On the Security of Pseudorandomized Information-Theoretically Secure Schemes*

Koji Nuida Goichiro Hanaoka

Research Center for Information Security (RCIS), National Institute of Advanced Industrial Science and Technology (AIST), Akihabara-Daibiru Room 1003, 1-18-13 Sotokanda, Chiyoda-ku, Tokyo 101-0021, Japan
{k.nuida,hanaoka-goichiro}@aist.go.jp

Abstract

In this article, we discuss a naive method of randomness reduction for cryptographic schemes, which replaces the required perfect randomness with output distribution of a computationally secure pseudorandom generator (PRG). We propose novel ideas and techniques for evaluating the indistinguishability between the random and pseudorandom cases, even against an adversary with computationally unbounded attack algorithm. Hence the PRG-based randomness reduction can be effective even for information-theoretically secure cryptographic schemes, especially when the amount of information received by the adversary is small. In comparison to a preceding result of Dubrov and Ishai (STOC 2006), our result removes the requirement of generalized notion of "nb-PRGs" and is effective for more general kinds of protocols. We give some numerical examples to show the effectiveness of our result in practical situations, and we also propose a further idea for improving the effect of the PRG-based randomness reduction.

Key words: Randomness reduction, information-theoretic security, pseudorandom generator

1 Introduction

1.1 Backgrounds

Randomness is an essential resource for cryptography, and is one of the most important ingredients of applications in information theory, e.g., efficient computation by probabilistic algorithms. Most of the existing schemes are designed by basing on an (implicit) assumption that perfect random sources are freely available. However, in practice such perfect (or even approximately perfect) sources are either not available, or available but cost-consuming. Hence it is necessary to relax the requirements for quality and amount of randomness used in the schemes. Some preceding works have shown that, although imperfect random sources (entropy sources) can be used for non-cryptographic schemes and some kinds of cryptographic schemes [10, 12, 18, 24, 28, 29], it is essentially impossible for many cryptographic purposes to replace the perfect random sources with imperfect ones without diminishing quality of the scheme [6, 12, 19]. Hence the possibility of relaxing the requirements for quality of randomness is limited, therefore it is significant, especially for cryptographic purposes, to relax the requirements for the amount of randomness, i.e., to perform randomness reduction or derandomization.

There have been proposed a lot of randomness reduction techniques, such as [1, 3, 8, 15, 23], which are *information-theoretically indistinguishable*, i.e., the result of the randomness-reduced protocol is statistically indistinguishable from that of the original protocol. However, those techniques are scheme-dependent, and

*A part of this work was supported by 2007 Research Grants of the Science and Technology Foundation of Japan (JSTF).

the negative results mentioned in the previous paragraph suggest that information-theoretically indistinguishable *universal* randomness reduction techniques using a single (imperfect) random source are unlikely to exist. (Here the condition of using one source is crucial in some sense, since two independent weak random sources can be used to extract almost perfect random bits [10, 25].) On the other hand, there exists a well-known *computationally* indistinguishable universal randomness reduction technique, which is to replace the required randomness with outputs of (computationally) secure pseudorandom generators (PRGs).

For an intermediate situation, Dubrov and Ishai introduced in their work [11] a generalization of PRGs, called *pseudorandom generators that fool non-boolean distinguishers* (*nb-PRGs*, in short). They gave a concrete example of nb-PRGs under a certain computational assumption. By the definition of nb-PRGs, for any efficient algorithm with sufficiently small output set, the algorithm with uniform input distribution and the one with input distribution replaced with the output of an nb-PRG have statistically indistinguishable output distributions. Hence *information-theoretically indistinguishable* randomness reduction for such a randomized algorithm is possible by using an nb-PRG under the corresponding *computational* assumption. More precisely, the statistical distance between the output distributions in random and pseudorandom cases is bounded in terms of hardness of the underlying computational problem. They also applied nb-PRGs to information-theoretically indistinguishable randomness reduction of private multi-party computation protocol (see [11, Section 6.2]). Hence their technique is also effective for some kinds of cryptographic protocols.

However, there are some drawbacks of the above-mentioned randomness reduction technique using nb-PRGs for cryptographic protocols, as follows. First, the security evaluation method of Dubrov and Ishai in [11] depends on the property of the considered protocol that calculation of a secret protected by the protocol does *not* use the randomness to be replaced with nb-PRGs, and this property fails for many cryptographic protocols. Secondly, the construction of nb-PRGs presented in [11] is based on a certain non-standard computational assumption, and no nb-PRGs based on standard assumptions (e.g., hardness assumptions of decisional or computational Diffie-Hellman problem) have been obtained so far. Moreover, in contrast to the notion of usual PRGs that is well-known even for non-experts of cryptography, the notion of nb-PRGs seems not yet popular even for experts of cryptography. Hence it is worthy to investigate a similar information-theoretically indistinguishable randomness reduction technique based on usual (secure) PRGs.

1.2 Our Contributions

In this article, we reveal that information-theoretically indistinguishable randomness reduction is possible by using secure PRGs in a naive manner. More precisely, we consider the situation of randomness reduction that (a part of) the required perfect randomness for a cryptographic protocol is replaced with output of a PRG whose indistinguishability is based on an underlying hard computational problem. Then our result implies that the difference of success probabilities of any attack by an adversary between the random and pseudorandom cases is bounded by a function of both of hardness of the underlying problem for the PRG and, roughly speaking, the amount of information used for the attack by the adversary.

A remarkable characteristic of this result is that *the bound is independent of any property, including computational complexity, of the attack algorithm.* This means that our result can be applied even to cryptographic schemes with information-theoretic security. Moreover, it is also noteworthy that, intuitively speaking, the computational environment in which the hardness of the underlying problem for the PRG is evaluated can be chosen *independently* of the adversary's computational environment. A typical example of the property is that the randomness reduction can be indistinguishable for quantum adversary even when the underlying problem for the PRG is classically hard but quantumly easy (e.g., Integer Factoring and Discrete Log).

In comparison to the preceding result of Dubrov and Ishai [11] mentioned in Section 1.1, our result has the following advantages. First, our result uses PRGs in a usual sense instead of the generalized and less popular notion of nb-PRGs. (In fact, we can prove that secure PRGs with sufficiently long seed lengths are also nb-PRGs, as mentioned in [11, Observation 3.1]; see Proposition 2.1 in Section 2.) As a result, our randomness reduction technique can be based on any standard security assumption (such as classical hardness of Integer Factoring or Discrete Log) instead of non-standard assumption used in [11] for constructing concrete nb-PRGs. Secondly, our result is applicable to more general kinds of cryptographic schemes than [11], since it is allowed that calculation of a secret protected by the protocol *does* use the randomness to be replaced with PRGs (it may *not* use the randomness in the case of [11]; see the discussion in Section 3.3). We notice that our result requires a condition that, intuitively speaking, the amount of information used by an adversary for the attack is sufficiently small (such a condition was also required in the case of [11]). However, the numerical example given later shows that our result is still applicable to some existing schemes; sufficiently indistinguishable randomness reduction is possible by using a PRG whose seed size is significantly shorter than the size of the original required randomness.

Here we present a special case of our result as an example. We consider a function $f : X \to Y$ whose output value $f(x)$ is to be protected. An adversary tries to make a guess about $f(x)$ by using leakage information $\ell \in L$ on the input x of f and an attack algorithm $\mathsf{At} : L \to Z$, where L and Z denote the sets of possible leakage information and of possible guesses, respectively. Let $\mathsf{Lk} : X \to L$ denote the leakage function. Moreover, we introduce an auxiliary algorithm $\mathsf{Ev} : Y \times Z \to \{0,1\}$ that evaluates whether the adversary's guess $z = \mathsf{At}(\mathsf{Lk}(x))$ about $y = f(x)$ is "correct" ($\mathsf{Ev}(y,z) = 1$) or not ($\mathsf{Ev}(y,z) = 0$). Then the success probability of the adversary's guess is the probability that $\mathsf{A}(x) = 1$, where we define $\mathsf{A} : X \to \{0,1\}$ by $\mathsf{A}(x) = \mathsf{Ev}(f(x), \mathsf{At}(\mathsf{Lk}(x)))$. This process is represented by the left-hand side of Fig. 1. Here we assume that the algorithms f, Lk and Ev are all efficient, while we do not any assumption on the computational complexity of At (denoted by a circled arrow in the picture).

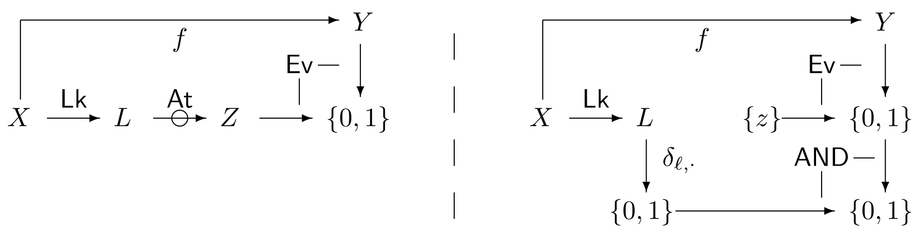

Figure 1: Flowchart for input leakage-resilient functions against unbounded attack algorithms (left-hand side) and the corresponding auxiliary flowchart (right-hand side)

Now suppose that the function f is secure in the sense that, when the input $x \in X$ of f is chosen uniformly at random, the adversary's success probability $\mathsf{succ}_{\mathrm{rnd}}$ is bounded by a sufficiently small value. We would like to bound, by a sufficiently small value, the difference between $\mathsf{succ}_{\mathrm{rnd}}$ and the adversary's success probability $\mathsf{succ}_{\mathrm{prnd}}$ in the case that the input of f is given by a PRG G (with output set X). If the adversary receives no information (i.e., $|L| = 1$), then even the computationally unbounded attack algorithm At can nothing better than the perfectly random case. On the other hand, if the adversary receives much information (i.e., $|L|$ is too large), then the adversary would be able to break the pseudorandomness of G and to make a much better guess than the perfectly random case. Now our result provides a quantitative argument for the separating point of those two extreme situations. Given any elements $\ell \in L$ and $z \in Z$, we introduce an auxiliary algorithm $\widetilde{\mathsf{A}} : X \to \{0,1\}$ such that

$$\widetilde{\mathsf{A}}(x) = \begin{cases} 1 & \text{if } \mathsf{Lk}(x) = \ell \text{ and } \mathsf{Ev}(f(x), z) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

(see also the lower half of Fig. 1). Our result tells us the way of deriving appropriate algorithms such as $\widetilde{\mathsf{A}}$ from the current situation (see also Section 3.3). Note that $\widetilde{\mathsf{A}}$ is composed of efficient algorithms only, not involving the attack algorithm At. In this situation, if the complexity of $\widetilde{\mathsf{A}}$ is bounded by a constant T and the advantage for T-time algorithms of distinguishing the outputs of G from perfectly random outputs is bounded by ε, then our result implies that

$$|\mathsf{succ}_{\mathrm{prnd}} - \mathsf{succ}_{\mathrm{rnd}}| \le |L| \cdot \varepsilon \ ,$$

therefore we have a lower bound of the allowable amount of information received by the adversary.

Our evaluation technique is effective especially in the situations that the information received by the adversary is sufficiently small. A typical case is that a small piece of the randomness, which is to be replaced with pseudorandomness, is distributed to each of a large number of players for a protocol, including a limited number of adversaries. Such applications include parallel computation over honest-but-curious modules, secret sharing [4, 26], broadcast encryption [14], traitor tracing [2, 9, 16], and collusion-secure fingerprint codes [5, 27]. In later section, we present a numerical example of applications of our result to randomness reduction of information-theoretically secure existing schemes, by using a collusion-secure code in [20] and a secure PRG in [13] based on the DDH assumption. For the parameter choices in the example, we see that the seed lengths of the PRG which are approximately 75% to 0.0002% of the original perfectly random bits suffice to bound the differences between random and pseudorandom cases by sufficiently small values. This shows that our result is indeed effective for existing cryptographic schemes.

Moreover, the observation for the case of collusion-secure codes provides a novel technique to improve the effect of randomness reduction. The technique is to divide the randomness that is the target of the randomness reduction into several pieces, in such a way that only a smaller component of the information received by the adversary depends on each piece of the randomness. Then we replace each piece of the randomness with output of an *independent* PRG, and we evaluate the total difference between random and pseudorandom cases by using "hybrid argument". By applying the technique to the above-mentioned example of collusion-secure codes, we see that in the setting, the total seed length of the independent PRGs is reduced to approximately 29 times as short as the case of the plain randomness reduction. This shows that our proposed technique is also effective.

1.3 Organization of the Article

This article is organized as follows. Section 2 summarizes some definitions, notations and terminology used throughout this article, and mentions some properties. In Section 3.1, we introduce a certain kind of mathematical expressions of cryptographic procedures and some relevant notions, which play a key role in our main theorem. Section 3.2 presents the main theorem of this article and its proof. Section 3.3 gives an example of the main theorem to help understanding, and Section 3.4 collects some remarks on our result. In Section 4, we summarize constructions and relevant properties of some existing cryptographic objects, namely secure PRGs in [13] (Section 4.1) and collusion-secure codes in [20] (Section 4.2), to give numerical examples of applications of our main theorem. Section 5 presents the numerical examples; first, in Section 5.1 we propose a technique to improve the effect of randomness reduction as mentioned in the final paragraph of Section 1.2. Then in Section 5.2 we give the numerical examples based on the results in previous sections. Finally, Section 6 concludes this article.

2 Definitions and Notations

In this section, we summarize some definitions and notations used throughout this article. In this article, any algorithm is probabilistic unless otherwise specified. Let U_X denote the uniform probability distribution over a finite set X. We often identify a probability distribution with the corresponding random variable. We write $x \leftarrow P$ to signify that x is a particular value of a random variable P. Let $\mathbb{R}_{\geq 0}$ denote the set of non-negative real numbers. Put $\Sigma = \{0, 1\}$. For any element x of a set X, let $\delta_{x,\cdot}$ denote an algorithm that takes an input y from X and outputs 1 if $y = x$ and 0 if $y \neq x$ (i.e., that computes Kronecker delta). We identify the set $\mathbb{Z}_q$ of integers modulo q naturally with $\{1, 2, \dots, q\}$. Moreover, we identify the set Σ^h of h-bit sequences with $\{0, 1, \dots, 2^h - 1\}$ via binary expressions of integers. Let $|q|_2$ denote the bit length of an integer q.

To explain the results of this work, here we clarify some terminology used in this article:

Definition 2.1. A *complexity measure* is a function $\mathsf{comp} : \mathsf{Alg} \to \mathbb{R}_{\geq 0}$ on a set Alg of algorithms that assigns to each algorithm $\mathsf{A} \in \mathsf{Alg}$ its *complexity* $\mathsf{comp}(\mathsf{A}) \geq 0$.

Definition 2.2. Among computational assumptions for security proofs, a *computational power assumption* means an assumption of the following type: "the adversary cannot solve a specified computational problem by a practical computational cost (e.g., computing time)". On the other hand, a *computational hardness assumption* means an assumption of the following type: "the complexity of an algorithm (in an explicitly or implicitly specified underlying set of algorithms) that solves a specified computational problem is lower bounded by a significantly large value".

In Definition 2.1, the "complexity" may take various meanings depending on the context, such as time complexity on a fixed Turing machine, circuit complexity with a fixed set of fundamental gates, average- or worst-case running time on a fixed real computer, or space complexity. An important point is that a complexity measure depends not only on the choice of the set Alg (e.g., the set of classical or quantum algorithms), but also on the choice of the computational environment in which each algorithm is executed. For example, when the computer is replaced with a new one which is twice as fast as the original, the complexity measure is also replaced with the one whose value is twice as small as the original. Therefore, any speedup of the adversary's computation induced just by an improvement of his computational environment (e.g., the number of computers for parallel computing), not by an algorithmic improvement, can be interpreted as a change of the complexity measure.

For Definition 2.2, we notice that most of the existing cryptosystems that provide computational security are in fact based on computational power assumptions (in the above sense), e.g., assumption on infeasibility for the adversary of factoring 1024-bit RSA composites. On the other hand, our result in this article (Theorem 3.1) is based on a computational hardness assumption.

Let $\mathsf{G} : S_{\mathsf{G}} \to O_{\mathsf{G}}$ be a PRG with seed set S_{G} and output set O_{G}. In this article, we deal with exact (concrete) security rather than asymptotic security, therefore G is a single algorithm rather than a sequence of algorithms with various seed lengths. The following notion of indistinguishability for PRGs is a natural translation of the conventional notion to the case of exact security and has appeared in the literature (except slight modification mentioned later), e.g., [13, Definition 1]:

Definition 2.3. An algorithm $\mathsf{D} : O_{\mathsf{G}} \to \{0,1\}$ is called a *distinguisher* for a PRG G. For any distinguisher D for G, its *advantage* $\mathsf{adv}_{\mathsf{G}}(\mathsf{D})$ is defined by

$$\mathsf{adv}_{\mathsf{G}}(\mathsf{D}) = |Pr[\mathsf{D}(\mathsf{G}(U_{S_{\mathsf{G}}})) = 1] - Pr[\mathsf{D}(U_{O_{\mathsf{G}}}) = 1]\,| \ .$$

Definition 2.4. Let $\mathsf{comp} : \mathsf{Alg} \to \mathbb{R}_{\geq 0}$ be a complexity measure, $\mathcal{C} \subset \mathsf{Alg}$, and $R(t) \geq 0$ a non-decreasing function. A PRG G is called $R(t)$*-secure with respect to* $(\mathsf{comp}, \mathcal{C})$ if for any distinguisher D for G that belongs to $\mathcal{C}$, its advantage is bounded by

$$\mathsf{adv}_{\mathsf{G}}(\mathsf{D}) \leq R(\mathsf{comp}(\mathsf{D})) \ .$$

When $\mathcal{C} = \mathsf{Alg}$, we say simply that G is $R(t)$*-secure with respect to* comp.

The above-mentioned difference of Definition 2.4 from the one in the literature is that the choice of the distinguisher is restricted to a subclass $\mathcal{C}$ of algorithms. This aims at enabling us to perform a refined analysis, but for the sake of intuitive understanding of our argument, one may ignore the issue of the subclass $\mathcal{C}$ by setting $\mathcal{C} = \mathsf{Alg}$. An instance of $R(t)$-secure PRGs was given by Farashahi et al. [13] under DDH assumption, which is used in our numerical examples below, where the function $R(t)$ is estimated in terms of complexity of the best known classical algorithm to solve the DDH problem (see Section 4.1 for details). Note that there is a general tendency such that, when the basic structure of the PRG is not changed but the seed length is increased, the PRG will be more indistinguishable, implying that the value of the function $R(t)$ in Definition 2.4 will be smaller.

We also recall the definition of statistical distance:

Definition 2.5. For two probability distributions P_1, P_2 over the same finite set X, their *statistical distance* $\mathsf{SD}(P_1, P_2)$ is defined by

$$\begin{aligned}\mathsf{SD}(P_1, P_2) &= \frac{1}{2} \sum_{x \in X} |Pr[x \leftarrow P_1] - Pr[x \leftarrow P_2]\,| \\ &= \max_{E \subset X} \left(Pr[x \leftarrow P_1 : x \in E] - Pr[x \leftarrow P_2 : x \in E] \right) \ .\end{aligned}$$

Note that $\mathsf{SD}(f(P_1), f(P_2)) \le \mathsf{SD}(P_1, P_2)$ for any (probabilistic) function f. We also notice that the definition of statistical distance implies the following fact, which shows that any $R(t)$-secure PRG with respect to $(\mathsf{comp}, \mathcal{C})$ is also an nb-PRG with suitable parameters (cf. [11, Observation 3.1]):

Proposition 2.1. *Let $\mathsf{G} : S_{\mathsf{G}} \to O_{\mathsf{G}}$ be an $R(t)$-secure PRG with respect to $(\mathsf{comp}, \mathcal{C})$. Let $F : O_{\mathsf{G}} \to Y$ be an efficient algorithm. Assume that, for each $y \in Y$, the algorithm $\delta_{y,\cdot} \circ F : O_{\mathsf{G}} \to \{0,1\}$ satisfies that $\delta_{y,\cdot} \circ F \in \mathcal{C}$ and that $\mathsf{comp}(\delta_{y,\cdot} \circ F) \le T$ for a common constant $T > 0$. Then $\mathsf{SD}(F(U_{O_{\mathsf{G}}}), F(\mathsf{G}(U_{S_{\mathsf{G}}}))) \le (|Y|/2) \cdot R(T)$.*

Proof. We have

$$\mathsf{SD}(F(U_{O_{\mathsf{G}}}), F(\mathsf{G}(U_{S_{\mathsf{G}}}))) = \frac{1}{2} \sum_{y \in Y} |Pr[y \leftarrow F(U_{O_{\mathsf{G}}})] - Pr[y \leftarrow F(\mathsf{G}(U_{S_{\mathsf{G}}}))]|$$

$$= \frac{1}{2} \sum_{y \in Y} |Pr[(\delta_{y,\cdot} \circ F)(U_{O_{\mathsf{G}}}) = 1] - Pr[(\delta_{y,\cdot} \circ F)(\mathsf{G}(U_{S_{\mathsf{G}}})) = 1]| = \frac{1}{2} \sum_{y \in Y} \mathsf{adv}_{\mathsf{G}}(\delta_{y,\cdot} \circ F) \enspace .$$

For each $y \in Y$, we have $\mathsf{adv}_{\mathsf{G}}(\delta_{y,\cdot} \circ F) \le R(\mathsf{comp}(\delta_{y,\cdot} \circ F))$ by the assumption on G, while $R(\mathsf{comp}(\delta_{y,\cdot} \circ F)) \le R(T)$ since $\mathsf{comp}(\delta_{y,\cdot} \circ F) \le T$ and $R(t)$ is a non-decreasing function. Hence we have

$$\mathsf{SD}(F(U_{O_{\mathsf{G}}}), F(\mathsf{G}(U_{S_{\mathsf{G}}}))) \le \frac{1}{2} \sum_{y \in Y} R(T) = \frac{|Y|}{2} \cdot R(T) \enspace ,$$

therefore Proposition 2.1 holds. ☐

3 Formal Description of the Main Result

3.1 Flowchart Expressions of Procedures

In this subsection, we introduce some objects used for a formal description of our main result. In what follows we assume, unless otherwise specified, that any directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$ is finite (i.e., $|\mathcal{V}|, |\mathcal{E}| < \infty$), acyclic (i.e., having no directed cycles) and simple (i.e., having no parallel edges). Let $\mathsf{Pre}(v) = \mathsf{Pre}_{\mathcal{G}}(v)$ denote the set of predecessors $u \in \mathcal{V}$ of v in $\mathcal{G}$; $\mathsf{Pre}(v) = \{u \in \mathcal{V} \mid e = (u \to v) \in \mathcal{E}\}$. Let $\mathcal{V}_{\mathrm{src}}$ and $\mathcal{V}_{\mathrm{sin}}$ denote the sets of sources (vertices with no predecessors) and of sinks (vertices that are predecessors of no vertices) of $\mathcal{G}$, respectively.

Definition 3.1. In this article, a *flowchart* signifies a tuple $\mathcal{F} = (\mathcal{V}, \mathcal{E}, \mathcal{X}, \mathcal{A})$ satisfying the following conditions:

- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a directed graph;
- To each vertex $v \in \mathcal{V}$ a finite set X_v is associated; $\mathcal{X} = (X_v)_{v \in \mathcal{V}}$;
- To each $v \in \mathcal{V} \setminus \mathcal{V}_{\mathrm{src}}$ an algorithm A_v is associated, where the output set of A_v is X_v and the input set of A_v is the product $\vec{X}_{\mathsf{Pre}(v)}$ of the sets X_u over all $u \in \mathsf{Pre}(v)$; $\mathcal{A} = (\mathsf{A}_v)_{v \in \mathcal{V} \setminus \mathcal{V}_{\mathrm{src}}}$.

Here, for a subset $\mathcal{V}'$ of $\mathcal{V}$, $\vec{X}_{\mathcal{V}'}$ denotes the product of the sets X_v over all $v \in \mathcal{V}'$. Fig. 2 shows an example of flowcharts, where the first algorithm $\mathsf{A}_4 : X_1 \times X_2 \to X_4$ is integer addition and the second one $\mathsf{A}_5 : X_4 \times X_3 \to X_5$ is integer multiplication. Then we associates to each flowchart $\mathcal{F}$ an algorithm $\mathsf{A}(\mathcal{F})$ in the following manner:

Definition 3.2. Let $\mathcal{F} = (\mathcal{V}, \mathcal{E}, \mathcal{X}, \mathcal{A})$ be a flowchart. We define an algorithm $\mathsf{A}(\mathcal{F})$ with input set $\vec{X}_{\mathcal{V}_{\mathrm{src}}}$ and output set $\vec{X}_{\mathcal{V}_{\mathrm{sin}}}$, in the following inductive manner. Suppose that an element $x_v \in X_v$ is given for each $v \in \mathcal{V}_{\mathrm{src}}$ as input for the algorithm $\mathsf{A}(\mathcal{F})$. Then, when an element $x_u \in X_u$ has been determined for every predecessor $u \in \mathsf{Pre}(v)$ of a vertex $v \in \mathcal{V}$ but $x_v \in X_v$ has not been determined, an element $x_v \in X_v$ is determined as the output of the algorithm A_v with input $(x_u)_{u \in \mathsf{Pre}(v)}$. Finally, $\mathsf{A}(\mathcal{F})$ outputs the tuple $(x_v)_{v \in \mathcal{V}_{\mathrm{sin}}}$ of elements x_v with $v \in \mathcal{V}_{\mathrm{sin}}$.

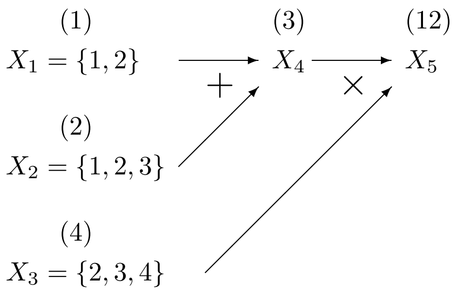

Figure 2: Example of a flowchart and the corresponding algorithm

In Fig. 2, the values in the parentheses show an example of calculation of the algorithm $\mathsf{A}(\mathcal{F})$ corresponding to the flowchart $\mathcal{F}$, where we have $\mathsf{A}(\mathcal{F})(x_1, x_2, x_3) = (x_1 + x_2) \cdot x_3$. We use such a flowchart and the corresponding algorithm for describing a procedure of security evaluations of cryptographic schemes. See the left-hand side of Fig. 1 in Section 1.2 for an example. In the figure, the circled arrow from L to Z representing the attack algorithm At signifies that no effective bound for computational complexity of the algorithm is given. Similar rules for pictures of flowcharts will be applied throughout this article.

We give some more definitions. Let $\mathcal{F}$ be a flowchart, and assume that $X_v = \Sigma = \{0, 1\}$ for every $v \in \mathcal{V}_{\mathrm{sin}}$. Given a subset $\mathcal{U} \subset \mathcal{V} \setminus \mathcal{V}_{\mathrm{src}}$ and a source $v_0 \in \mathcal{V}_{\mathrm{src}}$ of $\mathcal{G}$, we define another flowchart $\widetilde{\mathcal{F}} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}}, \widetilde{\mathcal{X}}, \widetilde{\mathcal{A}})$ in the following manner (see also the examples below). First, let $\mathcal{V}'$ be the set of all $v \in \mathcal{V}$ such that there is a path $(v_0 \to v_1 \to \cdots \to v_k = v)$ in $\mathcal{G}$ from v_0 to v which does not contain any vertex $u \in \mathcal{U}$. Note that $v_0 \in \mathcal{V}'$ and $\mathcal{U} \cap \mathcal{V}' = \emptyset$. Let

$$\mathcal{U}' = \{v \in \mathcal{V}' \mid v \in \mathsf{Pre}(u) \text{ for some } u \in \mathcal{U}\} \enspace ,$$
$$\mathcal{U}'' = \{v \in \mathcal{V} \setminus \mathcal{V}' \mid v \in \mathsf{Pre}(u) \text{ for some } u \in \mathcal{V}'\} \enspace .$$

We prepare a symbol v_u for each $u \in \mathcal{U}'$, and put $\widetilde{\mathcal{U}'} = \{v_u \mid u \in \mathcal{U}'\}$. Moreover, we prepare another symbol v_*. Then define

$$\widetilde{\mathcal{V}} = \mathcal{V}' \sqcup \mathcal{U}'' \sqcup \widetilde{\mathcal{U}'} \sqcup \{v_*\} \enspace ,$$

where $\sqcup$ denotes the disjoint union.

The remaining definition for $\widetilde{\mathcal{F}}$ depends on a collection of elements $a_v \in X_v$ for $v \in \mathcal{U}' \cup \mathcal{U}''$. Put

$$\begin{aligned}\widetilde{\mathcal{E}} = \{e = (v_1 \to v_2) \in \mathcal{E} \mid v_1 \in \mathcal{V}' \cup \mathcal{U}'', v_2 \in \mathcal{V}'\} \\ \sqcup \{e_u = (u \to v_u) \mid u \in \mathcal{U}'\} \sqcup \{e'_v = (v \to v_*) \mid v \in (\mathcal{V}' \cap \mathcal{V}_{\mathrm{sin}}) \cup \widetilde{\mathcal{U}'}\} \enspace .\end{aligned}$$

Then the pair $(\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}})$ defines an acyclic directed graph $\widetilde{\mathcal{G}}$. A direct argument shows that $\widetilde{\mathcal{G}}$ has the source set $\{v_0\} \cup \mathcal{U}''$ and the unique sink v_*, and we have $\mathsf{Pre}_{\mathcal{G}}(v) \subset \mathcal{V}' \cup \mathcal{U}''$ and $\mathsf{Pre}_{\widetilde{\mathcal{G}}}(v) = \mathsf{Pre}_{\mathcal{G}}(v)$ for every $v \in \mathcal{V}'$. To define the collection $\widetilde{\mathcal{X}} = (\widetilde{X}_v)_v$, we put

$$\widetilde{X}_v = X_v \text{ for } v \in \mathcal{V}' \, , \, \widetilde{X}_v = \{a_v\} \subset X_v \text{ for } v \in \mathcal{U}'' \, , \, \widetilde{X}_v = \Sigma \text{ for } v \in \widetilde{\mathcal{U}'} \cup \{v_*\} \enspace .$$

Moreover, to define the collection $\widetilde{\mathcal{A}} = (\widetilde{\mathsf{A}}_v)_v$, for each $v \in \mathcal{V}' \setminus \{v_0\}$ let $\widetilde{\mathsf{A}}_v$ be the same algorithm as A_v but each component of its input from the set X_u with $u \in \mathsf{Pre}_{\mathcal{G}}(v) \cap \mathcal{U}''$ is set to be the constant value $a_u \in \widetilde{X}_u$. For each $v_u \in \widetilde{\mathcal{U}'}$ put $\widetilde{\mathsf{A}}_{v_u} = \delta_{a_u,\cdot} : \widetilde{X}_u \to \Sigma$ (see Section 2 for the Kronecker delta algorithm $\delta_{y,\cdot}$). Finally, let $\widetilde{\mathsf{A}}_{v_*}$ be an algorithm, with input taken from the product of the sets $\widetilde{X}_v = \Sigma$ over $v \in (\mathcal{V}' \cap \mathcal{V}_{\mathrm{sin}}) \cup \widetilde{\mathcal{U}'}$, such that it outputs 1 if all components of the input are 1 and outputs 0 otherwise (i.e., an algorithm for bit-wise AND operation). Thus the flowchart $\widetilde{\mathcal{F}}$ is defined. Note that for the corresponding algorithm $\mathsf{A}(\widetilde{\mathcal{F}})$, all of the components x_v of the input $(x_v)_{v \in \{v_0\} \cup \mathcal{U}''}$ but x_{v_0} are set to be the constant values a_v, therefore $\mathsf{A}(\widetilde{\mathcal{F}})$ is essentially an algorithm with input set $\widetilde{X}_{v_0} = X_{v_0}$ and output set $\widetilde{X}_{v_*} = \Sigma$.

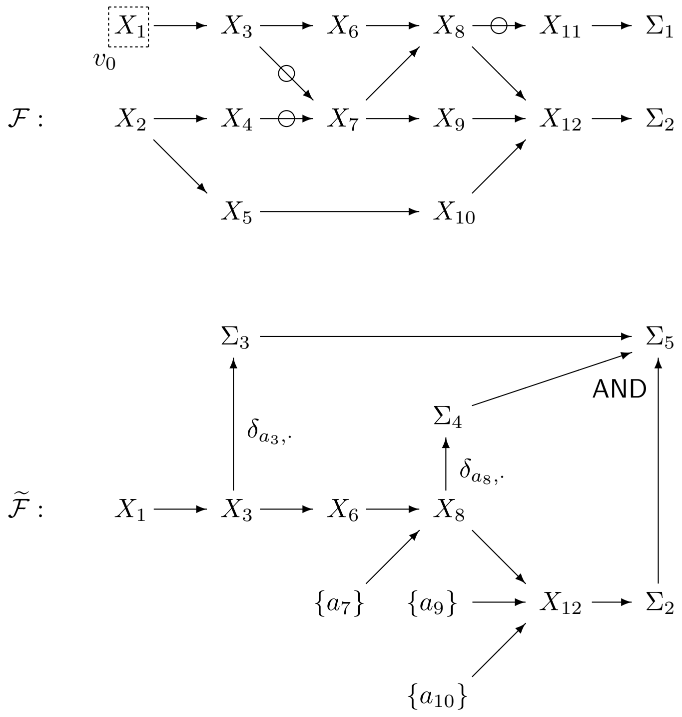

Figure 3: Example of “generic” flowchart

The upper half of Fig. 3 shows an example of a “generic” flowchart $\mathcal{F}$. Let Σ_k denote a copy of Σ. For simplicity, here we identify each vertex v of the graph with the corresponding set X_v; the same identification will be applied to other cases as well, unless some ambiguity occurs. In applications of our result, we will introduce the above flowchart $\widetilde{\mathcal{F}}$ by choosing the set $\mathcal{U}$ as consisting of all $v \in \mathcal{V}$ such that the algorithm A_v has no specified bound of complexity (in the picture of $\mathcal{F}$, $\mathcal{U}$ is the set of the terminal vertices of circled arrows). Hence we set $\mathcal{U} = \{X_7, X_{11}\}$ in the case of Fig. 3. On the other hand, we set $v_0 = X_1$. Then by the definition, we have $\mathcal{V}' = \{X_1, X_3, X_6, X_8, X_{12}, \Sigma_2\}$, $\mathcal{U}' = \{X_3, X_8\}$, and $\mathcal{U}'' = \{X_7, X_9, X_{10}\}$. Put $v_{X_7} = \Sigma_3$, $v_{X_{11}} = \Sigma_4$ and $v_* = \Sigma_5$, therefore $\widetilde{\mathcal{U}'} = \{\Sigma_3, \Sigma_4\}$. To obtain the edges of $\widetilde{\mathcal{G}}$, we start with the subgraph of $\mathcal{G}$ restricted to the vertex subset $\mathcal{V}' \subset \mathcal{V}$, and we add the arrows in $\mathcal{G}$ from a vertex in $\mathcal{U}''$ to a vertex in $\mathcal{V}'$, the arrows from some $u \in \mathcal{U}'$ to $v_u \in \widetilde{\mathcal{U}'}$, and the arrows from some vertex in $(\mathcal{V}' \cap \mathcal{V}_{\mathrm{sin}}) \cup \widetilde{\mathcal{U}'}$ to v_*. Thus we obtain the flowchart $\widetilde{\mathcal{F}}$ as in the lower half of Fig. 3, where (according to the definition) each X_v ($v \in \mathcal{U}''$) is replaced with $\{a_v\}$ ($a_v \in X_v$). As mentioned above, the corresponding algorithm $\mathsf{A}(\widetilde{\mathcal{F}})$ has essentially the input set X_1.

Fig. 1 in Section 1.2 gives another example of the flowchart. The upper half is the flowchart $\mathcal{F}$, and the lower half is the flowchart $\widetilde{\mathcal{F}}$ corresponding to elements $\ell \in L$ and $z \in Z$, where we set $v_0 = X$ and $\mathcal{U} = \{Z\}$, therefore $\mathcal{U}' = \{L\}$ and $\mathcal{U}'' = \{Z\}$. Now the resulting algorithm $\mathsf{A}(\widetilde{\mathcal{F}}) : X \to \{0,1\}$ outputs 1 for an input $x \in X$ if and only if $\mathsf{Lk}(x) = \ell$ and $\mathsf{Ev}(f(x), z) = 1$ (see also (1)).

3.2 Main Theorem

From now, we present our main theorem formally by using the above definitions. Here we introduce some notations. Given a flowchart $\mathcal{F}$ and a collection $\mathcal{R} = (r_v)_{v \in \mathcal{V}_{\mathrm{src}}}$ of random variables r_v on the sets X_v ($v \in \mathcal{V}_{\mathrm{src}}$), let $\mathsf{A}(\mathcal{F})(\mathcal{R})$ denote the output distribution of the algorithm $\mathsf{A}(\mathcal{F})$ with input given by the random variables r_v ($v \in \mathcal{V}_{\mathrm{src}}$). Let $\vec{1}$ denote a collection of copies of $1 \in \Sigma$. Then our main theorem is described as follows:

Theorem 3.1. *Let* $\mathsf{comp} : \mathsf{Alg} \to \mathbb{R}_{\geq 0}$ *be a complexity measure. Let* $\mathcal{F}$ *be a flowchart such that* $X_v = \Sigma = \{0,1\}$ *for every* $v \in \mathcal{V}_{\mathrm{sin}}$*. Let* $\mathcal{U} \subset \mathcal{V} \setminus \mathcal{V}_{\mathrm{src}}$ *and* $v_0 \in \mathcal{V}_{\mathrm{src}}$*. Let* $\mathsf{G} : S_{\mathsf{G}} \to X_{v_0}$ *be a PRG with output set* $O_{\mathsf{G}} = X_{v_0}$*. Let* $\mathcal{R}_{\mathrm{rnd}} = (r_v)_{v \in \mathcal{V}_{\mathrm{src}}}$ *be a collection of random variables* r_v *on* X_v *(*$v \in \mathcal{V}_{\mathrm{src}}$*) such that* r_{v_0} *is uniformly random, and let* $\mathcal{R}_{\mathrm{prnd}} = (r'_v)_{v \in \mathcal{V}_{\mathrm{src}}}$ *be obtained from* $\mathcal{R}_{\mathrm{rnd}}$ *by replacing* r_{v_0} *with the random variable* r'_{v_0} *given by the output of* G *for uniformly random seeds. Assume that*

- G *is* $R(t)$*-secure with respect to* $(\mathsf{comp}, \mathcal{C})$ *for a subset* $\mathcal{C} \subset \mathsf{Alg}$*,*
- *there exists a constant* $T > 0$ *such that for every collection of elements* $a_v \in X_v$ *for* $v \in \mathcal{U}' \cup \mathcal{U}''$*, the corresponding flowchart* $\widetilde{\mathcal{F}}$ *satisfies that* $\mathsf{A}(\widetilde{\mathcal{F}}) \in \mathcal{C}$ *and* $\mathsf{comp}(\mathsf{A}(\widetilde{\mathcal{F}})) \leq T$*.*

Then we have

$$|Pr[\vec{1} \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{rnd}})] - Pr[\vec{1} \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{prnd}})]| \leq \left(\prod_{v \in \mathcal{U}'} |X_v| \right) R(T) \enspace .$$

Proof. Given elements a_v of X_v ($v \in \mathcal{U}'$), we define an algorithm A' with input set $\vec{X}_{\mathcal{V}_{\mathrm{src}} \setminus \{v_0\}}$ and output set $\vec{X}_{\mathcal{U}'' \cup (\mathcal{V}_{\mathrm{sin}} \setminus \mathcal{V}')}$ in the following inductive manner:

1. Set $(x_v)_{v \in \mathcal{V}_{\mathrm{src}} \setminus \{v_0\}}$ to be the given input for A'.
2. For each $v \in \mathcal{U}'$, set $x_v = a_v$.
3. If $v \in \mathcal{V} \setminus \mathcal{V}'$ and x_u has been determined for every $u \in \mathsf{Pre}_{\mathcal{G}}(v)$ but x_v has not been determined, then set $x_v \leftarrow \mathsf{A}_v((x_u)_{u \in \mathsf{Pre}_{\mathcal{G}}(v)})$. Repeat the process until x_v is determined for every $v \in \mathcal{V} \setminus \mathcal{V}'$.
4. Finally, output $(x_v)_{v \in \mathcal{U}'' \cup (\mathcal{V}_{\mathrm{sin}} \setminus \mathcal{V}')}$.

Note that x_v is determined for every $v \in \mathcal{V} \setminus \mathcal{V}'$ by repeating the process in Step 3. Indeed, assume contrary that some x_v cannot be determined, and we choose such a $v \in \mathcal{V} \setminus \mathcal{V}'$ that is closest to sources in $\mathcal{G}$. Then we have $\mathsf{Pre}_{\mathcal{G}}(v) \subset (\mathcal{V} \setminus \mathcal{V}') \cup \mathcal{U}'$, therefore every x_u with $u \in \mathsf{Pre}_{\mathcal{G}}(v)$ can be determined by the choice of v, while x_v cannot be determined. This is a contradiction. Hence every x_v is determined, therefore the algorithm A' is well-defined (it is shown by induction that the calculation of the elements x_v is independent of the order of choices of vertices v). Let $\mathcal{R}'$ be the collection of random variables obtained by removing r_{v_0} from $\mathcal{R}_{\mathrm{rnd}}$, or equivalently, by removing r'_{v_0} from $\mathcal{R}_{\mathrm{prnd}}$; namely $\mathcal{R}' = (r_v)_{v \in \mathcal{V}_{\mathrm{src}} \setminus \{v_0\}}$. We define a probability distribution $\mathsf{A}'(\mathcal{R}')$ over $\vec{X}_{\mathcal{U}''}$ in the same way as $\mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{rnd}})$ and $\mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{prnd}})$.

For simplicity, given elements a_v, x_v ($v \in \mathcal{V}$), we write $\vec{a}_{\to v} = (a_u)_{u \in \mathsf{Pre}_{\mathcal{G}}(v)}$ and $\vec{x}_{\to v} = (x_u)_{u \in \mathsf{Pre}_{\mathcal{G}}(v)}$ for $v \in \mathcal{V}$; $p_v = Pr[a_v \leftarrow r_v]$ and $p'_v = Pr[x_v \leftarrow r_v]$ for $v \in \mathcal{V}_{\mathrm{src}}$; and $p_v = Pr[a_v \leftarrow \mathsf{A}_v(\vec{a}_{\to v})]$ and $p'_v = Pr[x_v \leftarrow \mathsf{A}_v(\vec{x}_{\to v})]$ for $v \in \mathcal{V} \setminus \mathcal{V}_{\mathrm{src}}$. Put $\mathcal{Z}_1 = \mathcal{V}' \cup \mathcal{U}'' \cup \mathcal{V}_{\mathrm{sin}}$. Then by the definition of $\mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{rnd}})$, we have

$$Pr[\vec{1} \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{rnd}})] = \sum_{\substack{(a_v)_{v \in \mathcal{V}} \in \vec{X}_{\mathcal{V}} \\ (a_v)_{v \in \mathcal{V}_{\mathrm{sin}}} = \vec{1}}} \prod_{v \in \mathcal{V}} p_v = \sum_{\substack{(a_v)_{v \in \mathcal{Z}_1} \\ (a_v)_{v \in \mathcal{V}_{\mathrm{sin}}} = \vec{1}}} \sum_{(a_v)_{v \in \mathcal{V} \setminus \mathcal{Z}_1}} \prod_{v \in \mathcal{V}} p_v \enspace .$$

By factoring out some terms, the last value is equal to

$$\sum_{\substack{(a_v)_{v \in \mathcal{Z}_1} \\ (a_v)_{v \in \mathcal{V}_{\mathrm{sin}}} = \vec{1}}} \left(\prod_{v \in \mathcal{V}'} p_v \sum_{(a_v)_{v \in \mathcal{V} \setminus \mathcal{Z}_1}} \prod_{v \in \mathcal{V} \setminus \mathcal{V}'} p_v \right) \tag{2}$$

(note that $\mathsf{Pre}_{\mathcal{G}}(v) \subset \mathcal{V}' \cup \mathcal{U}''$ for every $v \in \mathcal{V}' \setminus \{v_0\}$, therefore the terms p_v with $v \in \mathcal{V}'$ can indeed be factored out).

We would like to calculate the sum in the parenthesis in (2) for given elements $a_v \in X_v$ ($v \in \mathcal{Z}_1$). First, note that $\mathsf{Pre}_{\mathcal{G}}(v) \subset (\mathcal{V} \setminus \mathcal{V}') \cup \mathcal{U}'$ for every $v \in \mathcal{V} \setminus \mathcal{V}'$, therefore the values p_v in the sum depend on the elements a_u ($u \in (\mathcal{V} \setminus \mathcal{V}') \cup \mathcal{U}'$). Secondly, we have

$$((\mathcal{V} \setminus \mathcal{V}') \cup \mathcal{U}') \setminus (\mathcal{V} \setminus \mathcal{Z}_1) = (\mathcal{V} \setminus \mathcal{V}') \cup \mathcal{U}') \cap \mathcal{Z}_1 = (\mathcal{Z}_1 \setminus \mathcal{V}') \cup (\mathcal{U}' \cap \mathcal{Z}_1) = \mathcal{Z}_2 \enspace ,$$

where $\mathcal{Z}_2 = \mathcal{U}' \sqcup \mathcal{U}'' \sqcup (\mathcal{V}_{\mathrm{sin}} \setminus \mathcal{V}')$ (disjoint union). Hence, given elements $a_v \in X_v$ $(v \in \mathcal{Z}_1)$, we have

$$\sum_{(a_v)_{v \in \mathcal{V} \setminus \mathcal{Z}_1}} \prod_{v \in \mathcal{V} \setminus \mathcal{V}'} p_v = \sum_{\substack{(x_v)_{v \in (\mathcal{V} \setminus \mathcal{V}') \cup \mathcal{U}'} \\ x_v = a_v \ (\forall v \in \mathcal{Z}_2)}} \prod_{v \in \mathcal{V} \setminus \mathcal{V}'} p'_v$$

$$= \sum_{\substack{(x_v)_{v \in (\mathcal{V} \setminus \mathcal{V}') \cup \mathcal{U}'} \\ x_v = a_v \ (\forall v \in \mathcal{U}')}} Pr[(x_v)_{v \in \mathcal{V}_{\mathrm{src}} \setminus \{v_0\}} \leftarrow \mathcal{R}' \, , \, x_v \leftarrow \mathsf{A}_v(\vec{x}_{\to v}) \ (\forall v \in \mathcal{V} \setminus (\mathcal{V}' \cup \mathcal{V}_{\mathrm{src}})) \, , \, x_v = a_v \ (\forall v \in \mathcal{Z}_2 \setminus \mathcal{U}') \ ] \ .$$

By the definition of A', the last value is equal to

$$\sum_{(x_v)_{v \in \mathcal{V}_{\mathrm{src}} \setminus \{v_0\}}} Pr[(x_v)_{v \in \mathcal{V}_{\mathrm{src}} \setminus \{v_0\}} \leftarrow \mathcal{R}' \, , \, (a_v)_{v \in \mathcal{Z}_2 \setminus \mathcal{U}'} \leftarrow \mathsf{A}'((x_v)_{v \in \mathcal{V}_{\mathrm{src}} \setminus \{v_0\}})] = Pr[(a_v)_{v \in \mathcal{Z}_2 \setminus \mathcal{U}'} \leftarrow \mathsf{A}'(\mathcal{R}')] \ ,$$

where the algorithm A' is corresponding to the given elements $a_v \in X_v$ $(v \in \mathcal{U}')$.

By substituting the above equality for (2), we have

$$\begin{aligned} Pr[\vec{1} \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{rnd}})] &= \sum_{\substack{(a_v)_{v \in \mathcal{Z}_1} \\ (a_v)_{v \in \mathcal{V}_{\mathrm{sin}}} = \vec{1}}} \prod_{v \in \mathcal{V}'} p_v \cdot Pr[(a_v)_{v \in \mathcal{Z}_2 \setminus \mathcal{U}'} \leftarrow \mathsf{A}'(\mathcal{R}')] \\ &= \sum_{\substack{(a_v)_{v \in \mathcal{Z}_3} \\ (a_v)_{v \in \mathcal{V}_{\mathrm{sin}}} = \vec{1}}} \left(Pr[(a_v)_{v \in \mathcal{Z}_2 \setminus \mathcal{U}'} \leftarrow \mathsf{A}'(\mathcal{R}')] \sum_{(a_v)_{v \in \mathcal{Z}_4}} \prod_{v \in \mathcal{V}'} p_v \right) \ , \end{aligned} \tag{3}$$

where $\mathcal{Z}_3 = \mathcal{U}' \cup \mathcal{U}'' \cup \mathcal{V}_{\mathrm{sin}}$ and $\mathcal{Z}_4 = \mathcal{V}' \setminus (\mathcal{U}' \cup \mathcal{V}_{\mathrm{sin}})$ (note that $\mathcal{Z}_1 \setminus \mathcal{Z}_3 = \mathcal{Z}_4$).

Now we would like to calculate the sum in the parenthesis in the right-hand side of (3) for given elements $a_v \in X_v$ $(v \in \mathcal{Z}_3)$ such that $a_v = 1$ for all $v \in \mathcal{V}' \cap \mathcal{V}_{\mathrm{sin}}$. First, note that $\mathsf{Pre}_{\mathcal{G}}(v) \subset \mathcal{V}' \cup \mathcal{U}''$ for every $v \in \mathcal{V}'$, therefore the values p_v in the sum depend on the elements a_u $(u \in \mathcal{V}' \cup \mathcal{U}'')$. Secondly, we have

$$(\mathcal{V}' \cup \mathcal{U}'') \setminus \mathcal{Z}_4 = (\mathcal{V}' \setminus \mathcal{Z}_4) \cup (\mathcal{U}'' \setminus \mathcal{Z}_4) = (\mathcal{V}' \cap (\mathcal{U}' \cup \mathcal{V}_{\mathrm{sin}})) \cup \mathcal{U}'' = \mathcal{U}' \sqcup (\mathcal{V}' \cap \mathcal{V}_{\mathrm{sin}}) \sqcup \mathcal{U}''$$

(where the right-hand side is disjoint union). Hence, for given elements $a_v \in X_v$ $(v \in \mathcal{Z}_3)$ such that $a_v = 1$ for all $v \in \mathcal{V}' \cap \mathcal{V}_{\mathrm{sin}}$, we have

$$\sum_{(a_v)_{v \in \mathcal{Z}_4}} \prod_{v \in \mathcal{V}'} p_v$$

$$= \sum_{\substack{(x_v)_{v \in \mathcal{V}' \cup \mathcal{U}''} \\ x_v = a_v \ (\forall v \in \mathcal{U}' \cup \mathcal{U}'') \\ x_v = 1 \ (\forall v \in \mathcal{V}' \cap \mathcal{V}_{\mathrm{sin}})}} Pr[x_{v_0} \leftarrow r_{v_0} \, , \, x_v \leftarrow \mathsf{A}_v(\vec{x}_{\to v}) \ (\forall v \in \mathcal{V}' \setminus \{v_0\}) \ ]$$

$$= \sum_{\substack{(x_v)_{v \in \mathcal{V}' \cup \mathcal{U}''} \\ x_v = a_v \ (\forall v \in \mathcal{U}'')}} Pr[x_{v_0} \leftarrow r_{v_0}, x_v \leftarrow \mathsf{A}_v(\vec{x}_{\to v}) \ (\forall v \in \mathcal{V}' \setminus \{v_0\}) \, , \, x_v = a_v \ (\forall v \in \mathcal{U}') \, , \, x_v = 1 \ (\forall v \in \mathcal{V}' \cap \mathcal{V}_{\mathrm{sin}}) \ ] \ .$$

By definition of the algorithms $\delta_{a_v,\cdot}$, the last row is equal to

$$\sum_{\substack{(x_v)_{v \in \mathcal{V}' \cup \mathcal{U}''} \\ x_v = a_v \ (\forall v \in \mathcal{U}'')}} Pr[x_{v_0} \leftarrow r_{v_0}, x_v \leftarrow \mathsf{A}_v(\vec{x}_{\to v}) \ (\forall v \in \mathcal{V}' \setminus \{v_0\}) \, , \, \delta_{a_v,\cdot}(x_v) = 1 \ (\forall v \in \mathcal{U}') \, , \, x_v = 1 \ (\forall v \in \mathcal{V}' \cap \mathcal{V}_{\mathrm{sin}}) \ ] \ .$$

Moreover, by the definition of $\widetilde{\mathcal{F}}$, the last value is equal to

$$\sum_{x_{v_0} \in X_{v_0}} Pr[x_{v_0} \leftarrow r_{v_0}, 1 \leftarrow \mathsf{A}(\widetilde{\mathcal{F}})(x_{v_0})] = Pr[1 \leftarrow \mathsf{A}(\widetilde{\mathcal{F}})(r_{v_0})] \ ,$$

where the flowchart $\widetilde{\mathcal{F}}$ is corresponding to the given elements $a_v \in X_v$ $(v \in \mathcal{U}' \cup \mathcal{U}'')$.

By substituting the above equality for (3), we have

$$Pr[\vec{1} \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{rnd}})] = \sum_{\substack{(a_v)_{v \in \mathcal{Z}_3} \\ (a_v)_{v \in \mathcal{V}_{\mathrm{sin}}} = \vec{1}}} \left(Pr[(a_v)_{v \in \mathcal{Z}_2 \setminus \mathcal{U}'} \leftarrow \mathsf{A}'(\mathcal{R}')] Pr[1 \leftarrow \mathsf{A}(\widetilde{\mathcal{F}})(r_{v_0})] \right) \ . \tag{4}$$

The same argument for $\mathcal{R}_{\mathrm{prnd}}$ instead of $\mathcal{R}_{\mathrm{rnd}}$ implies that

$$Pr[\vec{1} \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{prnd}})] = \sum_{\substack{(a_v)_{v \in \mathcal{Z}_3} \\ (a_v)_{v \in \mathcal{V}_{\mathrm{sin}}} = \vec{1}}} \left(Pr[(a_v)_{v \in \mathcal{Z}_2 \setminus \mathcal{U}'} \leftarrow \mathsf{A}'(\mathcal{R}')] Pr[1 \leftarrow \mathsf{A}(\widetilde{\mathcal{F}})(r'_{v_0})] \right) \ . \tag{5}$$

By using (4), (5) and triangle inequality, we have

$$\begin{aligned} & |Pr[\vec{1} \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{rnd}})] - Pr[\vec{1} \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{prnd}})]| \\ & \leq \sum_{\substack{(a_v)_{v \in \mathcal{Z}_3} \\ (a_v)_{v \in \mathcal{V}_{\mathrm{sin}}} = \vec{1}}} \left(Pr[(a_v)_{v \in \mathcal{Z}_2 \setminus \mathcal{U}'} \leftarrow \mathsf{A}'(\mathcal{R}')] \cdot |Pr[1 \leftarrow \mathsf{A}(\widetilde{\mathcal{F}})(r_{v_0})] - Pr[1 \leftarrow \mathsf{A}(\widetilde{\mathcal{F}})(r'_{v_0})]| \right) \ . \end{aligned} \tag{6}$$

By the assumption, we have $X_{v_0} = O_{\mathsf{G}}$, $r_{v_0} = U_{O_{\mathsf{G}}}$ and $r'_{v_0} = \mathsf{G}(U_{S_{\mathsf{G}}})$, therefore

$$|Pr[1 \leftarrow \mathsf{A}(\widetilde{\mathcal{F}})(r_{v_0})] - Pr[1 \leftarrow \mathsf{A}(\widetilde{\mathcal{F}})(r'_{v_0})]| = \mathsf{adv}_{\mathsf{G}}(\mathsf{A}(\widetilde{\mathcal{F}})) \ .$$

Since G is $R(t)$-secure with respect to $(\mathsf{comp}, \mathcal{C})$, the assumption on $\widetilde{\mathcal{F}}$ implies that

$$\mathsf{adv}_{\mathsf{G}}(\mathsf{A}(\widetilde{\mathcal{F}})) \leq R(\mathsf{comp}(\mathsf{A}(\widetilde{\mathcal{F}}))) \leq R(T)$$

(recall that $R(t)$ is a non-decreasing function). By substituting these for (6), we have

$$|Pr[\vec{1} \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{rnd}})] - Pr[\vec{1} \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{prnd}})]| \leq \sum_{\substack{(a_v)_{v \in \mathcal{Z}_3} \\ (a_v)_{v \in \mathcal{V}_{\mathrm{sin}}} = \vec{1}}} Pr[(a_v)_{v \in \mathcal{Z}_2 \setminus \mathcal{U}'} \leftarrow \mathsf{A}'(\mathcal{R}')] R(T) \ .$$

Note that the value $Pr[(a_v)_{v \in \mathcal{Z}_2 \setminus \mathcal{U}'} \leftarrow \mathsf{A}'(\mathcal{R}')]$ in the right-hand side does not depend on the elements a_v $(v \in \mathcal{V}' \cap \mathcal{V}_{\mathrm{sin}})$. Since $\mathcal{Z}_3 \setminus (\mathcal{V}' \cap \mathcal{V}_{\mathrm{sin}}) = \mathcal{Z}_2$, the last sum is equal to

$$\begin{aligned} \sum_{\substack{(a_v)_{v \in \mathcal{Z}_2} \\ (a_v)_{v \in \mathcal{V}_{\mathrm{sin}} \setminus \mathcal{V}'} = \vec{1}}} Pr[(a_v)_{v \in \mathcal{Z}_2 \setminus \mathcal{U}'} \leftarrow \mathsf{A}'(\mathcal{R}')] R(T) & \leq \sum_{(a_v)_{v \in \mathcal{Z}_2}} Pr[(a_v)_{v \in \mathcal{Z}_2 \setminus \mathcal{U}'} \leftarrow \mathsf{A}'(\mathcal{R}')] R(T) \\ & = \sum_{(a_v)_{v \in \mathcal{U}'}} \left(R(T) \sum_{(a_v)_{v \in \mathcal{Z}_2 \setminus \mathcal{U}'}} Pr[(a_v)_{v \in \mathcal{Z}_2 \setminus \mathcal{U}'} \leftarrow \mathsf{A}'(\mathcal{R}')] \right) \ . \end{aligned}$$

By using the relation

$$\sum_{(a_v)_{v \in \mathcal{Z}_2 \setminus \mathcal{U}'}} Pr[(a_v)_{v \in \mathcal{Z}_2 \setminus \mathcal{U}'} \leftarrow \mathsf{A}'(\mathcal{R}')] = 1$$

it follows that

$$|Pr[\vec{1} \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{rnd}})] - Pr[\vec{1} \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{prnd}})]| \leq \sum_{(a_v)_{v \in \mathcal{U}'}} R(T) = |\vec{X}_{\mathcal{U}'}| \cdot R(T) = \left(\prod_{v \in \mathcal{U}'} |X_v| \right) R(T) \ .$$

Hence Theorem 3.1 holds. □

For practical applications, we consider the situation that an attack by an adversary for a protocol "succeeds" if and only if every component of output of the algorithm $\mathsf{A}(\mathcal{F})$ is 1. We assume that an element of X_{v_0} is originally given by a perfect random source and we would like to replace the perfect source with output of the PRG G. In this setting, the quantity $|Pr[\vec{1} \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{rnd}})] - Pr[\vec{1} \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_{\mathrm{prnd}})]|$ is the difference of the adversary's attack success probabilities between random and pseudorandom cases. Now we choose $\mathcal{U}$ as the set of all vertices corresponding to the output sets of the adversary's attack algorithms. Then each vertex in $\mathcal{U}'$ corresponds to (a part of) the set of information received by the adversary. In this case, it is an important property that, by the definition of $\widetilde{\mathcal{F}}$, *the algorithm* $\mathsf{A}(\widetilde{\mathcal{F}})$ *does not involve any attack algorithm of the adversary.* Hence the complexity of $\mathsf{A}(\widetilde{\mathcal{F}})$ can be effectively bounded *even if the attack algorithms have unbounded complexity*, therefore the assumption for Theorem 3.1 can be indeed satisfied. By Theorem 3.1, the difference between random and pseudorandom cases is bounded well when the product of sizes of the sets X_v ($v \in \mathcal{U}'$) is sufficiently small, which means intuitively the situation that the amount of information received by the adversary is sufficiently small. Moreover, the complexity measure comp can be chosen independently of the adversary's attack algorithms, therefore the bound of the difference between random and pseudorandom cases given by Theorem 3.1 is independent of the adversary's computational environment (for example, the adversary may use quantum computers even if the complexity measure comp is according to classical computation).

3.3 An Example

To help understanding the proof of Theorem 3.1, we write down the proof for the special case of Fig. 1 in Section 1.2. Here we set $v_0 = X$ and $\mathcal{U} = \{Z\}$, therefore $\mathcal{U}' = \{L\}$ and $\mathcal{U}'' = \{Z\}$. Let r_X be a random variable on X. Then we have

$$\begin{aligned} Pr[1 \leftarrow \mathsf{A}(\mathcal{F})(r_X)] &= \sum_{(x,y,\ell,z) \in X \times Y \times L \times Z} Pr[x \leftarrow r_X, y \leftarrow f(x), \ell \leftarrow \mathsf{Lk}(x)\,,\, z \leftarrow \mathsf{At}(\ell), 1 \leftarrow \mathsf{Ev}(y,z)] \\ &= \sum_{\ell,z} \sum_{x,y} \Big(Pr[x \leftarrow r_X] Pr[y \leftarrow f(x)] Pr[\ell \leftarrow \mathsf{Lk}(x)] Pr[z \leftarrow \mathsf{At}(\ell)] Pr[1 \leftarrow \mathsf{Ev}(y,z)] \Big) \\ &= \sum_{\ell,z} \left(Pr[z \leftarrow \mathsf{At}(\ell)] \sum_{x,y} \Big(Pr[x \leftarrow r_X] Pr[y \leftarrow f(x)] Pr[\ell \leftarrow \mathsf{Lk}(x)] Pr[1 \leftarrow \mathsf{Ev}(y,z)] \Big) \right) \enspace . \end{aligned}$$

Now for each $\ell \in L$ and $z \in Z$, we have

$$\begin{aligned} &\sum_{x,y} \Big(Pr[x \leftarrow r_X] Pr[y \leftarrow f(x)] Pr[\ell \leftarrow \mathsf{Lk}(x)] Pr[1 \leftarrow \mathsf{Ev}(y,z)] \Big) \\ &= \sum_{x} Pr[x \leftarrow r_X, 1 \leftarrow (\delta_{y,\cdot} \circ \mathsf{Lk})(x), 1 \leftarrow \mathsf{Ev}(f(x),z)] = \sum_{x} Pr[x \leftarrow r_X, 1 \leftarrow \mathsf{A}(\widetilde{\mathcal{F}})(x)] = Pr[1 \leftarrow \mathsf{A}(\widetilde{\mathcal{F}})(r_X)] \enspace , \end{aligned}$$

where the flowchart $\widetilde{\mathcal{F}}$ is corresponding to the elements ℓ and z. By using this equality, we have

$$Pr[1 \leftarrow \mathsf{A}(\mathcal{F})(r_X)] = \sum_{\ell,z} Pr[z \leftarrow \mathsf{At}(\ell)] Pr[1 \leftarrow \mathsf{A}(\widetilde{\mathcal{F}})(r_X)] \enspace .$$

By triangle inequality, it follows that

$$\begin{aligned} &|Pr[1 \leftarrow \mathsf{A}(\mathcal{F})(U_{O_{\mathsf{G}}})] - Pr[1 \leftarrow \mathsf{A}(\mathcal{F})(\mathsf{G}(U_{S_{\mathsf{G}}}))]| \\ &\leq \sum_{\ell,z} \Big(Pr[z \leftarrow \mathsf{At}(\ell)] \cdot |Pr[1 \leftarrow \mathsf{A}(\widetilde{\mathcal{F}})(U_{O_{\mathsf{G}}})] - Pr[1 \leftarrow \mathsf{A}(\widetilde{\mathcal{F}})(\mathsf{G}(U_{S_{\mathsf{G}}}))]| \Big) \leq \sum_{\ell,z} Pr[z \leftarrow \mathsf{At}(\ell)] \cdot \varepsilon \end{aligned}$$

(where we used in the last inequality the assumption on indistinguishability of G and complexity of $\widetilde{\mathsf{A}} = \mathsf{A}(\widetilde{\mathcal{F}})$, and $\varepsilon = R(T)$ in the notation of Theorem 3.1)

$$= \sum_{\ell \in L} \left(\varepsilon \cdot \sum_{z \in Z} Pr[z \leftarrow \mathsf{At}(\ell)] \right) = \sum_{\ell \in L} \varepsilon \cdot 1 = |L| \cdot \varepsilon \ .$$

Thus the desired bound mentioned in Section 1.2 is derived. Hence the difference between random and pseudorandom cases can be effectively bounded if the set L of possible information received by the adversary is sufficiently small (note that $\mathsf{A}(\widetilde{\mathcal{F}})$ is composed of efficient algorithms, therefore its complexity can be effectively bounded as well).

On the other hand, we notice that the evaluation technique by the preceding result of Dubrov and Ishai [11] using nb-PRGs is essentially *not* effective in the above case. Roughly speaking, an nb-PRG G is a PRG such that, even if the output set of a distinguisher D for G is not $\{0,1\}$ (i.e., D outputs more than one bits), the statistical distance of the output distributions of D between random and pseudorandom cases is effectively bounded provided the output set of D is sufficiently small. To apply their randomness reduction technique using the nb-PRG G, first we replace the uniform random variable on X with the output of G, and then we must find a decomposition of the algorithm $\mathsf{A}(\mathcal{F})$ of the form $\mathsf{A}(\mathcal{F}) = \mathsf{A} \circ \mathsf{D}$ such that A may have unbounded complexity but D has bounded complexity and output set of bounded size. (If such a decomposition is found, then the output distributions of D, hence those of $\mathsf{A}(\mathcal{F})$, in random and pseudorandom cases have a sufficiently small statistical distance, as desired.) However, in the case of Fig. 1 it is essentially impossible to find such a decomposition of $\mathcal{A}(\mathcal{F})$. Indeed, the possible choice of the efficient D is either $\mathsf{D} = f \times \mathsf{Lk} : X \to Y \times L$ or $\mathsf{D} : X \to X$ is trivial (doing nothing). In any case, the output set of D includes either X or Y, which should not be too small to make the original protocol secure in the random case (if X or Y is too small, then the success probability to guess the output $f(x)$ of f cannot be negligibly small). Hence the preceding technique in [11] is not effective for this example, therefore our result improves the preceding result significantly.

3.4 Miscellaneous Remarks

Here we collect some remarks on our result.

1. A Frequently Asked Question on our result: Why the adversary cannot recover the presently used seed of the *just computationally* secure PRG by using algorithms with *unbounded* complexity (which would break the proven indistinguishability between random and pseudorandom cases)? Answer: Our result requires the property of the situation that the set of possible information received by the adversary is sufficiently small. In such cases, the information actually received by the adversary is too scanty to recover the seed, even though the adversary can perform powerful computation.

2. Our result may provide a significant insight for randomness reduction of not only protocols with information-theoretic security, but also those with computational security. For instance, when the considered computationally secure protocol is post-quantum (i.e., secure against quantum adversaries), our result shows that indistinguishable randomness reduction is still possible even by using a PRG whose underlying computational problem is easy for quantum computers. The reason is that the indistinguishability of the PRG is evaluated with respect to a fixed complexity measure comp that is independent of the adversary's (quantum) algorithm, therefore comp may be classical.

3. Our result gives a bound of the difference of security between random and pseudorandom cases, which depends on computational complexity of the considered protocol. This means that the efficiency of the protocol contributes *directly* to the security level, which is a rare phenomenon. Indeed, in usual situations efficiency of the considered protocol contributes *just indirectly* to the security level, e.g., in such a way that the more efficient a protocol is, the larger the encryption/decryption keys used by the protocol can be, hence the higher the achieved security level will be.

4. A typical situation where our result works effectively is the following: There are a large number of players for the protocol, including a small number of adversaries, and just a small piece of an element generated from the randomness (which is the target of the randomness reduction) is distributed to each player. In such a situation, the amount of information on the randomness received by an adversary will be small, as required in our result. Now imagine that, if we could know in advance who are the adversaries among all players, then smaller randomness would suffice for fighting the exposed adversaries directly, since the information on the randomness received by the adversaries is now small. However, actually we have no practical way to know it in advance, and it is inevitable to fight huge possibilities of where the adversaries are hiding, requiring further randomness. The randomness for the latter purpose looks less essential than the former one, and our PRG-based randomness reduction can be intuitively thought of as reducing the latter inessential randomness. The security notion for PRGs (Definition 2.4) fits the purpose very well; advantages of distinguishers are bounded regardless of the bit positions (corresponding to the place of adversaries, in the above situation) that are picked up from outputs of a PRG.

5. In the above argument, we have carefully avoided the term "computationally unbounded adversary"; instead, we used, e.g., "computationally unbounded attack algorithm". The reason is that the exact meaning of "computationally unbounded adversary" seems depending on people, and someone may think that existence of "computationally unbounded adversary" breaks not only computational power assumptions but also computational hardness assumptions (in the sense of Definition 2.2). If it is the case, then our result cannot be applied against "computationally unbounded adversary", since our result is based on a computational hardness assumption on indistinguishability of the PRG. Nevertheless, our result can imply the following: By PRG-based randomness reduction, the random and pseudorandom cases can be indistinguishable even against an impractically strong adversary who is supposed to be able to perform *arbitrary* algorithms in *arbitrary* (theoretically consistent) computational environments. Hence anyway our result proves indistinguishability between random and pseudorandom cases much stronger than ordinary computational indistinguishability.

4 Preliminary for Numerical Example

In this section, we summarize some preceding results used in our numerical example given later, and also present further properties.

4.1 Pseudorandom Generators

The PRG used in our numerical example is the one given by Farashahi et al. [13, Section 4.1] under the DDH assumption. Here we summarize notations and some properties. Its construction uses two prime numbers p and q such that $p = 2q + 1$, hence p is a safe prime and q is a Sophie-Germain prime. Let $\mathbb{G}_1$ be the multiplicative group of nonzero quadratic residues modulo p, therefore $|\mathbb{G}_1| = q$. We identify the set $\mathbb{G}_1$ with $\mathbb{Z}_q$ via the bijection enum_1 used in [13, Section 4.1]. Under the identification, the PRG $\mathsf{G} = \mathsf{G}_{\mathrm{DDH}}$ called *DDH generator* with integer parameter $k_0 > 0$ has seed set $S_{\mathsf{G}} = (\mathbb{Z}_q)^3$ and output set $O_{\mathsf{G}} = (\mathbb{Z}_q)^{k_0}$. Note that in their construction, two elements x and y of $\mathbb{G}_1$ are randomly chosen as well as the "seed" s_0 of the PRG [13, Section 3.1], and here we include those random elements x and y in the seed of the PRG. We omit further details of the construction, since it is not relevant to our argument below.

In [13], indistinguishability of the PRG $\mathsf{G} = \mathsf{G}_{\mathrm{DDH}}$ is evaluated by using the data of computer experiments by Lenstra and Verheul [17]. Therefore we choose Alg as the set of classical algorithms, and let comp be the complexity measure such that $\mathsf{comp}(\mathsf{A})$ is the worst-case running time of $\mathsf{A} \in \mathsf{Alg}$ when executed on a fixed Pentium machine that was used in the experiments in [17]. (Note that it is not clear in [13] whether the running times are in the sense of average-case or of worst-case, and here we adopt the worst-case ones for safety since worst-case running time is longer than or equal to average-case running time.) The time unit is set to be 360 Pentium clock cycles that is, according to the experiment in [17], approximately the time for one encryption in a software implementation of DES (see also [13, Section 2.4]). Now [13, Theorem 2] shows that if there is a distinguisher $\mathsf{D} \in \mathsf{Alg}$ for $\mathsf{G}_{\mathrm{DDH}}$ such that $\mathsf{comp}(\mathsf{D}) \leq T$ and $\mathsf{adv}_{\mathsf{G}_{\mathrm{DDH}}}(\mathsf{D}) > \varepsilon$, then the DDH problem in the group $\mathbb{G}_1$ can be solved by some $\mathsf{A} \in \mathsf{Alg}$ such that $\mathsf{comp}(\mathsf{A}) \leq T$ with advantage larger than ε/k_0. Hence, by assuming that the time-success ratio T'/ε' for the complexity T' and the advantage ε' of any algorithm in Alg for the DDH problem in $\mathbb{G}_1$ does not exceed a constant R_{ts}, it follows that $\mathsf{G}_{\mathrm{DDH}}$ is $R(t)$-secure with respect to comp with $R(t) = k_0 t/R_{\mathrm{ts}}$. In [13, Assumption 1], the value R_{ts} is assumed to be the complexity of the best known algorithm for solving the DDH problem in $\mathbb{G}_1$, which is estimated according to the data in [17] as $R_{\mathrm{ts}} = L(|q|_2)$ where

$$L(n) = 4.7 \times 10^{-5} \exp(1.9229(n \ln 2)^{1/3}(\ln(n \ln 2))^{2/3})$$

(see [13, Section 2.4]). These arguments imply the following assumption which is adopted in our numerical examples given later:

$$\mathsf{G}_{\mathrm{DDH}} \text{ is } R(t)\text{-secure with respect to } \mathsf{comp} \text{ with } R(t) = k_0 t/L(|q|_2). \tag{7}$$

By the above definition, the seeds and outputs of $\mathsf{G} = \mathsf{G}_{\mathrm{DDH}}$ are sequences of finite field elements rather than bit sequences. For the purpose of our discussion, we try to convert them into bit sequences. First we give some notations. For integer parameters h_1 and h_2, define two maps $\gamma : \Sigma^{3h_1} \to (\mathbb{Z}_q)^3 = S_{\mathsf{G}}$ and $\gamma' : O_{\mathsf{G}} = (\mathbb{Z}_q)^{k_0} \to \Sigma^{k_0 h_2}$ by

$$\gamma(s_1, s_2, s_3) = (\gamma_0(s_1), \gamma_0(s_2), \gamma_0(s_3)) \, , \, \gamma'(s_1, \ldots, s_{k_0}) = (\gamma'_0(s_1), \ldots, \gamma'_0(s_{k_0}))$$

where $\gamma_0 : \Sigma^{h_1} \to \mathbb{Z}_q$ and $\gamma'_0 : \mathbb{Z}_q \to \Sigma^{h_2}$ are defined by

$$\gamma_0(x) = (x \bmod q) + 1 \text{ and } \gamma'_0(x) = (x \bmod 2^{h_2})$$

and we let $(x \bmod n) \in \{0, 1, \ldots, n-1\}$. Then the following property holds:

Lemma 4.1. *We have*

$$\mathsf{SD}(\gamma(U_{\Sigma^{3h_1}}), U_{(\mathbb{Z}_q)^3}) \leq 3f(2^{h_1}, q) \, , \, \mathsf{SD}(U_{\Sigma^{k_0 h_2}}, \gamma'(U_{(\mathbb{Z}_q)^{k_0}})) \leq k_0 f(q, 2^{h_2}) \ ,$$

where

$$f(z_1, z_2) = \frac{(z_1 \bmod z_2) \cdot (z_2 - (z_1 \bmod z_2))}{z_1 z_2} \ .$$

Proof. First note that, if P_i and P'_i are random variables on the same set for each $i \in \{1, 2\}$, P_1 and P_2 are independent, and P'_1 and P'_2 are independent, then we have

$$\mathsf{SD}(P_1 \times P_2, P'_1 \times P'_2) \leq \mathsf{SD}(P_1, P'_1) + \mathsf{SD}(P_2, P'_2) \ .$$

Owing to this fact, it suffices to show that

$$\mathsf{SD}(\gamma_0(U_{\Sigma^{h_1}}), U_{\mathbb{Z}_q}) = f(2^{h_1}, q) \, , \, \mathsf{SD}(U_{\Sigma^{h_2}}, \gamma'_0(U_{\mathbb{Z}_q})) = f(q, 2^{h_2}) \ .$$

For the former equality, write $2^{h_1} = aq + b$ with $b = (2^{h_1} \bmod q)$. Then we have $|\gamma_0^{-1}(x)| = a + 1$ for b out of the q elements $x \in \mathbb{Z}_q$, while $|\gamma_0^{-1}(x)| = a$ for the remaining $q - b$ elements $x \in \mathbb{Z}_q$. This implies that

$$\begin{aligned} \mathsf{SD}(\gamma_0(U_{\Sigma^{h_1}}), U_{\mathbb{Z}_q}) &= \frac{1}{2} \cdot \left(b \left| \frac{a+1}{aq+b} - \frac{1}{q} \right| + (q-b) \left| \frac{a}{aq+b} - \frac{1}{q} \right| \right) \\ &= \frac{1}{2} \cdot \left(b \cdot \frac{q-b}{q(aq+b)} + (q-b) \frac{b}{q(aq+b)} \right) = \frac{b(q-b)}{2^{h_1} q} = f(2^{h_1}, q) \ . \end{aligned}$$

The latter equality is similarly proven. Hence Lemma 4.1 holds. $\square$

Let $\mathsf{G}' = \mathsf{G}'_{\mathrm{DDH}}$ denote the composition $\gamma' \circ \mathsf{G}$ of $\mathsf{G} = \mathsf{G}_{\mathrm{DDH}}$ followed by γ', which is also a PRG with seed set $S_{\mathsf{G}'} = S_{\mathsf{G}} = (\mathbb{Z}_q)^3$ and output set $O_{\mathsf{G}'} = \Sigma^{k_0 h_2}$. Note that the map γ' just outputs some lower bits of the original output of G, therefore the issue of complexity of γ' may be ignored for simplicity in practical situations. Then Lemma 4.1 and the assumption (7) imply (by ignoring complexity of γ') that the PRG G' is $R'(t)$-secure with respect to the same comp, where

$$R'(t) = k_0 \left(\frac{t}{L(|q|_2)} + f(q, 2^{h_2}) \right) \quad . \tag{8}$$

4.2 Collusion-Secure Codes

In our numerical example, we choose collusion-secure codes [5] as an example of existing information-theoretically secure cryptographic schemes. In this article, we formulate a *collusion-secure code* as a pair $(\mathsf{Gen}, \mathsf{Tr})$ of *codeword generation algorithm* Gen and *tracing algorithm* Tr, introduced in the following context. First, a *provider* executes Gen to generate *secret information* $s \in S$ that consists of N codewords corresponding to the N *users* $1, 2, \dots, N$ and some (possibly empty) auxiliary element which we refer to as a *state element*. We suppose that the codewords are binary and of common length m. Then the provider distributes each codeword to the corresponding user. Let $C \subset \{1, \dots, N\}$ denote the unknown coalition of adversarial users called *pirates*. Let Dist denote the map (or algorithm) that associates to s the collection $w = \mathsf{Dist}(s) \in W$ of all codewords for the pirates. The pirates execute an attack algorithm At to generate from w a *pirated word* $y = \mathsf{At}(w) \in Y$ that is a word of length m over an extended alphabet $\{0, 1, ?\}$, where '?' denotes an "erasure symbol". The only assumption on At is the standard one called *Marking Assumption* [5], which does *not* restrict the complexity of At. Finally, the provider receives the pirated word y and executes Tr, with input (y, s), to output a (possibly empty) set $a = \mathsf{Tr}(y, s) \in A$ of accused users. Then we consider that the attack "succeeded" if a contains no pirates or at least one innocent (i.e., non-pirate) user and "failed" otherwise; let $\mathsf{Ev} : A \to \Sigma$ denote the corresponding evaluation algorithm. Fig. 4 shows a flowchart $\mathcal{F}$ of the above process, where X denotes the set of random elements used in the algorithm Gen regarded as the input of Gen. (Note that Gen and Tr may vary with respect to some security parameters or other parameters such as the total number N of users, though such details are ignored in the picture.) Hence the attack success probability succ is given by the probability that the corresponding algorithm $\mathsf{A}(\mathcal{F})$ outputs 1.

$$\begin{tikzcd} X \arrow[r, "\mathsf{Gen}"] & S \arrow[r, "\mathsf{Dist}"] \arrow[rrr, bend left, no head] & W \arrow[r, "\mathsf{At}"] & Y \arrow[r, "\mathsf{Tr}"] & A \arrow[r, "\mathsf{Ev}"] & \Sigma \end{tikzcd}$$

Figure 4: Flowchart for collusion-secure codes (the circled arrow signifies an algorithm without bound of complexity)

In our example, we deal with a concrete scheme [20] of collusion-secure codes that is an improvement of the famous Tardos code [27]. A main reason of considering the scheme in [20] rather than Tardos code is that the finite probability distribution used in [20] is much simpler than the continuous distribution in Tardos code, which simplifies our argument. Now a state element consists of m random values $0 < p_j < 1$, $1 \le j \le m$, each being independently generated according to the common probability distribution $\mathcal{P}$ specified below. The algorithm Gen first generates the state element. Then it generates each, say, j-th bit $w_{i,j}$ of i-th user's codeword w_i independently by $Pr[w_{i,j} = 1] = p_j$ and $Pr[w_{i,j} = 0] = 1 - p_j$. On the other hand, the algorithm Tr first calculates the score $\mathsf{sc}_i = \sum_{j=1}^{m} \mathsf{sc}_{i,j}$ of i-th user, where the bit-wise score $\mathsf{sc}_{i,j}$ for j-th bit is a function of y_j, $w_{i,j}$, and p_j specified below. Then Tr outputs (any one of) the user(s) with highest score. Hence the output $a \in A$ is now a single user rather than a set of users.

We describe details of the choices of probability distributions $\mathcal{P}$ and scoring functions in [20]. From now, we consider only the case of three pirates ($|C| = c = 3$) for simplicity. First, let the probability distribution $\mathcal{P}$ take one of the two values $p^{(0)}$ and $p^{(1)}$ with equal probability $1/2$, where

$$p^{(0)} = 0.211334228515625 = (0.001101100001101)_2 \, , \; p^{(1)} = 1 - p^{(0)} \quad ,$$

which are approximations of values of the probability distribution given in [20, Definition 4] with approximation error less than 10^{-5} (here we require $p^{(0)}$ and $p^{(1)}$ to have short binary expressions rather than short decimal expressions; the same also holds for values u_0 and u_1 below). Secondly, for the scoring function, we define two auxiliary values u_0 and u_1 by

$$u_0 = 1.931793212890625 = (1.111011101000101)_2 \, , \, u_1 = 0.5176544189453125 = (0.1000010010000101)_2 \ ,$$

and define the bitwise score $\mathsf{sc}_{i,j}$ for j-th bit of i-th user in the following manner: If $p_j = p^{(\nu)}$, $\nu \in \{0,1\}$, then put

$$\mathsf{sc}_{i,j} = \begin{cases} u_\nu & \text{if } y_j = 1 \text{ and } w_{i,j} = 1 \ , \\ -u_{1-\nu} & \text{if } y_j = 1 \text{ and } w_{i,j} = 0 \ , \\ -u_\nu & \text{if } y_j \neq 1 \text{ and } w_{i,j} = 1 \ , \\ u_{1-\nu} & \text{if } y_j \neq 1 \text{ and } w_{i,j} = 0 \ . \end{cases}$$

These u_0 and u_1 are approximations of Tardos's scoring function $\sqrt{(1-x)/x}$ (which is also used in [20]) at $x = p^{(0)}$ and $x = p^{(1)}$, respectively, with approximation error $\Delta < 4.2 \times 10^{-6} < 10^{-5}$. We notice that effects of such approximation errors are already considered in the security proof of [20].

In our numerical example, we consider the case that the attack success probability $\mathsf{succ} = \mathsf{succ}_{\mathrm{rnd}}$, which is now the probability that the output a of Tr is not a pirate, is bounded by $\varepsilon = 10^{-3}$ when an input $x \in X$ for Gen is chosen uniformly at random. We vary the number N of users as $N = 10^3, 10^4, \dots, 10^9$. Then by the results of the first part of [20, Theorem 1], we can calculate the code lengths for these cases as in Table 1, where we used auxiliary values $\Delta = 4.2 \times 10^{-6}$, $\eta = 1.93180$, $\mathcal{R} = 0.40822$, and $\beta = 0.0613461$ in the calculation (see [20] for details of those auxiliary values).

Table 1: Code lengths of collusion-secure codes in [20] with $c = 3$ and $\varepsilon = 10^{-3}$

user number N	10^3	10^4	10^5	10^6	10^7	10^8	10^9
code length m	614	702	789	877	964	1052	1139

5 Numerical Examples and Further Improvement

In this section, we present numerical examples of our result by using the objects summarized in Section 4. Moreover, we also introduce a novel technique for PRG-based randomness reduction to improve the indistinguishability, by showing concrete examples.

5.1 An Improved Technique

The above observation for our main theorem shows that the indistinguishability between random and pseudorandom cases would be improved if the amount of variation of information received by the adversary is diminished. Therefore, if we can divide the randomness used in a protocol into several pieces in such a way that only a smaller component of the information received by the adversary depends on each piece of the randomness, and we use an *independent* PRG to generate each of the pieces, then replacement of each perfectly random piece with pseudorandom one would be more indistinguishable than the original situation. Now by using the "hybrid argument", the total indistinguishability between fully random and fully pseudorandom cases would be improved as well. In this subsection, we explain this idea further by applying it to a concrete scheme of collusion-secure codes [20] mentioned in Section 4.2.

To apply our idea, first we divide the set $\{1, 2, \dots, m\}$ of bit positions in the codewords into ℓ parts $I_1, I_2, \dots, I_\ell$. An important property of the scheme [20] is that only the j-th random value p_j among $p_1, \dots, p_m$ is used for generating j-th bit $w_{i,j}$ of each codeword w_i. Therefore we can also divide the set X of random input for Gen into ℓ pieces $X_1, \dots, X_\ell$ in such a way that the random values p_j and the bits $w_{i,j}$

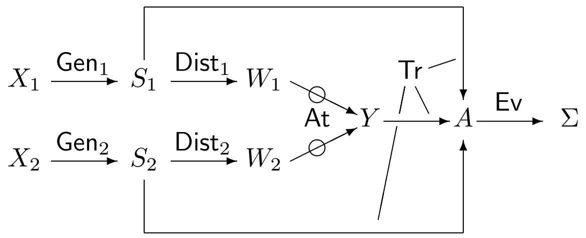

Figure 5: Modified flowchart for collusion-secure codes, with $\ell = 2$ (the circled arrows signify an algorithm without bound of complexity)

with $j \in I_\nu$ are generated by using the ν-th piece X_ν. The flowchart $\mathcal{F}$ of this modified situation is shown in Fig. 5 (we present the picture only for the case $\ell = 2$ for simplicity, but a more general case is analogous). Here the ν-th part $s_\nu \in S_\nu$ of the secret information (consisting of p_j and $w_{i,j}$ with $j \in I_\nu$) is generated by the algorithm Gen_ν with random input $x_\nu \in X_\nu$, and the ν-th part $\mathsf{Dist}_\nu(s_\nu) \in W_\nu$ of the pirates' codewords obviously depends solely on s_ν.

Since this new setting includes the original setting as the special case $\ell = 1$, in what follows we only discuss the generalized setting with an arbitrary $\ell \geq 1$. More precisely, we would like to compare the following two cases: The input x_ν for Gen_ν is generated by the uniform random variable U_{X_ν} for every ν (the "fully random" case); and x_ν is generated by an independent PRG $\mathsf{G}^\nu : S^\nu \to O^\nu$ with uniformly random seed for every ν, where $O^\nu = X_\nu$ (the "fully pseudorandom" case). Now for $0 \leq \nu \leq \ell$ and $1 \leq \mu \leq \ell$, let r_ν^μ be a random variable on X_μ such that $r_\nu^\mu = \mathsf{G}^\mu(U_{S^\mu})$ if $\mu \leq \nu$ and $r_\nu^\mu = U_{O^\mu}$ if $\mu > \nu$, and put $\mathcal{R}_\nu = (r_\nu^\mu)_{1 \leq \mu \leq \ell}$. Hence $\mathcal{R}_0$ and $\mathcal{R}_\ell$ correspond to fully random and fully pseudorandom cases, respectively. By the hybrid argument, the difference between fully random and fully pseudorandom cases is bounded by the sum of differences between the cases of $\mathcal{R}_{\nu-1}$ and $\mathcal{R}_\nu$ over $1 \leq \nu \leq \ell$, and $\mathcal{R}_{\nu-1}$ and $\mathcal{R}_\nu$ differ only at the ν-th components $r_{\nu-1}^\nu = U_{O^\nu}$ and $r_\nu^\nu = \mathsf{G}^\nu(U_{S^\nu})$. Hence it suffices to evaluate the indistinguishability for randomness reduction of each randomness piece X_ν.

For the purpose, we apply Theorem 3.1 to the above flowchart $\mathcal{F}$ by setting $v_0 = X_\nu$ and $\mathcal{U} = \{Y\}$. Then we have

$$\mathcal{V}' = \{X_\nu, S_\nu, W_\nu, A, \Sigma\}, \mathcal{U}' = W_\nu, \mathcal{U}'' = \{S_1, \dots, S_{\nu-1}, S_{\nu+1}, \dots, S_\ell, Y\} .$$

Put $\neg\nu = \{1, \dots, \ell\} \setminus \{\nu\}$. Given elements $w^\nu \in W_\nu$, $s_\mu \in S_\mu$ (with $\mu \in \neg\nu$) and $y \in Y$, the corresponding auxiliary flowchart $\widetilde{\mathcal{F}} = \widetilde{\mathcal{F}}_\nu$ is as shown in Fig. 6. Now assume that each PRG G^ν is $R_\nu(t)$-secure with respect to a common complexity measure comp. Assume further that the complexity $\mathsf{comp}(\mathsf{A}(\widetilde{\mathcal{F}}_\nu))$ is bounded by a constant $T_\nu > 0$. Then by applying Theorem 3.1, we have

$$|Pr[1 \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_{\nu-1})] - Pr[1 \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_\nu)]| \leq |W_\nu| R_\nu(T_\nu)$$

for each ν, therefore the difference between the attack success probabilities $\mathsf{succ}_{\mathrm{rnd}}$ and $\mathsf{succ}_{\mathrm{prnd}}$ in fully random and fully pseudorandom cases, respectively, is bounded by

$$\begin{aligned} |\mathsf{succ}_{\mathrm{rnd}} - \mathsf{succ}_{\mathrm{prnd}}| &= |Pr[1 \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_0)] - Pr[1 \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_\ell)]| \\ &\leq \sum_{\nu=1}^{\ell} |Pr[1 \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_{\nu-1})] - Pr[1 \leftarrow \mathsf{A}(\mathcal{F})(\mathcal{R}_\nu)]| \leq \sum_{\nu=1}^{\ell} |W_\nu| \cdot R_\nu(T_\nu) \ . \end{aligned} \tag{9}$$

5.2 Numerical Examples

From now, we present numerical examples of the bound in (9) by using the objects and data in Section 4. For the purpose, it is required to estimate the complexity of the algorithms $\mathsf{A}(\widetilde{\mathcal{F}}_\nu)$ with respect to the complexity measure comp introduced in Section 4.1. For simplicity, we choose the partition $I_1, \dots, I_\ell$ of bit

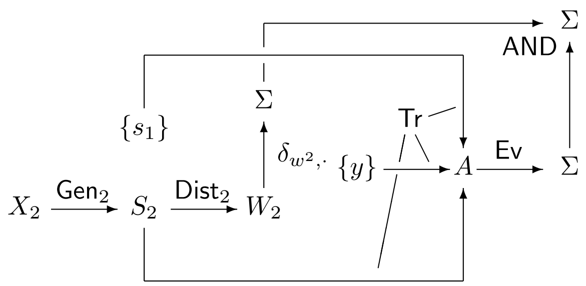

Figure 6: Auxiliary flowchart $\widetilde{\mathcal{F}} = \widetilde{\mathcal{F}}_\nu$ corresponding to Fig. 5, with $\nu = 2$

positions $\{1, \dots, m\}$ such that $I_\nu = \{j \mid \overline{m}_{\nu-1} + 1 \le j \le \overline{m}_\nu\}$, where $m_\nu = |I_\nu|$ and $\overline{m}_\nu = \sum_{\mu=1}^{\nu} m_\mu$ (hence $\overline{m}_0 = 0$ and $\overline{m}_\ell = m$). Let $1 \le i_1 < i_2 < i_3 \le N$ be the three pirates. Then we give a pseudo-program for $\mathsf{A}(\widetilde{\mathcal{F}}_\nu)$. Here we encode each digit y_j of $y \in Y$ in such a way that 2-bit sequences 00, 01, and 10 represent '0', '1' and '?', respectively (hence one can determine whether $y_j = 1$ or not by just one bit comparison at the lowest bit). The element $w^\nu \in W_\nu$ consists of the bits $w^\nu_{i,j} \in \Sigma$ with $i \in \{i_1, i_2, i_3\}$ and $j \in I_\nu$. For each $\mu \in \neg\nu$, the element $s_\mu \in S_\mu$ consists of the values p_j ($j \in I_\mu$) and bits $w_{i,j}$ ($1 \le i \le N$, $j \in I_\mu$). Since each p_j is chosen from the two values $p^{(0)}$ and $p^{(1)}$ given in Section 4.2, here we encode each p_j into $\xi \in \Sigma$ such that $p_j = p^{(\xi)}$. We also use the values u_0 and u_1 given in Section 4.2. In this setting, we describe a pseudo-program for $\mathsf{A}(\widetilde{\mathcal{F}}_\nu)$ together with an estimate of its complexity (see below for details) as follows, where `next_`$n(x_\nu)$ denotes an operation to load the next n bits from the input bit sequence x_ν (the subscript 'n' is omitted in the case $n = 1$), sc_0 denotes the constant $-mu_0$, and the values $w^\nu_{i,j}$ ($i \in \{i_1, i_2, i_3\}$, $j \in I_\nu$), p_j ($j \in \bigcup_{\mu \in \neg\nu} I_\mu$) and $w_{i,j}$ ($1 \le i \le N$, $j \in \bigcup_{\mu \in \neg\nu} I_\mu$) are given:

```
Input: x_ν ∈ X_ν     Output: 0 or 1
01: for j in m̄_{ν−1}+1,...,m̄_ν do {
02:  set p_j := next(x_ν)   \\ 1 TU
03: }  \\ 3m_ν + 2 TUs for 01 - 03
04: for i in 1,...,N do {
05:  for j in m̄_{ν−1}+1,...,m̄_ν do {
06:   if next_15(x_ν) < p^(0) then {
07:    set w_{i,j} := 1-p_j   \\ 2 TUs
08:   } else {
09:    set w_{i,j} := p_j   \\ 1 TU
10:   }  \\ 3 TUs for 06 - 10
11:   if i = i_1 or i = i_2 or i = i_3 then {
12:    if not w_{i,j} = w^ν_{i,j} then {
13:     return 0
14:    }  \\ 1 TU for 12 - 14
15:   }  \\ 4 TUs for 11 - 15
16:  }  \\ 9m_ν + 2 TUs for 05 - 16
17: }  \\ (9m_ν + 4)N + 2 TUs for 04 - 17
18: set sc_max := sc_0   \\ 1 TU
19: for i in 1,...,N do {
20:  set sc := 0   \\ 1 TU
21:  for j in 1,...,m do {
22:   if y_j = 1 then {
23:    if w_{i,j} = 1 then {
24:     if p_j = 0 then {
25:      set sc := sc + u_0   \\ 1 TU
```

```
26:      } else {
27:       set sc := sc + u_1   \\ 1 TU
28:      }  \\ 2 TUs for 24 - 28
29:     } else {
30:      if p_j = 0 then {
31:       set sc := sc - u_1   \\ 1 TU
32:      } else {
33:       set sc := sc - u_0   \\ 1 TU
34:      }  \\ 2 TUs for 30 - 34
35:     }  \\ 3 TUs for 23 - 35
36:    } else {
37:     if w_{i,j} = 0 then {
38:      if p_j = 0 then {
39:       set sc := sc + u_1   \\ 1 TU
40:      } else {
41:       set sc := sc + u_0   \\ 1 TU
42:      }  \\ 2 TUs for 38 - 42
43:     } else {
44:      if p_j = 0 then {
45:       set sc := sc - u_0   \\ 1 TU
46:      } else {
47:       set sc := sc - u_1   \\ 1 TU
48:      }  \\ 2 TUs for 44 - 48
49:     }  \\ 3 TUs for 37 - 49
50:    }  \\ 4 TUs for 22 - 50
51:   }  \\ 6m + 2 TUs for 21 - 51
52:   if not sc < sc_max then {
53:    set sc_max := sc, a := i \\ 2 TUs
54:   }  \\ 3 TUs for 52 - 54
55: }  \\ (6m + 8)N + 2 TUs for 19 - 55
56: if a = i_1 or a = i_2 or a = i_3 then {
57:  return 0
58: }  \\ 3 TUs for 56 - 58
59: return 1
```

Recall from Section 4.1 that our complexity measure comp is defined in terms of the worst-case running time on a computer used by the work [17]. Since it is infeasible to determine the precise running time, in the above estimate we approximated the worst-case running time according to the following two rules. First, we regard each operation of substitution, addition, subtraction, and comparison as taking 1 time unit (in the above description, "TU" stands for "time unit") that is approximately the time for 1 DES encryption. This first rule would be justified since, for the current choice of parameters, every such operation in the above pseudo-program is either an operation between fixed-point numbers with just 12-bit or shorter integer parts and just 16-bit or shorter fractional parts, or an operation between just 30-bit or shorter integers, which would be much more efficient than DES encryption (in fact, this is likely to be overestimation, but it does not cause any serious problem since we need only an upper bound of the complexity). Secondly, we ignore the complexity of operations of loading a next bit from the input (i.e., an operation next_$n(x_\nu)$), outputting an element (i.e., an operation return), and jumping in the execution flow (implicitly used in for loops and if statements), which (together with any other missed issue on complexity) seem negligibly small and would be absorbed by the above-mentioned overestimation. From the two rules, it follows that the worst-case running time of a for loop of the form "for CN in ST,...,EN do $\mathsf{JOB}_{\mathsf{CN}}$ end for" is (over)estimated to be the sum of $2(\mathsf{EN} - \mathsf{ST} + 2)$ time units (composed of 1 initialization of the counter CN, $\mathsf{EN} - \mathsf{ST} + 1$ increments for CN, and $\mathsf{EN} - \mathsf{ST} + 2$ checks for the terminating condition) and the sum of running times of $\mathsf{JOB}_{\mathsf{CN}}$ for all $\mathsf{ST} \le \mathsf{CN} \le \mathsf{EN}$. In particular, if the running time of $\mathsf{JOB}_{\mathsf{CN}}$ is constantly equal to T time units, then the estimated running time of this loop is $(\mathsf{EN} - \mathsf{ST} + 1)(\mathsf{T} + 2) + 2$ time units. The above estimates of running times of each line, each `for` loop and each `if` statement are thus obtained. By summing the estimated running times presented at lines `03`, `17`, `18`, `55`, and `58`, we have $\mathsf{comp}(\mathsf{A}(\widetilde{\mathcal{F}}_\nu)) \le T_\nu$ where

$$T_\nu = (3m_\nu + 2) + ((9m_\nu + 4)N + 2) + 1 + ((6m + 8)N + 2) + 3 = (6m + 9m_\nu + 12)N + 3m_\nu + 10 \ .$$

Since $|W_\nu| = |(\Sigma^{m_\nu})^3| = 2^{3m_\nu}$, by using the value of T_ν and the bound in (9), we have

$$|\mathsf{succ}_{\mathrm{rnd}} - \mathsf{succ}_{\mathrm{prnd}}| \le \sum_{\nu=1}^{\ell} 2^{3m_\nu} R_\nu(T_\nu) \ . \tag{10}$$

To simplify the argument, we choose the sizes m_ν of the partitions I_ν such that $|m_\nu - m/\ell| < 1$. Let each PRG G^ν be a copy of the PRG $\mathsf{G}'_{\mathrm{DDH}}$ introduced in the final paragraph of Section 4.1, therefore we have $R_\nu(t) = R'(t)$ (see (8) for definition of $R'(t)$). Then we have $m_\nu \le \lceil m/\ell \rceil$, and it follows from (10) that

$$\begin{aligned} |\mathsf{succ}_{\mathrm{rnd}} - \mathsf{succ}_{\mathrm{prnd}}| &\le \sum_{\nu=1}^{\ell} 2^{3\lceil m/\ell \rceil} k_0 \left(\frac{T_\nu}{L(|q|_2)} + f(q, 2^{h_2}) \right) \\ &= 2^{3\lceil m/\ell \rceil} k_0 \cdot \left(\frac{(6\ell m + 9m + 12\ell)N + 3m + 10\ell}{L(|q|_2)} + \ell f(q, 2^{h_2}) \right) \ . \end{aligned} \tag{11}$$

On the other hand, by the above pseudo-program, the necessary and sufficient bit length of the input x_ν is $(15N + 1)m_\nu$. Hence the total number of required random bits in fully random case is $(15N + 1)m$, and the parameters k_0 and h_2 for $\mathsf{G}'_{\mathrm{DDH}}$ should satisfy $k_0 h_2 \ge (15N + 1)\lceil m/\ell \rceil$. For simplicity, we suppose that the integer k_0 is as small as possible, namely we set $k_0 = \lceil (15N + 1)\lceil m/\ell \rceil / h_2 \rceil$.

Since the bound of attack success probability for fully random case is set to $\varepsilon = 10^{-3}$, the difference of the probabilities between fully random and fully pseudorandom cases should be significantly smaller than 10^{-3} to make the fully pseudorandom case secure as well. In the numerical example below, we require the right-hand side of (11) to be smaller than 10^{-6}. Moreover, since the PRG $\mathsf{G}' = \mathsf{G}'_{\mathrm{DDH}}$ uses originally non-binary seeds chosen from $S_{\mathsf{G}'} = (\mathbb{Z}_q)^3$, we approximate the seeds of each $\mathsf{G}^\nu = \mathsf{G}'$ by outputs of the map $\gamma : \Sigma^{3h_1} \to (\mathbb{Z}_q)^3$ introduced in Section 4.1 with uniformly random inputs, therefore the new "seed set" is $(\Sigma^{3h_1})^\ell = \Sigma^{3\ell h_1}$. By Lemma 4.1, the statistical distance between the distribution over $(\mathbb{Z}_q)^{3\ell}$ induced by outputs of γ and the uniform distribution is bounded by $3\ell f(2^{h_1}, q)$, therefore we also require the value $3\ell f(2^{h_1}, q)$ to be smaller than 10^{-6}.

We determine the total seed lengths $3\ell h_1$ and other parameters in fully pseudorandom cases under the above conditions. Table 2 shows the results of calculation for three cases $\ell \in \{1, 2, 5\}$. In the table, "difference" signifies the sum of $3\ell f(2^{h_1}, q)$ and the value in the right-hand side of (11), which should be smaller than 10^{-6}, and "ratio" signifies the ratio of the seed length $3\ell h_1$ in fully pseudorandom case to the number of required random bits in fully random case. For each case in the table where the choice of Sophie-Germain prime q is specified, we used the following values:

$$\begin{aligned} &q_{(1)} = 790717071 \times 2^{54254} - 1 \ , \ q_{(2)} = 2566851867 \times 2^{70001} - 1 \ , \ q_{(3)} = 18912879 \times 2^{98395} - 1 \ , \\ &q_{(4)} = 7068555 \times 2^{121301} - 1 \ , \ q_{(5)} = 137211941292195 \times 2^{171960} - 1 \ , \end{aligned}$$

where the last four Sophie-Germain primes are quoted from the July 2009 version of a list by Caldwell [7], while the first one is quoted from the September 2008 version of that list. On the other hand, for each of the remaining cases, an approximation of q was performed since the authors could not find in the literature a concrete Sophie-Germain prime with appropriate size. In such a case, we calculated the "difference" and the corresponding total seed length under the assumption that both $f(2^{h_1}, q)$ and $f(q, 2^{h_2})$ vanish and $h_1 = h_2 = |q|_2$. This approximation would be allowable, since h_1 and h_2 are not significantly far from q in the five cases with precise values of q.

Table 2: Numerical examples of effects of randomness reduction

user number N		10^3	10^4	10^5	10^6	10^7	10^8	10^9
code length m		614	702	789	877	964	1052	1139
# of random bits		9.21×10^6	1.05×10^8	1.18×10^9	1.31×10^{10}	1.44×10^{11}	1.57×10^{12}	1.70×10^{13}
$\ell = 1$	q	—	—	—	—	—	—	—
	$\lvert q\rvert_2$	2.29×10^6	3.24×10^6	4.41×10^6	5.82×10^6	7.47×10^6	9.41×10^6	1.17×10^7
	h_2							
	h_1							
	difference	1.48×10^{-7}	6.69×10^{-7}	2.63×10^{-7}	5.03×10^{-7}	5.81×10^{-7}	7.40×10^{-7}	1.15×10^{-9}
	seed length	6.87×10^6	9.72×10^6	1.33×10^7	1.75×10^7	2.25×10^7	2.83×10^7	3.51×10^7
	ratio	7.46×10^{-1}	9.26×10^{-2}	1.13×10^{-2}	1.34×10^{-3}	1.57×10^{-4}	1.81×10^{-5}	2.07×10^{-6}
$\ell = 2$	q	—	—	—	—	—	—	—
	$\lvert q\rvert_2$	4.07×10^5	5.73×10^5	7.76×10^5	1.02×10^6	1.30×10^6	1.63×10^6	2.01×10^6
	h_2							
	h_1							
	difference	9.57×10^{-7}	8.66×10^{-7}	8.09×10^{-7}	5.15×10^{-7}	3.88×10^{-7}	4.43×10^{-7}	3.28×10^{-7}
	seed length	2.45×10^6	3.44×10^6	4.66×10^6	6.12×10^6	7.80×10^6	9.78×10^6	1.21×10^7
	ratio	2.67×10^{-1}	3.28×10^{-2}	3.95×10^{-3}	4.68×10^{-4}	5.42×10^{-5}	6.23×10^{-6}	7.12×10^{-7}
$\ell = 5$	q	$q_{(1)}$	$q_{(2)}$	$q_{(3)}$	$q_{(4)}$	$q_{(5)}$	—	—
	$\lvert q\rvert_2$	$54,284$	$70,033$	$98,420$	$121,324$	$172,007$	1.90×10^5	2.30×10^5
	h_2	$54,254$	$70,001$	$98,395$	$121,301$	$171,960$		
	h_1	$54,306$	$70,056$	$98,441$	$121,347$	$172,029$		
	difference	4.56×10^{-7}	8.24×10^{-7}	9.67×10^{-7}	3.66×10^{-7}	4.78×10^{-7}	4.39×10^{-7}	9.57×10^{-7}
	seed length	8.15×10^5	1.06×10^6	1.48×10^6	1.83×10^6	2.59×10^6	2.84×10^6	3.45×10^6
	ratio	8.85×10^{-2}	1.01×10^{-2}	1.26×10^{-3}	1.40×10^{-4}	1.80×10^{-5}	1.81×10^{-6}	2.03×10^{-7}

In Table 2, for every choice of ℓ, the ratio of the seed length to the original number of required random bits decreases (namely, the effect of randomness reduction improves) as the number N of users increases. More precisely, the original numbers of required random bits are almost linear in N, while the seed lengths are almost independent of the values of N. This can be interpreted as that the amount of required randomness "inessential" for the security increases as the number of users increases; see the fourth remark in Section 3.4.

In the table, for each choice of user number N and code length m, the ratio is significantly low already in the case $\ell = 1$, i.e., when the improved technique presented in Section 5.1 is not applied. This shows that even the plain PRG-based randomness reduction can be effective for information-theoretically secure cryptographic schemes, by using our indistinguishability evaluation technique.

Moreover, in the table the ratios for the cases $\ell = 2, 5$ are significantly better than the plain case $\ell = 1$. Note that the ratios for the case $\ell = 5$ are better than the case $\ell = 2$ further. Also, Fig. 7 shows a relation between the value ℓ and the approximated total seed length in the case $N = 10^3$ (written in scientific E notation), where the approximation was performed by the same rule as above. (By the above observation, the overall tendency would be similar for the other choices of N.) In the graph, the approximated seed length takes the minimum value $236,220$ at the case $\ell = 31$, which is approximately 2.57% of the original number of required random bits (this ratio would be further improved in the case of larger N) and is about 29 times as short as the plain case $\ell = 1$. These results show that our improved technique in Section 5.1 indeed works effectively. We also notice that, as a by-product, our technique in Section 5.1 reduces the computational cost of the PRGs as well, since the sizes of the Sophie-Germain primes q used in the PRGs are significantly

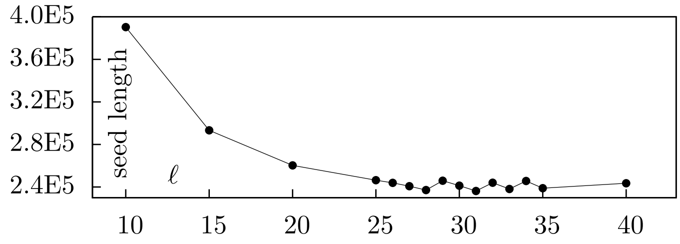

Figure 7: Values of ℓ and approximated seed lengths for the example, with $N = 10^3$

decreased as ℓ increases.

6 Conclusion

In this article, we proposed novel ideas and techniques for evaluation of indistinguishability between random and pseudorandom cases in PRG-based randomness reduction of cryptographic schemes. Our evaluation technique can prove the indistinguishability even against an adversary with computationally unbounded attack algorithm, especially when the amount of information received by the adversary is small, hence it reveals that PRG-based randomness reduction can be effective for not only computationally secure but also information-theoretically secure schemes. In comparison to a preceding result of Dubrov and Ishai [11], our result removes the requirement of the generalized notion of nb-PRGs and is effective for more general kinds of protocols. We presented the effectiveness of our result by giving numerical examples of randomness reduction for collusion-secure codes. Moreover, we also proposed another idea of dividing the required randomness into several smaller pieces for improving the effect of randomness reduction, and presented numerical examples to show that the idea also works effectively.

Acknowledgment

This article is the mergence of two preliminary versions of this work [21, 22] presented at The 4th International Conference on Information Theoretic Security (ICITS 2009), Shizuoka, Japan, December 3–6, 2009. The authors would like to thank Hideki Imai, Hajime Watanabe, Kentaro Imafuku, Takayuki Miyadera, Gen Kimura, and the anonymous referees of ICITS 2009, for their precious comments.

References

[1] E. Bach, "Realistic analysis of some randomized algorithms," *J. Comput. Syst. Sci.,* vol. 42, pp. 30–53, 1991.

[2] O. Billet and D. H. Phan, "Efficient traitor tracing from collusion secure codes," in *Proc. ICITS 2008,* pp. 171–182.

[3] I. Binder and M. Braverman, "Derandomization of Euclidean random walks," in *Proc. APPROX–RANDOM 2007,* pp. 353–365.

[4] G. R. Blakley, "Safeguarding cryptographic keys," in *Proc. 1979 National Computer Conference,* pp. 313–317.

[5] D. Boneh and J. Shaw, "Collusion-secure fingerprinting for digital data," *IEEE Trans. Inform. Theory,* vol. 44, pp. 1897–1905, 1998.

[6] C. Bosley and Y. Dodis, "Does privacy require true randomness?" in *Proc. TCC 2007,* pp. 1–20.

[7] C. Caldwell. (July, 2009). The Top Twenty: Sophie Germain (p). [Online]. Available: `http://primes.utm.edu/top20/page.php?id=2`

[8] Q. Cheng, "Derandomization of sparse cyclotomic integer zero testing," in *Proc. 48th FOCS,* 2007, pp. 74–80.

[9] B. Chor, A. Fiat and M. Naor, "Tracing traitors," in *Proc. CRYPTO 1994,* pp. 257–270.

[10] B. Chor and O. Goldreich, "Unbiased bits from sources of weak randomness and probabilistic communication complexity," *SIAM J. Comput.,* vol. 17(2), pp. 230–261, 1988.

[11] B. Dubrov and Y. Ishai, "On the randomness complexity of efficient sampling," in *Proc. STOC 2006,* pp. 711–720.

[12] Y. Dodis, S. J. Ong, M. Prabhakaran and A. Sahai, "On the (im)possibility of cryptography with imperfect randomness," in *Proc. 45th FOCS,* 2004, pp. 196–205.

[13] R. R. Farashahi, B. Schoenmakers and A. Sidorenko, "Efficient pseudorandom generators based on the DDH assumption," in *Proc. PKC 2007,* pp. 426–441.

[14] A. Fiat and M. Naor, "Broadcast encryption," in *Proc. CRYPTO 1993,* pp. 480–491.

[15] E. Kaplan, M. Naor and O. Reingold, "Derandomized constructions of k-wise (almost) independent permutations," in *Proc. APPROX–RANDOM 2005,* pp. 354–365.

[16] A. Kiayias and M. Yung, "Traitor tracing with constant transmission rate," in *Proc. EUROCRYPT 2002,* pp. 450–465.

[17] A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes," *J. Cryptology,* vol. 14, pp. 255–293, 2001.

[18] U. Maurer and S. Wolf, "Privacy amplification secure against active adversaries," in *Proc. CRYPTO 1997,* pp. 307–321.

[19] J. L. McInnes and B. Pinkas, "On the impossibility of private key cryptography with weakly random keys," in *Proc. CRYPTO 1990,* pp. 421–435.

[20] K. Nuida, S. Fujitsu, M. Hagiwara, T. Kitagawa, H. Watanabe, K. Ogawa and H. Imai, "An improvement of discrete Tardos fingerprinting codes," *Des. Codes Cryptogr.,* vol. 52, pp. 339–362, 2009.

[21] K. Nuida and G. Hanaoka, "On the security of pseudorandomized information-theoretically secure schemes," in *Proc. ICITS 2009,* to be published.

[22] K. Nuida and G. Hanaoka, "An improvement of pseudorandomization against unbounded attack algorithms – the case of fingerprint codes," in *Proc. ICITS 2009,* to be published.

[23] R. Peralta and V. Shoup, "Primality testing with fewer random bits," *Comput. Complexity,* vol. 3, pp. 355–367, 1993.

[24] R. Renner and S. Wolf, "Unconditional authenticity and privacy from an arbitrary weak secret," in *Proc. CRYPTO 2003,* pp. 78–95.

[25] M. Santha and U. V. Vazirani, "Generating quasi-random sequences from semi-random sources," *J. Comput. Syst. Sci.,* vol. 33, pp. 75–87, 1986.

[26] A. Shamir, "How to share a secret," *Commun. ACM,* vol. 22(11), pp. 612–613, 1980.

[27] G. Tardos, "Optimal probabilistic fingerprint codes," *J. ACM,* vol. 55(2), pp. 1–24, 2008.

[28] U. V. Vazirani and V. V. Vazirani, "Random polynomial time is equal to slightly-random polynomial time," in *Proc. 26th FOCS,* 1985, pp. 417–428.

[29] D. Zuckerman, "Simulating BPP using a general weak random source," *Algorithmica,* vol. 16(4/5), pp. 367–391, 1996.