TTA Standard

정보통신단체표준(국문표준)

TTAK.KO-12.0190-Part1 개정일: 2018년 12월 xx일

해시 함수 기반 결정론적 난수발생기 - 제1부: 일반

Deterministic Random Bit Generator based on Hash Function - Part1: General

TTA 한국정보통신기술협회
Telecommunications Technology Association

표준초안 검토 위원회 정보보호기반 프로젝트그룹(PG501)

표준안 심의 위원회 정보보호 기술위원회(TC5)

	성명	소 속	직위	위원회 및 직위	표준번호
표준(과제) 제안	박제홍	NSR	책임연구원	정보보호기반 프로젝트그룹 위원	TTAK.KO-12.0190-Part1
표준 초안 작성자	박제홍	NSR	책임연구원	정보보호기반 프로젝트그룹 위원	TTAK.KO-12.0190-Part1
	김우환	NSR	책임연구원	-	TTAK.KO-12.0190-Part1
	이향진	KISA		-	TTAK.KO-12.0190-Part1
	김기문	KISA	선임연구원	정보보호기반 프로젝트그룹 위원	TTAK.KO-12.0190-Part1
사무국 담당	박수정	TTA	책임	-	

본 표준 발간 이전에 접수된 지식재산권 확약서 정보는 본 표준의 '부록(지식재산권 확약서 정보)'에 명시하고 있으며, 이후 접수된 지식재산권 확약서는 TTA 웹사이트에서 확인할 수 있습니다.
본 표준과 관련하여 접수된 확약서 외의 지식재산권이 존재할 수 있습니다.

발행인 : 한국정보통신기술협회 회장
발행처 : 한국정보통신기술협회
13591, 경기도 성남시 분당구 분당로 47
Tel : 031-724-0114, Fax : 031-724-0109
발행일 : 2018.12

서 문

1 표준의 목적

난수 발생기는 암호 응용에서 다양한 목적을 위해 요구되는 난수를 생성하는 용도로 사용된다. 이 표준의 목적은 암호학적 해시 함수(cryptographic hash function)를 사용하여 난수를 생성하는 결정론적 난수 발생 메커니즘(DRBG 메커니즘)을 제시하는 것이다.

2 주요 내용 요약

이 표준은 DRBG 메커니즘을 사용하는 결정론적 난수 발생기의 기능 모델을 소개하고 DRBG 메커니즘의 주요 구성 요소를 제시한다. 이어서 해시 함수(hash function)를 기반으로 정의된 DRBG 메커니즘인 Hash_DRBG의 상세 규격을 제시한다.

3 인용 표준과의 비교

3.1 인용 표준과의 관련성

이 표준은 NIST SP 800-90A에 규정된 Hash_DRBG의 상세 규격을 준용한다.

3.2 인용 표준과 본 표준의 비교표

<table>
<tr><th>TTAK.KO-12.0190-Part1</th><th>NIST SP 800-90A</th><th>비고</th></tr>
<tr><td>1. 적용 범위</td><td>-</td><td>추가</td></tr>
<tr><td>2. 인용 표준</td><td>-</td><td>추가</td></tr>
<tr><td>3. 용어 정의</td><td>4. Terms and Definitions</td><td rowspan="6">선별적으로 인용
(Hash_DRBG 규격 위주)</td></tr>
<tr><td>4. 약어</td><td>5. Symbols and abbreviated Terms</td></tr>
<tr><td rowspan="2">5. DRBG 메커니즘의 기능 모델</td><td>7. Functional Model of a DRBG</td></tr>
<tr><td>8. DRBG Mechanism Concepts and General Requirements</td></tr>
<tr><td rowspan="2">6. Hash_DRBG</td><td>9. DRBG Mechanism Functions</td></tr>
<tr><td>10. DRBG Algorithm Specifications</td></tr>
</table>

Preface

1 Purpose

Random number generators are used in numerous cryptographic applications to generate random values for various purposes. The standard provides a mechanism for the generation of a random value using a secure hash function.

2 Summary

The standard provides a functional model for a deterministic random bit generator(DRBG) based on DRBG mechanism, and discusses the major components of the DRBG mechanism. In addition, the standard specifies the functions of the DRBG mechanism Hash_DRBG using a generic cryptographic hash function.

3 Relationship to Reference Standards

The standard conforms to the specifications of the Hash_DRBG in NIST SP 800-90A.

목 차

해시 함수 기반 결정론적 난수 발생기
- 제1부: 일반
(Deterministic Random Bit Generator based on Hash Function - Part1: General)

1 적용 범위

랜덤 비트열(random bit string)은 무작위 값(random value)이 요구되는 다양한 정보보호 시스템과 암호제품에서 직접 또는 난수(random number)로 변환되어 사용된다. 참고로 랜덤 비트열을 난수로 변환하는 방법은 [3]을 참조할 수 있으며, 이 표준에서는 변환 방법에 대해서는 별도로 규정하지 않는다. 그리고 아래에서는 “랜덤 비트열”과 “난수”를 구분하지 않고 “난수”로 통일하여 사용한다. 난수를 생성하는 방법은 크게 두 가지로 분류할 수 있다. 첫 번째는 난수를 구성하는 모든 비트를 예측 불가능한 물리적인 절차에 기반하여 생성하는 것이고, 두 번째는 알고리즘을 사용하여 주어진 입력에 대해 규정된 절차에 따른 계산을 통해 난수를 생성하는 것이다. 첫 번째 방식을 따르는 난수 발생기를 “비결정론적 난수 발생기(NRBG, non-deterministic random bit generator)”라고 하며, 두 번째 방식을 따르는 경우를 “결정론적 난수 발생기(DRBG, deterministic random bit generator)”라고 한다. 통상적으로는 첫 번째를 실난수 발생기(true random number generator), 두 번째를 의사난수 발생기(pseudorandom number generator)라고 부른다.

결정론적 난수 발생기는 결정론적 난수 발생 메커니즘(이하 DRBG 메커니즘)을 기반으로 DRBG 메커니즘이 생성하는 값에 대한 난수성을 보장할 수 있는 엔트로피 소스(entropy source)의 관리를 포함하는 개념이다. DRBG 메커니즘은 초기값인 씨드(seed)로부터 비트열을 생성하는 알고리즘을 사용하며, 이때 씨드는 엔트로피 소스로부터 선택된 값에 의해 결정된다. 그리고 DRBG 메커니즘에 의해 생성되는 난수는 알고리즘과 씨드, 그리고 다른 입력 정보가 주어지는 경우 예측 및 재생성 가능한 특성을 가진다.

이 표준에서는 해시 함수를 사용하는 DRBG 메커니즘인 Hash_DRBG를 규정한다. 엔트로피 소스의 관리에 대해서는 이 표준에서 구체적으로 다루지 않으며, 관련한 구체적인 내용은 [5]를 참고할 수 있다.

Hash_DRBG를 기반으로 동작하는 결정론적 난수 발생기는 암호 키 생성, 키 공유 등 난수가 필요한 다양한 정보보호시스템 및 암호제품에 활용되어 국내 정보통신망의 안전성과 신뢰성을 제고할 수 있다.

2 인용 표준

NIST SP 800-90A (2015), Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revision 1).
(7절(Functional Model of a DRBG)과 8절(DRBG Mechanism Concepts and General Requirements)에서 DRBG 모델을 선별적으로 인용하였으며, 9절(DRBG Mechanism Functions)과 10절(DRBG Algorithm Specifications)에서 Hash_DRBG 규격만 선별하여 인용함)

3 용어 정의

3.1 결정론적 알고리즘(deterministic algorithm)
동일한 입력에 대해서 항상 동일한 출력을 생성하는 특성을 가진 알고리즘

3.2 결정론적 난수 발생기(DRBG, Deterministic Random Bit Generator)
엔트로피 소스를 비롯한 여러 출처로부터 확보한 충분한 엔트로피를 가지는 씨드와 부가 입력을 DRBG 메커니즘에 적용하여 난수를 생성하는 장치 또는 알고리즘

3.3 결정론적 난수 발생 메커니즘(DRBG mechanism)
DRBG의 인스턴스 생성과 난수 생성, 그리고 리씨드 과정에 필요한 함수를 정의한 DRBG 구성 요소

3.4 개별화 문자열(personalization string)
인스턴스 생성 함수에서 씨드를 생성할 때 사용되는 입력으로 DRBG 인스턴스마다 서로 다른 씨드를 생성하도록 하기 위한 입력

3.5 난수(random number)
집합에서 동일한 확률로 선택 가능하여 예측불가능성(unpredictability)을 가지는 값

3.6 내부 상태(internal state)
DRBG 운용과 관련한 정보들의 집합으로 동작 상태(3.8절 참조)와 관리 정보로 구성

3.7 논스(nonce)
인스턴스 생성 함수에서 씨드를 생성하기 위해 사용되는 입력으로, 재사용되지 않아야 하는 값이며 DRBG에 추가적인 엔트로피를 제공하는 등 안전성을 강화하는 용도로 사용

3.8 동작 상태(working state)
DRBG 메커니즘에 의해 난수 생성에 사용되는 내부 상태의 구성 요소로, 다음 난수 생성

전에 갱신됨

3.9 비결정론적 난수 발생기(NRBG, Non-deterministic Random Bit Generator)
안전성을 엔트로피 소스의 샘플링에 의존하는 난수 발생 장치 또는 알고리즘

3.10 씨드(seed)
DRBG 메커니즘의 입력으로 사용되어 초기 동작 상태를 결정하는 비트열

3.11 씨드 유효기간(seedlife)
DRBG 인스턴스를 초기화한 후 다른 씨드로 재초기화하기까지 시간 구간

3.12 엔트로피(entropy)
데이터가 가지는 정보량을 수치로 나타낸 것. 무질서도(disorder) 또는 난수성(randomness)의 측정 기준

3.13 엔트로피 입력(entropy input)
엔트로피를 가지는 데이터로 인스턴스 생성 함수 및 리씨드 함수에서 씨드에 엔트로피를 추가할 때 사용

3.14 엔트로피 소스(entropy source)
특정 방식에 따라 포착되거나 처리됨에 따라 엔트로피를 포함하는 비트열을 생성할 수 있는 시스템 구성 요소, 장치 또는 이벤트. 일반적으로 잡음원(3.17절 참조)과, 잡음원에서 확보된 데이터를 보정하기 위한 함수, 그리고 건전성 시험으로 구성

3.15 예측 내성(prediction resistance)
공격자가 특정 시점의 내부 상태 정보를 이용하여 이후의 출력값을 예측할 수 없도록 하는 성질

3.16 예측불가능성(unpredictability)
난수 발생기의 출력값을 정확하게 예측할 수 있는 확률이 우연하게 맞추는 것과 거의 차이가 없는 성질

3.17 잡음원(noise source)
엔트로피 소스에서 생성되는 비트열의 불확정성(uncertainty)을 보장할 수 있는 비결정적(non-deterministic)인 엔트로피 제공 절차를 포함하는 엔트로피 소스의 핵심 구성 요소

3.18 추가 입력(additional input)
리씨드 함수에서 씨드에 엔트로피를 추가하거나 생성 함수에서 내부 상태에 엔트로피를 추가하기 위해 사용되는 입력

3.19 해시 함수(hash function)

임의 길이의 메시지를 일정 길이의 출력값으로 압축하며, 다음 두 가지 성질을 가지는 암호 알고리즘

- (역상 저항성) 주어진 출력값에 대응하는 입력 값을 찾는 것이 어려움
- (충돌 저항성) 동일한 출력값을 가지는 서로 다른 입력값들을 찾는 것이 어려움

3.20 DRBG 운용 주체(consuming application)

DRBG를 직접 동작시켜 무작위 수를 생성하고, 이를 사용하는 응용

[출처(3.1, 3.2, 3.9~3.12, 3.14, 3.19)]

KS X ISO/IEC 18031. 정보기술 - 보안기술 - 난수발생기, 2013

[출처(3.3~3.8, 3.13, 3.15, 3.16, 3.18, 3.20)]

NIST SP 800-90A, Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revision 1), 2015

[출처(3.17)]

NIST SP 800-90B, Recommendation for the Entropy Sources Used for Random Bit Generation, 2018

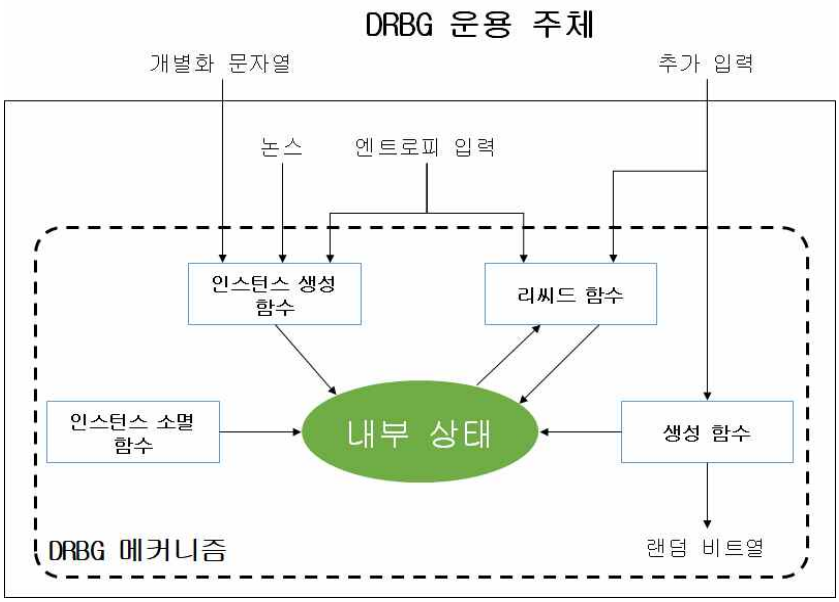
4 약어

FIPS	Federal Information Processing Standard
FIPS PUB	FIPS Publication
NIST	National Institute of Standards and Technology
SHA	Secure Hash Algorithm
SP	Special Publication

5 DRBG 메커니즘의 기능 모델

5.1 개요

본 절에서는 DRBG 메커니즘의 주요 구성 요소를 소개하고, DRBG 메커니즘의 안전한 운용을 위해 요구되는 각 구성 요소의 역할을 정리한다. 먼저 DRBG 메커니즘의 동작 과정을 도시하면 (그림 5-1)과 같다.



(그림 5-1) DRBG 메커니즘의 기능 모델

DRBG 메커니즘을 구성하는 주요 함수로는 인스턴스 생성(instantiate) 함수, 생성(generate) 함수, 리씨드(reseeding) 함수, 그리고 인스턴스 소멸(uninstantiate) 함수가 있다. 인스턴스 생성 함수(instantiate function)는 충분한 엔트로피를 가지는 엔트로피 입력과 부가 데이터를 이용하여 DRBG 출력값의 예측불가능성을 보장하는 씨드를 생성한다. 그리고 생성된 씨드를 이용하여 DRBG 메커니즘의 주요 설정과 이에 따른 동작 방식을 결정하고 출력값 생성에 직접 사용되는 내부 상태(internal state)를 초기화한다. 생성 함수(generate function)는 설정된 내부 상태를 이용하여 요구된 길이의 출력값을 생성하고, 다음 출력값 생성을 위해 내부 상태를 갱신한다. 리씨드 함수(reseeding function)는 내부 상태와 새로운 엔트로피 입력, 그리고 부가 데이터를 이용하여 새로운 씨드를 생성하고, 생성된 씨드를 이용하여 내부 상태를 재초기화한다. 인스턴스 소멸 함수(uninstantiate function)는 인스턴스 생성 함수로부터 생성되어 관리되고 있는 내부 상태를 해제시켜 해당 DRBG 메커니즘 인스턴스가 더 이상 사용될 수 없도록 한다.

5.2 엔트로피 입력과 논스

5.2.1 엔트로피 입력(entropy input)

엔트로피 입력은 DRBG의 안전성에 부합하는 엔트로피를 가지는 기본 씨드 생성 요소(seed material)이다. 엔트로피 입력은 엔트로피 소스 또는 다른 DRBG나 NRBG로부터 확보할 수 있다. 엔트로피 입력과 이를 이용하여 생성한 씨드는 DRBG 외부에 노출되지 않도록 비밀로 관리되어야 하며, 이는 DRBG 안전성의 근간이 된다.

엔트로피 입력을 얻기 위해 엔트로피 소스를 사용하는 경우, <표 5-1>과 같은 잡음원을 이용할 수 있다.

<표 5-1> 잡음원 구성 예

고정된 시스템정보	가변적 시스템정보	사용자/외부 주변장치
시스템 설정정보 네트워크 주소 하드웨어 ID 디스크 S/N	화면정보 시스템의 시간/클럭 네트워크 트래픽 프로세스 통계	사용자 입력 키보드 입력 주기 마우스 움직임 마우스 클릭 주기 마이크 잡음
← 예측이 쉬움		예측이 어려움 →

DRBG 운용 주체는 DRBG의 운용에 앞서, 엔트로피 소스로부터 DRBG에 요구하는 안전성 수준에 부합하는 엔트로피를 확보할 수 있는지 여부를 확인해야 한다. 이와 관련하여 SW 환경에서 잡음원에 대한 엔트로피 측정을 위해서는 [2]에서 제시하는 방법을 사용할 수 있다.

5.2.2 논스(nonce)

논스는 DRBG의 인스턴스 생성 과정에서 엔트로피 입력과 함께 내부 상태 초기화를 위한 씨드 생성에 사용된다. 논스는 적어도 해당 DRBG 인스턴스에 요구되는 안전성 수준의 절반 크기의 엔트로피를 가지거나, DRBG 인스턴스 운용 과정에서 재사용되지 않는 특성을 가져야 한다. 이러한 특성을 가진 정보의 예로 충분한 정밀도를 가지는 타임스탬프(timestamp)나 단조 증가 수열(monotonically increasing sequence), 또는 이들의 조합을 들 수 있으며, 실제 논스로 사용되는 경우 유효성이 확인되어야 한다. 참고로 엔트로피를 보강하기 위한 용도로 논스를 사용할 경우, 논스는 엔트로피 입력과 동일한 출처로부터 동시에 얻을 수 있다.

5.3 씨드(seed)

씨드는 DRBG 메커니즘의 인스턴스 생성 과정과 리씨드 과정에서 DRBG의 최초 출력값 생성에 필요한 초기 내부 상태(initial internal state)를 결정하는 데 사용된다. 따라서 씨드는 DRBG 인스턴스를 구별하는 요소로, DRBG의 출력값이 생성되기 전에 DRBG 인스턴스의 초기화에 적용되어야 한다.

5.3.1 초기 씨드 생성

DRBG의 인스턴스 생성 함수는 엔트로피 입력과 논스를 확보하고, 이를 외부에서 입력받는 개별화 문자열(5.4.1절 참조)과 결합하여 씨드 생성 요소를 구성한다.

씨드의 최소 엔트로피를 보장하기 위해 엔트로피 소스에서 수집된 데이터인 엔트로피 입력과 인스턴스 생성 과정마다 매번 다른 값이 사용되는 논스는 씨드 생성을 위한 필수

요소이다. 반면 개별화 문자열은 씨드 생성을 위한 선택 요소(option)이지만, 씨드의 비밀성을 강화하거나 동일한 엔트로피 소스를 기반으로 동작하는 다른 DRBG 인스턴스에서 사용되는 씨드와의 독립성을 제공할 수 있는 요소로써 사용을 권고한다.

이러한 씨드 생성 요소로부터 씨드를 유도하기 위해서 유도 함수(derivation function)를 사용하며, Hash_DRBG는 해시 함수를 이용하여 유도 함수를 정의한다(6.2절 참조). 유도 함수는 씨드의 유일성(uniqueness)을 보장할 수 있어야 한다.

5.3.2 씨드 교체

현재 사용되고 있는 내부 상태에 반영된 씨드의 교체는 ① DRBG 운용 주체에 의해 새로운 씨드가 주입되거나, ② 인스턴스 생성 함수에 예측 내성 요구가 입력되었거나, ③ 씨드의 유효기간이 만료되었을 때 수행된다.

씨드의 유효기간에 대해서는 5.3.7절에서 구체적으로 다룬다. 예측 내성은 공격자가 특정 시간의 내부 상태에 대한 정보를 이용하여 이후 생성되는 DRBG 출력값을 예측하려는 공격에 대한 안전성을 의미한다. 따라서 DRBG가 예측 내성을 엄밀하게 보장하기 위해서는 출력값을 생성할 때마다 씨드를 충분한 엔트로피를 가지는 새로운 씨드로 교체하여 내부 상태 사이의 연관성을 없애는 것이 필요하다.

씨드 교체를 수행하는 리씨드 함수(reseeding function)는, 현재 사용하고 있는 내부 상태와 엔트로피 소스로부터 새로 수집된 엔트로피 입력, 그리고 선택적으로 추가 입력을 결합하여 씨드 생성 요소를 구성한다. 그리고 씨드 생성 요소를 유도 함수에 입력하여 씨드를 생성한다.

5.3.3 씨드 사용

DRBG 메커니즘은 씨드를 사용하여 초기화되며, 필요한 경우 새로운 씨드를 사용하여 재초기화될 수 있다. 물론 출력값 생성 과정에서 부가 입력이 씨드로부터 생성된 초기 상태와 같이 사용될 수 있지만, 부가 입력의 사용은 선택 사항이다. 따라서 개별 DRBG 인스턴스의 독립성을 보장하기 위해, 특정 DRBG 인스턴스의 초기화에 사용된 씨드는 리씨드 과정에서 반복 사용되거나 다른 DRBG 인스턴스의 씨드로 사용되지 않아야 한다. 그리고 씨드가 생성되고 내부 상태가 초기화되는 (재)초기화 과정이 정상적으로 종료되기 전에는 해당 DRBG 인스턴스가 출력값을 생성하지 않아야 한다.

5.3.4 씨드 엔트로피

씨드 생성에 사용되는 엔트로피 입력은 DRBG 인스턴스에 요구되는 안전성 수준에 부합하는 충분한 엔트로피를 가져야 하고, 유도 함수에 의해 씨드에 고르게 분포되어야 한다.

(재)초기화 과정에서 씨드를 생성할 때, 씨드 생성 요소를 구성하는 엔트로피 입력 이외의 부가 입력 정보(5.4절 참조)를 사용하여 씨드에 엔트로피를 추가할 수 있다. 이러한 부가 입력 정보는 씨드가 노출되더라도 DRBG 출력값의 예측불가능성을 유지하는 데 도움이 될 수 있다. 그러나 부가 정보의 사용은 선택 사항임을 고려할 때, DRBG의 안전성은 근본적으로 엔트로피 입력이 가지는 엔트로피만으로 보장 가능해야 한다.

5.3.5 씨드 길이

씨드의 최소 길이는 DRBG 메커니즘, DRBG 운용 주체가 요구하는 안전성 수준, 그리고 엔트로피 소스의 품질 등에 따라 가변적이다. 그러나 씨드 길이는 최소한 해당 DRBG 인스턴스의 안전성과 같아야 하며, 엔트로피 소스의 품질에 따라서는 더 길게 설정되어야 한다.

5.3.6 씨드 보호

씨드는 DRBG 출력의 용도에 의해 요구되는 보안과 동등한 방식으로 취급되어야 한다. 예를 들어 암호 시스템에서 유일한 비밀 요소가 암호 키인 경우, 암호 키 생성에 사용된 씨드는 암호 키와 동일한 기준으로 보호되어야 한다.

5.3.7 씨드 사용기간

씨드로부터 결정된 내부 상태를 이용하여 많은 출력을 생성하는 것은 경우에 따라 이후 생성되는 DRBG 출력에 대한 예측 가능성을 높이는 요인이 될 수 있다. 이에 대한 대책으로, DRBG는 씨드의 유효기간(seedlife)을 설정하고 주기적으로 교체하여 DRBG의 안전성 저하 요인을 배제할 수 있다.

씨드는 주기적으로 교체(reseeding)되거나, 유효기간이 지나면 교체를 통해 사용이 불가능하도록 관리되어야 한다. 그리고 유효기간이 지난 후 교체되기 전까지는 DRBG가 출력값을 생성할 수 없어야 한다. 씨드의 유효기간은 시간 범위(time span)로 주어지거나 씨드로부터 생성될 수 있는 출력값의 최대 개수로 설정될 수 있다.

5.4 부가 입력 정보

DRBG 메커니즘을 구성하는 핵심 함수인 인스턴스 생성 함수, 리씨드 함수, 그리고 생성 함수를 기준으로, 각 함수에서 씨드나 내부 상태의 관리를 위해 사용하는 요소를 정리하면 <표 5-2>와 같다.

<표 5-2> DRBG 메커니즘의 각 함수에서 사용되는 입력들({ • }는 선택 요소)

함수명	사용되는 입력
인스턴스 생성 함수 (씨드)	엔트로피 입력, 논스, {개별화 문자열}
리씨드 함수 (씨드)	동작 상태, 엔트로피 입력, {추가 입력}
생성 함수 (내부 상태)	동작 상태, {추가 입력}

주요 부가 입력 정보로는 개별화 문자열(personalization string)과 추가 입력(additional input)이 있다. 이러한 부가 입력 정보는 엔트로피 입력과 별개로 DRBG의 출력값이 예측불가능성을 가지도록 엔트로피를 추가할 수 있으나, 선택 입력임을 고려할 때 DRBG의 안전성은 전적으로 씨드에 의존해야 한다. 따라서 공격자가 부가 입력 정보를 알더라도 엔트로피 입력이 비밀로 관리되는 경우에는 DRBG의 안전성이 저하되지 않아야 한다. 적용되는 부가 입력 정보는 개별 정보의 특성에 대한 유효성이 확인되어야 한다.

5.4.1 개별화 문자열(personalization string)

개별화 문자열은 인스턴스 생성 함수의 선택 입력으로 씨드 생성에 사용된다. 개별화 문자열은 동일한 엔트로피 소스를 기반으로 동작하는 동일 유형의 개별 DRBG 인스턴스의 초기화에 사용되는 씨드와 차별성을 제공할 수 있는 정보를 포함할 수 있다. 이러한 특성을 가지는 정보로 기기 고유번호(device serial numbers), 사용자 식별자(user identification), 타임스탬프(timestamps), 네트워크 주소(network addresses) 등을 개별화 문자열에 포함시킬 수 있다.

5.4.2 추가 입력(additional input)

추가 입력은 갱신 함수와 생성 함수의 선택 입력으로 사용된다. 추가 입력은 내부 상태에 엔트로피를 추가하여 DRBG 출력값의 예측불가능성을 유지시키는 수단으로 사용할 수 있다. 또한 추가 입력이 비밀로 관리되고 충분한 엔트로피를 가지는 경우, 씨드, 엔트로피 입력, 또는 내부 상태의 노출로부터 DRBG의 안전성 저하를 최소화하는 수단이 될 수 있다.

5.5 내부 상태(internal state)

내부 상태는 DRBG 메커니즘이 동작하는 데 필요한 모든 파라미터, 변수 및 다른 저장값들로 구성된다. 내부 상태는 특히 생성 함수에서 출력값을 생성하는 데 필요한 정보를 모두 포함한다. 기본적인 구성은 다음과 같다.

1. 동작 상태(working state)
   - 씨드로부터 직접 유도되는 값으로, 출력값 생성에 직접 사용되는 주요 정보

- 씨드로부터 초기화된 이후 출력값을 생성한 횟수

2. 관리 정보(administrative information): DRBG 메커니즘의 주요 설정과 이에 따른 동작 방식을 결정하는 정보

내부 상태는 생성 함수에서 출력값을 생성할 때마다 잠재적으로 갱신되는 동작 상태와, 일정하게 유지되면서 DRBG 운용 주체에 의해 갱신될 수 있는 관리 정보의 조합으로 볼 수 있다. 내부 상태의 세부 구성은 DRBG 메커니즘에 따라 다르며, Hash_DRBG의 동작 상태는 6.1절에 정리한다.

각 DRBG 인스턴스는 독립적인 내부 상태를 가져야 하며, 특정 DRBG 인스턴스의 내부 상태가 다른 DRBG 인스턴스의 내부 상태로 사용되지 않아야 한다. DRBG 메커니즘의 출력은 동작 상태에 의해 결정되므로, 동작 상태는 반드시 비밀로 관리되어야 한다.

5.6 출력값 생성

생성 함수는 내부 상태를 이용하여 출력값을 생성한다. 생성 함수는 주어진 입력에 대해 결정적으로 동작하며, 출력 형태는 DRBG를 사용하는 응용 환경에 따라 결정된다. 생성 함수는 내부 상태에 대한 어떠한 정보도 노출시키지 않아야 한다.

출력값 생성 과정에서는 내부 상태의 갱신이 발생하며, 갱신된 내부 상태는 다음 출력값의 생성에 사용된다. 내부 상태 갱신 과정에서는 현재 내부 상태의 모든 정보가 갱신되는 내부 상태에 반영되어야 하며, 부가 정보인 추가 입력을 통해 엔트로피를 보강할 수 있다. 내부 상태는 DRBG의 출력을 결정하는 정보이므로, 내부 상태의 갱신 과정은 부채널 공격에 의한 노출과 분석으로부터 보호되어야 한다.

6 해시 함수 기반 난수 발생 메커니즘(Hash_DRBG)

6.1 개요

Hash_DRBG는 해시 함수를 사용하여 정의된 DRBG 메커니즘이다. Hash_DRBG는 인스턴스 생성과 리씨드 과정을 포함하여 다양한 DRBG 메커니즘의 동작 과정에 동일한 해시 함수를 사용해야 하며, Hash_DRBG가 보장 가능한 최대 안전성은 기반 해시 함수의 역상 저항성에 의존한다. 따라서 적절한 해시 함수를 사용하고 씨드로부터 충분한 엔트로피가 확보될 경우, Hash_DRBG는 다양한 수준의 안전성을 요구하는 응용에서 사용될 수 있다.

<표 6-1>은 해시 함수 SHA-2[4]를 사용하여 Hash_DRBG를 구현할 경우 필요한 주요 설정을 정리한 것이다. 다른 해시 함수를 사용하는 경우에는 해당 해시 함수의 안전성

수준을 고려하여 파라미터를 설정할 필요가 있으며, 해시 함수 LSH[1]를 사용할 경우 <표 6-1>의 설정을 준용한다.

<표 6-1> SHA-2 기반 Hash_DRBG 주요 파라미터 설정 예 (길이 단위: 비트)

	SHA-224, SHA-512/224	SHA-256, SHA-512/256	SHA-384	SHA-512
지원 가능한 안전성 수준 (supported security strengths)	112, 128, 192	112, 128, 192, 256		
highest_supported_security_strength	224	256	384	512
출력 블록 길이 (*outlen*)	224	256	384	512
인스턴스 생성/리씨드 단계에서 요구되는 최소 엔트로피	*security_strength*			
엔트로피 입력 최소 길이 (*min_length*)	*security_strength*			
엔트로피 입력 최대 길이 (*max_length*)	2^{35}			
씨드 길이 (*seedlen*)	440		888	
개별화 문자열 최대 허용 길이 (*max_personalization_string_length*)	2^{35}			
추가 입력 최대 허용 길이 (*max_additional_input_length*)	2^{35}			
난수 최대 출력 길이 (*max_no_of_bits_per_request*)	2^{19}			
씨드별 출력값 생성 횟수 (*reseed_interval*)	2^{48}			

Hash_DRBG 인스턴스 생성 단계에서 DRBG 운용 주체는 해당 인스턴스가 가져야 할 안전성 수준을 파라미터(*security_strength*)로 입력하고, 이에 대응하여 인스턴스 생성 함수는 입력된 안전성 수준(*security_strength*)에 부합하는 크기의 엔트로피를 확보한다(알고리즘 2 참조). 따라서 특정 Hash_DRBG 인스턴스가 지원하는 실제 안전성은 Hash_DRBG 구현물이 보장하는 최대 안전성 수준(*highest_supported_security_strength*) 내에서 인스턴스 생성 함수에 제공되는 엔트로피 크기에 의해 결정된다. 예를 들어 최대 256 비트 안전성을 지원할 수 있는 Hash_DRBG를 사용하더라도, 인스턴스 생성 과정에서 제공되는 씨드 생성 요소의 전체 엔트로피 크기가 128 비트일 경우 해당 Hash_DRBG 인스턴스의 안전성은 128 비트이다.

Hash_DRBG 출력값의 안전성은 해당 DRBG 인스턴스의 안전성과 출력값 길이 중 작은 값으로 결정된다. Hash_DRBG 운용 주체는 개별 난수 요청 시 해당 출력값의 안전성 수준을 제시한다(알고리즘 4 참조). 이때 Hash_DRBG 인스턴스의 안전성 수준을 벗어나지 않는 범위 내의 임의의 안전성 수준을 요구할 수 있으며, 이러한 조건이 유효한 경우 생

성 함수의 출력값이 DRBG 인스턴스의 출력값으로 반환된다. 그리고 Hash_DRBG의 출력값은 해당 DRBG 인스턴스의 안전성을 초과하는 안전성 수준을 요구하는 운용 주체에게는 반환되지 않아야 한다.

특정 Hash_DRBG 인스턴스가 여러 가지 용도로 사용될 경우, 각 용도에서 요구되는 최소 안전성 수준 중 가장 큰 안전성 수준을 지원할 수 있도록 인스턴스 생성 과정에서 요구되어야 한다.

Hash_DRBG를 구성하는 주요 함수인 인스턴스 생성, 리씨드, 생성, 그리고 인스턴스 소멸 함수는 실행 결과의 성공/실패 여부를 *status* 값으로 반환한다. DRBG 운용 주체는 반환된 *status* 값을 반드시 확인하여 각 함수가 성공적으로 동작하였는지, 아니면 실패하였을 경우 별도의 추가 작업(remedial action)이 필요한지 여부를 결정해야 한다. 예를 들어, 인스턴스 생성 함수가 오류 메시지를 반환하는 경우 내부 상태가 초기화되지 못함에 따라 무의미한 핸들값(*state_handle*)이 반환된다. 그러나 유효한 핸들값의 부재는 뒤이어 호출되는 리씨드 함수나 생성 함수에서 자연스럽게 검출된다. 반면 생성 함수가 오류 메시지를 반환하는 경우, 출력 비트열은 *Null* 값이 반환되기 때문에 DRBG 인스턴스의 출력값으로 사용되지 않아야 한다.

Hash_DRBG의 내부 상태(5.5절 참조)는 <표 6-2>와 같이 구성된다.

<표 6-2> Hash_DRBG 내부 상태 구성

구분	기호	의미
동작 상태	*V*	*seedlen* 비트 길이의 변수로, DRBG 인스턴스의 출력값 생성 시 갱신
	C	씨드로부터 계산되는 *seedlen* 비트 길이의 상수
	reseed_counter	씨드 생성 이후 DRBG 인스턴스의 출력값 생성 횟수
관리 정보	*security_strength*	DRBG 인스턴스의 안전성 수준
	prediction_resistance_flag	DRBG 인스턴스의 예측 내성 보장 여부

내부 상태에서 *reseed_counter*는 DRBG 인스턴스의 출력값 생성 횟수를 기록하는 카운터 변수로, 정해진 최대 출력값 생성 횟수(*seed_interval*)을 초과하는 즉시 리씨드 함수를 호출하기 위해 사용된다(알고리즘 4 참조). 그리고 *prediction_resistance_flag*는 해당 DRBG 인스턴스가 예측 내성을 보장하도록 운용할지 여부를 설정한 값이다.

5.5절에서 기술한 바와 같이, 동작 상태를 구성하는 *V*와 *C*는 DRBG 인스턴스의 안전성 보장을 위해 비밀로 관리되어야 한다.

Hash_DRBG의 주요 함수를 설명하기 위해 사용하는 기호를 정리하면 다음과 같다.

⌈x⌉	x와 같거나 큰 가장 작은 정수. x의 올림
leftmost(V, a)	V의 맨 왼쪽 a 비트
len(a)	문자열 a의 비트 길이
x **mod** n	정수 x와 정수 n(>0)에 대하여 x를 n으로 나눈 나머지. y=x mod n 이면 y는 다음을 만족하는 유일한 정수 - 0 ≤ y < n - (y−x)는 n의 배수

Hash_DRBG의 주요 함수를 구체적으로 기술하기에 앞서, 먼저 인스턴스 생성 함수와 리씨드 함수의 씨드 생성 과정에 사용되는 유도 함수(derivation function)를 제시한다.

6.2 유도 함수(derivation function)

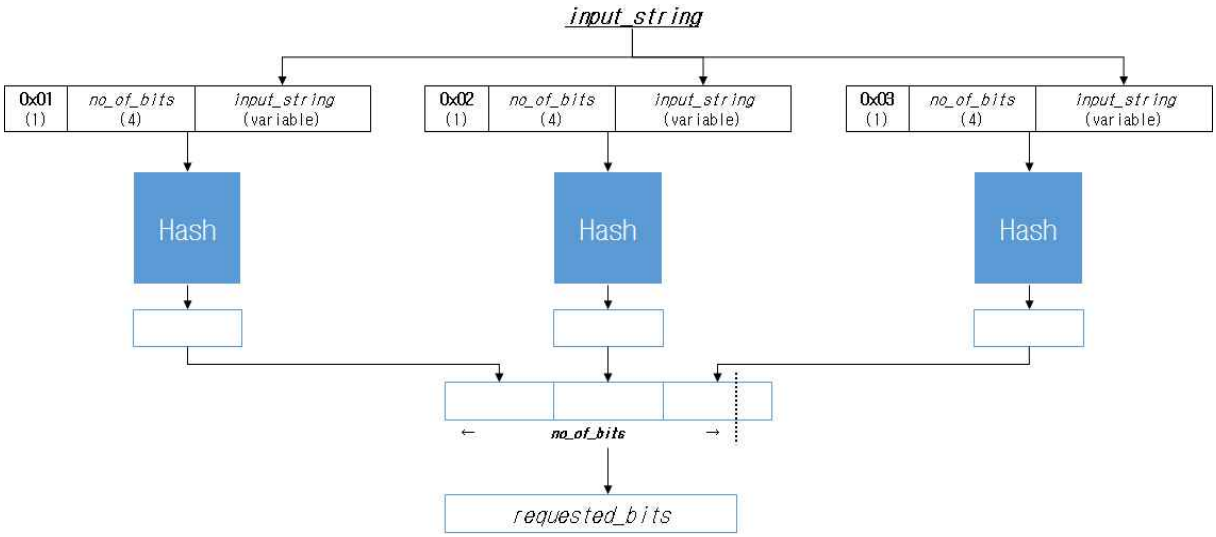
Hash_DRBG의 유도 함수 Hash_df는 내부 상태 구성 요소를 생성하거나 입력이 가진 엔트로피를 출력에 고르게 분포되도록 한다. 유도 함수 Hash_df는 임의 길이의 입력으로부터 고정 길이의 출력을 반환하는 일종의 해시 함수로 볼 수 있다. 유도 함수 Hash_df의 구체적인 동작 과정은 알고리즘 1과 같다.

알고리즘 1 유도 함수 Hash_df(·,·)

입력: 비트열 *input_string*, 정수 *no_of_bits*
출력: 비트열 *requested_bits*

```
1:  temp ← Null
2:  len ← ⌈no_of_bits/outlen⌉
3:  for i = 1 to len do
4:      temp ← temp‖Hash(i‖no_of_bits‖input_string)
5:  end do
6:  requested_bits ← leftmost(temp, no_of_bits)
7:  return requested_bits
```

유도 함수 Hash_df는 8비트 이진값으로 표현되는 카운터와 출력 값의 비트 길이(*no_of_bits*), 그리고 입력받은 비트열(*input_string*)을 연접하고 기반 해시 함수 Hash에 입력하여 원하는 길이(*no_of_bits*)의 출력을 얻는다. 여기에서 출력값의 비트 길이(*no_of_bits*)는 32비트로 표현한다. 따라서 *no_of_bits*는 255×*outlen*보다 같거나 작아야 한다. 유도 함수 Hash_df의 동작 과정을 도시하면 (그림 6-1)과 같다.



(그림 6-1) Hash_DRBG의 유도 함수 Hash_df

6.3 인스턴스 생성 함수(instantiate function)

Hash_DRBG의 인스턴스 생성 함수 Instantiate_Hash_DRBG는 엔트로피 입력, 논스, 개별화 문자열로부터 씨드를 생성하고, 이 씨드를 이용하여 내부 상태를 초기화한다.

인스턴스 생성 함수의 입력 파라미터는 <표 6-3>과 같다.

<표 6-3> 인스턴스 생성 함수의 입력 파라미터

파라미터	의미
requested_strength	- 생성하는 DRBG 인스턴스의 안전성 수준 - DRBG 구현물이 하나의 안전성 수준만 지원하는 경우 생략 가능 (DRBG 운용 주체는 사전 인지 필요)
prediction_resistance_flag	- DRBG 인스턴스의 예측 내성 보장 여부 - DRBG 구현물이 예측 내성을 항상 보장하거나 아예 기능을 제공하지 않는 경우에는 생략 가능
personalization_string	- 개별화 문자열(소절 5.4.1 참조)

입력 파라미터인 *prediction_resistance_flag*는 DRBG 운용 주체가 해당 DRBG 인스턴스에 대해 예측 내성을 보장하도록 운용할지 여부를 설정한 표시자(indicator)이다. 예측 내성 기능을 상시 지원하거나(엔트로피 소스로부터 새로운(fresh) 엔트로피 입력 확보 가능) 아니면 예측 내성 기능 제공을 제공하지 않는(새로운 엔트로피 입력 확보 불가능) DRBG 구현물은, 이 파라미터를 지원할 필요는 없다. 그러나 DRBG 운용 주체의 사용자는 DRBG 구현물을 선택함에 앞서 구현물의 예측 내성 기능 지원 가능 여부를 인지하고, 예측 내성이 해당 응용에 필요한지 여부를 결정해야 한다.

인스턴스 생성 함수의 구체적인 동작 예시는 알고리즘 2와 같다.

알고리즘 2 인스턴스 생성 함수 Instantiate_Hash_DRBG(·,·,·)

입력: 정수 *requested_strength*, 정수 *prediction_resistance_flag*,
{비트열 *personalization_string*}
출력: 비트열 *status*, 정수 *state_handle*

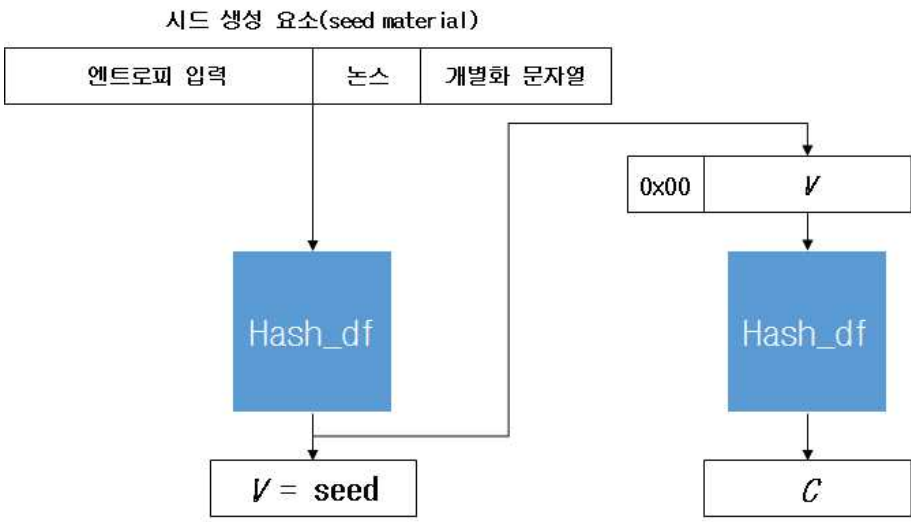
```
1:  if (requested_strength > highest_supported_security_strength) then
2:      return (ERROR_FLAG, Invalid) // ex. ( "Invalid requested_security_strength" , -1)
3:  end if
4:  if ((prediction_resistance_flag is set) AND
        (prediction resistance is not supported) then
5:      return (ERROR_FLAG, Invalid)
6:  end if
7:  if (len(personalization_string)> max_personalization_string_length) then
8:      return (ERROR_FLAG, Invalid) // ex. ( "Personalization_string too long" , -1)
9:  end if
10: if (requested_strength ≤ 112) then
11:     security_strength ← 112
12: else if (requested_strength ≤ 128) then
13:     security_strength ← 128
14: else if (requested_strength ≤ 192) then
15:     security_strength ← 192
16: else
17:     security_strength ← 256
18: end if
19: (status, entropy_input) ← Get_entropy(min_entropy, min_len, max_len, request )
        // min_len = min_length, max_len = max_length, request = prediction_resistance_request
20: if (status ≠ SUCCESS) then
21:     return (status, Invalid)
22: end if
23: obtain a nonce and check its acceptability
24: seed_material ← entropy_input ‖ nonce ‖ personalization_string
25: seed ← Hash_df(seed_material, seedlen )
26: (status, state_handle) ← Find_state_space()
27: if (status ≠ SUCCESS) then
```

```
28:     return (status, Invalid)
29: end if
30: state(state_handle).V ← seed
31: state(state_handle).C ← Hash_df((0x00 ‖ V), seedlen)
32: state(state_handle).reseed_counter ← 1
33: state(state_handle).security_strength ← security_strength
34: state(state_handle).prediction_resistance_flag ← prediction_resistance_flag
35: return (SUCCESS, state_handle)
```

인스턴스 생성 함수의 세부 과정을 살펴보면 다음과 같다.

- 입력 파라미터 유효성 확인 (단계 1 - 단계 9)
- DRBG 인스턴스의 안전성 수준(*security_strength*) 결정 (단계 10 - 단계 18)
- DRBG 인스턴스의 안전성 수준(*security_strength*)에 부합하는 엔트로피 입력 (*entropy_input*) 확보 (단계 19 - 단계 22)
- (필요시) 논스(*nonce*) 확보 (단계 23)
- 씨드 생성 및 초기 내부 상태 결정 (단계 24 - 단계 34)
- DRBG 인스턴스의 내부 상태를 관리하기 위한 핸들(*state_handle*) 반환 (단계 35)

특히 씨드 생성 및 초기 내부 상태 결정과정(단계 24 - 단계 34)을 도시하면 (그림 6-2)와 같다.



(그림 6-2) 인스턴스 생성 함수의 씨드 생성과 내부 상태 (*V*, *C*) 초기화

알고리즘 2의 단계 19에서는 엔트로피 소스로부터 엔트로피 입력(*entropy_input*)을 확보하기 위해 함수 Get_entropy를 사용한다. 여기에서 엔트로피 소스는 다른 DRBG나

NRBG를 포함한다. 함수 Get_entropy는 길이가 *min_length* 이상, *max_length* 미만이고, *min_entropy* 이상의 엔트로피를 가지는 비트열을 엔트로피 입력으로 반환한다. 이때 *min_length*는 *min_entropy* 이상으로 설정되어야 하며, 입력 파라미터 중 다른 입력 파라미터와의 연관 관계로부터 쉽게 도출 가능한 경우는 생략할 수 있다. 이 표준에서는 함수 Get_entropy의 구체적인 구현에 대해서는 다루지 않는다.

알고리즘 2의 단계 26에서는 미사용 내부 상태를 찾기 위해 함수 Find_state_space를 사용한다. 함수 Find_state_space는 호출 결과로 성공 또는 사용 가능한 미사용 내부 상태가 없음을 알리는 *status* 값을 반환한다. 그리고 성공일 경우 내부 상태 배열에서 가용인 핸들(*state_handle*)이 같이 반환되고, 오류 메시지를 반환하는 경우 내부 상태가 초기화되지 못함에 따라 잘못된 핸들(*state_handle*)이 반환된다.

DRBG 운용 주체가 예측 내성을 보장하도록 설정(*prediction_resistance_flag* = 1)한 경우, 생성되는 DRBG 인스턴스는 예측 내성을 상시 보장해야 한다. 이를 위해 파라미터 유효성 검증(단계 4 - 단계 6) 과정에서 DRBG 구현이 예측 내성 기능을 지원하지 않는 경우, 유효한 인스턴스 생성에 실패하도록 한다. DRBG 구현이 예측 내성을 보장하기 위해서는 엔트로피 소스로부터 항상 새로운(fresh) 엔트로피 입력을 확보할 수 있어야 한다. 이를 위해 DRBG 인스턴스의 Get_entropy 함수는 엔트로피 소스나 NRBG에 기반하지 않고 동작하는 DRBG를 엔트로피 소스로 사용하는 것은 허용되지 않는다. 이는 함수 Get_entropy의 입력 파라미터인 *prediction_resistance_request*의 설정에 따른다.

인스턴스 생성 함수가 *prediction_resistance_flag* 파라미터를 입력으로 받지 않는 경우(DRBG 운용 주체의 사용자가 DRBG 구현물이 예측 내성을 항상 보장하거나 아예 기능을 제공하지 않는 것을 인지한 경우), 알고리즘 2에서 관련 파라미터 유효성 검증 과정(단계 4 - 단계 6)과 내부 상태 설정 과정(단계 34)은 생략한다. 그리고 엔트로피 입력을 확보하는 과정(단계 19)에서 Get_entropy 함수는 *prediction_resistance_request* 파라미터를 입력으로 받을 필요가 없다.

그리고 입력 파라미터에서 개별화 문자열(*personalization_string*)을 사용하지 않을 경우, 알고리즘 2에서 관련 파라미터 유효성 검증 과정(단계 7 - 단계 9)은 생략한다. 그리고 단계 24의 씨드 생성요소(*seed_material*) 구성에서 개별화 문자열을 생략한다.

알고리즘 2의 동작 과정에서 사용되는 논스가 난수일 경우, 엔트로피 입력과 함께 Get_entropy 함수로부터 확보할 수 있다. 이 경우 Get_entropy 함수의 입력 파라미터들은 논스가 가져야 하는 엔트로피를 고려하여 조정되어야 한다. 그리고 알고리즘 2에서 논스 확보 과정(단계 23)을 생략하고 단계 24의 씨드 생성요소 구성에서 논스를 생략한다. 논스를 사용하지 않는 경우에도 알고리즘 2는 동일하게 수정한다.

알고리즘 2의 결과는 *status* 값으로 DRBG 운용 주체에게 반환된다. *status* 값이

SUCCESS가 아닌 경우, Hash_DRBG 인스턴스 관리를 위한 핸들(*state_handle*)이 반환되지 않거나 무의미한 핸들(*state_handle*)이 DRBG 운용 주체에게 반환된다. DRBG 운용 주체는 Hash_DRBG 인스턴스가 올바르게 생성되었는지 여부를 결정하기 위해 반환된 *status* 값을 확인해야 한다.

6.4 리씨드 함수(reseeding function)

Hash_DRBG의 리씨드 함수 Reseed_Hash_DRBG는 인스턴스의 내부 상태와 엔트로피 입력, 그리고 추가 입력을 이용하여 새로운 씨드를 생성하고, 이 씨드를 이용하여 내부 상태를 초기화한다.

리씨드 함수의 입력 파라미터는 <표 6-4>와 같다.

<표 6-4> 리씨드 함수의 입력 파라미터

파라미터	의미
state_handle	- 재초기화되는 내부 상태 식별자(핸들)
prediction_resistance_request	- 엔트로피 소스로부터 확보된 새로운 엔트로피 입력의 사용 여부 - DRBG 구현물이 예측 내성을 항상 보장하거나 아예 기능을 제공하지 않는 경우에는 생략 가능
additional_input	- 추가 입력(소절 5.4.2 참조)

입력 파라미터인 *prediction_resistance_request*는 엔트로피 소스로부터 새로운(fresh) 엔트로피 입력을 확보할지 여부를 설정한 표시자(indicator)이며, Get_entropy 함수에 입력으로 반영된다. Get_entropy 함수에서 *prediction_resistance_request* 파라미터를 0으로 설정할 경우, 엔트로피 소스에 대한 제약을 두지 않는다. 반면 1로 설정하는 경우에는 엔트로피 소스나 NRBG에 기반하지 않고 동작하는 DRBG를 엔트로피 소스로 사용하는 것이 허용되지 않는다.

리씨드 함수의 구체적인 동작 예시는 알고리즘 3과 같다.

알고리즘 3 리씨드 함수 Reseed_Hash_DRBG(・,・)

입력: 정수 *state_handle*, 정수 *prediction_resistance_request*, 비트열 *additional_input*
출력: 비트열 *status*

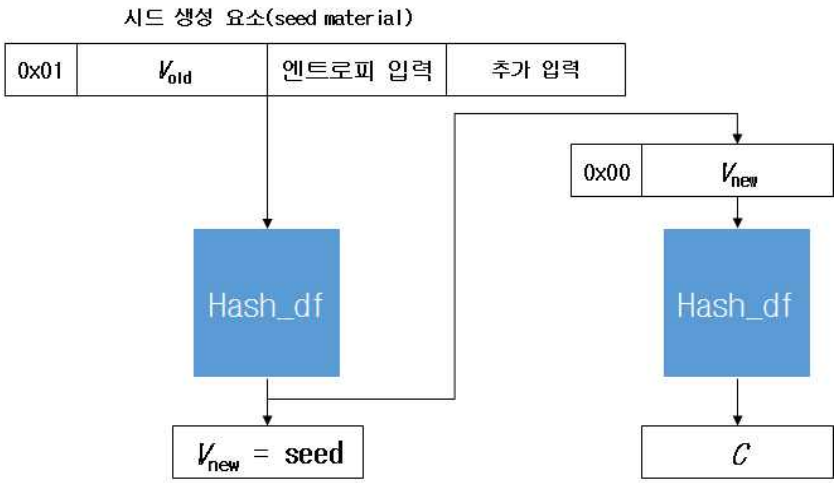
```
1:  if (a state(state_handle) is not available) then
2:      return ERROR_FLAG  // ex. "State not available for the state_handle"
3:  end if
4:  V ← state(state_handle).V
```

```
5:  security_strength ← state(state_handle).security_strength
6:  (status, entropy_input) ← Get_entropy(min_entropy, min_len, max_len, request)
        // min_len = min_length, max_len = max_length, request = prediction_resistance_request
7:  if (status ≠ SUCCESS) then
8:      return (status)
9:  end if
10: seed_material ← 0x01 ‖ V ‖ entropy_input ‖ additional_input
11: seed ← Hash_df(seed_material, seedlen)
12: state(state_handle).V ← seed
13: state(state_handle).C ← Hash_df((0x00 ‖ V), seedlen)
14: state(state_handle).reseed_counter ← 1
15: return (SUCCESS)
```

리씨드 함수의 세부 과정을 살펴보면 다음과 같다.

- DRBG 인스턴스 내부 상태 유효성 확인 (단계 1 - 단계 3)
- DRBG 인스턴스의 안전성 수준(*security_strength*)에 부합하는 엔트로피 입력 확보 (단계 6 - 단계 9)
- 현재 내부 상태와 새로운 엔트로피 입력, 그리고 부가 입력을 이용하여 새로운 씨드를 생성하고, 이 씨드를 이용하여 내부 상태를 갱신 (단계 10 - 단계 14)

특히 새로운 씨드를 생성하고 내부 상태를 갱신하는 과정(단계 10 - 단계 14)을 도시하면 (그림 6-3)과 같다.



(그림 6-3) 리씨드 함수의 씨드 생성과 내부 상태 (*V*, *C*) 갱신

리씨드 함수의 입력 파라미터에서 *prediction_resistance_request* 파라미터를 사용하지

않을 경우(생성 함수로부터 해당 파라미터가 입력되지 않는 경우), 단계 6의 Get_entropy 함수는 *prediction_resistance_request* 파라미터를 입력으로 받지 않는다.

리씨드 함수의 입력 파라미터에서 추가 입력(*additional_input*)을 사용하지 않을 경우, 알고리즘 3의 씨드 생성 요소(*seed_material*) 구성에 반영한다.

6.5 생성 함수(generate function)

Hash_DRBG의 생성 함수 Generate_Hash_DRBG는 인스턴스의 내부 상태 중 동작 상태의 *V*를 이용하여 출력 비트열을 생성하고, 동작 상태의 *C*와 *reseed_counter*를 이용하여 *V*를 갱신한다.

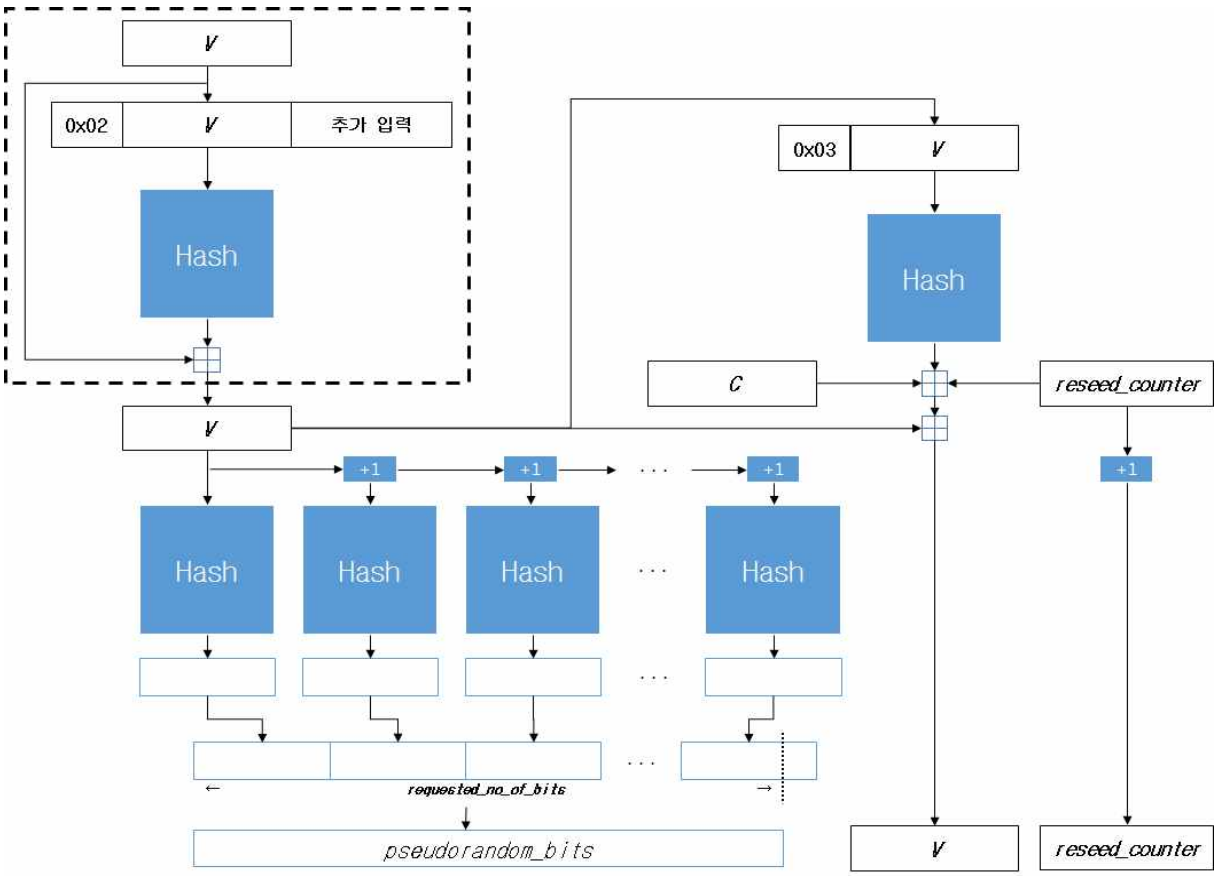
생성 함수의 입력 파라미터는 <표 6-5>와 같다.

<표 6-5> 생성 함수의 입력 파라미터

파라미터	의미
state_handle	- 재초기화되는 내부 상태를 가리키는 식별자(핸들)
requested_no_of_bits	- 생성 함수로부터 반환되어야 하는 출력값의 비트 길이
requested_strength	- DRBG 인스턴스 출력값의 안전성 수준 - DRBG 구현물이 하나의 안전성 수준만 지원하는 경우 생략 가능 (DRBG 운용 주체는 사전 인지 필요)
prediction_resistance_request	- 엔트로피 소스로부터 확보된 새로운 엔트로피 입력의 사용 여부 - DRBG 구현물이 예측 내성을 항상 보장하거나 아예 기능을 제공하지 않는 경우에는 생략 가능
additional_input	- 추가 입력(소절 5.4.2 참조)

입력 파라미터인 *prediction_resistance_request*는 엔트로피 소스로부터 새로운(fresh) 엔트로피 입력을 확보할지 여부를 설정한 표시자(indicator)이며, 리씨드 함수(reseeding function)의 입력에 반영된다. 엔트로피 소스로부터 새로운(fresh) 엔트로피 입력을 항상 확보할 수 있거나 아니면 새로운 엔트로피 입력 확보가 아예 불가능한 DRBG 구현물은 이 파라미터를 지원할 필요가 없다. 그러나 새로운 엔트로피 입력 확보가 불가능한 구현의 경우, DRBG 운용 주체는 DRBG 구현물의 선택에 앞서 예측 내성의 필요 여부를 결정해야 한다.

생성 함수의 동작 과정을 도시하면 (그림 6-4)와 같다.



(그림 6-4) 생성 함수

그리고 생성 함수의 구체적인 동작 예시는 알고리즘 4와 같다. 실행 과정에서 오류가 발생하는 경우, 출력 *pseudorandom_bits*는 *Null* 값을 반환한다.

알고리즘 4 생성 함수 Generate_Hash_DRBG(·,·,·,·,·)
입력: 정수 *state_handle*, 정수 *requested_no_of_bits*, 정수 *requested_strength*, 정수 *prediction_resistance_request*, {비트열 *additional_input*}
출력: 비트열 *status*, 비트열 *pseudorandom_bits*

```
1:  if (a state(state_handle) is not available) then
2:      return (ERROR_FLAG, Null)
3:  end if
4:  V ← state(state_handle).V
5:  C ← state(state_handle).C
6:  reseed_counter ← state(state_handle).reseed_counter
7:  security_strength ← state(state_handle).security_strength
8:  prediction_resistance_flag ← state(state_handle).prediction_resistance_flag
9:  if (requested_no_of_bits > max_no_of_bits_per_request) then
```

```
10:     return (ERROR_FLAG, Null) // ex. ("Too many bits requested" , Null)
11: end if
12: if (requested_strength > security_strength) then
13:     return (ERROR_FLAG, Null) // ex. ("Invalid requested_security_strength" , Null)
14: end if
15: if (len(additional_input) > max_additional_input_length) then
16:     return (ERROR_FLAG, Null) // ex. ("additional_input too long" , Null)
17: end if
18: if ((prediction_resistance_request is set) AND
       (prediction_resistance_flag is not set)) then
19:     return (ERROR_FLAG, Null)
20: end if
21: if ((reseed_counter > reseed_interval) OR
       (prediction_resistance_request is set)) then
22:     if reseeding is not available then
23:         return (ERROR_FLAG, Null)
24:     end if
25:     status ← Reseed_Hash_DRBG(handle, request, additional_input)
                        // handle = state_handle, request = prediction_resistance_request
26:     if (status ≠ SUCCESS) then
27:         return (status, Null)
28:     end if
29:     V ← state(state_handle).V
30:     C ← state(state_handle).C
31:     reseed_counter ← state(state_handle).reseed_counter
32:     additional_input ← Null
33: end if
34: if (additional_input ≠ Null) then
35:     w ← Hash(0x02 ‖ V ‖ additional_input)
36:     V ← (V + w) mod 2^seedlen
37: end if
38: pseudorandom_bits ← Hashgen(requested_no_of_bits, V)
39: H ← Hash(0x03 ‖ V)
40: V ← (V + H + C + reseed_counter) mod 2^seedlen
41: state(state_handle).V ← V
```

42: *state*(*state_handle*).*reseed_counter* ← *reseed_counter* + 1
43: **return** (SUCCESS, *pseudorandom_bits*)

알고리즘 4의 출력값 생성 과정(단계 39)에서 사용되는 함수 Hashgen의 구체적인 동작 방식은 알고리즘 5와 같다.

알고리즘 5 Hashgen(· , ·)

입력: 정수 *requested_no_of_bits*, 비트열 *V*
출력: 비트열 *pseudorandom_bits*

1: $m \leftarrow \lceil requested\_no\_of\_bits / outlen \rceil$
2: $data \leftarrow V$
3: $W \leftarrow Null$
4: **for** i = 1 **to** m **do**
5: $W \leftarrow W \| Hash(data)$
6: $data \leftarrow (data + 1) \bmod 2^{seedlen}$
7: **end for**
8: $pseudorandom\_bits \leftarrow leftmost(W, requested\_no\_of\_bits)$
9: **return** (*pseudorandom_bits*)

생성 함수(알고리즘 4)의 세부 과정을 살펴보면 다음과 같다.

- 입력 파라미터 유효성 확인 (단계 1 - 단계 3, 단계 9 - 단계 20)
- 씨드의 유효 기간이 만료되거나 예측 내성 보장이 필요한 경우 리씨드 함수를 호출하고 내부 상태 갱신 (단계 21 - 단계 33)
- 난수 생성 (단계 34 - 단계 38)
- 내부 상태 갱신 (단계 39 - 단계 42)
- DRBG 운용 주체가 요구한 난수 반환 (단계 43)

DRBG 구현물이 예측 내성을 지원하지 않을 경우, 입력 파라미터 *prediction_resistance_request*를 생략하고 알고리즘 4에서 파라미터 유효성 검증(단계 18 - 단계 20) 과정을 생략한다. 그리고 단계 21과 단계 25에서 *prediction_resistance_request* 파라미터를 생략한다.

DRBG 구현물이 예측 내성을 항상 보장하는 경우 입력 파라미터 *prediction_resistance_request*는 생략할 수 있고, 생략할 경우 이를 반영하여 알고리즘 4를 다음과 같이 수정한다.

- 파라미터 유효성 검증(단계 18 - 단계 20) 생략
- 단계 21: if (*reseed_counter* > *reseed_interval*) then
- 단계 25: *status* ← Reseed_Hash_DRBG(*handle*, *additional_input*)

리씨드 함수는 *prediction_resistance_request* 파라미터의 입력 여부를 반영하여 알고리즘 2의 함수 Get_entropy 호출(단계 6)에서 *prediction_resistance_request* 파라미터를 입력으로 사용하지 않는다.

생성 함수는 출력을 생성하기 전에 리씨드 함수의 수행 여부를 판단하며(단계 21), 아래와 같은 경우에는 출력값 생성 과정 이전에 리씨드 함수를 수행하여 예측 공격을 방지한다.

- 예측 내성을 보장하도록 새로운 엔트로피 입력의 사용이 요구되는 경우
- 출력값 생성 횟수가 상태갱신 주기(reseed_interval)보다 커진 경우

예측 내성을 보장하도록 운용되면서 출력값 생성 횟수가 상태갱신 주기를 초과함에 따라 리씨드 함수를 수행할 경우(단계 21), 리씨드 함수는 엔트로피 소스로부터 확보된 새로운 엔트로피 입력을 사용하여 내부 상태를 재초기화한다(단계 25).

생성 함수 실행 과정에서 리씨드 함수가 호출된 경우에는 갱신 함수 이후의 추가 입력을 *Null* 값으로 설정(단계 32)하여 불필요한 내부 상태 갱신(단계 34 - 단계 37)을 생략한다.

리씨드 함수를 실행하지 않더라도 추가 입력이 있으면 이를 이용하여 내부 상태를 갱신한다(단계 34 - 단계 37). 이를 통해 이전까지의 동작 상태가 공격자에게 알려졌더라도 추가 입력이 알려지지 않았다면 실제 출력을 생성하는데 사용한 동작 상태를 알 수 없게 한다. 또한, 추가 입력이 부가적인 엔트로피를 제공하는 경우, 내부 상태의 엔트로피 저하를 방지한다.

DRBG 출력값 생성 과정(알고리즘 5)은 해시 함수를 카운터 모드로 운용하며 카운터 초기값은 동작 상태 *V*를 이용한다. 출력값 생성에 사용된 동작 상태 *V*를 모르면 출력을 예측할 수 없고, 반대로 출력을 알더라도 해시 함수의 일방향성에 의해 동작 상태 *V*를 알 수 없다.

DRBG 출력값 생성이 끝나면 내부 상태를 갱신하여 다음번 출력값 생성 과정에서 리씨드 함수나 추가 입력을 이용한 내부 상태의 갱신이 일어나지 않더라도 반드시 다른 *V*가 사용되도록 한다.

6.6 인스턴스 소멸 함수(uninstantiate function)

Hash_DRBG의 인스턴스 소멸 함수 Uninstantiate_Hash_DRBG는 인스턴스의 내부 상태를 해제하여 인스턴스가 더 이상 동작할 수 없도록 한다.

인스턴스 소멸 함수의 입력 파라미터는 <표 6-6>과 같다.

<표 6-6> 인스턴스 소멸 함수의 입력 파라미터

파라미터	의미
state_handle	- 해제되는 내부 상태를 가리키는 식별자(핸들)

인스턴스 소멸 함수의 구체적인 동작 예시는 알고리즘 6과 같다.

알고리즘 6 인스턴스 소멸 함수 Uninstantiate_Hash_DRBG(・)

입력: 정수 *state_handle*
출력: 비트열 *status*

1: **if** (a *state*(*state_handle*) is not available) **then**
2: **return** (ERROR_FLAG, *Null*)
3: **end if**
4: *state*(*state_handle*).*V* ← $0^{seedlen}$
5: *state*(*state_handle*).*C* ← $0^{seedlen}$
6: *state*(*state_handle*).*reseed_counter* ← 0
7: *state*(*state_handle*).*security_strength* ← 0
8: *state*(*state_handle*).*prediction_resistance_flag* ← 0
9: **return** (SUCCESS)

부 록 Ⅰ-1

(본 부록은 표준을 보충하기 위한 내용으로 표준의 일부는 아님)

지식재산권 확약서 정보

해당 사항 없음

※ 상기 기재된 지식재산권 확약서 이외에도 본 표준이 발간된 후 접수된 확약서가 있을 수 있으니, TTA 웹사이트에서 확인하시기 바랍니다.

부 록 Ⅰ-2

(본 부록은 표준을 보충하기 위한 내용으로 표준의 일부는 아님)

시험인증 관련 사항

Ⅰ-2.1 시험인증 대상 여부

해당 사항 없음

Ⅰ-2.2 시험표준 제정 현황

해당 사항 없음

부 록 Ⅰ-3

(본 부록은 표준을 보충하기 위한 내용으로 표준의 일부는 아님)

본 표준의 연계(family) 표준

Ⅰ-3.1 TTAK.KO-12.0190-Part2

이 표준에서 제시하는 Hash_DRBG의 기반 해시 함수로 SHA-2[4]를 사용할 경우의 참조 구현값을 제시함

Ⅰ-3.2 TTAK.KO-12.0190-Part3

이 표준에서 제시하는 Hash_DRBG의 기반 해시 함수로 LSH[1]를 사용할 경우의 참조 구현값을 제시함

Ⅰ-3.3 TTAK.KO-12.0191-Part1

해시 함수 기반 메시지 인증 코드 알고리즘인 HMAC을 사용하여 정의된 DRBG 메커니즘인 HMAC_DRBG의 상세 규격을 제시함

Ⅰ-3.4 TTAK.KO-12.0189/R1

블록 암호를 기반으로 정의된 DRBG 메커니즘인 CTR_DRBG의 상세 규격을 정의하고, 국내 개발 주요 블록 암호 알고리즘(SEED, ARIA, HIGHT, LEA)을 기반 함수로 사용하는 경우의 참조 구현값을 제시함

부 록 Ⅰ-4

(본 부록은 표준을 보충하기 위한 내용으로 표준의 일부는 아님)

참고 문헌

[1] TTA TTAK.KO-12.0276, “해시 함수 LSH”.
[2] TTA TTAK.KO-12.0306, “소프트웨어 환경에서의 잡음원 엔트로피 검증 알고리즘”.
[3] ISO/IEC 18031, “Information technology - Security techniques - Random bit generation”, 2011.
[4] NIST FIPS PUB 180-4, “Secure Hash Standard (SHS)”, 2015. 8.
[5] NIST SP 800-90B, “Recommendation for the Entropy Sources Used for Random Bit Generation”, 2018. 1.

※ 상기 기재된 참고 문헌의 발간일이 기재된 경우, 해당 표준(문서)의 해당 버전에 대해서만 유효하며, 연도를 표시하지 않은 경우에는 해당 표준(권고)의 최신 버전을 따름

부 록 Ⅰ-5

(본 부록은 표준을 보충하기 위한 내용으로 표준의 일부는 아님)

영문표준 해설서

해당 사항 없음

부 록 Ⅰ-6

(본 부록은 표준을 보충하기 위한 내용으로 표준의 일부는 아님)

표준의 이력

판수	채택일	표준번호	내용	담당 위원회
제1판	2012.12.21	제정 TTAK.KO-12.0190	-	정보보호기반 (PG501)
제2판	2018.12.xx	개정 TTAK.KO-12.0190 -Part1	① Hash_DRBG의 자체적인 연계 표준 관리를 위해 표준명 변경 ② 개별 해시 함수를 이용한 참조 구현값을 연계 표준으로 분리 ③ 표준 대상인 DRBG 메커니즘에 대한 일반 내용(씨드 관리 포함) 추가 ④ Hash_DRBG 규격 오류 수정 및 상세화	정보보호기반 (PG501)