Generic Hardness of the Multiple Discrete Logarithm Problem

Aaram Yun

Ulsan National Institute of Science and Technology (UNIST)
Republic of Korea
aaramyun@unist.ac.kr

Abstract. We study generic hardness of the multiple discrete logarithm problem, where the solver has to solve n instances of the discrete logarithm problem simultaneously. There are known generic algorithms which perform $O(\sqrt{np})$ group operations, where p is the group order, but no generic lower bound was known other than the trivial bound. In this paper we prove the tight generic lower bound, showing that the previously known algorithms are asymptotically optimal. We establish the lower bound by studying hardness of a related computational problem which we call the search-by-hyperplane-queries problem.

Keywords: multiple discrete logarithm, search-by-hyperplane-queries, generic group model

1 Introduction

Multiple discrete logarithm problem. Let G be a cyclic group of order p, where p is prime, and let g be a generator of G. Then the Discrete Logarithm (DL) problem is defined as follows: given (G, p, g, g^α) for a uniform random $\alpha \xleftarrow{\$} \mathbb{Z}_p$, find out α.

Similarly, the Multiple Discrete Logarithm (MDL) problem is: given $(G, p, g, g^{\alpha_1}, \ldots, g^{\alpha_n})$, for independently chosen uniform random elements $\alpha_1, \ldots, \alpha_n \xleftarrow{\$} \mathbb{Z}_p$, find out $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$.

The discrete logarithm problem (and related variants like the Diffie-Hellman problem) is used for many cryptographic constructions and its hardness was studied widely. On the other hand, as far as we know, there are no cryptographic constructions whose security are based on the multiple discrete logarithm problem.

Still, the multiple discrete logarithm problem is relevant in the context of standard curves in the elliptic curve cryptography. Since generating good elliptic curves is rather computationally expensive, some standards like NIST's FIPS 186 [NIS13] recommend using a few standard curves to instantiate cryptographic schemes. Hence, in such a setting, we naturally have to consider the multiple discrete logarithm problem. Hitchcock et al. [HMCD04] analyzed efficiency of algorithms solving the multiple discrete logarithm problem to see how using such a standard curve affects security.

Moreover, some cryptographic constructions *require* a user to solve 'small' discrete logarithm problems: either the group order p is small, or the exponent α is chosen from a small subset $I \subseteq \mathbb{Z}_p$. One such example is the Boneh-Goh-Nissim homomorphic encryption [BGN05], where in order to decrypt a ciphertext, a user has to first compute g^m from the given ciphertext and then solve the discrete logarithm to recover the message m. Another example is the Maurer-Yacobi identity-based encryption [MY96]. Their construction uses a *trapdoor discrete logarithm group*, where the discrete logarithm problem is feasible to a user who has the trapdoor information, while hard for those who do not. They achieve this by using a composite-order group, and then the trapdoor information is the factorization of the group order. A user who has the factorization can solve DL on small groups so the discrete logarithm problem is feasible, but an adversary has to solve the DL problem in a large group. For these cases, efficient algorithms for solving DL is crucial, and for example, Bernstein and Lange [BL12] showed how to speed up the solution of the discrete logarithm problem via precomputation. When considered as a whole, this becomes an algorithm for solving the multiple discrete logarithm problem.

Generic group model. In general, hardness of a cryptographic problem based on a group does not depend solely on the isomorphism class of the underlying group. For example, while we believe that, if we carefully choose an elliptic curve and a subgroup G of prime order p on it, then the discrete logarithm problem on G would be difficult, we also know that the same problem is trivial on the additive group $\mathbb{Z}_p$ which is nonetheless isomorphic to G. What is important is how the same isomorphism class is encoded to a concrete 'representation'. When $\xi : \mathbb{Z}_p \to \{0,1\}^t$ is an injective function, we say that ξ is an *encoding* of the group $\mathbb{Z}_p$. In such a case, we may define $G := \xi(\mathbb{Z}_p)$, and make G into a group by giving G the unique group structure induced from the bijection $\xi : \mathbb{Z}_p \xrightarrow{\sim} G$. Conversely, we can see that any concrete cyclic group with prime order p should come from such an encoding $\xi : \mathbb{Z}_p \to \{0,1\}^t$ together with functions $\mu : \{0,1\}^t \times \{0,1\}^t \to \{0,1\}^t$, $\iota : \{0,1\}^t \to \{0,1\}^t$ such that $\mu|_{G\times G}$ and $\iota|_G$ give multiplication and inversion on G, respectively.

Also, a sophisticated algorithm may analyze and exploit structures of such an encoding to solve group-based computational problems. Naturally, such an algorithm is specific to that particular encoding. On the other hand, there are many 'generic' algorithms which are agnostic to the particular encoding used. One such example is the Baby-Step-Giant-Step algorithm for solving the discrete logarithm problem: BSGS algorithm does not assume anything about the group encoding, except that it is indeed a group encoding, therefore it works for any cyclic group, even though better algorithms exist for some specific groups.

'Generic hardness' of a cryptographic problem, that is, hardness against such generic algorithms, was studied for many group-based cryptographic problems. While a proof of generic hardness cannot really replace serious cryptanalyses for such a problem, at least it serves as a sanity check, in the sense that if a problem can be solved efficiently even by a generic algorithm, certainly one cannot base cryptographic constructions on such an easy problem. Also, for example on elliptic curves, so far no better non-generic algorithms are known.

To analyze such generic algorithms, the generic group model was proposed by Nechaev [Nec94] and Shoup [Sho97]. In the generic group model, to ensure that a generic algorithm cannot exploit the encoding of a group, a *random encoding*, an encoding $\xi : \mathbb{Z}_p \to \{0,1\}^t$ which is uniform randomly chosen from the set of all injections $\mathbb{Z}_p \hookrightarrow \{0,1\}^t$, is used. Since in such a case we cannot expect any efficient algorithms for group laws, the group laws are given by oracles: the algorithm makes oracle queries by giving encodings of group elements like $\xi(\alpha), \xi(\beta)$, and the oracle returns the result of multiplication or division of these elements in encoded form. In the generic group model, we consider the query complexity of an algorithm to measure its efficiency.

Generic algorithms for DL and MDL problems. Shoup [Sho97] analyzed generic hardness of the discrete logarithm problem. He showed that any generic DL solver which makes at most q queries to the group law oracles have the success probability at most $O(q^2/p)$. In other words, any generic DL solver with some constant success probability should make at least $\Omega(\sqrt{p})$ queries.

As explained before, there are generic algorithms for DL with asymptotically tight matching upper bounds. The Baby-Step-Giant-Step algorithm is an example, and Pollard's rho algorithm is another. Both algorithms perform $O(\sqrt{p})$ group operations. And this gives us a trivial generic algorithm for solving MDL: simply repeat such an asymptotically optimal generic algorithm n times, where n is the total number of DL instances. The total complexity would be $O(n\sqrt{p})$.

In fact, there is a better generic algorithm for MDL. Kuhn and Struik [KS01] extended Pollard's rho to a generic algorithm solving MDL. Their algorithm performs $O(\sqrt{np})$ group operations.

On the other hand, as far as we know, precise generic hardness of MDL is not known. Clearly, solving n DL instances would be at least as hard as solving one single DL instance, therefore Shoup's lower bound $\Omega(\sqrt{p})$ applies here. Kuhn and Struik [KS01] conjectured that the tight lower bound would be $\Omega(\sqrt{np})$, but this has never been proved yet. This means that even the (admittedly unlikely) possibility of a generic algorithm solving n DL instances within $O(\sqrt{p})$, independent of n, is not yet eliminated.

In this paper, we show that the conjecture of Kuhn and Struik is indeed correct: any generic algorithm solving MDL with constant success probability should make at least $\Omega(\sqrt{np})$ queries to the group law oracles.

Search-by-Hyperplane-Queries problem. We establish the generic lower bound of MDL by analyzing a closely related problem, which we call Search-by-Hyperplane-Queries (SHQ) problem. In the SHQ problem, a uniform random point $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$ of the n-dimensional affine space $\mathbb{Z}_p^n$ is hidden, and the goal of the solver is to find the point $\vec{\alpha}$. Of course, the success probability of any unaided solver is at most $1/p^n$. Therefore, we allow any solver to make adaptive *hyperplane queries*. Recall that an affine hyperplane $H \subseteq \mathbb{Z}_p^n$ can be described by an equation of form $a_1X_1 + \cdots + a_nX_n = b$, where $a_1, \ldots, a_n, b \in \mathbb{Z}_p$. A hyperplane query is asked by specifying a hyperplane H via the coefficients $a_1, \ldots, a_n, b$, and the intended meaning of the query is 'is $\vec{\alpha} \in H$?' A SHQ solver may make a series of adaptive hyperplane queries, and use the information gained by such queries to find the hidden point $\vec{\alpha}$.

We are going to show that any SHQ solver which makes at most q hyperplane queries has success probability at most $O((eq/np)^n)$, where e is the base of the natural logarithm. Therefore, any SHQ solver with some constant success probability should make $\Omega(np)$ queries. Then, we are going to show that this lower bound for the SHQ problem implies the $\Omega(\sqrt{np})$ lower bound for the MDL problem.

2 Multiple discrete logarithm problem in the generic group model

2.1 Generic group model

Let p be a prime number, and let $\xi : \mathbb{Z}_p \to \{0,1\}^t$ be a *random encoding* of $\mathbb{Z}_p$, that is, a uniform randomly chosen function among all injective functions of form $\mathbb{Z}_p \to \{0,1\}^t$ for some t satisfying $t \geq \log_2 p$. We define the *group law oracle* μ as the oracle satisfying the following:

$$\mu(b, \xi(\alpha), \xi(\beta)) = \xi(\alpha + (-1)^b\beta \bmod p),$$

where $b \in \{0,1\}$ is a bit indicating whether multiplication or division is intended.

In the *generic group model*, we consider the generic algorithm, which is a probabilistic algorithm A which is initially given a list of group elements $\xi(\beta_1), \ldots, \xi(\beta_k)$, encoded by the random encoding ξ. Also, while running, the algorithm A can make group law queries to the oracle μ. Finally A halts with an output. Note that ξ is never explicitly given to A, but only implicitly via the initial input and the group law oracles.

2.2 Multiple discrete logarithm problem

Let G be a cyclic group of order p, where p is prime, and let g be a generator of G. Also, let n be an integer. We require that n is unbounded and $n = o(p)$: formally, we consider a family of such numbers, so that there is a security parameter λ, and both n and p are functions of λ, and $n(\lambda) \to \infty$ and $n(\lambda)/p(\lambda) \to 0$, as $\lambda \to \infty$.

Then the *Multiple Discrete Logarithm (MDL) problem* is: given $(G, p, g, g^{\alpha_1}, \ldots, g^{\alpha_n})$, for independently chosen uniform random elements $\alpha_1, \ldots, \alpha_n \xleftarrow{\$} \mathbb{Z}_p$, find out $(\alpha_1, \ldots, \alpha_n)$.

We consider the MDL problem in the generic group model. Hence, for a generic algorithm A, we define $\mathbf{Adv}^{\mathsf{mdl}}_{p,n}(A)$, the *advantage* of A in solving the MDL problem as

$$\mathbf{Adv}^{\mathsf{mdl}}_{p,n}(A) = \mathbf{Pr}[A^{\mu}(p, \xi(1), \xi(\alpha_1), \ldots, \xi(\alpha_n)) = (\alpha_1, \ldots, \alpha_n)],$$

where the probability is over the random choice of ξ, $\alpha_1, \ldots, \alpha_n$, and the internal randomness of A.

For any generic MDL solver A, let us say that A *solves MDL with constant advantage* if there exists some constant $c > 0$ such that

$$\mathbf{Adv}^{\mathsf{mdl}}_{p,n}(A) \geq c,$$

for any value of the security parameter λ.

3 Search-by-Hyperplane-Queries problem

In this section, we describe the Search-by-Hyperplane-Queries (SHQ) problem. Let p be a prime number and $\mathbb{Z}_p^n$ be the n-dimensional affine space over the finite field $\mathbb{Z}_p$. As in the MDL problem, we assume that n is unbounded and $n = o(p)$.

Let $X_1, \ldots, X_n$ be the canonical coordinate functions of $\mathbb{Z}_p^n$. Then, an affine hyperplane H in $\mathbb{Z}_p^n$ can be written by a formula of form $a_1X_1 + \cdots + a_nX_n = b$ for some $a_1, \ldots, a_n, b \in \mathbb{Z}_p$, with $a_i \neq 0$ for some i. Sometimes we represent such a hyperplane H by the linear function $a_1X_1 + \cdots + a_nX_n - b$, or even simply by the tuple $(a_1, \ldots, a_n, b)$.

Let $\vec{\alpha} \in \mathbb{Z}_p^n$ be a point in the affine space. We define

$$\mathsf{H}(\vec{\alpha}, H) := \begin{cases} 1 & \text{if } \vec{\alpha} \in H, \\ 0 & \text{otherwise.} \end{cases}$$

The SHQ problem is as follows: pick a uniform random point $\vec{\alpha}$ of $\mathbb{Z}_p^n$. The goal of the problem is to correctly guess the hidden point $\vec{\alpha}$. Without anything else, the probability of correct guess is p^{-n}. Therefore, up to some q adaptive hyperplane queries are allowed: a solver for this problem is allowed to submit up to q hyperplane queries $H_1, \ldots, H_q$ adaptively, and for each such query, the result $\mathsf{H}(\vec{\alpha}, H_i)$ is given. In other words, the solver is given the hyperplane query oracle $\mathsf{H}(\vec{\alpha}, \cdot)$.

For a SHQ solver A, we define $\mathbf{Adv}_{p,n}^{\mathsf{shq}}(A)$, the advantage of A in solving SHQ, as

$$\mathbf{Adv}_{p,n}^{\mathsf{shq}}(A) = \mathbf{Pr}[A^{\mathsf{H}(\vec{\alpha},\cdot)}(p, n) = \vec{\alpha}],$$

where the probability is over the random choice of $\vec{\alpha}$ and the internal randomness of A.

For any SHQ solver A, let us say that *A solves SHQ with constant advantage* if there exists some constant $c > 0$ such that

$$\mathbf{Adv}_{p,n}^{\mathsf{shq}}(A) \geq c,$$

for any value of the security parameter λ.

Worst-case SHQ. We may also consider the worst-case version of the SHQ problem: instead of searching for the uniform randomly chosen $\vec{\alpha}$ with constant advantage, the worst-case SHQ problem is to find any instance $\vec{\alpha} \in \mathbb{Z}_p^n$. Formally, we say that a generic algorithm A *solves SHQ in the worst case within q queries*, if for any $\vec{\alpha} \in \mathbb{Z}_p^n$, $A^{\mathsf{H}(\vec{\alpha},\cdot)}(p, n)$ always outputs $\vec{\alpha}$ after at most q queries.

Example 1. *Here we exhibit a very simple, 'brute-force' SHQ solver. Identify $\mathbb{Z}_p$ with $\{0, 1, \ldots, p-1\}$, and consider hyperplanes of form $X_i = j$, where $i = 1, \ldots, n$, and $j = 1, \ldots, p-1$. There are total $n(p-1)$ such hyperplanes, and we see that non-adaptive hyperplane queries for these $q := n(p-1)$ hyperplanes are enough to correctly find any $\vec{\alpha}$: let $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)$. For any i, if $\mathsf{H}(\vec{\alpha}, X_i = j) = 1$ for some $j = 1, \ldots, p-1$, then $\alpha_i = j$. On the other hand, if $\mathsf{H}(\vec{\alpha}, X_i = j) = 0$ for all $j = 1, \ldots, p-1$, then clearly $\alpha_i = 0$. So in this way the brute-force solver completely determines all coordinates of $\vec{\alpha}$.*

While the above brute-force solver looks very trivial, it turns out that it is actually optimal, by Theorem 2 at Section 5.

4 Relationship between the two problems

In this section, we show that MDL and SHQ are closely related, and any hardness result for SHQ immediately produces a hardness result for MDL.

Theorem 1. *Let A be any generic MDL solver which makes at most q queries. Then, using A, it is possible to construct a SHQ solver B which makes at most $(q+n)(q+n+1)/2$ queries, and satisfying*

$$\mathbf{Adv}_{p,n}^{\mathsf{shq}}(B) \geq \mathbf{Adv}_{p,n}^{\mathsf{mdl}}(A).$$

Proof. We describe how B works. First B receives (p, n) as the input, and B also has access to the oracle $\mathsf{H}(\vec{\alpha}, \cdot)$, for a uniform randomly chosen $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n) \xleftarrow{\$} \mathbb{Z}_p^n$. For convenience, let us define $\alpha_0 := 1$. The solver B has to simulate a random encoding $\xi : \mathbb{Z}_p \to \{0,1\}^t$ for A. To do this, B maintains two sequences, $\{s_i\}_i$ and $\{L_i\}_i$, where $s_i \in \{0,1\}^t$ are random bitstrings generated by B and given to A as simulated output of the encoding function ξ, and L_i are linear functions of form $L_i(X_1, \ldots, X_n) = a_1 X_1 + \cdots + a_n X_n + b \in \mathbb{Z}_p[X_1, \ldots, X_n]$. The idea is to simulate the random encoding ξ, by pretending $s_i = \xi(L_i(\vec{\alpha}))$ for $(s_i, L_i) \in T$.

- Initialization: Here B prepares the simulation of the initial input to A: B chooses $s_0 \xleftarrow{\$} \{0,1\}^t$, and defines $L_0 := 1$. Next, B chooses $s_1, \ldots, s_n$ recursively as follows: when choosing s_i, if there is some $j < i$ with $\mathsf{H}(\vec{\alpha}, X_i = X_j) = 1$ then B picks smallest such j and defines $s_i := s_j$. Otherwise, B chooses $s_i \xleftarrow{\$} \{0,1\}^t \setminus \{s_0, \ldots, s_{i-1}\}$. And, L_i is defined as X_i. Let ctr be n. Finally, B runs $A(p, s_0, s_1, s_2, \ldots, s_n)$.
- Queries: when A makes a query[1] $\mu(s_i, s_j, b)$ for some $0 \le i, j \le ctr$ and $b \in \{0,1\}$, B increments $ctr \leftarrow ctr + 1$, then defines s_{ctr} and L_{ctr} as follows: L_{ctr} is simply defined as $L_i + (-1)^b L_j$. Now, if there is $k < ctr$ with $\mathsf{H}(\vec{\alpha}, L_{ctr} = L_k) = 1$, then B picks the smallest such k and defines $s_{ctr} := s_k$. Otherwise, B randomly picks $s_{ctr} \xleftarrow{\$} \{0,1\}^t \setminus \{s_0, \ldots, s_{ctr-1}\}$. Finally, B returns s_{ctr} as the answer to the query.
- Output: eventually, A halts with output $\vec{\beta} = (\beta_1, \ldots, \beta_n) \in \mathbb{Z}_p^n$. B then also outputs $\vec{\beta}$ and halts.

Now, let us analyze the SHQ solver B. At the initialization phase, B can choose s_i after making i hyperplane queries; so B makes $1 + \cdots + n = n(n+1)/2$ hyperplane queries up to this point. Similarly, to determine s_{ctr}, B has to make ctr hyperplane queries. In total, the number of hyperplane queries B makes is bounded by

$$\begin{aligned}\frac{n(n+1)}{2} + \sum_{ctr=n+1}^{n+q} ctr &= \frac{n(n+1)}{2} + nq + \frac{q(q+1)}{2} \\ &= \frac{n^2 + n + q^2 + q + 2nq}{2} \\ &= \frac{(q+n)(q+n+1)}{2}.\end{aligned}$$

Next we have to show that

$$\mathbf{Adv}^{\mathsf{shq}}_{p,n}(B) \ge \mathbf{Adv}^{\mathsf{mdl}}_{p,n}(A).$$

In fact, we will show that $\mathbf{Adv}^{\mathsf{shq}}_{p,n}(B) = \mathbf{Adv}^{\mathsf{mdl}}_{p,n}(A)$. For this, we need only to show that the simulated input $(p, s_0, s_1, \ldots, s_n)$ given to A has the same distribution as in the original generic MDL problem, and also the simulated group law oracle has the same distribution as in the original generic MDL problem. Let $\xi : \mathbb{Z}_p \to \{0,1\}^t$ be a random encoding, and let $s'_i := \xi(\alpha_i)$ for $i = 0, 1, \ldots, n$, and let $s'_{n+1}, s'_{n+2}, \ldots$ be the sequence of bitstrings which would be given as the answers to the oracle queries made by A, when A is engaged in the real MDL game with ξ. Finally, let $\alpha_i := \xi^{-1}(s'_i)$ for $i = n+1, n+2, \ldots$. Then, we need only to show the following: the random variables s_{ctr} and s'_{ctr} are identically distributed for any $ctr \in \{1, \ldots, q+n\}$, conditioned on the event that

$$s_i = s'_i \text{ and } \alpha_i = L_i(\vec{\alpha}), \quad \text{for all } i = 0, 1, 2, \ldots, ctr - 1.$$

Let us prove this only for $ctr > n$: the case for $s_0, \ldots, s_n$ can be done similarly. Suppose that the group law query of A is $\mu(s_i, s_j, b)$ when determining the bitstring s_{ctr}. Then, s'_{ctr} is easy to compute: $s'_i = \xi(\alpha_i), s'_j = \xi(\alpha_j)$, so $s'_{ctr} = \xi(\alpha_i + (-1)^b \alpha_j)$. Also, $\alpha_{ctr} = \xi^{-1}(s'_{ctr}) = \alpha_i + (-1)^b \alpha_j = L_i(\vec{\alpha}) + (-1)^b L_j(\vec{\alpha}) = (L_i + (-1)^b L_j)(\vec{\alpha}) = L_{ctr}(\vec{\alpha})$. We need to compare this s'_{ctr} with s_{ctr} computed by the algorithm B.

[1] Here we may assume that μ never makes group law queries using bitstrings outside of s_i, because B may ensure that A can guess bitstrings in $\xi(\mathbb{Z}_p)$ only with negligible probability, by sufficiently enlarging the bit length t.

– When there is no $k < ctr$ with $H(\vec{\alpha}, L_{ctr} = L_k) = 1$: in this case, we have $s_{ctr} \xleftarrow{\$} \{0,1\}^t \setminus \{s_0, \ldots, s_{ctr-1}\}$. But, this means that $L_{ctr}(\vec{\alpha}) \neq L_k(\vec{\alpha})$, that is, $\alpha_{ctr} \neq \alpha_k$ for $k = 0, \ldots, ctr-1$. So $s'_i = \xi(\alpha_{ctr})$ is uniformly distributed on $\{0,1\}^t \setminus \{\xi(\alpha_0), \ldots, \xi(\alpha_{ctr-1})\}$. Since $s_i = s'_i = \xi(\alpha_i)$ by assumption, we see that s_{ctr} and s'_{ctr} are identically distributed in this case.
– Otherwise: let k be the smallest index such that $H(\vec{\alpha}, L_{ctr} = L_k) = 1$. Then s_{ctr} is defined to be s_k. On the other hand, this means that $L_{ctr}(\vec{\alpha}) = L_k(\vec{\alpha})$, in other words $\alpha_{ctr} = \alpha_k$, so $s'_{ctr} = \xi(\alpha_{ctr}) = \xi(\alpha_k) = s'_k$. Since we have $s_k = s'_k$ by assumption, we see that s_{ctr} and s_k are in fact the same in this case.

Hence, in both cases, we see that s_{ctr} and s'_{ctr} are identically distributed. Therefore the theorem follows. □

5 Query complexity of the SHQ problem

In this section, we analyze the complexity of the SHQ problem. In fact, we are going to analyze both the worst-case version and the average-case version.

5.1 Pointless queries

For technical reasons which will soon become obvious, we need to define the notion of pointless queries. Let us define a hyperplane query H *pointless*, if it is possible to know that the return value $\mathsf{H}(\vec{\alpha}, H)$ should be 1 before making the query, based on the return values for the previous hyperplane queries made: for example, if the solver A previously made a query H and received the answer $\mathsf{H}(\vec{\alpha}, H) = 1$, then making the same query H again will definitely give the same answer 1. Another example is that, if A previously made $p-1$ queries $X_1 = j$ for $j = 1, \ldots, p-1$ and received answer $\mathsf{H}(\vec{\alpha}, X_1 = j) = 0$ for all $j = 1, \ldots, p-1$, then the A can deduce that $\mathsf{H}(\vec{\alpha}, X_1 = 0) = 1$, so the hyperplane query $X_1 = 0$ is pointless. In general, suppose so far A made $q = r + s$ hyperplane queries $H_1, \ldots, H_r$, $H'_1, \ldots, H'_s$, and assume that $\mathsf{H}(\vec{\alpha}, H_i) = 1$ for $i = 1, \ldots, r$, and $\mathsf{H}(\vec{\alpha}, H'_j) = 0$ for $j = 1, \ldots, s$. Then the information given by the answers to the queries is exactly

$$\vec{\alpha} \in \bigcap_{i=1}^{r} H_i \setminus \bigcup_{j=1}^{s} H'_j.$$

Hence, we may formally define a hyperplane query H made at this point as *pointless* if

$$\bigcap_{i=1}^{r} H_i \setminus \bigcup_{j=1}^{s} H'_j \subseteq H.$$

Note that it is possible to determine if H is pointless or not algorithmically. Since we consider only query complexity of solvers, this does not even have to be efficient.

Remark 1. While it is possible to extend the definition of pointless queries to include all queries which are destined to return 0 as the answer, but for technical reasons we define pointless queries only as above.

5.2 Worst-case SHQ

Theorem 2. *Any worst-case SHQ solver should make at least $n(p-1)$ queries.*

Proof. Let A be a worst-case SHQ solver. We show that, without loss of generality, we may assume that A never asks pointless queries. Suppose that A is a solver which may ask pointless queries. Then, we construct a solver B as follows: B runs A internally, and eventually outputs A's output. When A asks a hyperplane query H, B first determines if it is pointless or not. If it is pointless, then B replies with 1. If it is not pointless, then B makes the same oracle query, receives the

answer bit b, and returns the same bit b to the solver A. So, B is a worst-case SHQ solver which makes no more queries than A, and B also does not make any pointless queries. If we show this theorem for B, then the result for A immediately follows.

Now, let A be a worst-case SHQ solver which never makes pointless queries. Suppose that A makes at most q queries, and $q < n(p-1)$. Let $H_1, H_2, \ldots, H_q$ be the affine hyperplanes represented by linear equations: let

$$H_i(X_1, \ldots, X_n) = a_{i1}X_1 + \cdots + a_{in}X_n - b_i.$$

Then we may show that $|\cup_{i=1}^{q} H_i| \leq p^n - 2$. First, we cannot have that $|\cup_{i=1}^{q} H_i| = p^n$; in this case, we have $\cup_{i=1}^{q} H_i = \mathbb{Z}_p^n$, so

$$\mathbb{Z}_p^n \setminus \bigcup_{i=1}^{q-1} H_i \subseteq H_q,$$

which shows that the last query H_q is pointless.

Next, suppose that $|\cup_{i=1}^{q} H_i| = p^n - 1$. Let $\mathbb{Z}_p^n \setminus \cup_{i=1}^{q} H_i$, which is a singleton, be $\{\vec{\beta} = (\beta_1, \ldots, \beta_n)\}$.

Then, we define $F \in \mathbb{Z}_p[X_1, \ldots, X_n]$ as

$$F(X_1, \ldots, X_n) := \prod_{i=1}^{q} \left(a_{i1}(X_1 + \beta_1) + \cdots + a_{in}(X_n + \beta_n) - b_i\right).$$

We can easily see that $\deg(F) = q < n(p-1)$, $F(\vec{0}) \neq 0$, and $F(\vec{x}) = 0$ for any $\vec{x} \neq \vec{0}$, which contradicts Theorem 1.8 of Bruen [Bru92], which we quote as Theorem 3 below.

Therefore, whenever $q < n(p-1)$, there should be at least two points $\vec{\beta} \neq \vec{\gamma} \in \mathbb{Z}_p^n$ which are not on $\cup_{i=1}^{q} H_i$. This allows us to use the standard adversary argument against A: for any such SHQ solver A, whenever A asks a hyperplane query H, answer with 0. In the end, if A outputs $\vec{\beta}$, pretend that $\vec{\alpha} = \vec{\gamma}$, and if A outputs any point other than $\vec{\beta}$, pretend that $\vec{\alpha} = \vec{\beta}$. This shows that A in general does not solve the worst-case SHQ problem. Therefore, q should be at least $n(p-1)$ if A is any worst-case SHQ solver. □

Theorem 3 (Theorem 1.8 of [Bru92])**.** *Let F in $\mathbb{Z}_p[X_1, \ldots, X_n]$ satisfy the following conditions.*

1. $F(\vec{0}) \neq 0$
2. $F(\vec{x}) = 0$ *if* $\vec{x} \neq \vec{0}$

Then $\deg(F) \geq n(p-1)$.

For the proof of Theorem 3, we refer to [Bru92]. The proof is done using algebraic techniques.

5.3 Average-case SHQ

Theorem 4. *Let A be any SHQ solver which makes at most q hyperplane queries. Then,*

$$\mathbf{Adv}_{p,n}^{\mathsf{shq}}(A) \leq \frac{1}{p^n} \sum_{i=0}^{n} \binom{q}{i}.$$

Proof. Let A be a SHQ solver which makes at most q hyperplane queries. We are going to argue that we may safely assume that A satisfies certain properties.

First, using essentially the same argument as in Theorem 2, WLOG we may assume that A never makes pointless queries.

Second, we may also assume that A makes exactly q queries: if A is a SHQ solver never making pointless queries, then we define a SHQ solver B as follows: B initializes a counter $ctr \leftarrow 0$, runs A internally, and whenever A makes a query H, then B makes the same query, receives the answer bit b, then returns the bit b to the solver A, and increments the counter: $ctr \leftarrow ctr + 1$. Eventually,

A will halt with an output $\vec{\alpha}'$. Since ctr counts the number of hyperplane queries made by A, we have $ctr \leq q$. Then B makes $q - ctr$ additional hyperplane queries which are not pointless as follows: in case there was at least one hyperplane query H made by A with 0 as the answer, all of the $q - ctr$ remaining queries made by B will be H: surely this query is not pointless, for the answer should be 0. On the other hand, in case there was at least one hyperplane query H made by A with 1 as the answer, let us write H as $H(X_1, \ldots, X_n) = a_1X_1 + \cdots + a_nX_n - b$. Then, let H_0 be the corresponding linear hyperplane defined by $H_0(X_1, \ldots, X_n) = a_1X_1 + \cdots + a_nX_n$. Clearly, $H_0 \neq \mathbb{Z}_p^n$, so there exists a vector $\vec{v} \in \mathbb{Z}_p^n$ satisfying $\vec{v} \notin H_0$. In fact, we may easily find such a $\vec{v}$: since $(a_1, \ldots, a_n) \neq \vec{0}$, WLOG we may assume $a_1 \neq 0$. Then, $\vec{v} := (a_1, 0, 0, \ldots, 0)$ is such an example. Now, let H' be the hyperplane $H + \vec{v}$, which is a parallel translation of H by $\vec{v}$. We may show that $\mathsf{H}(\vec{\alpha}, H') = 0$: suppose not, then $\vec{\alpha} \in H' = H + \vec{v}$, and $\vec{\alpha} \in H$ by assumption. Then, from these two we may conclude that $\vec{v} \in H_0$, which contradicts the construction of $\vec{v}$. Therefore, in this case B makes $q - ctr$ queries, all of them H'. Again these queries are not pointless. Finally, B halts with the answer $\vec{\alpha}'$, which was the output of A.

By the construction, B makes exactly q non-pointless queries, but since the output of B is identical to that of A, we have $\mathbf{Adv}^{\mathsf{shq}}_{p,n}(B) = \mathbf{Adv}^{\mathsf{shq}}_{p,n}(A)$. So, if we prove this theorem for B, the theorem for A clearly follows.

Therefore, now assume that our SHQ solver A makes exactly q non-pointless queries. In general, A may be probabilistic, consuming finite but unbounded number of random bits. Therefore, let us write $A^{\mathsf{H}(\vec{\alpha},\cdot)}(p, n; \vec{r})$ as the output of the algorithm A with input p, n, while having access to the oracle $\mathsf{H}(\vec{\alpha}, \cdot)$ and when the randomness used is $\vec{r} = (r_1, r_2, r_3, \ldots) \in \{0, 1\}^\infty$.

Then we observe that, once $\vec{\alpha}$, $\vec{r}$, and the algorithm A are fixed, the queries made by A and the corresponding answers are also fixed. More precisely, let $H_1, \ldots, H_q$ be the hyperplane queries made by A with some fixed $\vec{\alpha}$, $\vec{r}$, and let $b_1, \ldots, b_q$ be the answer bits: $b_i = \mathsf{H}(\vec{\alpha}, H_i)$. Let us define $\vec{H} := (H_1, \ldots, H_q)$ and $\vec{b} := (b_1, \ldots, b_q)$. Then, in fact, we can see that A, $\vec{r}$, and $\vec{b}$ completely determine $\vec{H}$, and A, $\vec{r}$, and $\vec{\alpha}$ completely determine $\vec{b}$. So we use the following notation: $\vec{H} = \mathcal{H}^{(A)}_{\vec{r}}(\vec{b})$, $\vec{b} = \mathcal{B}^{(A)}_{\vec{r}}(\vec{\alpha})$. Sometimes we just write $\mathcal{H}(\vec{b})$, $\mathcal{B}(\vec{\alpha})$ to simplify notation, when the context is clear.

Moreover, we see that the output $A^{\mathsf{H}(\vec{\alpha},\cdot)}(p, n; \vec{r})$ of the algorithm A is completely determined by A, $\vec{r}$, and the vector $\vec{b}$. So we may write $A^{\mathsf{H}(\vec{\alpha},\cdot)}(p, n; \vec{r})$ as $\mathcal{A}_{\vec{r}}(\vec{b})$. Again, sometimes we just write $\mathcal{A}(\vec{b})$, suppressing $\vec{r}$. For a randomly chosen $\vec{\alpha}$, since the output $\mathcal{A}_{\vec{r}}(\vec{b})$ is determined by $\vec{b} = \mathcal{B}(\vec{\alpha})$, which is in turn determined by $\vec{\alpha}$, we may write $A^{\mathsf{H}(\vec{\alpha},\cdot)}(p, n; \vec{r}) = \mathcal{A}(\mathcal{B}(\vec{\alpha}))$.

Now, let us fix randomness $\vec{r}$, and compute the advantage of A, which is $\mathbf{Pr}[A^{\mathsf{H}(\vec{\boldsymbol{\alpha}},\cdot)}(p, n; \vec{r}) = \vec{\boldsymbol{\alpha}}]$, where the probability is only over the random choice of $\vec{\boldsymbol{\alpha}}$. Here, to emphasize that it is a random variable, we used bold font to write $\vec{\boldsymbol{\alpha}}$. We then have

$$\begin{aligned}
\mathbf{Pr}[A^{\mathsf{H}(\vec{\boldsymbol{\alpha}},\cdot)}(p, n; \vec{r}) = \vec{\boldsymbol{\alpha}}] &= \mathbf{Pr}[\mathcal{A}(\mathcal{B}(\vec{\boldsymbol{\alpha}})) = \vec{\boldsymbol{\alpha}}] \\
&= \sum_{\vec{\alpha}} \mathbf{Pr}[\vec{\boldsymbol{\alpha}} = \vec{\alpha}] \cdot \mathbf{Pr}[\mathcal{A}(\mathcal{B}(\vec{\boldsymbol{\alpha}})) = \vec{\boldsymbol{\alpha}} \mid \vec{\boldsymbol{\alpha}} = \vec{\alpha}] \\
&= \frac{1}{p^n} \sum_{\vec{\alpha}} \mathbf{Pr}[\mathcal{A}(\mathcal{B}(\vec{\alpha})) = \vec{\alpha}],
\end{aligned}$$

where $\vec{\boldsymbol{\alpha}}$ is a random variable with uniform distribution on $\mathbb{Z}_p^n$, and $\vec{\alpha}$ is used as a variable for possible concrete values of $\vec{\boldsymbol{\alpha}}$. Note that $\mathbf{Pr}[\mathcal{A}(\mathcal{B}(\vec{\alpha})) = \vec{\alpha}]$ should be either 0 or 1, for any $\vec{\alpha}$, because all randomness is fixed: we have $\mathbf{Pr}[\mathcal{A}(\mathcal{B}(\vec{\alpha})) = \vec{\alpha}] = 1$ iff $\mathcal{A}(\mathcal{B}(\vec{\alpha})) = \vec{\alpha}$. Continuing,

$$\begin{aligned}
\mathbf{Pr}[A^{\mathsf{H}(\vec{\boldsymbol{\alpha}},\cdot)}(p, n; \vec{r}) = \vec{\boldsymbol{\alpha}}] &= \frac{1}{p^n} \sum_{\vec{\alpha}} \mathbf{Pr}[\mathcal{A}(\mathcal{B}(\vec{\alpha})) = \vec{\alpha}], \\
&= \frac{1}{p^n} \sum_{\vec{b}} \sum_{\vec{\alpha}:\mathcal{B}(\vec{\alpha})=\vec{b}} \mathbf{Pr}[\mathcal{A}(\mathcal{B}(\vec{\alpha})) = \vec{\alpha}], \\
&= \frac{1}{p^n} \sum_{\vec{b}} \sum_{\vec{\alpha}:\mathcal{B}(\vec{\alpha})=\vec{b}} \mathbf{Pr}[\mathcal{A}(\vec{b}) = \vec{\alpha}],
\end{aligned}$$

We can see that, in the above, for any $\vec{b}$,

$$\sum_{\vec{\alpha}:\mathcal{B}(\vec{\alpha})=\vec{b}} \mathbf{Pr}[\mathcal{A}(\vec{b}) = \vec{\alpha}] \leq 1,$$

where the sum is over all $\vec{\alpha}$ satisfying $\mathcal{B}(\vec{\alpha}) = \vec{b}$. Indeed, the only $\vec{\alpha}$ which can possibly make $\mathbf{Pr}[\mathcal{A}(\vec{b}) = \vec{\alpha}] = 1$ is $\vec{\alpha} = \mathcal{A}(\vec{b})$, so if $\mathcal{B}(\mathcal{A}(\vec{b})) = \vec{b}$, then the above value is 1, and if $\mathcal{B}(\mathcal{A}(\vec{b})) \neq \vec{b}$ then the above value is 0.

Therefore, we see that

$$\begin{aligned}\mathbf{Pr}[A^{\mathsf{H}(\vec{\boldsymbol{\alpha}},\cdot)}(p,n;\vec{r}) = \vec{\boldsymbol{\alpha}}] &\leq \frac{1}{p^n}\sum_{\vec{b}} 1 \\ &= \frac{\text{the number of all possible } \vec{b}\text{'s}}{p^n}.\end{aligned}$$

Any $\vec{b} = \mathcal{B}(\alpha)$ is a bitstring of length q, and moreover, in any such $\vec{b}$, 1 cannot occur more than n times: this is because we assumed that the algorithm A never makes pointless queries: suppose $H_1, \ldots, H_m$ are hyperplane queries made by A with 1 as the answer. Then, all of these hyperplanes intersect ($\vec{\alpha}$ is on all of them). Moreover, due to the fact that A does not make pointless queries, we have

$$H_1 \cap \cdots \cap H_i \not\subseteq H_{i+1},$$

for all $i = 1, 2, \ldots, m-1$. But then each hyperplane should decrement the dimension of the intersection by 1, so there can be at most n such hyperplanes, and there can be at most n 1s in any $\vec{b}$. Hence we have,

$$\mathbf{Pr}[A^{\mathsf{H}(\vec{\boldsymbol{\alpha}},\cdot)}(p,n;\vec{r}) = \vec{\boldsymbol{\alpha}}] \leq \frac{1}{p^n}\sum_{i=0}^{n}\binom{q}{i}.$$

Finally, the theorem is satisfied for general A, because when conditioned on any randomness $\vec{r}$, the success probability is bounded by the same upper bound $p^{-n}\sum_{i=0}^{n}\binom{q}{i}$. □

Corollary 1. *Let A be any SHQ solver which makes at most q hyperplane queries. Then,*

$$\mathbf{Adv}^{\mathsf{shq}}_{p,n}(A) \leq \frac{1}{p^n} + \frac{1}{2}\left(\frac{eq}{np}\right)^n.$$

Proof. The proof follows from Theorem 4 and the following Theorem 5. □

Remark 2. If we write q as $q = np\delta$ for some δ, then Corollary 1 says that the advantage of the solver A is bounded by $p^{-n} + (e\delta)^n/2$. Since we assume that $n = o(p)$, certainly $p^{-n} \to 0$ as $\lambda \to \infty$. So, since we assume n is unbounded, if $\delta < 1/e$, then $\mathbf{Adv}^{\mathsf{shq}}_{p,n}(A) \to 0$ as $\lambda \to \infty$. This shows that if A solves SHQ with constant advantage, then $\delta \geq 1/e$. In short, a SHQ solver with constant advantage should make $\Omega(np)$ queries.

Theorem 5. *We have*

$$\sum_{i=1}^{n}\binom{q}{i} \leq \frac{1}{2}\left(\frac{eq}{n}\right)^n$$

for any positive integers q, n satisfying $1 \leq n \leq q$.

The proof of Lemma 5 is in the Appendix A.

6 Conclusion

6.1 Generic hardness of MDL

By combining the results so far, we obtain the following corollary:

Corollary 2. *Let A be any generic MDL solver which makes at most q queries. Then,*

$$\mathbf{Adv}^{mdl}_{p,n}(A) \leq \frac{1}{p^n} + \frac{1}{2}\left(\frac{e(q+n+1)^2}{2np}\right)^n.$$

Proof. This follows directly from Theorem 1 and Corollary 1. □

Let us write $q = \sqrt{np}\delta$ for some δ. Then, the upper bound of $\mathbf{Adv}^{\mathsf{mdl}}_{p,n}(A)$ in Corollary 2 can be expanded as

$$\frac{1}{p^n} + \frac{1}{2}\left(\frac{e(\sqrt{np}\delta+n+1)^2}{2np}\right)^n = \frac{1}{p^n} + \frac{1}{2}\left(\frac{e\delta^2}{2} + e\delta\sqrt{\frac{n}{p}} + \frac{e\delta}{\sqrt{np}} + \frac{en}{2p} + \frac{e}{p} + \frac{e}{2np}\right)^n.$$

Suppose that A solves MDL with constant advantage. Then we can see that $\delta \geq \sqrt{2/e}$: suppose not and assume that $\delta < \sqrt{2/e}$. Since we assume that $n = o(p)$, all the terms except $e\delta^2/2$ converges to 0 as $\lambda \to \infty$. So, we have

$$\mathbf{Adv}^{\mathsf{mdl}}_{p,n}(A) \leq \frac{1}{p^n} + \frac{1}{2}\left(\frac{e\delta^2}{2} + o(1)\right)^n.$$

But, since we assume that n is unbounded, from $\delta < \sqrt{2/e}$ we obtain $\mathbf{Adv}^{\mathsf{mdl}}_{p,n}(A) \to 0$ as $\lambda \to \infty$, contradicting that A has constant advantage. Therefore, if a generic MDL solver has constant advantage, then it should make $\Omega(\sqrt{np})$ queries. This affirmatively settles Kuhn and Struik's conjecture [KS01].

6.2 Interval-MDL

We may also consider Interval-MDL, where instead of the exponents $\alpha_1, \ldots, \alpha_n$ are chosen from the whole group $\mathbb{Z}_p$, they are chosen from an interval $\{0, 1, \ldots, l-1\} \subseteq \mathbb{Z}_p$ of size l. For example, Boneh-Goh-Nissim homomorphic encryption [BGN05] requires solving DL for exponents chosen from such an interval, and Bernstein and Lange [BL12] suggested preprocessing methods to speed up such computations.

We remark that with trivial modifications, all of our results (except the worst-case SHQ) also apply to Interval-MDL and the corresponding Interval-SHQ: in the upper bounds for advantages, simply replace the group order p with the interval size l. For example, the bound in Corollary 2 becomes

$$\frac{1}{l^n} + \frac{1}{2}\left(\frac{e(q+n+1)^2}{2nl}\right)^n,$$

and a generic Interval-MDL solver with constant advantage should make $\Omega(\sqrt{nl})$ queries, assuming n is unbounded and $n = o(l)$. This is because our proof techniques, especially that of Theorem 4, work equally well for the interval version. For that matter, the size-l subset does not even have to be an interval: any subset of size l would do.

Acknowledgments. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MEST) (No. 2011-0025127).

References

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography*, pages 325–341. Springer, 2005.

[BL12] Daniel J. Bernstein and Tanja Lange. Computing small discrete logarithms faster. In *Progress in Cryptology — INDOCRYPT 2012*, pages 317–338. Springer, 2012.

[Bru92] Aiden A. Bruen. Polynomial multiplicities over finite fields and intersection sets. *Journal of Combinatorial Theory, Series A*, 60(1):19–33, 1992.

[HMCD04] Yvonne Hitchcock, Paul Montague, Gary Carter, and Ed Dawson. The efficiency of solving multiple discrete logarithm problems and the implications for the security of fixed elliptic curves. *International Journal of Information Security*, 3(2):86–98, 2004.

[KS01] Fabian Kuhn and René Struik. Random walks revisited: Extensions of Pollard's Rho algorithm for computing multiple discrete logarithms. In *Selected Areas in Cryptography*, pages 212–229. Springer, 2001.

[MY96] Ueli M. Maurer and Yacov Yacobi. A non-interactive public-key distribution system. *Designs, Codes and Cryptography*, 9(3):305–316, 1996.

[Nec94] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 02 1994. Translated from Matematicheskie Zametki, Vol. 55, No. 2, pp. 91–101, February, 1994.

[NIS13] Digital signature standard (DSS). NIST (National Institute of Standards and Technology) FIPS 186-4, 2013.

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

A Proof of Theorem 5

Before proving Theorem 5, we need a technical lemma:

Lemma 1. *Suppose that* $q \geq 5$ *and* $2 \leq n \leq q-3$. *Then,*

$$q \sum_{i=1}^{n} \binom{q}{i} \geq (n+1) \sum_{i=1}^{n+1} \binom{q}{i}. \tag{1}$$

Proof. Letting $S := \sum_{i=1}^{n} \binom{q}{i}$, we may write the inequality (1) as

$$qS \geq (n+1)\left(S + \binom{q}{n+1}\right). \tag{2}$$

This can be simpified as

$$\frac{n+1}{q-n-1}\binom{q}{n+1} \leq S. \tag{3}$$

But,

$$\begin{aligned} \frac{n+1}{q-n-1}\binom{q}{n+1} &= \frac{n+1}{q-n-1} \cdot \frac{q!}{(n+1)!(q-n-1)!} \\ &= \frac{q-n}{q-n-1} \cdot \frac{q!}{n!(q-n)!} \\ &= \frac{q-n}{q-n-1}\binom{q}{n}. \end{aligned} \tag{4}$$

So, the inequality (1) is equivalent to

$$\left(1 + \frac{1}{q-n-1}\right)\binom{q}{n} \leq \sum_{i=1}^{n} \binom{q}{i}, \tag{5}$$

which in turn is equivalent to

$$\frac{1}{q-n-1}\binom{q}{n} \leq \sum_{i=1}^{n-1}\binom{q}{i}. \tag{6}$$

So let us prove this inequality (6).

Consider the function $f(n) := (n-1)(q-1-n)$. As a function of n, this is a quadratic concave function with $f(1) = f(q-1) = 0$. Since we assume $2 \leq n \leq q-3$, we have $f(n) \geq \min(f(2), f(q-3))$. Since $f(2) = q-3 \geq 2$ and $f(q-3) = 2(q-4) \geq 2$, we have

$$(n-1)(q-1-n) \geq 2, \tag{7}$$

for any $n = 2, \ldots, q-3$. Simple calculation shows that this is equivalent to

$$\frac{1}{(q-n-1)n} \leq \frac{1}{q-n+1}. \tag{8}$$

Then,

$$\begin{aligned}
\frac{1}{q-n-1}\binom{q}{n} &= \frac{1}{(q-n-1)n}\cdot n\binom{q}{n} \\
&\leq \frac{1}{q-n+1}\cdot n\binom{q}{n} \\
&\leq \frac{n}{q-n+1}\cdot\frac{q!}{n!(q-n)!} = \frac{q!}{(n-1)!(q-n+1)!} \\
&= \binom{q}{n-1} \leq \sum_{i=1}^{n-1}\binom{q}{n}.
\end{aligned} \tag{9}$$

□

Now we are ready to prove Theorem 5:

Theorem 5. *We have*

$$\sum_{i=1}^{n}\binom{q}{i} \leq \frac{1}{2}\left(\frac{eq}{n}\right)^n \tag{10}$$

for any positive integers q, n satisfying $1 \leq n \leq q$.

Proof. The proof is based on case analysis. First, we prove the inequality when $q \geq 5$ and $1 \leq n \leq q-2$.

From Lemma 1, we have

$$q\sum_{i=1}^{n}\binom{q}{i} \geq (n+1)\sum_{i=1}^{n+1}\binom{q}{i}, \tag{11}$$

for $q \geq 5$ and $2 \leq n \leq q-3$.

Then, since $e \geq (1+1/n)^n$, we have

$$eq\sum_{i=1}^{n}\binom{q}{i} \geq \left(1+\frac{1}{n}\right)^n(n+1)\sum_{i=1}^{n+1}\binom{q}{i}, \tag{12}$$

which is equivalent to

$$\left(\frac{n}{eq}\right)^n\sum_{i=1}^{n}\binom{q}{i} \geq \left(\frac{n+1}{eq}\right)^{n+1}\sum_{i=1}^{n+1}\binom{q}{i}. \tag{13}$$

Also, when $n = 1$, the above inequality (13) is

$$\frac{1}{eq}\binom{q}{1} \geq \left(\frac{2}{eq}\right)^2\sum_{i=1}^{2}\binom{q}{i}, \tag{14}$$

which is equivalent to

$$\frac{e}{2} \geq 1 + \frac{1}{q}, \tag{15}$$

which is certainly satisfied when $q \geq 5$. So,

$$\left(\frac{n}{eq}\right)^n \sum_{i=1}^{n} \binom{q}{i} \tag{16}$$

is a decreasing function for $n \in \{1, 2, \ldots, q-2\}$. Then, for any $n = 1, 2, \ldots, q-2$, we have

$$\left(\frac{n}{eq}\right)^n \sum_{i=1}^{n} \binom{q}{i} \leq \left(\frac{1}{eq}\right)^1 \sum_{i=1}^{1} \binom{q}{i} = \frac{1}{e} \leq \frac{1}{2}, \tag{17}$$

proving the inequality (10) when $q \geq 5$ and $1 \leq n \leq q-2$.

Therefore, we need to handle the remaining cases: when $q \leq 4$, or when $n = q-1$, q.

- Case $n = q$: Then the inequality (10) is equivalent to

$$2^q - 1 \leq \frac{1}{2}\left(\frac{eq}{q}\right)^q = \frac{e^q}{2}. \tag{18}$$

  This holds when $2^q \leq e^q/2$, which can be written as $q/(q+1) \geq \log 2 \approx 0.693\cdots$. So this inequality holds when $q \geq 3$; then $q/(q+1) \geq 0.75 > \log 2$. We can also check that $2^q - 1 \leq e^q/2$ holds for $q = 1, 2$ separately.
- Case $n = q-1$: Then the inequality (10) is equivalent to

$$2^q - 2 \leq \left(\frac{q}{q-1}\right)^{q-1} \frac{e^{q-1}}{2}. \tag{19}$$

  Since the RHS is greater than $e^{q-1}/2$, the inequality is satisfied if $2^q - 2 \leq e^{q-1}/2$. First, we can check that $2^q \leq e^{q-1}/2$ holds if $q \geq 6$. And we can also separately check the inequality (19) for $q = 2, \ldots, 5$. This finishes this case.
- Case $q \leq 4$: Here, we need only to show that the inequality (10) holds when $n = 1$ or 2 (of course when $n \leq q$). This is because, when $q = 1, 2$, then $n = 1, 2$ cases cover all possibilities. Also, when $q = 3, 4$, then $n = 1, 2$, and $n = q-1, q$ cases cover all possibilities. Hence,
- Case $n = 1$: Then the inequality (10) is equivalent to

$$q \leq \frac{1}{2}\left(\frac{eq}{1}\right), \tag{20}$$

  which holds trivially, since $e \geq 2$.
- Case $n = 2$: Then the inequality (10) is equivalent to

$$q + \frac{q(q-1)}{2} = \frac{q(q+1)}{2} \leq \frac{1}{2}\left(\frac{eq}{2}\right)^2. \tag{21}$$

  Simplifying, we get

$$\frac{1}{q} \geq \frac{e^2}{4} - 1 \approx 0.847\cdots, \tag{22}$$

  which holds for $q \geq 2$. □