On the Security of the Cryptographic Mask Generation Functions Standardized by ANSI, IEEE, ISO/IEC, and NIST

Koutarou Suzuki and Kan Yasuda

Abstract

We revisit the security of mask generation functions (MGFs) in light of the indifferentiability framework. MGFs are a kind of hash function having variably long outputs and they are frequently utilized for designing public-key cryptographic schemes such as digital signatures. First, we clarify that there are weak and strong versions of indifferentiability, depending on the order of quantifies in the definition. Next, we prove that the classical, counter-based MGF standardized by ANSI, IEEE, and ISO/IEC satisfies only the weak version of indifferentiability, whereas the Double-Pipeline Iteration Mode specified in SP800-108 by the National Institute of Standards and Technology (NIST) satisfies the strong version. While our analysis does not necessarily imply that counter-based MGFs are insecure, our results show that MGF constructions have different levels of security (i.e., indifferentiability).

1. Introduction

1.1 Background

Ever since its establishment by Bellare and Rogaway [1], the notion of *random oracles* has played an essential role in the design of asymmetric cryptographic schemes [2], [3]. Informally, random oracles are objects that should behave like public random functions, accepting variable input length (VIL) data and returning variable output length (VOL) random strings. Random oracles are ideal objects: they cannot be implemented without additional assumptions. In practice, random oracles are replaced with concrete functions.

It is not an easy task to construct a random-looking VIL-VOL concrete function from scratch. So we usually start with a small concrete function that is restricted to a fixed input length (FIL) and fixed output length (FOL). Such functions are often called compression functions. We then iterate the compression functions in some way to obtain VIL and/or VOL functions.

Concrete functions that accept VIL strings but return only FOL strings are commonly called hash functions. The construction of *secure* hash functions has been theoretically investigated in various ways. In particular, the security of hash functions as VIL (but FOL) random oracles was studied by Coron et al. [4], where the underlying compression functions were modeled as FIL-FOL (restricted) random oracles in light of the indifferentiability framework [5]. Subsequent to the work reported in [4], the domain extension of random oracles has been analyzed in depth [6]–[11].

On the other hand, the range extension of random oracles has attracted less attention from the theoretical aspect. Despite the lack of formal treatment, VOL random oracles are regularly used in designing public-key cryptographic schemes, in particular digital signatures [3]. For the random oracles utilized in those signature schemes, which achieve full message recovery [12]–[16], the *variability* in output length becomes absolutely crucial.

There already exist several constructions for

Table 1. Summary of our results.

Definition of indifferentiability	Construction	
	Counter-based MGF	Chained MGF
Local ($\forall A \exists S$, Maurer et al. [5])	Secure [Theorem 1]	Secure [Theorem 3]
Universal ($\exists S \forall A$, Coron et al. [4])	Cannot be proven [Theorem 2]	Secure [Theorem 3]

Note: By "cannot be proven", we mean that it is impossible to prove that the construction is secure. We prove impossibility rather than give an attack; we do not mean that the construction is insecure.

VIL-VOL concrete functions. They are called by the common name mask generation functions (MGFs). The majority of existing MGFs follow the counter-based design and have been standardized by ANSI (American National Standards Institute), IEEE (Institute of Electrical and Electronics Engineers), and ISO/IEC (International Organization for Standardization, International Electrotechnical Commission). For example, the algorithm MGF1 [13], [17]–[19] takes a hash function H: $\{0, 1\}^* \longrightarrow \{0, 1\}^n$, computes upon input x the string

$$H\,(x\|\langle 0\rangle_{32}) \,\|\, H\,(x\|\langle 1\rangle_{32}) \,\|\, H\,(x\|\langle 2\rangle_{32}) \;\|\; \cdots,$$

and truncates this string to the leftmost l bits, where $\langle i\rangle_\alpha$ denotes an α-bit representation of integer i and l denotes the requested length. The main motivation behind the current work is to provide a formal security analysis for this type of construction.

The same types of algorithms are often called key derivation functions (KDFs), mostly when they take secret inputs. These are standardized in SP800-108 by NIST (National Institute of Standards and Technology) [20]. The security of KDFs is formally treated in [21]. We analyze the security of the Double-Pipeline Iteration Mode specified in SP800-108 as an MGF that takes only public inputs.

1.2 Our results

We take the systematic approach proposed by Coron et al. [4] and apply the indifferentiability framework [5] to our study of MGFs. That is, we consider two MGF constructions whose security is analyzed under the condition that an ideal hash function (a VIL/FOL random oracle) H: $\{0, 1\}^* \longrightarrow \{0, 1\}^n$ is given. Using this basic strategy, we obtain the following results, which are summarized in **Table 1**:

- Local vs. universal. We point out that in the literature there are two different versions of indifferentiability notions.
- Analysis of counter-based MGFs. We obtain two impossibility results for the counter-based MGF1. The first result says that MGF1 cannot be proven to be indifferentiable from the ideal MGF in the sense that there exist no *natural* simulators. The second result says that MGF1 *itself* cannot be proven to be insecure in the sense that there exists no *strong* adversary.
- Analysis of chained MGFs. We analyze the security of the Double-Pipeline Iteration Mode specified in NIST SP800-108, which can be shown to be indifferentiable from an ideal MGF. We provide concrete security bounds for the Double-Pipeline Iteration Mode. Unlike the case of domain extension, the security of this scheme degrades only *linearly* with the number of oracle queries[*1].

1.3 Organization

Section 2 defines our notation and provides other preliminaries. Section 3 reviews the notion of indifferentiability, identifies a class of natural simulators, and defines an MGF. Section 4 defines the counter-based MGF and analyzes its security. Section 5 defines the Double-Pipeline Iteration Mode and analyzes its security. Section 6 concludes with a brief summary and some concluding remarks about future work.

2. Preliminaries

2.1 Basic notation

$\{0, 1\}^m$ denotes the set of bit strings whose length is equal to $m > 0$. $\{0, 1\}^0$ denotes the set consisting of only the null string ε. $\{0, 1\}^*$ denotes the set of finite bit strings.

$|x|$ denotes the bit length of a string $x \in \{0, 1\}^*$.

$[x]^m$ represents the leftmost m bits of a string

*1 In the case of domain extension, a collision in the chaining values immediately leads to insecurity, which implies that the degradation is quadratic in query complexity.

$x \in \{0, 1\}^*$. $[x]_m$ represents the rightmost m bits.

Given two strings x and y, we let $x \| y$ be the concatenation of x and y.

$\lceil m \rceil$ indicates the smallest integer greater than or equal to an integer m.

We write $\mathbf{N}$ for the set of positive integers and write $\mathbf{Z}_{\geq 0}$ for the set of nonnegative integers.

By writing $x \in_U X$, we mean that x is an element chosen uniformly at random from the set X.

2.2 Security parameters and length encoding

A security parameter is a positive integer $\kappa \in \mathbf{N}$. It is customary to write $1^\kappa \in \{0, 1\}^*$ instead of $\kappa \in \mathbf{N}$ to emphasize the fact that κ is a security parameter. Whenever possible, we omit the security parameter κ and make it implicit in our statements.

2.3 Oracle machines

Throughout the paper, the computation model is fixed. Specifically, we regard any probabilistic algorithm as a (probabilistic) Turing machine. We consider an oracle machine, which is a Turing machine given access to an oracle. Interaction with an oracle is done via the machine's communication tape, and a reply from an oracle is given immediately, i.e., the time for interaction is 1 (unit time) irrespective of the query length, the reply length, and the oracle's behavior. Note, however, that the machine consumes the time taken to write its query onto the communication tape. Moreover, if the machine wants to read partially or wholly the reply written on the tape, the corresponding amount of time is consumed.

We write $A^{\mathcal{O}}$ to indicate the fact that a Turing machine A interacts with an oracle $\mathcal{O}$. We also let $A^{\mathcal{O}}$ denote the output value returned by A after its interaction with $\mathcal{O}$. We can always replace $\mathcal{O}$ with any other machine B that has a compatible interface, in which case we write A^B. We write $A^{\mathcal{O}_1, \mathcal{O}_2, \cdots}$ when A has access to multiple oracles.

2.4 Modes and distinguishers

A mode is a deterministic algorithm M that takes as its input a security parameter 1^κ and a finite string $x \in X$, where domain X is a subset of $\{0, 1\}^*$, and computes as its output a finite string $y \in \{0, 1\}^*$. A mode M has access to an oracle $\mathcal{H}$, and the interface between M and $\mathcal{H}$ depends on the security parameter κ. In other words, we can consider a family of oracles $\{\mathcal{O}_\kappa\}_\kappa$, from among which an appropriate oracle is chosen by M according to the value κ. Succinctly, we can write $y \leftarrow M^{\mathcal{O}_\kappa}(1^\kappa, x)$. Obviously, the algorithm $M^{\mathcal{H}}$ may not be deterministic if $\mathcal{H}$ is not, even though the mode M itself must be deterministic.

A distinguisher is a probabilistic algorithm D that takes as its input a security parameter 1^κ and outputs a bit $b \in \{0, 1\}$. A distinguisher D is given access to multiple oracles, and one of them is frequently mode M. In such a setting, we say that "the distinguisher D attacks the mode M." Succinctly written, $b \leftarrow D^{M^{\mathcal{O}_\kappa}(1^\kappa, \cdot), \cdots}(1^\kappa)$. Note that the same security parameter κ is used for both D and M.

2.5 Time and query complexities

Generally speaking, we may want to restrict the capacity of an oracle machine in terms of its time complexity and query complexity. In the present work, however, we treat only query complexity because an oracle machine's running time is irrelevant to the context of our security analysis[*2]. The query complexity is measured in terms of two quantities q_A and l_A for a given oracle machine A, where q_A represents the limit on the total number of queries that machine A can send to its oracles and l_A represents the limit on the maximum length of each query or reply.

A construction F is said to be *tractable* if its bounds q_F and l_F are polynomials in the following three variables: security parameter κ, input length $|x|$, and output length $|y|$. A distinguisher D is said to be *efficient* if its bounds q_D and l_D are polynomials in the security parameter κ. A simulator S is said to be *efficient* if its bounds q_S and l_S, as well as the size $|\sigma'|$ of updated state σ', are polynomials in the following four variables: security parameter κ, input state length $|\sigma|$, input length $|x|$, and output length $|y|$.

3. Indifferentiability framework and security of the MGF

In this section, we revisit the notion of indifferentiability. There are two points that we would like to clarify: (1) the definition of a simulator and (2) the order of quantifiers with respect to the simulator. Now, we define the security of the MGF.

3.1 Simulator division and connector extraction

In order to define indifferentiability, we need to introduce a simulator. A simulator S is a probabilistic algorithm that takes as its input a security parameter 1^κ, current state information $\sigma \in \Sigma$ (where the set Σ of state information is a subset of $\{0, 1\}^*$), and an input value $x \in X$ (where the domain X is a subset of $\{0, 1\}^*$)

*2 In our analysis, the source of randomness always involves random oracles, and we never deal with computational assumptions.

and computes as its output a pair of updated state information $\sigma' \in \Sigma$ and a finite string $y \in \{0, 1\}^*$. For convenience, we assume that the empty string ε is in the set Σ. Simulator S is always an oracle machine having access to some oracle M. Succinctly, we can write

$$(\sigma', y) \leftarrow S^M(1^\kappa, \sigma, x).$$

S's goal is to mimic some oracle $\mathcal{O}_\kappa$: $X \longrightarrow \{0, 1\}^*$ that is expected to return $y \in \{0, 1\}^*$ in response to the query $x \in X$.

We introduce a connector C, which is a dummy functionality whose purpose is merely to connect simulator S to an oracle machine D. A connector C is a stateful machine; that is, it has an internal memory that can store current state information $\sigma \in \Sigma$. The state σ is initially set to the empty string ε. Connector C works as follows. Upon receiving an oracle query $x \in X$ from distinguisher D, connector C forwards (σ, x) to simulator S and lets S compute $(\sigma', y) \leftarrow S^M(1^\kappa, \sigma, x)$. Connector C receives the output (σ', y) from S, updates its own state information from σ to σ', and returns the value y to D.

Consider a distinguisher $D^{\mathcal{O}_\kappa}$ interacting with an oracle $\mathcal{O}_\kappa$: $X \longrightarrow \{0, 1\}^*$. We can replace the oracle $\mathcal{O}_\kappa$ with the machine C^S and hence obtain D^{C^S}. Since the connector C does nothing but provide a trivial interface, we write (with abuse of notation) D^S instead of D^{C^S}.

3.2 Definition of indifferentiability: local vs. universal

There are two different versions of the indifferentiability notion. The setting for the notion of indifferentiability is as follows. Let D be an adversary. D's goal is to distinguish between the real world and the ideal world. In either world, D has access to two oracles. In the real world, we define efficient construction F having access to oracle ϕ to be indifferentiable from oracle Φ as follows. Consider a polynomial-time simulator S having access to oracle Φ and trying to simulate ϕ. Simulator S has complete knowledge of F. Consider a polynomial-time adversary D that has access to two oracles and is expected to output a bit at the end of each game execution. D has complete knowledge of not only F but also S. The notion of indifferentiability for F (together with S and D) is given by the following two different games: in the real game, D is given access to two oracles F and ϕ, while in the ideal game, D is given access to two oracles ϕ and S. We define the advantage $\mathrm{Adv}^{\mathrm{indiff}}_{F,S}(D)$ of adversary D as

$$\mathrm{Adv}^{\mathrm{indiff}}_{F,S}(D) = \left| \Pr[D^{F^\phi, \Phi} = 1] - \Pr[D^{\Phi, S^\phi} = 1] \right|,$$

where the probability is taken over the coin tosses ϕ and Φ.

Definition 1 (Local: Maurer et al. [5]). *Let F be an efficient construction. We say that F is indifferentiable from the random oracle (in the sense of Maurer et al.'s definition) if for any polynomial-time adversary D there exists an efficient simulator S and a negligible function $\epsilon(\kappa)$ such that the inequality $\mathrm{Adv}^{\mathrm{indiff}}_{F,S}(D) \leq \epsilon$ holds.*

Definition 2 (Universal: Coron et al. [4]). *Let F be an efficient construction. We say that F is indifferentiable from the random oracle (in the sense of Coron et al.'s definition) if there exists an efficient simulator S such that for any polynomial-time adversary D there exists a negligible function $\epsilon(\kappa)$ satisfying the inequality $\mathrm{Adv}^{\mathrm{indiff}}_{F,S}(D) \leq \epsilon$.*

To avoid confusion, we give specific names to these two notions: we say that an efficient construction F *is locally indifferentiable if* it is indifferentiable in the former sense and *universally indifferentiable* if it is indifferentiable in the latter sense. Clearly, universal indifferentiability implies local indifferentiability.

Remark 1. It seems that the two definitions arise from the difference in purpose. The main purpose of the former definition is to discuss security under the system compositions, and the definition indeed gives a necessary and sufficient condition for composability. On the other hand, the purpose of the latter is to measure *how good* a construction is, because the existence of a universal simulator shows that it is indeed a good construction, with the simulator being the inverse construction.

Remark 2. We emphasize that adversaries D and simulators S are merely algorithms (Turing machines). Hence, a simulator S is not allowed to *observe* the queries/replies made in the interaction between D and Φ[*3]. Moreover, note that D is not allowed to observe the running time of oracles with which it interacts because any oracle interaction takes exactly unit time.

*3 In fact, when D is interacting with the Φ-oracle, simulator S is not even invoked.

3.3 Definition of MGF functionality/security

To formalize the functionality of MGFs, we define MGFs and their corresponding random oracle (i.e., ideal MGF function), and we define hash functions and their corresponding random oracle (i.e., ideal hash function).

We start by giving a definition of an MGF. Intuitively, an MGF is defined as a concrete function that takes as its input a seed x together with the requested length l and returns a string of l bits. An ideal MGF is simply a monolithic random function having such an interface.

Definition 3 (MGF). *An MGF is a VIL-VOL function* F: $\{0, 1\}^* \times \{1\}^* \longrightarrow \{0, 1\}^*$ *satisfying the following two properties:*

1. Length: For all $x \in \{0, 1\}^* X$ *and* $l \in \mathbf{Z}_{\geq 0}$*, we have*

$$|F(x, 1^l)| = l.$$

2. Prefix: For all l; $l' \in \mathbf{Z}_{\geq 0}$ *such that* $l \leq l'$*, we have*

$$F(x, 1^l) = [F(x, 1^{l'})]^l.$$

The MGF random oracle $\mathcal{O}^{\mathrm{mgf}}$ *is a function chosen uniformly at random from the set of MGFs.*

The notion of MGFs corresponds to the original definition of VIL/VOL random oracles by Bellare and Rogaway [6]; an MGF specifies the length of outputs. The notion of MGFs is also compatible with the interface of the MGF1, which has been widely standardized [13], [17]–[19].

Below, we give one of several possible definitions of a hash function.

Definition 4 (Hash function). *A hash function is a VIL/FOL function* H: $\{0, 1\}^* \longrightarrow \{0, 1\}^n$*, where* $n \in \mathbf{N}$.

The random oracle $\mathcal{O}_n^{\mathrm{hash}}$ *is a function chosen uniformly at random from the set of hash functions with n-bit outputs.*

We say that efficient construction F of an MGF using random oracle $\phi = \mathcal{O}_n^{\mathrm{hash}}$ is secure if it is indifferentiable from MGF random oracle $\Phi = \mathcal{O}^{\mathrm{mgf}}$.

4. Analysis of counter-based MGFs

First, we define the counter-based MGF F. Then, we show that F cannot be proven to be indifferentiable from $\mathcal{O}^{\mathrm{mgf}}$ in the sense that there exists no unaffected simulator. On the other hand, we also show that F cannot be proven to be insecure in the sense that there exists no unaffected adversary.

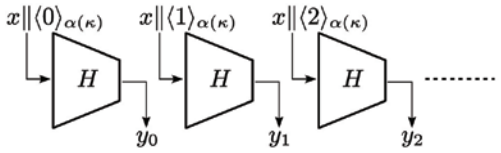


Fig. 1. Description of the counter-based MGF.

4.1 Description of the counter-based MGF

The counter-based MGF F: $\{0, 1\}^* \times \mathbf{N} \longrightarrow \{0, 1\}^*$ uses a hash function H: $\{0, 1\}^* \longrightarrow \{0, 1\}^n$. Here, the output length n is a polynomial function of the security parameter κ. The description of F is as follows:

1. Receive an input $(x; l) \in \{0, 1\}^* \times \mathbf{N}$.
2. Set $t = \lceil l/n \rceil$ and $r = l - n(t - 1)$.
3. Compute $y_i = \mathrm{H}(x \| \langle i \rangle_{\alpha(\kappa)})$ for $i = 0, \ldots, t - 1$.
4. Output $F(x, l) = y_0 \| \cdots \| y_{t-2} \| [y_{t-1}]^r$.

In the above, $\langle i \rangle_{\alpha(\kappa)}$ denotes an $\alpha(\kappa)$-bit representation of integer i, where $\alpha(\kappa)$ is a polynomial in κ. The counter-based MGF is illustrated in **Fig. 1**.

4.2 Proof that the counter-based MGF is locally indifferentiable

We prove that the counter-based MGF M defined above is *locally* indifferentiable. Intuitively, the proof goes as follows. Given an efficient distinguisher D, there exist polynomials $q_D(\kappa)$ and $l_D(\kappa)$. Using these polynomials, we can construct an efficient simulator S that sets the advantage of D to zero.

Theorem 1. *The counter-based MGF* M *is locally indifferentiable from an ideal MGF* M.

Proof. Let D be an efficient distinguisher attacking the counter-based MGF F. We show that there exists an efficient simulator S that makes the advantage function of D equal to 0.

Let $q_D(\kappa)$ and $l_D(\kappa)$ be polynomial functions restricting the capacity of D. Using these polynomials, we construct a simulator $S^{\Phi}(1^{\kappa}, \sigma, x)$ as follows.

1. Receive an $\mathcal{O}_n^{\mathrm{hash}}$-query $X \in \{0, 1\}^*$ from adversary D.
2. If (X, Y) is already in state σ, then return (σ, Y) to adversary D.
3. If $X = x \| \langle i \rangle_{\alpha(\kappa)}$ and $i \cdot n(\kappa) \leq l_D(\kappa)$, then compute $Y = [\Phi(x, (i+1) \cdot n(\kappa))]_{n(\kappa)}$ by asking Φ oracle and obtain updated state σ' by adding (X, Y) to σ and return (σ', Y) to adversary D.

4. If $X = x\|\langle i\rangle_{\alpha(\kappa)}$ and $i \cdot n(\kappa) > l_D(\kappa)$, then choose a random string $Y \in_U \{0, 1\}^{n(\kappa)}$ and obtain updated state σ' by adding (X, Y) to σ and return (σ', Y) to adversary D.

We see that S is an efficient simulator because we have $t_S = O(q_D l_D)$, $q_S = q_D$, and $l_S = l_D + n$. We also observe that S perfectly mimics the oracle $\mathcal{O}_n^{\mathrm{hash}}$ in a way consistent with the oracle $\mathcal{O}^{\mathrm{mgf}}$ (up to length l_D). Hence, we can set $\epsilon(\kappa) = 0$.

4.3 Proof that the counter-based MGF is not universally indifferentiable

Now we prove that the counter-based MGF F is not universally indifferentiable from an ideal MGF Φ. Intuitively, we argue that any single simulator S will fail when a distinguisher D starts by raising a query $x\|\langle i\rangle_{\alpha(\kappa)}$ for some huge i. In such a case, S is forced to decide whether or not to send a query $(x, n(i + 1)$ to its Φ oracle. If the value i is within the resource bounds of D, then S should certainly send such a query (and return the *consistent* value). On the other hand, if i is beyond the resource bounds of D, then S should simply ignore making such a query (and return a random string). However, S cannot make such a decision intelligently because it is not allowed to have any information about D.

Theorem 2. *The counter-based MGF F is not universally indifferentiable from an ideal MGF* Φ.

Proof. Suppose, on the contrary, that there exists a single simulator S that works against any efficient distinguisher. We show that this leads to a contradiction.

Let κ be the security parameter. Throughout the proof, we set the seed x to be a one-bit string "0," i.e., $x = 0$.

First, we define two types of events, Query_i and Reply_i, for $i \in \mathbf{Z}_{\geq 0}$. Every distinguisher that we construct in the current proof sends a query of the form $x\|\langle i\rangle_{\alpha(\kappa)}$, for some $i \in \mathbf{Z}_{\geq 0}$, to its H oracle at the beginning of each game execution. Let Query_i denote this event. After the event Query_i, the simulator S is given a pair $(\varepsilon, x\|\langle i\rangle_{\alpha(\kappa)})$ (ε being the null state) and is required to return updated state σ' and a string $y \in \{0, 1\}^{n(\kappa)}$. Let Reply_i denote this event, i.e., the event that $(\sigma', y) \leftarrow S^{\Phi}(1^{\kappa}, \varepsilon, x\|\langle i\rangle_{\alpha(\kappa)})$ is computed and returned to the intermediary I.

Next, we define probabilities $p_i(\kappa)$ for $i \in \mathbf{Z}_{\geq 0}$. Let Tune_i denote the event, which occurs between Query_i and Reply_i, of simulator S sending a query $(x, 1^l)$ to its Φ oracle for some $l \geq n(\kappa) \cdot i + 1$. Put $p_i(\kappa) = \Pr[\mathrm{Tune}_i]$. Since we fix the seed x, the probability $p_i(\kappa)$ is well-defined for each pair of an integer $i \in \mathbf{Z}_{\geq 0}$ and a security parameter $\kappa \in \mathbf{N}$. Note that the probability $p_i(\kappa)$ does not depend on the description of distinguishers and is defined over the coins of S and Φ (S may send some other queries to its Φ oracle in the interval).

We define a function $\mathbf{j}: \mathbf{N} \longrightarrow \mathbf{Z}_{\geq 0} \cup \{\infty\}$ as follows. For security parameter $\kappa \in \mathbf{N}$, let $\mathbf{j}(\kappa)$ be the smallest index i such that $p_i(\kappa) \leq 1/3$ (This fraction can be any constant strictly larger than 0 and strictly smaller than 1/2). If no such index exists, then we define $\mathbf{j}(\kappa)$ as the special symbol ∞, which is defined to be larger than any $i \in \mathbf{Z}_{\geq 0}$. Again, note that $\mathbf{j}$ is determined as soon as we fix the simulator S; the description of $\mathbf{j}$ is independent of distinguishers.

We show that $\mathbf{j}$ cannot be bounded by a polynomial function. Suppose, on the contrary, that there exists some polynomial function $f(\kappa)$ and an integer $N_1 \in \mathbf{N}$ such that for all security parameters $\kappa > N_1$ the inequality $\mathbf{j}(\kappa) < f(\kappa)$ holds. If such a polynomial function f exists, then it implies that there also exists a distinguisher $D_f^{\Phi, S}(1^{\kappa})$ as follows.

1. Choose a random index $i \in_U \{0, 1 \ldots, f(\kappa) - 1\}$,
2. Send a query $x\|\langle i\rangle_{\alpha(\kappa)}$ to its S oracle and receive a string $y \in \{0, 1\}^{n(\kappa)}$,
3. Send a query $(x, 1^{n(\kappa) \cdot (i + 1)})$ to its Φ oracle and receive a string $y \in \{0, 1\}^*$,
4. $y' \leftarrow [y]_{n(\kappa)}$,
5. If $y = y'$, return 1; otherwise, return 0.

Observe that the distinguisher D_f makes exactly two queries, each being at most $n(\kappa) \cdot f(\kappa)$ bits. Therefore, D_f is an efficient distinguisher.

We show that this is in direct contradiction to the requirement that the success probability of the distinguisher D_f be negligible. To see this, let us compute the advantage $\mathrm{Adv}^{\mathrm{mgf}}(D_f{}^{(1^{\kappa})})$ for sufficiently large security parameters $\kappa > N_1$. If D_f interacts with the pair $(F, \phi_{n(\kappa)})$, we can easily verify that D_f outputs 1 with probability 1. On the other hand, if D_f interacts with the pair (Φ, S), we claim that the probability of $D_f(1^{\kappa})$ returning 1 is at most $1 - f(\kappa)^{-1} \cdot (2/3 - 2^{-n(\kappa)})$. To see this, let One denote the event $D_f^{\Phi, S} = 1$ and Hit denote the event $i = \mathbf{j}(\kappa)$ in line 1 of the description of distinguisher $D_f^{\Phi, S}(1^{\kappa})$. We have

$$\begin{aligned}\Pr[\mathrm{One}] &= \Pr[\mathrm{Hit} \wedge \mathrm{One}] + \Pr[\overline{\mathrm{Hit}} \wedge \mathrm{One}] \\ &= \Pr[\mathrm{Hit}] \cdot \Pr[\mathrm{One} \mid \mathrm{Hit}] + \Pr[\overline{\mathrm{Hit}}] \cdot \Pr[\mathrm{One} \mid \overline{\mathrm{Hit}}] \\ &\leq \frac{1}{f(\kappa)} \cdot \Pr[\mathrm{One} \mid \mathrm{Hit}] + \frac{f(\kappa) - 1}{f(\kappa)} \cdot 1,\end{aligned}$$

and we also have

$$\begin{aligned}\Pr[\text{One} \mid \text{Hit}] &= \Pr[\text{Tune}_i \wedge \text{One} \mid \text{Hit}] + \Pr[\overline{\text{Tune}_i} \wedge \text{One} \mid \text{Hit}] \\ &= \Pr[\text{Tune}_i \mid \text{Hit}] \cdot \Pr[\text{One} \mid \text{Hit} \wedge \text{Tune}_i] \\ &\quad + \Pr[\overline{\text{Tune}_i} \mid \text{Hit}] \cdot \Pr[\text{One} \mid \text{Hit} \wedge \overline{\text{Tune}_i}] \\ &\leq \frac{1}{3} \cdot 1 + 1 \cdot \frac{1}{2^{n(\kappa)}},\end{aligned}$$

where the variable i denotes the value selected in line 1 of the description of the distinguisher $D_f^{\Phi, S}(1^\kappa)$. Hence, we get

$$\Pr\left[D_f^{\Phi, S}=1\right] \leq \frac{1}{f(\kappa)}\left(\frac{1}{3}+\frac{1}{2^{n(\kappa)}}\right)+\frac{f(\kappa)-1}{f(\kappa)}=1-\frac{1}{f(\kappa)}\left(\frac{2}{3}-\frac{1}{2^{n(\kappa)}}\right),$$

and we also get

$$\mathrm{Adv}_{F, S}^{\mathrm{mgf}}(D_f(1^\kappa)) \geq 1-1+\frac{1}{f(\kappa)}\left(\frac{2}{3}-\frac{1}{2^{n(\kappa)}}\right)=\frac{1}{f(\kappa)}\left(\frac{2}{3}-\frac{1}{2^{n(\kappa)}}\right) \geq \frac{1}{6f(\kappa)},$$

which is clearly not a negligible function.

Thus, we have shown that function $\mathbf{j}(\kappa)$ is not bounded by any polynomial function. We show that this also leads to a contradiction, creating another type of distinguisher.

To construct the distinguisher, we first identify a polynomial $g(\kappa)$ as follows. Consider an initial query $x\|\langle i\rangle_{\alpha(\kappa)}$. This leads to an input $(1^\kappa, \sigma, x\|\langle i\rangle_{\alpha(\kappa)})$ to the simulator S, where $\sigma = \varepsilon$ and $x = 0$. Hence, the length of such an input is $\kappa + 0 + 1 + \alpha(\kappa)$, which is a polynomial in κ. Since the bound l_S is a polynomial in the input length, we can regard l_S as a polynomial in κ, which we define as $g(\kappa) = l_S(\kappa + 1 + \alpha(\kappa))$. Now that we have identified a polynomial function $g(\kappa)$, we construct the distinguisher $D_g^{\Phi, S}(1^\kappa)$ as follows.

1. $i \leftarrow \min(g(\kappa) + 1, 2^{\alpha(\kappa)} - 1)$
2. Send a query $x\|\langle i\rangle_{\alpha(\kappa)}$ to its S oracle and discard whatever is received,
3. Return 1.

Next, we find a security parameter κ_1 for which running D_g with S leads to a contradiction. Observe that there exists some integer $N_0 \in \mathbf{N}$ such that for all $\kappa > N_0$, the inequality $g(\kappa) + 1 < 2^{\alpha(\kappa)} - 1$ holds because the left-hand side is a polynomial in κ whereas the right-hand side is an exponential function of κ. Now recall that $\mathbf{j}(\kappa)$ is not bounded by any polynomial function, which implies that there exists some integer $\kappa_1 > N_0$ such that $g(\kappa_1) + 1 < \mathbf{j}(\kappa_1)$. Here, it is important to note that we have $P_{g(\kappa_1)+1}(\kappa_1) > 1/3$ from the definition of $\mathbf{j}$.

Finally, by setting the security parameter κ to κ_1, we find a contradiction in the simulator's bound l_S when running D_g. The distinguisher D_g sends its S oracle a query $x\|\langle g(\kappa_1) + 1\rangle_{\alpha(\kappa_1)}$, which forces S with probability of more than 1/3 to send a query (x, l) to its Φ oracle for some $l \geq n(\kappa_1) \cdot (g(\kappa_1) + 1)$. Then, we have

$$g(\kappa_1) = l_S(\kappa_1 + 1 + \alpha(\kappa_1)) \geq l \geq n(\kappa_1) \cdot (g(\kappa_1) + 1),$$

which is a contradiction.

5. Analysis of chained MGFs

Our results for the counter-based MGF raise the question of whether there exists an MGF construction that can be proven to be universally indifferentiable from an ideal MGF. In this section, we present one such construction: the chained MGF. As an example of the chained MGF, we describe the Double-Pipeline Iteration Mode specified in NIST SP800-108 [20]. We then prove that the Double-Pipeline Iteration Mode is universally indifferentiable from an ideal MGF.

5.1 Description of the Double-Pipeline Iteration Mode

The Double-Pipeline Iteration Mode specified in NIST SP800-108 [20] F: $\{0, 1\}^* \times \mathrm{N} \longrightarrow \{0, 1\}^*$ uses a hash function H: $\{0, 1\}^* \longrightarrow \{0, 1\}^n$. Here, the output length $n = n(\kappa)$ is a polynomial function of the security parameter κ such that the inequality $n(\kappa) > \kappa$ holds for all $\kappa \in \mathbf{N}$. The description of F is as follows:

1. Receive an input $(x, l) \in \{0, 1\}^* \times \mathrm{N}$.
2. Set $t = \lceil l/n \rceil$, $r = l - n(t - 1)$ and $\upsilon_0 = x \in \{0, 1\}^*$.
3. Compute $\upsilon_i = H(\upsilon_{i-1})$ and $y_i = H(\upsilon_i \| \langle i\rangle_{\alpha(\kappa)} \| x)$ for $i = 1, \ldots, t$.
4. Output $F(x, l) = y_1 \| \cdots \| y_{t-1} \| [y_t]^r$.

In the above, $\langle i\rangle_{\alpha(\kappa)}$ denotes an $\alpha(\kappa)$-bit representation of integer i, where $\alpha(\kappa)$ is a polynomial in κ. The Double-Pipeline Iteration Mode is illustrated in **Fig. 2**.

Since the resource bound of the Double-Pipeline Iteration Mode F is $O(|x|l)$ and the output length is $O(l)$, the Double-Pipeline Iteration Mode F is efficient.

5.2 Proof that the Double-Pipeline Iteration Mode is universally indifferentiable

The Double-Pipeline Iteration Mode F is indifferentiable from MGF random oracle $\mathcal{O}^{\mathrm{mgf}}$.

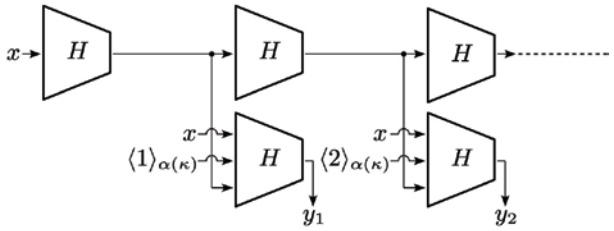


Fig. 2. Description of the Double-Pipeline Iteration Mode.

Theorem 3. *The Double-Pipeline Iteration Mode* $F^{\mathcal{O}_n^{\text{hash}}}$ *is universally indifferentiable from oracle* $\mathcal{O}^{\text{mgf}}$ *in the sense that there exists a universal simulator S having bounds* $t_S = O(q_D^2)$, $q_S = q_D$, $l_S = q_D n$, *and* $\epsilon = q_D/2^n$.

Proof. We construct a simulator S that has access to oracle $\mathcal{O}^{\text{mgf}}$ and that tries to simulate oracle $\mathcal{O}_n^{\text{hash}}$. S works as follows:

1. Receive an $\mathcal{O}_n^{\text{hash}}$-query $X \in \{0, 1\}^*$ from adversary D.
2. If the query $X \in \{0, 1\}^*$ is stored, return the stored answer to adversary D.
3. If $X = x$, return a random string $v_1 \in_U \{0, 1\}^n$ to D and store $(v_0 = x, v_1, 1, \text{“chained”})$.
4. If $X = v_i$ and $(v_{i-1}, v_i, i, \text{“chained”})$ is stored, return a random string $v_{i+1} \in_U \{0, 1\}^m$ to D and store $(v_i, v_{i+1}, i + 1, \text{“chained”})$.
5. If $X = v_i \,\|\langle i\rangle_{\alpha(\kappa)}\|x$ and $(v_{i-1}, v_i, i, \text{“chained”})$ is stored, return $y_i = [\mathcal{O}^{\text{mgf}}(x, i \cdot n)]_n \in \{0, 1\}^n$ to D and store $((v_i, i, x), y_i, i, \text{“chained”})$.
6. Otherwise, return a random string $Y \in_U \{0, 1\}^n$ to D and store $(X, Y, \text{“junk”})$.

Now we argue that simulator S is a polynomial-time adversary. To see this, let t_D, q_D, l_D be polynomial functions such that $D(\kappa) \in D(t_D, q_D, l_D)$. Since S makes $\mathcal{O}^{\text{mgf}}$-oracle queries only if distinguisher D makes a *chained* query, the number of queries sent by S to $\mathcal{O}^{\text{mgf}}$-oracle is at most q_D, and each query is of length at most $q_D n$ bits. Hence, we have $q_S = q_D$, $l_S = q_D n$. S needs to search at most q_D stored queries at most q_D times. Hence, we have $t_S = O(q_D^2)$.

S perfectly simulates oracle $\mathcal{O}_n^{\text{hash}}$ in a way consistent with the construction F except in the case that distinguisher D asks $X = v_i \,\|\langle i\rangle_{\alpha(\kappa)}\|x$ with the *correct* v_i before asking $X = v_{i-1}$. Hence, we have $\epsilon(\kappa) = q_D/2^n$.

However, the Double-Pipeline Iteration Mode outputs $n/2^n$ bits per hash function computation, so it is less efficient than a counter-based MGF, which outputs n bits per hash function computation.

6. Conclusion

We have shown that the counter-based MGF cannot be proven to be naturally indifferentiable from the ideal MGF. As a solution to this problem, we have shown that a chained MGF is proven to be indifferentiable from the ideal MGF. However, the chained MGF is less efficient than the counter-based MGF because it outputs fewer bits per invocation and operates in a non-parallelizable manner. It might be worth performing a more detailed study of this security/performance tradeoff because the current work opens up other possibilities for new MGF constructions that are indifferentiable from the ideal MGF and at the same time more efficient (or more secure) than the chained MGF.

References

[1] M. Bellare and P. Rogaway, “Random oracles are practical: A paradigm for designing efficient protocols,” Proc. of the 1st ACM Conference on Computer and Communications Security, pp. 62–73, Fairfax, VA, USA, 1993.

[2] M. Bellare and P. Rogaway, “Optimal asymmetric encryption,” In Alfredo De Santis, editor, EUROCRYPT 1994, Lecture Notes in Computer Science, Vol. 950, pp. 92–111, Heidelberg, 1995, Springer.

[3] M. Bellare and P. Rogaway, “The exact security of digital signatures—How to sign with RSA and Rabin,” In Ueli M. Maurer, editor, EUROCRYPT 1996, Lecture Notes in Computer Science, Vol. 1070, pp. 399–416, Heidelberg, 1996, Springer.

[4] J. -S. Coron, Y. Dodis, C. Malinaud, and P. Puniya, “Merkle-Damgård revisited: How to construct a hash function. In Victor Shoup, editor, CRYPTO 2005, Lecture Notes in Computer Science, Vol. 3621, pp. 430–448, Heidelberg, 2005, Springer.

[5] U. M. Maurer, R. Renner, and C. Holenstein, “Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology,” In Moni Naor, editor, TCC 2004, Lecture Notes in Computer Science, Vol. 2951, pp. 21–39, Heidelberg, 2004, Springer.

[6] M. Bellare and T. Ristenpart, “Multi-property-preserving Hash Domain Extension and the EMD Transform,” In Xuejia Lai and Kefei Chen, editors, ASI-ACRYPT 2006, Lecture Notes in Computer Science, Vol. 4284, pp. 299–314, Heidelberg, 2006, Springer.

[7] D. Chang, S. Lee, M. Nandi, and M. Yung, “Indifferentiable Security Analysis of Popular Hash Functions with Prefix-free Padding,” In Xuejia Lai and Kefei Chen, editors, ASIACRYPT 2006, Lecture Notes in Computer Science, Vol. 4284, pp. 283–298, Heidelberg, 2006, Springer.

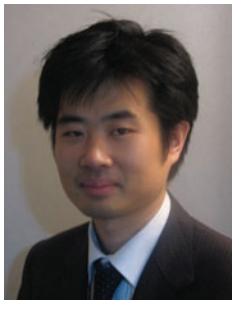
[8] M. Bellare and T. Ristenpart, “Hash functions in the dedicated-key setting: Design choices and MPP transforms,” In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, ICALP 2007, Lecture Notes in Computer Science, Vol. 4596, pp. 399–410, Heidelberg, 2007, Springer.

[9] S. Hirose, J. H. Park, and A. Yun, “A simple variant of the Merkle-Damgård scheme with a permutation,” In Kaoru Kurosawa, editor, ASIACRYPT 2007, Lecture Notes in Computer Science, Vol. 4833, pp. 113–129, Heidelberg, 2007, Springer.

[10] U. M. Maurer and S. Tessaro, “Domain extension of public random functions: Beyond the birthday barrier,” In Alfred Menezes, editor,

CRYPTO 2007, Lecture Notes in Computer Science, Vol. 4622, pp. 187–204, Heidelberg, 2007, Springer.
[11] D. Chang and M. Nandi, "Improved Indifferentiability Security Analysis of ChopMD Hash Function," In Kaisa Nyberg, editor, FSE 2008, Lecture Notes in Computer Science, Vol. 5086, pp. 429–443, Heidelberg, 2008, Springer.
[12] ISO/IEC, Geneva. ISO/IEC 9796-3 Information technology—Security techniques—Digital signature schemes giving message recovery—Part 3: Discrete logarithm based mechanisms, 2006.
[13] IEEE Computer Society, New York, "IEEE 1363.1 Standard Specifications For Public-Key Cryptography," 2000.
[14] L. A. Pintsov and S. A. Vanstone, "Postal revenue collection in the digital age," In Yair Frankel, editor, Financial Cryptography 2000, Lecture Notes in Computer Science, Vol. 1962, pp. 105–120, Heidelberg, 2001, Springer.
[15] M. Abe and T. Okamoto, "A Signature Scheme with Message Recovery as Secure as Discrete Logarithm," In Kwok-Yan Lam, Eiji Okamoto, and Chaoping Xing, editors, ASIACRYPT 1999, Lecture Notes in Computer Science, Vol. 1716, pp. 378–389, Heidelberg, 1999, Springer.
[16] M. Abe, T. Okamoto, and K. Suzuki, "Message Recovery Signature Schemes from Sigma-protocols," NTT Technical Review, Vol. 6, No. 1, 2008.
https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr200801sp2.html
[17] ANSI, New York, "ANSI X9.44 Draft D2," 2002.
[18] RSA Security, Bedford, "PKCS#1 v2.1," 2002.
[19] ISO/IEC, Geneva, "ISO/IEC 18033-2 Information technology—Security techniques—Encryption algorithms—Part 2: Asymmetric ciphers," 2006.
[20] NIST, "NIST SP800-108," 2009.
[21] H. Krawczyk, "Cryptographic Extraction and Key Derivation: The HKDF Scheme," CRYPTO 2010, pp. 631-648, Tal Rabin, editor, Lecture Notes in Computer Science Vol. 6223, Heidelberg, 2010, Springer.

Koutarou Suzuki
Senior Research Scientist, Information Security Project, NTT Secure Platform Laboratories.
He received the B.S., M.S., and Ph.D. degrees from the University of Tokyo in 1994, 1996, and 1999, respectively. He joined NTT in 1999. He has been engaged in research on public key cryptography, especially on cryptographic protocols and digital signatures.



Kan Yasuda
Senior Research Scientist, Information Security Project, NTT Secure Platform Laboratories.
He received the Ph.D. degree in mathematical sciences from the University of Tokyo in 2003. He has been working for NTT since 2004.