SCIS 2013 The 30th Symposium on
Cryptography and Information Security
kyoto, Japan, Jan. 22 - 25, 2013
The Institute of Electronics,
Information and Communication Engineers

Algorithmic Randomness and the Generic Group Model

Kohtaro Tadaki [*] Norihisa Doi [*]

Abstract— In modern cryptography, the generic group model is used as an *imaginary* framework in which the security of a cryptographic scheme is discussed. In particular, the generic group model is often used to discuss the computational hardness of problems, such as the discrete logarithm problem and the Diffie-Hellman problem, which are used as a computational hardness assumption to prove the security of a cryptographic scheme. In this paper, we apply the concepts and methods of *algorithmic randomness* to the generic group model, and consider the secure instantiation of the generic group, i.e., a random encoding of the group elements. We show that the generic group can be instantiated by a specific computable function while keeping the computational hardness originally proved in the generic group model.

Keywords: generic group model, provable security, instantiation, random encoding function, discrete logarithms, Diffie-Hellman problem, algorithmic randomness, computable function

1 Introduction

In modern cryptography, the generic group model is used as an *imaginary* framework in which the security of a cryptographic scheme is discussed. In particular, the generic group model is often used to discuss the computational hardness of problems, such as the discrete logarithm problem and the Diffie-Hellman problem, which are used as a computational hardness assumption to prove the security of a cryptographic scheme. Since the generic group, i.e., a random encoding of the group elements, is an imaginary object, even if the security of a cryptographic scheme or the hardness of a computational problem is proved in the generic group model, the generic group has to be instantiated using a concrete finite cyclic group whose group operations are efficiently computable if we want to use the cryptographic scheme in the real world. Once the generic group is instantiated, however, the original security proof or hardness proof in the generic group model is spoiled and goes back to square one. Actually, it is not clear how much the instantiation can maintain the security or hardness originally proved in the generic group model, nor is it clear whether the generic group can be instantiated somehow while keeping the original security or hardness.

In the present paper we investigate this problem, based on concepts and methods of *algorithmic randomness*, also known as *algorithmic information theory*. In algorithmic randomness, the notion of a *random real* plays a central role. It is an individual infinite binary sequence which is classified as "random", and not a random variable such as the generic group. Algorithmic randomness enables us to classify an individual infinite binary sequence into random or not. It originated in the groundbreaking works of Solomonoff, Kolmogorov, and Chaitin in the mid-1960s. They independently introduced the notion of *program-size complexity*, also known as *Kolmogorov complexity*, in order to quantify the randomness of an individual object. In the 21st century, algorithmic randomness is making remarkable progress through close interaction with recursion theory [11, 4].

In this paper we show that the generic group can be instantiated by a specific computable function while keeping the computational hardness originally proved in the generic group model.

In our former works [13, 14] we investigated the secure instantiation of the random oracle in the random oracle model [2]. In [13] we investigated the instantiation of the random oracle by a random real in a cryptographic scheme already proved secure in the random oracle model. In this line we presented equivalent conditions for a specific oracle instantiating the random oracle to keep a cryptographic scheme secure, using a concept of algorithmic randomness, i.e., a variant of Martin-Löf randomness. Based on this, in particular we showed that the security proved in the random oracle model is firmly maintained after instantiating the random oracle by a random real.

In the sequel [14] we introduced the notion of *effective security*, which is a constructive strengthen of conventional (non-constructive) notions of security. We considered signature schemes in the random oracle model, and showed that some specific *computable* function can instantiate the random oracle while keeping the effective security originally proved in the random oracle model. We demonstrated that the effective security is a natural alternative to the conventional security notions

* Research and Development Initiative, Chuo University, 1-13-27 Kasuga, Bunkyo-ku, Tokyo 112-8551, Japan. E-mail: tadaki@kc.chuo-u.ac.jp, doi@doi.ics.keio.ac.jp WWW: http://www2.odn.ne.jp/tadaki/

in modern cryptography by reconsidering the security notions required in modern cryptography.

In this paper, we show that the same concepts and methods which we developed for investigating the secure instantiation of the random oracle can be properly applied to the problem of the secure instantiation of the generic group.

2 Preliminaries

We start with some notation about numbers and strings which will be used in this paper. $\#S$ is the cardinality of S for any set S. $\mathbb{N} = \{0, 1, 2, 3, \dots\}$ is the set of natural numbers, and $\mathbb{N}^+$ is the set of positive integers. $\mathbb{Q}$ is the set of rational numbers, and $\mathbb{R}$ is the set of real numbers. For any integer $N \geq 2$, $\mathbb{Z}_N$ denotes the additive group of integers modulo N and sometimes the set $\{0, 1, \dots, N-1\}$.

$\{0,1\}^*$ is the set of finite binary strings. For any $x \in \{0,1\}^*$, $|x|$ is the *length* of x. For any $n \in \mathbb{N}$, we denote by $\{0,1\}^n$ the set $\{\, x \mid x \in \{0,1\}^* \ \&\ |x| = n\}$.

3 Lebesgue Outer Measure on Families of Encoding Functions

Let $n \in \mathbb{N}^+$. An *encoding function into n bitstrings* is a bijective function mapping $\{0, 1, \dots, 2^n - 1\}$ to $\{0,1\}^n$. We denote by Encf_n the set of all encoding functions into n bitstrings. Note that $\#\mathsf{Encf}_n = (2^n)!$.

A *family of encoding functions* is an infinite sequence $\{\sigma_n\}_{n\in\mathbb{N}^+}$ such that σ_n is an encoding function into n bitstrings for every $n \in \mathbb{N}^+$. A family of encoding functions serves as an instantiation of an infinite sequence of the generic groups over all security parameters. We denote by Encf^∞ the set of all families of encoding functions. Namely,

$$\mathsf{Encf}^\infty := \prod_{k=1}^{\infty} \mathsf{Encf}_k = \mathsf{Encf}_1 \times \mathsf{Encf}_2 \times \mathsf{Encf}_3 \times \cdots\cdots .$$

On the other hand, a *finite family of encoding functions* is a finite sequence $s = (\sigma_1, \dots, \sigma_n)$ such that σ_k is an encoding function into k bitstrings for every $k = 1, \dots, n$. Here, n is called the *length* of s and denoted by $|s|$. A finite family of encoding functions is an initial segment (finite prefix) of a family of encoding functions. For each $n \in \mathbb{N}$, we denote by Encf^n the set of all finite families of encoding functions of length n. Namely,

$$\mathsf{Encf}^n := \prod_{k=1}^{n} \mathsf{Encf}_k = \mathsf{Encf}_1 \times \cdots \times \mathsf{Encf}_n.$$

Note that $\mathsf{Encf}^0 = \{\lambda\}$ where $\lambda := ()$ is the *empty sequence*. We denote by Encf^* the set of all finite families of encoding functions, i.e., $\mathsf{Encf}^* := \bigcup_{n=0}^{\infty} \mathsf{Encf}^n$. For any sequences $s = (\sigma_1, \dots, \sigma_n)$ and $t = (\tau_1, \dots, \tau_m)$ in Encf^*, we say that s is a *prefix* of t if $n \leq m$ and $\sigma_k = \tau_k$ for all $k \leq n$. A subset P of Encf^* is called *prefix-free* if no sequence in P is a prefix of another sequence in P.

In this paper we use the notion of Lebesgue outer measure on Encf^∞, which is defined as follows. For any sequence $s = (\sigma_1, \dots, \sigma_n) \in \mathsf{Encf}^*$, $I(s)$ is defined as the set of all families $\{\tau_k\}_{k\in\mathbb{N}^+}$ of encoding functions for which $\sigma_k = \tau_k$ for all $k \leq n$, and $|I(s)|$ is defined by

$$|I(s)| := \prod_{k=1}^{n} \frac{1}{\#\mathsf{Encf}_k} = \frac{1}{\#\mathsf{Encf}_1 \times \cdots \times \#\mathsf{Encf}_n}.$$

Note that $I(\lambda) = \mathsf{Encf}^\infty$ and $|I(\lambda)| = 1$. *Lebesgue outer measure $\mathcal{L}$ on* Encf^∞ is a function mapping any subset A of Encf^∞ to a non-negative real, and is defined by

$$\mathcal{L}(A) := \inf \sum_{n=1}^{\infty} |I(s_n)|,$$

where the infimum extends over all infinite sequences $s_1, s_2, \dots \in \mathsf{Encf}^*$ for which $A \subset \bigcup_{n=1}^{\infty} I(s_n)$.

In this paper, we use the properties of $\mathcal{L}$ presented in Proposition 3.1 below. For any subset T of Encf^*, we denote by $[T]^\prec$ the set $\bigcup_{s\in T} I(s)$.

Proposition 3.1.

(i) For every prefix-free set $P \subset \mathsf{Encf}^*$,

$$\mathcal{L}\left([P]^\prec\right) = \sum_{s\in P} |I(s)|.$$

Therefore $\mathcal{L}(\emptyset) = \mathcal{L}\left([\emptyset]^\prec\right) = 0$ and $\mathcal{L}(\mathsf{Encf}^\infty) = \mathcal{L}\left([\{\lambda\}]^\prec\right) = 1$.

(ii) $\mathcal{L}(A) \leq \mathcal{L}(B)$ for every sets $A \subset B \subset \mathsf{Encf}^\infty$.

(iii) $\mathcal{L}\left(\bigcup_i A_i\right) \leq \sum_i \mathcal{L}(A_i)$ for every sequence $\{A_i\}_{i\in\mathbb{N}}$ of subsets of Encf^∞.

(iv) $\mathcal{L}\left(\bigcup_i [P_i]^\prec\right) = \sum_i \mathcal{L}\left([P_i]^\prec\right)$ for every finite or infinite sequence $\{P_i\}_i$ of subsets of Encf^* such that $[P_i]^\prec \cap [P_j]^\prec = \emptyset$ for every $i \neq j$. □

A function $f\colon \mathbb{N} \to \mathbb{Q}$ is called *computable* if there exists a deterministic Turing machine which on every input $n \in \mathbb{N}$ halts and outputs $f(n)$. A computable function is also called a *total recursive function*. A real a is called *computable* if there exists a computable function $g\colon \mathbb{N} \to \mathbb{Q}$ such that $|a - g(k)| < 2^{-k}$ for all $k \in \mathbb{N}$. For any subset S of Encf^*, we say that S is *recursively enumerable* (*r.e.*, for short) if there exists a deterministic Turing machine which on every input $s \in \mathsf{Encf}^*$ halts if and only if $s \in S$. Note here that any sequence in Encf^* is a finite object, which can be represented as a finite binary string, and thus can be manipulated by a Turing machine. Finally, a family $\{\sigma_n\}_{n\in\mathbb{N}^+}$ of encoding functions is called *computable* if there exists a deterministic Turing machine which on every input (n, x) with $0 \leq x \leq 2^n - 1$ halts and outputs $\sigma_n(x)$.

Theorem 3.2 below plays a crucial role in this paper. It is a modification of the result presented as Exercise 1.9.21 in Nies's textbook [11] of algorithmic randomness. We can prove this theorem based on the properties of $\mathcal{L}$ in Proposition 3.1, as well as the computability of the mapping $\mathbb{N}^+ \ni n \mapsto \#\mathsf{Encf}_n$.

Theorem 3.2. *Let S be an r.e. subset of Encf^*. Suppose that $\mathcal{L}\left([S]^{\prec}\right) < 1$ and $\mathcal{L}\left([S]^{\prec}\right)$ is a computable real. Then there exists a computable family of encoding functions which is not in $[S]^{\prec}$.*

Proof. We define $F\colon \mathsf{Encf}^* \to [0,1]$ by

$$F(t) = \mathcal{L}\left([S]^{\prec} \cap I(t)\right).$$

First, we show that the real-valued function F is computable, i.e., there exists a computable function $f\colon \mathsf{Encf}^* \times \mathbb{N} \to \mathbb{Q}$ such that

$$|F(t) - f(t,k)| < 2^{-k} \tag{1}$$

for all $t \in \mathsf{Encf}^*$ and $k \in \mathbb{N}$.

Let $n \in \mathbb{N}$. Since $\bigcup_{t\in\mathsf{Encf}_n} I(t) = \mathsf{Encf}^\infty$ we have

$$\bigcup_{t\in\mathsf{Encf}_n} [S]^{\prec} \cap I(t) = [S]^{\prec}$$

and $\left([S]^{\prec} \cap I(t)\right) \cap \left([S]^{\prec} \cap I(t')\right) = \emptyset$ for any distinct $t, t' \in \mathsf{Encf}_n$. Note that, for every $t \in \mathsf{Encf}^*$, there is $S' \subset \mathsf{Encf}^*$ such that $[S]^{\prec} \cap I(t) = [S']^{\prec}$.[1] It follows from (iv) of Proposition 3.1 that

$$\sum_{u\in\mathsf{Encf}_n} F(u) = \mathcal{L}\left([S]^{\prec}\right) \tag{2}$$

for every $n \in \mathbb{N}$.

Since S is an r.e. set, there is a deterministic Turing machine which enumerates S, i.e., there is a deterministic Turing machine which on every input $m \in \mathbb{N}^+$ outputs a finite subset S_m of S, where $S_m \subset S_{m+1}$ for every $m \in \mathbb{N}^+$ and $\bigcup_{m=1}^{\infty} S_m = S$. Therefore, for each $t \in \mathsf{Encf}^*$, we have $S_m \cap I(t) \subset S_{m+1} \cap I(t)$ for every $m \in \mathbb{N}^+$ and $\bigcup_{m=1}^{\infty} (S_m \cap I(t)) = S \cap I(t)$. Using (ii) and (iv) of Proposition 3.1 it is easy to show that, for each $t \in \mathsf{Encf}^*$, $\mathcal{L}(S_m \cap I(t)) \le \mathcal{L}(S_{m+1} \cap I(t))$ for every $m \in \mathbb{N}^+$ and $\lim_{m\to\infty} \mathcal{L}(S_m \cap I(t)) = F(t)$. It follows from (2) that, for each $n \in \mathbb{N}$,

$$\begin{aligned}&\sum_{u\in\mathsf{Encf}_n} \mathcal{L}(S_m \cap I(u)) \\ &\le F(t) + \sum_{u\in\mathsf{Encf}_n \text{ and } u\neq t} \mathcal{L}(S_m \cap I(u)) \\ &\le \mathcal{L}\left([S]^{\prec}\right)\end{aligned} \tag{3}$$

for every $m \in \mathbb{N}^+$ and

$$\lim_{m\to\infty} \sum_{u\in\mathsf{Encf}_n} \mathcal{L}(S_m \cap I(u)) = \mathcal{L}\left([S]^{\prec}\right). \tag{4}$$

Note that, any given finite set $P \subset \mathsf{Encf}^*$, one can compute a finite prefix-free set $Q \subset \mathsf{Encf}^*$ such that $[Q]^{\prec} = [P]^{\prec}$. It follows from (i) of Proposition 3.1 and the computability of the mapping $\mathbb{N}^+ \ni l \mapsto \#\mathsf{Encf}_l$ that, any given n and t, one can compute the rational $\mathcal{L}(S_n \cap I(t))$. Therefore, any given n and m, one can compute the rational $\sum_{u\in\mathsf{Encf}_n} \mathcal{L}(S_m \cap I(u))$.

[1] As such S', the set $T \cup \{s \in S \mid t \text{ is a prefix of } s\}$ suffices, where $T = \{t\}$ if there is a prefix $s \in S$ of t and $T = \emptyset$ otherwise.

Now, since $\mathcal{L}(S)$ is a computable real by the assumption, there exists a computable function $g\colon \mathbb{N} \to \mathbb{Q}$ such that

$$\left|\mathcal{L}\left([S]^{\prec}\right) - g(k)\right| < 2^{-k} \tag{5}$$

for all $k \in \mathbb{N}$. It follows from (4) that there exists a computable function $h\colon \mathsf{Encf}^* \times \mathbb{N} \to \mathbb{N}^+$ such that, for every $t \in \mathsf{Encf}^*$ and $k \in \mathbb{N}$,

$$g(k) - 2^{-k} < \sum_{u\in\mathsf{Encf}_{|t|}} \mathcal{L}\left(S_{h(t,k)} \cap I(u)\right).$$

But, by (3) and (5), the right-hand side is at most

$$F(t) + \sum_{u\in\mathsf{Encf}_n \text{ and } u\neq t} \mathcal{L}\left(S_{h(t,k)} \cap I(u)\right) \le g(k) + 2^{-k}.$$

Thus we define a function $f\colon \mathsf{Encf}^* \times \mathbb{N} \to \mathbb{Q}$ by

$$f(t,k) = g(k) - \sum_{u\in\mathsf{Encf}_n \text{ and } u\neq t} \mathcal{L}\left(S_{h(t,k)} \cap I(u)\right).$$

We then see that the rational-valued function f is computable and (1) holds, as desired.

Next, we construct a computable family $\{\sigma_n\}_{n\in\mathbb{N}^+}$ of encoding functions such that

$$\mathcal{L}\left([S]^{\prec} \cap I((\sigma_1,\dots,\sigma_m))\right) < \prod_{k=1}^{m} \frac{1}{\#\mathsf{Encf}_k} \tag{6}$$

holds for all $m \in \mathbb{N}$. We do this by the following inductive procedure: We first note by (ii) and (i) of Proposition 3.1 that

$$\begin{aligned}\mathcal{L}\left([S]^{\prec} \cap I(t)\right) &\le \mathcal{L}(I(t)) = |I(t)| \\ &= \prod_{k=1}^{m} \frac{1}{\#\mathsf{Encf}_k}\end{aligned} \tag{7}$$

holds for every $m \in \mathbb{N}$ and $t \in \mathsf{Encf}^m$. Since

$$\bigcup_{\tau\in\mathsf{Encf}_{m+1}} I((\tau_1,\dots,\tau_m,\tau)) = I((\tau_1,\dots,\tau_m))$$

holds for every $m \in \mathbb{N}$ and $(\tau_1,\dots,\tau_m) \in \mathsf{Encf}^m$, we also note by (iv) of Proposition 3.1 that

$$\begin{aligned}&\sum_{\tau\in\mathsf{Encf}_{m+1}} \mathcal{L}\left([S]^{\prec} \cap I((\tau_1,\dots,\tau_m,\tau))\right) \\ &\qquad = \mathcal{L}\left([S]^{\prec} \cap I((\tau_1,\dots,\tau_m))\right)\end{aligned} \tag{8}$$

for every $m \in \mathbb{N}$ and $(\tau_1,\dots,\tau_m) \in \mathsf{Encf}^m$. Let $s_n = (\sigma_1,\dots,\sigma_n) \in \mathsf{Encf}^n$ for each $n \in \mathbb{N}$.

Initially, we set $n := 0$ and $s_n := \lambda$. Then, obviously, the property (6) holds for $m = n$, which is exactly the assumption $\mathcal{L}\left([S]^{\prec}\right) < 1$ of the theorem.

For an arbitrary $n \in \mathbb{N}$, assume that we have constructed $s_n = (\sigma_1,\dots,\sigma_n)$ and (6) holds for $m = n$. It follows from (7) with $m = n+1$ and (6) with $m = n$ that

$$F((\sigma_1,\dots,\sigma_n,\tau_0)) < \prod_{k=1}^{n+1} \frac{1}{\#\mathsf{Encf}_k} \tag{9}$$

for some $\tau_0 \in \mathsf{Encf}_{n+1}$. Since F is a computable real function, by computing the approximation of $F((\sigma_1, \dots, \sigma_n, \tau))$ with an arbitrary precision for each $\tau \in \mathsf{Encf}_{n+1}$, one can find τ_0 for which (9) holds, and then set $\sigma_{n+1} := \tau_0$ and $s_{n+1} := (\sigma_1, \dots, \sigma_n, \tau_0)$. It follows that (6) holds for $m = n + 1$.

Thus, any given $n \in \mathbb{N}^+$, one can compute σ_n by the above procedure. This implies that the family $\{\sigma_n\}_{n\in\mathbb{N}^+}$ of encoding functions is computable.

Now, assume contrarily that $\{\sigma_n\}_{n\in\mathbb{N}^+} \in [S]^{\prec}$. Then there is $n \in \mathbb{N}$ such that $(\sigma_1, \dots, \sigma_n) \in S$. It follows that

$$\mathcal{L}\left([S]^{\prec} \cap I((\sigma_1, \dots, \sigma_n))\right) = \mathcal{L}\left(I((\sigma_1, \dots, \sigma_n))\right) = \prod_{k=1}^{n} \frac{1}{\#\mathsf{Encf}_k}.$$

However, this contradicts (6) with $m = n$. Hence we have $\{\sigma_n\}_{n\in\mathbb{N}^+} \notin [S]^{\prec}$, and the proof is completed. □

4 The Discrete Logarithm Problem

In this section we consider the hardness of the discrete logarithm problem in the generic group model. We show that the generic group can be securely instantiated by a deterministic and computable one.

We first review the notion of generic algorithm [12]. A *generic algorithm* is a probabilistic oracle Turing machine $\mathcal{A}$ which behaves as follows: Let $n \in \mathbb{N}^+$, and let σ be an encoding function into n bitstrings and N a positive integer with $N \le 2^n$.

(i) $\mathcal{A}$ takes as input a list $\sigma(x_1), \dots, \sigma(x_k)$ with $x_1, \dots, x_k \in \mathbb{Z}_N$, as well as (the binary representations of) N and its prime factorization.

(ii) As $\mathcal{A}$ is executed, it is allowed to make calls to oracles which compute the functions *add*: $\sigma(\mathbb{Z}_N) \times \sigma(\mathbb{Z}_N) \to \sigma(\mathbb{Z}_N)$ and *inv*: $\sigma(\mathbb{Z}_N) \to \sigma(\mathbb{Z}_N)$ with $add(\sigma(x), \sigma(y)) = \sigma(x+y)$ and $inv(\sigma(x)) = \sigma(-x)$.

(iii) Eventually, $\mathcal{A}$ halts and outputs a finite binary string, denoted by $\mathcal{A}(N; \sigma; x_1, \dots, x_k)$.

Consider the following experiment for a polynomial-time generic algorithm $\mathcal{A}$, a parameter n, and a positive integer $N \le 2^n$:

The discrete logarithm experiment $\mathsf{DLog}_{\mathcal{A}}(n, N)$:

1. *Generate an encoding function σ into n bitstrings uniformly.*

2. *Generate $x \in \mathbb{Z}_N$ uniformly.*

3. *The output of the experiment is defined to be* 1 *if $\mathcal{A}(N; \sigma; 1, x) = x$ and* 0 *otherwise.*

Shoup [12] showed the following lower bound for the hardness of the discrete logarithm problem in the generic group model.

Theorem 4.1 (Shoup [12])**.** *There exists $C \in \mathbb{N}^+$ such that, for every generic algorithm $\mathcal{A}$, $n \in \mathbb{N}^+$, and N with $2 \le N \le 2^n - 1$,*

$$\Pr[\mathsf{DLog}_{\mathcal{A}}(n, N) = 1] \le \frac{Cm^2}{p},$$

where p is the largest prime divisor of N and m is the maximum number of the oracle queries among all the computation paths of $\mathcal{A}$. □

Next, consider the following experiment for a polynomial-time generic algorithm $\mathcal{A}$, a parameter n, and an encoding function σ into n bitstrings:

The discrete logarithm experiment $\mathsf{DLog}_{\mathcal{A}}(n, \sigma)$:

1. *Generate an n-bit prime p uniformly.*

2. *Generate $x \in \mathbb{Z}_p$ uniformly.*

3. *The output of the experiment is defined to be* 1 *if $\mathcal{A}(p; \sigma; 1, x) = x$ and* 0 *otherwise.*

The hardness of the discrete logarithm problem in the generic group model is then formulated as follows.

Definition 4.2. *We say that* the discrete logarithm problem is hard in the generic group model *if for all polynomial-time generic algorithms $\mathcal{A}$ and all $d \in \mathbb{N}^+$ there exists $N \in \mathbb{N}^+$ such that, for all $n > N$,*

$$\frac{1}{\#\mathsf{Encf}_n} \sum_{\sigma \in \mathsf{Encf}_n} \Pr[\mathsf{DLog}_{\mathcal{A}}(n, \sigma) = 1] \le \frac{1}{n^d}.$$

□

In this paper we consider a stronger notion of the hardness of the discrete logarithm problem than that given by Definition 4.2 above. This stronger notion, called the *effective* hardness of the discrete logarithm problem, is defined as follows: We first choose a particular recursive enumeration $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \dots$ of all polynomial-time generic algorithms. It is easy to show that such an enumeration exists. In fact, the kth polynomial-time generic algorithm $\mathcal{A}_k$ can be chosen as a generic algorithm obtained by executing the kth generic algorithm $\mathcal{M}_k$ in at most $n^k + k$ steps, where n is the length of the input of $\mathcal{M}_k$. We use this specific enumeration as the standard one throughout the rest of this paper.

Definition 4.3. *We say that* the discrete logarithm problem is effectively hard in the generic group model *if there exists a computable function $f\colon \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$ such that, for all $i, d, n \in \mathbb{N}^+$, if $n \ge f(i, d)$ then*

$$\frac{1}{\#\mathsf{Encf}_n} \sum_{\sigma \in \mathsf{Encf}_n} \Pr[\mathsf{DLog}_{\mathcal{A}_i}(n, \sigma) = 1] \le \frac{1}{n^d}.$$

□

Theorem 4.4. *The discrete logarithm problem is effectively hard in the generic group model.* □

In order to prove Theorem 4.4, we need the following lemma.[2]

Lemma 4.5. *Let $d \geq 4$. Then $2^n \geq n^d$ for all $n \geq d^2$.*

Proof. We first show that

$$2^d \geq d^2 \tag{10}$$

by induction. Obviously, $2^k \geq k^2$ holds for $k = 4$. For an arbitrary $k \geq 4$, assume that $2^k \geq k^2$ holds. Then $2^{k+1} \geq 2k^2 \geq (k+1)^2$, where the second inequality follows from the inequality $\sqrt{2}x \geq x+1$ for all $x \geq \sqrt{2}+1$. Thus (10) holds.

Now, we show that

$$2^n \geq n^d \tag{11}$$

holds for all $n \geq d^2$ by induction. First, it follows from (10) that $2^{d^2} \geq (d^2)^d$, which implies that (11) holds for $n = d^2$. For an arbitrary $k \geq d^2$, assume that (11) holds for $n = k$. We note that $2^{1/d} - 1 \geq \ln 2/d > 1/(2d) > 1/d^2$, where the first inequality follows from the mean-value theorem. Then, since $k \geq d^2$, we see that $2^{(k+1)/d} \geq 2^{1/d}k \geq k + k/d^2 \geq k+1$. This implies that (11) holds for $n = k+1$. Thus, (11) holds for all $n \geq d^2$. □

Then the proof of Theorem 4.4 is given as follows.

Proof of Theorem 4.4. Let $k \in \mathbb{N}^+$, and consider the kth generic algorithm $\mathcal{A}_k$. Since the number of oracle queries along any computation path of $\mathcal{A}_k$ is bounded to the above by $n^k + k$, it follows from theorem 4.1 that there exists $C \in \mathbb{N}^+$ such that, for every $n \in \mathbb{N}^+$ and n-bit prime p,

$$\Pr[\mathsf{DLog}_{\mathcal{A}_k}(n,p) = 1] \leq \frac{C(n^k+k)^2}{p} \leq \frac{C(n^k+k)^2}{2^{n-1}}.$$

Therefore, for every $n \geq \max\{k, 2C\}$,

$$\frac{1}{\#\mathsf{Encf}_n} \sum_{\sigma \in \mathsf{Encf}_n} \Pr[\mathsf{DLog}_{\mathcal{A}_k}(n,\sigma) = 1] \leq \frac{n^{2k+1}}{2^n}. \tag{12}$$

Note by Lemma 4.5 that, for each $d \in \mathbb{N}^+$,

$$\frac{n^{2k+1}}{2^n} \leq \frac{1}{n^d} \tag{13}$$

for every $n \geq (2k+d+1)^2$.

Thus we define a function $f\colon \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$ by $f(k,d) = \max\{(2k+d+1)^2, 2C\}$. Then f is computable, and it follows from (12) and (13) that, for all $k, d, n \in \mathbb{N}^+$, if $n \geq f(k,d)$ then

$$\frac{1}{\#\mathsf{Encf}_n} \sum_{\sigma \in \mathsf{Encf}_n} \Pr[\mathsf{DLog}_{\mathcal{A}_k}(n,\sigma) = 1] \leq \frac{1}{n^d}.$$

This completes the proof. □

[2] In order to prove Theorem 4.4, it is suffice to use the inequality $2^n \geq n^d$ which holds for all $n \geq ((d+1)/\ln 2)^{d+1}$ and not for all $n \geq d^2$ as in Lemma 4.5. The former follows immediately from the inequality $e^x \geq x$ which holds for all $x \in \mathbb{R}$. However, we prefer a more "insightful" polynomial lower bound $n \geq d^2$ than the super-exponential lower bound $n \geq ((d+1)/\ln 2)^{d+1}$. See Section 6 for further remarks.

The hardness of the discrete logarithm problem in the generic group model given by Definition 4.2 follows immediately from Theorem 4.4.

Corollary 4.6. *The discrete logarithm problem is hard in the generic group model.* □

We are interested in the instantiation of the generic group in the discrete logarithm problem. Thus, it is convenient to define the hardness of the discrete logarithm problem relative to a specific family of encoding functions.

Definition 4.7. *Let $\{\sigma_n\}_{n \in \mathbb{N}^+}$ be a family of encoding functions. We say that* the discrete logarithm problem is hard relative to $\{\sigma_n\}_{n \in \mathbb{N}^+}$ *if for all polynomial-time generic algorithms $\mathcal{A}$ and all $d \in \mathbb{N}^+$ there exists $N \in \mathbb{N}^+$ such that, for all $n > N$,*

$$\Pr[\mathsf{DLog}_{\mathcal{A}}(n,\sigma) = 1] \leq \frac{1}{n^d}.$$

□

The corresponding effective hardness is defined as follows.

Definition 4.8. *Let $\{\sigma_n\}_{n \in \mathbb{N}^+}$ be a family of encoding functions. We say that* the discrete logarithm problem is effectively hard relative to $\{\sigma_n\}_{n \in \mathbb{N}^+}$ *if there exists a computable function $f\colon \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$ such that, for all $i, d, n \in \mathbb{N}^+$, if $n \geq f(i,d)$ then*

$$\Pr[\mathsf{DLog}_{\mathcal{A}_i}(n,\sigma) = 1] \leq \frac{1}{n^d}.$$

□

Theorem 4.9 (main result I). *There exists a computable family of encoding functions relative to which the discrete logarithm problem is effectively hard.* □

While the proof of Theorem 4.9 is mainly based on Theorem 3.2, we also need Lemmas 4.10 and 4.11 below.

Lemma 4.10. *Let $f_1, \dots, f_N$ be reals. Suppose that $\frac{1}{N}\sum_{i=1}^N f_i \leq \varepsilon$. Then, for every $\alpha > 0$, the number of i for which $\alpha\varepsilon < f_i$ is less than N/α.*

Proof. We prove the contraposition of Lemma 4.10. Assume that the number of i for which $\alpha\varepsilon < f_i$ is at least N/α. Then $\sum_{i=1}^N f_i > \alpha\varepsilon N/\alpha = \varepsilon N$ and therefore $\frac{1}{N}\sum_{i=1}^N f_i > \varepsilon$. □

Lemma 4.11. *Let $d \geq 2$. Then $\sum_{k=n}^\infty 1/k^d \leq 2/n$ for every $n \in \mathbb{N}^+$.*

Proof. In the case of $n \geq 2$, we have

$$\begin{aligned}\sum_{k=n}^\infty \frac{1}{k^d} &\leq \sum_{k=n}^\infty \int_{k-1}^k \frac{1}{k^d} = \int_{n-1}^\infty \frac{1}{x^d}dx \\ &= \frac{1}{(d-1)(n-1)^{d-1}} \\ &\leq \frac{1}{n-1} \leq \frac{1}{n-n/2} = \frac{2}{n}.\end{aligned} \tag{14}$$

On the other hand, in the case of $n = 1$, using (14) we have

$$\sum_{k=n}^{\infty} \frac{1}{k^d} = 1 + \sum_{k=2}^{\infty} \frac{1}{k^d} \leq 1 + \frac{2}{2} = 2 = \frac{2}{n}.$$

Thus $\sum_{k=n}^{\infty} 1/k^d \leq 2/n$ holds in any case. □

The proof of Theorem 4.9 is then given as follows.

Proof of Theorem 4.9. First, by Theorem 4.4 there exists a computable function $f\colon \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$ such that, for all $i, d, n \in \mathbb{N}^+$, if $n \geq f(i, d)$ then

$$\frac{1}{\#\mathsf{Encf}_n} \sum_{\sigma \in \mathsf{Encf}_n} \Pr[\mathsf{DLog}_{\mathcal{A}_i}(n, \sigma) = 1] \leq \frac{1}{n^d}.$$

It follows from Lemma 4.10 that, for all $i, d, n \in \mathbb{N}^+$, if $n \geq f(i, 2d)$ then

$$\begin{aligned} &\#\left\{\sigma \in \mathsf{Encf}_n \;\middle|\; \Pr[\mathsf{DLog}_{\mathcal{A}_i}(n, \sigma) = 1] > \frac{1}{n^d}\right\} \\ &< \frac{\#\mathsf{Encf}_n}{n^d}. \end{aligned} \tag{15}$$

In order to apply the method of algorithmic randomness, i.e., Theorem 3.2, for each $i, d, n \in \mathbb{N}^+$ we define a subset $[C_{i,d,n}]^{\prec}$ of Encf^{∞} as the set of all families $\{\sigma_n\}_{n \in \mathbb{N}^+}$ of encoding functions such that

$$\Pr[\mathsf{DLog}_{\mathcal{A}_i}(n, \sigma_n) = 1] > \frac{1}{n^d}. \tag{16}$$

Namely, we define a subset $C_{i,d,n}$ of Encf^* as the set of all finite families $(\sigma_1, \dots, \sigma_n)$ of encoding functions where only σ_n is required to satisfy the inequality (16). Since $C_{i,d,n}$ is a prefix-free set for every $i, d, n \in \mathbb{N}^+$, it follows from (i) of Proposition 3.1 and (15) that, for each $i, d, n \in \mathbb{N}^+$, if $n \geq f(i, 2d)$ then

$$\mathcal{L}\left([C_{i,d,n}]^{\prec}\right) = \sum_{s \in C_{i,d,n}} |I(s)| < \frac{1}{n^d}. \tag{17}$$

We choose a particular computable bijection $\varphi\colon \mathbb{N}^+ \to \{\, (i, d) \mid i \in \mathbb{N}^+ \;\&\; d \geq 2 \,\}$, and define $(\varphi_1(m), \varphi_2(m)) = \varphi(m)$. We then define a computable function $g\colon \mathbb{N}^+ \to \mathbb{N}^+$ by $g(m) = \{f(\varphi_1(m), 2\varphi_2(m)) + 1\}^{m+1}$. For each $m \in \mathbb{N}^+$, we define a subset C_m of $\{0,1\}^*$ by

$$C_m = \bigcup_{n=g(m)}^{\infty} C_{\varphi_1(m), \varphi_2(m), n}. \tag{18}$$

It follows from (iii) of Proposition 3.1, (17), and Lemma 4.11 that, for each $m \in \mathbb{N}^+$,

$$\begin{aligned} \mathcal{L}\left([C_m]^{\prec}\right) &\leq \sum_{n=g(m)}^{\infty} \mathcal{L}\left(\left[C_{\varphi_1(m), \varphi_2(m), n}\right]^{\prec}\right) \\ &< \sum_{n=g(m)}^{\infty} \frac{1}{n^{\varphi_2(m)}} \leq \frac{2}{g(m)} \leq \frac{1}{2^m}. \end{aligned} \tag{19}$$

We then define C by

$$C = \bigcup_{m=1}^{\infty} C_m. \tag{20}$$

Therefore, using (iii) of Proposition 3.1,

$$\mathcal{L}\left([C]^{\prec}\right) \leq \sum_{m=1}^{\infty} \mathcal{L}\left([C_m]^{\prec}\right) < \sum_{m=1}^{\infty} \frac{1}{2^m} = 1. \tag{21}$$

Next we show that C is an r.e. subset of Encf^*. It is easy to see that, given i, d, and n, one can decide the finite subset $C_{i,d,n}$ of Encf^*, since the dyadic rational $\Pr[\mathsf{DLog}_{\mathcal{A}_i}(n, \sigma) = 1]$ is computable, given i, n, and an encoding function σ into n bitstrings. Thus, since φ and g are computable functions, it follows from (18) and (20) that C is an r.e. subset of Encf^*.

We then show that $\mathcal{L}\left([C]^{\prec}\right)$ is a computable real. For each $k \in \mathbb{N}$, we define a finite subset D_k of C by

$$D_k = \bigcup_{m=1}^{k} \bigcup_{n=g(m)}^{g(m)2^k - 1} C_{\varphi_1(m), \varphi_2(m), n}.$$

Given $k \in \mathbb{N}$, one can decides the finite set D_k, since φ and g are computable functions and moreover one can decide the finite set $C_{i,d,n}$, given i, d, and n. Therefore, given $k \in \mathbb{N}$, one can calculate the dyadic rational $\mathcal{L}\left([D_k]^{\prec}\right)$ based on (i) of Proposition 3.1. On the other hand, note that

$$C \setminus D_k \subset \left(\bigcup_{m=1}^{k} \bigcup_{n=g(m)2^k}^{\infty} C_{\varphi_1(m), \varphi_2(m), n}\right) \cup \bigcup_{m=k+1}^{\infty} C_m.$$

Thus, using (ii) and (iii) of Proposition 3.1, (17), Lemma 4.11, and (19) we see that, for each $k \in \mathbb{N}$,

$$\begin{aligned} &\mathcal{L}\left([C \setminus D_k]^{\prec}\right) \\ &\leq \sum_{m=1}^{k} \sum_{n=g(m)2^k}^{\infty} \mathcal{L}\left(\left[C_{\varphi_1(m), \varphi_2(m), n}\right]^{\prec}\right) + \sum_{m=k+1}^{\infty} \mathcal{L}\left([C_m]^{\prec}\right) \\ &< \sum_{m=1}^{k} \frac{2}{g(m)2^k} + \sum_{m=k+1}^{\infty} \frac{1}{2^m} \leq \sum_{m=1}^{k} \frac{1}{2^{m+k}} + \frac{1}{2^k} \\ &< \frac{1}{2^{k-1}}. \end{aligned}$$

Therefore, since $[C]^{\prec} = [D_{k+1}]^{\prec} \cup [C \setminus D_{k+1}]^{\prec}$, using (ii) and (iii) of Proposition 3.1 we have

$$\left|\mathcal{L}\left([C]^{\prec}\right) - \mathcal{L}\left([D_{k+1}]^{\prec}\right)\right| \leq \mathcal{L}\left([C \setminus D_{k+1}]^{\prec}\right) \leq 2^{-k}$$

for each $k \in \mathbb{N}$. Hence, $\mathcal{L}\left([C]^{\prec}\right)$ is a computable real.

Now, it follows from Theorem 3.2 that there exists a computable family $\{\sigma_n\}_{n \in \mathbb{N}^+}$ of encoding functions which is not in $[C]^{\prec}$. Let $i, d, n \in \mathbb{N}^+$ with $n \geq g(\varphi^{-1}(i, d+1))$. We then define $m = \varphi^{-1}(i, d+1)$, i.e., $\varphi(m) = (i, d+1)$. Since $\{\sigma_n\}_{n \in \mathbb{N}^+} \notin [C]^{\prec}$ and $n \geq g(m)$, it follows from (20) and (18) that $\{\sigma_n\}_{n \in \mathbb{N}^+} \notin \left[C_{\varphi_1(m), \varphi_2(m), n}\right]^{\prec} = [C_{i,d+1,n}]^{\prec}$. Therefore, we see that the family $\{\sigma_n\}_{n\in\mathbb{N}^+}$ of encoding functions satisfies that $\Pr[\mathsf{DLog}_{\mathcal{A}_i}(n,\sigma_n)=1] \le 1/n^{d+1} < 1/n^d$ for each $n \in \mathbb{N}^+$. Thus, since the mapping $\mathbb{N}^+ \times \mathbb{N}^+ \ni (i,d) \mapsto g(\varphi^{-1}(i,d+1))$ is a computable function, it follows from Definition 4.8 that the discrete logarithm problem is effectively hard relative to $\{\sigma_n\}_{n\in\mathbb{N}^+}$. □

Corollary 4.12. *There exists a computable family of encoding functions relative to which the discrete logarithm problem is hard.*

Proof. The result follows immediately from Theorem 4.9. □

5 The Diffie-Hellman Problem

In this section we consider the hardness of the computational Diffie-Hellman (CDH) problem in the generic group model. We can show the analogues of all the results in the previous section for the CDH problem, in place of the discrete logarithm problem. In this section, in particular we present the analogues of Theorem 4.9 and Corollary 4.12 for the CDH problem.

We first recall the analogue of Theorem 4.1 for the CDH problem. We thus consider the following experiment for a polynomial-time generic algorithm $\mathcal{A}$, a parameter n, and a positive integer $N \le 2^n$:

The computational Diffie-Hellman experiment $\mathsf{CDH}_{\mathcal{A}}(n,N)$:

1. *Generate an encoding function σ into n bitstrings uniformly.*
2. *Generate $x \in \mathbb{Z}_N$ uniformly.*
3. *Generate $y \in \mathbb{Z}_N$ uniformly.*
4. *The output of the experiment is defined to be 1 if $\mathcal{A}(N;\sigma;1,x,y)=\sigma(xy)$ and 0 otherwise.*

Shoup [12] showed the following lower bound for the hardness of the CDH problem in the generic group model, which is the analog of Theorem 4.1.

Theorem 5.1 (Shoup [12])**.** *There exists $C \in \mathbb{N}^+$ such that, for every generic algorithm $\mathcal{A}$, $n \in \mathbb{N}^+$, and N with $2 \le N \le 2^n - 1$,*

$$\Pr[\mathsf{CDH}_{\mathcal{A}}(n,N)=1] \le \frac{Cm^2}{p},$$

where p is the largest prime divisor of N and m is the maximum number of the oracle queries among all the computation paths of $\mathcal{A}$. □

Now, consider the following experiment for a polynomial-time generic algorithm $\mathcal{A}$, a parameter n, and an encoding function σ into n bitstrings:

The computational Diffie-Hellman experiment $\mathsf{CDH}_{\mathcal{A}}(n,\sigma)$:

1. *Generate an n-bit prime p uniformly.*
2. *Generate $x \in \mathbb{Z}_p$ uniformly.*
3. *Generate $y \in \mathbb{Z}_p$ uniformly.*
4. *The output of the experiment is defined to be 1 if $\mathcal{A}(p;\sigma;1,x,y)=\sigma(xy)$ and 0 otherwise.*

Then the hardness of the CDH problem relative to a specific family of encoding functions is defined as follows.

Definition 5.2. *Let $\{\sigma_n\}_{n\in\mathbb{N}^+}$ be a family of encoding functions. We say that* the CDH problem is hard relative to $\{\sigma_n\}_{n\in\mathbb{N}^+}$ *if for all polynomial-time generic algorithms $\mathcal{A}$ and all $d \in \mathbb{N}^+$ there exists $N \in \mathbb{N}^+$ such that, for all $n > N$,*

$$\Pr[\mathsf{CDH}_{\mathcal{A}}(n,\sigma)=1] \le \frac{1}{n^d}.$$

□

The corresponding effective hardness is defined as follows.

Definition 5.3. *Let $\{\sigma_n\}_{n\in\mathbb{N}^+}$ be a family of encoding functions. We say that* the CDH problem is effectively hard relative to $\{\sigma_n\}_{n\in\mathbb{N}^+}$ *if there exists a computable function $f\colon \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$ such that, for all $i,d,n \in \mathbb{N}^+$, if $n \ge f(i,d)$ then*

$$\Pr[\mathsf{CDH}_{\mathcal{A}_i}(n,\sigma)=1] \le \frac{1}{n^d}.$$

□

Based on Theorem 5.1, we can show the following analogue of Theorem 4.9 in the same manner as the proof of Theorem 4.9.

Theorem 5.4 (main result II)**.** *There exists a computable family of encoding functions relative to which the CDH problem is effectively hard.* □

Corollary 5.5. *There exists a computable family of encoding functions relative to which the CDH problem is hard.* □

6 Concluding Remarks

In this section, we first see that the *effective* hardness notions introduced in the previous two sections are a natural alternative to the *conventional* hardness notions in modern cryptography. In this section, we consider the effective hardness notions for the discrete logarithm problem in particular. The same remarks apply to the effective hardness notions for the CDH problem.

On the one hand, in Definitions 4.2 and 4.7 for the conventional hardness of the discrete logarithm problem, the number N is only required to exist, depending on a polynomial-time generic algorithm $\mathcal{A}$ and a number d, that is, the success probability of the attack by $\mathcal{A}$ against the discrete logarithm problem on a security parameter n is required to be at most than $1/n^d$ for all sufficiently large n, where the lower bound of such n is not required to be computable from $\mathcal{A}$ and d. On the other hand, in Definitions 4.3 and 4.8 for the effective hardness of the discrete logarithm problem, it is required that the lower bound N of such n can be computed from the code of $\mathcal{A}$ and d.

In modern cryptography based on computational security, it is important to choose the security parameter n of a cryptographic scheme as small as possible to the extent that the security requirements are satisfied, in order to make the efficiency of the scheme as high as possible. For that purpose, it is desirable to be able to calculate a concrete value of N, given the code of $\mathcal{A}$ and d, since N gives a lower bound of the security parameter for which the security requirements specified by $\mathcal{A}$ and d are satisfied. This results in the notion of effective hardness. Thus the notions of effective hardness are considered to be more desirable than the notions of conventional hardness in modern cryptography.

In the above we have demonstrated the validity of the effective hardness notions in modern cryptography. However, it would seem more natural to require that the functions $f\colon \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$ in Definitions 4.3 and 4.8 are *polynomial-time computable* rather than simply computable. We call this type of effective hardness *polynomial-time effective hardness*. In Theorem 4.4 we have shown that the discrete logarithm problem is effectively hard in the generic group model. In the proof of Theorem 4.4, the function f has the form $f(i,d) = \max\{(2i+d+1)^2, 2C\}$. This is a polynomial-time computable function. Thus, the proof of Theorem 4.4 actually shows that the discrete logarithm problem is polynomial-time effectively hard in the generic group model.

Conjecture 1 below is a polynomial-time effective version of Theorem 4.9, which states that the discrete logarithm problem is hard in the standard model for some finite cyclic group. In the future, it would be challenging to prove Conjecture 1 (or its appropriate modification) with identifying an appropriate computational assumption COMP which is weaker than the hardness of the discrete logarithm problem itself.

Conjecture 1. *Under the assumption* COMP*, there exists a* polynomial-time computable *family of encoding functions (or a* polynomial-time computable *family of families of encoding functions) relative to which the discrete logarithm problem is* polynomial-time effectively *hard.* □

Acknowledgments

This work was supported by the Ministry of Economy, Trade and Industry of Japan. The first author was also supported by JSPS KAKENHI Grant Numbers 23340020, 23650001, 24540142.

References

[1] D. Aggarwal and U. Maurer, Breaking RSA Generically Is Equivalent to Factoring. *Proc.* EUROCRYPT 2009, Lecture Notes in Computer Science, Springer-Verlag, Vol.5479, pp.36–53, 2009.

[2] M. Bellare and P. Rogaway, Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. *Proc.* 1st ACM Conference on Computer and Communications Security, ACM, pp.62–73, 1993.

[3] A. W. Dent, Adapting the Weaknesses of the Random Oracle Model to the Generic Group Model. *Proc.* ASIACRYPT 2002, Lecture Notes in Computer Science, Springer-Verlag, Vol.2501, pp.100–109, 2002.

[4] R. G. Downey and D. R. Hirschfeldt, *Algorithmic Randomness and Complexity.* Springer-Verlag, New York, 2010.

[5] O. Goldreich, *Foundations of Cryptography: Volume 1 – Basic Tools.* Cambridge University Press, New York, 2001.

[6] O. Goldreich, *Foundations of Cryptography: Volume 2 – Basic Applications.* Cambridge University Press, New York, 2004.

[7] J. Katz and Y. Lindell, *Introduction to Modern Cryptography.* Chapman & Hall/CRC Press, 2007.

[8] U. Maurer and S. Wolf, Lower Bounds on Generic Algorithms in Groups. *Proc.* EUROCRYPT'98, Lecture Notes in Computer Science, Springer-Verlag, Vol.1403, pp.72–84, 1998.

[9] U. Maurer, Abstract Models of Computation in Cryptography. *Proc.* Cryptography and Coding 2005, Lecture Notes in Computer Science, Springer-Verlag, Vol.3796, pp.1–12, 2005.

[10] D. Moriyama, R. Nishimaki and T. Okamoto, *Theory of Public-Key Cryptography.* Industrial and Applied Mathematics Series Vol.2. JSIAM, Kyoritsu Shuppan Co., Ltd., Tokyo, 2011. In Japanese.

[11] A. Nies, *Computability and Randomness.* Oxford University Press, Inc., New York, 2009.

[12] V. Shoup, Lower Bounds for Discrete Logarithms and Related Problems. *Proc.* EUROCRYPT'97, Lecture Notes in Computer Science, Springer-Verlag, Vol.1233, pp.256–266, 1997.

[13] K. Tadaki and N. Doi, Instantiating the Random Oracle Using a Random Real. *Proc.* 29th Symposium on Cryptography and Information Security (SCIS2012), 2A3-4, January 30-February 2, 2012, Kanazawa, Japan.

[14] K. Tadaki and N. Doi, A Secure Instantiation of the Random Oracle by a Computable Function. *Proc.* 35th Symposium on Information Theory and its Applications (SITA2012), 3.4.1, pp.212–217, December 11-14, 2012, Beppu, Oita, Japan.