{I}$, which determine the (maximal) sizes of steps performed in each random walk iteration.

Given a set of parameters, Lemmas 6 and 8 allow us to compute the failure probability of IteratedRandW under that set of parameters. A "naive" choice of parameters leads to a failure probability of $O(T^{-2}\log T)$. However, we show in the following theorem that the parameters can be chosen in such a way that the failure probability becomes $O(T^{-2})$.

Theorem 2. *There exists a parameter set PS for which the error probability of the iterated random walk DDL algorithm is*

$$\begin{aligned}&\Pr_{\mathbf{err}}(\text{IteratedRandW}_{PS}, [1,1], T)\\ &= \Pr[\text{IteratedRandW}_{PS}(g^x) - \text{IteratedRandW}_{PS}(g^{x+1}) \neq 1]\\ &\leq 2^{10.2+o(1)}/T^2.\end{aligned}$$

A simple *distance extension* argument (see Section 5.5, Lemma 13) allows us to obtain a similar result for larger distances between the starting points.

Corollary 1. *Consider Algorithm 3 with the parameter set PS chosen in Theorem 2. Then for any distribution of the initial distance b that has expectation $\mathbb{E}[|b|]$, the error probability of Algorithm 3 is at most $O(\mathbb{E}\,|b|/T^2)$. In particular, $\Pr_{\mathbf{err}}(\text{IteratedRandW}_{PS}, [-M, M], T) = O(M/T^2)$.*

We note that when $\mathbb{E}[|b|] \gg 1$, it is more efficient to start the sequence of iterations directly with a random walk of expected step length of roughly $\sqrt{\mathbb{E}[|b|]}$ (instead of starting it with Algorithm 1). This reduces the error probability by a constant factor.

We also note that for specific (not too small) values of T, one can strengthen the bound of Theorem 2, using a computer-aided choice of parameters. Such results, for several sample values of T, are presented in Appendix B. These results also show that for small values of T, the $o(1)$ term does not dominate the probability bound, and so, the result of Theorem 2 applies (up to a small constant factor) also for small values of T.

Proof (of Theorem 2). In the proof we use some additional notation. We assume that the party called A executes IteratedRandW with $h = h_{-1}$, while the party called B executes IteratedRandW with $h = h_{-1} \cdot g$. The starting distance between the parties is thus $b = 1$, and the distance after stage i, denoted by b_i, is $|p_i^A - p_i^B - b|$. For $i \in \{0, 1, \ldots, I\}$, we write $\mathbf{err}_i$ for the event that A, B do not agree on h_i (i.e. $b_i \neq 0$), and $\mathbf{err} = \mathbf{err}_I$. Note that once the parties agree on some h_i, then the rest of their (iterated) walks coincide and they are successful, i.e., $p_I^A = p_I^B + b$.

In order to prove the theorem, we should choose values for I, t_i and L_i so that $\Pr[\mathbf{err}]$ is small. For this, we start by upper-bounding $\Pr[\mathbf{err}]$ in terms of these parameters. We define the sequence of functions $\{f_k : \mathbb{N} \to [0,1]\}_{k=0}^{I+1}$ by

$$f_k(x) = \Pr\left[\mathbf{err} \,|\, b_{k-1} = x\right],$$

where b_{-1} is interpreted as the initial distance between the parties, i.e., $b_{-1} = b = 1$. We wish to bound these functions inductively, starting with $k = I+1$ and going down to $k = 0$.

Instead of working directly with the functions $\{f_k\}$, we shall bound them from above by concave functions $\{g_k\}$, and then bound the g_k's from above. This will allow us to combine upper bounds on the error probability after the $(k+1)$'th step conditioned on a *specific* distance between the parties after the k'th step, into an upper bound on the same error probability conditioned on the *expected* distance after the k'th step, that we already computed.

We start with $g_{I+1} \equiv 1$ (and so, clearly $f_{I+1} \leq g_{I+1}$), and define g_k by a backward induction. Assume g_{k+1} was already defined. We have

$$\begin{aligned}
f_k(x) &= \Pr\left[\mathbf{err} \,|\, b_{k-1} = x\right] \\
&= \sum_m \Pr\left[\mathbf{err} \,|\, b_k = m \wedge b_{k-1} = x\right] \Pr\left[b_k = m \,|\, b_{k-1} = x\right] \\
&\overset{(a)}{=} \sum_m f_{k+1}(m) \Pr\left[b_k = m \,|\, b_{k-1} = x\right] \\
&\overset{(b)}{=} \mathbb{E}\left[f_{k+1}(b_k) \,|\, b_{k-1} = x \wedge b_k \neq 0\right] \cdot \Pr\left[b_k \neq 0 \,|\, b_{k-1} = x\right] \\
&\overset{(c)}{\leq} g_{k+1}\left(\mathbb{E}\left[b_k \,|\, b_{k-1} = x \wedge \mathbf{err}_k\right]\right) \cdot \Pr\left[\mathbf{err}_k \,|\, b_{k-1} = x\right].
\end{aligned} \tag{10}$$

Equality (a) follows from the fact that the performance of the algorithm from stage $k+1$ onwards is conditionally independent from its performance on previous stages, given b_k. Equality (b) follows from the fact that $f_{k+1}(0) = 0$. Inequality (c) is a consequence of Jensen's inequality, together with the equality $\mathbf{err}_k = \{b_k \neq 0\}$.

The k'th stage of Algorithm 3 is applied on h'_{k-1}, which sufficiently precedes h_{k-1}, to ensure that the elements considered by both parties in the k'th stage had not been used in previous stages, and so the performance of the algorithm on stage k is conditionally independent of the previous stages, given p^A_{k-1}, p^B_{k-1}. Thus, stage k may be analyzed using the tools of Section 4.1, with b_{k-1} playing the role of b. Lemma 6 provides the bound

$$\Pr\left[\mathbf{err}_k \,|\, b_{k-1} = x\right] \leq \frac{4x}{t_k L_k} + \frac{L_k + \sqrt{32 L_k}}{t_k}. \tag{11}$$

Similarly, by setting $\epsilon = 1/T^2$, Lemma 8 asserts that if $\Pr\left[\mathbf{err}_k \,|\, b_{k-1} = x\right] \geq \epsilon$, then

$$\mathbb{E}\left[b_k \,|\, b_{k-1} = x \wedge \mathbf{err}_k\right] \leq x + \frac{t_k L_k}{4} + L_k \sqrt{32 t_k \log(2/\epsilon)}. \tag{12}$$

Combining (10), (11), and (12), we obtain

$$f_k(x) \leq \max\left\{\epsilon, g_{k+1}\left(x + \frac{t_k L_k}{4} + L_k\sqrt{32 t_k \log(2/\epsilon)}\right) \cdot \frac{\frac{4x}{L_k} + L_k + \sqrt{32 L_k}}{t_k}\right\}. \tag{13}$$

Since we eventually choose the functions $g_k(x)$ to be linear functions (as will be shown below), $g_k(x) = u_k x + v_k$ with $u_k, v_k \geq 0$, either we have $g_k(x) \geq g_k(1) \geq \epsilon$, or $g_k(1) < \epsilon$ and the theorem holds (even when truncating stages $0..k-1$ in Algorithm 3); hence, we may forget that we must take $g_k(x)$ to be at least ϵ.

A straightforward choice of $g_k(x)$ that guarantees $f_k(x) \leq g_k(x)$ is to set $g_k(x)$ to be the right hand side of (13). For the sake of simplicity, we write

$$\tilde{g}_k(x) = \left(\frac{\frac{4x}{L_k} + L_k}{t_k}\right) g_{k+1}\left(x + \frac{t_k L_k}{4}\right), \tag{14}$$

omitting 'error terms' of (13). We shall show below that for all k we have $f_k(x) \leq \tilde{g}_k(x)(1 + o(1))$, and thus, bounding $\tilde{g}_k$ is sufficient for bounding f_k.

However, since $g_{k+1}(x)$ is linear in x, $\tilde{g}_k(x)$ is quadratic in x, and so, not concave. To remedy this, we let

$$\forall k \geq 1 : C_k = \sum_{i=0}^{k-1} (L_i - 1) t_i,$$

which is an upper bound for b_{k-1}, and write $b_{k-1}^2 \leq C_k b_{k-1}$. Equation (14), together with $x^2 \leq C_k x$, allows us to define recursively a sequence of pairs (u_k, v_k), by

$$\begin{pmatrix} u_k \\ v_k \end{pmatrix} = A_k \begin{pmatrix} u_{k+1} \\ v_{k+1} \end{pmatrix}, \tag{15}$$

where

$$\forall k \geq 1 : A_k = \frac{1}{L_k t_k} \begin{pmatrix} (t_k + L_k) L_k + 4 C_k & 4 \\ t_k L_k^3 / 4 & L_k^2 \end{pmatrix}, \tag{16}$$

and then set $g_k(x) = u_k x + v_k$ to be a linear function that bounds $\tilde{g}_k(x)$ from above (and thus, also essentially bounds $f_k(x)$ from above).

Analogously, since we are only interested in $g_0(1)$, Lemmas 1 and 2 yield $f_0(1) \leq (1 + o(1)) g_1((t_0 + 1)/2) \cdot 2/(t_0 + 1)$, which allows extending the definition of g_k to $k = 0$ by setting

$$\begin{pmatrix} u_0 \\ v_0 \end{pmatrix} = A_0 \begin{pmatrix} u_1 \\ v_1 \end{pmatrix},$$

where

$$A_0 = \begin{pmatrix} 1 & 0 \\ 0 & 2/t_0 \end{pmatrix}.$$

Finally, we achieve a bound on Pr[**err**]:

$$\Pr[\mathbf{err}] \leq g_0(b) = (b\ 1)A_0A_1\cdots A_I\begin{pmatrix}0\\1\end{pmatrix} \tag{17}$$

To analyze Equation (17), we define (x_k, y_k) so that

$$(x_k\ \ \frac{4y_k}{t_kL_k}) = (1\ 1)A_0\cdots A_k.$$

The recursion formula for (x_k, y_k) is $(x_0\ y_0) = (1\ 1)$, and for $k \geq 1$,

$$(x_k\ y_k) = (x_{k-1}\ y_{k-1})\begin{pmatrix}1 + L_k/t_k + 4C_k/(t_kL_k) & 1\\ L_k^2/(L_{k-1}t_{k-1}) & L_k^2/(L_{k-1}t_{k-1})\end{pmatrix}.$$

It is clear from the formula that $y_k \leq x_k$, and thus, we can restrict ourselves to understanding x_k. We have

$$x_k \leq \left(1 + \frac{4C_k + L_k^2}{t_kL_k} + \frac{L_k^2}{t_{k-1}L_{k-1}}\right)x_{k-1}, \qquad x_0 = 1. \tag{18}$$

Suppose we prove $x_I \leq Q$. This implies $y_I \leq Q$, and so, $\Pr[\mathbf{err}] \leq 4Q/(t_IL_I)$. Hence it is also desirable to have $L_I \geq \Omega(T)$.

At this point, we are ready to choose values for t_k and L_k. We set

$$\begin{aligned}\sum_{k=0}^{I} t_k = T, \qquad t_k = 2^{1+c_1(I-k)}t_{k-1}.\\ L_0 = 2, \qquad L_k = 2^{c_2(k-I)}\sqrt{L_{k-1}t_{k-1}}.\end{aligned} \tag{19}$$

We start by computing a lower bound for L_I. Write $z_k = \log(t_k/L_k) \leq \log(T)$. Using the recursive definitions of t_k, L_k we get

$$z_k = \log\left(2^{1+(c_1+c_2)(I-k)}t_{k-1}/\sqrt{t_{k-1}L_{k-1}}\right) = z_{k-1}/2 + 1 + (c_1+c_2)(I-k).$$

Thus, inductively one obtains $z_k \leq \log(T)/2^k + 2 + 2(c_1+c_2)(I-k+1)$. We will later choose $I \geq \log\log(T) + \omega(1)$, so that in particular, $L_I \geq t_I/4^{1+c_1+c_2+o(1)}$.

Next, to bound x_I we should bound the quantities in (18). Writing $q_k = C_k/(t_kL_k)$, we have

$$q_kt_kL_k = C_k = C_{k-1} + t_{k-1}L_{k-1} = (1+q_{k-1})(t_{k-1}L_{k-1}).$$

We assume $I = o(\sqrt{\log(T)})$, so that $z_0 \geq 2 + 2(c_1+c_2)(I+1)$, and by induction,

$$z_k \geq 2 + 2(c_1+c_2)(I-k+1).$$

Note that $t_kL_k/(t_{k-1}L_{k-1}) = t_k/(2^{2c_2(I-k)}L_k)$, and hence,

$$\log(t_kL_k/(t_{k-1}L_{k-1})) \geq z_k - 2c_2(I-k).$$

We conclude,

$$q_k \leq \frac{1+q_{k-1}}{4^{1+c_2+(I-k+1)c_1}}.$$

This recursion remains true even if we artificially define $q_0 = 0$. Equation (18) now gives $x_k \leq m_k x_{k-1}$, where

$$m_k = 1 + 4q_k + \frac{1}{4^{1+(c_1+c_2)(I-k+1)}} + \frac{1}{4^{c_2(I-k)}}.$$

Lastly, notice $t_I \geq T/K$ where $K = (1 + 2^{-1} + 2^{-2+\binom{2}{2}c_1} + 2^{-3+\binom{3}{2}c_1} + \ldots)$.

Combining these estimates, we obtain the following upper bound on $\Pr[\mathbf{err}]$:

$$\Pr[\mathbf{err}] \leq 4 \cdot K^2 \cdot 4^{1+c_1+c_2} \prod_{k=1}^{I} m_k / T^2.$$

Computer-aided calculations suggests choosing $(c_1, c_2) = (0.44, 0.75)$ in (19), which results in

$$\Pr[\mathbf{err}] \leq 1133/T^2.$$

A few issues are due. The first is that I can indeed be taken to be anything which is $\log\log(T) + \omega(1)$ but still $o(\sqrt{\log(T)})$, without significant effect on the (provable) performance of the algorithm. Another issue is that the steps L_k must be taken to be integers. This does not affect the analysis much, as this only replaces the L_k's in the proof by others which multiplicatively differ by at most $1 + O(1/L_1)$ from the original values, and we have $L_1 = \omega(T^{1/3})$ since $I = o(\sqrt{\log(T)})$. Yet another issue is the error terms neglected in the pass from (13) to (14). These relative errors are only $1+O(\max(1/\sqrt{L_1}, \sqrt{\log(T)/t_k}))$, which is negligible since there are only I steps. ■

4.3 Experimental verification

In order to evaluate Algorithm 3 in practice, we programmed a simulator which simulates $\mathrm{IteratedRandW}_{PS}(g^x)$ and $\mathrm{IteratedRandW}_{PS}(g^{x+1})$, and empirically approximates $\Pr[\mathbf{err}]$ as the percentage of pairs of simulations which disagree. The parameters for these simulations were chosen using a numerical optimizer based on the analysis above. We used $\mathbb{G} = \mathbb{Z}$, $g = 1$ and $h = 0$. Since $\Pr[\mathbf{err}] \sim T^{-2}$, where T is the number of group elements computed by the two parties, a naive simulator must perform $\sim T^2$ simulations, each taking $\sim T$ group operations, to obtain only one instance for which the parties failed to synchronize, while we wish to obtain at least a few hundreds of such failures. Since T^3 time is too much, we performed the two following optimizations, which do not affect the simulator's reliability.

1. The first optimization we use is to trim the first stage of the algorithm, where we perform $\mathrm{Basic}_{\mathbb{G},g}(h, t_0)$. We know that the probability that $c_0^A \neq c_0^B$ is precisely $2/(t_0+1)$ and in this case, the absolute value of the distance between c_0^A and c_0^B of Algorithm 3 distributes according to $U(1, t_0 + 1)$.

2. The second optimization we use is the observation that once $d_i^A = d_i^B$ in Algorithm 3, we will end up with $d_I^A = d_I^B$, since if the parties managed to synchronize in an early stage, they will keep being synchronized.

The results of the simulations are given in Table 1.[9] In order to obtain experimental data for larger values of T, we performed separate experiments in which we obtained an additional optimization – a much more subtle one. We describe this optimization and the experimental results obtained using it in Appendix C.

T	I	$\sim \log_2(t_k)$ $\sim \log_2(L_k)$	$T^2 \cdot \Pr[\mathbf{err}]$ σ (SD)	$T^2 \cdot \Pr[\mathbf{err}]$ σ (SD)
2^{13}	5	6.0, 8.6, 9.6, 10.4, 11.1, 11.7 1.6, 3.6, 5.6, 7.5, 9.4	334 2	336.6 0.3
2^{16}	6	7.1, 10.3, 11.6, 12.5, 13.3, 14.1, 14.7 1.6, 3.9, 6.2, 8.3, 10.3, 12.2	390 10	382.5 1
2^{19}	7	7.0, 10.3, 12.6, 13.7, 14.7, 15.5, 16.3, 17.1, 17.7 1.6, 2.8, 5.2, 7.4, 9.6, 11.5, 13.4, 15.2	391 25	394 2
2^{22}	8	8.2, 12.6, 15.1, 16.4, 17.5, 18.5, 19.3, 20.1, 20.7 1.6, 3.7, 6.7, 9.4, 11.8, 14.1, 16.2, 18.1	— —	420 4
2^{25}	9	8.4, 13.0, 16.5, 18.0, 19.3, 20.5, 21.5, 22.3, 23.1, 23.8 1.6, 3.2, 6.4, 9.4, 12.2, 14.7, 17.0, 19.1, 21.1	— —	427 10

The fourth column gives the result of simulations without the third optimization (detailed in Appendix C), and the last column uses that optimization.

Table 1. Experimental Results

5 Error Probability Lower Bounds in Concrete Group Families

5.1 Overview of the Lower Bound Proof

We outline the main ideas of the lower bound proof in concrete family groups. For the sake of simplicity, we only consider DDL with $M = 1$ in this overview (whereas the proof considers general M).

[9] We note that according to the proof of Theorem 2, I can be taken to be any value between $\log\log(T)+\omega(1)$ and $o(\sqrt{\log(T)})$, without a significant effect on the provable performance of the algorithm. On the other hand, according to Table 1, I seems to grow more sharply. However, the restriction of $I = o(\sqrt{\log(T)})$ is merely an artifact of the proof and the optimal value of I could be asymptotically larger. Furthermore, the sharp increase of the values of I in Table 1 could be attributed to low-order terms that have a more noticeable effect for small T values.

We first prove in Lemma 9 that in a DDL protocol, using different algorithms for A, B cannot give a significant advantage in the error probability. As a result, we can assume that both A and B use A's algorithm. This simplifies the analysis, but we note that the lower bound can be proved without this simplification.

Let us assume that we can solve DDL with error probability $\delta \ll 1$ in time T for $M = 1$. Our main reduction shows how to use A's algorithm to solve DLI in an interval of length about $R \approx 1/\delta$ in time less than $4T$ with probability $1/2$. If we assume that in a specific family of groups, DLI in an interval of length $c \cdot T^2$ (for a sufficiently large constant c) cannot be solved in complexity lower than $4T$ with probability $1/2$,[10] we must have $R \approx 1/\delta < c \cdot T^2$ or $\delta = \Omega(T^{-2})$, which gives our main lower bound for the case of $M = 1$. It is important to stress that A is a DDL algorithm for $M = 1$ that is not explicitly given the DLI interval length parameter R. Yet the reduction below will apply A's algorithm to solve DLI with parameter R in a black-box manner.

Recall that a DLI algorithm obtains as input a group element $h = g^x$, where x is in a known interval of length $R \approx 1/\delta$. By the self-reducibility of discrete log, we can assume that h is a uniform group element (i.e., we can multiply the input by a randomly chosen group element). Our reduction picks a point g^z in the interval (for a known z) and runs A on inputs g^x and g^z, where $|x - z| \leq R$. We hope that $A(g^x) - A(g^z) = z - x$ and thus we return $z - (A(g^x) - A(g^z)) = x$.

Clearly, the DLI algorithm runs in time less than $4T$ and it remains to upper bound its error probability by $1/2$. In other words, we need to upper bound the probability of $A(g^x) - A(g^z) \neq z - x$ by $1/2$. We know that the DDL error probability is δ for $M = 1$, namely, if $|x - z| \leq 1$, then the required probability is[11] δ. Next, assume that $z = x + 2$. Then, if $A(g^x) - A(g^{x+2}) \neq 2$ this implies that either $A(g^x) - A(g^{x+1}) \neq 1$ or $A(g^{x+1}) - A(g^{x+2}) \neq 1$ (or both). Since the probability of each of these two events is δ, we can use a union bound to upper bound the probability that $A(g^x) - A(g^{x+2}) \neq 2$ by 2δ. Using a similar argument (which we refer to as *distance extension*, formalized in Lemma 13), we can upper bound the probability of the event $A(g^x) - A(g^z) \neq z - x$ for $|x - z| \leq R$ by $O(R \cdot \delta)$ and for $R = O(1/\delta)$, this gives error probability $1/2$, as required. Note that the same algorithm A is used for any distance $|x - z| \leq R$ (which is unknown in advance) and conditioning on this distance is only done for the sake of analysis.

5.2 The Single Algorithm Distributed Discrete Log Problem

We now define the *single algorithm* DDL problem, which is the same problem as general DDL with the restriction that the algorithms of the parties are the same (i.e., both parties use A's algorithm). Denote by $\mathbf{err}(A, x, b, T)$ the event

[10] We consider only uniform algorithms that can be applied to families of groups (such as elliptic curve groups) and not non-uniform algorithms that are specialized to a specific group $\mathbb{G}$. Indeed, in the non-uniform model, there exist algorithms that solve DLI in an interval of length R in time $o(\sqrt{R})$ for any specific group (see, e.g., [2]).

[11] More accurately, it is $O(\delta)$, as δ is the average error probability in the interval $[-1, 1]$.

$A(g^x) - A(g^{x+b}) \neq b$ and by $\Pr_{\mathbf{err}}(A, [M_1, M_2], T)$ its probability (over $x \in \mathbb{Z}_N, b \in [M_1, M_2]$). Obviously, the optimal 2-party DDL error probability is a lower bound on the optimal single algorithm DDL error probability. In this section, we prove that the bound in the other direction holds as well up to a constant factor in case $M_2 = -M_1 = M$.[12]

Lemma 9. $\Pr_{\mathbf{err}}(A, B, [-M, M], T) \geq \frac{1}{8} \cdot \Pr_{\mathbf{err}}(A, [-M, M], T)$.

Proof. Note that if $A(g^{x+b_1}) - A(g^{x+b_2}) \neq b_2 - b_1$, then $A(g^{x+b_1}) - B(g^x) \neq -b_1$ or $A(g^{x+b_2}) - B(g^x) \neq -b_2$ (or both). Therefore, for uniform $b_1, b_2 \in [-M, M]$,

$$\begin{aligned}
&\Pr_{x,b_1,b_2}[\mathbf{err}(A, x + b_1, b_2 - b_1, T)] = \Pr_{x,b_1,b_2}[A(g^{x+b_1}) - A(g^{x+b_2}) \neq b_2 - b_1] \\
&\leq \Pr_{x,b_1,b_2}[(A(g^{x+b_1}) - B(g^x) \neq -b_1) \cup (A(g^{x+b_2}) - B(g^x) \neq -b_2)] \\
&\leq \Pr_{x,b_1}[A(g^{x+b_1}) - B(g^x) \neq -b_1] + \Pr_{x,b_2}[A(g^{x+b_2}) - B(g^x) \neq -b_2] \\
&= 2 \cdot \Pr_{\mathbf{err}}(A, B, [-M, M], T).
\end{aligned}$$

It remains to relate $\Pr_{\mathbf{err}}(A, [-M, M], T)$ to $\Pr_{x,b_1,b_2}[\mathbf{err}(A, x + b_1, b_2 - b_1, T)]$. Denote the event $|b_2 - b_1| \leq M$ by $\mathcal{E}$ and note that $\Pr_{b_1,b_2}[\mathcal{E}] \geq 1/2$. Conditioned on $\mathcal{E}$, if $b_2 - b_1$ was uniform in $[-M, M]$, then we would have

$$\Pr_{x,b_1,b_2}[\mathbf{err}(A, x + b_1, b_2 - b_1, T) \,|\, \mathcal{E}] = \Pr_{\mathbf{err}}(A, [-M, M], T).$$

Although it is not uniform, $b_2 - b_1$ is almost uniform in the sense that for each $i \in [-M, M]$, we have $\Pr_{b_1,b_2}[b_2 - b_1 = i] \geq \Pr_{b_1,b_2}[b_2 - b_1 = M] \geq (M + 1)/(4M^2)$ and $\Pr_{b_1,b_2}[b_2 - b_1 = i] \leq \Pr_{b_1,b_2}[b_2 - b_1 = 0] \leq (2M + 1)/(4M^2)$. As the minimal and maximal probabilities assigned to $|b_2 - b_1|$ in $[-M, M]$ are within a factor of 2,

$$\Pr_{x,b_1,b_2}[\mathbf{err}(A, x + b_1, b_2 - b_1, T) \,|\, \mathcal{E}] \geq \frac{1}{2} \cdot \Pr_{\mathbf{err}}(A, [-M, M], T)$$

and

$$\begin{aligned}
\Pr_{x,b_1,b_2}[\mathbf{err}(A, x + b_1, b_2 - b_1, T)] &\geq \Pr_{x,b_1,b_2}[\mathbf{err}(A, x + b_1, b_2 - b_1, T) \,|\, \mathcal{E}] \cdot \Pr_{b_1,b_2}[\mathcal{E}] \\
&\geq \frac{1}{4} \cdot \Pr_{\mathbf{err}}(A, [-M, M], T).
\end{aligned}$$

Finally,

$$\begin{aligned}
&\Pr_{\mathbf{err}}(A, B, [-M, M], T) \geq \frac{1}{2} \cdot \Pr_{x,b_1,b_2}[\mathbf{err}(A, x + b_1, b_2 - b_1, T)] \\
&\geq \frac{1}{8} \cdot \Pr_{\mathbf{err}}(A, [-M, M], T),
\end{aligned}$$

[12] It is also possible to prove a similar bound in case the interval $[M_1, M_2]$ is not symmetric around the origin.

concluding the proof. ■

The consequence of the lemma is that for the sake of proving lower bounds on the error probability, we can restrict our attention to A's algorithm by analyzing $\Pr_{\mathbf{err}}(A, [-M, M], T)$. The lemma immediately gives us the same lower bound on $\Pr_{\mathbf{err}}(A, B, [-M, M], T)$, up to a constant factor.

Furthermore, note that by symmetry we have $\Pr_{\mathbf{err}}(A, B, [-M, M], T) \geq 1/8 \cdot \Pr_{\mathbf{err}}(B, [-M, M], T)$, hence the general DDL error probability is lower bounded by the maximal error probability of the (single) algorithms of the two parties (up to constant factors). Therefore, running different algorithms for the two parties cannot give a much better result than simply having both players run the best algorithm in the single algorithm setting.

5.3 Limitation on Randomness

The effect of the internal randomness of a DDL algorithm A on its outcome is quantified by $\Pr_{\mathbf{err}}(A, [0, 0], T)$. This quantity measures the probability that two different executions of A on the same input differ, where the probability is taken over A's input g^x and its internal randomness. We prove that A's internal randomness cannot significantly influence its outcome.

Lemma 10. *Assume* $\Pr_{\mathbf{err}}(A, [-M, M], T) = \delta$. *Then* $\Pr_{\mathbf{err}}(A, [0, 0], T) \leq 2\delta$.

Proof. To be more explicit, we denote by $A(r, g^x)$ the execution of A with a randomness string r. Assume we fix the output of $A(r, g^{x+b})$ for some $b \in [-M, M]$ and randomness string r. Then, if $A(r_1, g^x) \neq A(r_2, g^x)$ for r_1, r_2, either $A(r_1, g^x) - A(r, g^{x+b}) \neq b$ or $A(r_2, g^x) - A(r, g^{x+b}) \neq b$ (or both). Hence,

$$\begin{aligned}
&\Pr_{\mathbf{err}}(A, [0, 0], T) \\
&= \Pr_{r_1, r_2, x}[A(r_1, g^x) \neq A(r_2, g^x)] \\
&\leq \Pr_{r_1, r_2, r, x, b}[(A(r_1, g^x) - A(r, g^{x+b}) \neq b) \cup (A(r_2, g^x) - A(r, g^{x+b}) \neq b)] \\
&\leq \Pr_{r_1, r, x, b}[A(r_1, g^x) - A(r, g^{x+b}) \neq b] + \Pr_{r_2, r, x, b}[A(r_2, g^x) - A(r, g^{x+b}) \neq b] \\
&= 2\delta.
\end{aligned}$$

■

5.4 Symmetry

We prove the following symmetric property.

Lemma 11. *For* $M_2 \geq M_1$, $\Pr_{\mathbf{err}}(A, [M_1, M_2], T) = \Pr_{\mathbf{err}}(A, [-M_2, -M_1], T)$.

Proof. It is sufficient to prove that for any positive integer b, $\Pr_{\mathbf{err}}(A, [b, b], T) = \Pr_{\mathbf{err}}(A, [-b, -b], T)$. This indeed holds, since $\mathbf{err}(A, x, b, T)$ and $\mathbf{err}(A, x + b, -b, T)$ are identical events, and thus,

$$\begin{aligned}
\Pr_{\mathbf{err}}(A, [b, b], T) &= \Pr_x[\mathbf{err}(A, x, b, T)] = \Pr_x[\mathbf{err}(A, x + b, -b, T)] \\
&= \Pr_{\mathbf{err}}(A, [-b, -b], T).
\end{aligned}$$

■

5.5 Distance Extension

We now show that the distance parameter M of any DDL algorithm A can be extended at the expense of a linear loss in the error probability.

First, we prove the following lemma which reduces the error probability of $\Pr_{\mathbf{err}}(A, [-M, M], T)$ to each one of the indices in the interval. As mentioned in the Introduction (see Footnote 4), this implies that a DDL algorithm with error probability δ for uniform $b \in [-M, M]$ also solves DDL with an error probability $O(\delta)$ for any distribution on $b \in [-M, M]$.

Lemma 12. *Assume that* $\Pr_{\mathbf{err}}(A, [-M, M], T) = \delta$. *Then, for every* $b \in [-M, M]$, $\Pr_{\mathbf{err}}(A, [b, b], T) \leq 4\delta$.

Proof. We first assume that $b \in [1, M]$ and let $i \in [-M, M - b]$. Clearly, if g^x is a uniform group element, then so is g^{x+i}. Therefore, $\Pr_{\mathbf{err}}(A, [b, b], T) = \Pr_x[\mathbf{err}(A, x + i, b, T)]$. Furthermore, if the event $\mathbf{err}(A, x + i, b, T)$ occurs, then $A(g^{x+i}) - A(g^{x+i+b}) \neq b$, implying that at least one of the events $A(g^x) - A(g^{x+i+b}) \neq i+b$ and $A(g^x)-A(g^{x+i}) \neq i$ must occur. Consequently, $\Pr_{\mathbf{err}}(A, [b, b], T)$ (the error probability associated with index b) is upper bounded by $\Pr_{\mathbf{err}}(A, [i + b, i + b], T) + \Pr_{\mathbf{err}}(A, [i, i], T)$ (the sum of error probabilities associated with indices i and $i + b$). Formally,

$$\begin{aligned}
\Pr_{\mathbf{err}}(A, [b, b], T) &= \Pr_x[\mathbf{err}(A, x + i, b, T)] \\
&\leq \Pr_x[\mathbf{err}(A, x, i + b, T) \cup \mathbf{err}(A, x, i, T)] \\
&\leq \Pr_x[\mathbf{err}(A, x, i + b, T)] + \Pr_x[\mathbf{err}(A, x, i, T)] \\
&= \Pr_{\mathbf{err}}(A, [i + b, i + b], T) + \Pr_{\mathbf{err}}(A, [i, i], T).
\end{aligned}$$

We map the indices in $[-M, M]$ into disjoint pairs of the form $i, i + b$ (this implies that $i \in [-M, M - b]$). We can obtain at least $\lfloor (2M - b + 1)/2 \rfloor$ such pairs, which is at least $(M + 1)/2$ for $b \in [1, M - 1]$. On the other hand, for $b = M$, the number of pairs is $M \geq (M + 1)/2$. We apply the above inequality to each of the pairs:

$$\begin{aligned}
\delta &= \Pr_{\mathbf{err}}(A, [-M, M], T) \\
&= \frac{1}{2M + 1} \cdot \sum_{i=-M}^{M} \Pr_{\mathbf{err}}(A, [i, i], T) \\
&\geq \frac{1}{2M + 1} \cdot \frac{M + 1}{2} \cdot \Pr_{\mathbf{err}}(A, [b, b], T) \\
&\geq \frac{1}{4} \Pr_{\mathbf{err}}(A, [b, b], T).
\end{aligned}$$

Thus, $\Pr_{\mathbf{err}}(A, [b, b], T) \leq 4\delta$ for $b \in [1, M]$.

The proof for $b \in [-M, -1]$ follows by symmetry (Lemma 11). Finally, for $b = 0$, we have $\Pr_{\mathbf{err}}(A, [0, 0], T) \leq 2\delta$ by Lemma 10. ■

Lemma 13. *Let* $\Pr_{\mathbf{err}}(A, [-M, M], T) = \delta$. *Then for any* $\beta > 1$,

$$\Pr_{\mathbf{err}}(A, [-\beta M, \beta M], T) \leq 8\beta \cdot \delta.$$

For the sake of simplicity we assume that βM is an integer (otherwise, we only consider integer values in $[-\beta M, \beta M]$).

Proof. First, we analyze $\Pr_{\mathbf{err}}(A, [1, \beta M], T)$. Let $b \in [1, \beta M]$ and divide it by M, writing $b = b_1 \cdot M + b_2$, for integers $b_1 \leq \beta$ and $b_2 \in [0, M)$. We examine the following $b_1 + 1$ success events:

$$\begin{aligned}
\mathcal{E}_1 &: \mathbf{suc}(A, x, M, T) \\
\mathcal{E}_2 &: \mathbf{suc}(A, x + M, M, T) \\
&\dots \\
\mathcal{E}_{b_1} &: \mathbf{suc}(A, x + (b_1 - 1)M, M, T) \\
\mathcal{E}_{b_1+1} &: \mathbf{suc}(A, x + b_1 M, b_2, T)
\end{aligned}$$

Observe that if all b_1+1 events hold (i.e. $\cap_{i=1}^{b_1+1}\mathcal{E}_i$), then $A(g^x) - A(g^{x+b_1M+b_2}) = A(x) - A(g^{x+b}) = b$, i.e., $\mathbf{suc}(A, x, b, T)$ holds.

By Lemma 12, we have $\Pr_x[\bar{\mathcal{E}}_i] \leq 4\delta$ for each $i \in 1, 2, \dots, b_1$, and $\Pr_{x,b}[\bar{\mathcal{E}}_{b_1+1}] \leq 4\delta$. Hence,

$$1 - \Pr_{\mathbf{err}}(A, [1, \beta M], T) \geq \Pr_{x,b}[\cap_{i=1}^{b_1+1}\mathcal{E}_i] = 1 - \Pr_{x,b}[\cup_{i=1}^{b_1+1}\bar{\mathcal{E}}_i]$$

$$\geq 1 - \sum_{i=1}^{b_1+1} \Pr_{x,b}[\bar{\mathcal{E}}_i] \geq 1 - (b_1 + 1)4\delta \geq 1 - (\beta + 1)4\delta \geq 1 - 8\beta \cdot \delta.$$

Therefore, $\Pr_{\mathbf{err}}(A, [1, \beta M], T) \leq 8\beta \cdot \delta$. By symmetry (Lemma 11), we have $\Pr_{\mathbf{err}}(A, [-\beta M, -1], T) \leq 8\beta \cdot \delta$ as well. Since $\Pr_{\mathbf{err}}(A, [0, 0], T) \leq 2\delta$ by Lemma 10, we conclude that $\Pr_{\mathbf{err}}(A, [-\beta M, \beta M], T) \leq 8\beta \cdot \delta$, as claimed. ■

Remark 1. An open question of [5] asked whether the DDL error probability can be eliminated completely. If we apply the above lemma with no error (i.e., $\Pr_{\mathbf{err}}(A, [-M, M], T) = \delta = 0$), we obtain $\Pr_{\mathbf{err}}(A, [-\beta M, \beta M], T) = 0$, implying that the two parties (running A's algorithm) never err for any distance. This allows the parties to collectively solve the discrete log problem in $\mathbb{G}$ with probability 1 (a similar reduction will be formally presented in Algorithm 4), thus violating the security assumption of the underlying HSS scheme. Namely, the DDL error probability cannot be eliminated (in fact it is easy to show that it must be superpolynomial in $1/N$), answering negatively the open question of Boyle et al.

5.6 Reduction from Discrete Log in an Interval to Distributed Discrete Log

Recall that the discrete log problem in an interval (DLI) is parametrized by an interval length R for a cyclic multiplicative group $\mathbb{G}$ of size N with generator g.

The input to the problem is a group element $h = g^x$, where[13] $x \in [0, R-1]$ and the goal is to recover x with high probability (which is at least a constant).

The following lemma reduces the DLI problem to DDL.

Lemma 14. *For a family of groups, assume that* $\mathrm{Pr}_{\mathbf{err}}(A, [-M, M], T) = \delta$, *where* $T \geq \log N$ *and* $\delta < 1/32$. *Then discrete log in an interval of length* $R = M/(32\delta)$ *can be solved in complexity* $4T$ *with probability* $1/2$.

Proof. Consider Algorithm 4 for solving DLI on input $h = g^x$ for $x \in [0, R-1]$. For the sake of simplicity we assume that R is even.

Algorithm 4: DLI(h)

```
begin
    y ←_R Z_N;
    d_1 ← A(h · g^y);
    d_2 ← A(g^{y+(R/2)});
    Output (R/2) − (d_1 − d_2);
end
```

The algorithm computes $h \cdot g^y$ and $g^{y+(R/2)}$, which can be carried out by performing $2 \log N$ group operations using the square-and-multiply algorithm. It further invokes A twice in complexity $2T$ and therefore its total complexity is $2T + 2 \log N \leq 4T$ (since $T \geq \log N$).

It remains to upper bound the error probability of the algorithm by $1/2$. The algorithm succeeds to return x if $(R/2) - (d_1 - d_2) = x$, or equivalently,

$$A(g^{y+x}) - A(g^{y+(R/2)}) = d_1 - d_2 = (R/2) - x.$$

Since $y \in \mathbb{Z}_N$ is uniform, then g^{y+x} is a uniform group element. Moreover, since $x \in [0, R-1]$, then $(R/2) - x \in [-R/2, R/2]$. Therefore, by Lemma 12, the error probability of the algorithm is at most

$$\begin{aligned} 4 \cdot \mathrm{Pr}_{\mathbf{err}}(A, [-R/2, R/2], T) &\leq 4 \cdot 8 \cdot \frac{R}{2M} \cdot \mathrm{Pr}_{\mathbf{err}}(A, [-M, M], T) \\ &= 16 \cdot \frac{R}{M} \cdot \delta = 1/2, \end{aligned}$$

where the first inequality is due to Lemma 13. Note that we use Lemma 12 (and pay a factor of 4 in the error probability), as $x \in [0, R-1]$ may be selected by an adversary (whereas $\mathrm{Pr}_{\mathbf{err}}(A, [-R/2, R/2], T)$ averages the error probability). ■

Theorem 3 is a simple corollary of Lemma 14.

[13] Alternatively, x could be in any fixed interval of length R. The exact interval is not important as one can easily reduce the problem in a given interval to any other interval.

Theorem 3. *For a specific family of groups, assume there exists a constant c such that for any group in the family of size N, DLI in an interval of length at least $c \cdot T^2$ cannot be solved in complexity $4T$ with probability at least $1/2$ (where $\log N \leq T < \mathcal{B}$ for some bound $\mathcal{B}$).*[14]*. Moreover, assume that there is a DDL protocol A for this family with time complexity parameter T, maximal distance parameter M and error probability $\Pr_{\mathbf{err}}(A, [-M, M], T) = \delta$ for $\delta < 1/32$. Then $\delta = \Omega(M \cdot T^{-2})$.*

Proof. By Lemma 14, discrete log in an interval of length $R = M/(32\delta)$ can be solved in complexity $4T$ with probability $1/2$. By our assumption, $R = M/(32\delta) < c \cdot T^2$ implying that $\delta = \Omega(M \cdot T^{-2})$ as claimed. ■

6 Error Probability Lower Bounds for Non-Adaptive Algorithms in the Generic Group Model

In this section, we prove lower bounds on DDL algorithms in the generic group model (GGM), focusing on non-adaptive algorithms. We first review the generic group model (GGM) we consider (which is slightly different than the one proposed by Shoup [20]) and formulate DDL in this model. This formulation is given for the additive group $\mathbb{Z}_N$, which is isomorphic to the cyclic multiplicative group $\mathbb{G}$ of size N.

6.1 Distributed Discrete Log in the Generic Group Model

Let $\mathbb{Z}_N$ be the additive group of integers, and let S be a set of bit strings of cardinality at least N. An encoding function of $\mathbb{Z}_N$ on S is an injective map $\sigma : \mathbb{Z}_N \to S$.

A generic algorithm A for $\mathbb{Z}_N$ on S for the discrete logarithm problem is a probabilistic algorithm that takes as input an *encoding list* that consists of $\sigma(1), \sigma(x)$, namely, the encodings of a generator of $\mathbb{Z}_N$ and a uniform $x \in \mathbb{Z}_N$, where σ is an encoding function of $\mathbb{Z}_N$ on S. Throughout its execution, A continues to maintain the encoding list, and is allowed to extend it using oracle queries. An oracle query in our model specifies two indices $i, j \in \mathbb{Z}_N$. The oracle computes $\sigma(i \cdot x + j)$ and the returned bit string is appended to the encoding list. The algorithm A succeeds to solve the discrete log problem if $A(\sigma; 1, x) = x$, and its success probability is taken over the uniform choices of $\sigma : \mathbb{Z}_N \to S$ and $x \in \mathbb{Z}_N$ (and perhaps additional randomness of its own coin tosses). We measure the complexity of A according to the number of oracle queries it makes. The following success probability upper bound was proved in [20].

Theorem 4 ([20]). *If a generic discrete log algorithm A is allowed T oracle queries, then $\Pr_{\sigma,x}[A(\sigma; 1, x) = x] = O(T^2/N)$, assuming that N is prime.*

[14] We note that the parameter $\mathcal{B}$ for which our assumption on the hardness of DLI is believed to hold, depends on the specific family of groups. For example, for some subgroups of $\mathbb{Z}_p^*$, $\mathcal{B}$ is subexponential in $\log N$.

We note that our GGM formulation is slightly stronger than the one of [20], where the queries of A are limited to linear combinations with coefficients of ± 1 of elements in its encoding list. Since any query (i, j) can be issued in Shoup's original GGM after at most $O(\log N)$ queries using the double-and-add algorithm, a stronger GGM algorithm can be simulated by a standard one by increasing the query complexity by a multiplicative factor of $\log N$. However, by following its original proof in [20], it is easy to verify that Theorem 4 actually holds with no modification in our stronger GGM. Obviously, any algorithm in the original GGM is also an algorithm in the stronger GGM. Therefore, any lower bounds we obtain in the stronger GGM also apply in the original GGM.

We now describe the basic game of distributed discrete log in GGM. Obviously, all the results of Section 5 also hold in the generic group model. In particular, by Lemma 9 it is sufficient to consider single algorithm DDL to obtain general DDL lower bounds.

A party (algorithm) A is given as input $\sigma(1)$ and the encoding of an additional group element $\sigma(x)$ for $x \in \mathbb{Z}_N$, selected uniformly at random. Algorithm A is allowed to make T oracle queries. After obtaining the answers from the oracle, A returns an integer value. Two parties (both running A's algorithm) win the DDL game in GGM if

$$A(\sigma; 1, x) - A(\sigma; 1, x + b) = b.$$

Otherwise, they lose the game, or err.

We are interested in proving lower bounds on the DDL error probability as a function of T, namely

$$\Pr_{\sigma,x,b}[A(\sigma; 1, x) - A(\sigma; 1, x + b) \neq b].$$

Analogously to our notation for multiplicative groups, we denote by $\mathbf{err}(A, \sigma, x, b, T)$ the error event $A(\sigma; 1, x) - A(\sigma; 1, x + b) \neq b$, and by $\Pr_{\mathbf{err}}(A, \sigma, [M_1, M_2], T)$ its probability $\Pr_{\sigma,x,b}[\mathbf{err}(A, \sigma, x, b, T)]$, where $b \in [M_1, M_2]$ is a uniform integer. We further denote by $\mathbf{suc}(A, \sigma, x, b, T)$ the complementary success event.

6.2 An Error Probability Lower Bound for Arbitrary Generic Algorithms

The following theorem gives a DDL error probability lower bound in GGM. The theorem is a somewhat weaker statement than Theorem 3 (which has implications in concrete group families).

Theorem 5. *For any generic DDL algorithm A, $\Pr_{\mathbf{err}}(A, \sigma, [-M, M], T) = \Omega(M \cdot T^{-2})$, given that $M = O(T^2)$, $T = o(\sqrt{N})$, and N is prime.*

We omit the proof, as it is similar to the proof of Theorem 3. It applies a reduction to discrete log, while using Theorem 4 to obtain the error probability lower bound. Alternatively, one could obtain a hardness result for DLI in GGM (extending Theorem 4 to smaller intervals) and apply Theorem 3 directly.

6.3 An Error Probability Lower Bound for Non-Adaptive Generic Algorithms

In this section, we prove a lower bound on the DDL error probability of non-adaptive generic algorithms, whose oracle queries $\{(i_1, j_1), (i_2, j_2), \ldots, (i_T, j_T)\}$ are fixed in advance and do not depend on previous answers.

We will prove the following lower bound:

Theorem 6. *Any non-adaptive DDL algorithm A satisfies*

$$\Pr_{\mathbf{err}}(A, \sigma, [-1, 1], T) = \Omega(1/T),$$

given that $T = o(N^{1/2})$, and N is prime.

Overview of the Lower Bound Proof on Non-Adaptive Algorithms in the Generic Group Model Let us first consider the class of algorithms that make T consecutive oracle queries to group elements (such as Algorithm 1 and the ones of [4–6] in general). Consider the executions $A(\sigma; 1, x)$ and $A(\sigma; 1, x+T)$, which query $2T$ distinct group elements. In GGM, algorithm executions that query disjoint sets of elements are essentially independent, which implies that the probability that $A(\sigma; 1, x) - A(\sigma; 1, x + T) \neq T$ is at least $1/2$. Indeed, one can construct a bijective mapping between encodings σ for which $A(\sigma; 1, x) - A(\sigma; 1, x+T) = T$ and encodings σ' for which $A(\sigma'; 1, x) - A(\sigma'; 1, x+T) = -T$ (by defining $\sigma'(x + i) = \sigma(x + i + T)$ for $i \in [0, T - 1]$, $\sigma'(x + i) = \sigma(x + i - T)$ for $i \in [T, 2T - 1]$ and $\sigma'(x + i) = \sigma(x + i)$ otherwise).

Recall that we are interested in the probability that $A(\sigma; 1, x) - A(\sigma; 1, x + 1) \neq 1$ and it can be lower bounded by $\Omega(T^{-1})$ using distance extension (Lemma 13). A similar lower bound applies if A only queries group elements in a short interval of length $O(T)$.[15]

Of course, we are interested in proving the $\Omega(T^{-1})$ lower bound for arbitrary non-adaptive algorithms. The main idea that allows us to achieve this is to define a transformation that takes an arbitrary non-adaptive algorithm A' and maps its T queries to a small interval of size $O(T)$, obtaining a new algorithm A, for which the error lower bound $\Omega(T^{-1})$ holds. We require that the query mapping preserves the error probability of A', thus proving that the above error probability lower bound $\Omega(T^{-1})$ applies to non-adaptive algorithms in general. In order to preserve the error probability of A', the mapping will ensure that the joint input distribution of $A'(\sigma; 1, x)$ and $A'(\sigma; 1, x + 1)$ is equal to that of $A(\sigma; 1, x)$ and $A(\sigma; 1, x + 1)$. In the generic group model, this means that the mapping should preserve joint queries, namely, satisfy the condition that query i of $A'(\sigma; 1, x)$ and query j of $A'(\sigma; 1, x+1)$ evaluate the same group element if and only if query i of $A(\sigma; 1, x)$ and query j of $A(\sigma; 1, x+1)$ evaluate the same group element.[16] Based on this observation, it is possible to define an appropriate query

[15] Our actual proof is slightly more general than outlined here and uses the notion of query chains.

[16] Our proof relaxes this strong condition, and requires that it holds unless a low-probability event (called a collision) occurs.

mapping and complete the proof, since for non-adaptive algorithms we know in advance (independently of σ) whether query i of $A'(\sigma;1,x)$ and and query j of $A'(\sigma;1,x+1)$ evaluate the same group element.

Additional Notation We begin by defining additional notation. Given a query (i,j), denote its evaluation on x as $(i,j)[x] = ix+j$. Thus, its oracle answer is $\sigma(ix+j)$. We denote by $Q(A(\sigma;1,x))$ the query set of $A(\sigma;1,x)$, excluding queries (i,j) for which $i=0$ (which we call *constant queries*). Denote by $QE(A(\sigma;1,x))$ the set of evaluations of all (non-constant) queries $Q(A(\sigma;1,x))$.

We further denote by $Q(A)$ the set containing all of the potential (non-constant) queries of A on any input x and encoding σ. Note that for non-adaptive algorithms, $|Q(A)| \leq T$ and any adaptive algorithm A' can be simulated by a non-adaptive algorithm that makes $T' \stackrel{\text{def}}{=\!=} |Q(A')|$ queries.

For the rest of this section, we focus on non-adaptive algorithms. For such algorithms, we can write $QE(A,x)$ (instead of $QE(A(\sigma;1,x))$), as the query evaluations are independent of σ (obviously, the oracle answers on these evaluations invoke σ).

Restricted Queries We examine pairs of executions $A(\sigma;1,x)$ and $A(\sigma;1,x+b)$ for some $b \in [-M,M]$. For such a pair, we define a (non-trivial) collision as the event that two queries issued by these executions (i,j) and (i',j') with $i \neq i'$ have the same evaluation. The actual evaluations depend on which algorithm issued the queries, and there are 4 cases (e.g., $ix+j \bmod N = i'x+j' \bmod N$ if $A(\sigma;1,x)$ issued both and $ix+j \bmod N = i'(x+b)+j' \bmod N$ if $A(\sigma;1,x)$ issued (i,j) and $A(\sigma;1,x+b)$ issued (i',j'), etc). In each of these four cases, both algorithms can exploit the collision to jointly solve the discrete logarithm problem using at most $2T$ queries (e.g., in the first case above, $x = (j'-j)\cdot(i-i')^{-1} \bmod N$). According to Theorem 4, the probability of this event is $O(T^2/N) = o(1)$ (by our assumption $T = o(N^{1/2})$), which is negligible. In the following we generally denote collision events by COL.

Most of the analysis below will be conditioned on the event $\overline{COL}$ (whose probability is $1-o(1)$), but we will omit this explicit conditioning for simplicity, while ignoring a negligible factor in the probability calculation.

Lemma 15. *Assume that $T = o(N^{1/2})$. Then, any non-adaptive algorithm A' with $\Pr_{\mathbf{err}}(A',\sigma,[-M,M],T) = \delta$ enables defining a non-adaptive query-restricted algorithm A with $\Pr_{\mathbf{err}}(A,\sigma,[-M,M],T) \leq \delta\cdot(1+o(1))$ such that A only issues restricted queries of the form (i,j) with $i \in \{0,1\}$.*

Proof. First, as analyzed above, a collision event COL' in the algorithm A' occurs with probability $o(1)$ and we condition the analysis on the complementary event $\overline{COL'}$. Essentially, once we have excluded collisions, then each query (i,j) of the algorithm A' can be assigned a domain i such that queries with different domains do not influence each other. Such an algorithm enables defining an algorithm A that only queries a single domain with $i=1$ (in addition to $i=0$),

while preserving the error probability, unless the event COL (which denotes a collision in A) occurs. Overall, we show that conditioned on $\overline{COL'} \cap \overline{COL}$, we have

$$\Pr_{\mathbf{err}}(A, \sigma, [-M, M], T) = \Pr_{\mathbf{err}}(A', \sigma, [-M, M], T) = \delta.$$

Since $\Pr[\overline{COL'} \cap \overline{COL}] = 1 - o(1)$, by lifting the conditioning we get

$$\delta = \Pr_{\mathbf{err}}(A', \sigma, [-M, M], T) \geq \Pr_{\mathbf{err}}(A, \sigma, [-M, M], T) \cdot (1 - o(1))$$

or

$$\Pr_{\mathbf{err}}(A, \sigma, [-M, M], T) \leq \delta \cdot (1 + o(1)),$$

as claimed.

In the following, operations on oracle query indices are performed in $\mathbb{Z}_N$. The algorithm A simply translates the queries of A' using some predefined translation function $tr : \mathbb{Z}_N \times \mathbb{Z}_N \to \mathbb{Z}_N \times \mathbb{Z}_N$. We will define tr such that

$$(i, j)[x] = (i^*, j^*)[x + b] \Leftrightarrow tr((i, j))[x] = tr((i^*, j^*))[x + b], \tag{20}$$

for all $x, i, j, i^*, j^* \in \mathbb{Z}_N$ and $b \in [-M, M]$ (unless a collision occurs in A or A'). This guarantees that $A(\sigma; 1, x)$ and $A(\sigma; 1, x + b)$ have the same joint input distribution as $A'(\sigma; 1, x)$ and $A'(\sigma; 1, x + b)$ and hence A has the same error probability as A' (assuming no collisions).

Assume that we are given some fixed values of domain separators $D^v_u \in \mathbb{Z}_N$ for non-zero $v \in \mathbb{Z}_N$ and $u \in \mathbb{Z}_v$. We define $tr(i, j) = (1, D^i_{j \bmod i} + \lfloor j/i \rfloor)$ for $i \neq 0$ (and $tr(0, j) = (0, j)$ otherwise), where addition is performed in $\mathbb{Z}_N$.

We now need to show that (20) holds. The condition $(i, j)[x] = (i^*, j^*)[x + b]$ is equivalent to $ix + j = i^*(x + b) + j^*$. Assuming $\overline{COL'}$, we have $i = i^*$ and $j = i^*b + j^* = ib + j^*$.

For $i, i^* \neq 0$, the condition $tr((i, j))[x] = tr((i^*, j^*))[x + b]$ is equivalent to

$$D^i_{j \bmod i} + \lfloor j/i \rfloor + x = D^{i^*}_{j^* \bmod i^*} + \lfloor j^*/i^* \rfloor + x + b. \tag{21}$$

We will choose the domain separators such that there are no collisions between queries in different domains of A. Namely, the above equality gives $i = i^*$ and $j \bmod i = j^* \bmod i^*$ (i.e., $D^i_{j \bmod i} = D^{i^*}_{j^* \bmod i^*}$). Thus, (21) is equivalent to

$$\lfloor j/i \rfloor = \lfloor j^*/i^* \rfloor + b = \lfloor j^*/i \rfloor + b. \tag{22}$$

We write $j = k \cdot i + j^*$ (as j, j^* differ by a multiple of i), and thus $\lfloor j/i \rfloor = \lfloor (k \cdot i + j^*)/i \rfloor = \lfloor j^*/i \rfloor + k$. From (22) we obtain $k = b$, namely $j = ib + j^*$, which implies that (20) holds, as required.

We are left with the problem of choosing the domain separators D^v_u. Since $T^2 \ll N$ it is sufficient to choose the domain separators in a (pseudo) random manner in order to avoid collisions between queries in different domains of A (except with negligible probability).[17] Moreover, since A is non-adaptive, its query

[17] In fact, it is sufficient to choose them to be pairwise independent to avoid collisions with high probability.

set is known in advance. Hence, there are at most T possible domain separators that A will use and they can be chosen to avoid such collisions with probability 1. (Note that collisions between queries of type $tr(i,j) = (1, D^i_{j \bmod i} + \lfloor j/i \rfloor)$ (for $i \neq 0$) and $tr(0,j^*) = (0,j^*)$ are still possible (yet occur with negligible probability of $O(T^2/N) = o(1)$)). ■

Query Disjoint Indices We say that a non-adaptive DDL algorithm A has a query disjoint index b if $QE(A,x) \cap QE(A,x+b) = \emptyset$ for any $x \in \mathbb{Z}_N$. We note that A can have many query disjoint indices. We prove the following error probability lower bound on algorithms with a (small) query disjoint index.

Lemma 16. *Any non-adaptive algorithm which is query disjoint on index $b \geq 1$ satisfies $\Pr_{\mathbf{err}}(A,\sigma,[-1,1],T) = \Omega(1/b)$.*

Proof. We prove that $\Pr_{\mathbf{err}}(A,\sigma,[b,b],T) \geq 1/2$. This implies the assertion of the lemma, since

$$\begin{aligned} 1/2 &\leq \Pr_{\mathbf{err}}(A,\sigma,[b,b],T) \leq 4 \cdot \Pr_{\mathbf{err}}(A,\sigma,[-b,b],T) \\ &\leq 4 \cdot 8b \cdot \Pr_{\mathbf{err}}(A,\sigma,[-1,1],T), \end{aligned}$$

where the second inequality follows from Lemma 12 and the third inequality follows from Lemma 13.

To prove that $\Pr_{\mathbf{err}}(A,\sigma,[b,b],T) \geq 1/2$, assume that $QE(A,x) \cap QE(A,x+b) = \emptyset$ and $A(\sigma;1,x) - A(\sigma;1,x+b) = b$. We further assume for the sake of simplicity that A is deterministic, but the proof can be easily extended to randomized algorithms. Since A is non-adaptive, both executions have the same query set $Q(A)$. Given a triplet x, b, σ, we define an encoding σ' that interchanges the encoding values of $QE(A,x)$ and $QE(A,x+b)$, while maintaining all the other values (including those of the constant queries). (We remark that for a randomized algorithm A, the above mapping should be defined between quartets that also involve the randomness string of A.)

More specifically, for every query (i,j) such that $i \neq 0$,[18] we set $\sigma'(i(x+b)+j) = \sigma(ix+j)$ and $\sigma'(ix+j) = \sigma(i(x+b)+j)$. Thus, $A(\sigma';1,x+b) = A(\sigma;1,x)$ and $A(\sigma';1,x) = A(\sigma;1,x+b)$, implying that $A(\sigma';1,x) - A(\sigma';1,x+b) = -b \neq b$.

We prove that $\Pr_{\mathbf{err}}(A,\sigma,[b,b],T) \geq 1/2 = \Omega(1)$ by analyzing the mapping between the triplets x, b, σ and x, b, σ'. Clearly, both σ and σ' are equally likely and it remains to show that the mapping is one-to-one. This indeed holds, as given x, b, σ', we can invert the mapping (i.e. reconstruct σ) by interchanging the encoding values of $QE(A,x)$ and $QE(A,x+b)$. ■

Query Chains Given a query $(1,j)$, we refer to the value j as a query offset. For a non-adaptive algorithm A, we define a query chain of length c as a sequence of $c+1$ query offsets $j, j+1, j+2, \ldots, j+c$ such that for each $k \in \{0,1,\ldots,c\}$,

[18] By Lemma 15 we can assume that $i = 1$, although this assumption is not explicitly required here.

$(1, j+k) \in Q(A)$, while $(1, j+c+1) \notin Q(A)$ and $(1, j-1) \notin Q(A)$ (i.e., the sequence is maximal).

Denote the length of the longest query chain of A by $C(A)$.

Lemma 17. *Any non-adaptive query-restricted algorithm A satisfies*

$$\Pr_{\mathbf{err}}(A, \sigma, [-1, 1], T) \geq \Omega(1/C(A)).$$

Proof. Given a non-adaptive query-restricted algorithm A whose queries (i, j) satisfy $i \in \{0, 1\}$ (as in Lemma 15), we transform it into an algorithm A' that is query disjoint on index $C' \stackrel{\text{def}}{=\!=} C(A) + 1$, while preserving its error probability. Once this is done, by Lemma 16, we have

$$\Pr_{\mathbf{err}}(A, \sigma, [-1, 1], T) = \Pr_{\mathbf{err}}(A', \sigma, [-1, 1], T) = \Omega\left(\frac{1}{C(A)}\right).$$

A is transformed to A' using a static query transformation tr. We partition the (non-constant) queries $(1, j)$ of A into chains (in some arbitrary order) and map each chain into an interval of length C'. The chains are mapped into every second interval, namely, $[0, C'-1], [2C', 3C'-1], [4C', 5C'-1], \ldots, [2kC', (2k+1)C'-1], \ldots$. Since the maximal chain length is $C(A)$, then no chain crosses its interval boundary. Hence, $QE(A, x) \cap QE(A, x+C') = \emptyset$, as these executions query different intervals: the query evaluations of $A(\sigma; 1, x)$ are in $[x+2kC', x+(2k+1)C'-1]$, while the query evaluations of $A(\sigma; 1, x+C')$ are in $[x+(2k+1)C', x+(2k+2)C'-1]$.

Next, we show that the joint input distribution of $A(\sigma; 1, x)$ and $A(\sigma; 1, x+1)$ is preserved, which implies that the error probabilities of A and A' are identical. To show that tr preserves the joint input distribution, it suffices to prove that

$$(1, j)[x] = (1, j^*)[x+1] \Leftrightarrow tr((1, j))[x] = tr((1, j^*))[x+1],$$

for all $x \in \mathbb{Z}_N$ and $(1, j), (1, j^*) \in Q(A)$. Indeed, $(1, j)[x] = (1, j^*)[x+1] \Leftrightarrow j = j^* + 1$, which occurs if and only if $(1, j), (1, j^*)$ are in the same query chain of A. By the properties of tr, this occurs if and only if $tr((1, j)), tr((1, j^*))$ are in the same query chain of A', namely, $tr((1, j))[x] = tr((1, j^*))[x+1]$, as claimed. Finally, we note that the analysis assumes no collisions in A, A', but this event can be neglected as previously specified. ■

Proof of Theorem 6 Now we are ready to present the proof of Theorem 6.

Proof. The assertion of the theorem is an immediate corollary of Lemmas 15 and 17. Given a non-adaptive algorithm A, transform it into a query-restricted algorithm A' using Lemma 15, with a multiplicative loss of $1 + o(1)$ in error probability. Clearly, $C(A') \leq T$, hence by Lemma 17 we have

$$\begin{aligned}\Pr_{\mathbf{err}}(A, \sigma, [-1, 1], T) &\geq \Pr_{\mathbf{err}}(A', \sigma, [-1, 1], T) \cdot 1/(1 + o(1)) = \Omega\left(\frac{1}{C(A')}\right) \\ &= \Omega\left(\frac{1}{T}\right),\end{aligned}$$

concluding the proof. ■

A Generalization of Theorem 6 The above Theorem 6 does not completely render non-adaptive algorithms as inefficient since (for example) it does not rule out the possibility that $\Pr_{\mathbf{err}}(A, \sigma, [-T, T], T) = O(1/T)$ (which is optimal according to Theorem 5). The following theorem states that this is impossible, and non-adaptive algorithms have a linear query-error tradeoff at best.

Theorem 7. *For all $1 \leq M \leq T$, any non-adaptive generic DDL algorithm A satisfies $\Pr_{\mathbf{err}}(A, \sigma, [-M, M], T) = \Omega(M/T)$ given that $T = o(N^{1/2})$ and N is prime. In particular, for $M = T$, $\Pr_{\mathbf{err}}(A, \sigma, [-T, T], T) = \Omega(1)$.*

Note that Theorem 6 is a special case of Theorem 7, obtained for $M = 1$. We chose to present both theorems, as their proofs use entirely different techniques.

Proof. For the sake of simplicity, in this proof we omit the generic group notation $A(\sigma, 1, x)$, and simply refer to $A(x)$ as a function. By Lemma 15, we can assume that A uses only restricted queries which are at a fixed distance from x. We assume that the output of the encoding function σ is completely uniform, which is justified since $T \ll \sqrt{N}$ (i.e., the probability of collisions in σ is negligible).

By Lemma 13, it is sufficient to prove that $\Pr_{\mathbf{err}}(A, \sigma, [-T, T], T) = \Omega(1)$. In fact, we will prove that for any algorithm A, we have $\Pr_{x,b}[A(x + b) - A(x) \neq b] = \Omega(1)$, where $b \in [-2T, 2T]$. This implies Theorem 7 by Lemma 13. Note that we slightly changed the problem formulation which requires proving that $\Pr_{x,b}[A(x) - A(x+b) \neq b] = \Omega(1)$, but this is equivalent since we can simply negate the output of A.

Let w be a complex N-th root of unity. For this proof, we will treat every element e of $\mathbb{Z}_N$ as if it was w^e. Notice that we have

$$2\Pr[A(x^b) - A(x) = b] - 1 \leq \left| \mathbb{E}\left[A(x) w^b \overline{A(x^b)} \right] \right|,$$

since the product in the right hand side is 1 if $A(x^b) - A(x) = b$. We now consider the Fourier expansion of A as a function of $x_{i_1}, \ldots, x_{i_T}$. This just means we write

$$A(x) = \sum_{S \subseteq J} \widehat{A}(S) x^S,$$

where the sum is over all *multisets* of indices S of J, where each index appears at most $N - 1$ times, and x^S is just the monomial obtained by multiplying the corresponding entries of x, with the multiplicities specified by S. (For formal treatment of the Fourier expansion of Boolean functions, we refer the reader to [16].)

A basic property of the Fourier expansion is the Plancherel identity:

$$\mathop{\mathbb{E}}_{x}\left[A(x)\overline{B(x)} \right] = \sum_{S \subseteq J} \widehat{A}(S)\overline{\widehat{B}(S)}.$$

Plancherel's identity implies Parseval's identity $\sum_S |\widehat{A}(S)|^2 = \mathbb{E}_x[|A(x)|^2] = 1$, where the rightmost equality is due to the fact that always $|A(x)| = 1$, as we embedded $\mathbb{Z}_N$ in the complex unit circle.

Hence, our task is to prove

$$\left| \mathop{\mathbb{E}}_{b} \left[w^b \sum_S \widehat{A}(S) \overline{\widehat{A}(S+b)} \right] \right| \leq 1 - \Omega(1),$$

where $S+b$ is a multiset similar to S, but with each element shifted by b (modulo length of x).

First, notice that we may arrange the contribution from $|\widehat{A}(\emptyset)|^2$ to be small, by choosing w so that $\mathbb{E}_b[w^b] \approx 0$, i.e., for example, $w = \exp\left(\frac{2\pi i \lfloor N/2T \rfloor}{N}\right)$.[19]

We are now ready to present the main argument. Consider the following Cauchy-Schwarz bound

$$|\widehat{A}(S)\widehat{A}(S+b)| \leq \frac{1}{2}\left(|\widehat{A}(S)|^2 + |\widehat{A}(S+b)|^2\right) \mathbb{1}\left\{\widehat{A}(S) \cdot \widehat{A}(S+b) \neq 0\right\}. \quad (23)$$

If we temporarily neglect the indicator on the right, we get the inequality

$$\left| \mathop{\mathbb{E}}_{b} \left[w^b \sum_S \widehat{A}(S) \overline{\widehat{A}(S+b)} \right] \right| \leq \left| \sum_{S \neq \emptyset} \frac{1}{2} \mathop{\mathbb{E}}_{b} \left[|\widehat{A}(S)|^2 + |\widehat{A}(S+b)|^2 \right] \right| \leq 1,$$

by trivially using the triangle inequality.

Fix some S. The important observation is that for at least $1/2$ of the values of $b \in [1, 2T]$, we have $\widehat{A}(S) \cdot \widehat{A}(S+b) = 0$ (which gives better understanding for the RHS of (23)). To see this, notice that in order to have $\widehat{A}(T) \neq 0$, we must have $T \subseteq J$. Hence, for any fixed $i \in S$, only for at most T shifts $S+b$ we have $S+b \subseteq J$, as this requires $i+b \in J$, while J is of size T. Since there are $2T$ available shifts, $\frac{1}{2}|\widehat{A}(S)|^2$ is counted only for $1/2$ of the b's. By a symmetric argument, $\frac{1}{2}|\widehat{A}(S+b)|^2$ is counted only $1/2$ of the times. Therefore, we have

$$\left| \mathop{\mathbb{E}}_{x,b} \left[A(x) w^b \overline{A(x^b)} \right] \right| \leq \frac{T}{2T} + o(1),$$

which means $\Pr[A(x^b) = A(x) + b] \leq 3/4 + o(1)$. The $o(1)$ corresponds to the contribution of $|A(\emptyset)|^2 \, \mathbb{E}_b[w^b]$ which we explained that may be assumed to be $o(1)$. This completes the proof. ■

7 Conclusions

In this paper we introduced a new distributed discrete log protocol that achieves an error probability of $O(M \cdot T^{-2})$. We further showed that this protocol is optimal (up to a constant factor) in all prime order group families relevant for homomorphic secret sharing (HSS), unless there is a breakthrough in cryptanalysis of discrete log in a short interval. Moreover, we showed that adaptivity is

[19] Note w must be a primitive root of unity. In our case, this holds trivially since N is prime.

necessary to break the linear error barrier $M \cdot T^{-1}$ in the generic group model. We leave it to future work to implement and optimize the protocol for conversion friendly groups in an HSS scheme and to find additional applications for our new random walk.

Acknowledgements The authors would like to thank Elette Boyle, Niv Gilboa, Yuval Ishai and Yehuda Lindell for discussions and helpful suggestions that improved this work significantly.
This research was supported by the European Research Council under the ERC starting grant agreement n. 757731 (LightCrypt) and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.
The first author was additionally supported by the Israeli Science Foundation through grant No. 573/16.

References

1. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In Simon [21], pages 1–10.
2. D. J. Bernstein and T. Lange. Computing Small Discrete Logarithms Faster. In S. D. Galbraith and M. Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings*, volume 7668 of *Lecture Notes in Computer Science*, pages 317–338. Springer, 2012.
3. D. Boneh, E. Goh, and K. Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In J. Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341. Springer, 2005.
4. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, and M. Orrù. Homomorphic Secret Sharing: Optimizations and Applications. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. 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.
5. E. Boyle, N. Gilboa, and Y. Ishai. Breaking the Circuit Size Barrier for Secure Computation Under DDH. In M. Robshaw and J. 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.
6. E. Boyle, N. Gilboa, and Y. Ishai. Group-Based Secure Computation: Optimizing Rounds, Communication, and Computation. In J. Coron and J. B. Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 163–193, 2017.
7. E. Boyle, N. Gilboa, Y. Ishai, H. Lin, and S. Tessaro. Foundations of Homomorphic Secret Sharing. In A. R. Karlin, editor, *9th Innovations in Theoretical*

Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA, volume 94 of *LIPIcs*, pages 21:1–21:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
8. D. Chaum, C. Crépeau, and I. Damgård. Multiparty Unconditionally Secure Protocols (Extended Abstract). In Simon [21], pages 11–19.
9. B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private Information Retrieval. *J. ACM*, 45(6):965–981, 1998.
10. I. Dinur, N. Keller, and O. Klein. An Optimal Distributed Discrete Log Protocol with Applications to Homomorphic Secret Sharing. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 213–242. Springer, 2018.
11. N. Fazio, R. Gennaro, T. Jafarikhah, and W. E. Skeith III. Homomorphic Secret Sharing from Paillier Encryption. In T. Okamoto, Y. Yu, M. H. Au, and Y. Li, editors, *Provable Security - 11th International Conference, ProvSec 2017, Xi'an, China, October 23-25, 2017, Proceedings*, volume 10592 of *Lecture Notes in Computer Science*, pages 381–399. Springer, 2017.
12. S. D. Galbraith, J. M. Pollard, and R. S. Ruprai. Computing discrete logarithms in an interval. *Math. Comput.*, 82(282):1181–1195, 2013.
13. C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178. ACM, 2009.
14. D. M. Gordon. Discrete Logarithms in *GF(P)* Using the Number Field Sieve. *SIAM J. Discrete Math.*, 6(1):124–138, 1993.
15. A. K. Lenstra and H. W. Lenstra. *The Development of the Number Field Sieve*. Number 1554 in Lecture Notes in Mathematics. Springer, 1993.
16. R. O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.
17. J. M. Pollard. Monte Carlo methods for index computation (mod$p$). *Math. Comput.*, 32(143):918–924, 1978.
18. J. M. Pollard. Kangaroos, Monopoly and Discrete Logarithms. *J. Cryptology*, 13(4):437–447, 2000.
19. R. L. Rivest, L. Adleman, and M. L. Dertouzos. On Data Banks and Privacy Homomorphisms. *Foundations of Secure Computation*, pages 169–179, 1978.
20. V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In W. 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.
21. J. Simon, editor. *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. ACM, 1988.
22. A. C. Yao. Protocols for Secure Computations (Extended Abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164. IEEE Computer Society, 1982.

A Group-Based Homomorphic Secret Sharing Scheme [5]

We describe very briefly the main ideas behind the group HSS scheme of [5]. For more details, refer to the original publication and its extensions [4, 6].

The (2-party) HSS scheme of [5] randomly splits an input w into a pair of shares (w_0, w_1) such that: (1) each share w_i computationally hides w, and (2) there exists a polynomial-time local evaluation algorithm Eval such that for any program P from a given class (branching programs in this case), the output $P(w)$ can be efficiently reconstructed as $\text{Eval}(w_0, P) + \text{Eval}(w_1, P)$.

The scheme works in a multiplicative cyclic group $\mathbb{G}$ of prime order N with a generator g. First, the scheme allows the parties to locally multiply an encrypted input $w \in \mathbb{Z}_N$ with an additively secret-shared value $y \in \mathbb{Z}_N$, such that the result $z = wy$ is shared between the parties. For this purpose, the parties also require an additive share of (cy) (where $h = g^c$ is the public key). The sharing is such that each party $i \in \{0, 1\}$ has a group element g^{z_i} satisfying $g^{z_0} \cdot g^{z_1} = g^z$. We note that all intermediate computation values are bounded by a small integer parameter M.

More specifically, the encryption of w is $(g^r, h^r g^w)$, which is an ElGamal encryption, modified to be additively homomorphic by placing the plaintext in the exponent. The value y (and similarly (cy)) is additively shared as (y_0, y_1) such that $y = y_0 + y_1 \bmod N$ and party i obtains y_i. Given $(g^r, h^r g^w)$, y_i and $(cy)_i$, party i locally computes a share of $z = wy$ as $\gamma_i = (h^r g^w)^{y_i} \cdot (g^r)^{-(cy)_i}$. Thus, $\gamma_0 \cdot \gamma_1 = (h^r g^w)^y \cdot (g^r)^{-(cy)} = g^{(cr+w)y - cry} = g^{wy} = g^z$ as required.

At this stage, g^z is multiplicatively shared by the parties, so they cannot multiply z with a new encrypted input w'. Next, [5] shows how the parties can convert multiplicative shares of g^z into additive shares of z without any interaction via a share conversion procedure. The parties implement the share conversion procedure by collectively solving the distributed discrete log (DDL) problem, described in Section 2.1. Once the parties have an additive sharing of z, they can add it to other additive shares. They can also multiply it with a new encrypted input w' (provided that they also compute an additive share of (cz)).[20]

This allows the parties to perform the following operations on their input: (1) load an input into memory, (2) add the values of two memory locations, (3) multiply a memory location by an input, and (4) output the value of a memory location modulo some integer.[21] Such programs, referred to as restricted multiplication straight-line (RMS) programs, can emulate any branching program of size S by a sequence of $O(S)$ instructions.

[20] An additional difficulty is that cz may be a large number, while the parties are restricted to performing computations on small integers. This problem is overcame by providing an encryption of each input w multiplied by each bit of the secret key c_i, and then applying a linear combination whose coefficients are powers of 2 to the additive shares of the products $c_i yw$, obtaining additive shares of cz.

[21] The parties cannot multiply two memory locations, which would allow evaluation of arbitrary circuits.

B Refined Provable Parameters for Theorem 2

In Theorem 2, we proved that the failure probability of Algorithm 3, namely, $\Pr_{\mathbf{err}}(\text{IteratedRandW}_P, [1,1], T)$, is bounded by $2^{10.2+o(1)}/T^2$. One may wonder whether for small values of T, the $o(1)$ term is actually dominant and makes the result meaningless. It turns out that this is not the case, and on the contrary, for specific values of T one may obtain bounds which are similar to those of Theorem 2 and usually even stronger, using a computer-aided choice of parameters. In order to obtain such bounds, we programmed a routine that uses all the bounds proved in this paper to obtain a bound on $\Pr_{\mathbf{err}}(\text{IteratedRandW}_P, [1,1], T)$, given a set of parameters $SP = \{I, \{L_k\}, \{t_k\}\}$. Then, we optimized the parameters, and obtained for several values of T an upper bound on the probability $\Pr_{\mathbf{err}}(\text{IteratedRandW}_P, [1,1], T)$.

The bounds were used together with their lower order terms, so that the results given below are precise, assuming the the group $\mathbb{G}$ is sufficiently large. Notice the lower order terms are dominant for small T, but the resulting bound is still not much worse than the one claimed in Theorem 2. We emphasize that these bounds are *rigorously provable*, by plugging the choices of parameters given below into the proof of Theorem 2; the computer routine only helps in choosing the parameters.

T	I	$\sim \log_2(t_k)$ $\sim \log_2(L_k)$	$T^2 \cdot \Pr[\mathbf{err}] \leq$
2^{15}	2	12.4, 13.5, 14.0 5.7, 9.6	1440
2^{25}	5	19.2, 20.7, 21.5, 22.3, 23.1, 23.8 7.6, 12.4, 15.8, 18.4, 20.8	755
2^{50}	10	32.1, 36.2, 38.8, ..., 47.3, 48.1, 48.8 9.1, 16.8, ..., 41.5, 43.8, 46.0	555
2^{100}	12	77.9, 80.9, 83.6, ..., 97.3, 98.1, 98.8 30.5, 47.4, ..., 91.5, 93.8, 96.0	555

C An Additional Optimization Used in Part of the Experiments

In this section we describe the third optimization used in part of the experiments. The results obtained using this optimization are described in the rightmost column of Table 1. As can be seen in the table, for all values of T for which computation using only the first two optimizations is feasible, the expected error probability obtained using the third optimization is very close to the probability obtained without it. The advantage of this optimization is that it leads to a much smaller standard deviation, and allows performing the simulation for much larger values of T.

The optimization is based on the fact that Algorithm 3 is composed of several stages, and $\Pr[\mathbf{err}]$ expresses the failure of the parties to synchronize in any of these stages. Roughly speaking, it estimates efficiently the probability of failure in each stage, assuming failure in all previous stages, and then multiplies these probabilities. First we describe the simulation and then we justify its correctness.

For each stage i (starting at $i = 0$), we simulate a specific pattern of jumps of both parties for the i'th stage, and compute the probability p_i of the event that the parties are not synchronized at the end of that stage, given the simulated pattern of jumps. We then uniformly draw c_i^A, c_i^B (that is, the values of the parties at the end of the i'th step), given the pattern of jumps and the assumption that the parties have not synchronized yet. We then repeat the process for the $(i+1)$'th stage, until $i = I$. The simulation outputs the product of probabilities $X = p_0 \cdot p_1 \cdots p_I$. We claim that we have $\mathbb{E}[X] = \Pr[\mathbf{err}]$, which allows us to approximate $\Pr[\mathbf{err}]$ efficiently.

The claim is proved by induction on I, using repeated application of the law of total expectation and the law of total probability, in their conditional version. Recall that the law of total expectation states that if $\{B_n\}_{n=1,2,\ldots}$ is a partition of the entire space, then for each pair of events A, C we have

$$\mathbb{E}[A|C] = \sum_n \mathbb{E}[A|C \cap B_n] \Pr[B_n|C].$$

The law of total probability asserts the same, with expectation replaced by probability.

In our case, recall that for each $0 \leq i \leq I$, $\mathbf{err}_i$ denotes the event that the parties fail to synchronize until the end of stage i, and c_i^A, c_i^B denote the positions of the parties at the end of stage i. Let us denote by $\{E_i^j\}_{j=1,2,\ldots}$ the possible pairs of positions (c_i^A, c_i^B).

For $I = 0$ (i.e., if Algorithm 3 consists of a single stage), we clearly have $\mathbb{E}[X] = \mathbb{E}[p_0] = \Pr[\mathbf{err}_0] = \Pr[\mathbf{err}]$. Assume that for all $j \leq I - 1$ we have $\mathbb{E}[p_0 \cdot p_1 \cdot \ldots \cdot p_j] = \Pr[\mathbf{err}_j]$, and consider stage I.

We have

$$\Pr[\mathbf{err}] = \Pr[\mathbf{err}_I] = \Pr[\mathbf{err}_{I-1}] \Pr[\mathbf{err}_I|\mathbf{err}_{I-1}]$$

By the law of total probability,

$$\Pr[\mathbf{err}_I|\mathbf{err}_{I-1}] = \sum_j \Pr[\mathbf{err}_I|E_{I-1}^j \wedge \mathbf{err}_{I-1}] \Pr[E_{I-1}^j|\mathbf{err}_{I-1}].$$

By the definition of the p_i's, we have

$$\Pr[\mathbf{err}_I|E_{I-1}^j \wedge \mathbf{err}_{I-1}] = \mathbb{E}[p_I|E_{I-1}^j],$$

and so by the law of total expectation,

$$\begin{aligned}\Pr[\mathbf{err}_I|\mathbf{err}_{I-1}] &= \sum_j \Pr[\mathbf{err}_I|E^j_{I-1} \wedge \mathbf{err}_{I-1}] \Pr[E^j_{I-1}|\mathbf{err}_{I-1}] \\ &= \sum_j \mathbb{E}[p_I|E^j_{I-1} \wedge \mathbf{err}_{I-1}] \Pr[E^j_{I-1}|\mathbf{err}_{I-1}] \\ &= \mathbb{E}[p_I].\end{aligned}$$

Hence, using the induction hypothesis, we obtain

$$\begin{aligned}\Pr[\mathbf{err}] &= \Pr[\mathbf{err}_{I-1}] \Pr[\mathbf{err}_I|\mathbf{err}_{I-1}] \\ &= \mathbb{E}[p_0 \cdot p_1 \cdot \ldots \cdot p_{I-1}] \mathbb{E}[p_I] \\ &= \mathbb{E}[p_0 \cdot p_1 \cdot \ldots \cdot p_I],\end{aligned}$$

as asserted.

We note that an important difference between the first two optimizations, described in Section 4, and the third optimization described above, is the type of random variables we obtain. The variables outputted from a simulation based solely on the first two optimizations are Bernoulli variables, indicating the failure of the parties to synchronize, and we seek to compute their expectation. In contrast, the distribution of the variables outputted from a simulation using the third optimization is not clear, but still has expectation equal to $\Pr[\mathbf{err}]$. Therefore, we performed some simulations also without this optimization.

In both cases we approximate $\Pr[\mathbf{err}]$ by the empirical expectation of the samples drawn with the described simulations. In addition, we estimate the standard deviation (σ) of our approximation of $\Pr[\mathbf{err}]$ using the *empirical variance*

$$\frac{\sum_{i<j}(X_i - X_j)^2}{n^3 - n^2},$$

where $X_1, \ldots, X_n$ are independent samples from the corresponding distribution, as this is the standard way to estimate the variance of a random variable whose distribution is not known a-priori, based on samples.

D Variants and Optimizations of Algorithm 3

D.1 Error Flag

The DDL protocols of [4, 6] were adapted to raise a flag in case there was a potential error in the protocol (namely, the parties fail to synchronize). This feature was important for some of the HSS applications. In the tweaked protocol, A simulates B's algorithm for all its potential locations (i.e., $b \in [-M, M]$). This can be performed very efficiently for small M by computing about M additional group elements. A then raises a flag if there is a potential disagreement (i.e., an error) in one of these simulations.

Another option (described in [11]) is for each party to maintain a footprint (e.g., a hash) of all the final group elements it computes during the DDL executions of an RMS program. One can then check if an error occurred during one of the DDL executions by comparing these footprints.

Our iterated random walk algorithm can be adapted to support each of these two modifications with a small overhead. In particular, the expected overhead in the first modification is small since each iteration in all simulations can be performed by A in parallel. Note that almost all simulations are likely to agree on the same group element after the first iteration, which executes the basic algorithm and requires computing only about M additional consecutive group elements for all simulations. Hence, the number of simulations that A has to test in the remaining iterations drops sharply.

D.2 Practical Considerations

In the HSS implementations of [4, 6], the authors used a specific type of groups (called "conversion friendly" groups) for which their share conversion procedure can be implemented very efficiently. In such groups, multiplication by g is very efficient,[22] and moreover, given g^x it is possible to scan very quickly a consecutive interval $g^{x+1}, g^{x+2}, \ldots, g^{x+c}$ (for some value of c) to determine whether one of these elements has a specific pattern (e.g., is the minimal element in this set). In the following, we refer to this procedure as a *consecutive scan*. Consequently, in [4], the authors were able to perform $T \approx 5 \cdot 10^9$ "conversion operations" per second, which gives an error probability of roughly $2 \cdot 1/5 \cdot 10^{-9}$ using their algorithm. On the other hand, the authors reported that using an elliptic curve group (offering a similar level of security), they could only perform about $T = 4 \cdot 10^6$ group operations per second. This gives an error probability that is higher by a factor of about 1000 compared to the conversion friendly group.

If we run our algorithm on a similar elliptic curve group, according to Table 1, we expect the error probability to become roughly $400 \cdot T^{-2} = 400 \cdot (4 \cdot 10^6)^{-2} \approx 1/4 \cdot 10^{-10}$, which is lower than the probability obtained with a conversion friendly group in [4] by a factor of about 16. We note that an additional benefit of elliptic curve groups is that they reduce the size of the HSS ciphertexts.

We would like to further optimize our algorithm in conversion friendly groups, but its direct application would not yield a similar benefit as in [4, 6] since (besides the first iteration) it does not compute consecutive group elements. Nevertheless, we can tweak our algorithm to benefit from conversion friendly groups by combining our random walk iterations with efficient consecutive scans as above, resulting in "alternating" random walks. Optimizing the parameters for this algorithm in practice is quite involved and we leave it to future work. Below, we sketch the algorithm and its analysis at a high level.

[22] Conversion friendly groups are subgroups of $\mathbb{Z}_p^*$ of a prime order q, where $p = 2q + 1$ is a prime that is close to a power of 2, and $g = 2$ is a generator. Multiplying a group element h by the generator g can be very efficiently performed by shifting h by one bit to the left, and adding the difference between p and the closest power of 2 in case the removed leftmost bit is 1.

We combine the iterated random walk of Algorithm 3 with basic Algorithm 1 as follows: for parameters T_1, T_2, Algorithm 3 is viewed as an outer algorithm that operates on blocks of size T_1 (in the original algorithm $T_1 = 1$) and makes T_2 invocations of the basic Algorithm 1, where each such inner invocation makes T_1 consecutive queries to its block. Thus, the combined algorithm makes a total of $T_1 \cdot T_2$ queries.

More specifically, in each step, the combined algorithm moves to the group element which has the lowest value under the mapping ϕ among the next T_1 elements. It then computes the step function ψ on the chosen element, where its output defines the number of blocks (of size T_1) to jump over in this step (the domain and range of ψ are the same as in the standard Algorithm 3 with T_2 queries). Calculation shows that for $M = 1$, the combined algorithm has error probability of $\delta = O(T_1^{-1} \cdot T_2^{-2})$ (for the cases of $T_1 = 1$ and $T_2 = 1$ this is directly implied by our analysis of algorithms 1 and 3, respectively).

Let us assume that in conversion friendly groups, scanning a block of size T' using Algorithm 1 requires the same amount of time as performing a single jump in the iterated random walk of Algorithm 3 (the exact value of T' should be determined by experiments). Furthermore, assume that we can make T_u jumps in Algorithm 3 in a fixed time unit, or alternatively, scan T_u blocks of size T' using Algorithm 1. Since the combined algorithm makes the same number of consecutive scans and jumps, we can set the block size as $T_1 = T'$, i.e., we make $T_u/2$ jumps and scan $T_u/2$ blocks of size T' in a time unit, obtaining $\delta = O(T'^{-1} \cdot T_u^{-2})$ (precise optimization will set T_1 more accurately in order to optimize the constants). In other words, we get an advantage of $O(T'^{-1})$ in error probability compared to the iterated random walk of Algorithm 3.