Improved Bounds on Security Reductions for Discrete Log Based Signatures

Sanjam Garg[1,⋆], Raghav Bhaskar[2], and Satyanarayana V. Lokam[2]

[1] IIT, Delhi, India
sanjamg@yahoo.com
[2] Microsoft Research India, Bangalore, India
{rbhaskar,satya}@microsoft.com

Abstract. Despite considerable research efforts, no efficient reduction from the discrete log problem to forging a discrete log based signature (e.g. Schnorr) is currently known. In fact, negative results are known. Paillier and Vergnaud [PV05] show that the forgeability of several discrete log based signatures *cannot* be equivalent to solving the discrete log problem in the standard model, *assuming* the so-called one-more discrete log assumption and algebraic reductions. They also show, under the same assumptions, that, any security reduction in the Random Oracle Model (ROM) from discrete log to forging a Schnorr signature must lose a factor of at least $\sqrt{q_h}$ in the success probability. Here q_h is the number of queries the forger makes to the random oracle. The best known positive result, due to Pointcheval and Stern [PS00], also in the ROM, gives a reduction that loses a factor of q_h. In this paper, we improve the negative result from [PV05]. In particular, we show that any algebraic reduction in the ROM from discrete log to forging a Schnorr signature must lose a factor of at least $q_h^{2/3}$, assuming the one-more discrete log assumption. We also hint at certain circumstances (by way of restrictions on the forger) under which this lower bound may be tight. These negative results indicate that huge loss factors may be inevitable in reductions from discrete log to discrete log based signatures.

Keywords: Provable Security, Random Oracle Model, Schnorr Signature Scheme.

1 Introduction

Discrete Log (DL) based signatures, such as those proposed by Schnorr [Sch90], are among the simplest and the most efficient signature schemes. The small size of the produced signatures and the scope for pre-computation to efficiently generate signatures on-line make them particularly attractive for many applications. Though they have steadily withstood cryptanalytic attacks over the years, only recently has something been known about their *provable* security. Unfortunately, as we discuss below, this knowledge seems to derive largely from *negative* results.

[⋆] Part of the work done while visiting Microsoft Research India.

D. Wagner (Ed.): CRYPTO 2008, LNCS 5157, pp. 93–107, 2008.

The best known positive result, due to Pointcheval and Stern [PS96, PS00], is a security reduction from the Discrete Log (DL) problem to forging a Schnorr signature *in the Random Oracle Model* (ROM) [BR93]. Their reduction rewinds a forger algorithm and uses a certain *Forking Lemma* to obtain two distinct forgeries on the same message which permits it to solve the discrete log problem. However, the reduction incurs a loss factor in efficiency in the sense that the obtained DL solver will lose a factor of q_h either in time complexity or success probability as compared to the forger. Here q_h is the number of queries the forger makes to the random oracle. Despite several efforts, no better reduction is known in the ROM. Nor is any reduction known at all in the Standard model. This situation remained until a major result was obtained in 2005 by Paillier and Vergnaud. In [PV05], they showed that no efficient reduction can exist in the standard model from DL to forging a Schnorr signature, *assuming*[1] (i) the so-called n-DL problem is hard (also called the one-more discrete log assumption) and (ii) the reduction is algebraic. (We explain both these notions a bit more in Section 2.) This indicates that the discrete log problem and forgeability of Schnorr signatures are unlikely to be equivalent in the standard model. A similar situation is known to exist in the case of RSA, by a result due to Boneh and Venkatesan [BV98] (where they also consider reductions similar to algebraic reductions). In the ROM, [PV05] also proved that any algebraic reduction must lose a factor of at least $\sqrt{q_h}$ in its success probability if it were to convert an efficient forger of the Schnorr signature scheme into an efficient DL-solver, again assuming that n-DL is hard. Thus, there remained a gap between the lower bound of $\sqrt{q_h}$ and the upper bound of q_h on the loss factor of algebraic reductions in the ROM from DL to forging Schnorr signatures. This paper is an attempt to close this gap.

Our Contributions: We improve the lower bound from $\sqrt{q_h}$ to $q_h^{2/3}$. More precisely, we show that any efficient algebraic reduction from DL to forging (a universal forger under key-only attack) a Schnorr signature scheme must lose a factor of $q_h^{2/3}$ in its success probability, assuming that the n-DL problem is hard. Our proof, as in [PV05], constructs a meta-reduction that uses the supposed algebraic reduction converting a forger into a DL-solver to solve the n-DL problem. In this process, the meta-reduction needs to simulate the forger (or adversary) that is used by the reduction. Our improvement hinges on a more careful construction of this simulation and a more refined analysis of the success probability of the meta-reduction in this simulation. In this analysis, we make use of known estimates [Pil90] on the expected length of a longest increasing subsequence of a random sequence. The adversary (simulated by the meta-reduction) in our lower bound proof has a certain structure. We observe that a reduction in the ROM that exploits an adversary *adhering to such structure* can indeed solve DL with a loss factor of *at most* $q_h^{2/3}$, i.e., under these restrictions on the forger, our lower bound is tight. These insights and our concrete lower bound indicate that

[1] Obviously, given our state of knowledge on lower bounds, some assumption is needed for such impossibility results.

huge loss factors may be inevitable, even in the ROM, in security reductions that convert a Schnorr-forger into a DL-solver. In other words, while forging Schnorr signatures and extracting discrete logs are known to be *equivalent in the sense of polynomial time reductions* in the ROM, the loss factors incurred in such reductions might be impractically large (under certain assumptions, as always). We note that proving negative results on security reductions (such as lower bounds on loss factors) in the Random Oracle Model is a harder task than in the standard model.

While we state and prove our results for the Schnorr signature scheme, they are valid for many schemes based on the discrete log problem. This follows from the same arguments as in [PV05] since we use the same meta-reduction.

The rest of the paper is organized as follows: In Section 2, we review some definitions and present technical preliminaries. In Section 3, we provide the main result which shows that any reduction from the Discrete Log problem to forgeability of Schnorr signatures must lose a factor of $q_h^{2/3}$ in its success probability and comment on the tightness of the result. In Section 4 we state conclusions and mention some open problems.

2 Preliminaries

Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q generated by g.

Definition 1 (DL Problem). *Given $r \in \mathbb{G}$, computing $k \in \mathbb{Z}_q$ such that $r = g^k$ is known as the Discrete Log (DL) problem over the group $\mathbb{G}$.*

A probabilistic algorithm $\mathcal{A}$ is said to be an (ε, τ)-solver for DL if

$$\Pr_{k \overset{\$}{\leftarrow} \mathbb{Z}_q} \left[\mathcal{A}(g^k) = k\right] \geq \varepsilon,$$

where the probability is taken over the random tape of $\mathcal{A}$ and random choices of k and $\mathcal{A}$ stops after time at most τ.

The (ε, τ)-discrete log assumption (for group $\mathbb{G}$) says that no (ε, τ)-solver can exist for DL over $\mathbb{G}$. The (asymptotic) *DL-assumption* says that the (ε, τ)-discrete log assumption holds whenever $\tau = \text{poly}(\log q)$ and ε is a non-negligible function of $\log q$.

Definition 2 (n-DL Problem). *Given an $(n+1)$-tuple $(r_0, r_1, \ldots, r_n)$ of distinct elements in $\mathbb{G}$ and up to n queries to a discrete log oracle, computing the $(n+1)$-tuple of elements $(k_0, k_1, \ldots, k_n)$ $(k_i \in \mathbb{Z}_q)$ such that $r_i = g^{k_i}$ for $0 \leq i \leq n$ is known as the n-DL problem.*

A probabilistic algorithm $\mathcal{A}$ is said to be an (ε, τ)-solver for n-DL if

$$\Pr_{k_0, k_1, \ldots, k_n \overset{\$}{\leftarrow} \mathbb{Z}_q} \left[\mathcal{A}^{\text{DL-oracle}}(g^{k_0}, g^{k_1}, \ldots, g^{k_n}) = k_0, k_1, \ldots, k_n\right] \geq \varepsilon,$$

where the probability is taken over the random tape of $\mathcal{A}$ and random choices of the k_i and $\mathcal{A}$ stops after time at most τ.

The (ε, τ)-n-DL assumption says that no (ε, τ)-solver can exist for n-DL over $\mathbb{G}$ in time τ and with probability greater than ε. The (asymptotic) *n-DL assumption* says that the (ε, τ)-n-DL assumption holds whenever $\tau = \text{poly}(\log q)$ and ε is a non-negligible function of $\log q$.

Note that the DL problem is at least as hard as the n-DL problem. Hence the n-DL assumption is a stronger assumption than the DL assumption. It is not known if it is strictly stronger.

Definition 3 (Schnorr Signature Scheme). *Let p and q be primes such that $q \mid (p-1)$ Let g be a generator of the cyclic subgroup $\mathbb{G}$ of order q in $\mathbb{Z}_p^*$. Let H be a secure hash function with range $\{1, \ldots, q-1\}$. The Schnorr signature scheme consists of the following three algorithms:*

1. Key Generation: *Choose a random x with $0 < x < q$. x is the private key and $y := g^x$ is the public key.*
2. Signing: *Given the input message m, choose a random $k \mod q$. Let $c := H(m, r)$, and $s := k + cx$. Return (c, s) as the signature.*
3. Verification: *Given the message m and the signature pair (c, s), calculate $r = g^s y^{-c}$. Let $c' = H(m, r)$. If $c = c'$ then return $true$ else return $false$.*

Attack and Forgery types: An adversary can broadly mount two kinds of attacks against signature schemes: *Key-only attack* (KOA, also called no-message attack) and *Chosen message attack* (CMA). In the first attack, the attacker knows only the public key of the signer while in the latter, the attacker can also obtain signatures on messages of his choice adaptively. The result of the attacks by the adversary are classified as follows:

1. Total Break - The adversary learns the secret key of the signer.
2. Universal Forgery (UF) - The adversary can produce a valid signature for any message.
3. Existential Forgery(EF) - The adversary can produce a new message signature pair.

Thus, by combining the attack type and the attack result, one can talk about various levels of security for digital signatures. For instance, a (ε, τ)-universal forger under key-only attack is an adversary who, knowing only the public key, can produce a signature on any given message with probability ε in time at most τ. An (ε, τ, q_h)-universal forger under key-only attack is the same adversary *in the Random Oracle Model* who, makes at most q_h hash queries to the random oracle. For details refer to [MvOV96, PV05].

Algebraic Reductions: We assume our reductions to be *algebraic algorithms*. An algorithm $\mathcal{R}$ is said to be algebraic *with respect to a group* $\mathbb{G}$ if the only operations $\mathcal{R}$ performs on group elements are group operations; on objects that are not group elements, $\mathcal{R}$ is allowed to perform any (standard basic) operations. In particular, given $g_1, g_2 \in \mathbb{G}$, $\mathcal{R}$ can only (i) check if g_1 and g_2 are the same, (ii) compute $g_1 \cdot g_2$ (we represent the group multiplicatively), and (iii) raise g_1 to a power (including to -1, thus computing inverse). Other natural variations on this definition are possible. Algebraic algorithms encompass many natural algorithms/reductions used in cryptography and impose weaker conditions than other known models such as the generic group model (GGM). They were originally and more generally defined by [BV98] in the context of the *ring* $\mathbb{Z}_{pq}$ in their study of RSA versus factoring.

A formal property characterizing an algebraic algorithm $\mathcal{R}$ may be described as follows. Suppose $\mathcal{R}$ takes group elements $g_1, \dots g_k$ (and possibly other objects) and produces a group element h after τ steps. Then, there is an associated function `Extract` that takes all of inputs to $\mathcal{R}$, the code/program of $\mathcal{R}$ (so `Extract` could have non-black-box access to $\mathcal{R}$), and produces integers $\alpha_1, \dots, \alpha_k$ such that $h = g_1^{\alpha_1} \cdots g_k^{\alpha_k}$ in time polynomial in τ and $|\mathcal{R}|$, where $|\mathcal{R}|$ denotes the length of the program of $\mathcal{R}$.

In particular, suppose $\mathcal{R}$ is algebraic with respect to group $\mathbb{G} = \langle g \rangle$. Suppose $\mathcal{R}$ produces elements $y_1, \dots y_n$ during its computation. Then, given $\mathcal{R}$'s code and all its inputs, `Extract` would be able to produce $x_1, \dots, x_n$ such that $y_i = g^{x_i}$ in time polynomial in $\mathcal{R}$'s running time to produce y_i and its code length $|\mathcal{R}|$.

3 Improved Lower Bound

Theorem 1. *Suppose there exists an algebraic reduction $\mathcal{R}$ that converts an (ε, τ, q_h)-universal forger $\mathcal{A}$ under a key-only attack in the random oracle model on the Schnorr signature scheme into an (ε', τ')-solver for the discrete logarithm problem. Further, assume that $\mathcal{R}$ invokes $\mathcal{A}$ at most n times.*

Then there exists a probabilistic algorithm $\mathcal{M}$ that (ε'', τ'')-solves n-DL, where

$$\varepsilon'' \geq \varepsilon' \left(1 - \frac{2n^{3/2}}{q_h} - \frac{1}{q-1}\right) \text{ and } \tag{1}$$

$$\tau'' \leq \mathrm{poly}(\tau', |\mathcal{R}|, n, q_h, \log q). \tag{2}$$

Recall from Section 2 that $|\mathcal{R}|$ denotes the length of the code of $\mathcal{R}$.

Corollary 1. *Under the n-DL assumption, any efficient algebraic reduction that converts a UF-KO attack on the Schnorr signature scheme to an algorithm for the discrete log problem must incur a loss factor of $q_h^{2/3}$ in its success probability or its running time.*

Proof. If there is such a reduction from a feasible attack, the time complexity τ'' of the meta-reduction $\mathcal{M}$ is polynomial (in $\log q$). Also, by the n-DL assumption, ε'' must be negligibly small. Suppose now that $\varepsilon' \leq \varepsilon / q_h^{2/3}$ (any non-negligible ε' will do). By the assumed feasibility of the attack, ε is non-negligible and hence so is ε'. Thus, we must have $\varepsilon''/\varepsilon'$ negligibly close to zero. From (1), we obtain $\frac{2n^{3/2}}{q_h} + \frac{1}{q-1}$ must be negligibly close to 1. Since q (the size of the group) is exponentially large, this means that $\frac{2n^{3/2}}{q_h}$ is negligibly close to 1. Hence $n = \Omega(q_h^{2/3})$. Since the reduction makes n calls to the attacker, we must then have $\tau' \geq n\tau = \Omega(q_h^{2/3}\tau)$. □

Structure of the proof of Theorem 1: Our proof proceeds, as in [PV05], by constructing a meta reduction $\mathcal{M}$ that solves n-DL using the supposed algebraic reduction $\mathcal{R}$. Note that $\mathcal{R}$ in turn makes n calls to *any* given adversary (universal forger) $\mathcal{A}$ to solve the discrete log problem. Thus our meta reduction $\mathcal{M}$ will simulate a particular kind of adversary $\mathcal{A}$ that responds to $\mathcal{R}$ by giving signatures on $\mathcal{R}$'s chosen (public key, message) pairs. To generate these forgeries by the simulated adversary, $\mathcal{M}$ takes advantage of the fact that it gets to make n calls to a discrete log oracle. Using the algebraic nature of $\mathcal{R}$ and its ability to solve the discrete log problem (given n calls to the universal forger), $\mathcal{M}$ will be able to extract the discrete logs of all the $n + 1$ elements given as its own input. Note that, since the reduction $\mathcal{R}$ is in the Random Oracle Model, the hash function H to which the adversary $\mathcal{A}$ may make up to q_h calls is under the control (in the sense of simulation) of $\mathcal{R}$.

In the next two subsections, we will describe the meta-reduction $\mathcal{M}$ and analyze its performance. The bound on $\mathcal{M}$'s running time τ'' will follow easily from the construction of $\mathcal{M}$. The bound on its success probability ε'', however, needs an intricate analysis and differs significantly from that in [PV05].

3.1 The Meta-reduction $\mathcal{M}$

The meta-reduction $\mathcal{M}$ gets an $n + 1$-tuple $(r_0, \ldots, r_n)$ of group elements. It feeds r_0 to the reduction $\mathcal{R}$ on an arbitrary random tape. It uses the remaining n elements $r_1, \ldots, r_n$ to simulate $\mathcal{A}$. At the end of the simulation it gets the discrete log $k_0 = \log_g r_0$ from $\mathcal{R}$. It extracts $k_i = \log_g r_i$ for $1 \leq i \leq n$ from the transcript of $\mathcal{R}$ as described in Section 2.

Simulation of $\mathcal{A}$: The reduction $\mathcal{R}$ is allowed to invoke the universal forger $\mathcal{A}$ up to n times with freely chosen public keys $y_i = g^{x_i}$, messages m_i, and random tapes ϖ_i where $i = 1, \ldots, n$. We note that the input of ϖ_i is necessary to carry out an oracle replay attack. In fact, we can prove that an algebraic reduction cannot solve the discrete log problem using independent runs of the adversary (adversaries which derive their randomness from external sources) using techniques very similar to those in [PV05] for the standard model. Without loss of generality, we may assume that these n invocations of $\mathcal{A}$ are pairwise distinct, i.e., that two distinct executions of $\mathcal{A}$ differ in the value of the public key and/or the random tape, and/or at least one value returned by the random oracle H of $\mathcal{R}$. The adversary $\mathcal{A}$ is allowed to make up to q_h hash queries to the random oracle H which is under the control (in the sense of simulation) of the reduction $\mathcal{R}$. In its simulation of $\mathcal{A}$, the meta-reduction $\mathcal{M}$ makes $\mathcal{A}$ return a forgery on *a randomly chosen hash query from these q_h queries.*

We denote the i-th execution of the adversary by $\mathcal{A}_i$. The state of $\mathcal{A}_i$ at any given time is determined by its input $(y_i,\ m_i,\ \varpi_i)$ and the return values to its hash queries till that time. Let us define $c_k(i)$ to be the response of the k-th hash query made by the the i-th execution of the adversary. Let us also define

$$\text{History}_h(i) := \langle (y_i, m_i, \varpi_i), c_1(i), c_2(i), \ldots, c_{h-1}(i) \rangle,$$

to be the history of computation of $\mathcal{A}_i$ up to the h-th query. If the j-th execution of the adversary and its i-th execution have the same history till before the h-th query, then the h-th hash query of both executions must be the same (however, the responses to these queries could differ in these two executions). This puts constraints on the way meta-reduction $\mathcal{M}$ should model the adversary $\mathcal{A}$. *In particular, all random selections made by $\mathcal{A}_i$ are in fact pseudo-random in History(i) when the selection takes place.*

During its simulation $\mathcal{M}$ can make up to n calls to the discrete log oracle DL_{OM}. Naturally, its queries to DL_{OM} will depend on the vector of group elements $\boldsymbol{r} = (r_1, \ldots, r_n)$. To economize on the number of its queries, $\mathcal{M}$ will call DL_{OM} through a discrete-log "stub-routine" DL_{stub}. The stub maintains a list of already asked queries and makes sure that a query asked multiple times is not actually queried to the oracle but answered by the stub itself.

The meta-reduction $\mathcal{M}$ will simulate the adversary with perfect forgeries (i.e. that succeed with probability 1). Since the reduction $\mathcal{R}$ solves the DL problem with probability at least ε' while interacting with an adversary that produces a forgery with probability at least ε, $\mathcal{R}$ will succeed with at least the same probability with a perfect forger.

Notation: For vectors $\boldsymbol{g} = (g_1, \ldots, g_w) \in \mathbb{G}^w$ and $\boldsymbol{b} = (b_1, \ldots, b_w) \in \mathbb{Z}$, we define $\boldsymbol{g}^{\boldsymbol{b}}$ as $\boldsymbol{g}^{\boldsymbol{b}} := \prod_{k=1}^{w} g_k^{b_k}$.

We now describe the simulation by $\mathcal{M}$ of $\mathcal{A}_i$ for $1 \leq i \leq n$:

```
1. Receive $(y_i, m_i, \varpi_i) \in \mathbb{G} \times \{0,1\}^* \times \{0,1\}^*$ from $\mathcal{R}$
2. For $h \in [1, q_h]$
   (a) Randomly[2] select $\boldsymbol{\alpha}_h \leftarrow (\mathbb{F}_q)^n$
   (b) Query $H$ to get $c_h(i) = H(m_i, \boldsymbol{r}^{\boldsymbol{\alpha}_h})$
3. (a) Randomly[3] select $l_i \leftarrow [1, q_h]$
         i. Set $c_i \leftarrow c_{l_i}(i)$ and $\boldsymbol{\beta}_i \leftarrow \boldsymbol{\alpha}_{l_i}$
        ii. Request $s_i \leftarrow DL_{stub}(\boldsymbol{r}^{\boldsymbol{\beta}_i} \cdot {y_i}^{c_i})$
       iii. Append $(y_i, m_i, \varpi_i) \mapsto (s_i, c_i)$ and $(l_i, \boldsymbol{\beta_i})$ to Transcript of $\mathcal{R}$
   (b) Return $\sigma_i = (s_i, c_i)$
```

Extraction of Discrete Logs: The reduction $\mathcal{R}$ uses the n simulations of its calls to $\mathcal{A}$ as described above and returns the discrete log k_0 of r_0 (with probability ε'). As described in Preliminaries, using the transcript of the algebraic reduction $\mathcal{R}$'s computation that produced k_0, we can extract the discrete logs $x_1, \ldots, x_n$ of $y_1, \ldots, y_n$, respectively, i.e., $Extract(k_0, \texttt{Trasncript}) = (x_1, \ldots, x_n)$. Now, $\mathcal{M}$ has the following system of linear equations[4] over $\mathbb{F}_q$:

$$\begin{cases} \boldsymbol{\beta_1} \cdot \boldsymbol{k} = s_1 - c_1 \cdot x_1 \\ \vdots \\ \boldsymbol{\beta_n} \cdot \boldsymbol{k} = s_n - c_n \cdot x_n \end{cases} \tag{3}$$

in the unknowns $\boldsymbol{k} = (k_1, \ldots, k_n)$, where $k_i = \log_g r_i$, $1 \leq i \leq n$.

[2] In fact, pseudorandomly in History$_h(i)$.

[3] In fact, pseudorandomly in History$_{q_h}(i)$.

[4] $\boldsymbol{a} \cdot \boldsymbol{b}$ denotes the dot product of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$.

Let $B = \begin{bmatrix} \boldsymbol{\beta}_1 \\ \vdots \\ \boldsymbol{\beta}_n \end{bmatrix}$ denote the $n \times n$ matrix over $\mathbb{F}_q$ with rows $\boldsymbol{\beta}_i \in \mathbb{F}_q^n$ for $1 \leq i \leq n$.

If B is nonsingular $\mathcal{M}$ can directly solve for $\boldsymbol{k}$. If not, $\mathcal{M}$ may not be able to solve for $\boldsymbol{k}$, for instance, if the above system is inconsistent. We consider the following three[5] mutually exclusive events:

Event A: All the l_i are distinct, i.e., for $i \neq j$, $l_i \neq l_j$. In this case, $\mathcal{M}$ checks if the matrix B is nonsingular and if so, solves for $\mathbf{k}$ from (3). Otherwise $\mathcal{M}$ outputs `FAIL`.

Event B: For some $i < j$, $l_i = l_j$ and for every such pair i, j, $\boldsymbol{\beta}_i = \boldsymbol{\beta}_j$ implies $s_i - c_i x_i = s_j - c_j x_j$. In this case, we have $m \leq n$ distinct equations and matrix B has m distinct rows.

If $\text{rank}(B) = m$, then we claim that $\mathcal{M}$ can obtain all discrete logs $k_1, \ldots, k_n$. If $m = n$, this is obvious. If $m < n$, then note that $\mathcal{M}$ has not used up all its n calls to the discrete log oracle DL_{OM}, thanks to the stub DL_{stub}. Indeed, since $s_i = \text{DLOG}(\boldsymbol{r}^{\boldsymbol{\beta}_i} \cdot g^{c_i x_i})$, it is easy to see that the i-th and j-th equations are identical if and only if $\boldsymbol{\beta}_i = \boldsymbol{\beta}_j$ and $c_i x_i = c_j x_j$. Hence, this can be detected by DL_{stub} by keeping track of its input arguments. It follows that $\mathcal{M}$ can ask $n - m$ more queries to DL_{stub} and get $n - m$ k_i's. Using these and the fact that $\text{rank}(B) = m$, $\mathcal{M}$ can compute all the k_i, $1 \leq i \leq n$.

If $\text{rank}(B) < m$, then $\mathcal{M}$ outputs `FAIL`.

Event C: For some $i < j$, $l_i = l_j$, $\boldsymbol{\beta}_i = \boldsymbol{\beta}_j$, but $s_i - c_i x_i \neq s_j - c_j x_j$. In this case, the system (3) is clearly inconsistent and so $\mathcal{M}$ cannot solve for $\boldsymbol{k}$. It outputs `FAIL`.

It is clear that $\mathcal{M}$ will correctly solve the n-DL problem on $(r_0, r_1, \ldots, r_n)$ except when it outputs `FAIL`. In the next subsection, we analyze the probability of failure of $\mathcal{M}$.

3.2 Analysis of $\mathcal{M}$

The bound on $\mathcal{M}$'s running time is easy. The only extra time it needs, compared to $\mathcal{R}$, is to compute the elements $\boldsymbol{r}^{\boldsymbol{\beta}_i}$, for using the pseudorandom generator in steps (2) and (3), for the extraction procedure on the program of $\mathcal{R}$, and to solve the linear system in Event A or Event B. We let $\mathcal{M}$ use an efficient and secure pseudorandom generator G in making its random choices from its input (y, m, ϖ). It is clear that all of these can be done in time $\text{poly}(\tau', |\mathcal{R}|, n, \log q, q_h)$.

From now on, we identify all pseudorandom choices of $\mathcal{M}$ with truly random choices for simplicity of analysis. The difference in the estimates is clearly negligible. Indeed, by standard arguments, if the difference were not negligible, then $\mathcal{M}$ can be used to efficiently distinguish truly random bits and the output of G contradicting the security of the pseudorandom generator G.

[5] Events A and B can actually be combined into one, albeit with a complicated definition, for the purposes of analysis. We separate them for clarity of presentation.

It is easy to estimate the probability of $\mathcal{M}$'s failure given Event B. The estimate in case of Event A is the same. In these cases, $\mathcal{M}$ fails if the rank of the $m \times n$ matrix B whose rows are the distinct $\boldsymbol{\beta}$'s is less than m. Since the $\boldsymbol{\beta}$'s are chosen randomly and independently, this probability is bounded by the probability that a random $m \times n$ matrix over $\mathbb{F}_q$ has rank $< m$. We recall this well-known estimate and prove it for completeness.

Lemma 1. *Let $M \in {\mathbb{F}_q}^{m\times n}$, $m \le n$, be a random matrix (its entries are uniformly and independently chosen from $\mathbb{F}_q$. Then*

$$\Pr[\mathrm{rank}(M) < m] \le \frac{q^{-(n-m)}(1-q^{-m})}{q-1}.$$

Proof. It is easy to see that the probability that the m rows of M are linearly independent is given by

$$\begin{aligned}\Pr[\mathrm{rank}(M) = m] &= \frac{(q^n-1)(q^n-q)\cdots(q^n-q^{m-1})}{q^{mn}} \\ &= \left(1-\frac{1}{q^n}\right)\left(1-\frac{1}{q^{n-1}}\right)\cdots\left(1-\frac{1}{q^{n-m+1}}\right) \\ &\ge 1-\sum_{i=0}^{m-1} q^{-n+i} \\ &= 1-q^{-n+m}\frac{1-q^{-m}}{q-1}. \qquad \square\end{aligned}$$

We thus conclude that

$$\Pr[\mathcal{M} \text{ fails } |B] \le \frac{q^{-(n-m)}(1-q^{-m})}{q-1} \le \frac{1}{q-1}. \tag{4}$$

Estimating the probability of $\mathcal{M}$'s failure given Event C takes more work. We state the bound as the following lemma and complete the analysis. We prove the lemma in Section 3.3.

Lemma 2. $\Pr[C] \le \frac{2n^{3/2}}{q_h}$.

Clearly,

$$\Pr[\mathcal{M} \text{ fails }] = \Pr[\mathcal{M} \text{ fails } |A]\Pr[A] + \Pr[\mathcal{M} \text{ fails } |B]\Pr[B] + \Pr[C].$$

Note that Event A happens when all the randomly chosen $l_i \in_R [1..q_h]$, $1 \le i \le n$, are distinct and that Event B and Event C are subsumed by the event that there are collisions among the l_i, i.e., $\exists i \ne j$ such that $l_i = l_j$. Hence their probabilities can be fairly tightly bounded by using well-known estimates on the *Birthday Problem.* However, we do not need them here.

Combining the estimates (4) and Lemma 2, we obtain

$$\Pr[\mathcal{M} \text{ fails }] \le \frac{1}{q-1}(\Pr[A]+\Pr[B]) + \frac{2n^{3/2}}{q_h} \le \frac{1}{q-1} + \frac{2n^{3/2}}{q_h}.$$

Completing the proof of Theorem 1: Meta-reduction $\mathcal{M}$ will solve the n-DL instance $(r_0, \ldots, r_n)$ if $\mathcal{M}$ succeeds (i.e. does not fail in the above sense) given that $\mathcal{R}$ succeeds in solving the discrete log instance on r_0. We assumed that $\mathcal{R}$ solves the discrete log problem with probability at least ε' given the appropriate adversary (that we simulated using $\mathcal{M}$). Hence the probability ε'' with which $\mathcal{M}$ solves n-DL is bounded by

$$\varepsilon'' \geq \varepsilon' \left(1 - \frac{2n^{3/2}}{q_h} - \frac{1}{q-1}\right). \qquad \square$$

3.3 Proof of Lemma 2

Recall that Event C occurs when, for some $i < j$, $l_i = l_j$ and $\boldsymbol{\beta}_i = \boldsymbol{\beta}_j$, but $s_i - c_i x_i \neq s_j - c_j x_j$ resulting in an inconsistent system of equations in (3). Since $\boldsymbol{\beta}_i$ and $\boldsymbol{\beta}_j$ were chosen pseudorandomly in $\text{History}_{l_i}(i)$ and $\text{History}_{l_j}(j)$, apart from negligible differences in the probability estimates, we can assume that $\boldsymbol{\beta}_i = \boldsymbol{\beta}_j$ only if $\text{History}_{l_i}(i) = \text{History}_{l_j}(j)$. In particular, this implies we can assume that $y_i = y_j$ and hence $x_i = x_j$. Note further that since $s_i = \text{DLOG}(\boldsymbol{r}^{\boldsymbol{\beta}_i} \cdot y_i^{c_i})$, $c_i x_i = c_j x_j$ if and only if $s_i = s_j$. Hence, given $\boldsymbol{\beta}_i = \boldsymbol{\beta}_j$, we can conclude (for the purposes of probability estimates) that $s_i - c_i x_i \neq s_j - c_j x_j$ if and only if $c_i \neq c_j$. Thus, Event C occurs only if two executions $\mathcal{A}_i$ and $\mathcal{A}_j$ of the adversary not only output forgeries at the same hash query ($l_i = l_j$) but also at that query instance $\boldsymbol{\beta_i} = \boldsymbol{\beta_j}$ and $c_i \neq c_j$. In particular, the two histories $\text{History}_{l_i}(i)$ and $\text{History}_{l_j}(j)$ are identical till the point l_i and then they diverge after the hash responses from the respective random oracles (controlled by $\mathcal{R}$) to the query at l_i. We call this point of divergence, the *forking point* between executions $\mathcal{A}_i$ and $\mathcal{A}_j$.

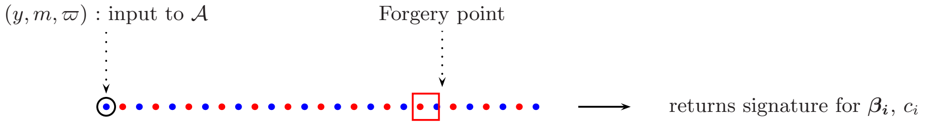
(y, m, ϖ) : input to $\mathcal{A}$
Forgery point
returns signature for $\boldsymbol{\beta_i}$, c_i

Fig. 1. A single execution of the adversary

Pictorially, we can represent an execution of the adversary by a line as in Fig. 1. The blue points symbolize inputs from $\mathcal{R}$. The first blue point is the input (y, m, ϖ) to $\mathcal{A}$ and the rest of the blue points are the responses by the random oracle (in control of $\mathcal{R}$). The red points represent the hash queries made by the adversary $\mathcal{A}$ (simulated by $\mathcal{M}$). On completing its execution, $\mathcal{A}$ returns a forgery for $(\boldsymbol{\beta}_i, c_i)$ pair for a random selection of $l_i \in [1..q_h]$. This is denoted by the red square and we will call this the *forgery point*. We also abbreviate this execution of the adversary as in Fig. 2. We will use two lines to represent two different executions of $\mathcal{A}$. Two lines with overlapping initial segments represent identical histories over the shared region. At the forking point, the responses from the respective random oracles are different. This is shown in Fig. 3.

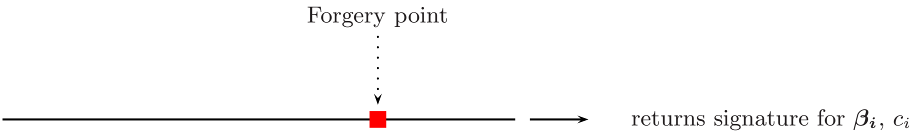

Fig. 2. A single execution of the adversary: a simplified view

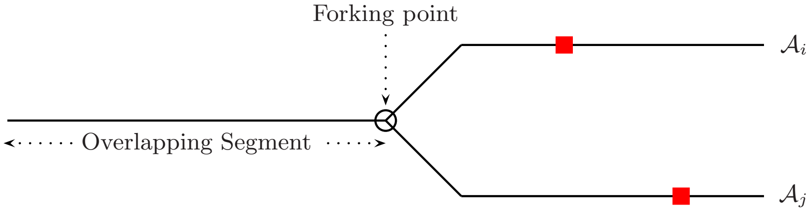

Fig. 3. Two executions sharing a common history

Thus, the simulation of an execution $\mathcal{A}_j$ fails if its forgery point l_j (the red square) is also a point of forking (a black circle), i.e., $c_{l_j} \neq c_{l_i}$, from a previous execution $\mathcal{A}_i$ that returned a forgery at that point, i.e., $l_i = l_j$.

Let F_i be the set of points on which if a forgery is returned by $\mathcal{A}_i$, then the meta-reduction $\mathcal{M}$ fails. We call F_i the set of *failure causing points* of $\mathcal{A}_i$. In other words, it is the set of points at which $\mathcal{A}_i$ forks from some previous executions of $\mathcal{A}$ and at each of these points, the corresponding previous executions of $\mathcal{A}$ have returned a forgery. More formally,

$$\mathrm{F}_i := \{l_j : j < i \text{ such that } \mathrm{History}_{l_j}(i) = \mathrm{History}_{l_j}(j) \text{ and } c_{l_j}(i) \neq c_{l_j}(j)\}.$$

To illustrate, in Fig. 4, we see the execution $\mathcal{A}_i$ of the adversary and it can be seen that there are previous executions $\mathcal{A}_{j_1}$, $\mathcal{A}_{j_2}$ and $\mathcal{A}_{j_3}$ from which $\mathcal{A}_i$ forks off at the points l_{j_1}, y, and l_{j_3}, respectively. The executions $\mathcal{A}_{j_1}$, $\mathcal{A}_{j_2}$ and $\mathcal{A}_{j_3}$ had previously returned forgeries at points l_{j_1}, l_{j_2}, and l_{j_3}, respectively. Now, if $\mathcal{A}_i$ returns a forgery at l_{j_1} or l_{j_3}, then $\mathcal{M}$ will fail. Thus the set F_i consists of points l_{j_1} and l_{j_3}. Note that we can comment on the set F_i only when the forking of $\mathcal{A}_i$ from the previous executions of $\mathcal{A}$ is known.

If, for the i^{th} execution of the adversary, F_i is the set of failure causing points, then we define $\mathfrak{F}_i$ as

$$\mathfrak{F}_i := \{x \mid x \in \mathrm{F}_i \wedge x < l_i\} \cup \{l_i\}.$$

Let $\mathfrak{M}_i$ denote the collection of all $\mathfrak{F}_i$'s till the i^{th} execution of the adversary, *i.e.*, $\mathfrak{M}_i = \{\phi, \mathfrak{F}_1, \mathfrak{F}_2 \ldots \mathfrak{F}_i\}$.

The following claim explains why we include $\mathfrak{F}_i$ instead of $\mathrm{F}_i \cup \{l_i\}$ in $\mathfrak{M}_i$. The main idea is to enable a clean description of F_{i+1} based on F_j for $j < i$.

Claim. For the $(i+1)^{th}$ execution of the adversary, F_{i+1} will be one of the sets in $\mathfrak{M}_i$, i.e., $F_{i+1} \in \mathfrak{M}_i$.

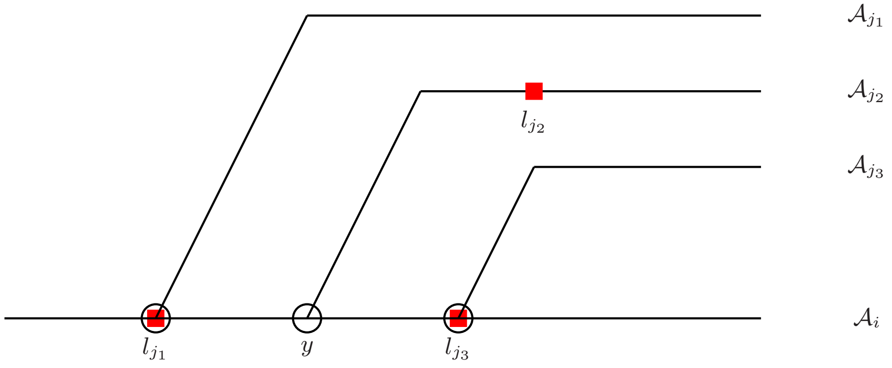

Fig. 4. Forging points, forking points, and the set of failure points

Proof of Claim: Let $\mathrm{F}_{i+1} = \{l_{j_1} < l_{j_2} < \cdots < l_{j_t}\}$, where $j_1, j_2, \cdots, j_t < i+1$ are previous executions that forge and fork (from $\mathcal{A}_{i+1}$) at the locations $l_{j_1}, \ldots, l_{j_t}$ respectively. For simplicity of notation, let $k := j_t$. We claim that $\mathrm{F}_{i+1} = \mathfrak{F}_k$.

First, we prove that $\mathrm{F}_{i+1} \subseteq \mathfrak{F}_k$. Clearly, $l_k \in \mathfrak{F}_k$. Hence, let us assume $l_j \in \mathrm{F}_{i+1}$, where $l_j < l_k$. It suffices to show that $l_j \in \mathrm{F}_k$. Since $l_j \in \mathrm{F}_{i+1}$, $j < i+1$, $\mathrm{History}_{l_j}(j) = \mathrm{History}_{l_j}(i+1)$, and $c_{l_j}(j) \neq c_{l_j}(i+1)$. On the other hand, since l_k is also in F_{i+1}, $k < i+1$ and $\mathrm{History}_{l_k}(k) = \mathrm{History}_{l_k}(i+1)$. But $l_j < l_k$. It follows that $\mathrm{History}_{l_j}(k) = \mathrm{History}_{l_j}(i+1) = \mathrm{History}_{l_j}(j)$ and $c_{l_j}(k) = c_{l_j}(i+1) \neq c_{l_j}(j)$. Hence $l_j \in \mathrm{F}_k$.

Next, we prove that $\mathfrak{F}_k \subseteq \mathrm{F}_{i+1}$. By definition, $l_k \in \mathfrak{F}_k$. So, consider an $l_k \neq l_j \in \mathfrak{F}_k$. By construction of $\mathfrak{F}_k$, $l_j < l_k$ for some $j < k$ and $l_j \in F_k$. Hence $\mathrm{History}_{l_j}(j) = \mathrm{History}_{l_j}(k)$ and $c_{l_j}(j) \neq c_{l_j}(k)$. Since $l_k \in \mathrm{F}_{i+1}$, $\mathrm{History}_{l_k}(k) = \mathrm{History}_{l_k}(i+1)$ and $c_{l_k}(k) \neq c_{l_k}(i+1)$. Since $l_j < l_k$, $\mathrm{History}_{l_j}(k) = \mathrm{History}_{l_j}(i+1)$ and $c_{l_j}(k) = c_{l_j}(i+1)$. Combining this with the previous relation, we obtain $\mathrm{History}_{l_j}(j) = \mathrm{History}_{l_j}(i+1)$ and $c_{l_j}(j) \neq c_{l_j}(i+1)$. It follows that $l_j \in \mathrm{F}_{i+1}$.

This concludes the proof of the claim.

Clearly, the size of $\mathfrak{F}_i$ can be at most one greater than the maximum of the sizes of all $\mathfrak{F}_j$ $(1 \leq j < i)$ since $|\mathfrak{F}_i| \leq |\mathrm{F}_i|+1$ and $|\mathrm{F}_i| \leq \max\{|\mathfrak{F}_j| : 1 \leq j \leq i-1\}$.

Thus, by construction, $\mathfrak{F}_i$ are sets comprising of integers in increasing order. The size of a $\mathfrak{F}_i$ increases only if it is the set of failure points for the current execution and the forgery point for this execution is greater than all elements in the set $\mathfrak{F}_i$. As the forgery point is randomly picked by $\mathcal{M}$ from the set $[1, q_h]$, the maximum of $|\mathfrak{F}_i|$ is at most the length of a longest increasing sub-sequence in a random sequence of n distinct integers from $[1, q_h]$ (we may assume $n \ll q_h$ as otherwise, we are already done). It is easy to see that any permutation of $[n]$ is equally represented by the ordering on such a random sequence. Let λ_n denote the random variable denoting the length of a longest increasing subsequence in a random permutation of $[n]$. The following result due to Kerov and Versik (see Pilpel's paper [Pil90]) will be useful.

Theorem 2 (Kerov-Versik 1977). *For sufficiently large n, $\mathbb{E}[\lambda_n] \leq 2\sqrt{n}$.*

We can now estimate $\Pr[C]$. Clearly, C occurs if for at least one execution $\mathcal{A}_i$, the forgery point l_i falls into the failure causing set of points F_i. Since l_i is chosen uniformly at random from $[1..q_h]$, we have $\Pr[C] \leq \sum_{i=1}^{n} |F_i|/q_h$. Now, $|F_i|$ is at most $\max_{j=1}^{i-1} |\mathfrak{F}_j| \leq \max_{j=1}^{n} |\mathfrak{F}_j|$. For every n-sequence (l_i) from $[1..q_h]^n$, $\max_{j=1}^{n} |\mathfrak{F}_j|$ is upper bounded by the length of the longest subsequence in the corresponding permutation of $[n]$. Hence $\mathbb{E}[\max_{j=1}^{n} |\mathfrak{F}_j|] \leq \mathbb{E}[\lambda_n]$. Thus,

$$\begin{aligned}
\Pr[C] &= \sum_{t=1}^{n} \Pr[C|\max_{j=1}^{n} |\mathfrak{F}_j| = t] \cdot \Pr[\max_{j=1}^{n} |\mathfrak{F}_j| = t] \\
&\leq \sum_{t=1}^{n} \sum_{i=1}^{n} \frac{|F_i| \text{ given } \max_{j=1}^{n} |\mathfrak{F}_j| = t}{q_h} \cdot \Pr[\max_{j=1}^{n} |\mathfrak{F}_j| = t] \\
&\leq \frac{n}{q_h} \sum_{t=1}^{n} t \Pr[\max_{j=1}^{n} |\mathfrak{F}_j| = t] \\
&\leq \frac{n}{q_h} \mathbb{E}[\max_{j=1}^{n} |\mathfrak{F}_j|] \leq \frac{n}{q_h} \mathbb{E}[\lambda_n] \\
&\leq \frac{2n^{3/2}}{q_h} \quad \text{using Theorem 2.}
\end{aligned}$$

This completes the proof of Lemma 2. □

3.4 Remarks on Tightness of the Lower Bound

The adversary we have simulated in Section 3 behaves randomly in the sense that the probability of a forgery being returned for any of the hash queries in a given execution is the same. It also has the property that the probability of success is uniformly distributed across all executions. For this restricted adversary we claim that the lower bound of $q_h^{2/3}$ is indeed tight.

To justify our claim we construct a reduction $\mathcal{R}$ that, using the adversary $\mathcal{A}$ (an (ε, τ) UF-KOA 'uniform' adversary) breaks the Discrete Log problem. The reduction $\mathcal{R}$ tries to obtain two forgeries on the same message m (under the same public key y) but for different values of the hash responses. The adversary $\mathcal{A}$ accepts (y, m, ϖ) as input and then makes queries to the hash oracle. We represent the sequence of responses given by the oracle by $\boldsymbol{c} = \langle c_1(j), c_2(j), \ldots, c_{q_h}(j)\rangle$ and the subsequence of the first i responses by $\boldsymbol{c_i} = \langle c_1(j), c_2(j), \ldots, c_i(j)\rangle$. The reduction $\mathcal{R}$ calls the adversary $\mathcal{A}$ with the same (y, m, ϖ) but different $\boldsymbol{c}$. We define $Ind(\boldsymbol{c})$ as the index of the hash query on which the forgery is returned by an execution of the adversary. We let $Ind(\boldsymbol{c}) = \infty$ in case the forgery is returned for a hash value never obtained by calling the hash oracle (or if $\mathcal{A}$ fails).

Let $\mathcal{S} = \{\boldsymbol{c} \mid Ind(\boldsymbol{c}) < \infty\}$ and $\mathcal{S}_i = \{\boldsymbol{c} \mid Ind(\boldsymbol{c}) = i\}$. Then $Pr[\mathcal{S}] \geq \varepsilon - \frac{1}{q-1} = \nu$, as the probability that the adversary correctly guesses the hash value (without making the hash query) is $\frac{1}{q-1}$. Assuming that the adversary outputs its forgeries uniformly across the hash-query indices and its several executions *i.e.* $Ind(\boldsymbol{c})$ is pseudo-random in $\boldsymbol{c}$, we get $\Pr[\mathcal{S}_i] \geq \nu/q_h, \forall i \in [1, q_h]$.

The reduction $\mathcal{R}$ divides the q_h hash query points into λ intervals of equal width q_h/λ as shown in Fig. 5. The circles represent the hash query points $[1, q_h]$ at which an instance of execution of the adversary could return a forgery and $\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_\lambda$ are equal-sized partitions of these points. Then, $\Pr[Ind(\boldsymbol{c}) \in \mathcal{K}_i] \geq \nu/\lambda, \forall i \in [1, \lambda]$. $\mathcal{R}$ runs the adversary $\mathcal{A}$ in $\lambda + 1$ phases. In phase 1, $\mathcal{R}$ invokes the adversary at most λ/ν times (changing $\boldsymbol{c}$ each time) with the hope of getting a forgery in partition $\mathcal{K}_1$ and moves to the next phase as soon as it gets the desired forgery. The probability of obtaining a forgery in the first interval in λ/ν runs of $\mathcal{A}$ is $1-(1-\nu/\lambda)^{\lambda/\nu} \geq 1-e^{-1}$. Let $\boldsymbol{c}^{(1)}$ denote the set of hash responses for the successful execution in phase 1 and l_1 be the index of the forgery for this execution. In case of success in phase 1, $\mathcal{R}$ calls the adversary in phase 2 at most λ/ν times with $\boldsymbol{c}$'s such that $\boldsymbol{c}_{l_1} = \boldsymbol{c}_{l_1}^{(1)}$ is satisfied for all. Otherwise in case of failure in phase 1 ($l_1 = \infty$), $\mathcal{R}$ executes the phase 2 in a similar fashion as phase 1 but hoping this time to get the forgery in the second partition $\mathcal{K}_2$. In general, in phase i ($i \in [2, \lambda]$), $\mathcal{R}$ calls the adversary at most λ/ν times with $\boldsymbol{c}$'s such that $\boldsymbol{c}_{l_j} = \boldsymbol{c}_{l_j}^{(j)}$ (l_j - index of hash query for the successful execution in the most recent successful phase (j) and $\boldsymbol{c}^{(j)}$ is the sequence of hash responses for that execution) with the hope of getting a forgery on some hash query $l_i \in \mathcal{K}_i$. The probability of getting a forgery in the interval $\mathcal{K}_i$ is also $1-(1-\nu/\lambda)^{\lambda/\nu} \geq 1-e^{-1}$. Hence, at the end of phase λ, $\mathcal{R}$ expects to have $(1-e^{-1}) \cdot \lambda$ forgeries in $\frac{\lambda}{\nu} \cdot \lambda$ executions of the adversary $\mathcal{A}$. Let the set of forgery points be $\mathcal{I}$. In phase $\lambda+1$, $\mathcal{R}$ runs the adversary $\mathcal{A}$ with $\boldsymbol{c}$'s such that $\boldsymbol{c}_{l_j} = \boldsymbol{c}_{l_j}^{(j)}$, where j, l_j are defined as before. Then, $\Pr[Ind(\boldsymbol{c}) \in \mathcal{I}] \geq \nu\lambda(1-e^{-1})/q_h$. The number of executions required to get a forgery on one of the points in $\mathcal{I}$ is $q_h/(\nu\lambda(1-e^{-1}))$, and this happens with a probability $1-e^{-1}$. The total number of required executions of $\mathcal{A}$ are $\frac{\lambda^2}{\nu} + \frac{q_h}{\nu\lambda(1-e^{-1})}$, which takes the optimal value for $\lambda = \Theta(q_h^{1/3})$ for which the number of executions is $\Omega(\frac{q_h^{2/3}}{\nu})$. Thus at the end of phase $\lambda+1$, $\mathcal{R}$ obtains two forgeries (c_1, s_1) and (c_2, s_2) on the same message m under the same public key y and the same randomness k (see Definition 3) but different hash responses c_1 and c_2. If the reduction R uses the discrete-log challenge g^x as the public key in the above interactions, it can obtain x as $\frac{s_1-s_2}{c_1-c_2}$.

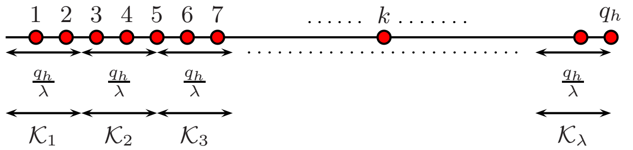

Fig. 5. Dividing the hash query points into equal intervals of width q_h/λ

4 Conclusions and Open Problems

In this paper we improved the lower bound from $q_h^{1/2}$ to $q_h^{2/3}$ on the loss factor in a security reduction that converts a forgery attack on Schnorr signature scheme to an algorithm to the discrete log problems. This means that to achieve the same level of security as before, one needs to employ larger parameters than before. We also presented a new attack strategy for solving the discrete log problem using a restricted class of Schnorr signature forgers more efficiently. This attack strategy indicates that the lower bound $q_h^{2/3}$ is tight for the restricted adversary we simulate. Since the lower bound proof relies on the n-DL assumption and restricts itself to algebraic reductions, the gap between the lower bound of $q_h^{2/3}$ and the upper bound of q_h may in some sense be inevitable.

One of the most interesting open problems is to prove that the lower bound of $q_h^{2/3}$ is tight for a general adversary. Another major open question is to understand relationship between the DL problem and the n-DL problem.

References

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

[BV98] Boneh, D., Venkatesan, R.: Breaking RSA not be equivalent to factoring. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 59–71. Springer, Heidelberg (1998)

[MvOV96] Menezes, A.J., van Oorschot, P.C., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)

[Pil90] Pilpel, S.: Descending subsequences of random permutations. J. Comb. Theory Ser. A 53(1), 96–116 (1990)

[PS96] Pointcheval, D., Stern, J.: Security proofs for signature schemes. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 387–398. Springer, Heidelberg (1996)

[PS00] Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of Cryptology 13(3), 361–396 (2000)

[PV05] Paillier, P., Vergnaud, D.: Discrete-log-based signatures not be equivalent to discrete log. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 1–20. Springer, Heidelberg (2005)

[Sch90] Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 688–689. Springer, Heidelberg (1990)