GVI Part 2
('25. 10.)

암호모듈 구현안내서 Guide for Vendor Implementations

Part 2

검증대상 암호알고리즘 구현안내서

국가정보원 NATIONAL INTELLIGENCE SERVICE

NSR 국가보안기술연구소

GVI Part 2

Guide for Vendor Implementations

Contents

1장 개요 2

2장 블록암호 운영모드(일반) 12

3장 블록암호 운영모드(인증암호화) 28

4장 블록암호 기반 메시지인증코드 42

5장 해시함수 기반 메시지인증코드 54

6장 블록암호 기반 난수발생기 60

7장 해시함수 기반 난수발생기 80

8장 RSA 기반 공개키 암호 100

9장 RSA 기반 전자서명 112

10장 이산대수 기반 전자서명 124

11장 타원곡선 기반 전자서명 136

12장 이산대수 기반 키 설정 152

13장 타원곡선 기반 키 설정 160

14장 패스워드 기반 키 유도 168

15장 의사난수 함수 기반 키 유도 174

부록 186

GVI Part 2

Guide for Vendor Implementations

1장 개요

개 요

1 목적

- 암호모듈 검증제도는 국가·공공기관에서 소통·저장되는 비밀이 아닌 업무자료를 보호하기 위해 사용되는 암호모듈의 안전성과 구현 적합성을 검증하는 제도이다.
- 안내서에서는 「암호알고리즘 검증기준」에 수록된 검증대상 암호알고리즘의 요구사항에 대하여 구현 안전성 관점에서 고려사항을 명세한다.
- 안내서는 암호모듈 검증제도의 검증대상 암호알고리즘별로 안전성에 영향을 미치는 항목을 제공함으로써, 암호모듈에 대한 잠재적인 취약점에 대응할 수 있도록 한다.

2 범위

- 안내서에서 다루는 알고리즘별 검토항목은 암호모듈 검증기준과 시험기준에서 요구하는 보안요구사항(키 관리, 파라미터 형식검증, 예외 처리 등)을 검증대상 암호알고리즘에 적용하여 도출하였다.
- 안내서 내용은 표준 일치성 여부 확인과 암호알고리즘에 대한 보안요구사항 준수여부 확인으로 구성된다.
- 안내서의 알고리즘별 약어 및 기호는 참조표준을 준수하였으며, 알고리즘 참조구현값은 해당 표준을 참고해야 한다.

3 활용대상 및 방법

- 안내서는 암호모듈 개발자와 암호모듈 시험자가 활용할 수 있다.
- 암호모듈 개발자는 암호모듈 개발 시, 안내서를 참고하여 암호알고리즘 구현 안전성 요소를 파악하여 적합하게 적용할 수 있다.
- 암호모듈 시험자는 암호모듈에 포함된 검증대상 암호알고리즘에 대한 구현 안전성 요소를 확인할 수 있다.

4 법적 근거

- 「사이버안보 업무규정」 제9조(사이버보안 예방 조치 등)
- 「전자정부법 시행령」 제69조(전자문서의 보관·유통 관련 보안조치)
- 「암호모듈 시험 및 검증지침」 (국가정보원, 2025)

5 검증대상 암호알고리즘 목록

❑ 다음은 검증기관이 선정한 검증대상 암호알고리즘 목록이다.

구 분	암호알고리즘		참조표준	
블록암호	ARIA	운영모드 ·기밀성(ECB, CBC, OFB, CFB, CTR) ·기밀성/인증(GCM, CCM)	국내	[KS X 1213-1] 128비트 블록 암호 알고리즘 ARIA - 제1부: 일반 (2024) [KS X 1213-2] 128비트 블록 암호 알고리즘 ARIA - 제2부: 운영모드(2024) [KS X ISO/IEC 10116] n비트 블록 암호 운영 모드 (2021) [KS X ISO/IEC 19772] 인증된 암호화 (2024)
			국외	[IETF RFC 5794] A Description of the ARIA Encryption Algorithm (2010) [ISO/IEC 10116] Modes of operation for an n-bit block cipher (2017) [ISO/IEC 19772] Authenticated Encryption (2020)
	SEED	운영모드 ·기밀성(ECB, CBC, OFB, CFB, CTR) ·기밀성/인증(GCM, CCM)	국내	[KS X ISO/IEC 18033-3] 암호 알고리즘 - 제3부: 블록 암호 (2023) [TTAS.KO-12.0004/R1] 128비트 블록암호알고리즘 SEED (2005) [KS X ISO/IEC 10116] n비트 블록 암호 운영 모드 (2021) [KS X ISO/IEC 19772] 인증된 암호화 (2024)
			국외	[ISO/IEC 18033-3] Encryption algorithms Part 3: Block ciphers (2010) [ISO/IEC 10116] Modes of operation for an n-bit block cipher (2017) [ISO/IEC 19772] Authenticated Encryption (2020)
	LEA	운영모드 ·기밀성(ECB, CBC, OFB, CFB, CTR) ·기밀성/인증(GCM, CCM)	국내	[KS X ISO/IEC 29192-2] 경량 암호 - 제2부: 블록 암호 (2024) [KS X ISO/IEC 10116] n비트 블록 암호 운영 모드 (2021) [KS X ISO/IEC 19772] 인증된 암호화 (2024)
			국외	[ISO/IEC 29192-2] Lightweight cryptography Part 2: Block ciphers (2019) [ISO/IEC 10116] Modes of operation for an n-bit block cipher (2017) [ISO/IEC 19772] Authenticated Encryption (2020)
	HIGHT	운영모드 ·기밀성(ECB, CBC, OFB, CFB, CTR)	국내	[KS X ISO/IEC 18033-3] 암호 알고리즘 - 제3부: 블록 암호 (2023) [TTAS.KO-12.0040/R1] 64비트 블록암호 HIGHT (2008) [KS X ISO/IEC 10116] n비트 블록 암호 운영모드 (2021)
			국외	[ISO/IEC 18033-3] Encryption algorithms Part 3: Block ciphers (2010) [ISO/IEC 10116] Modes of operation for an n-bit block cipher (2017)
	AES	운영모드 ·기밀성(ECB, CBC, OFB, CFB, CTR) ·기밀성/인증(GCM, CCM)	국내	[KS X ISO/IEC 18033-3] 암호 알고리즘 - 제3부: 블록 암호 (2023) [KS X ISO/IEC 10116] n비트 블록 암호 운영 모드 (2021) [KS X ISO/IEC 19772] 인증된 암호화 (2024)
			국외	[ISO/IEC 18033-3] Encryption algorithms Part 3: Block ciphers (2010) [ISO/IEC 10116] Modes of operation for an n-bit block cipher (2017) [ISO/IEC 19772] Authenticated Encryption (2020)
해시함수	SHA-2	SHA2-224/256/384/512	국내	[KS X ISO/IEC 10118-3] 해시함수 - 제3부: 전용 해시 함수 (2023)
			국외	[ISO/IEC 10118-3] Hash-functions Part 3: Dedicated hash-functions (2018)
	LSH	LSH-224/256/384/512/ 512-224/512-256	국내	[KS X 3262] 해시 함수 LSH (2023)
	SHA-3	SHA3-224/256/384/512	국내	-
			국외	[ISO/IEC 10118-3] Hash-functions Part 3: Dedicated hash-functions (2018)

구 분	암호알고리즘		참조표준	
메시지 인증	해시함수 기반	HMAC	국내	[KS X ISO/IEC 9797-2] 메시지 인증 코드 - 제2부: 전용 해시 함수를 이용한 메커니즘 (2024) [TTAK.KO-12.0330-Part1] 해시 함수 기반 메시지 인증 코드 (HMAC) - 제1부: 일반 (2018) [TTAK.KO-12.0330-Part2] 해시 함수 기반 메시지 인증 코드 (HMAC) - 제2부: 해시 함수 SHA-2 (2018) [TTAK.KO-12.0330-Part3] 해시 함수 기반 메시지 인증 코드 (HMAC) - 제3부: 해시 함수 LSH (2018) [TTAK.KO-12.0330-Part4] 해시 함수 기반 메시지 인증 코드 (HMAC) - 제4부: 해시 함수 SHA-3 (2019)
			국외	[ISO/IEC 9797-2] Message authentication codes (MACs) Part 2: Mechanisms using a dedicated hash-function (2021)
	블록암호 기반	CMAC, GMAC	국내	[KS X ISO/IEC 9797-1] 메시지 인증 코드 - 제1부: 블록 암호를 이용한 메커니즘 (2023) [KS X ISO/IEC 9797-3] 메시지 인증 코드 - 제3부: 유니버셜 해시 함수를 사용하는 메커니즘 (2024) [KS X ISO/IEC 19772] 인증된 암호화 (2024)
			국외	[ISO/IEC 9797-1] Message Authentication Codes (MACs) - Part 1: Mechanisms using a block cipher (2011) [ISO/IEC 9797-3] Message Authentication Codes (MACs) - Part 3: Mechanisms using a universal hash-function (2011) [ISO/IEC 19772] Authenticated Encryption (2020)
난수발생기	해시함수 기반	Hash-DRBG HMAC-DRBG	국내	[KS X ISO/IEC 18031] 난수발생기 (2023) [TTAK.KO-12.0331-Part1/R1] 해시 함수 기반 결정론적 난수발생기 - 제1부: 일반 (2022) [TTAK.KO-12.0331-Part2] 해시 함수 기반 결정론적 난수발생기 - 제2부: 해시 함수 SHA-2 (2018) [TTAK.KO-12.0331-Part3] 해시 함수 기반 결정론적 난수발생기 - 제3부: 해시 함수 LSH (2018) [TTAK.KO-12.0331-Part4] 해시 함수 기반 결정론적 난수발생기 - 제4부: 해시 함수 SHA-3 (2019) [TTAK.KO-12.0332-Part1/R1] HMAC 기반 결정론적 난수발생기 - 제1부: 일반 (2022) [TTAK.KO-12.0332-Part2] HMAC 기반 결정론적 난수발생기 - 제2부: 해시 함수 SHA-2 (2018) [TTAK.KO-12.0332-Part3] HMAC 기반 결정론적 난수발생기 - 제3부: 해시 함수 LSH (2018) [TTAK.KO-12.0332-Part4] HMAC 기반 결정론적 난수발생기 - 제4부: 해시 함수 SHA-3 (2019)
			국외	[ISO/IEC 18031] Random bit generation (2025)
	블록암호 기반	CTR-DRBG	국내	[KS X ISO/IEC 18031] 난수발생기 (2023) [TTAK.KO-12.0189/R2] 블록 암호 기반 결정론적 난수발생기 (2022)
			국외	[ISO/IEC 18031] Random bit generation (2025)
공개키 암호	RSAES	공개키 길이: 2048, 3072 해시함수: SHA2-224/256	국내	[KS X ISO/IEC 18033-2] 암호 알고리즘 - 제2부: 비대칭형 암호 (2022)
			국외	[ISO/IEC 18033-2] Encryption algorithms Part 2: Asymmetric ciphers (2017) [IETF RFC 8017] PKCS #1: RSA Cryptography Specifications Version 2.2 (2016)

구 분	암호알고리즘		참조표준	
전자서명	RSA-PSS	공개키 길이: 2048, 3072 해시함수: SHA2-224/256	국내	[KS X ISO/IEC 14888-2] 부가형 디지털 서명 - 제2부: 정수 인수분해 기반 메커니즘 (2021)
			국외	[ISO/IEC 14888-2] Digital signatures with appendix Part 2: Integer factorization based mechanisms (2015) [FIPS 186-5] Digital Signature Standard (DSS) (2023) [IETF RFC 8017] PKCS #1: RSA Cryptography Specifications Version 2.2 (2016)
	KCDSA	공개키, 개인키 길이: (2048, 224) (2048, 256) (3072, 256) 해시함수: SHA2-224/256	국내	[KS X ISO/IEC 14888-3] 부가형 디지털 서명 - 제3부: 이산대수 기반 메커니즘 (2021) [TTAK.KO-12.0001/R4] 부가형 전자 서명 방식 표준 - 제2부: 한국형 인증서 기반 전자 서명 알고리즘(KCDSA) (2016)
			국외	[ISO/IEC 14888-3] Digital signatures with appendix Part 3: Descrete logarithm based mechanisms (2018)
	ECDSA	P-224/256, B-233/283, K-233/283 해시함수: SHA2-224/256	국내	[KS X ISO/IEC 14888-3] 부가형 디지털 서명 - 제3부: 이산대수 기반 메커니즘 (2021) [KS X ISO/IEC 15946-1] 타원곡선에 기반한 암호 기술 - 제1부: 일반 (2024) [KS X ISO/IEC 15946-5] 타원곡선에 기초한 암호기법 - 제5부: 타원곡선 생성 (2024)
			국외	[ISO/IEC 14888-3] Digital signatures with appendix Part 3: Descrete logarithm based mechanisms (2018) [ISO/IEC 15946-1] Cryptographic techniques based on elliptic curves Part 1: General (2016) [ISO/IEC 15946-5] Cryptographic techniques based on elliptic curves Part 5: Elliptic curve generation (2022) [FIPS 186-5] Digital Signature Standard (DSS) (2023)
	EC-KCDSA	P-224/256, B-233/283, K-233/283 해시함수: SHA2-224/256	국내	[KS X ISO/IEC 14888-3] 부가형 디지털 서명 - 제3부: 이산대수 기반 메커니즘 (2021) [KS X ISO/IEC 15946-1] 타원곡선에 기반한 암호 기술 - 제1부: 일반 (2024) [KS X ISO/IEC 15946-5] 타원곡선에 기초한 암호기법 - 제5부: 타원곡선 생성 (2024) [TTAK.KO-12.0015/R3] 부가형 전자 서명 방식 표준 - 제3부: 타원 곡선을 이용한 한국형 인증서 기반 전자 서명 알고리즘 (EC-KCDSA) (2016)
			국외	[ISO/IEC 14888-3] Digital signatures with appendix Part 3: Descrete logarithm based mechanisms (2018) [ISO/IEC 15946-1] Cryptographic techniques based on elliptic curves Part 1: General (2016) [ISO/IEC 15946-5] Cryptographic techniques based on elliptic curves Part 5: Elliptic curve generation (2022) [FIPS 186-5] Digital Signature Standard (DSS) (2023)
키 설정	DH	공개키, 개인키 길이: (2048, 224) (2048, 256) (3072, 256)	국내	[KS X ISO/IEC 11770-3] 키 관리 - 제3부: 비대칭 기법을 이용한 메커니즘 (2024) [TTAK.KO-12.0001/R4] 부가형 전자 서명 방식 표준 - 제2부: 한국형 인증서 기반 전자 서명 알고리즘(KCDSA) (2016)
			국외	[ISO/IEC 11770-3] Key management Part 3: Mechanisms using asymmetric techniques (2021)

구 분	암호알고리즘		참조표준	
키 설정	ECDH	P-224/256, B-233/283, K-233/283	국내	[KS X ISO/IEC 11770-3] 키 관리 – 제3부: 비대칭 기법을 이용한 메커니즘 (2024) [KS X ISO/IEC 15946-1] 타원곡선에 기반한 암호 기술 – 제1부: 일반 (2024) [KS X ISO/IEC 15946-5] 타원곡선에 기초한 암호기법 – 제5부: 타원곡선 생성 (2024)
			국외	[ISO/IEC 11770-3] Key management Part 3: Mechanisms using asymmetric techniques (2021) [ISO/IEC 15946-1] Cryptographic techniques based on elliptic curves Part 1: General (2016) [ISO/IEC 15946-5] Cryptographic techniques based on elliptic curves Part 5: Elliptic curve generation (2022) [FIPS 186-5] Digital Signature Standard (DSS) (2023)
키 유도	KBKDF	HMAC, CMAC	국내	[TTAK.KO-12.0333-Part1] HMAC 기반 키 유도 함수 – 제1부: 일반 (2018) [TTAK.KO-12.0333-Part2] HMAC 기반 키 유도 함수 – 제2부: 해시 함수 SHA-2 (2018) [TTAK.KO-12.0333-Part3] HMAC 기반 키 유도 함수 – 제3부: 해시 함수 LSH (2018) [TTAK.KO-12.0333-Part4] HMAC 기반 키 유도 함수 – 제4부: 해시 함수 SHA-3 (2019) [TTAK.KO-12.0272] 블록 암호 기반 키 유도 함수 (2015)
			국외	[ISO/IEC 11770-6] Key management Part 6: Key derivation (2016)
	PBKDF	HMAC	국내	[TTAK.KO-12.0334-Part1] 패스워드 기반 키 유도 함수 – 제1부: 일반 (2018) [TTAK.KO-12.0334-Part2] 패스워드 기반 키 유도 함수 – 제2부: 해시 함수 SHA-2 (2018) [TTAK.KO-12.0334-Part3] 패스워드 기반 키 유도 함수 – 제3부: 해시 함수 LSH (2018) [TTAK.KO-12.0334-Part4] 패스워드 기반 키 유도 함수 – 제4부: 해시 함수 SHA-3 (2019)
			국외	[IETF RFC 8018] PKCS #5: Password-Based Cryptography Specification Version 2.1 (2017)

6 표준 일치

▸ 시험대상 구현물(Implementation Under Test, 이하 IUT)에 구현된 검증대상 암호알고리즘이 관련 표준에 따라 정확하게 구현되었음을 주장하기 위해서는 <암호알고리즘 검증기준>과 <소스코드 및 상세설계서 검토를 통한 검사>에서 다루는 시험들을 통과하여야 한다.

7 문서의 구성

- 안내서는 총 15장으로 구성되며, 1장에서는 안내서의 목적, 법적근거, 그리고 안내서의 대상이 되는 검증대상 알고리즘을 다룬다.
- 2장에서 15장까지는 검증대상 알고리즘에 포함된 각 알고리즘에 대하여, 간략한 소개와 함께 구현 시 고려사항을 기술한다.
- 고려사항은 암호모듈에 입력되는 데이터 처리 메커니즘을 주로 확인하는 입력조건에 대한 항목과 내부 데이터 생성, 제로화, 예외 사항처리 등을 포함한 내부조건에 대한 항목으로 나누어서 기술한다.
- 2장 블록암호 운영모드는 ARIA, SEED, LEA, HIGHT, AES 블록암호 운영모드 ECB, CBC, CTR, CFB, OFB에 대한 구현 시 고려사항을 다룬다.
- 3장 블록암호 운영모드(인증암호화)는 인증암호화 운영모드 GCM, CCM에 대한 구현 시 고려사항을 다룬다.
- 4장 블록암호 기반 메시지인증코드는 블록암호를 내부적으로 이용하는 CMAC, GMAC 알고리즘의 구현 시 고려사항을 다룬다.
- 5장 해시함수 기반 메시지인증코드는 HMAC에 대한 구현 시 고려사항을 다룬다.
- 6장 블록암호 기반 난수발생기는 내부적으로 블록암호(ARIA, SEED, LEA, AES, HIGHT)를 사용하는 난수발생기(CTR_DRBG)의 구현 시 고려사항을 다룬다.
- 7장 해시함수 기반 난수발생기는 해시함수를 사용하는 난수발생기(Hash_DRBG), HMAC을 사용하는 난수발생기(HMAC_DRBG)의 구현 시 고려사항을 다룬다.
- 8장 공개키 암호는 IF(Integer Factorization)에 기반을 둔 RSAES 공개키 암호의 구현 시 고려사항을 다룬다.
- 9장 RSA 기반 전자서명은 IF(Integer Factorization)에 기반을 둔 RSA-PSS 전자서명의 구현 시 고려사항을 다룬다.
- 10장 이산대수 기반 전자서명은 DLP(Discrete Logarithm Progblem)에 기반을 둔 KCDSA 전자서명의 구현 시 고려사항을 다룬다.
- 11장 타원곡선 기반 전자서명은 EC(Elliptic Curve)-DLP에 기반을 둔 ECDSA, EC-KCDSA 전자서명의 구현 시 고려사항을 다룬다.
- 12장 이산대수 기반 키 설정은 DLP(Discrete Logarithm Problem)에 기반을 둔 DH 키 설정 알고리즘의 구현 시 고려사항을 다룬다.
- 13장 타원곡선 기반 키 설정은 EC(Elliptic Curve)-DLP에 기반을 둔 ECDH 키 설정 알고리즘의 구현 시 고려사항을 다룬다.
- 14장 패스워드 기반 키 유도는 HMAC PRF를 사용하는 PBKDF의 구현 시 고려사항을 다룬다.
- 15장 의사난수 함수 기반 키 유도는 HMAC, CMAC PRF를 CTR, FB, DP 모드로 운용하는 KBKDF의 구현 시 고려사항을 다룬다.

8 용어 정의 및 약어

가. 용어 정의

정 의	의 미
난수	생성되기 전에 예측할 수 없는 값으로, 암호시스템에서 난수는 암호 키, 논스, 솔트 등의 목적으로 사용됨
논스	특정 조건하에서 한번만 사용되는 값
평문	암호화 대상이 되는 문자열 및 암호문을 복호화한 본래의 문자열
암호문	평문으로 된 정보를 암호 처리하여 특정인만 이용할 수 있도록 암호화한 문자열
암호화	암호화 키를 사용하여 평문을 암호문으로 변환하는 것
보안 강도	해당 암호알고리즘 또는 암호시스템이 공격자로부터 데이터를 보호할 수 있는 계량화된 수준
복호화	복호화 키를 사용하여 암호문을 본래의 평문으로 복원하는 것
부가인증 데이터	메시지 인증 함수의 입력 데이터로서 인증되지만 암호화되지 않은 데이터 (메시지 인증코드 부분으로 이동)
블록	운영모드의 처리 단위
블록암호 운영모드	블록 암호를 이용하여 임의의 크기를 갖는 데이터를 암/복호화하기 위해 적용하는 방식
비밀키	대칭키 암호알고리즘과 함께 사용되며, 하나 또는 여러 객체에서 유일하게 결합되는 암호키이므로, 공개되어서는 안 됨
씨드	암호함수 또는 난수생성의 초기화를 위해 사용되는 비밀 값
인증 값	우연히 발생한 에러와 데이터의 의도적인 조작을 모두 발견할 수 있도록 설계된 데이터로 해당 비밀키를 소유한 경우에만 확인할 수 있음
엔트로피	데이터가 가지는 정보량을 수치적으로 나타낸 것으로, 난수성에 대한 정량화된 지표
키 유도	하나의 암호 키로부터 대칭키 암호알고리즘의 암호 키들로 사용할 수 있는 키 요소를 얻는 과정
패딩	평문에 특정한 데이터를 채워서 운영모드 블록 크기의 배수로 만드는 기법
핵심보안매개변수	노출 또는 변경 시, 암호알고리즘 또는 암호모듈의 안전성을 크게 저하시키는 정보 (예 : 암호키, 개인키, 패스워드, 난수발생기 내부 상태 등)

나. 약어

약 어	의 미
CBC	Cipher Block Chaining
CFB	Cipher Feedback
PT / CT	Plaintext / Ciphertext
CTR	Counter Mode
CTR_DRBG	Block cipher-based Deterministic Random Bit Generator
CCM	Counter Mode with CBC-MAC
CMAC	Cipher-based Message Authentication Code
ENC / DEC	Encryption / Decryption
DH	Diffie-Hellman
DRBG	Deterministic Random Bit Generator
ECB	Electronic CodeBook
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
EC-KCDSA	Elliptic Curves-Korean Certificate based-Digital Signature Algorithm
EM	Encoded Message
FIPS	Federal Information Processing Standard
GCM	Galois/Counter Mode of Operation
Hash_DRBG	Hash-based Deterministic Random Bit Generator
HMAC	Hash-based Message Authentication Code
HMAC_DRBG	HMAC-based Deterministic Random Bit Generator
IV	Initial Vector
KBKDF	Key-Based Key Derivation Function
KCDSA	Korean Certificate based-Digital Signature Algorithm
MAC	Message Authentication Code
LSH	Lightweight Secure Hash
OFB	Output FeedBack
PBKDF	Password-Based Key Derivation Function
RSAES	RSA Encryption Scheme
RSA-PSS	RSA signature scheme with appendix based on the Probabilistic Signature Scheme
SHA	Secure Hash Algorithm
XOR	Exclusive OR

GVI Part 2

Guide for Vendor Implementations

2장 블록암호 운영모드

(일반)

블록암호 운영모드(일반)

1 범위

❑ 본 문서에서는 블록암호 운영모드 중 암·복호화 전용 운영모드를 구현하기 위한 고려사항을 기술한다.

암·복호화 전용 운영모드
ECB, CBC, CFB, OFB, CTR

2 관련표준

- [KS X ISO/IEC 10116] n비트 블록 암호 운영모드 (2021)
- [ISO/IEC 10116] Modes of operation for an n-bit block cipher (2017)

3 기호

기 호	의 미
$Len(X)$	비트열 X의 비트 길이
$blocklen$	블록암호 알고리즘이 한 번의 연산으로 처리할 수 있는 1블록의 길이 (비트)
$PT[i]$	i번째 평문 블록 ($Len(PT[i]) == blocklen$)
$CT[i]$	i번째 암호문 블록 ($Len(CT[i]) == blocklen$)
Key	암복호화 시 사용하는 키
ENC	암호화 알고리즘
DEC	복호화 알고리즘
IV	CBC, CFB, OFB 운영모드에서 사용되는 초기값
s	CFB 운영모드에서 사용되는 피드백 길이
CTR	CTR 운영모드에서 사용되는 초기값
$inc_n(X)$	비트열 X의 하위 n비트를 1만큼 증가시키는 함수 (상위 $Len(X)-n$비트는 고정) ※ $inc_n(X) = MSB_{Len(X)-n}(X) \| ((LSB_n(X)+1) \bmod 2^n)$
$MSB_i(x)$	주어진 비트열 x의 상위(왼쪽) i개 비트열
$LSB_i(x)$	주어진 비트열 x의 하위(오른쪽) i개 비트열

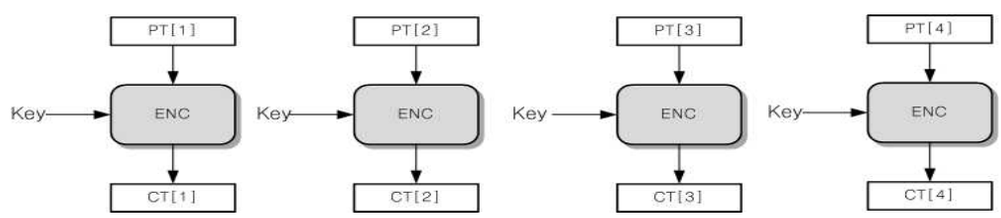
4 블록암호 운영모드

- 블록암호는 블록 단위로 암·복호화를 수행하는 알고리즘이다. 그러나 일반적으로 암호화 하고자 하는 평문의 길이는 한 블록 이상이며 매우 다양한 길이로 구성된다. 이처럼 임의의 길이를 갖는 데이터를 암·복호화하기 위해 입·출력 크기가 고정된 블록암호 알고리즘을 적용하는 방식을 블록암호 알고리즘의 운영모드라고 한다.
- 블록암호의 안전성에 영향을 끼치는 요소로는 비밀키의 길이, IV의 길이, 패딩 기법 등이 있다.

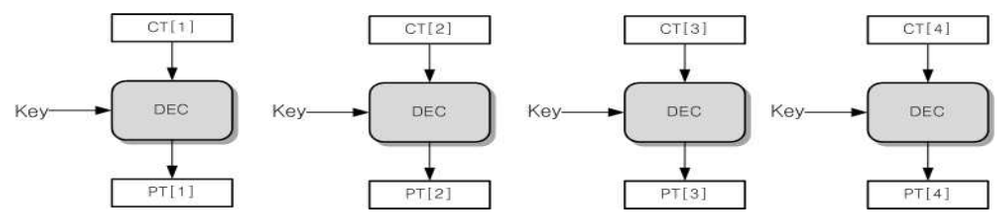
5 운영모드 명세 및 구현 시 고려사항

가. ECB (Electronic CodeBook)

	ECB 운영모드
특징	– 동일한 평문에 대하여 동일한 암호문을 생성함 – 암호화/복호화 모두 병렬처리 가능 – 특정 암호문 블록만 복호화 가능 – 메시지 길이가 1블록 이하인 경우 패딩 필요
권고사항	– 블록암호 알고리즘이 처리할 수 있는 1블록 길이를 초과하는 평문 입력 금지



[ECB 모드 암호화 과정]



[ECB 모드 복호화 과정]

ECB 모드 암호화(*ECB_Enc*)		고려사항
입력	– 평문 $PT = PT[1] \| PT[2] \| \cdots \| PT[m]$ – 비밀키 Key	①, ②, ③, ④
출력	– 암호문 $CT = CT[1] \| CT[2] \| \cdots \| CT[m]$	
1	for i from 1 to m do	
2	$X = PT[i]$	
3	$Y = ENC(X, Key)$	
4	$CT[i] = Y$	
5	end for	

ECB 모드 복호화(*ECB_Dec*)		고려사항
입력	- 암호문 $CT = CT[1] \| CT[2] \| \cdots \| CT[m]$ - 비밀키 Key	①, ③, ④
출력	- 평문 $PT = PT[1] \| PT[2] \| \cdots \| PT[m]$	
1	for i from 1 to m do	
2	$X = CT[i]$	
3	$Y = DEC(X, Key)$	
4	$PT[i] = Y$	
5	end for	

① 비밀키 길이 확인

블록암호별 지원 가능한 키 길이 확인
- ARIA, LEA, AES의 비밀키 길이 : 128/192/256 비트
- SEED, HIGHT의 비밀키 길이 : 128 비트

② 안전한 비밀키 생성

블록암호 비밀키는 “중요보안매개변수(SSP) 생성” 방법으로 생성되어야 함
(검증대상 난수발생기를 이용하여 생성하는 등 안전한 방법으로 생성)

③ 평문/암호문 길이 확인

블록암호별 처리 가능한 평문/암호문 길이 확인
- ARIA, SEED, LEA, AES의 평문 길이 : 128 비트
- HIGHT의 평문 길이 : 64 비트

※ 암호문의 경우 적용된 패딩 알고리즘을 고려하여 처리 가능한 길이를 확인해야 함

④ 패딩 알고리즘

블록암호의 평문은 기본 블록길이의 배수가 되어야 하며, 그렇지 않을 경우 입력되는 평문의 길이가 기본 블록길이의 배수가 되도록 권고된 패딩방법 적용
패딩이 적용된 평문으로부터 생성된 암호문을 복호화할 경우에는, 복호화한 결과에서 패딩방법의 유효성을 검증한 후 패딩값을 제외해야 함

※ [부록] “안전한 패딩 방법” 참고

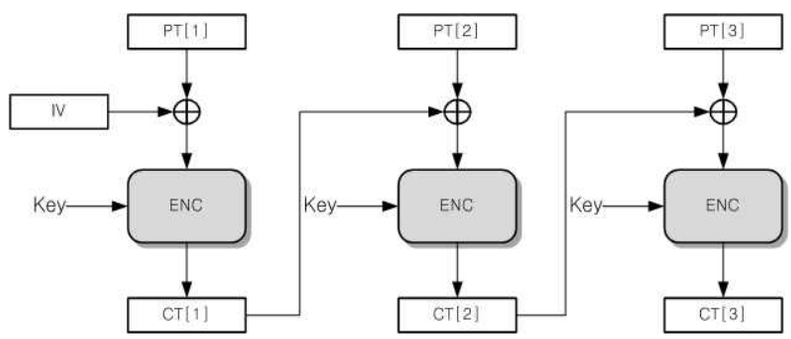
[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

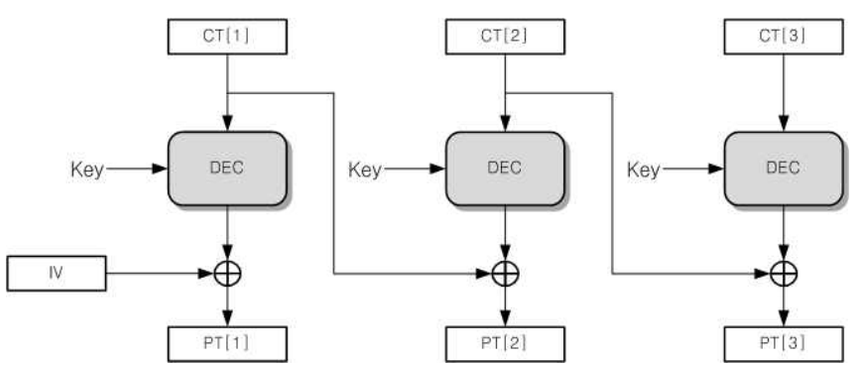
※ [부록] “안전한 제로화” 참고

나. CBC (Cipher Block Chaining)

	CBC 운영모드
특징	– 이전 평문 블록에 대한 암호문 블록이 다음 평문 블록과 XOR된 후 암호화됨 (첫 번째 평문 블록은 IV와 XOR된 후 암호화됨) – 평문, IV의 변조는 암호문 블록에 영향을 미침 – 한 암호문 블록의 변조는 해당 블록과 다음 블록까지 영향을 미침 – 복호화만 병렬처리 가능 – 특정 암호문 블록만 복호화 가능 – 메시지 길이가 1블록 이하인 경우 패딩 필요
권고사항	– IV는 예측 불가능한 값이어야 함



[CBC 모드 암호화 과정]



[CBC 모드 복호화 과정]

CBC 모드 암호화(*CBC_Enc*)		고려사항
입력	– 초기값 IV $(Len(IV) == blocklen)$ – 평문 $PT = PT[1] \| PT[2] \| \cdots \| PT[m]$ – 비밀키 Key	①, ②, ③, ④
출력	– 암호문 $CT = CT[1] \| CT[2] \| \cdots \| CT[m]$	
1	$CT[0] = IV$	
2	for i from 1 to m do	
3	$X = PT[i] \oplus CT[i-1]$	
4	$Y = ENC(X, Key)$	
5	$CT[i] = Y$	
6	end for	

CBC 모드 복호화(*CBC_Dec*)		고려사항
입력	– 초기값 IV $(Len(IV) == blocklen)$ – 암호문 $CT = CT[1] \| CT[2] \| \cdots \| CT[m]$ – 비밀키 Key	①, ③
출력	– 평문 $PT = PT[1] \| PT[2] \| \cdots \| PT[m]$	
1	$CT[0] = IV$	
2	for i from 1 to m do	
3	$X = CT[i]$	
4	$Y = DEC(X, Key)$	
5	$PT[i] = Y \oplus CT[i-1]$	
6	end for	

① 비밀키 길이 확인

블록암호별 지원 가능한 키 길이 확인
- ARIA, LEA, AES의 비밀키 길이 : 128/192/256 비트
- SEED, HIGHT의 비밀키 길이 : 128 비트

② 안전한 비밀키 생성

블록암호 비밀키는 "중요보안매개변수(SSP) 생성" 방법으로 생성되어야 함
(검증대상 난수발생기를 이용하여 생성하는 등 안전한 방법으로 생성)

③ 패딩 알고리즘

블록암호의 평문은 기본 블록길이의 배수가 되어야 하며, 그렇지 않을 경우 입력되는 평문의 길이가 기본 블록길이의 배수가 되도록 권고된 패딩방법 적용
패딩이 적용된 평문으로부터 생성된 암호문을 복호화할 경우에는, 복호화한 결과에서 패딩방법의 유효성을 검증한 후 패딩값을 제외해야 함
※ [부록] "안전한 패딩 방법" 참고

④ 안전한 IV 생성

초기값 IV는 각각의 암호화 단계에서 생성

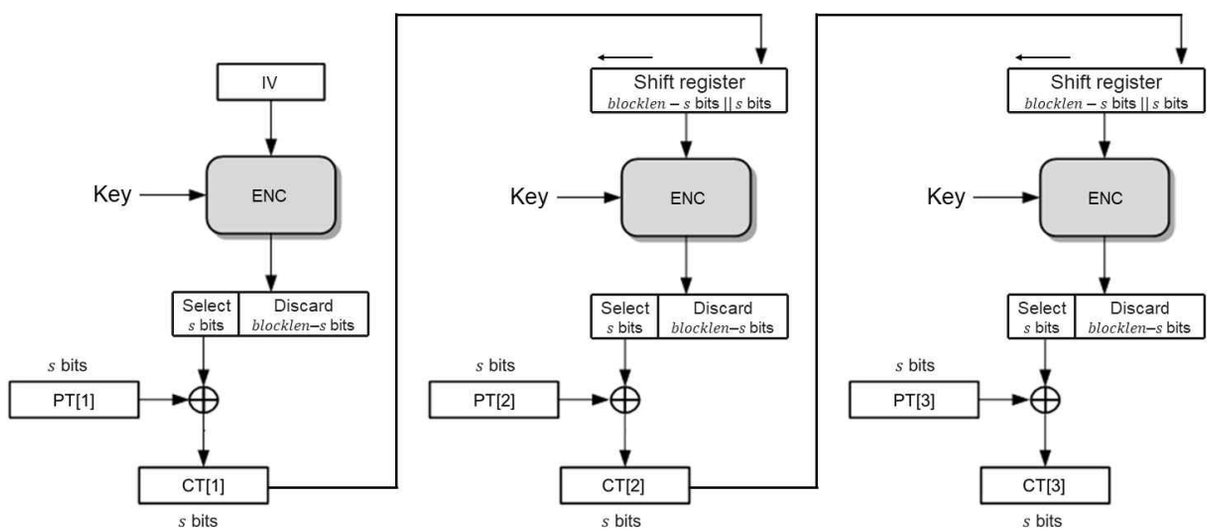
IV는 예측 불가능한 값이어야 하며, 다음 두 가지 방법을 통해 생성해야 함
1) 사용되지 않은(Unique) 값(예: nonce 등)을 암호화에 사용하는 비밀키 Key로 암호화한 결과값을 IV로 사용
2) 검증대상 난수발생기를 이용

[공통] 제로화 수행

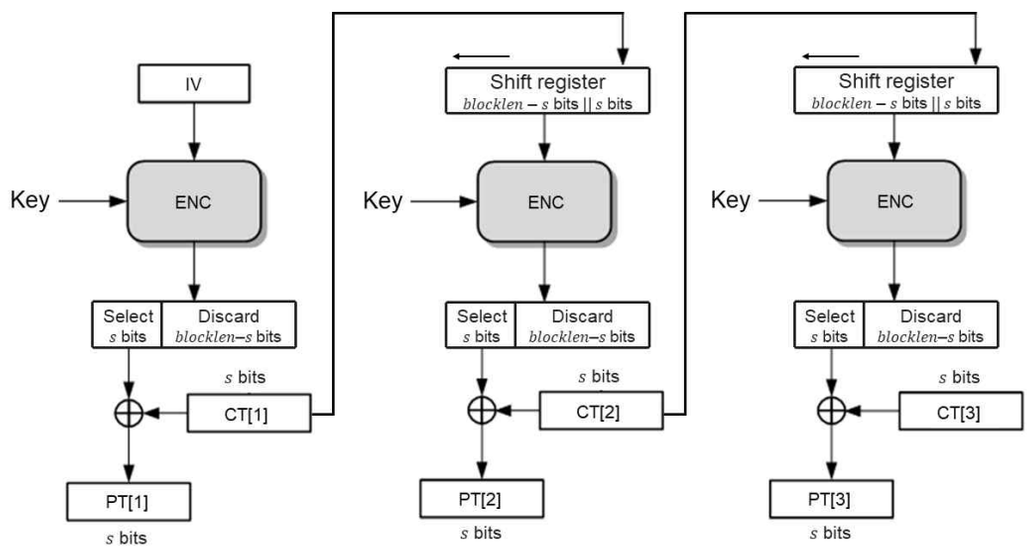
사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] "안전한 제로화" 참고

다. CFB (Cipher Feedback)

	CFB 운영모드
특징	– 이전 평문 블록에 대한 암호문 블록이 다음 암호화의 입력이 됨 – 자가 동기화된 스트림 암호형태이기 때문에 특정 암호문 블록이 소실되는 경우 원본 메시지의 일부분을 복호화하지 못함 – 평문 블록의 변조는 이후의 모든 암호문 블록에 영향을 미침 – 블록암호 알고리즘의 암호화 알고리즘만 구현하면 됨 – 복호화만 병렬처리 가능 – 특정 암호문 블록만 복호화 가능
권고사항	– IV는 예측 불가능한 값이어야 함



[CFB 모드 암호화 과정]



[CFB 모드 복호화 과정]

CFB 모드 암호화(*CFB_Enc*)		고려사항
입력	– 초기값 IV $(Len(IV) == blocklen)$ – 피드백 길이 s – 평문 $PT = PT[1] \| PT[2] \| \cdots \| PT[m]$ $(Len(PT[i]) == s\ (1 \le i \le (m-1)), Len(PT[m]) \le s)$ – 비밀키 Key	①, ②, ③, ④
출력	– 암호문 $CT = CT[1] \| CT[2] \| \cdots \| CT[m]$	
1	$X = IV$	
2	for i from 1 to $(m-1)$ do	
3	$Y = ENC(X, Key)$	
4	$CT[i] = PT[i] \oplus MSB_s(Y)$	
5	$X = LSB_{blocklen-s}(X) \| CT[i]$	
6	end for	
7	$Y = ENC(X, Key)$	
8	$CT[m] = PT[m] \oplus MSB_{Len(PT[m])}(Y)$	

CFB 모드 복호화(*CFB_Dec*)		고려사항
입력	– 초기값 IV $(Len(IV) == blocklen)$ – 피드백 길이 s – 암호문 $CT = CT[1] \| CT[2] \| \cdots \| CT[m]$ $(Len(CT[i]) == s\ (1 \le i \le (m-1)), Len(CT[m]) \le s)$ – 비밀키 Key	①, ④
출력	– 평문 $PT = PT[1] \| PT[2] \| \cdots \| PT[m]$	
1	$X = IV$	
2	for i from 1 to $(m-1)$ do	
3	$Y = ENC(X, Key)$	
4	$PT[i] = CT[i] \oplus MSB_s(Y)$	
5	$X = LSB_{blocklen-s}(X) \| CT[i]$	
6	end for	
7	$Y = ENC(X, Key)$	
8	$PT[m] = CT[m] \oplus MSB_{Len(CT[m])}(Y)$	

① 비밀키 길이 확인

블록암호별 지원 가능한 키 길이 확인
- ARIA, LEA, AES의 비밀키 길이 : 128/192/256 비트
- SEED, HIGHT의 비밀키 길이 : 128 비트

② 안전한 비밀키 생성

블록암호 비밀키는 “중요보안매개변수(SSP) 생성” 방법으로 생성되어야 함 (검증대상 난수발생기를 이용하여 생성하는 등 안전한 방법으로 생성)

③ 안전한 IV 생성

초기값 IV는 각각의 암호화 단계에서 생성
IV는 예측 불가능한 값이어야 하며, 다음 두 가지 방법을 통해 생성해야 함 1) 사용되지 않은(Unique) 값(예: nonce 등)을 암호화에 사용하는 비밀키 Key로 암호화한 결과값을 IV로 사용 2) 검증대상 난수발생기를 이용

④ 피드백 길이 확인

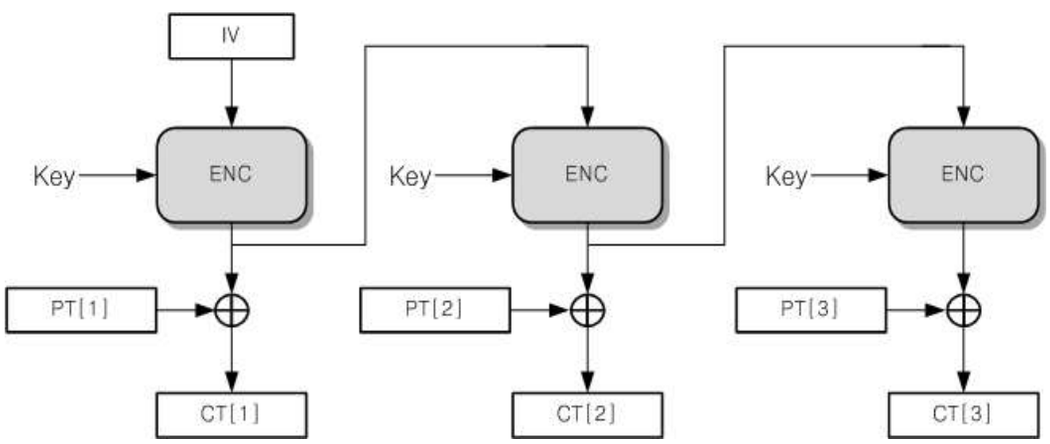
블록암호별 가능한 피드백 길이(비트 단위) 확인 - ARIA, SEED, LEA, AES의 피드백 길이 : $1 \leq s \leq 128$ - HIGHT의 피드백 길이 : $1 \leq s \leq 64$

[공통] 제로화 수행

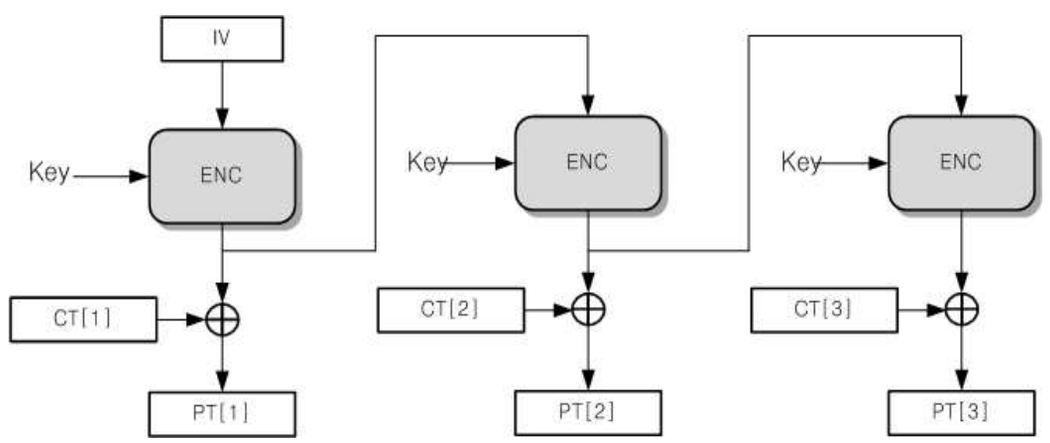
사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] “안전한 제로화” 참고

라. OFB (Output Feedback)

	OFB 운영모드
특징	– 동기화된 스트림 암호와 유사하게 평문은 암호연산의 출력과 XOR되어 암호문이 되며, 암호연산의 출력은 다음 암호연산의 입력이 됨 – 암호문 블록의 특정 비트 오류는 해당 평문 블록의 동일한 위치에서 비트 오류를 발생시킴 – 암호화와 복호화의 구조가 동일 – 블록암호 알고리즘의 암호화 알고리즘만 구현하면 됨 – 병렬처리 불가능 – 특정 암호문 블록 복호화 불가능
권고사항	– IV는 암호 연산의 각 실행에 대해 유일해야 함



[OFB 모드 암호화 과정]



[OFB 모드 복호화 과정]

OFB 모드 암호화(*OFB_Enc*)		고려사항
입력	– 초기값 IV $(Len(IV) == blocklen)$ – 평문 $PT = PT[1] \| PT[2] \| \cdots \| PT[m]$ $(Len(PT[i]) == blocklen\,(1 \le i \le (m-1)),$ $Len(PT[m]) \le blocklen)$ – 비밀키 Key	①, ②, ③
출력	– 암호문 $CT = CT[1] \| CT[2] \| \cdots \| CT[m]$	
1	$X = IV$	
2	for i from 1 to $(m-1)$ do	
3	$Y = ENC(X, Key)$	
4	$CT[i] = PT[i] \oplus Y$	
5	$X = Y$	
6	end for	
7	$Y = ENC(X, Key)$	
8	$CT[m] = PT[m] \oplus MSB_{Len(PT[m])}(Y)$	

OFB 모드 복호화(*OFB_Dec*)		고려사항
입력	– 초기값 IV $(Len(IV) == blocklen)$ – 암호문 $CT = CT[1] \| CT[2] \| \cdots \| CT[m]$ $(Len(CT[i]) == blocklen\,(1 \le i \le (m-1)),$ $Len(CT[m]) \le blocklen)$ – 비밀키 Key	①
출력	– 평문 $PT = PT[1] \| PT[2] \| \cdots \| PT[m]$	
1	$X = IV$	
2	for i from 1 to $(m-1)$ do	
3	$Y = ENC(X, Key)$	
4	$PT[i] = CT[i] \oplus Y$	
5	$X = Y$	
6	end for	
7	$Y = ENC(X, Key)$	
8	$PT[m] = CT[m] \oplus MSB_{Len(CT[m])}(Y)$	

① 비밀키 길이 확인

블록암호별 지원 가능한 키 길이 확인
- ARIA, LEA, AES의 비밀키 길이 : 128/192/256 비트
- SEED, HIGHT의 비밀키 길이 : 128 비트

② 안전한 비밀키 생성

블록암호 비밀키는 “중요보안매개변수(SSP) 생성” 방법으로 생성되어야 함
(검증대상 난수발생기를 이용하여 생성하는 등 안전한 방법으로 생성)

③ 안전한 IV 생성

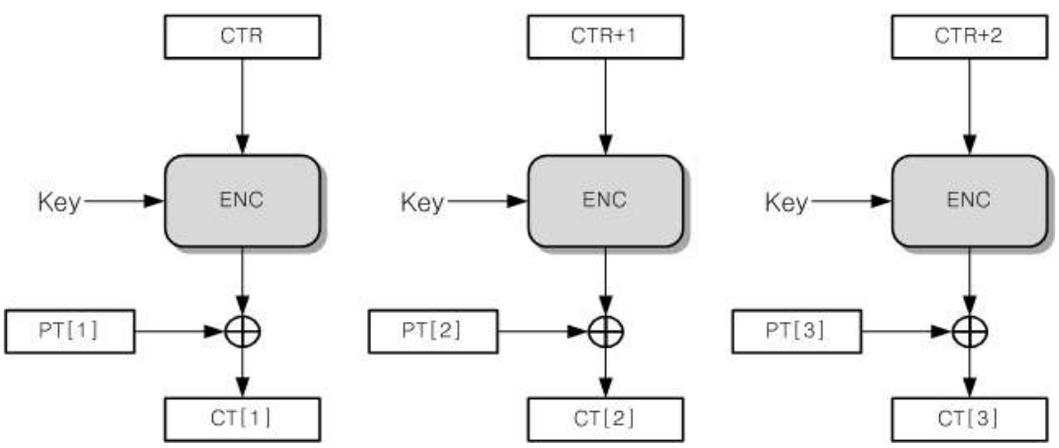
초기값 IV는 각각의 암호화 단계에서 생성
IV는 암호 연산의 각 실행에 대해 유일해야 함

[공통] 제로화 수행

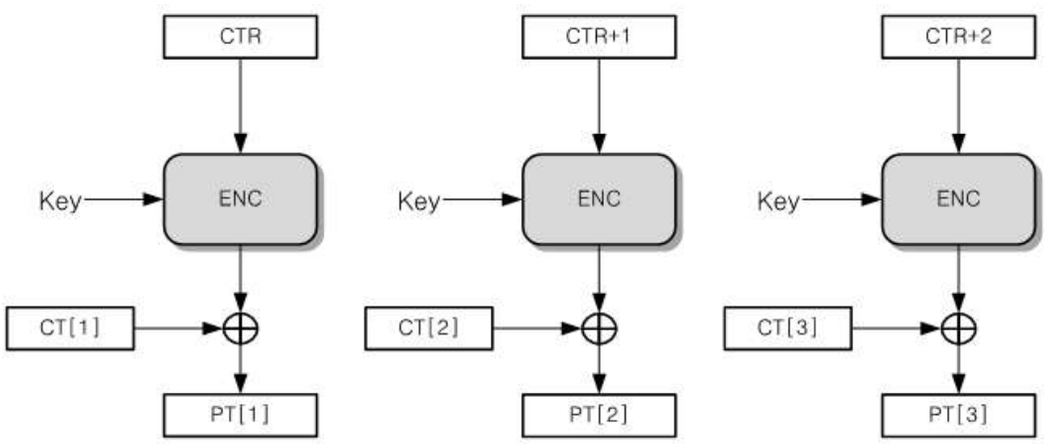
사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] “안전한 제로화” 참고

마. CTR (Counter)

	CTR 운영모드
특징	– 카운터를 암호화한 후 각 평문 블록과 XOR하여 암호문 블록을 계산 – 암호화와 복호화의 구조가 동일 – 블록암호 알고리즘의 암호화 알고리즘만 구현하면 됨 – 암호화/복호화 모두 병렬처리 가능 – 특정 암호문 블록만 복호화 가능
권고사항	– 카운터가 최대값에 도달하였을 때 적절한 처리 필요 – 카운터는 동일한 비밀키로 암호화되는 모든 평문 블록에 대해 유일해야 함



[CTR 모드 암호화 과정]



[CTR 모드 복호화 과정]

CTR 모드 암호화(*CTR_Enc*)		고려사항
설정	– b : CTR 가변 필드의 비트 길이 $(b \le blocklen)$	
입력	– 초기값 CTR $(Len(CTR) == blocklen)$ – 평문 $PT = PT[1] \| PT[2] \| \cdots \| PT[m]$ $(Len(PT[i]) == blocklen\,(1 \le i \le (m-1)), Len(PT[m]) \le blocklen)$ – 비밀키 Key	①, ②, ③
출력	– 암호문 $CT = CT[1] \| CT[2] \| \cdots \| CT[m]$	
1	$CTR_1 = CTR$	
2	for i from 1 to $(m-1)$ do	
3	$Y = ENC(CTR_i, Key)$	
4	$CT[i] = PT[i] \oplus Y$	
5	$CTR_{i+1} = inc_b(CTR_i)$	
6	end for	
7	$Y = ENC(CTR_m, Key)$	
8	$CT[m] = PT[m] \oplus MSB_{Len(PT[m])}(Y)$	

CTR 모드 복호화(*CTR_Dec*)		고려사항
설정	– b : CTR 가변 필드의 비트 길이 $(b \le blocklen)$	
입력	– 초기값 CTR $(Len(CTR) == blocklen)$ – 암호문 $CT = CT[1] \| CT[2] \| \cdots \| CT[m]$ $(Len(CT[i]) == blocklen\,(1 \le i \le (m-1)), Len(CT[m]) \le blocklen)$ – 비밀키 Key	①
출력	– 평문 $PT = PT[1] \| PT[2] \| \cdots \| PT[m]$	
1	$CTR_1 = CTR$	
2	for i from 1 to $(m-1)$ do	
3	$Y = ENC(CTR_i, Key)$	
4	$PT[i] = CT[i] \oplus Y$	
5	$CTR_{i+1} = inc_b(CTR_i)$	
6	end for	
7	$Y = ENC(CTR_m, Key)$	
8	$PT[m] = CT[m] \oplus MSB_{Len(CT[m])}(Y)$	

① 비밀키 길이 확인

블록암호별 지원 가능한 키 길이 확인
- ARIA, LEA, AES의 비밀키 길이 : 128/192/256 비트
- SEED, HIGHT의 비밀키 길이 : 128 비트

② 안전한 비밀키 생성

블록암호 비밀키는 "중요보안매개변수(SSP) 생성" 방법으로 생성되어야 함 (검증대상 난수발생기를 이용하여 생성하는 등 안전한 방법으로 생성)

③ 안전한 CTR 생성

초기값 CTR은 각각의 암호화 단계에서 생성
x번째 블록 연산에 사용되는 CTR_x는 동일한 비밀키로 암호화되는 모든 평문 블록에 대해 유일해야 함

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] "안전한 제로화" 참고

GVI Part 2

Guide for Vendor Implementations

3장 블록암호 운영모드

(인증암호화)

블록암호 운영모드(인증암호화)

1 범위

❑ 본 문서에서는 블록암호 운영모드 중 인증암호화 운영모드(GCM, CCM)를 구현하기 위한 고려사항을 기술한다.

블록암호	인증암호화 운영모드
ARIA, SEED, LEA, AES	GCM, CCM

2 관련표준

- [KS X ISO/IEC 19772] 인증된 암호화 (2024)
- [ISO/IEC 19772] Authenticated Encryption (2020)

3 기호

기 호	의 미
$Len(X)$	비트열 X의 비트 길이
$blocklen$	블록암호 알고리즘이 한 번의 연산으로 처리할 수 있는 1블록의 길이 (비트)
P	평문
A	부가 인증 데이터
K	인증암호화 시 사용하는 비밀키
C	암호문
T	인증값
IV	GCM 운영모드에서 사용되는 초기값
$E_K(X)$	비밀키 K를 이용하여 입력값 X를 암호화하는 블록암호 암호화 함수
0^n	n길이의 연속된 0 비트열 (Ex. $0^5 = 00000$, $0^3 = 000$)
1^n	n길이의 연속된 1 비트열 (Ex. $1^5 = 11111$, $1^3 = 111$)
$MSB_i(X)$	주어진 비트열 X의 상위(왼쪽) i개 비트열
$LSB_i(X)$	주어진 비트열 X의 하위(오른쪽) i개 비트열
$inc_n(X)$	비트열 X의 하위 n비트를 1만큼 증가시키는 함수 (상위 $Len(X)-n$비트는 고정) ※ $inc_n(X) = MSB_{Len(X)-n}(X) \| ((LSB_n(X)+1) \bmod 2^n)$
$\lceil n \rceil$	n보다 크거나 같은 정수 중에서 최소값

기 호	의 미
$[n]_s$	음이 아닌 정수 $n(n < 2^s)$의 s비트 이진수 표현
$CTR_K(A, B)$	초기값(카운터) A, 평문 B, 비밀키 K인 블록암호 CTR 운영모드 암호화 함수
$GHASH_H$	서브키 H를 사용하는 GCM 모드의 내부 해시함수
Mul_X	유한체 $GF(2^{128})$ 상에서 비트열 X와의 곱셈 연산
N	CCM 운영모드에서 사용되는 논스
q	CCM 운영모드에서 평문 P의 길이를 명세하는 첫 번째 블록(B_0)에 포함되는 필드 Q의 바이트 길이

4 블록암호 운영모드(인증암호화)

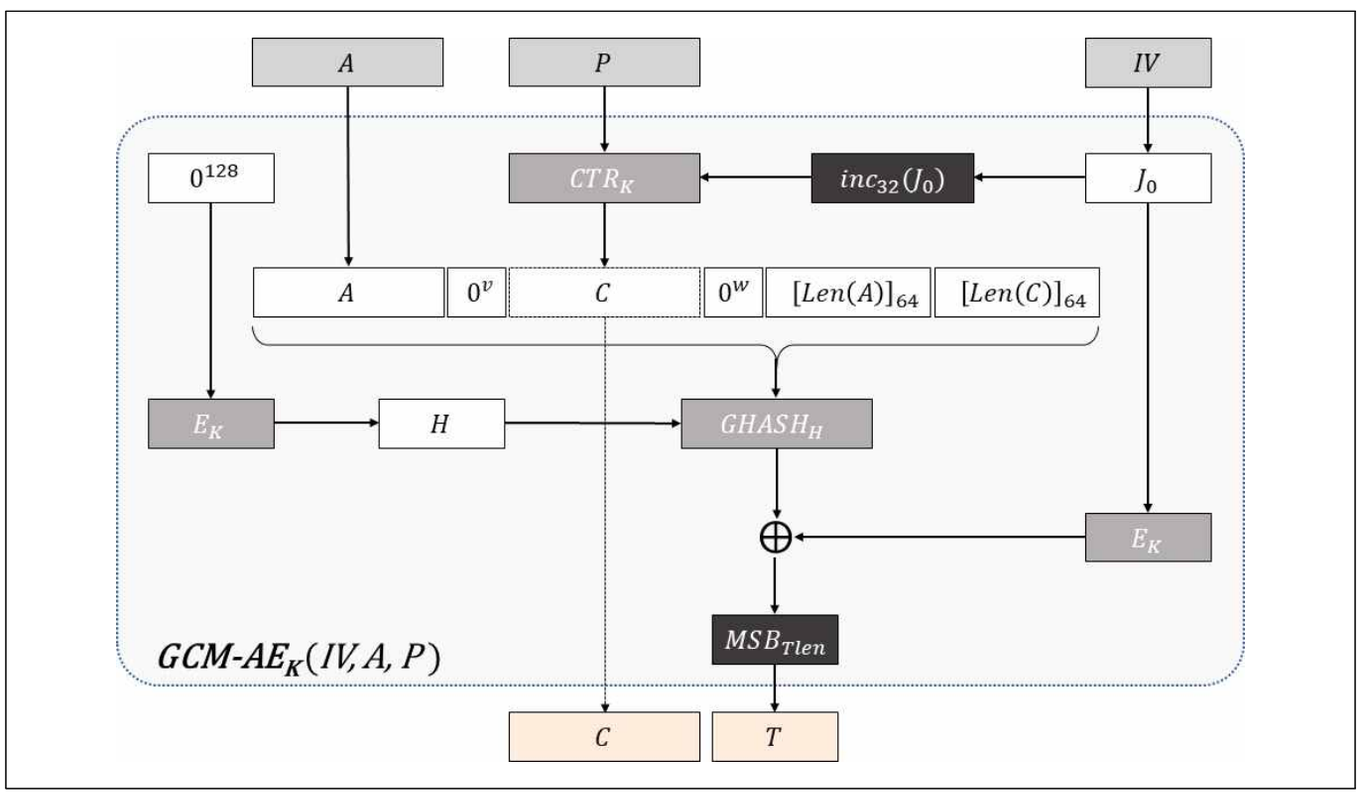
▸ [블록암호 운영모드(일반)]은 메시지의 기밀성만 제공하는 반면, 인증암호화 운영모드는 메시지의 기밀성과 인증 기능을 동시에 제공하는 운영모드로 GCM과 CCM으로 구성된다. 블록암호는 검증대상 암호알고리즘인 ARIA, SEED, LEA, AES를 사용할 수 있다.

5 알고리즘 명세 및 구현 시 고려사항

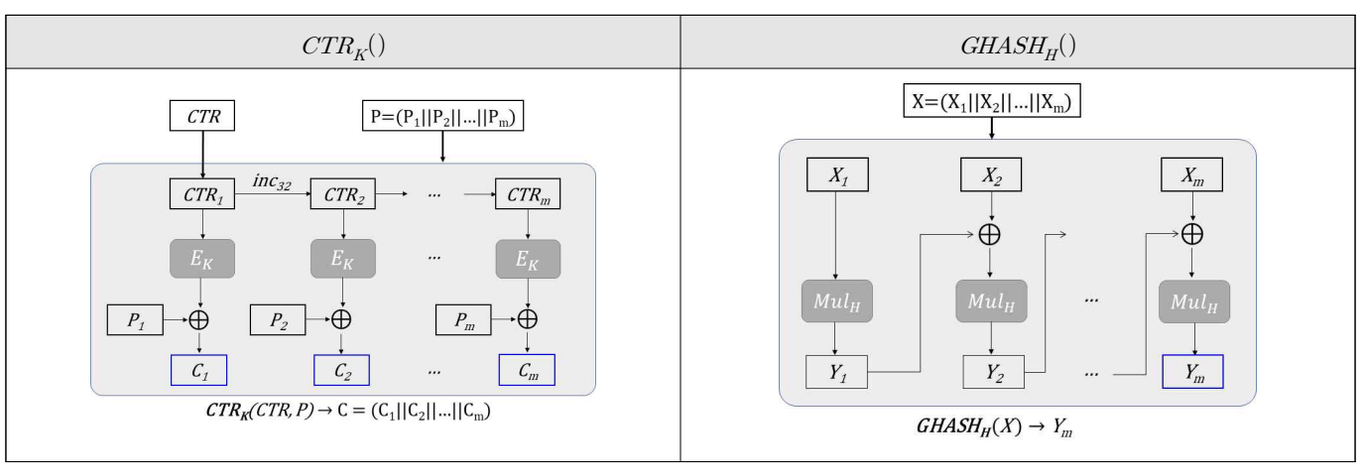
가. GCM

▸ GCM(Galois/Counter Mode)은 CTR 모드에 유한체 GF(2^{128})에서의 곱 연산을 이용한 메시지 인증 기능을 추가한 형태로 메시지의 기밀성과 인증을 동시에 제공한다. 입력 메시지에 대하여 인증만을 제공하기 위해 적용되는 GCM은 GMAC이라 한다. GCM은 AE(Authenticated Encryption)와 AD(Authenticated Decryption) 두 부분으로 구분된다.

□ 암호화 단계 (AE)



■ CTR 및 GHASH 함수



■ Mul_H 연산

$Mul_H : Z \leftarrow X \bullet H$ in $GF(2^{128})$	
1. X 입력 $X = x_0 x_1 \ldots x_{127}$ 2. $R = 1^3 \| 0^4 \| 1 \| 0^{120}$ 3. $Z_0 = 0^{128}$, $V_0 = H$	4. for i from 0 to 127 do $Z_{i+1} = \begin{cases} Z_i & (\text{if } x_i = 0) \\ Z_i \oplus V_i & (\text{if } x_i = 1) \end{cases}$, $V_{i+1} = \begin{cases} V_i \gg 1 & (\text{if } LSB_1(V_i) = 0) \\ (V_i \gg 1) \oplus R & (\text{if } LSB_1(V_i) = 1) \end{cases}$ 5. Z_{128} 출력

GCM 모드 암호화(*GCM-AE*)		고려사항
입력	– 평문 P – 부가 인증 데이터 A – 초기값 IV – 비밀키 K – 인증값 길이 $Tlen$	①, ②, ③
출력	– 암호문 C – 인증값 T ($Len(T) == Tlen$)	
1	$H = E_K(0^{128})$	

2	if($Len(IV) == 96$)
	$J_0 = IV \| 0^{31} \| 1$
	else
	$s = 128 \lceil Len(IV)/128 \rceil - Len(IV)$
	$J_0 = GHASH_H(IV \| 0^{s+64} \| [Len(IV)]_{64})$	

3	$C = CTR_K(inc_{32}(J_0), P)$	
4	$v = 128 \lceil Len(A)/128 \rceil - Len(A)$	
5	$w = 128 \lceil Len(C)/128 \rceil - Len(C)$	
6	$S = GHASH_H(A \| 0^v \| C \| 0^w \| [Len(A)]_{64} \| [Len(C)]_{64})$	
7	$T = MSB_{Tlen}(E_K(J_0) \oplus S)$	
8	(암호문 C, 인증값 T) 출력	

① 입력값 길이 확인

GCM 운영모드에서 허용되는 입력값 길이 확인

- 평문 P : $1 \le Len(P) \le 2^{39} - 256$ 비트
- 부가 인증 데이터 A : $0 \le Len(A) \le 2^{64} - 1$ 비트
- 초기값 IV : $1 \le Len(IV) \le 2^{64} - 1$ 비트
- 인증값 길이 $Tlen$: $112 \le Tlen \le 128$ 비트

② 비밀키 생성 방법 및 길이 확인

비밀키는 “중요보안매개변수(SSP) 생성” 방법으로 생성되어야 함 (검증대상 난수발생기를 이용하여 생성하는 등 안전한 방법으로 생성)
블록암호별 지원 가능한 키 길이 확인 - ARIA, LEA, AES의 비밀키 길이 : 128/192/256 비트 - SEED의 비밀키 길이 : 128 비트

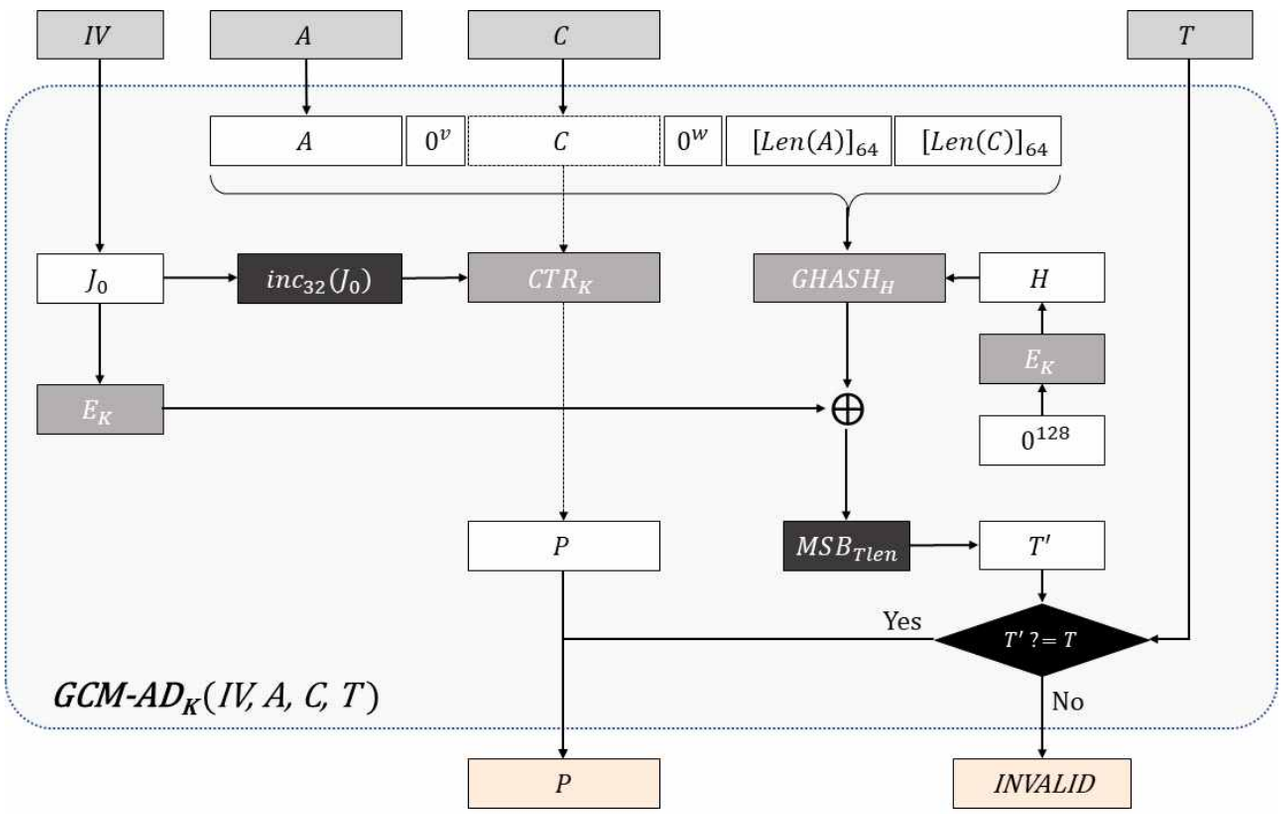
③ 안전한 IV 생성 및 관리

암호모듈 구현안내서(GVI Part 1) C.1 참조

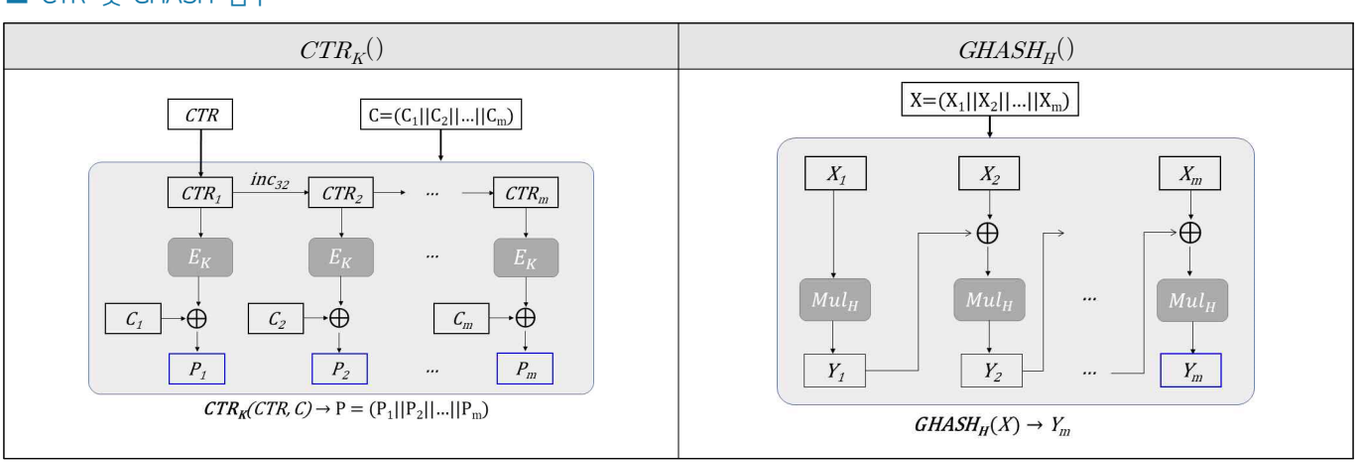
[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] “안전한 제로화” 참고

□ 복호화 단계 (AD)



■ CTR 및 GHASH 함수



■ Mul_H 연산

$Mul_H : Z \leftarrow X \bullet H$ in $GF(2^{128})$

1. X 입력
   $X = x_0 x_1 \cdots x_{127}$
2. $R = 1^3 \| 0^4 \| 1 \| 0^{120}$
3. $Z_0 = 0^{128}$, $V_0 = H$
4. for i from 0 to 127 do
   $$Z_{i+1} = \begin{cases} Z_i & (\text{if } x_i = 0) \\ Z_i \oplus V_i & (\text{if } x_i = 1) \end{cases}, \quad V_{i+1} = \begin{cases} V_i \gg 1 & (\text{if } LSB_1(V_i) = 0) \\ (V_i \gg 1) \oplus R & (\text{if } LSB_1(V_i) = 1) \end{cases}$$
5. Z_{128} 출력

GCM 모드 복호화(*GCM-AD*)		고려사항
입력	– 암호문 C – 부가 인증 데이터 A – 초기값 IV – 비밀키 K – 인증값 T	①, ②
출력	– 평문 P 또는 $FAIL$	
1	$H = E_K(0^{128})$	
2	if($Len(IV) == 96$) $J_0 = IV \| 0^{31} \| 1$ $else$ $s = 128 \lceil Len(IV)/128 \rceil - Len(IV)$ $J_0 = GHASH_H(IV \| 0^{s+64} \| [Len(IV)]_{64})$	
3	$P = CTR_K(inc_{32}(J_0), C)$	③
4	$v = 128 \lceil Len(A)/128 \rceil - Len(A)$	
5	$w = 128 \lceil Len(C)/128 \rceil - Len(C)$	
6	$S = GHASH_H(A \| 0^v \| C \| 0^w \| [Len(A)]_{64} \| [Len(C)]_{64})$	
7	$T' = MSB_{Len(T)}(E_K(J_0) \oplus S)$	
8	if($T == T'$) 평문 P 출력 $else$ $FAIL$ 출력	

① 입력값 길이 확인

GCM 운영모드에서 허용되는 입력값 길이 확인
- 암호문 C : $1 \le Len(C) \le 2^{39} - 256$ 비트
- 부가 인증 데이터 A : $0 \le Len(A) \le 2^{64} - 1$ 비트
- 초기값 IV : $1 \le Len(IV) \le 2^{64} - 1$ 비트
- 인증값 T : $112 \le Len(T) \le 128$ 비트

② 비밀키 길이 확인

블록암호별 지원 가능한 키 길이 확인
- ARIA, LEA, AES의 비밀키 길이 : 128/192/256 비트
- SEED의 비밀키 길이 : 128 비트

③ 복호화 수행 시점

구현 효율성을 고려하여 [단계 8]에서 if($T == T'$) 조건을 만족한 경우,
해당 시점에 [단계 3]을 수행하여 평문 P 출력 가능

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] "안전한 제로화" 참고

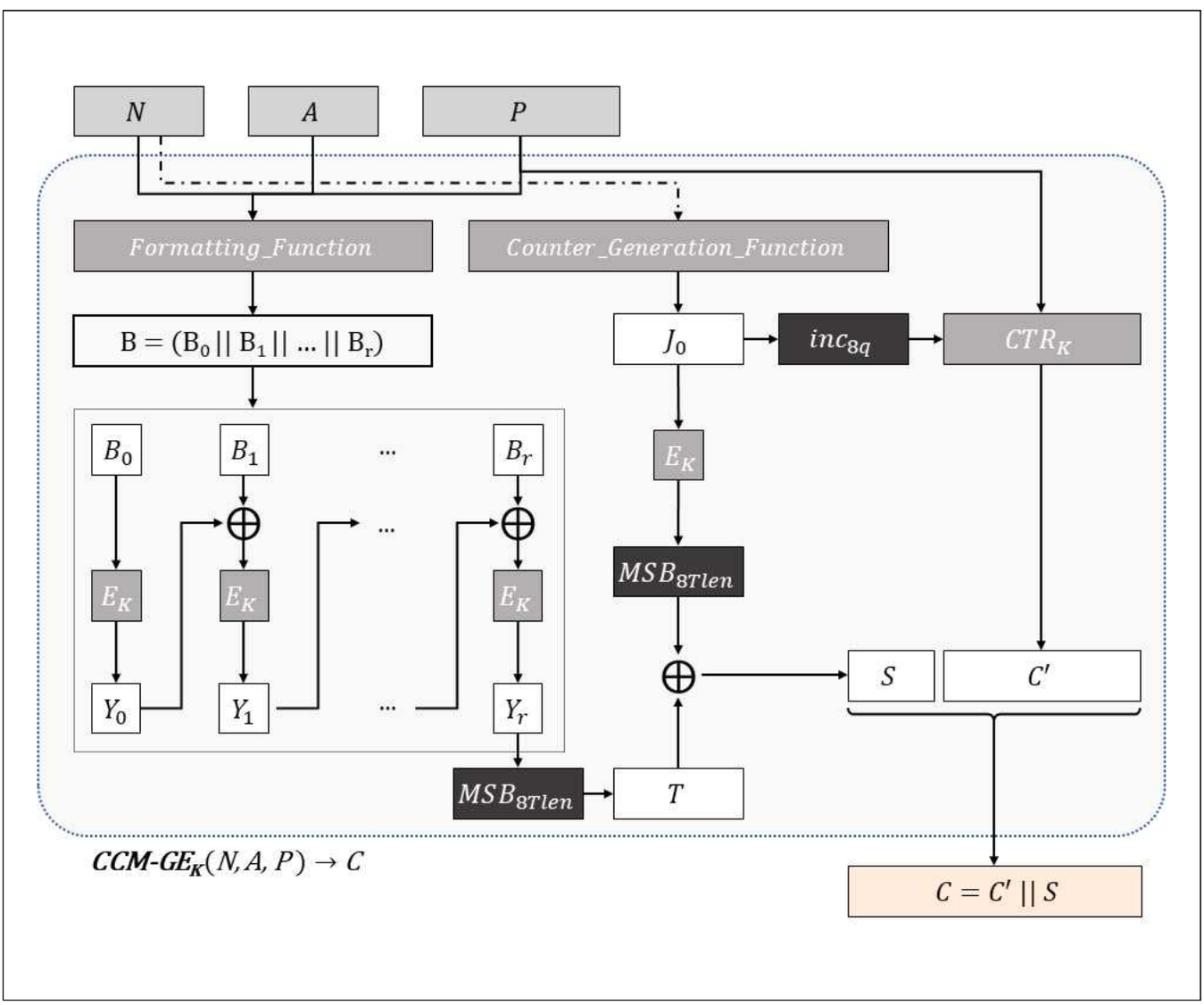
나. CCM

▸ CCM(Counter Mode with CBC-MAC)은 CTR 모드와 CBC-MAC 메시지 인증 방식을 결합한 구조이다. CCM은 인증값 생성 및 암호화(GE : Generation Encryption)와 인증값 검증 및 복호화(DV : Decryption Verification) 두 부분으로 구분된다.

▸ CCM 모드의 Formatting Function 및 Counter Generation Function에는 요구조건을 만족하는 다양한 방법이 적용될 수 있으나, 본 구현안내서에서는 IEEE Standard 802.11의 CCM 명세를 적용한다.

※ IEEE Standard 802.11 (Part 11)

□ 암호화 단계



Formatting_Function		고려사항
설정	– q : 필드 Q의 바이트 길이 – t : 인증값의 바이트 길이	
입력	– 평문 P – 부가 인증 데이터 A – 논스 N	
출력	– 인코딩된 비트열 B	
1	$\text{if}(A == Null)$ $\quad flag = 0 \| 0 \| [(t-2)/2]_3 \| [q-1]_3$ $else$ $\quad flag = 0 \| 1 \| [(t-2)/2]_3 \| [q-1]_3$	
2	$p = Len(P)/8$ $a = Len(A)/8$	
3	$B = flag \| N \| [p]_{8q}$	
4	$\text{if}(A != Null)$ $\quad \text{if}(0 < alen < 65{,}280)$ $\quad\quad B = B \| [a]_{16} \| A \| 0^{128-8x}, x = (a+2) \bmod 16$ $\quad else\,\text{if}(65{,}280 \le alen < 2^{32})$ $\quad\quad B = B \| 1^{15} \| 0 \| [a]_{32} \| A \| 0^{128-8x}, x = (a+6) \bmod 16$ $\quad else\,\text{if}(2^{32} \le alen < 2^{64})$ $\quad\quad B = B \| 1^{16} \| [a]_{64} \| A \| 0^{128-8x}, x = (a+10) \bmod 16$	
5	$B = B \| P \| 0^{128-8x}, x = p \bmod 16$	
6	비트열 B 출력	

Counter_Generation_Function		고려사항
설정	– q : 필드 Q의 바이트 길이	
입력	– 논스 N	
출력	– 카운터 비트열 J	
1	$flag = 0^5 \| [q-1]_3$	
2	$J = flag \| N \| [0]_{8q}$	
3	비트열 J 출력	

CCM 모드 암호화(*CCM-GE*)		고려사항
입력	– 평문 P – 부가 인증 데이터 A – 논스 N – 비밀키 K – 인증값 바이트 길이 $Tlen$	①, ③
출력	– 암호문 C	
1	$B = B_0 \| B_1 \| \cdots \| B_r = Formatting\_Function(N, A, P)$ $(Len(B_i) == blocklen)$	②
2	$Y_0 = E_K(B_0)$	
3	for i from 1 to r do	
4	$Y_i = E_K(B_i \oplus Y_{i-1})$	
5	end for	
6	$T = MSB_{8Tlen}(Y_r)$	
7	$J_0 = Counter\_Generation\_Function(N)$	
8	$S = T \oplus MSB_{8Tlen}(E_K(J_0))$	
9	$C' = CTR_K(inc_{8q}(J_0), P)$	
10	$C = C' \| S$	
11	암호문 C 출력	

① 비밀키 생성 방법 및 길이 확인

비밀키는 "중요보안매개변수(SSP) 생성" 방법으로 생성되어야 함 (검증대상 난수발생기를 이용하여 생성하는 등 안전한 방법으로 생성)
블록암호별 지원 가능한 키 길이 확인 – ARIA, LEA, AES의 비밀키 길이 : 128/192/256 비트 – SEED의 비밀키 길이 : 128 비트

② Formatting Function 파라미터 확인

Formatting Function의 파라미터가 아래의 조건을 만족하는지 확인 – 평문 P의 길이를 명세하는 첫 번째 블록(B_0)에 포함된 필드 Q의 바이트 길이(q) : $q \in \{2, 3, 4, 5, 6, 7, 8\}$ – 논스 N의 바이트 길이(n) : $n \in \{7, 8, 9, 10, 11, 12, 13\}$ – $n + q = 15$

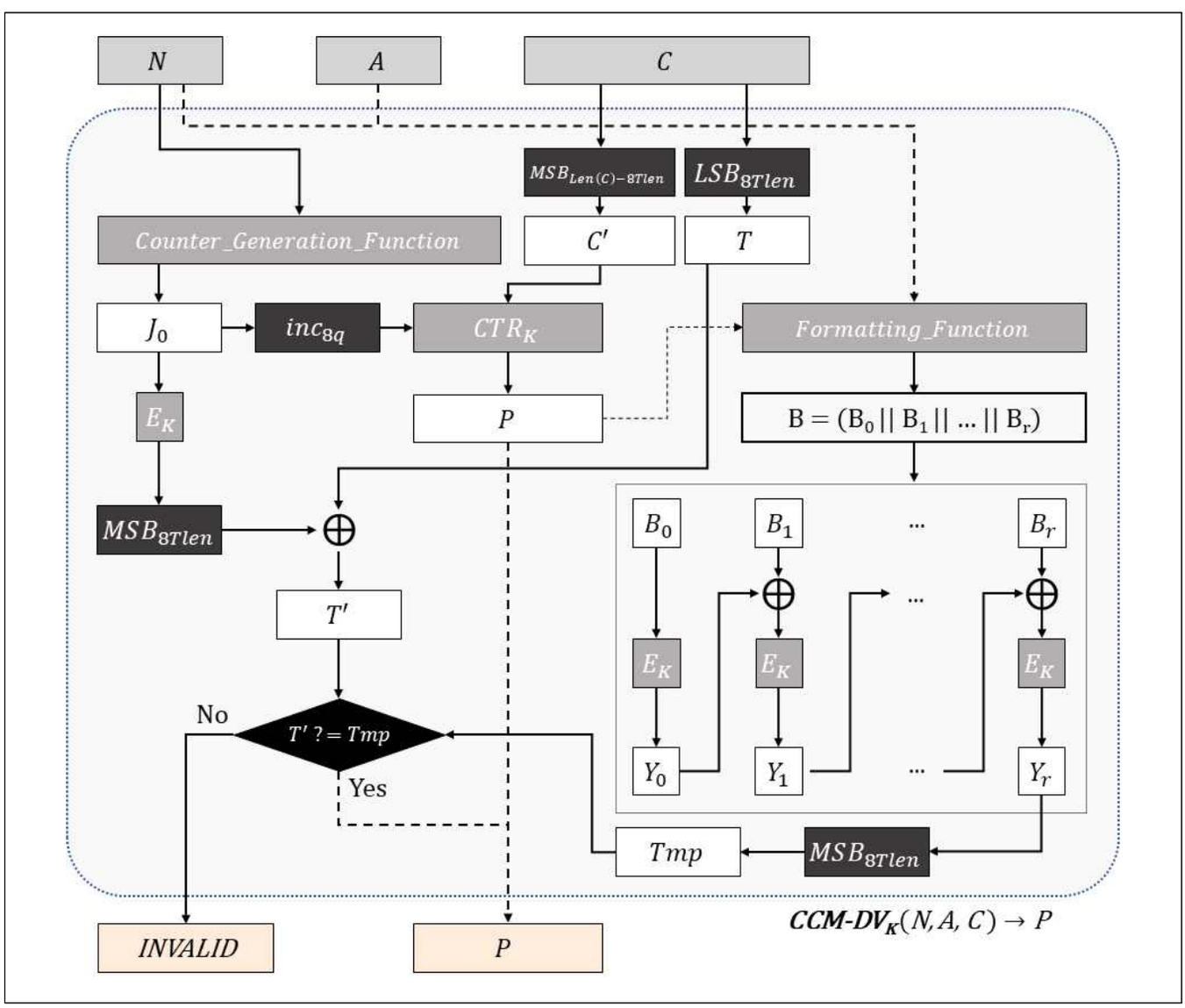
③ 입력값 길이 확인

CCM 운영모드에서 허용되는 입력값 길이 확인
- 평문 P의 바이트 길이(p) : $1 \leq p \leq 2^{8q}-1$ 바이트
- 부가 인증 데이터 A의 바이트 길이(a) : $0 \leq a \leq 2^{64}-1$ 바이트
- 논스 N : $Len(N)/8 == n$
- 인증값 바이트 길이 $Tlen$: (14 또는 16) 바이트

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] "안전한 제로화" 참고

□ 복호화 단계



CCM 모드 복호화(*CCM-DV*)		고려사항
입력	– 암호문 C – 부가 인증 데이터 A – 논스 N – 비밀키 K – 인증값 바이트 길이 $Tlen$	①, ③
출력	– 평문 P 또는 $INVALID$	
1	if($Len(C) \leq 8Tlen$)인 경우, $INVALID$ 출력	
2	$J_0 = Counter\_Generation\_Function(N)$	
3	$T = LSB_{8Tlen}(C)$	
4	$C' = MSB_{Len(C)-8Tlen}(C)$	
5	$T' = T \oplus MSB_{8Tlen}(E_K(J_0))$	
6	$P = CTR_K(inc_{8q}(J_0), C')$	
7	$B = B_0 \| B_1 \| \cdots \| B_r = Formatting\_Function(N, A, P)$ $(Len(B_i) == blocklen)$	②
8	$Y_0 = E_K(B_0)$	
9	for i from 1 to r do	
10	$Y_i = E_K(B_i \oplus Y_{i-1})$	
11	end for	
12	if($T' == MSB_{8Tlen}(Y_r)$) 평문 P 출력 $else$ $INVALID$ 출력	

① 비밀키 길이 확인

블록암호별 지원 가능한 키 길이 확인
- ARIA, LEA, AES의 비밀키 길이 : 128/192/256 비트
- SEED의 비밀키 길이 : 128 비트

② Formatting Function 파라미터 확인

Formatting Function의 파라미터가 아래의 조건을 만족하는지 확인
- 평문 P의 길이를 명세하는 첫 번째 블록(B_0)에 포함된 필드 Q의 바이트 길이(q)
  : $q \in \{2, 3, 4, 5, 6, 7, 8\}$
- 논스 N의 바이트 길이(n) : $n \in \{7, 8, 9, 10, 11, 12, 13\}$
- $n + q = 15$

③ 입력값 길이 확인

CCM 운영모드에서 허용되는 입력값 길이 확인

- 암호문 C의 바이트 길이(c) : $1 \leq c \leq 2^{8q} + Tlen - 1$ 바이트
- 부가 인증 데이터 A의 바이트 길이(a) : $0 \leq a \leq 2^{64} - 1$ 바이트
- 논스 N : $Len(N)/8 == n$
- 인증값 바이트 길이 $Tlen$: (14 또는 16) 바이트

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] “안전한 제로화” 참고

4장 블록암호 기반 메시지인증코드

블록암호 기반 메시지인증코드

1 범위

❑ 본 문서에서는 블록암호 기반 메시지인증코드(MAC)의 구현 시 고려사항을 기술한다.

❑ 블록암호별 적용되는 메시지인증코드는 다음과 같다.

블록암호	적용 가능한 MAC
ARIA, SEED, LEA, AES	GMAC, CMAC
HIGHT	CMAC

2 관련표준

- [KS X ISO/IEC 9797-1] 메시지 인증 코드 – 제1부: 블록 암호를 이용한 메커니즘 (2023)
- [KS X ISO/IEC 9797-3] 메시지 인증 코드 – 제3부: 유니버셜 해시 함수를 사용하는 메커니즘 (2024)
- [KS X ISO/IEC 19772] 인증된 암호화 (2024)
- [ISO/IEC 9797-1] Message Authentication Codes (MACs) - Part 1: Mechanisms using a block cipher (2011)
- [ISO/IEC 9797-3] Message Authentication Codes (MACs) - Part 3: Mechanisms using a universal hash-function (2011)
- [ISO/IEC 19772] Authenticated Encryption (2020)

3 기호

기 호	의 미
$Len(X)$	비트열 X의 비트 길이
$blocklen$	블록암호 알고리즘이 한 번의 연산으로 처리할 수 있는 1블록의 길이 (비트)
M	평문 메시지
K	비밀키
IV	초기값
MAC	메시지인증코드
m	메시지인증코드 길이(비트)
$E_K(X)$	비밀키 K를 이용하여 입력값 X를 암호화하는 블록암호 암호화 함수
0^n	n길이의 연속된 0 비트열 (Ex. $0^5 = 00000$, $0^3 = 000$)
1^n	n길이의 연속된 1 비트열 (Ex. $1^5 = 11111$, $1^3 = 111$)
$inc_n(X)$	비트열 X의 하위 n비트를 1만큼 증가시키는 함수 (상위 $Len(X)-n$비트는 고정) ※ $inc_n(X) = MSB_{Len(X)-n}(X) \| ((LSB_n(X)+1) \bmod 2^n)$
$\lceil n \rceil$	n보다 크거나 같은 정수 중에서 최소값
$[n]_s$	음이 아닌 정수 $n\,(n < 2^s)$의 s비트 이진수 표현
$CTR_K(A, B)$	초기값 A, 평문 B, 비밀키 K인 블록암호 CTR 모드 암호화 함수
$GHASH_H$	서브키 H를 사용하는 GCM 모드의 내부 해시함수
$MSB_i(X)$	주어진 비트열 X의 상위(왼쪽) i개 비트열
Mul_X	유한체 $GF(2^{128})$ 상에서 비트열 X와의 곱셈 연산

4 블록암호 기반 메시지 인증코드

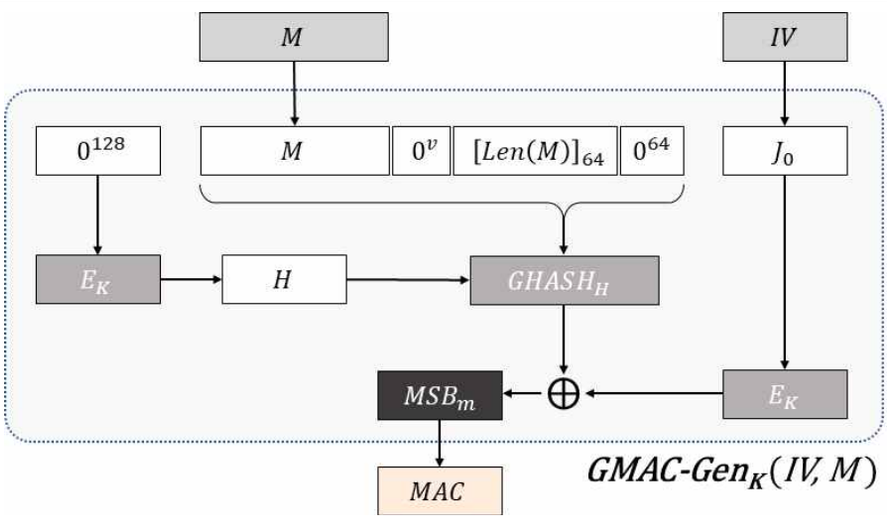
▸ 블록암호 기반 메시지 인증코드는 메시지 인증 기능을 제공하는 블록암호 운영모드 중 GMAC, CMAC으로 구성되며, 블록암호는 검증대상 암호알고리즘인 ARIA, SEED, LEA, AES를 사용할 수 있다. 블록암호 HIGHT는 CMAC에만 적용할 수 있다.

5 알고리즘 명세 및 구현 시 고려사항

가. GMAC

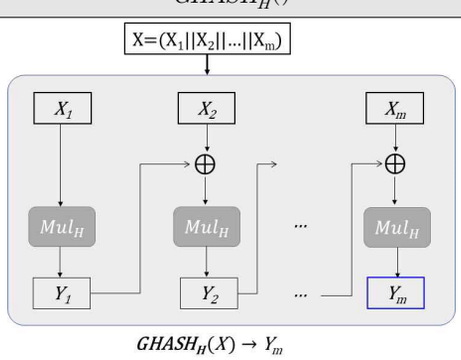
▸ GCM(Galois/Counter Mode)은 CTR 모드에 유한체 GF(2^{128})에서의 곱 연산을 이용한 메시지 인증 기능을 추가한 형태로 메시지의 기밀성과 인증을 동시에 제공한다. 입력 메시지에 대하여 인증만을 제공하기 위해 적용되는 GCM을 GMAC이라 한다.

□ MAC 생성 단계



■ GHASH 함수

$GHASH_H()$



■ Mul_H 연산

$Mul_H : Z \leftarrow X \bullet H$ in $GF(2^{128})$

1. X 입력
   $X = x_0 x_1 \dots x_{127}$
2. $R = 1^3 \| 0^4 \| 1 \| 0^{120}$
3. $Z_0 = 0^{128}$, $V_0 = H$
4. for i from 0 to 127 do
   $$Z_{i+1} = \begin{cases} Z_i & (\text{if } x_i = 0) \\ Z_i \oplus V_i & (\text{if } x_i = 1) \end{cases}, \quad V_{i+1} = \begin{cases} V_i \gg 1 & (\text{if } LSB_1(V_i) = 0) \\ (V_i \gg 1) \oplus R & (\text{if } LSB_1(V_i) = 1) \end{cases}$$
5. Z_{128} 출력

GMAC 생성 함수(*GMAC-Gen*)		고려사항
입력	– 메시지 M – 초기값 IV – 비밀키 K – 메시지인증코드 길이 m	①, ②, ③
출력	– 길이가 m인 메시지인증코드 MAC	
1	$H = E_K(0^{128})$	
2	if($Len(IV) = 96$) $J_0 = IV \| 0^{31} \| 1$ $else$ $s = 128 \lceil Len(IV)/128 \rceil - Len(IV)$ $J_0 = GHASH_H(IV \| 0^{s+64} \| [Len(IV)]_{64})$	
3	$v = 128 \lceil Len(M)/128 \rceil - Len(M)$	
4	$S = GHASH_H(M \| 0^v \| [Len(M)]_{64} \| [0]_{64})$	
5	$MAC = MSB_m(E_K(J_0) \oplus S))$	
6	메시지인증코드 MAC 출력	

① 입력값 길이 확인

GMAC 운영모드에서 허용되는 입력값 길이 확인
- 메시지 M : $0 \le Len(M) \le 2^{64}-1$ 비트
- 초기값 IV : $1 \le Len(IV) \le 2^{64}-1$ 비트
- 메시지인증코드 길이 m : $112 \le m \le 128$ 비트

② 비밀키 생성 방법 및 길이 확인

블록암호 비밀키는 "중요보안매개변수(SSP) 생성" 방법으로 생성되어야 함
(검증대상 난수발생기를 이용하여 생성하는 등 안전한 방법으로 생성)

블록암호별 지원 가능한 키 길이 확인
- ARIA, LEA, AES의 비밀키 길이 : 128/192/256 비트
- SEED의 비밀키 길이 : 128 비트

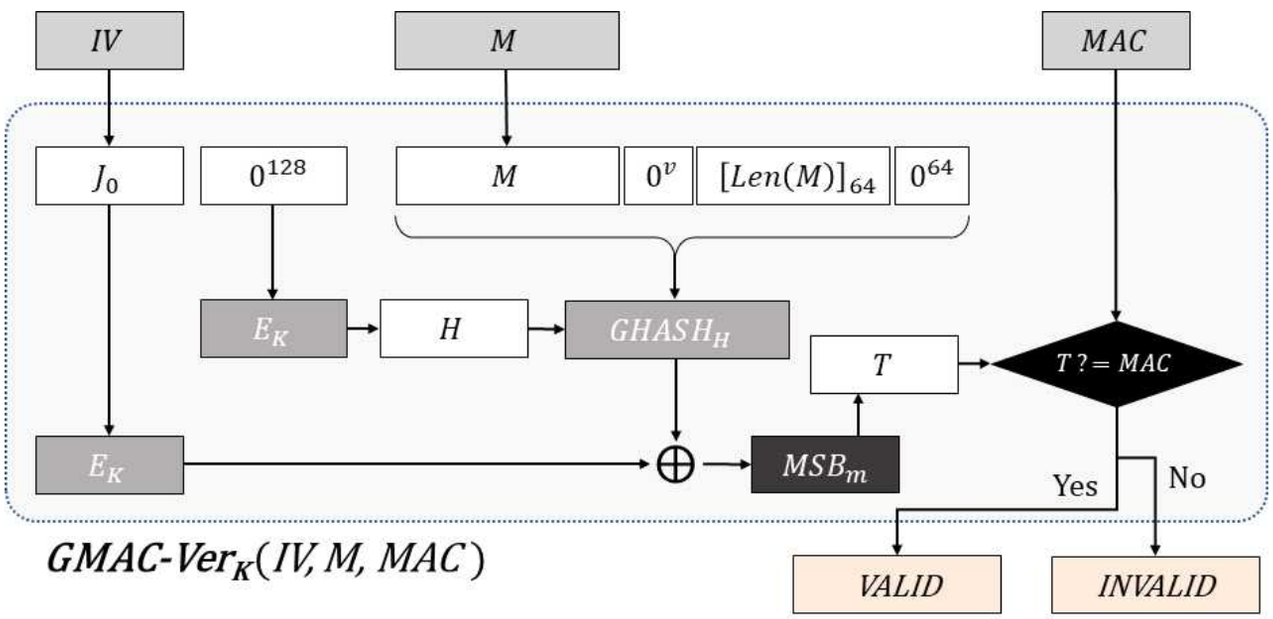
③ 안전한 IV 생성 및 관리

암호모듈 구현안내서(GVI Part 1) C.1 참조

[공통] 제로화 수행

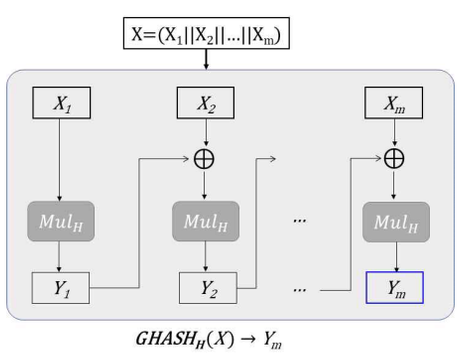
사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] "안전한 제로화" 참고

□ MAC 검증 단계



■ GHASH 함수

$GHASH_H()$



■ Mul_H 연산

$Mul_H: Z \leftarrow X \bullet H$ in $GF(2^{128})$	
1. X 입력 $X = x_0 x_1 \ldots x_{127}$ 2. $R = 1^3 \| 0^4 \| 1 \| 0^{120}$ 3. $Z_0 = 0^{128}$, $V_0 = H$	4. for i from 0 to 127 do $Z_{i+1} = \begin{cases} Z_i & (\text{if } x_i = 0) \\ Z_i \oplus V_i & (\text{if } x_i = 1) \end{cases}$, $V_{i+1} = \begin{cases} V_i \gg 1 & (\text{if } LSB_1(V_i) = 0) \\ (V_i \gg 1) \oplus R & (\text{if } LSB_1(V_i) = 1) \end{cases}$ 5. Z_{128} 출력

GMAC 검증 함수(*GMAC-Ver*)		고려사항
입력	– 메시지 M – 초기값 IV – 비밀키 K – 메시지인증코드 MAC	①, ②
출력	– $VALID$ 또는 $INVALID$	
1	$H = E_K(0^{128})$	
2	if($Len(IV) = 96$) $J_0 = IV \| 0^{31} \| 1$ $else$ $s = 128 \lceil Len(IV)/128 \rceil - Len(IV)$ $J_0 = GHASH_H(IV \| 0^{s+64} \| [Len(IV)]_{64})$	
3	$v = 128 \lceil Len(M)/128 \rceil - Len(M)$	
4	$S = GHASH_H(M \| 0^v \| [Len(M)]_{64} \| [0]_{64})$	
5	$T = MSB_{Len(MAC)}(E_K(J_0) \oplus S))$	
6	if($MAC == T$) $VALID$ 출력 $else$ $INVALID$ 출력	

① 입력값 길이 확인

GMAC 운영모드에서 허용되는 입력값 길이 확인
- 메시지 M : $0 \le Len(M) \le 2^{64} - 1$ 비트
- 초기값 IV : $1 \le Len(IV) \le 2^{64} - 1$ 비트
- 메시지인증코드 MAC : $112 \le Len(MAC) \le 128$ 비트

② 비밀키 길이 확인

블록암호별 지원 가능한 키 길이 확인
- ARIA, LEA, AES의 비밀키 길이 : 128/192/256 비트
- SEED의 비밀키 길이 : 128 비트

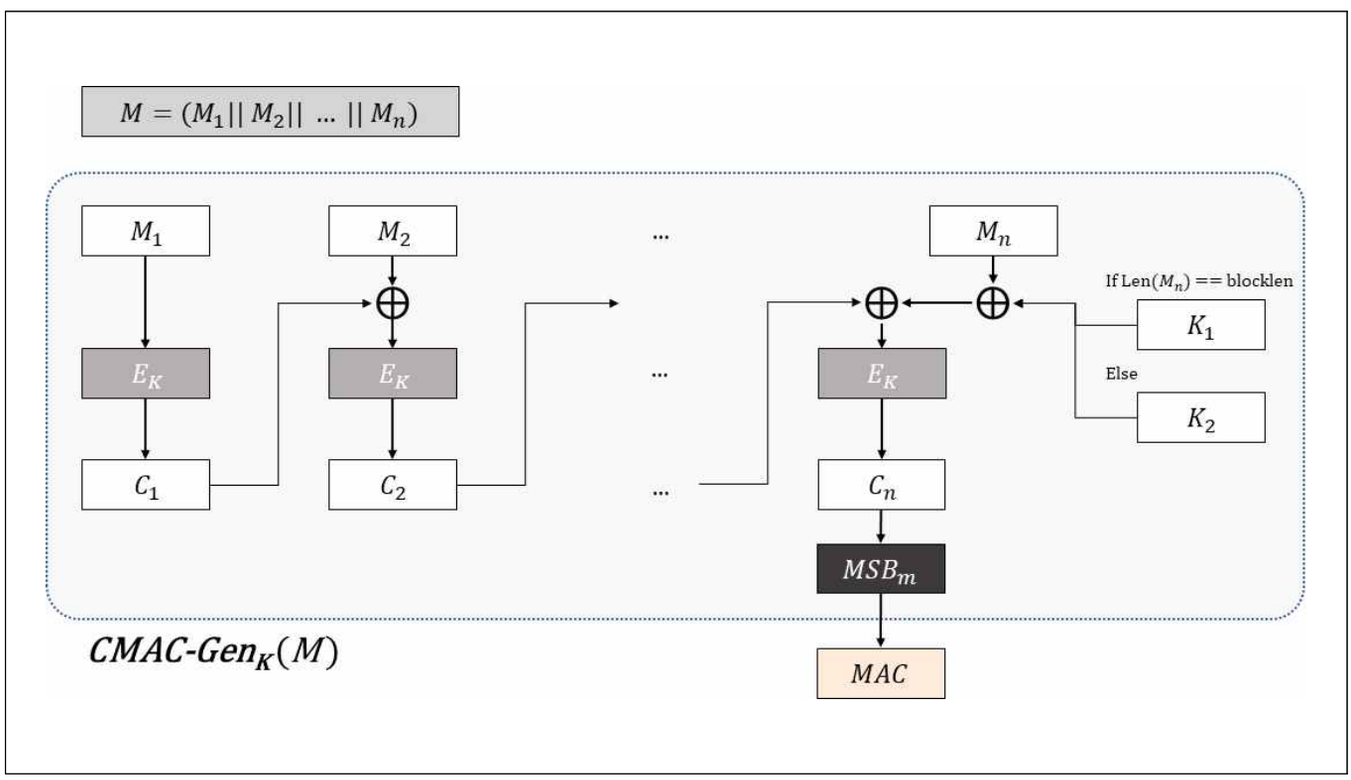
[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] “안전한 제로화” 참고

나. CMAC

▸ CMAC(Cipher-based Message Authentication Code)은 CBC-MAC의 한 변형이다. CMAC은 인증값의 생성(Generation)과 검증(Verification) 두 부분으로 구분된다.

□ MAC 생성 단계



CMAC 생성 함수(*CMAC-Gen*)		고려사항
입력	– 메시지 $M = M_1 \| M_2 \| \cdots \| M_n$ $(Len(M_i) == blocklen (1 \le i \le (n-1)), Len(M_n) \le blocklen)$ – 비밀키 K – 메시지인증코드 길이 m	①, ②
출력	– 길이가 m인 메시지인증코드 MAC	
1	$L = E_K(0^{blocklen})$	
2	if $(blocklen == 128)$ $R = 0^{120} \| 1 \| 0^4 \| 1^3$ $else$ if $(blocklen == 64)$ $R = 0^{59} \| 1^2 \| 0 \| 1^2$	
	보조 비밀키 K_1 생성	
3	if $(MSB_1(L) == 0)$ $K_1 = L \ll 1$ $else$ $K_1 = (L \ll 1) \oplus R$	

	보조 비밀키 K_2 생성	
4	if$(MSB_1(K_1) == 0)$ $K_2 = K_1 \ll 1$ $else$ $K_2 = (K_1 \ll 1) \oplus R$	
5	if$(Len(M) == 0)$ $n = 1$ $else$ $n = \lceil Len(M)/blocklen \rceil$	
6	if$(Len(M_n) == blocklen)$ $M_n = M_n \oplus K_1$ $else$ $M_n = (M_n \| 1 \| 0^j) \oplus K_2, j = blocklen - Len(M_n) - 1$	
7	$C_0 = 0^{blocklen}$	
8	for i from 1 to n do	
9	$C_i = E_K(M_i \oplus C_{i-1})$	
10	end for	
11	$MAC = MSB_m(C_n)$	
12	메시지인증코드 MAC 출력	

① 입력값 길이 확인

블록암호별 지원 가능한 키 및 MAC 길이(m) 확인
- ARIA, LEA, AES의 비밀키 길이 : 128/192/256 비트 - SEED, HIGHT의 비밀키 길이 : 128 비트
- ARIA, SEED, LEA, AES의 MAC 길이 : $112 \leq m \leq 128$ 비트 - HIGHT의 MAC 길이 : $m == 64$ 비트

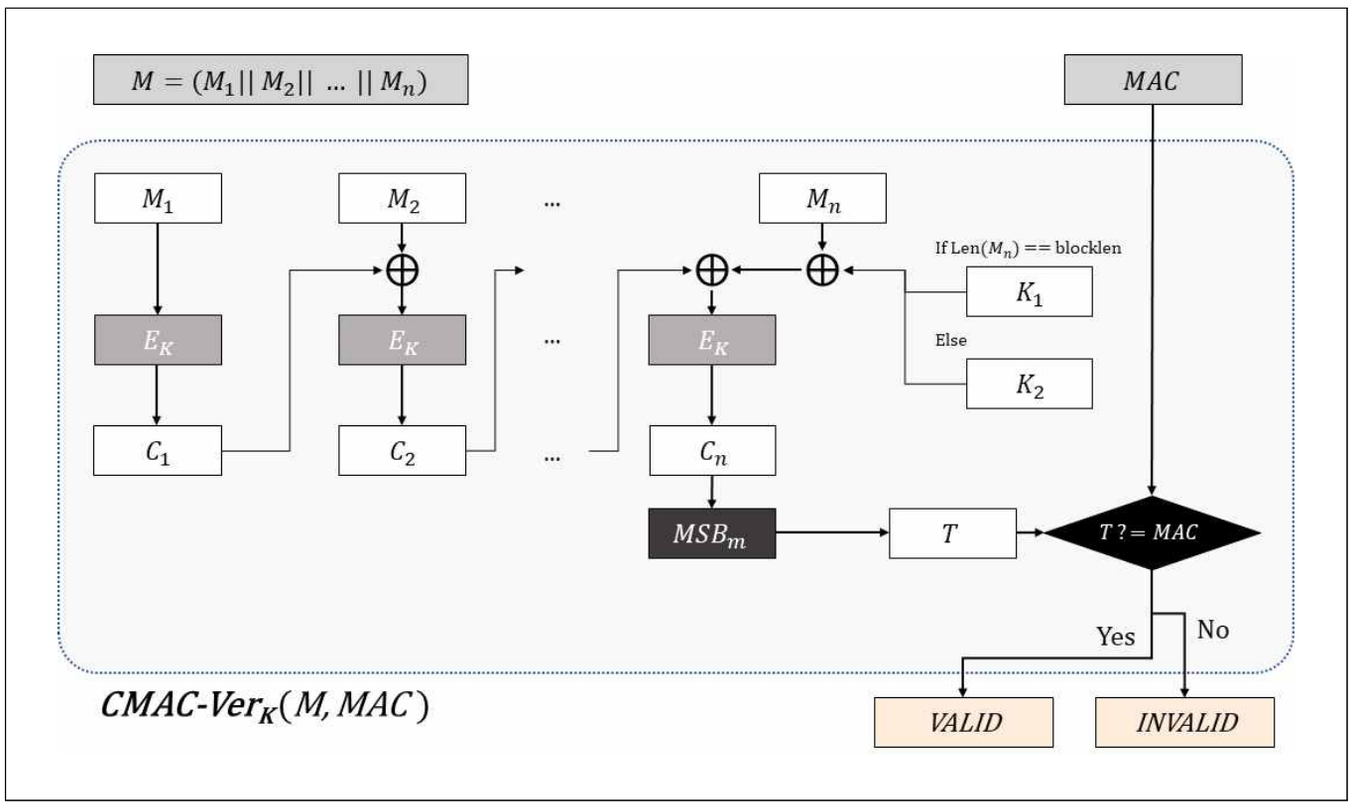
② 안전한 비밀키 생성

비밀키는 "중요보안매개변수(SSP) 생성" 방법으로 생성되어야 함 (검증대상 난수발생기를 이용하여 생성하는 등 안전한 방법으로 생성)

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] "안전한 제로화" 참고

□ MAC 검증 단계



CMAC 검증 함수(*CMAC-Ver*)		고려사항
입력	– 메시지 $M = M_1 \| M_2 \| \cdots \| M_n$ $(Len(M_i) == blocklen(1 \le i \le (n-1)), Len(M_n) \le blocklen)$ – 비밀키 K – 메시지인증코드 MAC	①, ②
출력	– $VALID$ 또는 $INVALID$	
1	$L = E_K(0^{blocklen})$	
2	if $(blocklen == 128)$ $R = 0^{120} \| 1 \| 0^4 \| 1^3$ $else$ if $(blocklen == 64)$ $R = 0^{59} \| 1^2 \| 0 \| 1^2$	
	보조 비밀키 K_1 생성	
3	if $(MSB_1(L) == 0)$ $K_1 = L \ll 1$ $else$ $K_1 = (L \ll 1) \oplus R$	
	보조 비밀키 K_2 생성	
4	if $(MSB_1(K_1) == 0)$ $K_2 = K_1 \ll 1$ $else$ $K_2 = (K_1 \ll 1) \oplus R$	

5	if($Len(M) == 0$)
	  $n = 1$
	$else$
	  $n = \lceil Len(M)/blocklen \rceil$	

6	if($Len(M_n) == blocklen$)
	  $M_n = M_n \oplus K_1$
	$else$
	  $M_n = (M_n \| 1 \| 0^j) \oplus K_2, j = blocklen - Len(M_n) - 1$	

7	$C_0 = 0^{blocklen}$	
8	for i from 1 to n do	
9	$C_i = E_K(M_i \oplus C_{i-1})$	
10	end for	
11	$T = MSB_{Len(MAC)}(C_n)$	

12	if($MAC == T$)
	  $VALID$ 출력
	$else$
	  $INVALID$ 출력	

① 비밀키 길이 확인

블록암호별 지원 가능한 키 길이 확인
- ARIA, LEA, AES의 비밀키 길이 : 128/192/256 비트
- SEED, HIGHT의 비밀키 길이 : 128 비트

② 메시지인증코드 길이 확인

블록암호별 MAC 길이 확인
- ARIA, SEED, LEA, AES의 MAC 길이 : $112 \leq Len(MAC) \leq 128$ 비트
- HIGHT의 MAC 길이 : $Len(MAC) == 64$비트

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] "안전한 제로화" 참고

GVI Part 2

Guide for Vendor Implementations

5장 해시함수 기반 메시지인증코드

해시함수 기반 메시지인증코드

1 범위

❑ 본 문서에서는 해시함수 기반 메시지인증코드(MAC)를 구현할 경우의 고려사항을 기술한다.

해시함수 기반 메시지인증코드	
HMAC	SHA2-224/256/384/512
	LSH-224/256/384/512 LSH-512_224/512_256
	SHA3-224/256/384/512

2 관련표준

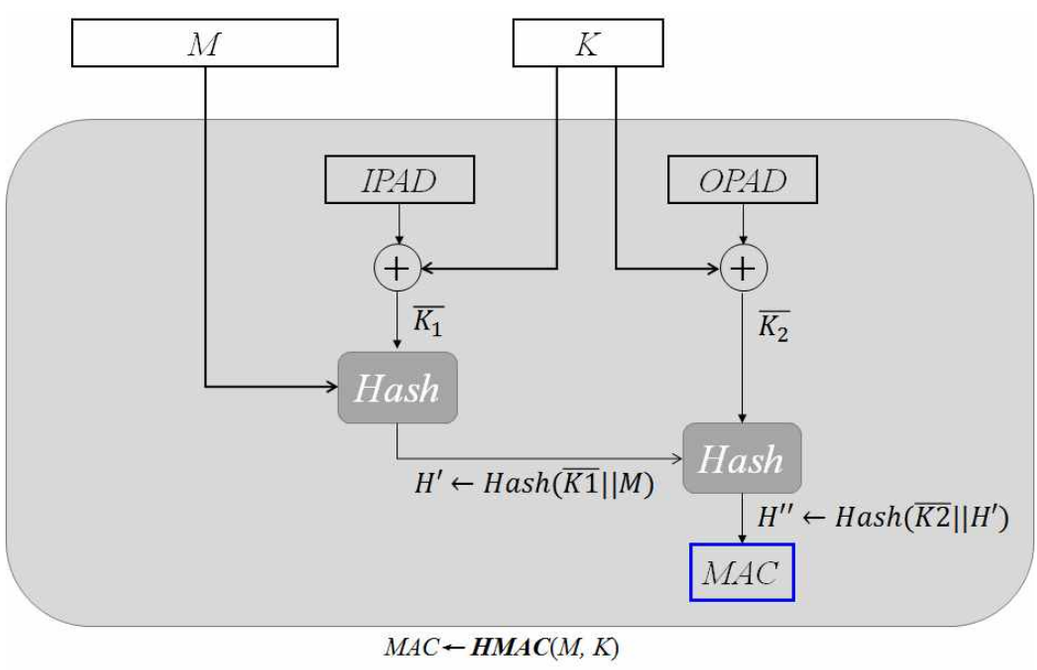
- [KS X ISO/IEC 9797-2] 메시지 인증 코드 – 제2부: 전용 해시 함수를 이용한 메커니즘 (2024)
- [TTAK.KO-12.0330-Part1] 해시 함수 기반 메시지 인증 코드 (HMAC) - 제1부: 일반 (2018)
- [TTAK.KO-12.0330-Part2] 해시 함수 기반 메시지 인증 코드 (HMAC) - 제2부: 해시 함수 SHA-2 (2018)
- [TTAK.KO-12.0330-Part3] 해시 함수 기반 메시지 인증 코드 (HMAC) - 제3부: 해시 함수 LSH (2018)
- [TTAK.KO-12.0330-Part4] 해시 함수 기반 메시지 인증 코드 (HMAC) - 제4부: 해시 함수 SHA-3 (2019)
- [ISO/IEC 9797-2] Message authentication codes (MACs) Part 2: Mechanisms using a dedicated hash-function (2021)

3 기호

기 호	의 미
$Len(X)$	비트열 X의 비트 길이
$IPAD$	HMAC 연산에 사용되는 패딩값
$OPAD$	HMAC 연산에 사용되는 패딩값
M	평문 메시지
K	HMAC 연산 시 사용되는 비밀키
L	해시함수의 출력값 크기(비트)
B	해시함수의 블록 크기(바이트)
MAC	메시지인증코드
m	메시지인증코드 길이(비트)
0^n	n길이의 연속된 0 비트열 (Ex. $0^5 = 00000,\ 0^3 = 000$)

4 해시함수 기반 메시지인증코드

- HMAC은 MAC의 한 종류로서 메시지와 비밀키를 해시함수의 입력으로 MAC값을 계산하는 방식이다. MAC값을 계산하기 위한 해시함수는 검증대상으로 지정된 SHA-2, LSH, SHA-3를 사용할 수 있다.
- HMAC에서 이용하는 비밀키 K의 길이는 L(해시함수 출력 크기)보다 크거나 같아야 한다.



5 알고리즘 명세 및 구현 시 고려사항

가. MAC 생성

HMAC 공통 알고리즘(*HMAC*)		고려사항
입력	– 평문 M – 비밀키 K – 메시지인증코드 길이 m – 해시함수 종류 $Hash$	①, ②, ③
출력	– 길이가 m인 메시지인증코드 MAC	
1	$\text{if}(Len(K) > 8B)$ $\quad \overline{K} = Hash(K)$ $else$ $\quad \overline{K} = K$	
2	$\overline{K} = \overline{K} \| 0^{8B - Len(\overline{K})}$	
3	$\overline{K_1} = \overline{K} \oplus IPAD$ ($IPAD$ = 0x36 * B)	
4	$\overline{K_2} = \overline{K} \oplus OPAD$ ($OPAD$ = 0x5c * B)	
5	$H' = Hash(\overline{K_1} \| M)$	
6	$H'' = Hash(\overline{K_2} \| H')$	
7	$MAC = MSB_m(H'')$	

① 메시지인증코드 길이

해시함수 별로 아래 범위가 아닌 메시지인증코드 길이 입력 시 오류를 반환하는지 확인
- SHA2-224, LSH-224/512_224, SHA3-224 : $112 \leq m \leq 224$ 비트
- SHA2-256, LSH-256/512_256, SHA3-256 : $128 \leq m \leq 256$ 비트
- SHA2-384, LSH-384, SHA3-384 : $192 \leq m \leq 384$ 비트
- SHA2-512, LSH-512, SHA3-512 : $256 \leq m \leq 512$ 비트

② 안전한 키 생성 및 사용

HMAC에 사용되는 키 K는 "중요보안매개변수(SSP) 생성" 방법으로 생성되어야 함
(검증대상 난수발생기를 이용하여 생성하는 등 안전한 방법으로 생성)

HMAC에 사용되는 키 K는 해시함수 별로 해시함수의 출력 길이 L보다 크거나 같아야 한다.

③ 입력메시지 길이 확인

M의 길이 $Len(M)$이 다음 $0 \leq Len(M) < (2^B) - 8B$(비트)의 범위에 속하는지 확인

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] “안전한 제로화” 참고

GVI Part 2

Guide for Vendor Implementations

6장 블록암호 기반 난수발생기

블록암호 기반 난수발생기

1 범위

❑ 본 문서에서는 ISO/IEC 18031에 명시된 난수발생기 중 블록암호 기반 난수발생기(CTR_DRBG)를 구현할 경우의 고려사항을 기술한다.

구분	적용 알고리즘	키 길이
CTR_DRBG	ARIA, LEA, AES	128/192/256
	SEED, HIGHT	128

2 관련표준

- [KS X ISO/IEC 18031] 난수발생기 (2023)
- [TTAK.KO-12.0189/R2] 결정론적 난수 발생기 -제1부- 블록 암호 기반 난수 발생기 (2022)
- [ISO/IEC 18031] Random bit generation (2025)

3 기호

기 호	의 미
$Len(X)$	비트열 X의 비트 길이
$bc\_security\_strength$	블록암호 알고리즘의 보안강도
$blocklen$	블록암호 알고리즘이 한 번의 연산으로 처리할 수 있는 1블록의 길이 (비트)
$keylen$	블록암호 알고리즘의 키 길이 (비트)
$seedlen$	DRBG 내부 메커니즘에 사용되는 $seed$의 비트 길이 ($seedlen = blocklen + keylen$)
$ctrlen$	카운터 필드의 비트 길이
$\lceil n \rceil$	n보다 크거나 같은 정수 중에서 최소값
$min\{s \in S : C\}$	집합 $S(=\{s_1, s_2, \dots, s_n\})$의 원소 중 조건 C를 만족하는 최소값
$Get\_Entropy(A)$	보안강도 A 이상을 갖는 임의 길이의 비트열을 잡음원으로부터 수집하는 함수
$nonce$	DRBG 초기화 단계에서 사용되는 논스
$reseed\_required$	리씨드가 요구됨을 나타내는 상태값
$Null$	Empty String
0^n	n길이의 연속된 0 비트열 (Ex. $0^5 = 00000$, $0^3 = 000$)
1^n	n길이의 연속된 1 비트열 (Ex. $1^5 = 11111$, $1^3 = 111$)
$[n]_s$	음이 아닌 정수 $n\,(n < 2^s)$의 s비트 이진수 표현
$min(A, B)$	A와 B 중에서 작은 값

기 호	의 미
$E_K(X)$	비밀키 K를 이용하여 입력값 X를 암호화하는 블록암호 암호화 함수
$MSB_i(X)$	주어진 비트열 X의 상위(왼쪽) i개 비트열
$LSB_i(X)$	주어진 비트열 X의 하위(오른쪽) i개 비트열
$inc_n(X)$	비트열 X의 하위 n비트를 1만큼 증가시키는 함수 (상위 $Len(X)-n$비트는 고정) ※ $inc_n(X) = MSB_{Len(X)-n}(X) \| ((LSB_n(X)+1) \bmod 2^n)$

4 블록암호 기반 난수발생기

▸ CTR_DRBG는 CTR모드(Counter mode)로 동작하는 블록암호 알고리즘을 사용하는 DRBG이다. CTR_DRBG의 초기화 함수, 리씨드 및 난수 생성 함수에 블록암호가 사용된다. CTR_DRBG는 초기화와 리씨드 함수에서 유도함수를 사용하는 경우와 유도함수를 사용하지 않는 경우의 2가지 형태로 구현될 수 있다. CTR_DRBG의 최대 안전성은 사용된 블록암호와 키 길이의 안전성에 의존한다.

❑ 난수발생기의 기본적인 구성요소는 다음과 같다.

구성요소	의 미
엔트로피 입력 (Entropy Input)	난수발생기에 (평가된 엔트로피에 해당하는) 예측 불가능성을 제공하기 위해 입력되는 비트열
인스턴스 초기화 (Instantiation)	논리적으로 독립적인 새로운 난수발생기 인스턴스를 초기 설정하는 과정
리씨드 (Reseed)	난수발생기 내부 상태에 영향을 미치는 새로운 정보(엔트로피 및 추가 입력값)을 획득하여 새로운 내부 상태로 갱신하는 과정
생성 (Generation)	인스턴스 초기화 또는 리씨드 후 수행하며, 요청된 의사난수 비트열을 생성하는 과정
내부 상태 (Internal State)	난수발생기 동작 시 필요에 의해 관리되는 모든 정보의 집합 보안강도 및 플래그와 같은 정책적인 데이터와 난수생성 시 사용되는 상태값 등 운영 데이터로 구성되며, 비밀 정보와 비밀이 아닌 정보가 모두 포함될 수 있음

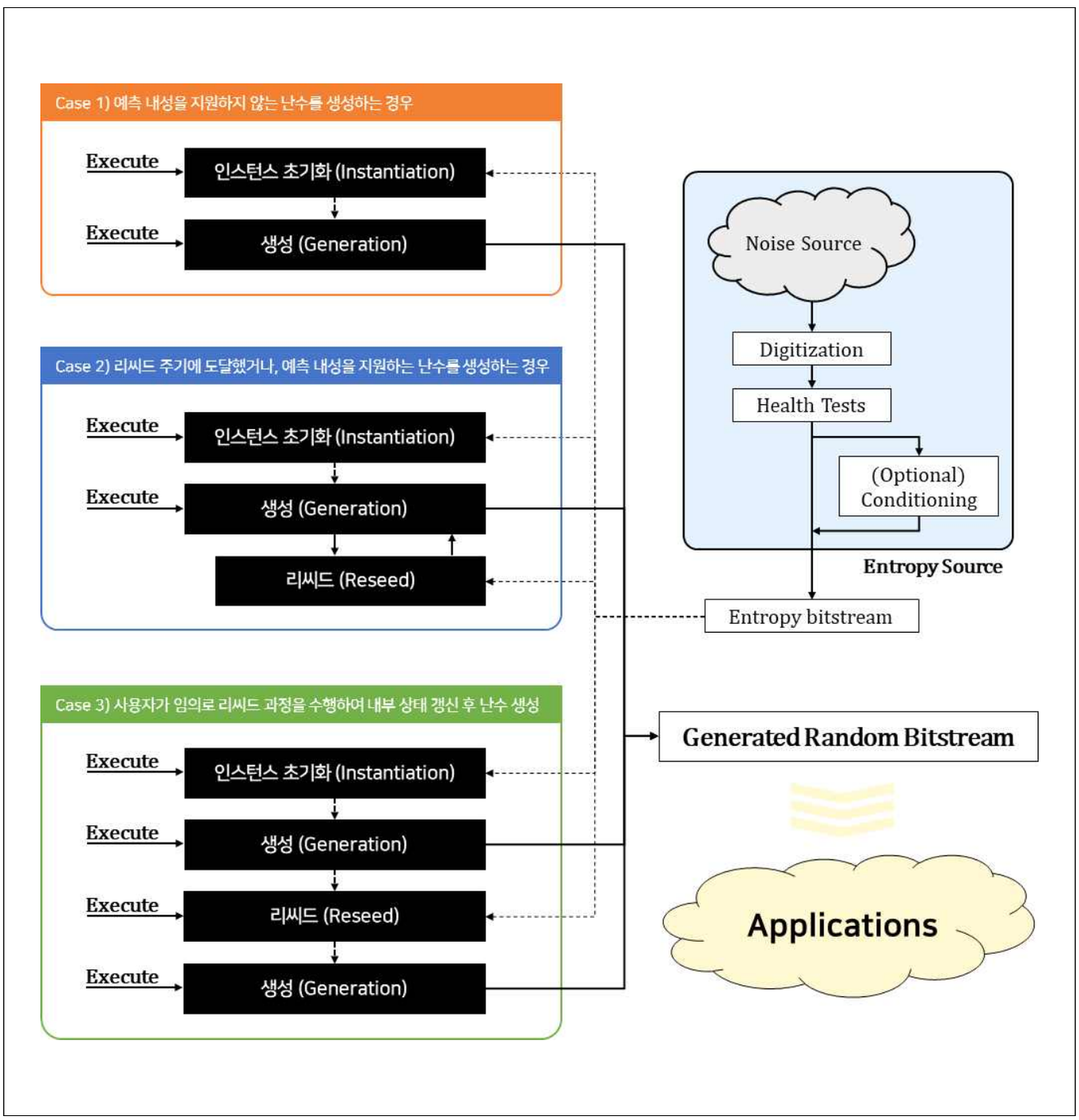
❑ 다음 요소들은 선택적인 요소이다.

구성요소	의 미
추가 입력 (Additional Input)	선택적으로 사용되는 입력값으로, 리씨드 및 생성 과정에서 활용
개별화 문자열 (Personalization String)	선택적으로 사용되는 입력값으로, 인스턴스 초기화 과정에서 개별화 정보를 제공하기 위해 사용됨 장치일련번호, 사용자 및 응용프로그램 ID 등이 사용될 수 있음
논스 (Nonce)	인스턴스 초기화 과정에 사용되는 반복되지 않는 가변적인 값으로 난수발생기 구현 방법에 따라 적용여부가 결정됨 예) 타임스탬프, 일련번호 또는 이들의 조합

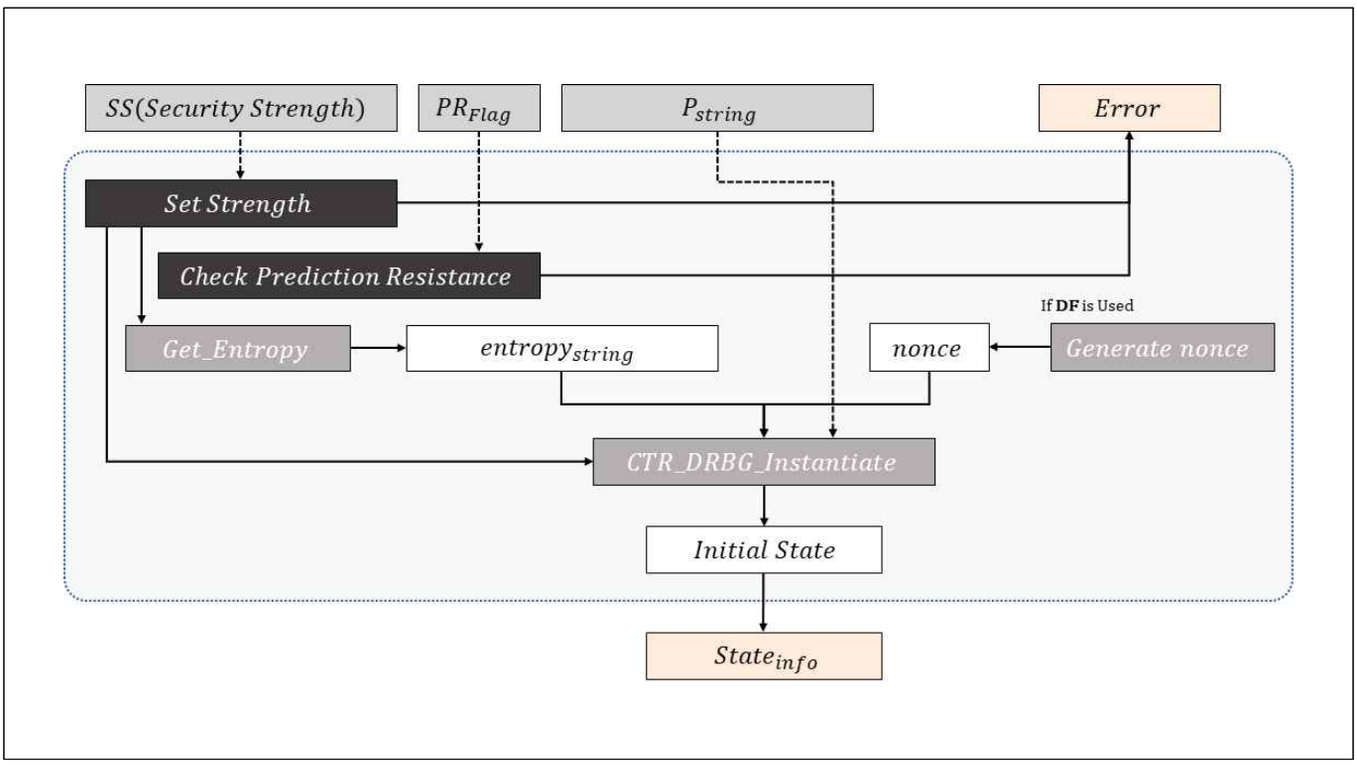
5 알고리즘 명세 및 구현 시 고려사항

가. 난수발생기 일반

☐ DRBG 난수 생성 과정



□ 인스턴스 초기화 단계



DRBG 인스턴스 초기화(*Instantiation*)		고려사항
입력	– 요구 보안강도 SS – 예측내성 지원 여부 PR_{Flag} ($TRUE$ 또는 $FALSE$) – 개별화 문자열 P_{string}	①
출력	– 내부 상태 또는 식별값 $State_{info}$	
1	if($SS > bc\_security\_strength$) Error 출력	①
2	if($PR_{Flag} == TRUE$) && (구현물이 예측내성을 지원하지 않는 경우) Error 출력	
3	$strength = min\{ tmp \in \{112, 128, 192, 256\} : tmp \geq SS \}$	
4	$entropy_{string} = Get\_Entropy(strength)$	②
5	if(유도함수 사용 시), $nonce$ 생성	③
6	$Initial\_State = \boldsymbol{CTR\_DRBG\_Instantiate}(entropy_{string}, nonce, P_{string}, strength)$	
7	$State_{info} = [Initial\_State]$ or $[Indicator\ of\ Initial\_State]$	
8	$State_{info}$ 출력	

① 입력값 확인

요구 보안강도 SS : 사용되는 블록암호 알고리즘의 보안강도보다 작거나 같아야 함
개별화 문자열 길이 확인 - 유도함수 사용 시 : $0 \leq Len(P_{string}) \leq 2^{35}$ 비트 - 유도함수 미사용 시 : $0 \leq Len(P_{string}) \leq seedlen$ 비트 $(seedlen = blocklen + keylen)$

② 엔트로피 수집 유효성 확인

엔트로피 수집함수가 건전성 테스트를 통과한 잡음원을 수집하는지 확인
수집된 엔트로피의 길이 및 보안강도 확인 - 유도함수 사용 시 : $Len(entropy_{string}) \leq 2^{35}$ 비트 - 유도함수 미사용 시 : $Len(entropy_{string}) == seedlen$ 비트 $(seedlen = blocklen + keylen)$ - $entropy_{string}$의 엔트로피가 $strength$ 이상이어야 함 ※ 유도함수 미사용 시, $entropy_{string}$은 Full-entropy를 가져야 함

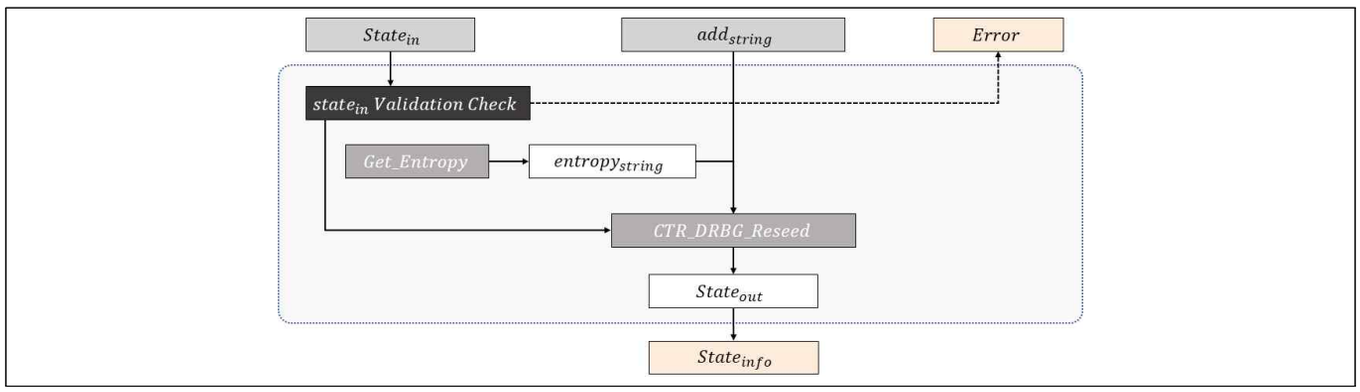
③ 논스 생성 확인

논스는 초기화 연산의 각 실행에 대해 유일해야 하며, 비밀값일 필요는 없음 - 사용된 논스 값은 CSP로 관리되어야 함
생성된 논스 길이 확인 - $Len(nonce) \geq (strength / 2)$

[공통] 작동상태 관리 및 제로화 수행

각 난수발생기 인스턴스는 독립적인 내부상태를 가져야 함
사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] “안전한 제로화” 참고

□ 리씨드 단계



DRBG 리씨드(*Reseed*)		고려사항
입력	– 내부 상태 또는 식별값 $State_{in}$ – 추가 입력 add_{string}	①
출력	– (갱신된) 내부 상태 또는 식별값 $State_{info}$	
1	입력된 $State_{in}$ 유효성 확인	
2	$entropy_{string} = Get\_Entropy(strength)$	②
3	$State_{out} = \boldsymbol{CTR\_DRBG\_Reseed}(State_{in}, entropy_{string}, add_{string})$	
4	$State_{info} = [State_{out}]$ or $[Indicator\ of\ State_{out}]$	
5	$State_{info}$ 출력	

① 입력값 확인

추가 입력 길이 확인
- 유도함수 사용 시 : $0 \leq Len(add_{string}) \leq 2^{35}$ 비트
- 유도함수 미사용 시 : $0 \leq Len(add_{string}) \leq seedlen$ 비트 $(seedlen = blocklen + keylen)$

② 엔트로피 수집 유효성 확인

엔트로피 수집함수가 건전성 테스트를 통과한 잡음원을 수집하는지 확인

수집된 엔트로피의 길이 및 보안강도 확인
- 유도함수 사용 시 : $Len(entropy_{string}) \leq 2^{35}$ 비트
- 유도함수 미사용 시 : $Len(entropy_{string}) == seedlen$ 비트 $(seedlen = blocklen + keylen)$
- $entropy_{string}$의 엔트로피가 $strength$ 이상이어야 함

※ 유도함수 미사용 시, $entropy_{string}$은 Full-entropy를 가져야 함

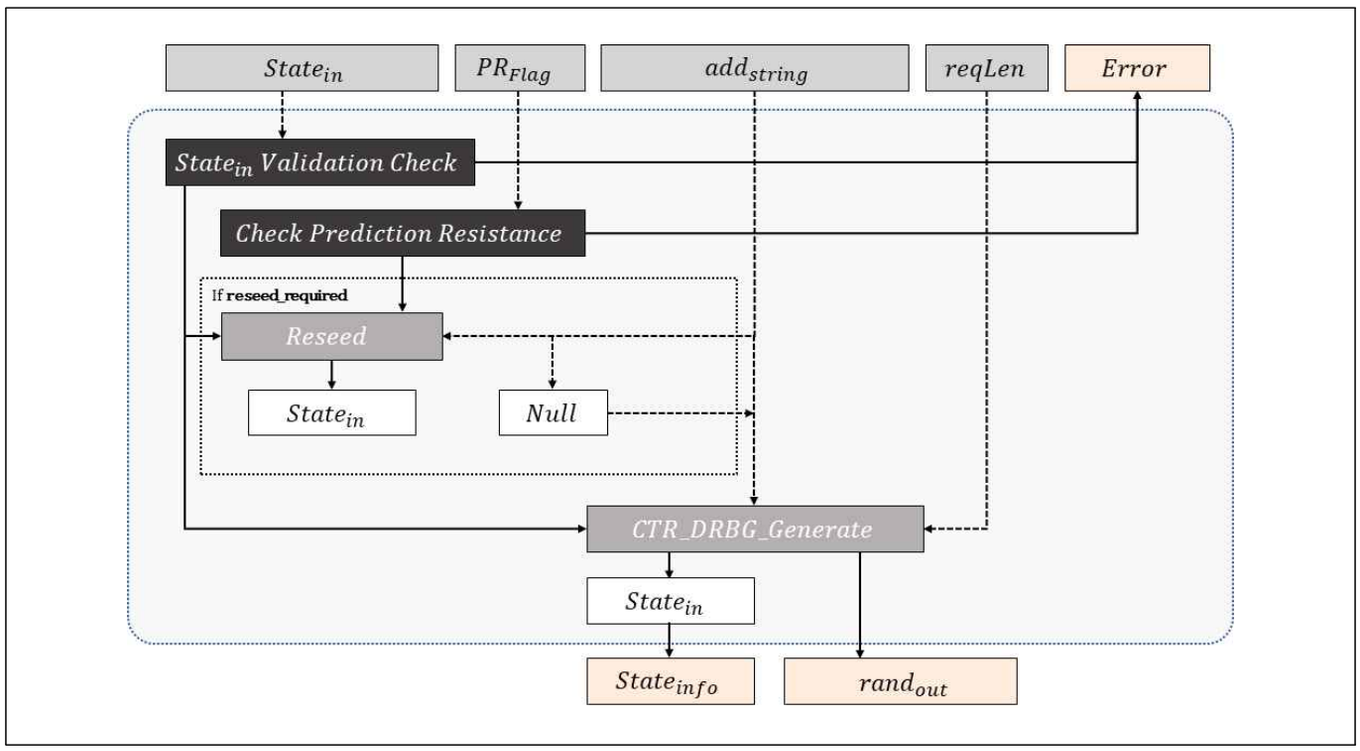
[공통] 작동상태 관리 및 제로화 수행

각 난수발생기 인스턴스는 독립적인 내부상태를 유지해야 함

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] "안전한 제로화" 참고

□ 생성 단계



DRBG 생성(*Generation*)		고려사항
입력	– 내부 상태 또는 식별값 $State_{in}$ – 예측내성 요청 여부 PR_{Flag} ($TRUE$ 또는 $FALSE$) – 추가 입력 add_{string} – 난수 출력 요청 길이 $reqLen$	①
출력	– (갱신된) 내부 상태 또는 식별값 $State_{info}$ – 생성된 난수 비트열 $rand_{out}$	
1	입력된 $State_{in}$ 유효성 확인	
2	if(구현물이 예측내성을 지원하지 않는 경우) && if($PR_{Flag} == TRUE$) Error 출력	
3	$reseed\_required\_flag = FALSE$	
4	if($reseed\_required\_flag == TRUE$) 또는 if($PR_{Flag} == TRUE$) $State_{in} = \textbf{\textit{Reseed}}(State_{in}, add_{string})$ $add_{string} = Null$ $reseed\_required\_flag = FALSE$	
5	$(status, rand_{out}, State_{in}) = \textbf{\textit{CTR\_DRBG\_Generate}}(State_{in}, add_{string}, reqLen)$	
6	if($status == reseed\_required$) $reseed\_required\_flag = TRUE$ 단계 4로 이동	
7	$State_{info} = [State_{in}]$ or $[Indicator\ of\ State_{in}]$	
8	$State_{info}$ 및 $rand_{out}$ 출력	

① 입력값 확인

추가 입력 길이 확인 - 유도함수 사용 시 : $0 \leq Len(add_{string}) \leq 2^{35}$ 비트 - 유도함수 미사용 시 : $0 \leq Len(add_{string}) \leq seedlen$ 비트 $(seedlen = blocklen + keylen)$
난수 출력 요청 길이 확인 (카운터 필드 길이 $ctrlen$: $4 \leq ctrlen \leq blocklen$ 비트) - $reqLen \leq min((2^{ctrlen} - 4) \times blocklen), LIMIT)$ 비트 · ARIA, SEED, LEA, AES : $LIMIT = 2^{19}$ · HIGHT : $LIMIT = 2^{13}$

[공통] 작동상태 관리 및 제로화 수행

각 난수발생기 인스턴스는 독립적인 내부상태를 유지해야 함
사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] “안전한 제로화” 참고

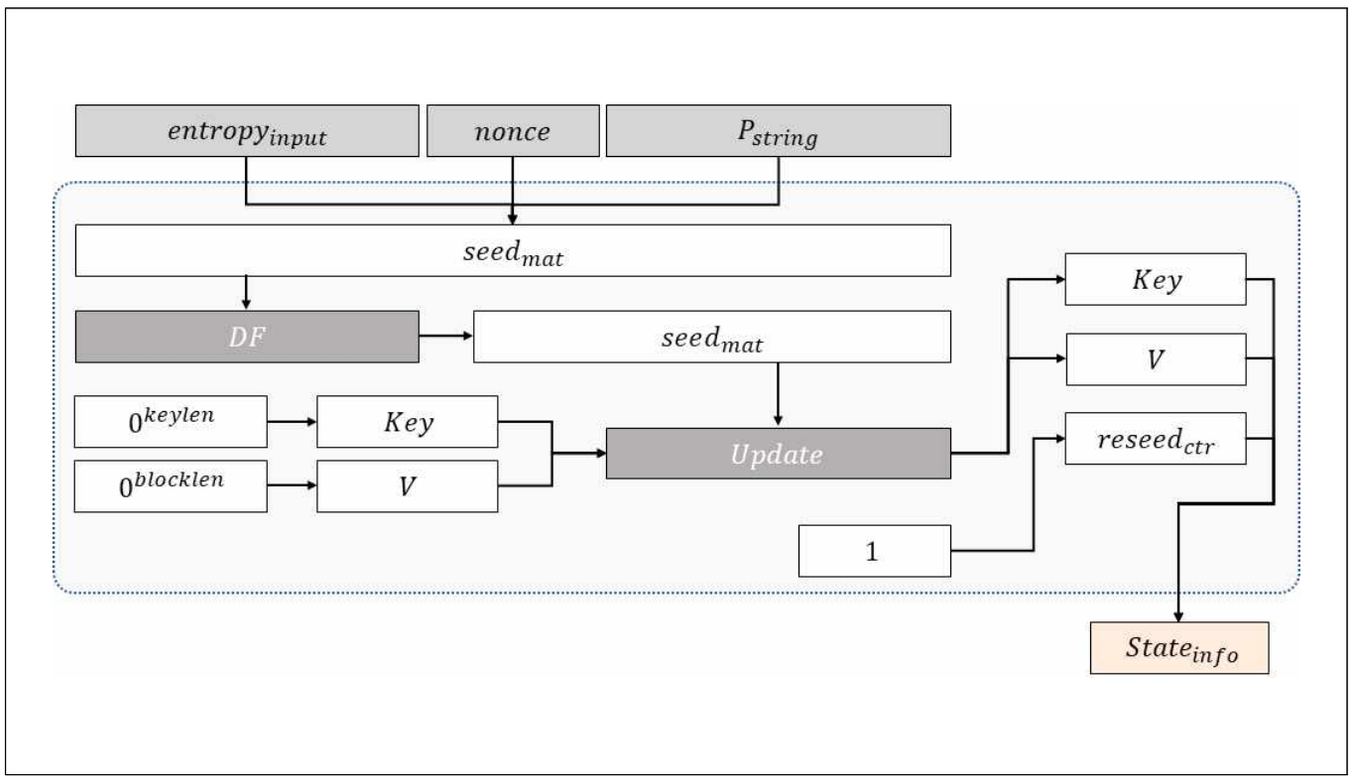
나. CTR-DRBG

■ CTR-DRBG 내부 블록암호 알고리즘별 주요 특징 (길이 단위 : 비트)

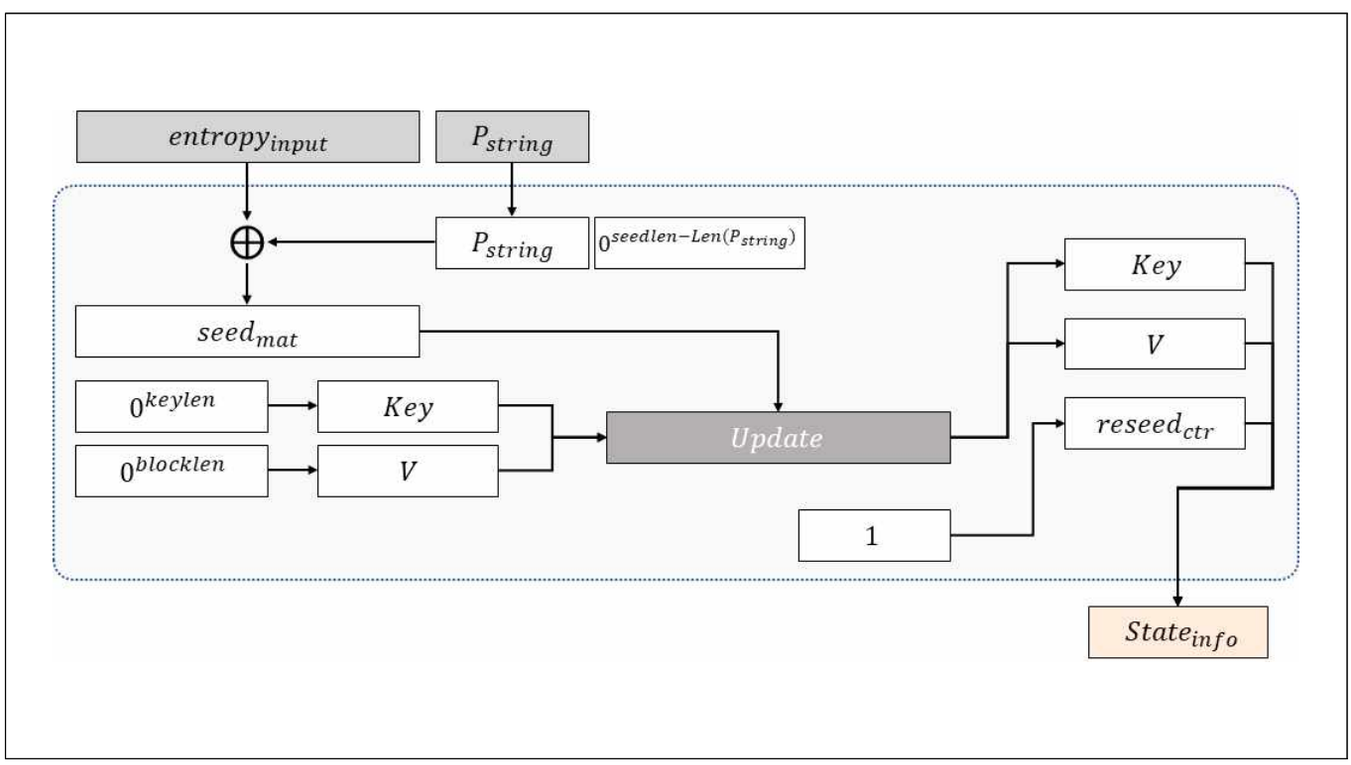
구분	ARIA/LEA/AES -128	ARIA/LEA/AES -192	ARIA/LEA/AES -256	SEED	HIGHT
보안강도 ($bc\_security\_strength$)	128	192	256	128	128
블록 길이 ($blocklen$)	128				64
키 길이 ($keylen$)	128	192	256	128	128
카운터 필드 길이 ($ctrlen$)	$4 \leq ctrlen \leq blocklen$				
시드 길이 ($seedlen$)	256	320	384	256	192
난수 생성 최대 요청 길이 ($C = (2^{ctrlen} - 4) \times blocklen$)	$min(C, 2^{19})$				$min(C, 2^{13})$
리씨드 주기 ($reseed\_interval$)	2^{48}				2^{32}
유도함수 사용 시					
엔트로피 입력 최소 길이	$strength\ (in\ \textbf{\textit{Instantiation}}\ Phase)$				
엔트로피 입력 최대 길이	2^{35}				
개별화문자열 최대 길이	2^{35}				
추가입력 최대 길이	2^{35}				
유도함수 미사용 시					
엔트로피 입력 최소 길이	$seedlen$				
엔트로피 입력 최대 길이	$seedlen$				
개별화문자열 최대 길이	$seedlen$				
추가입력 최대 길이	$seedlen$				

□ 초기화 함수 ($CTR\_DRBG\_Instantiate$)

■ 유도함수 DF를 사용하는 경우



■ 유도함수를 사용하지 않는 경우



CTR-DRBG 초기화 함수(*CTR_DRBG_Instantiate*)		고려사항
입력	– 엔트로피 입력 $entropy_{input}$ – (유도함수 사용 시) 논스 $nonce$ – 개별화 문자열 P_{string} – 보안강도 $strength$	①
출력	– 내부 상태 또는 식별값 $state_{info}$	
	유도함수를 사용하는 경우	
1	$seed_{mat} = entropy_{input} \| nonce \| P_{string}$	
2	$seed_{mat} = \boldsymbol{DF}(seed_{mat}, seedlen)$	②
3	-	
	유도함수를 사용하지 않는 경우	
1	$temp = Len(P_{string})$	
2	$\text{if}(temp < seedlen)$ $P_{string} = P_{string} \| 0^{seedlen-temp}$	
3	$seed_{mat} = entropy_{input} \oplus P_{string}$	
4	$Key = 0^{keylen}$	
5	$V = 0^{blocklen}$	
6	$(Key, V) = \boldsymbol{Update}(seed_{mat}, Key, V)$	
7	$reseed_{ctr} = 1$	
8	(Key, V, $reseed_{ctr}$)가 포함된 $state_{info}$ 출력	

① 입력된 보안강도($strength$) 이상의 안전성을 갖는 블록암호 알고리즘을 사용하는지 확인

② 블록암호 알고리즘별 설정되는 시드 길이 확인

- ARIA-128, LEA-128, AES-128, SEED : 256 비트
- ARIA-192, LEA-192, AES-192 : 320 비트
- ARIA-256, LEA-256, AES-256 : 384 비트
- HIGHT : 192 비트

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] “안전한 제로화” 참고

□ 업데이트 함수 ($Update$)

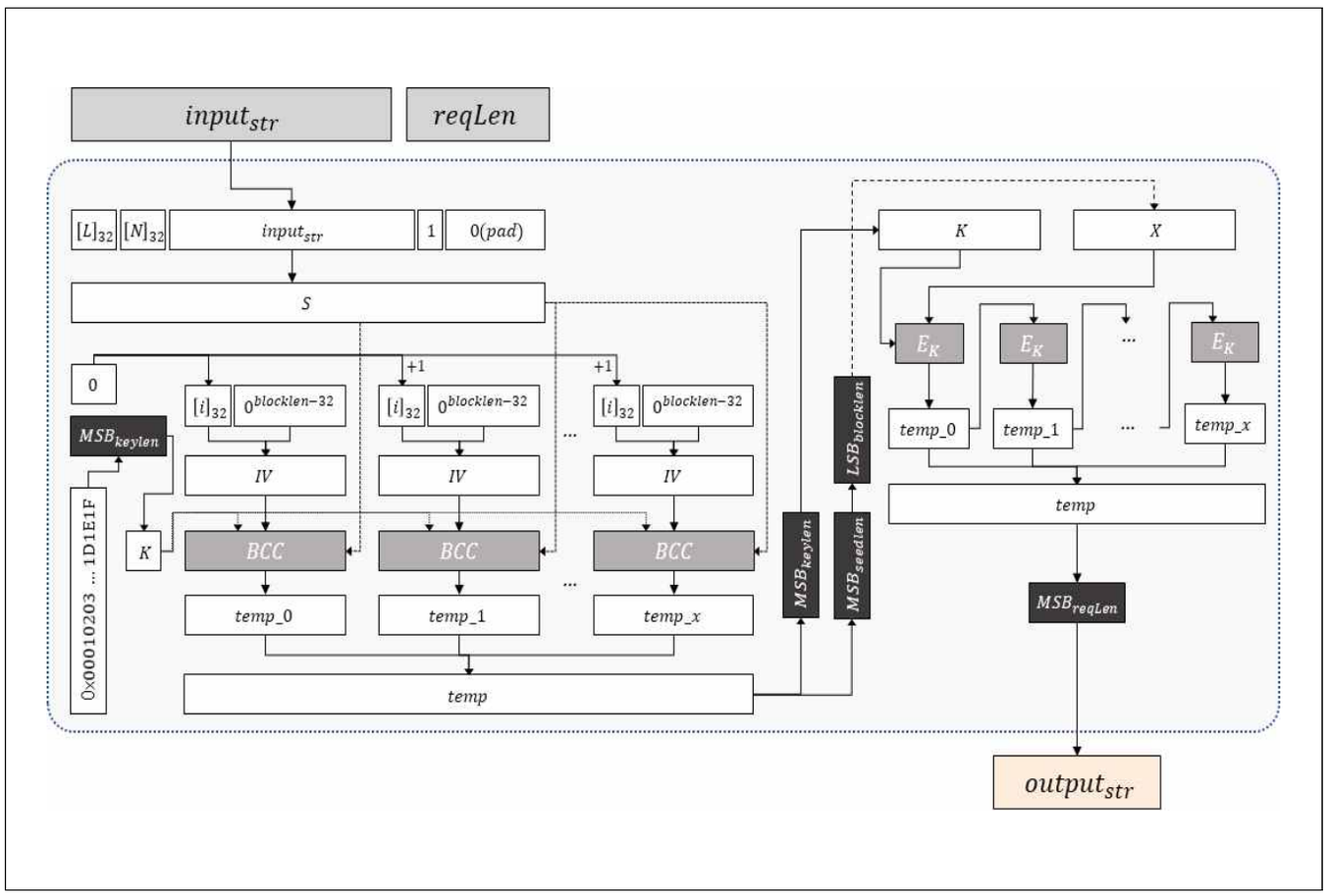
업데이트 함수(*Update*)		고려사항
입력	– 입력 비트열 $input_{str}$ – 난수발생기 상태값 Key – 난수발생기 상태값 V	
출력	– (갱신된) 난수발생기 상태값 Key – (갱신된) 난수발생기 상태값 V	
1	$temp = Null$	
2	$While(Len(temp) < seedlen)$	
3	$\text{if}(ctrlen < blocklen)$ $inc = (LSB_{ctrlen}(V) + 1) \bmod 2^{ctrlen}$ $V = MSB_{blocklen-ctrlen}(V) \| [inc]_{ctrlen}$	
4	$else$ $V = (V+1) \bmod 2^{blocklen}$	
5	$blk = E_{Key}(V)$	
6	$temp = temp \| blk$	
7	$temp = MSB_{seedlen}(temp)$	
8	$temp = temp \oplus input_{str}$	
9	$Key = MSB_{keylen}(temp)$	
10	$V = LSB_{blocklen}(temp)$	
11	(Key, V) 출력	

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] “안전한 제로화” 참고

□ 유도함수 (*DF*)



BCC 함수(*BCC*)		고려사항
입력	– 키 K – 입력 비트열 $input_{str}$	
출력	– 출력 비트열 $output_{str}$	
1	$chain = 0^{blocklen}$	
2	$n = Len(input_{str})/blocklen$	
3	$block_1 \| block_2 \| ... \| block_n = input_{str}$	
4	for i from 1 to n do	
5	$temp = chain \oplus block_i$ $chain = E_K(temp)$	
6	end for	
7	$output_{str} = chain$	
8	$output_{str}$ 출력	

CTR-DRBG 유도함수(*DF*)		고려사항
입력	– 입력 비트열 $input_{str}$ – 요청 길이(비트) $reqLen$	
출력	– 출력 비트열 $output_{str}$	
1	$L = Len(input_{str})/8$	
2	$N = reqLen/8$	
3	$S = [L]_{32} \| [N]_{32} \| input_{str} \| 1 \| 0^7$	
4	$While(Len(S) \bmod blocklen \neq 0)$ $S = S \| 0^8$	
5	$temp = Null$	
6	$i = 0$	
7	$K = MSB_{keylen}$(0x00010203 ... 1D1E1F)	
8	$While(Len(temp) < seedlen)$ $IV = [i]_{32} \| 0^{blocklen-32}$ $temp = temp \| \boldsymbol{BCC}(K, (IV \| S))$ $i = i + 1$	
9	$K = MSB_{keylen}(temp)$	
10	$X = LSB_{blocklen}(MSB_{seedlen}(temp))$	
11	$temp = Null$	
12	$While(Len(temp) < reqLen)$ $X = E_K(X)$ $temp = temp \| X$	
13	$output_{str} = MSB_{reqLen}(temp)$	
14	$output_{str}$ 출력	

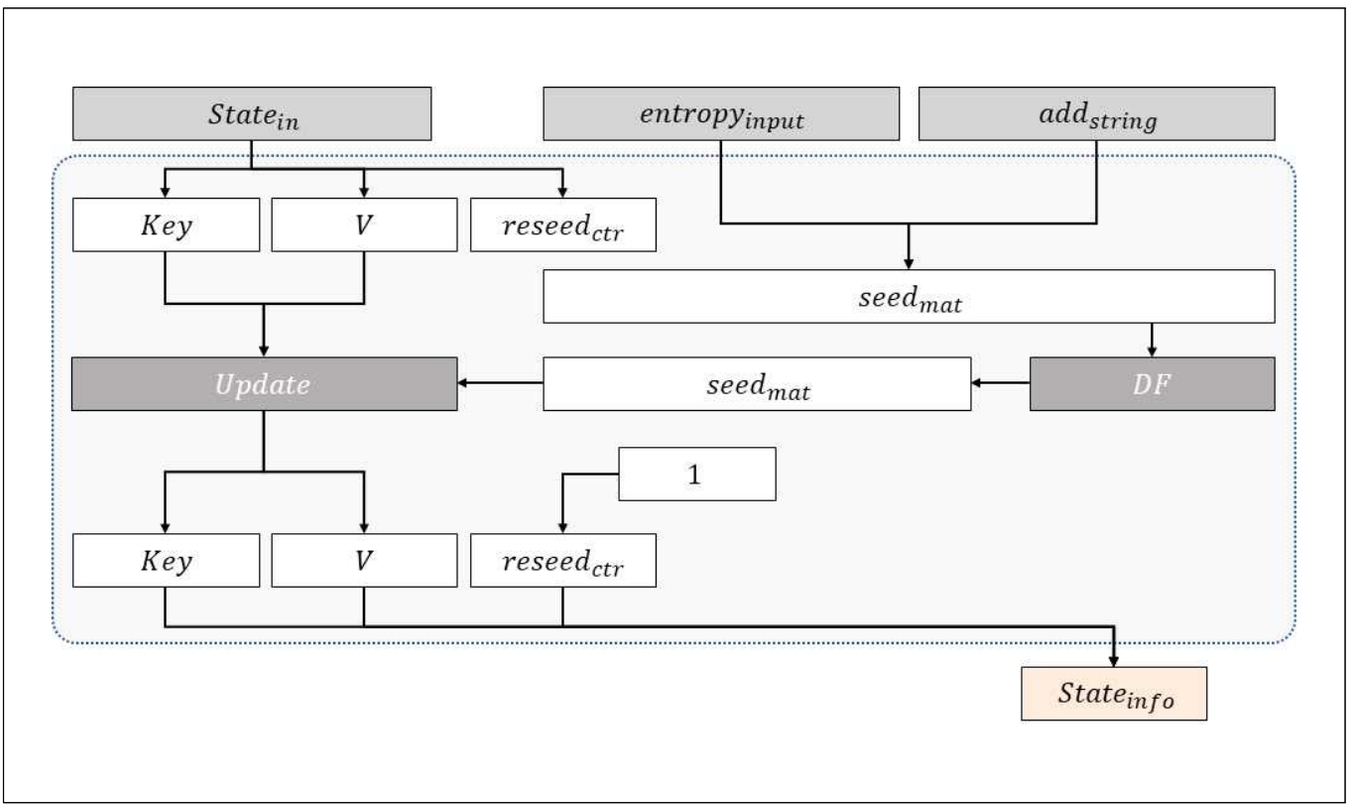
[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

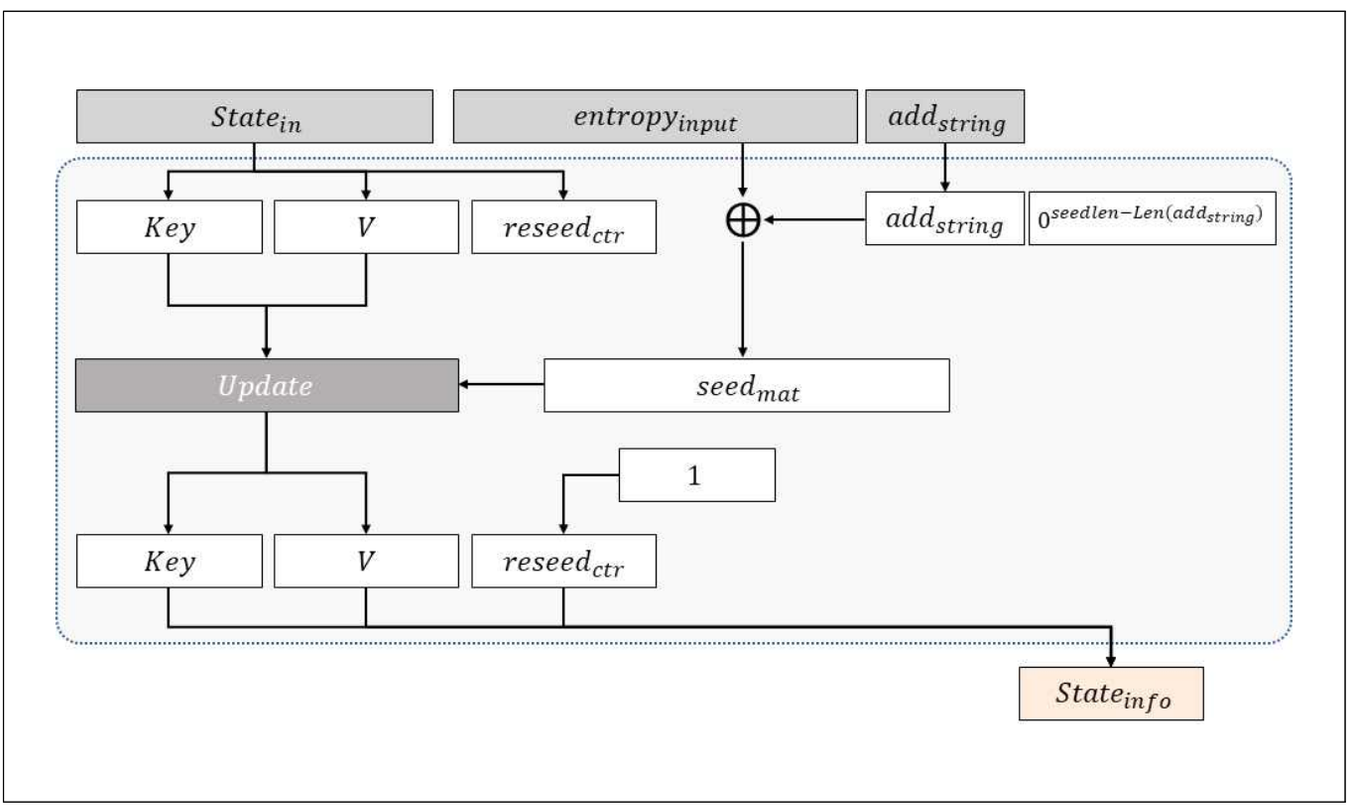
※ [부록] “안전한 제로화” 참고

□ 리씨드 함수 (*CTR_DRBG_Reseed*)

■ 유도함수 *DF*를 사용하는 경우



■ 유도함수를 사용하지 않는 경우



CTR-DRBG 리씨드 함수(*CTR_DRBG_Reseed*)		고려사항
입력	– 내부 상태 또는 식별값 $state_{in}$ – 엔트로피 입력 $entropy_{input}$ – 추가 입력 add_{string}	
출력	– (갱신된) 내부 상태 또는 식별값 $state_{info}$	
	유도함수를 사용하는 경우	
1	$seed_{mat} = entropy_{input} \| add_{string}$	
2	$seed_{mat} = \boldsymbol{DF}(seed_{mat}, seedlen)$	①
3	-	
	유도함수를 사용하지 않는 경우	
1	$temp = Len(add_{string})$	
2	$\text{if}(temp < seedlen)$ $\quad add_{string} = add_{string} \| 0^{seedlen-temp}$	
3	$seed_{mat} = entropy_{input} \oplus add_{string}$	
4	$(Key, V) = \boldsymbol{Update}(seed_{mat}, Key, V)$	
5	$reseed_{ctr} = 1$	
6	(Key, V, $reseed_{ctr}$)가 포함된 $state_{info}$ 출력	

① 블록암호 알고리즘별 설정되는 시드 길이 확인

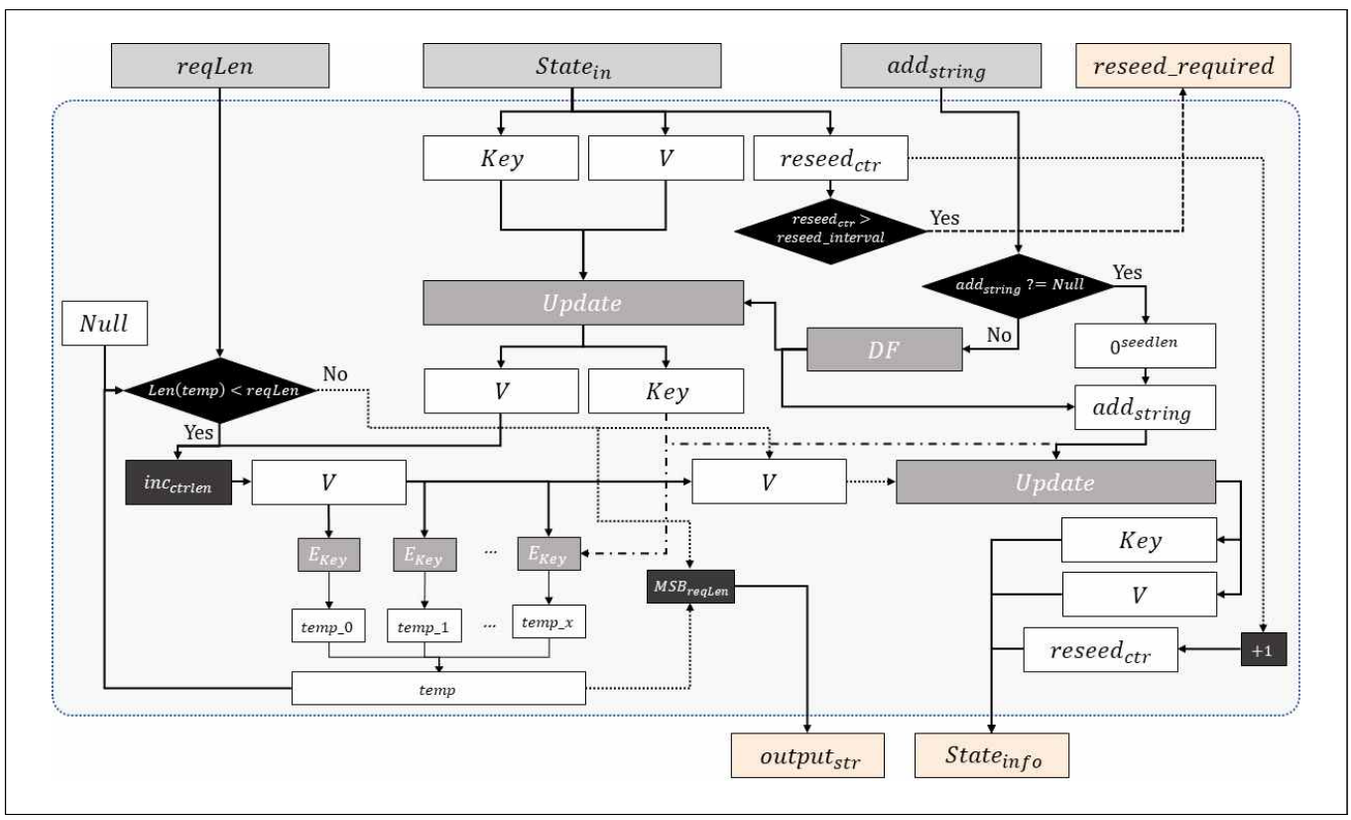
- ARIA-128, LEA-128, AES-128, SEED : 256 비트
- ARIA-192, LEA-192, AES-192 : 320 비트
- ARIA-256, LEA-256, AES-256 : 384 비트
- HIGHT : 192 비트

[공통] 제로화 수행

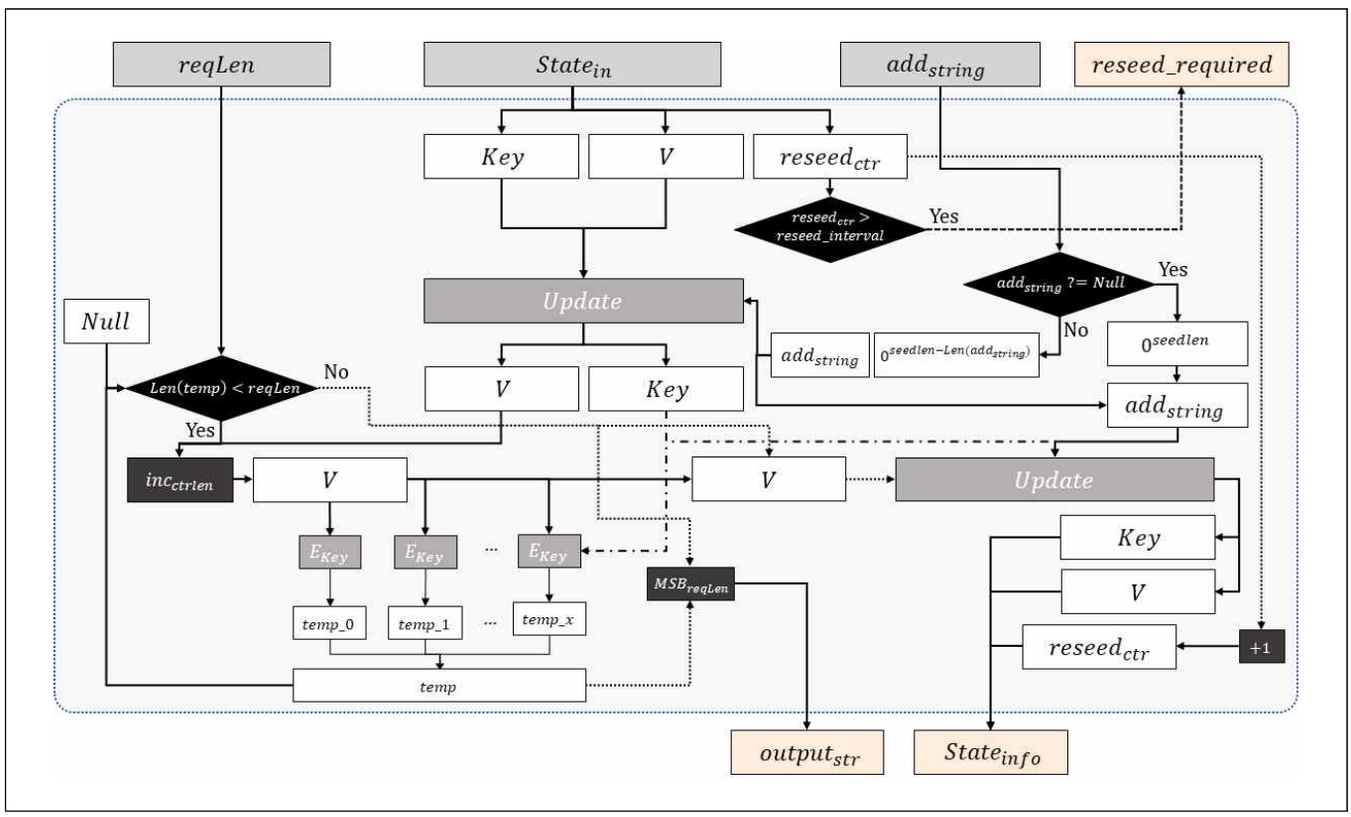
사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] “안전한 제로화” 참고

□ 생성 함수 ($CTR\_DRBG\_Generate$)

■ 유도함수 DF를 사용하는 경우



■ 유도함수를 사용하지 않는 경우



CTR-DRBG 생성 함수(*CTR_DRBG_Generate*)		고려사항
입력	– 내부 상태 또는 식별값 $state_{in}$ – 추가 입력 add_{string} – 난수 출력 요청 길이(비트) $reqLen$	
출력	– 동작 반환값 $indicator$ – 생성된 난수 비트열 $output_{str}$ – (갱신된) 내부 상태 또는 식별값 $state_{info}$	
1	if$(reseed_{ctr} > reseed_interval)$ $indicator$ = $reseed\_required$ $indicator$ 출력	①
	유도함수를 사용하는 경우	②
2	if$(add_{string} \neq Null)$ $add_{string} = \boldsymbol{DF}(add_{string}, seedlen)$ $(Key, V) = \boldsymbol{Update}(add_{string}, Key, V)$	
3	$else$ $add_{string} = 0^{seedlen}$	
4	-	
5	-	
	유도함수를 사용하지 않는 경우	②
2	if$(add_{string} \neq Null)$ $temp = Len(add_{string})$	
3	if$(temp < seedlen)$ $add_{string} = add_{string} \| 0^{seedlen-temp}$	
4	$(Key, V) = \boldsymbol{Update}(add_{string}, Key, V)$	
5	$else$ $add_{string} = 0^{seedlen}$	
6	$temp = Null$	
7	$While(Len(temp) < reqLen)$	
8	if$(ctrlen < blocklen)$ $inc = (LSB_{ctrlen}(V) + 1) \bmod 2^{ctrlen}$ $V = MSB_{blocklen-ctrlen}(V) \| [inc]_{ctrlen}$	
9	$else$ $V = (V+1) \bmod 2^{blocklen}$	
10	$blk = E_{Key}(V)$	
11	$temp = temp \| blk$	
12	$output_{str} = MSB_{reqLen}(temp)$	
13	$(Key, V) = \boldsymbol{Update}(add_{string}, Key, V)$	
14	$reseed_{ctr} = reseed_{ctr} + 1$	
15	$indicator = SUCCESS$	
16	$indicator$, $output_{str}$, (Key, V, $reseed_{ctr}$)가 포함된 $state_{info}$ 출력	

① 리씨드 주기 확인

- ARIA, SEED, LEA, AES : $reseed\_interval \leq 2^{48}$
- HIGHT : $reseed\_interval \leq 2^{32}$

② 블록암호 알고리즘별 설정되는 시드 길이 확인

- ARIA-128, LEA-128, AES-128, SEED : 256 비트
- ARIA-192, LEA-192, AES-192 : 320 비트
- ARIA-256, LEA-256, AES-256 : 384 비트
- HIGHT : 192 비트

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] “안전한 제로화” 참고

7장 해시함수 기반 난수발생기

해시함수 기반 난수발생기

1 범위

❑ 본 문서에서는 ISO/IEC 18031에 명시된 난수발생기 중 Hash_DRBG와 HMAC_DRBG의 구현 시 고려사항을 기술한다.

구분	적용 알고리즘	
Hash_DRBG	-	SHA2-224/256/384/512
		LSH-224/256/384/512 LSH-512_224/512_256
		SHA3-224/256/384/512
HMAC_DRBG	HMAC	SHA2-224/256/384/512
		LSH-224/256/384/512 LSH-512_224/512_256
		SHA3-224/256/384/512

2 관련표준

- [KS X ISO/IEC 18031] 난수발생기 (2023)
- [TTAK.KO-12.0331-Part1/R1] 해시 함수 기반 결정론적 난수발생기 - 제1부: 일반 (2022)
- [TTAK.KO-12.0332-Part1/R1] HMAC 기반 결정론적 난수발생기 - 제1부: 일반 (2022)
- [ISO/IEC 18031] Information technology - Security techniques - Random bit generation (2025)

3 기호

기 호	의 미
$Len(X)$	비트열 X의 비트 길이
$hash\_security\_strength$	해시함수 또는 HMAC 알고리즘의 보안강도
$outlen$	해시함수의 출력 해시값 또는 HMAC 알고리즘의 출력 MAC값 길이
$seedlen$	DRBG 내부 메커니즘에 사용되는 $seed$의 비트 길이
$\lceil n \rceil$	n보다 크거나 같은 정수 중에서 최소값
$min\{s \in S : C\}$	집합 $S(=\{s_1, s_2, \ldots, s_n\})$의 원소 중 조건 C를 만족하는 최소값
$Get\_Entropy(A)$	보안강도 A 이상을 갖는 임의 길이의 비트열을 잡음원으로부터 수집하는 함수
$nonce$	DRBG 초기화 단계에서 사용되는 논스
$reseed\_required$	리씨드가 요구됨을 나타내는 상태값
$Null$	Empty String
0^n	n길이의 연속된 0 비트열 (Ex. $0^5 = 00000$, $0^3 = 000$)

기 호	의 미
1^n	n길이의 연속된 1 비트열 (Ex. $1^5 = 11111$, $1^3 = 111$)
$[n]_s$	음이 아닌 정수 $n\,(n < 2^s)$의 s비트 이진수 표현
$min(A, B)$	A와 B 중에서 작은 값
$Hash(X)$	입력값 X에 대한 해시함수 출력 해시값
$HMAC_K(X)$	입력값 X에 대하여 비밀키 K를 이용한 HMAC 연산 출력 MAC값
$MSB_i(X)$	주어진 비트열 X의 상위(왼쪽) i개 비트열
$LSB_i(X)$	주어진 비트열 X의 하위(오른쪽) i개 비트열

4 해시함수 기반 난수발생기

- 해시 기반 난수발생기(Hash_DRBG 및 HMAC_DRBG)는 초기화 함수, 리씨드 및 난수 생성 함수에 해시함수 또는 HMAC 알고리즘을 사용하는 난수발생기이다.
- Hash_DRBG 및 HMAC_DRBG의 최대 안전성은 사용된 해시함수의 안전성에 의존한다.

❑ 난수발생기의 기본적인 구성요소는 다음과 같다.

구성요소	의 미
엔트로피 입력 (Entropy Input)	난수발생기에 (평가된 엔트로피에 해당하는) 예측 불가능성을 제공하기 위해 입력되는 비트열
인스턴스 초기화 (Instantiation)	논리적으로 독립적인 새로운 난수발생기 인스턴스를 초기 설정하는 과정
리씨드 (Reseed)	난수발생기 내부 상태에 영향을 미치는 새로운 정보(엔트로피 및 추가 입력값)을 획득하여 새로운 내부 상태로 갱신하는 과정
생성 (Generation)	인스턴스 초기화 또는 리씨드 후 수행하며, 요청된 의사난수 비트열을 생성하는 과정
내부 상태 (Internal State)	난수발생기 동작 시 필요에 의해 관리되는 모든 정보의 집합 보안강도 및 플래그와 같은 정책적인 데이터와 난수생성 시 사용되는 상태값 등 운영 데이터로 구성되며, 비밀 정보와 비밀이 아닌 정보가 모두 포함될 수 있음

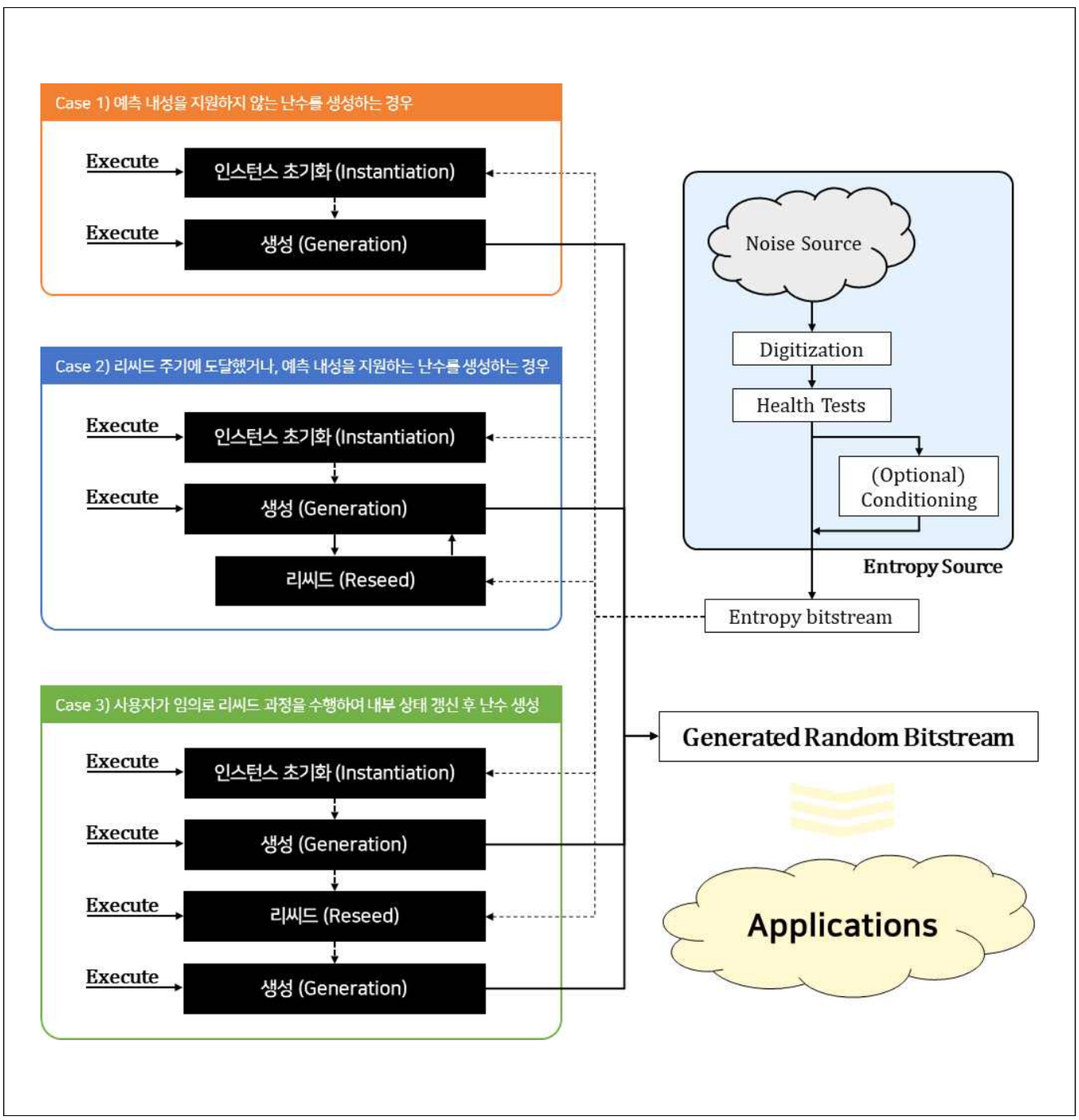
❑ 다음 요소들은 선택적인 요소이다.

구성요소	의 미
추가 입력 (Additional Input)	선택적으로 사용되는 입력값으로, 리씨드 및 생성 과정에서 활용
개별화 문자열 (Personalization String)	선택적으로 사용되는 입력값으로, 인스턴스 초기화 과정에서 개별화 정보를 제공하기 위해 사용됨 장치일련번호, 사용자 및 응용프로그램 ID 등이 사용될 수 있음
논스 (Nonce)	인스턴스 초기화 과정에 사용되는 반복되지 않는 가변적인 값으로 난수발생기 구현 방법에 따라 적용여부가 결정됨 예) 타임스탬프, 일련번호 또는 이들의 조합

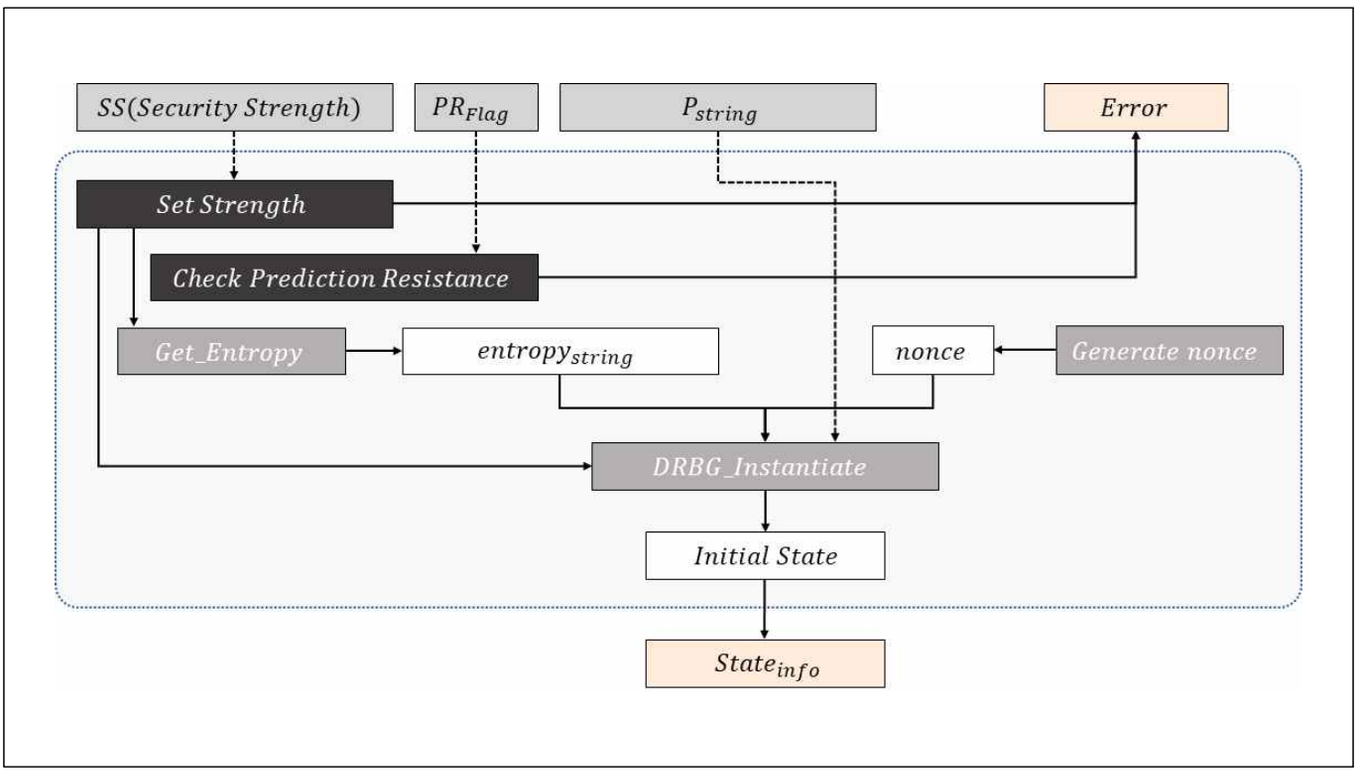
5 알고리즘 명세 및 구현 시 고려사항

가. 난수발생기 일반

☐ DRBG 난수 생성 과정



□ 인스턴스 초기화 단계



DRBG 인스턴스 초기화(*Instantiation*)		고려사항
입력	– 요구 보안강도 SS – 예측내성 지원 여부 PR_{Flag} ($TRUE$ 또는 $FALSE$) – 개별화 문자열 P_{string}	①
출력	– 내부 상태 또는 식별값 $State_{info}$	
1	$strength = min\{\, tmp \in \{112, 128, 192, 256\} : tmp \geq SS \,\}$	①
2	if$((SS > hash_security_strength) \| (strength > hash_security_strength))$ Error 출력	①
3	if$(PR_{Flag} == TRUE)$ && (구현물이 예측내성을 지원하지 않는 경우) Error 출력	
4	$entropy_{string} = Get\_Entropy(strength)$	②
5	$nonce$ 생성	③
6	$Initial\_State = \boldsymbol{DRBG\_Instantiate}(entropy_{string}, nonce, P_{string}, strength)$	
7	$State_{info} = [Initial\_State]$ or $[Indicator\ of\ Initial\_State]$	
8	$State_{info}$ 출력	

① 입력값 확인

요구 보안강도 SS : 사용되는 해시함수 기반 알고리즘의 보안강도보다 작거나 같아야 함
개별화 문자열 길이 확인 - $0 \leq Len(P_{string}) \leq 2^{35}$ 비트

② 엔트로피 수집 유효성 확인

엔트로피 수집함수가 건전성 테스트를 통과한 잡음원을 수집하는지 확인
수집된 엔트로피의 길이 및 보안강도 확인 - $Len(entropy_{string}) \leq 2^{35}$ 비트 - $entropy_{string}$의 엔트로피가 $strength$ 이상이어야 함

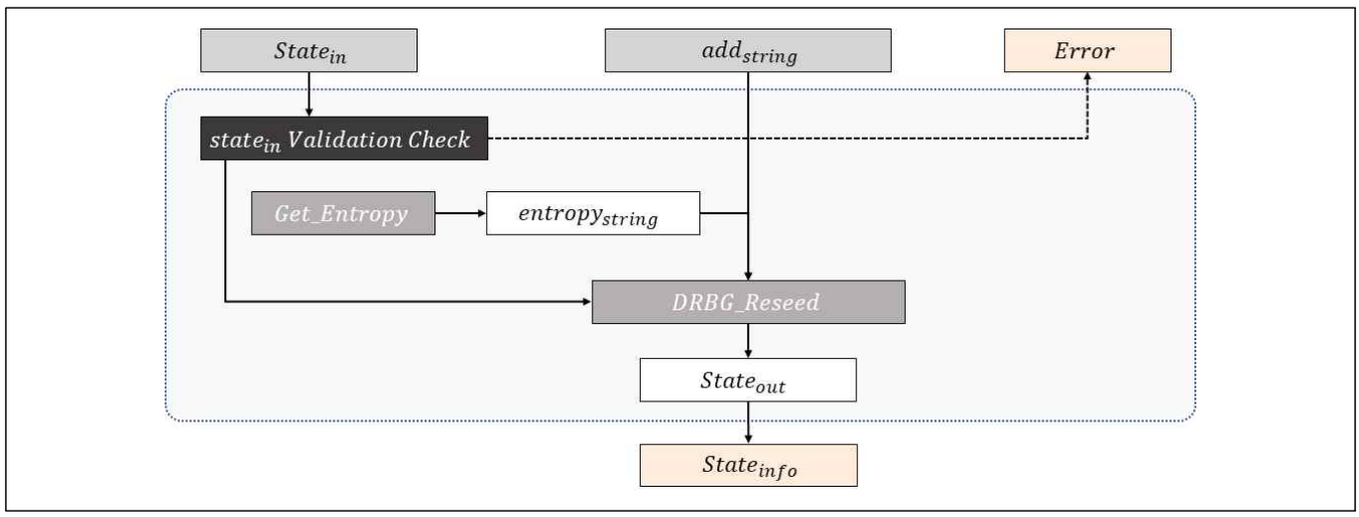
③ 논스 생성 확인

논스는 초기화 연산의 각 실행에 대해 유일해야 하며, 비밀값일 필요는 없음 - 사용된 논스 값은 CSP로 관리되어야 함
생성된 논스 길이 확인 - $Len(nonce) \geq (strength/2)$

[공통] 작동상태 관리 및 제로화 수행

각 난수발생기 인스턴스는 독립적인 내부상태를 가져야 함
사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] “안전한 제로화” 참고

□ 리씨드 단계



DRBG 리씨드(*Reseed*)		고려사항
입력	– 내부 상태 또는 식별값 $State_{in}$ – 추가 입력 add_{string}	①
출력	– (갱신된) 내부 상태 또는 식별값 $State_{info}$	
1	입력된 $State_{in}$ 유효성 확인	
2	$entropy_{string} = Get\_Entropy(strength)$	②
3	$State_{out} = \boldsymbol{DRBG\_Reseed}(State_{in}, entropy_{string}, add_{string})$	
4	$State_{info} = [State_{out}]$ or $[Indicator\ of\ State_{out}]$	
5	$State_{info}$ 출력	

① 입력값 확인

추가 입력 길이 확인
- $0 \leq Len(add_{string}) \leq 2^{35}$ 비트

② 엔트로피 수집 유효성 확인

엔트로피 수집함수가 건전성 테스트를 통과한 잡음원을 수집하는지 확인

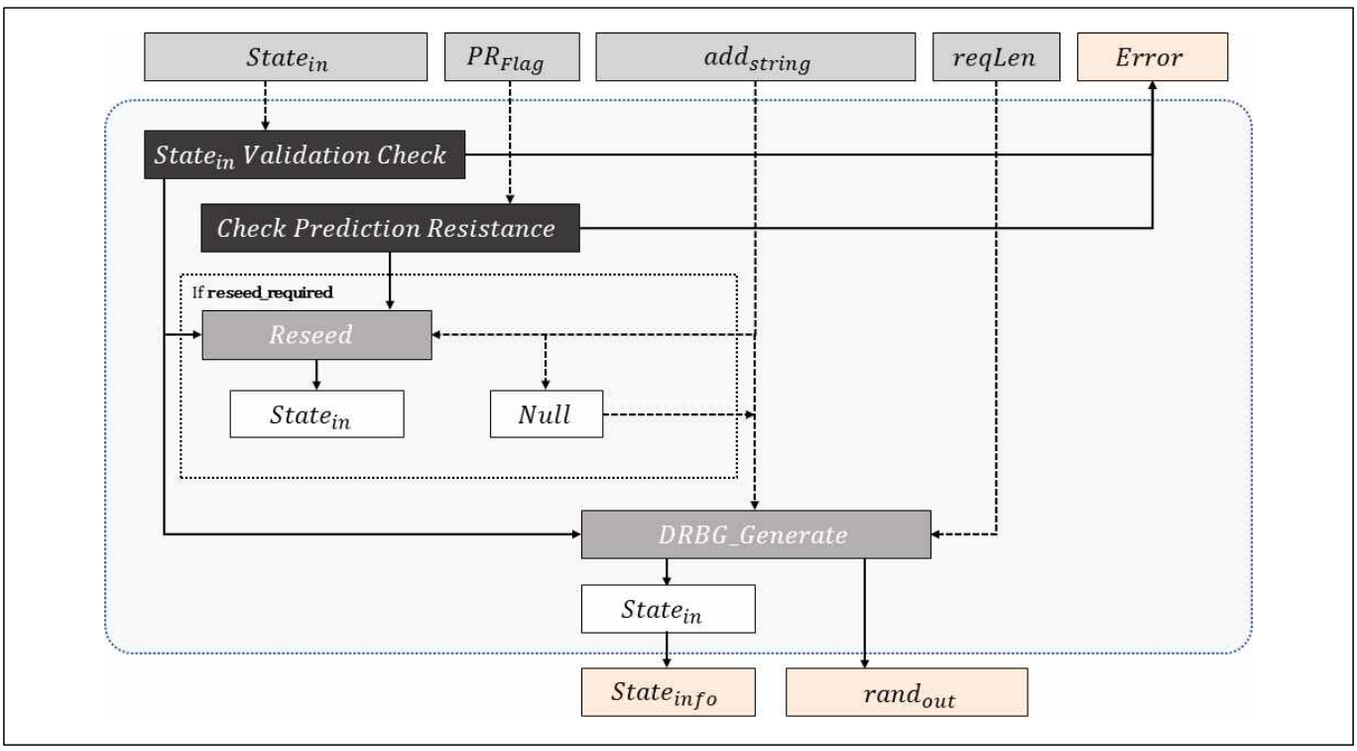
수집된 엔트로피의 길이 및 보안강도 확인
- $Len(entropy_{string}) \leq 2^{35}$ 비트
- $entropy_{string}$의 엔트로피가 $strength$ 이상이어야 함

[공통] 작동상태 관리 및 제로화 수행

각 난수발생기 인스턴스는 독립적인 내부상태를 유지해야 함

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] “안전한 제로화” 참고

□ 생성 단계



DRBG 생성(*Generation*)		고려사항
입력	– 내부 상태 또는 식별값 $State_{in}$ – 예측내성 요청 여부 PR_{Flag} ($TRUE$ 또는 $FALSE$) – 추가 입력 add_{string} – 난수 출력 요청 길이(비트) $reqLen$	①
출력	– (갱신된) 내부 상태 또는 식별값 $State_{info}$ – 생성된 난수 비트열 $rand_{out}$	
1	입력된 $State_{in}$ 유효성 확인	
2	if(구현물이 예측내성을 지원하지 않는 경우) && if($PR_{Flag} == TRUE$) Error 출력	
3	$reseed\_required\_flag = FALSE$	
4	if($reseed\_required\_flag == TRUE$) 또는 if($PR_{Flag} == TRUE$) $State_{in} = \textbf{Reseed}(State_{in}, add_{string})$ $add_{string} = Null$ $reseed\_required\_flag = FALSE$	
5	$(status, rand_{out}, State_{in}) = \textbf{DRBG\_Generate}(State_{in}, add_{string}, reqLen)$	
6	if($status == reseed\_required$) $reseed\_required\_flag = TRUE$ 단계 4로 이동	
7	$State_{info} = [State_{in}]$ or $[Indicator\ of\ State_{in}]$	
8	$State_{info}$ 및 $rand_{out}$ 출력	

① 입력값 확인

추가 입력 길이 확인 - $0 \leq Len(add_{string}) \leq 2^{35}$ 비트
난수 출력 요청 길이 확인 - $reqLen \leq 2^{19}$ 비트

[공통] 작동상태 관리 및 제로화 수행

각 난수발생기 인스턴스는 독립적인 내부상태를 유지해야 함
사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] "안전한 제로화" 참고

■ Hash-DRBG 및 HMAC-DRBG 내부 해시함수(SHA-2)별 주요 특징 (길이 단위 : 비트)

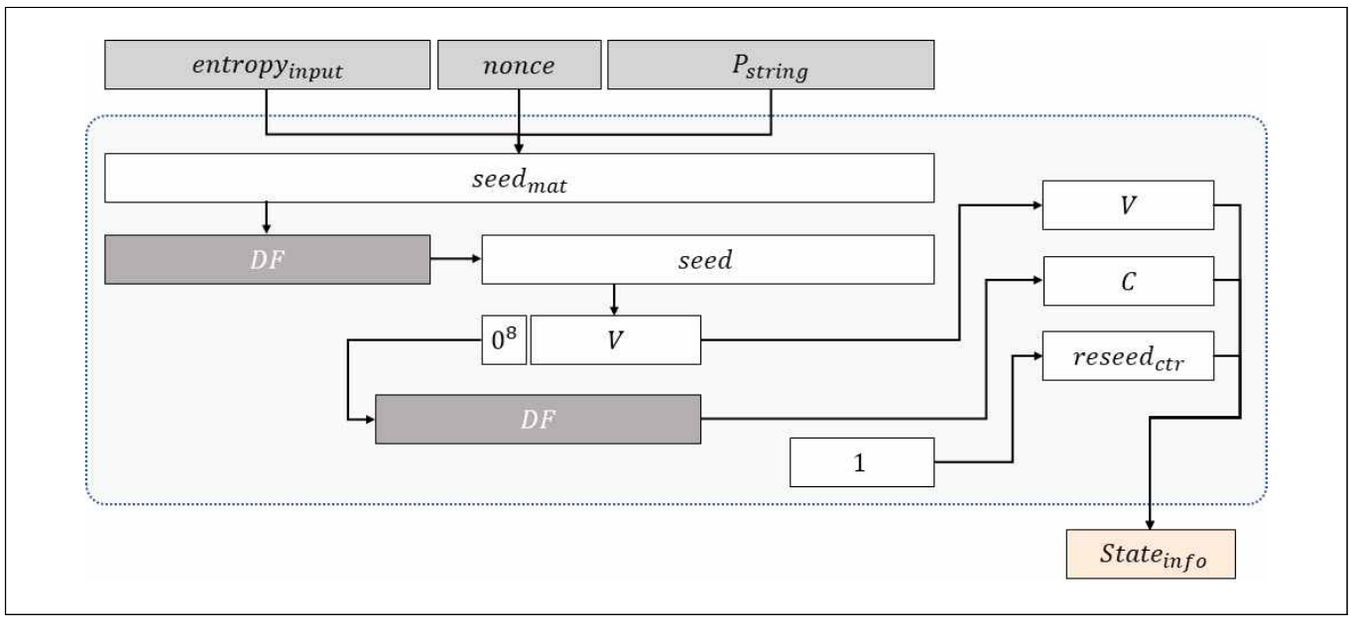
구분	224 Series	256 Series	384 Series	512 Series
보안강도 ($hash\_security\_strength$)	224	256	384	512
출력 길이 ($outlen$)	224	256	384	512
시드 길이 ($seedlen$)	440		888	
엔트로피 입력 최소 길이	$strength\ (in\ \textbf{\textit{Instantiation}}\ Phase)$			
엔트로피 입력 최대 길이	2^{35}			
개별화문자열 최대 길이	2^{35}			
추가입력 최대 길이	2^{35}			
난수 생성 최대 요청 길이	2^{19}			
리씨드 주기 ($reseed\_interval$)	2^{48}			

■ Hash-DRBG 및 HMAC-DRBG 내부 해시함수(LSH, SHA-3)별 주요 특징 (길이 단위 : 비트)

구분	224 Series	256 Series	384 Series	512 Series
보안강도 ($hash\_security\_strength$)	224	256	384	512
출력 길이 ($outlen$)	224	256	384	512
시드 길이 ($seedlen$)	440		888	
엔트로피 입력 최소 길이	$strength\ (in\ \textbf{\textit{Instantiation}}\ Phase)$			
엔트로피 입력 최대 길이	2^{35}			
개별화문자열 최대 길이	2^{35}			
추가입력 최대 길이	2^{35}			
난수 생성 최대 요청 길이	2^{19}			
리씨드 주기 ($reseed\_interval$)	2^{48}			

나. Hash-DRBG

□ 초기화 함수 (*DRBG_Instantiate*)



Hash-DRBG 초기화 함수(*DRBG_Instantiate*)		고려사항
입력	– 엔트로피 입력 $entropy_{input}$ – 논스 $nonce$ – 개별화 문자열 P_{string} – 보안강도 $strength$	①
출력	– 내부 상태 또는 식별값 $state_{info}$	
1	$seed_{mat} = entropy_{input} \| nonce \| P_{string}$	
2	$seed = \boldsymbol{DF}(seed_{mat}, seedlen)$	②
3	$V = seed$	
4	$C = \boldsymbol{DF}((0^8 \| V), seedlen)$	②
5	$reseed_{ctr} = 1$	
6	(V, C, $reseed_{ctr}$)가 포함된 $state_{info}$ 출력	

① 입력된 보안강도($strength$) 이상의 안전성을 갖는 해시함수를 사용하는지 확인

② 해시함수별 설정되는 시드 길이 확인

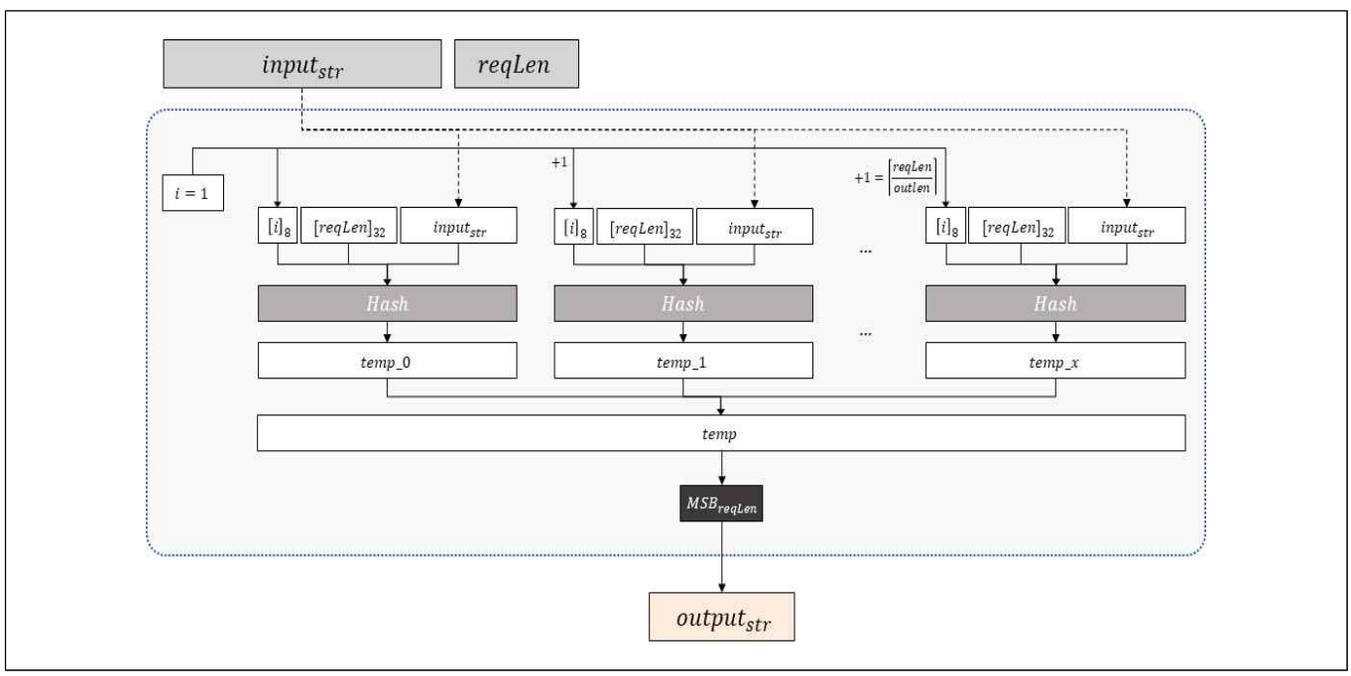
- SHA2-224, LSH-224/512_224, SHA3-224 : 440 비트
- SHA2-256, LSH-256/512_256, SHA3-256 : 440 비트
- SHA2-384, LSH-384, SHA3-384 : 888 비트
- SHA2-512, LSH-512, SHA3-512 : 888 비트

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] "안전한 제로화" 참고

□ 유도함수 (DF)



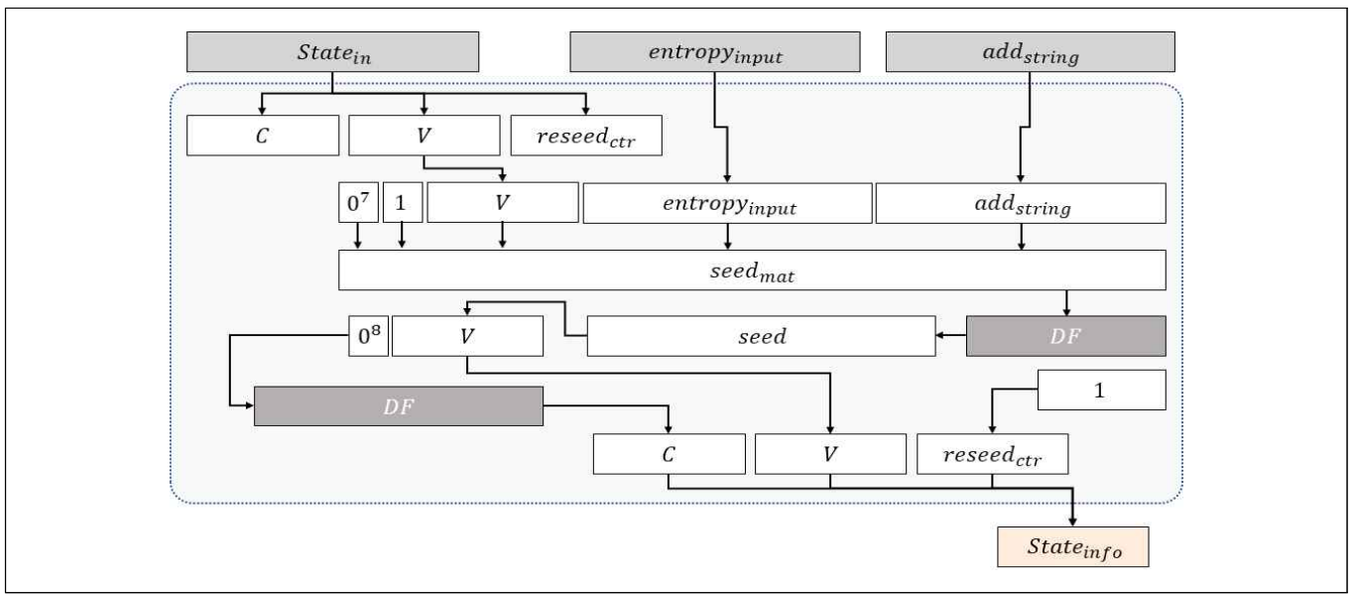
Hash-DRBG 유도함수(DF)		고려사항
입력	– 입력 비트열 $input_{str}$ – 요청 길이(비트) $reqLen$	
출력	– 출력 비트열 $output_{str}$	
1	$temp = Null$	
2	$len = \lceil \frac{reqLen}{outlen} \rceil$	
3	$counter = 1$	
4	for i from 1 to len do	
5	$temp = temp \parallel Hash([counter]_8 \parallel [reqLen]_{32} \parallel input_{str})$ $counter = counter + 1$	
6	end for	
7	$output_{str} = MSB_{reqLen}(temp)$	
8	$output_{str}$ 출력	

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] "안전한 제로화" 참고

□ 리씨드 함수 ($DRBG\_Reseed$)



Hash-DRBG 리씨드 함수(*DRBG_Reseed*)		고려사항
입력	– 내부 상태 또는 식별값 $state_{in}$ – 엔트로피 입력 $entropy_{input}$ – 추가 입력 add_{string}	
출력	– (갱신된) 내부 상태 또는 식별값 $state_{info}$	
1	$seed_{mat} = 0^7 \| 1 \| V \| entropy_{input} \| add_{string}$	
2	$seed = \boldsymbol{DF}(seed_{mat}, seedlen)$	①
3	$V = seed$	
4	$C = \boldsymbol{DF}((0^8 \| V), seedlen)$	①
5	$reseed_{ctr} = 1$	
6	(V, C, $reseed_{ctr}$)가 포함된 $state_{info}$ 출력	

① 해시함수별 설정되는 시드 길이 확인

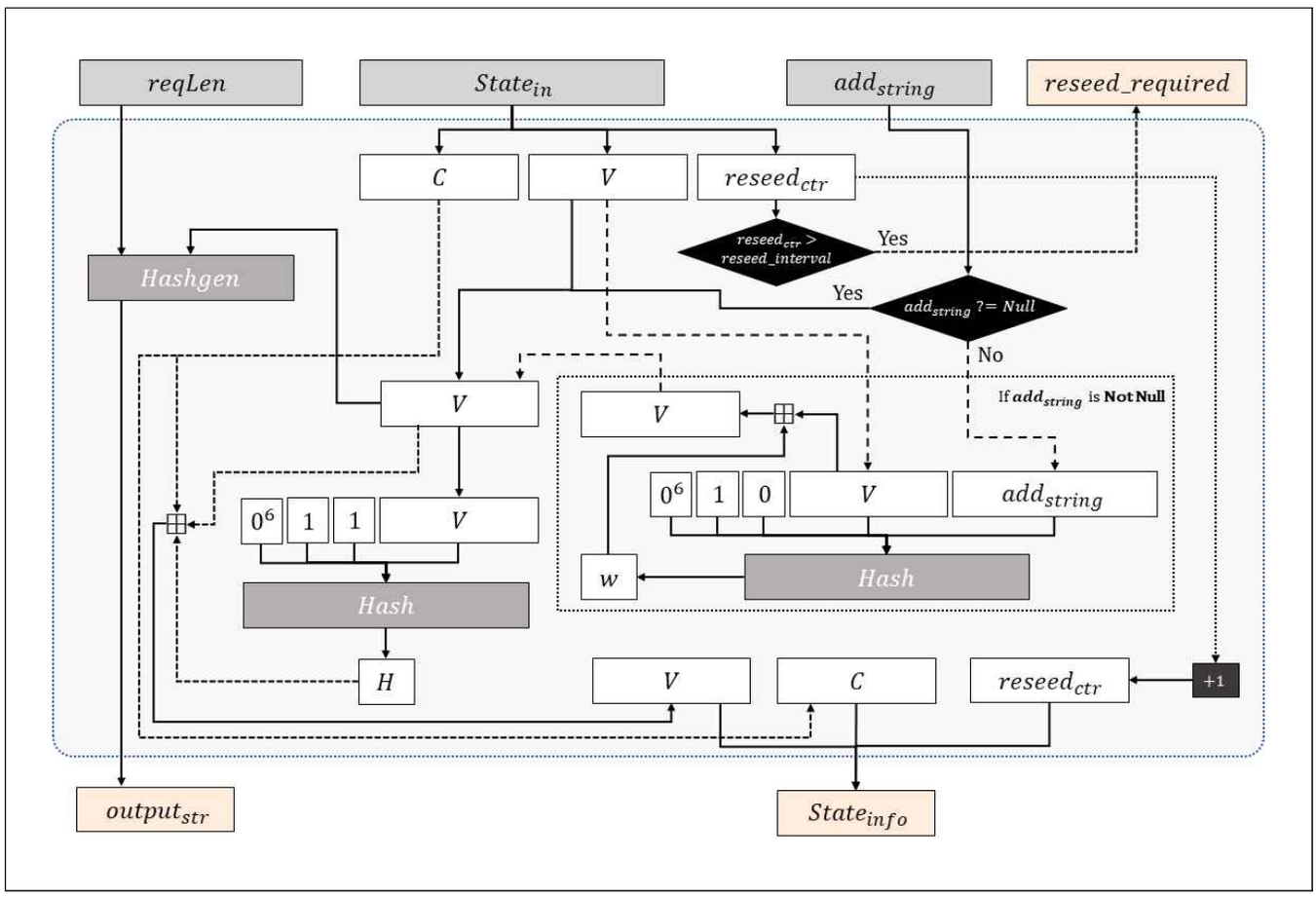
- SHA2-224, LSH-224/512_224, SHA3-224 : 440 비트
- SHA2-256, LSH-256/512_256, SHA3-256 : 440 비트
- SHA2-384, LSH-384, SHA3-384 : 888 비트
- SHA2-512, LSH-512, SHA3-512 : 888 비트

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] "안전한 제로화" 참고

□ 생성 함수 (*DRBG_Generate*)



Hashgen 함수(*Hashgen*)		고려사항
입력	– 요청 길이 $reqLen$ – 입력 비트열 $input_{str}$	
출력	– 출력 비트열 $output_{str}$	
1	$m = \left\lceil \frac{reqLen}{outlen} \right\rceil$	
2	$data = input_{str}$	
3	$W = Null$	
4	for i from 1 to m do	
5	$w = Hash(data)$ $W = W \| w$ $data = (data + 1) \bmod 2^{seedlen}$	
6	end for	
7	$output_{str} = MSB_{reqLen}(W)$	
8	$output_{str}$ 출력	

Hash-DRBG 생성 함수(*DRBG_Generate*)		고려사항
입력	– 내부 상태 또는 식별값 $state_{in}$ – 추가 입력 add_{string} – 난수 출력 요청 길이(비트) $reqLen$	
출력	– 동작 반환값 $indicator$ – 생성된 난수 비트열 $output_{str}$ – (갱신된) 내부 상태 또는 식별값 $state_{info}$	
1	if $(reseed_{ctr} > reseed\_interval)$ $indicator$ = $reseed\_required$ $indicator$ 출력	①
2	if $(add_{string} \neq Null)$ $w = Hash(0^6 \| 1 \| 0 \| V \| add_{string})$ $V = (V + w) \bmod 2^{seedlen}$	②
3	$output_{str} = \boldsymbol{Hashgen}(reqLen, V)$	
4	$H = Hash(0^6 \| 1 \| 1 \| V)$	
5	$V = (V + H + C + reseed_{ctr}) \bmod 2^{seedlen}$	②
6	$reseed_{ctr} = reseed_{ctr} + 1$	
7	$indicator = SUCCESS$	
8	$indicator$, $output_{str}$, (V, C, $reseed_{ctr}$)가 포함된 $state_{info}$ 출력	

① 리씨드 주기 확인

$reseed\_interval \leq 2^{48}$

② 해시함수별 설정되는 시드 길이 확인

- SHA2-224, LSH-224/512_224, SHA3-224 : 440 비트
- SHA2-256, LSH-256/512_256, SHA3-256 : 440 비트
- SHA2-384, LSH-384, SHA3-384 : 888 비트
- SHA2-512, LSH-512, SHA3-512 : 888 비트

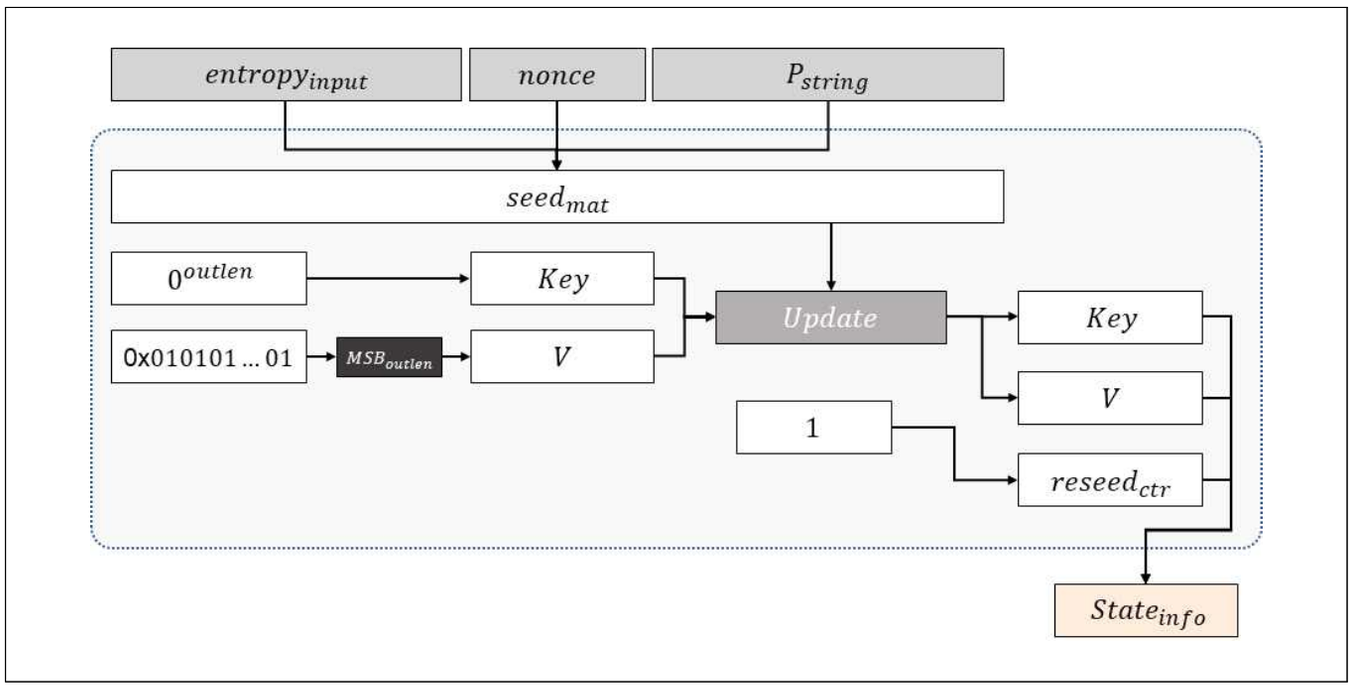
[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] “안전한 제로화” 참고

다. HMAC-DRBG

□ 초기화 함수 ($DRBG\_Instantiate$)



HMAC-DRBG 초기화 함수(*DRBG_Instantiate*)		고려사항
입력	– 엔트로피 입력 $entropy_{input}$ – 논스 $nonce$ – 개별화 문자열 P_{string} – 보안강도 $strength$	①
출력	– 내부 상태 또는 식별값 $state_{info}$	
1	$seed_{mat} = entropy_{input} \| nonce \| P_{string}$	
2	$Key = 0^{outlen}$	
3	$V = MSB_{outlen}((0^7 \| 1) \| (0^7 \| 1) \| ... \| (0^7 \| 1))$	
4	$(Key, V) = \textbf{\textit{Update}}\ (seed_{mat}, Key, V)$	
5	$reseed_{ctr} = 1$	
6	(Key, V, $reseed_{ctr}$)가 포함된 $state_{info}$ 출력	

① 입력된 보안강도($strength$) 이상의 안전성을 갖는 HMAC 알고리즘을 사용하는지 확인

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] "안전한 제로화" 참고

□ 업데이트 함수 ($Update$)

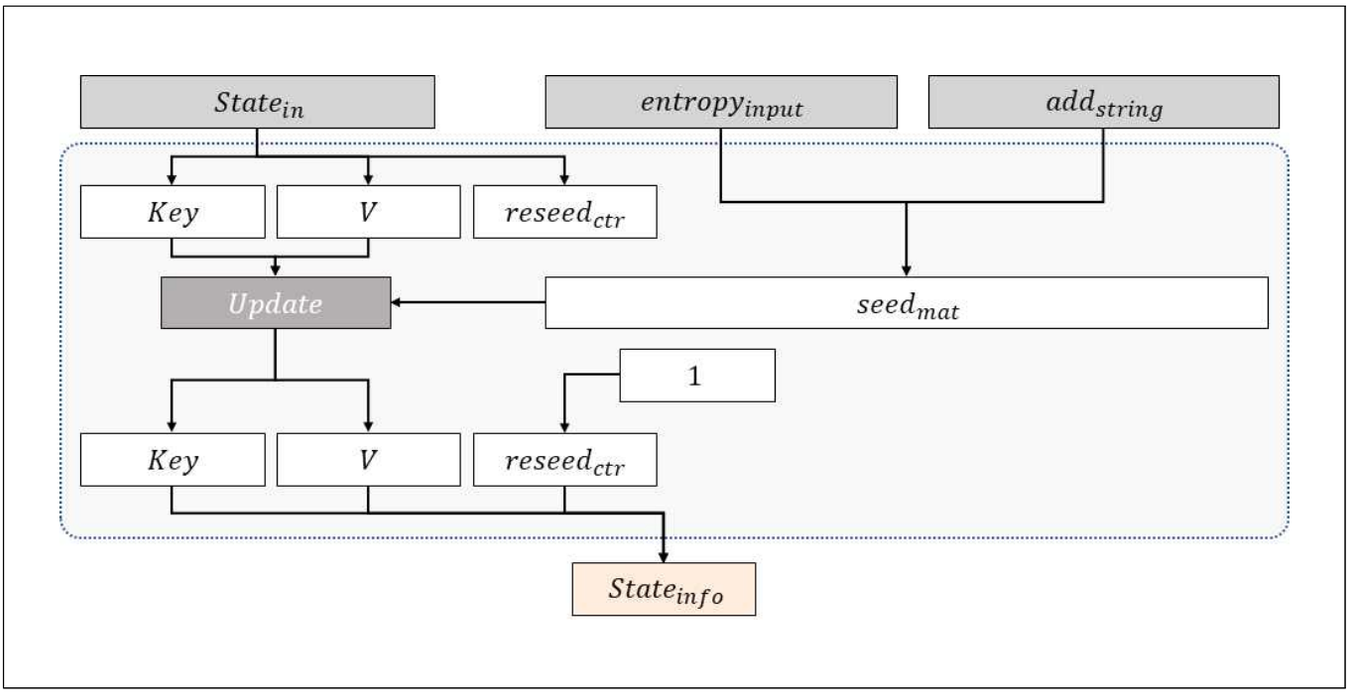
업데이트 함수(*Update*)		고려사항
입력	– 입력 비트열 $input_{str}$ – 난수발생기 상태값 Key – 난수발생기 상태값 V	
출력	– (갱신된) 난수발생기 상태값 Key – (갱신된) 난수발생기 상태값 V	
1	$Key = HMAC_{Key}(V \| 0^8 \| input_{str})$	
2	$V = HMAC_{Key}(V)$	
3	$\text{if}(input_{str} == Null)$ (Key, V) 출력	
4	$Key = HMAC_{Key}(V \| 0^7 \| 1 \| input_{str})$	
5	$V = HMAC_{Key}(V)$	
6	(Key, V) 출력	

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] “안전한 제로화” 참고

□ 리씨드 함수 ($DRBG\_Reseed$)



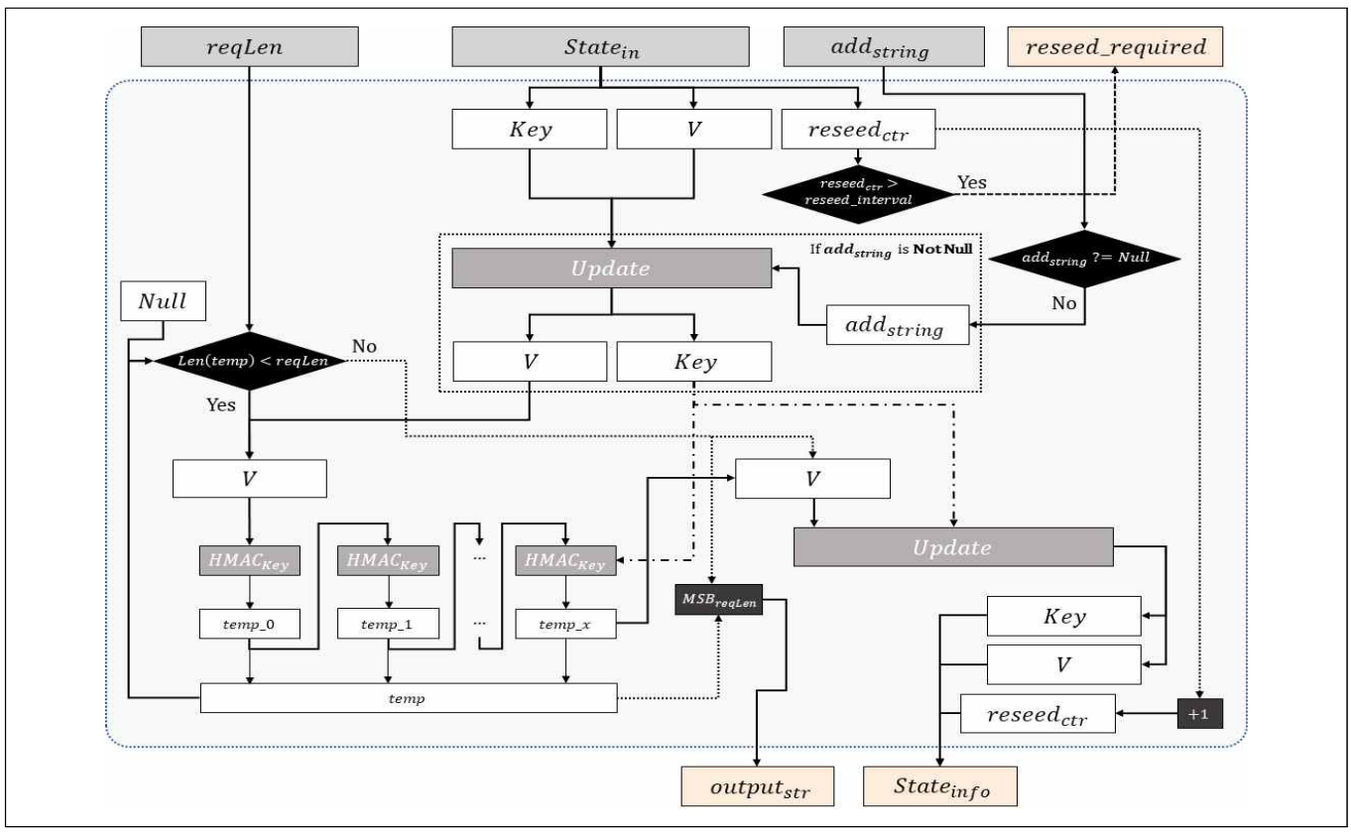
HMAC-DRBG 리씨드 함수($DRBG\_Reseed$)		고려사항
입력	– 내부 상태 또는 식별값 $state_{in}$ – 엔트로피 입력 $entropy_{input}$ – 추가 입력 add_{string}	
출력	– (갱신된) 내부 상태 또는 식별값 $state_{info}$	
1	$seed_{mat} = entropy_{input} \| add_{string}$	
2	$(Key, V) = \textbf{\textit{Update}}\ (seed_{mat}, Key, V)$	
3	$reseed_{ctr} = 1$	
4	(Key, V, $reseed_{ctr}$)가 포함된 $state_{info}$ 출력	

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] “안전한 제로화” 참고

□ 생성 함수 (*DRBG_Generate*)



HMAC-DRBG 생성 함수(*DRBG_Generate*)		고려사항
입력	– 내부 상태 또는 식별값 $state_{in}$ – 추가 입력 add_{string} – 난수 출력 요청 길이(비트) $reqLen$	
출력	– 동작 반환값 $indicator$ – 생성된 난수 비트열 $output_{str}$ – (갱신된) 내부 상태 또는 식별값 $state_{info}$	
1	if$(reseed_{ctr} > reseed_interval)$ $indicator$ = $reseed\_required$ $indicator$ 출력	①
2	if$(add_{string} \neq Null)$ $(Key, V) = \textbf{\textit{Update}}\ (add_{string}, Key, V)$	
3	$temp = Null$	
4	$While(Len(temp) < reqLen)$ $V = HMAC_{Key}(V)$ $temp = temp \parallel V$	
5	$output_{str} = MSB_{reqLen}(temp)$	
6	$(Key, V) = \textbf{\textit{Update}}\ (add_{string}, Key, V)$	
7	$reseed_{ctr} = reseed_{ctr} + 1$	
8	$indicator = SUCCESS$	
9	$indicator$, $output_{str}$, (Key, V, $reseed_{ctr}$)가 포함된 $state_{info}$ 출력	

① 리씨드 주기 확인

$reseed\_interval \leq 2^{48}$

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] “안전한 제로화” 참고

8장 RSA 기반 공개키 암호

RSA 기반 공개키 암호

1 범위

❑ 본 문서에서는 ISO/IEC 18033-2에 기술된 RSA 기반 공개키 암호알고리즘(RSAES)을 구현할 경우의 고려사항을 기술한다.

구분	파라미터
모듈러스(n)의 비트 크기	2048
	3072
해시함수	SHA2-224
	SHA2-256

2 관련표준

- [KS X ISO/IEC 18033-2] 암호 알고리즘 - 제2부: 비대칭형 암호 (2022)
- [ISO/IEC 18033-2] Encryption algorithms Part 2: Asymmetric ciphers (2017)
- [RFC 8017] PKCS #1: RSA Cryptography Specifications Version 2.2 (2016)

3 기호

기 호	의 미
$Len(X)$	비트열 X의 비트 길이
$GenPrime(X)$	X 비트 길이의 소수를 생성하는 함수
$LCM(A, B)$	A와 B의 최소공배수 (Largest Common Multiple)
$GCD(A, B)$	A와 B의 최대공약수 (Greatest Common Divisor)
$\|X\|$	$\|X\|$의 절대값 (Absoulte value)
$GenRandRange(A, B)$	A보다 크고 B보다 작은 임의의 난수값을 생성하는 함수 (A 초과 ~ B 미만)
$GenRandom(len)$	len비트 길이의 난수 비트열을 생성하는 함수
$Null$	Empty String
$\lceil n \rceil$	n보다 크거나 같은 정수 중에서 최소값
$[n]_s$	음이 아닌 정수 $n\,(n < 2^s)$의 s비트 이진수 표현
0^n	n길이의 연속된 0 비트열 (Ex. $0^5 = 00000,\ 0^3 = 000$)

기 호	의 미
1^n	n길이의 연속된 1 비트열 (Ex. $1^5 = 11111$, $1^3 = 111$)
$MSB_i(X)$	주어진 비트열 X의 상위(왼쪽) i개 비트열
$Hash(X)$	입력된 임의 길이의 메시지 X에 대해 고정된 길이의 해시값을 출력하는 해시함수
$StoI(X)$	비트열 X를 양의 정수값으로 변환하는 함수
$ItoS(N, len)$	양의 정수값 N을 len길이의 비트열로 변환하는 함수
$LPos_b(X)$	비트열 X의 MSB부터 시작하여 비트 b가 첫 번째로 나타나는 위치값 (Ex. $LPos_0(10110) = 2$, $LPos_1(00001) = 5$)

4 RSA 기반 공개키 암호

- RSA 암호는 정수 소인수 분해 문제의 어려움에 그 안전성의 기초를 두고 있다.
- RSA의 안전성에 영향을 끼치는 요소로는 모듈러스 n의 크기, 비밀정보로 사용되는 소수 p, q의 선택, 암호화 지수 e의 크기, 복호화 지수 d의 크기 등이 있다.

5 알고리즘 명세 및 구현 시 고려사항

가. 키 생성

■ 모듈러스 n : 비슷한 크기의 충분히 큰 서로 다른 소수 p, q를 선택하고 $n = p \times q$를 생성한다.

■ 암호화 지수 e : $GCD(e, \lambda(n)) = 1$을 만족하는 랜덤한 수 e를 선택한다.

여기서 $\lambda(n) = LCM(p-1, q-1)$이다.

■ 복호화 지수 d : $e \times d = 1 \pmod{\lambda(n)}$를 만족하는 유일한 정수 d를 선택한다.

키 생성 알고리즘(*RSAKeyGen*)		고려사항
입력	– 모듈러스 n의 비트 길이 $nLen$ – (Optional) 공개키 요소 e_{in}	①
출력	– 공개키 $PubKey = (n, e)$ – 개인키 $PrivKey = (n, d)$	
1	$p = GenPrime(nLen/2)$	②, ③
2	if($p < \sqrt{2}(2^{(nLen/2)-1})$) 단계 1로 이동	
3	$q = GenPrime(nLen/2)$	②, ③
4	if($q < \sqrt{2}(2^{(nLen/2)-1})$) 단계 3으로 이동	
5	if($\lvert p-q \rvert \leq 2^{(nLen/2-100)}$) 단계 3으로 이동	
6	$n = p \times q$	
7	$\lambda(n) = LCM(p-1, q-1)$	
8	$e = e_{in}$	
9	if($e_{in} == Null$) $e = GenRandRange(2^{16}, 2^{256})$	②

10	if($(e \leq 2^{16}) \| (e \geq 2^{256}) \| (e \bmod 2 == 0)$)
	if($e_{in} == Null$) then 단계 9로 이동
	$else$ then Error 출력	

11	if($GCD(e, \lambda(n)) != 1$)
	if($e_{in} == Null$) then 단계 9로 이동
	$else$ then 단계 1로 이동	

12	$d = e^{-1} \bmod \lambda(n)$	

13	if($Len(d) \leq (nLen/2)$)
	if($e_{in} == Null$) then 단계 9로 이동
	$else$ then 단계 1로 이동	

14	$PubKey = (n, e)$와 $PrivKey = (n, d)$ 출력	

① 검증대상 모듈러스 크기 확인

지원 가능한 모듈러스 크기 확인
- $nLen \in \{2048, 3072\}$ 비트

② 검증대상 난수발생기 사용 확인

난수를 생성하는 경우, 검증대상 난수발생기를 사용해야 함

③ 소수 생성방법 및 Miller-Rabin 소수 판정법 반복 횟수 확인

- 소수 생성방법은 암호모듈 구현안내서(GVI Part 1) 9.1 참조
- 소수 판정법으로는 Miller-Rabin 소수 판정법만이 허용되며 반복 횟수는 아래와 같음
  - $nLen == 2048$ 비트인 경우 : 최소 56회 수행 (오류 확률 : 2^{-112})
  - $nLen == 3072$ 비트인 경우 : 최소 64회 수행 (오류 확률 : 2^{-128})

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] "안전한 제로화" 참고

나. RSA-OAEP 암호화

■ RSA-OAEP 암호화는 메시지 인코딩 과정과 RSA 암호화 과정으로 구성된다.

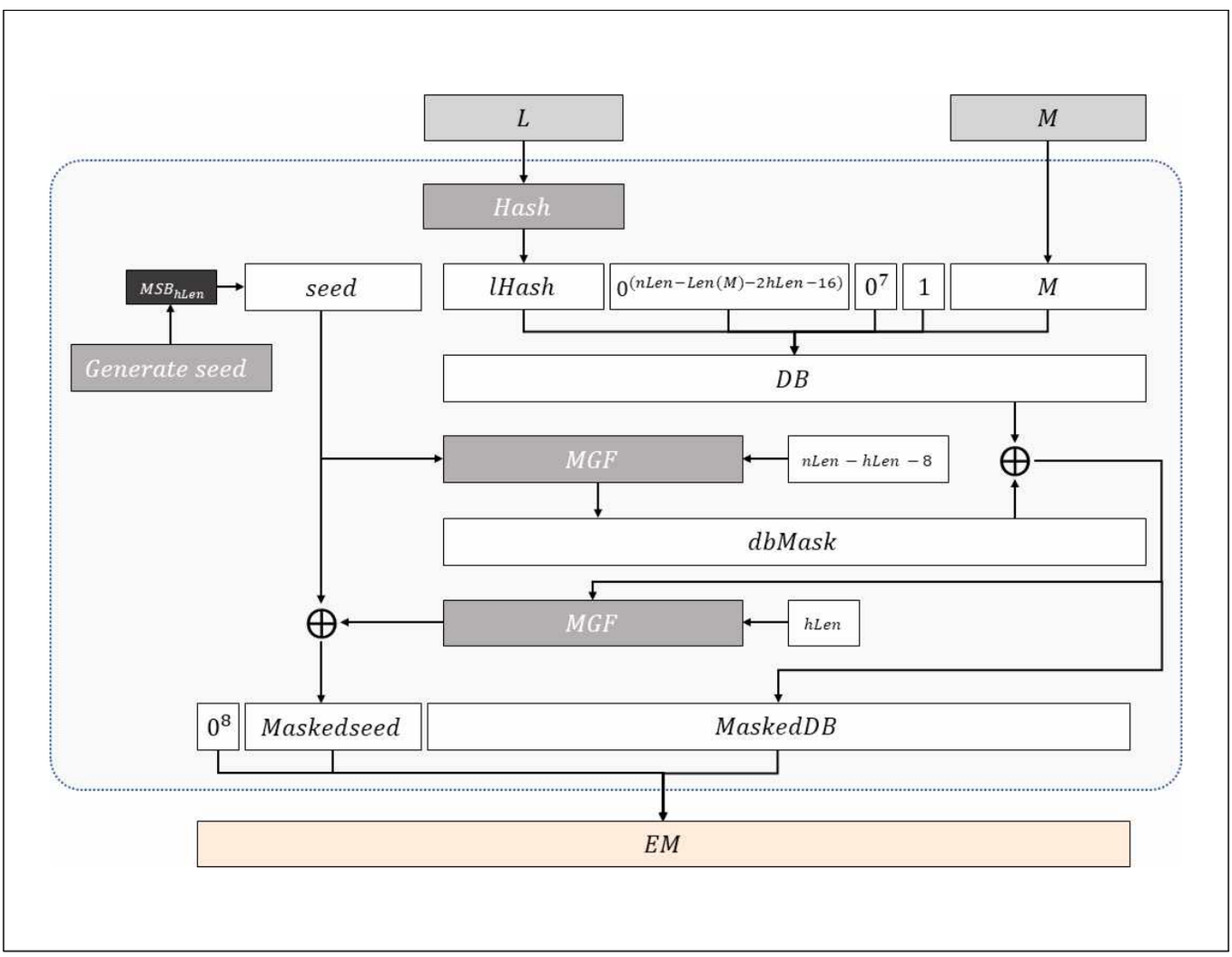
■ 메시지 인코딩 과정에서는 메시지 M과 라벨 L값을 이용하여 인코딩된 메시지 EM를 구한다.

$$EM = OAEP\_Encode(M, L)$$

■ RSA 암호화 과정에서는 인코딩된 메시지 EM에 대해서 공개키 (n, e)를 이용하여 암호문 C를 계산한다.

$$C = EM^e \bmod n$$

□ 메시지 인코딩 (*OAEP_Encode*)



인코딩 알고리즘(*OAEP_Encode*)		고려사항
설정	– $nLen$: 모듈러스 n의 비트 길이 – $Hash$: 내부 해시함수 (출력 길이 : $hLen$ 비트)	①, ②
입력	– 메시지 M – (Optional) 레이블 L	
출력	– 인코딩된 메시지 EM	
1	$lHash = Hash(L)$	
2	$PS = 0^{nLen - Len(M) - (2 \times hLen) - 16}$	
3	$DB = lHash \| PS \| 0^7 \| 1 \| M$	
4	$seed = GenRandom(hLen)$	③
5	$dbMask = MGF(seed, nLen - hLen - 8)$	
6	$maskedDB = DB \oplus dbMask$	
7	$seedMask = MGF(maskedDB, hLen)$	
8	$maskedSeed = seed \oplus seedMask$	
9	$EM = 0^8 \| maskedSeed \| maskedDB$	
10	인코딩된 메시지 EM 출력	

Mask 생성 함수(*MGF*)		고려사항
설정	– $Hash$: 내부 해시함수 (출력 길이 : $hLen$ 비트)	②
입력	– 씨드값 $seed$ – 생성할 마스킹 값의 길이(비트) $maskLen$	
출력	– 마스킹 결과값 $mask$	
1	$T = Null$	
2	for i from 0 to $(\lceil maskLen/hLen \rceil - 1)$ do	
3	$C = [i]_{32}$ $T = T \| Hash(seed \| C)$	
4	end for	
5	$mask = MSB_{maskLen}(T)$	
6	$mask$ 출력	

① 검증대상 모듈러스 크기 확인

지원 가능한 모듈러스 크기 확인
- $nLen \in \{2048, 3072\}$ 비트

② 검증대상 해시함수 사용 확인

내부 해시함수로 검증대상 해시함수를 사용해야 함
- SHA2-224, SHA2-256만 허용됨

③ 검증대상 난수발생기 사용 확인

$seed$ 생성 시, 검증대상 난수발생기를 사용해야 함

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] "안전한 제로화" 참고

□ RSA-OAEP 암호화 (*RSA_OAEP_Encrypt*)

RSA 암호화(*RSA_OAEP_Encrypt*)		고려사항
입력	– RSA 공개키 (n, e) – 메시지 M – (Optional) 레이블 L	①, ②, ③
출력	– 암호문 C	
1	$EM = OAEP\_Encode(M, L)$	
2	$m = StoI(EM)$	
3	$c = m^e \bmod n$	
4	$C = ItoS(c, Len(n))$	
5	암호문 C 출력	

① 검증대상 모듈러스 크기 확인

지원 가능한 모듈러스 크기 확인
- $Len(n) \in \{2048, 3072\}$ 비트

② 메시지 길이 확인

메시지 최대 길이 확인 : $Len(M) \leq (nLen - (2 \times hLen) - 16)$
1) $nLen == 2048$, SHA2-224를 사용하는 경우 : $Len(M) \leq 1{,}584$ 비트 2) $nLen == 2048$, SHA2-256을 사용하는 경우 : $Len(M) \leq 1{,}520$ 비트 3) $nLen == 3072$, SHA2-256을 사용하는 경우 : $Len(M) \leq 2{,}544$ 비트

③ 레이블 L 길이 확인

레이블 L은 해시함수의 입력 메시지 길이 조건을 충족해야 함

- SHA2-224, SHA2-256 : $Len(L) \leq 2^{64} - 1$ 비트

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] “안전한 제로화” 참고

다. RSA-OAEP 복호화

■ RSA-OAEP 복호화는 RSA 복호화 과정과 메시지 디코딩 과정으로 구성된다.

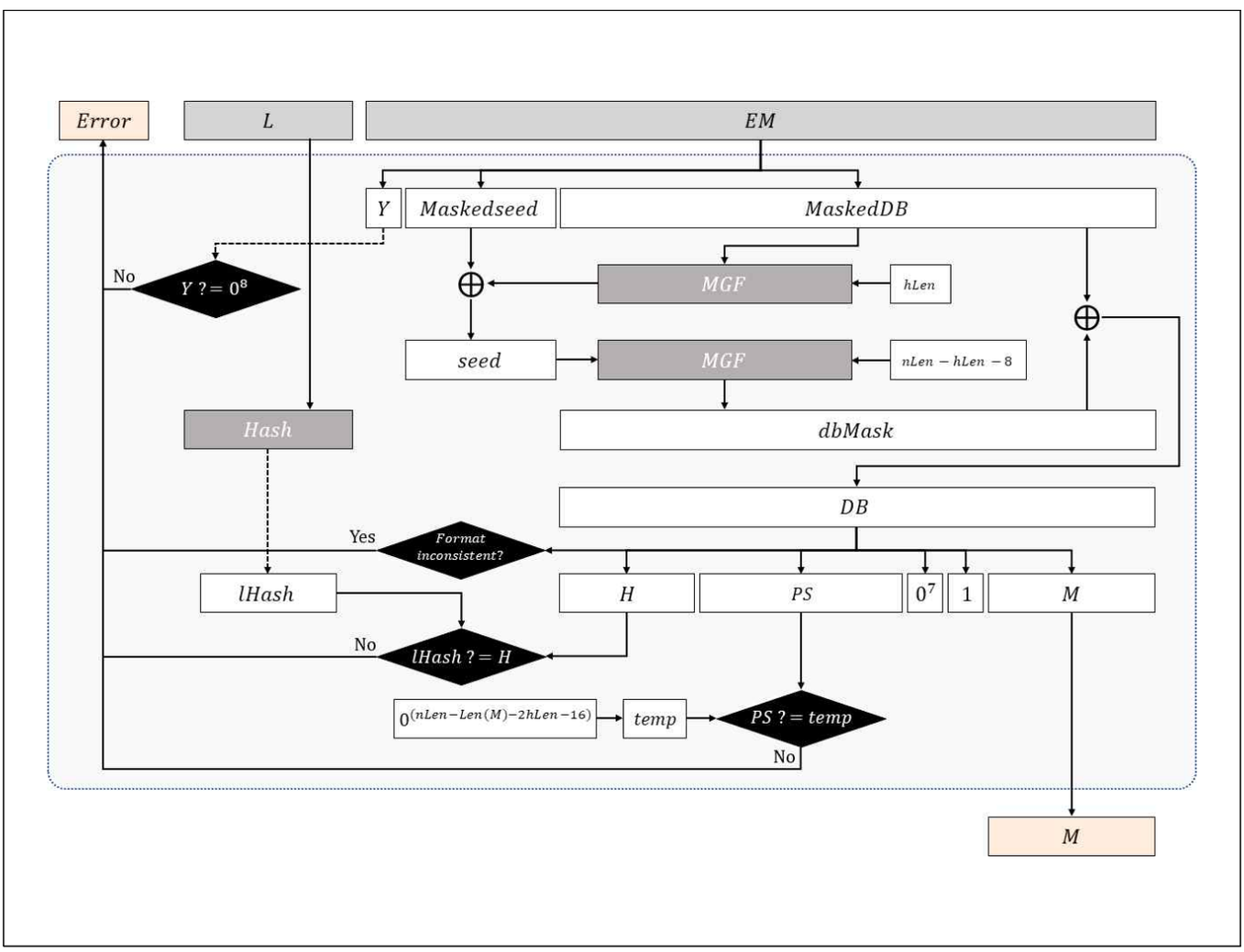
■ RSA 복호화 과정에서는 암호문 C와 개인키 (n, d)를 이용하여 인코딩된 메시지 EM을 계산한다.

$$EM = C^d \bmod n$$

■ 메시지 디코딩 과정에서는 인코딩된 메시지 EM과 라벨 L값을 이용하여 메시지 M을 구한다.

$$M = OAEP\_Decode(EM, L)$$

□ 메시지 디코딩 (*OAEP_Decode*)



디코딩 알고리즘(*OAEP_Decode*)		고려사항
설정	– $nLen$: 모듈러스 n의 비트 길이 – $Hash$: 내부 해시함수 (출력 길이 : $hLen$ 비트)	①, ②
입력	– 인코딩된 메시지 EM – (Optional) 레이블 L	
출력	– 메시지 M	
1	$lHash = Hash(L)$	
2	다음과 같이 EM을 분리 $EM = Y \| maskedSeed \| maskedDB$ $(Len(Y) == 8, Len(maskedSeed) == hLen, Len(maskedDB) == nLen - hLen - 8)$	
3	$\text{if}(Y != 0^8)$ Error 출력	
4	$seedMask = MGF(maskedDB, hLen)$	
5	$seed = maskedSeed \oplus seedMask$	
6	$dbMask = MGF(seed, nLen - hLen - 8)$	
7	$DB = maskedDB \oplus dbMask$	
8	$H = MSB_{hLen}(DB)$	
9	$DB = LSB_{Len(DB)-hLen}(DB)$	
10	$pos = LPos_1(DB)$	
11	$M = LSB_{Len(DB)-pos}(DB)$	
12	$PS = MSB_{pos}(DB)$	
13	$\text{if}(LSB_8(PS) != '00000001')$ Error 출력	
14	$PS = MSB_{pos-8}(PS)$	
15	$\text{if}((lHash != H) \| (PS != 0^{nLen - Len(M) - (2 \times hLen) - 16}))$ Error 출력	
16	메시지 M 출력	

① 검증대상 모듈러스 크기 확인

지원 가능한 모듈러스 크기 확인
– $nLen \in \{2048, 3072\}$ 비트

② 검증대상 해시함수 사용 확인

내부 해시함수로 검증대상 해시함수를 사용해야 함
– SHA2-224, SHA2-256만 허용됨

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] “안전한 제로화” 참고

□ RSA-OAEP 복호화 (*RSA_OAEP_Decrypt*)

RSA 복호화(*RSA_OAEP_Decrypt*)		고려사항
입력	– RSA 개인키 (n, d) – 암호문 C – (Optional) 레이블 L	①, ②, ③
출력	– 메시지 M	
1	$c = StoI(C)$	
2	$m = c^d \bmod n$	
3	$EM = ItoS(m, Len(n))$	
4	$M = OAEP\_Decode(EM, L)$	
5	$OAEP\_Decode$ 과정에서 Error가 발생하는 경우 Error 출력	
6	메시지 M 출력	

① 검증대상 모듈러스 크기 확인

지원 가능한 모듈러스 크기 확인
– $Len(n) \in \{2048, 3072\}$ 비트

② 암호문 C 길이 확인

$Len(C) \leq Len(n)$

③ 레이블 L 길이 확인

레이블 L은 해시함수의 입력 메시지 길이 조건을 충족해야 함
– SHA2-224, SHA2-256 : $Len(L) \leq 2^{64} - 1$ 비트

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] “안전한 제로화” 참고

9장 RSA 기반 전자서명

RSA 기반 전자서명

1 범위

❑ 본 문서에서는 ISO/IEC 14888-2에 기술된 RSA 기반 전자서명 알고리즘(RSASSA-PSS)를 구현할 경우의 고려사항을 기술한다.

구분	파라미터
모듈러스(n)의 비트 크기	2048
	3072
해시함수	SHA2-224
	SHA2-256

2 관련표준

- [KS X ISO/IEC 14888-2] 부가형 디지털 서명 – 제2부: 정수 인수분해 기반 메커니즘 (2021)
- [ISO/IEC 14888-2] Digital signatures with appendix Part 2: Integer factorization based mechanisms (2015)
- [FIPS 186-5] Digital Signature Standard (DSS) (2023)
- [RFC 8017] PKCS #1: RSA Cryptography Specifications Version 2.2 (2016)

3 기호

기 호	의 미
$Len(X)$	비트열 X의 비트 길이
$GenPrime(X)$	X 비트 길이의 소수를 생성하는 함수
$LCM(A, B)$	A와 B의 최소공배수 (Largest Common Multiple)
$GCD(A, B)$	A와 B의 최대공약수 (Greatest Common Divisor)
$\|X\|$	$\|X\|$의 절대값 (Absoulte value)
$GenRandRange(A, B)$	A보다 크고 B보다 작은 임의의 난수값을 생성하는 함수 (A 초과 ~ B 미만)
$GenRandom(len)$	len비트 길이의 난수 비트열을 생성하는 함수
$Null$	Empty String
$\lceil n \rceil$	n보다 크거나 같은 정수 중에서 최소값
$[n]_s$	음이 아닌 정수 $n\,(n < 2^s)$의 s비트 이진수 표현

기 호	의 미
0^n	n길이의 연속된 0 비트열 (Ex. $0^5 = 00000$, $0^3 = 000$)
1^n	n길이의 연속된 1 비트열 (Ex. $1^5 = 11111$, $1^3 = 111$)
$MSB_i(X)$	주어진 비트열 X의 상위(왼쪽) i개 비트열
$LSB_i(X)$	주어진 비트열 X의 하위(오른쪽) i개 비트열
$Hash(X)$	입력된 임의 길이의 메시지 X에 대해 고정된 길이의 해시값을 출력하는 해시함수
$StoI(X)$	비트열 X를 양의 정수값으로 변환하는 함수
$ItoS(N, len)$	양의 정수값 N을 len길이의 비트열로 변환하는 함수
$LPos_b(X)$	비트열 X의 MSB부터 시작하여 비트 b가 첫 번째로 나타나는 위치값 (Ex. $LPos_0(10110) = 2$, $LPos_1(00001) = 5$)

4 RSA 기반 전자서명

- RSA 암호는 정수 소인수 분해 문제의 어려움에 그 안전성의 기초를 두고 있다.
- RSA의 안전성에 영향을 끼치는 요소로는 모듈러스 n의 크기, 비밀정보로 사용되는 소수 p, q의 선택, 서명 생성 지수 d의 크기, 서명 검증 지수 e의 크기 등이 있다.
- RSA 기반 전자서명 단계는 키 생성, 전자서명 생성, 전자서명 검증으로 구분된다.

5 알고리즘 명세 및 구현 시 고려사항

가. 키 생성

■ 모듈러스 n : 비슷한 크기의 충분히 큰 서로 다른 소수 p, q를 선택하고 $n = p \times q$를 생성한다.

■ 암호화 지수 e : $GCD(e, \lambda(n)) = 1$을 만족하는 랜덤한 수 e를 선택한다.

여기서 $\lambda(n) = LCM(p-1, q-1)$이다.

■ 복호화 지수 d : $e \times d = 1 \pmod{\lambda(n)}$를 만족하는 유일한 정수 d를 선택한다.

키 생성 알고리즘(*RSAKeyGen*)		고려사항
입력	– 모듈러스 n의 비트 길이 $nLen$ – (Optional) 공개키 요소 e_{in}	①
출력	– 공개키 $PubKey = (n, e)$ – 개인키 $PrivKey = (n, d)$	
1	$p = GenPrime(nLen/2)$	②, ③
2	if($p < \sqrt{2}(2^{(nLen/2)-1})$) 단계 1로 이동	
3	$q = GenPrime(nLen/2)$	②, ③
4	if($q < \sqrt{2}(2^{(nLen/2)-1})$) 단계 3으로 이동	
5	if($\lvert p-q \rvert \leq 2^{(nLen/2-100)}$) 단계 3으로 이동	
6	$n = p \times q$	
7	$\lambda(n) = LCM(p-1, q-1)$	
8	$e = e_{in}$	
9	if($e_{in} == Null$) $e = GenRandRange(2^{16}, 2^{256})$	②

10	if($(e \leq 2^{16}) \| (e \geq 2^{256}) \| (e \bmod 2 == 0)$)
	if($e_{in} == Null$) then 단계 9로 이동
	$else$ then Error 출력	

11	if($GCD(e, \lambda(n)) != 1$)
	if($e_{in} == Null$) then 단계 9로 이동
	$else$ then 단계 1로 이동	

12	$d = e^{-1} \bmod \lambda(n)$	

13	if($Len(d) \leq (nLen/2)$)
	if($e_{in} == Null$) then 단계 9로 이동
	$else$ then 단계 1로 이동	

14	$PubKey = (n, e)$와 $PrivKey = (n, d)$ 출력	

① 검증대상 모듈러스 크기 확인

지원 가능한 모듈러스 크기 확인

– $nLen \in \{2048, 3072\}$ 비트

② 검증대상 난수발생기 사용 확인

난수를 생성하는 경우, 검증대상 난수발생기를 사용해야 함

③ 소수 생성방법 및 Miller-Rabin 소수 판정법 반복 횟수 확인

- 소수 생성방법은 암호모듈 구현안내서(GVI Part 1) 9.1 참조
- 소수 판정법으로는 Miller-Rabin 소수 판정법만이 허용되며 반복 횟수는 아래와 같음
  - $nLen == 2048$ 비트인 경우 : 최소 56회 수행 (오류 확률 : 2^{-112})
  - $nLen == 3072$ 비트인 경우 : 최소 64회 수행 (오류 확률 : 2^{-128})

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] “안전한 제로화” 참고

나. RSASSA-PSS 서명 생성

■ RSASSA-PSS 서명 생성은 메시지 인코딩 과정과 RSA 서명 과정으로 구성된다.

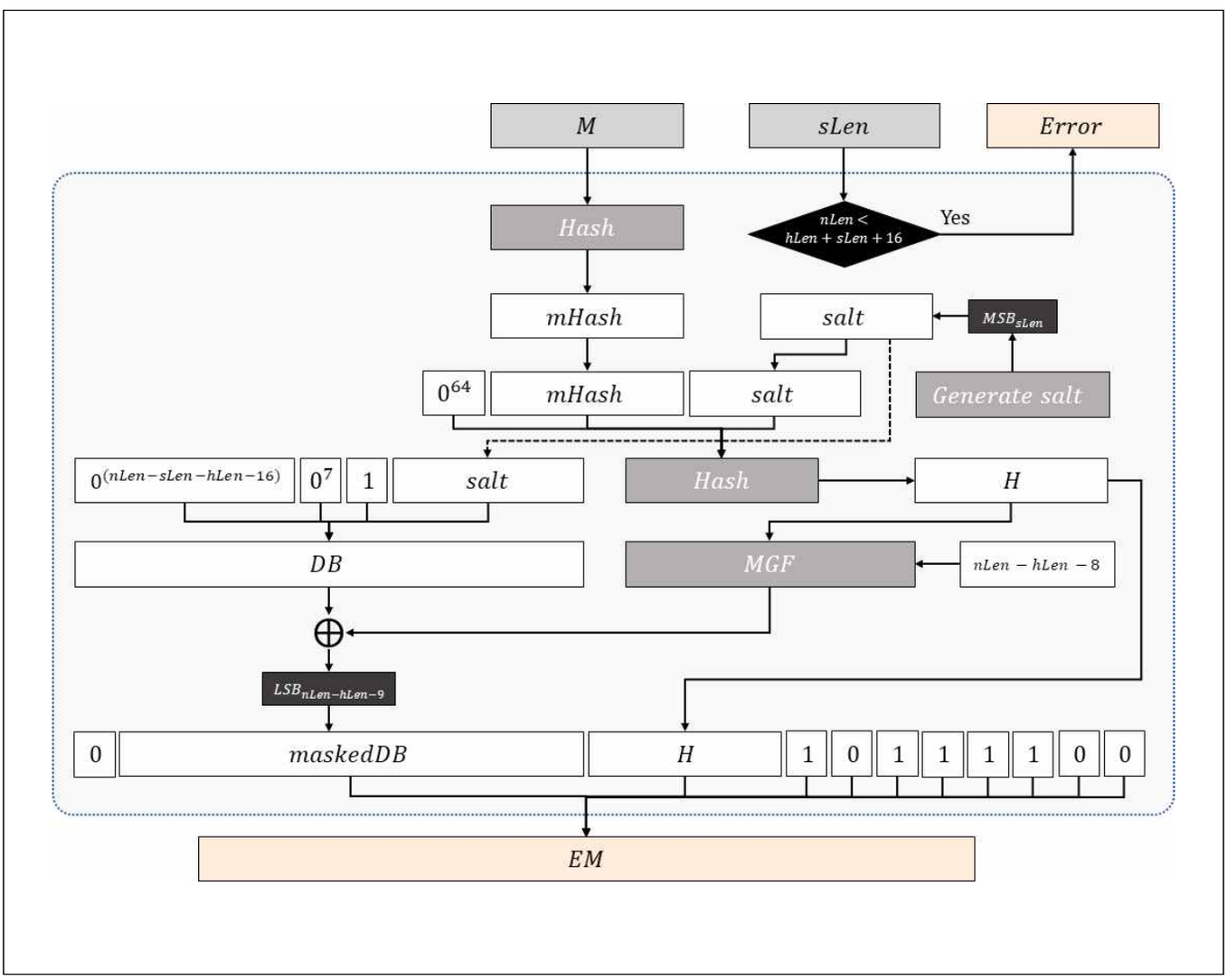
■ 메시지 인코딩 과정에서는 메시지 M을 이용하여 인코딩된 메시지 EM를 구한다.

$$EM = PSS\_Encode(M)$$

■ RSA 서명 과정에서는 인코딩된 메시지 EM에 대해서 개인키 (n, d)를 이용하여 서명 S를 계산한다.

$$S = EM^d \bmod n$$

□ 메시지 인코딩 (*PSS_Encode*)



인코딩 알고리즘(*PSS_Encode*)		고려사항
설정	– $nLen$: 모듈러스 n의 비트 길이 – $Hash$: 내부 해시함수 (출력 길이 : $hLen$ 비트)	①, ②
입력	– 메시지 M – 내부에서 생성될 $salt$의 비트 길이 $sLen$	
출력	– 인코딩된 메시지 EM	
1	$mHash = Hash(M)$	
2	$\text{if}(nLen < hLen + sLen + 16)$ Error 출력	
3	$salt = GenRandom(sLen)$	③
4	$tempM = 0^{64} \| mHash \| salt$	
5	$H = Hash(tempM)$	
6	$PS = 0^{nLen - sLen - hLen - 16}$	
7	$DB = PS \| 0^7 \| 1 \| salt$	
8	$dbMask = MGF(H, nLen - hLen - 8)$	
9	$maskedDB = DB \oplus dbMask$	
10	$maskedDB = 0 \| LSB_{nLen - hLen - 9}(maskedDB)$	
11	$EM = maskedDB \| H \| 1 \| 0 \| 1 \| 1 \| 1 \| 1 \| 0 \| 0$	
12	인코딩된 메시지 EM 출력	

Mask 생성 함수(*MGF*)		고려사항
설정	– $Hash$: 내부 해시함수 (출력 길이 : $hLen$ 비트)	②
입력	– 씨드값 $seed$ – 생성할 마스킹 값의 길이(비트) $maskLen$	
출력	– 마스킹 결과값 $mask$	
1	$T = Null$	
2	for i from 0 to $(\lceil maskLen/hLen \rceil - 1)$ do	
3	$C = [i]_{32}$ $T = T \| Hash(seed \| C)$	
4	end for	
5	$mask = MSB_{maskLen}(T)$	
6	$mask$ 출력	

① 검증대상 모듈러스 크기 확인

지원 가능한 모듈러스 크기 확인
- $nLen \in \{2048, 3072\}$ 비트

② 검증대상 해시함수 사용 확인

내부 해시함수로 검증대상 해시함수를 사용해야 함
- SHA2-224, SHA2-256만 허용됨

③ 검증대상 난수발생기 사용 확인

$salt$ 생성 시, 검증대상 난수발생기를 사용해야 함

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] "안전한 제로화" 참고

□ RSASSA-PSS 서명 생성 (RSA_PSS_Sign)

RSA 서명 생성(*RSA_PSS_Sign*)		고려사항
입력	– RSA 개인키 (n, d) – 메시지 M	①, ②
출력	– 전자서명값 S	
1	$sLen = GenRandRange(hLen - 1, Len(n) - hLen - 15)$	③
2	$EM = PSS\_Encode(M, sLen)$	
3	$m = StoI(EM)$	
4	$s = m^d \bmod n$	
5	$S = ItoS(s, Len(n))$	
6	전자서명값 S 출력	

① 검증대상 모듈러스 크기 확인

지원 가능한 모듈러스 크기 확인
- $Len(n) \in \{2048, 3072\}$ 비트

② 메시지 길이 확인

메시지 M은 해시함수의 입력 메시지 길이 조건을 충족해야 함

- SHA2-224, SHA2-256 : $Len(M) \leq 2^{64} - 1$ 비트

③ $salt$ 길이($sLen$) 확인

$sLen$의 길이는 해시함수 출력 길이 이상이어야 함

1) $nLen == 2048$, SHA2-224를 사용하는 경우 : $224 \leq sLen \leq 1{,}808$ 비트
2) $nLen == 2048$, SHA2-256을 사용하는 경우 : $256 \leq sLen \leq 1{,}776$ 비트
3) $nLen == 3072$, SHA2-256을 사용하는 경우 : $256 \leq sLen \leq 2{,}800$ 비트

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] "안전한 제로화" 참고

다. RSASSA-PSS 서명 검증

■ RSASSA-PSS 서명 검증은 RSA 검증변환 과정과 메시지 디코딩 및 검증 과정으로 구성된다.

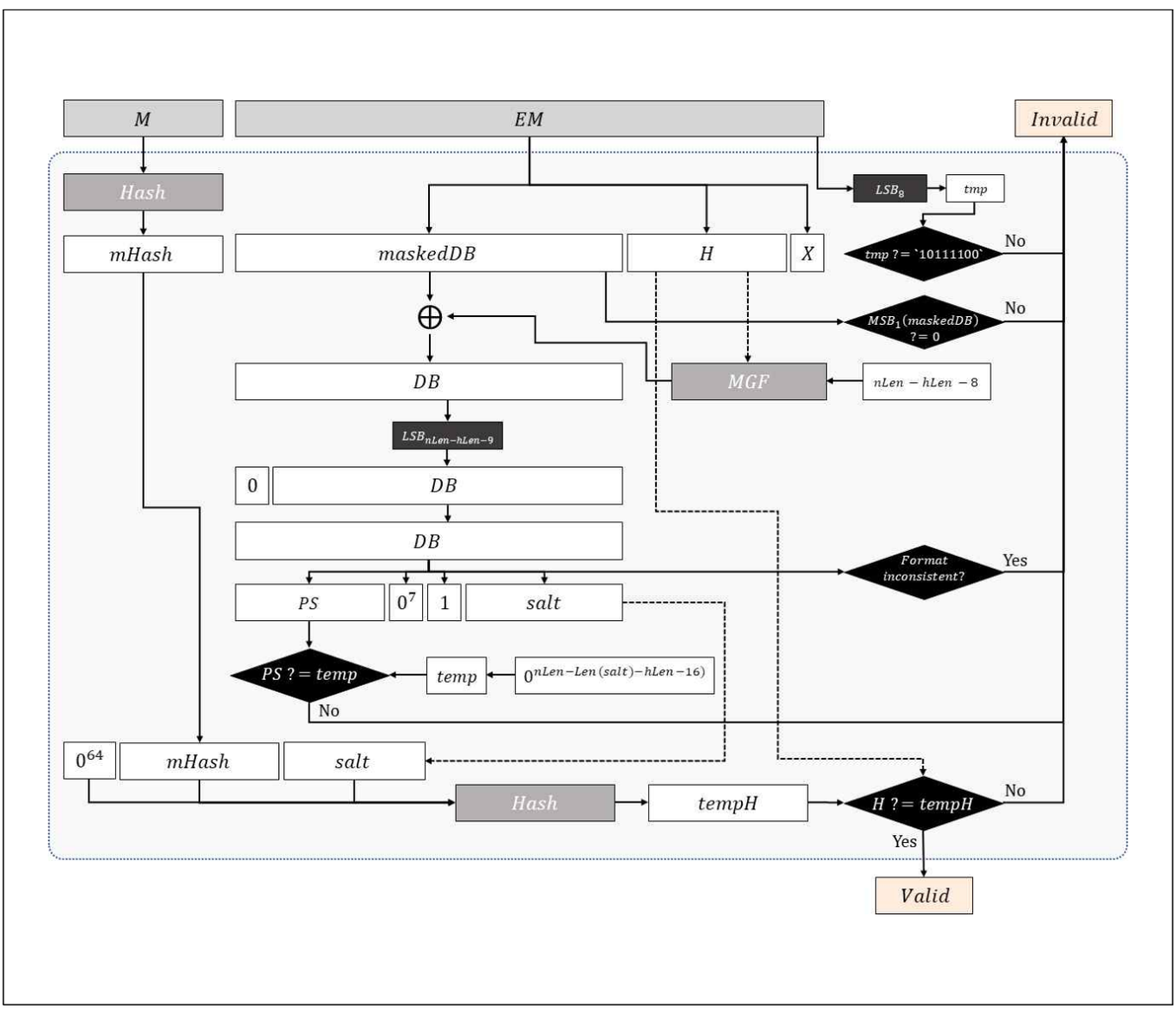
■ RSA 검증변환 과정에서는 서명 S와 공개키 (n, e)를 이용하여 인코딩된 메시지 EM을 계산한다.

$$EM = S^e \bmod n$$

■ 메시지 디코딩 및 검증 과정에서는 인코딩된 메시지 EM과 메시지 M값을 이용하여 서명 진위 여부를 판단한다.

$$Result = PSS\_Decode\,Verify(EM, M)$$

□ 메시지 디코딩 ($PSS\_Decode\,Verify$)



디코딩 및 서명 검증 알고리즘(*PSS_DecodeVerify*)		고려사항
설정	– $nLen$: 모듈러스 n의 비트 길이 – $Hash$: 내부 해시함수 (출력 길이 : $hLen$ 비트)	①, ②
입력	– 인코딩된 메시지 EM – 메시지 M	
출력	– 서명검증 결과 $Valid$ or $Invalid$	
1	$mHash = Hash(M)$	
2	$\text{if}(LSB_8(EM) != '10111100')$ $Invalid$ 출력	
3	다음과 같이 EM을 분리 $EM = maskedDB \| H \| X$ $(Len(maskedDB) == nLen - hLen - 8, Len(H) == hLen, Len(X) == 8)$	
4	$\text{if}(MSB_1(maskedDB) != '0')$ $Invalid$ 출력	
5	$dbMask = MGF(H, nLen - hLen - 8)$	
6	$DB = maskedDB \oplus dbMask$	
7	$DB = 0 \| LSB_{nLen-hLen-9}(DB)$	
8	$pos = LPos_1(DB)$	
9	$salt = LSB_{Len(DB)-pos}(DB)$	
10	$PS = MSB_{pos}(DB)$	
11	$\text{if}(LSB_8(PS) != '00000001')$ $Invalid$ 출력	
12	$PS = MSB_{pos-8}(PS)$	
13	$\text{if}(PS != 0^{nLen - Len(salt) - hLen - 16})$ $Invalid$ 출력	
14	$tempM = 0^{64} \| mHash \| salt$	
15	$tempH = Hash(tempM)$	
16	$\text{if}(H == tempH)$ $Valid$ 출력 $else$ $Invalid$ 출력	

① 검증대상 모듈러스 크기 확인

지원 가능한 모듈러스 크기 확인

– $nLen \in \{2048, 3072\}$ 비트

② 검증대상 해시함수 사용 확인

내부 해시함수로 검증대상 해시함수를 사용해야 함
- SHA2-224, SHA2-256만 허용됨

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] "안전한 제로화" 참고

□ RSASSA-PSS 서명 검증 (RSA_PSS_Verify)

RSA 서명 검증(*RSA_PSS_Verify*)		고려사항
입력	- RSA 공개키 (n, e) - 전자서명 S - 메시지 M	①, ②, ③
출력	- 서명검증 결과 $Valid$ or $Invalid$	
1	$s = StoI(S)$	
2	$m = s^e \bmod n$	
3	$EM = ItoS(m, Len(n))$	
4	$result = PSS\_Decode\,Verify(EM, M)$	
5	$result$ 출력	

① 검증대상 모듈러스 크기 확인

지원 가능한 모듈러스 크기 확인
- $Len(n) \in \{2048, 3072\}$ 비트

② 전자서명 S 길이 확인

$Len(S) \leq Len(n)$

③ 메시지 M 길이 확인

메시지 M은 해시함수의 입력 메시지 길이 조건을 충족해야 함
- SHA2-224, SHA2-256 : $Len(M) \leq 2^{64} - 1$ 비트

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] "안전한 제로화" 참고

10장 이산대수 기반 전자서명

이산대수 기반 전자서명

1 범위

❑ 본 문서에서는 ISO/IEC 14888-3에 기술된 이산대수 기반 전자서명 알고리즘(KCDSA)를 구현할 경우의 고려사항을 기술한다.

도메인 변수 길이(비트)	해시함수
$(P,\ Q)$ = (2048, 224)	SHA2-224/256
$(P,\ Q)$ = (2048, 256)	SHA2-256
$(P,\ Q)$ = (3072, 256)	

2 관련표준

- [KS X ISO/IEC 14888-3] 부가형 디지털 서명 - 제2부: 이산대수 기반 메커니즘 (2021)
- [TTAK.KO-12.0001/R4] 부가형 전자 서명 방식 표준 - 제2부: 한국형 인증서 기반 전자서명 알고리즘 (KCDSA) (2016)
- [ISO/IEC 14888-3] Digital signatures with appendix - Part 3: Discrete logarithm based mechanisms (2018)

3 기호

기 호	의 미
$Len(X)$	비트열 X의 비트 길이
$GenPrime(X)$	X 비트 길이의 소수를 생성하는 함수
$TestPrime(X)$	Miller-Rabin 소수 판정법을 이용하여 입력값 X의 소수 여부를 판단하는 함수
$GenRandRange(A, B)$	A보다 크고 B보다 작은 임의의 난수값을 생성하는 함수 (A 초과 ~ B 미만)
$GenRandom(len)$	len비트 길이의 난수 비트열을 생성하는 함수
$Null$	Empty String
$\lfloor n \rfloor$	n보다 작거나 같은 정수 중에서 최대값
$[n]_s$	음이 아닌 정수 $n\,(n < 2^s)$의 s비트 이진수 표현
0^n	n길이의 연속된 0 비트열 (Ex. $0^5 = 00000,\ 0^3 = 000$)
1^n	n길이의 연속된 1 비트열 (Ex. $1^5 = 11111,\ 1^3 = 111$)
$LSB_i(X)$	주어진 비트열 X의 하위(오른쪽) i개 비트열
$Hash(X)$	입력된 임의 길이의 메시지 X에 대해 고정된 길이의 해시값을 출력하는 해시함수
$StoI(X)$	비트열 X를 양의 정수값으로 변환하는 함수
$ItoS(N, len)$	양의 정수값 N을 len길이의 비트열로 변환하는 함수

4 이산대수 기반 전자서명

▸ 이산대수 기반 전자서명은 이산대수 문제의 어려움에 그 안전성의 기초를 두고 있으며, KCDSA는 한국형 인증서 기반 부가형 전자서명 알고리즘[1]으로서 임의의 길이를 갖는 메시지에 대한 전자서명을 생성 및 검증할 수 있다.

▸ KCDSA 전자서명 단계는 도메인 변수 및 키 생성, 전자서명 생성, 그리고 전자서명 검증으로 구분된다.

5 알고리즘 명세 및 구현 시 고려사항

가. 도메인 변수 및 키 생성

■ 도메인 변수 P, Q는 다음의 조건을 만족해야 한다.

- P : $2^{\alpha-1} < P < 2^{\alpha}$ 의 크기를 가지는 소수이며, $(P-1)/2Q$ 이 소수이거나 최소한 Q보다 큰 소수들의 곱으로 구성한다. 본 문서는 $J=(P-1)/2Q$ 가 소수인 경우를 다룬다.
- Q : $P-1$ 을 나누는 소수이며, $2^{\beta-1} < Q < 2^{\beta}$ 의 크기를 가진다.

■ 생성원 G는 다음의 조건을 만족해야 한다.

- G : 위수 Q인 부분군의 생성원이다. 즉, $G = g^{(P-1)/Q} \bmod P$ $(1 < g < P-1,\ G > 1)$을 만족하는 정수를 선택한다.

■ 개인키(서명키) X, 공개키(검증키) Y는 다음을 만족해야 한다.

- X : $1 < X < Q$ 을 만족하는 랜덤한 정수 X를 선택한다.
- Y : $Y = G^{X^{-1}} \bmod P$ 로 계산되는 유일한 정수를 선택한다.

□ 일방향 함수 PPGF를 이용한 도메인 변수 생성 방법

일방향 함수(PPGF)		고려사항
설정	– $Hash$: 내부 해시함수 (출력 길이 : LH 비트)	①
입력	– 비트열 $Seed$ – 출력 길이(비트) n	
출력	– 길이 n인 비트열 U	
1	$k = \lfloor (n-1)/LH \rfloor$	
2	$r = n \bmod LH$	
3	$U = Null$	
4	for i from 0 to $(k-1)$ do	
5	$U = Hash(Seed \| [i]_8) \| U$	
6	end for	
7	$U = LSB_r(Hash(Seed \| [k]_8)) \| U$	

1) 부가형 전자서명 알고리즘(Digital signature mechanism with appendix)은 전자문서의 해시 코드에 공개키 연산을 적용하여 전자서명을 생성한다. 검증자는 수신된 메시지와 전자서명을 대상으로 전자서명 생성에 사용된 것과 동일한 해시함수와 서명자의 서명키에 대응하는 검증키를 사용하여 전자서명의 유효성을 확인한다.

도메인 변수 생성 알고리즘(*KCDSA_GenParam*)		고려사항
입력	– 소수 P의 비트 길이 α – 소수 Q의 비트 길이 β	②
출력	– 도메인 변수 P, Q, G – 증거값 J, $Seed$, $Count$, h	
1	$Seed = GenRandom(\beta)$	③
2	$U = PPGF(Seed, \alpha - \beta - 4)$	
3	$tmpJ = (1 \| 0^3 \| U) \vee (0^{\alpha-\beta-1} \| 1)$	
4	$J = StoI(tmpJ)$	
5	if($TestPrime(J) == False$) 단계 1로 이동	④
6	$Count = 0$	
7	$Count = Count + 1$	
8	if($Count > 2^{24}$) 단계 1로 이동	
9	$U = PPGF(Seed \| [Count]_{32}, \beta)$	
10	$tmpQ = U \vee (1 \| 0^{\beta-2} \| 1)$	
11	$Q = StoI(tmpQ)$	
12	$P = 2JQ + 1$	
13	if($Len(P) > \alpha$) 단계 7로 이동	
14	if($TestPrime(Q) == False$) 단계 7로 이동	④
15	if($TestPrime(P) == False$) 단계 7로 이동	④
16	$h = GenRandRange(1, P-1)$	③
17	$G = h^{2J} \bmod P$	
18	if($G == 1$) 단계 16으로 이동	
19	도메인 변수 P, Q, G와 증거값 J, $Seed$, $Count$, h 출력	

① 검증대상 해시함수 사용 확인

사용 가능한 해시함수 확인
- $Len(Q) == 224$인 경우, SHA2-224 및 SHA2-256 사용 가능
- $Len(Q) == 256$인 경우, SHA2-256 사용 가능

② 도메인 변수 길이 확인

지원 가능한 도메인 변수 길이 확인
- $(\alpha, \beta) \in \{(2048, 224), (2048, 256), (3072, 256)\}$ 비트

③ 검증대상 난수발생기 사용 확인

난수를 생성하는 경우, 검증대상 난수발생기를 사용해야 함

④ Miller-Rabin 소수 판정법 반복 횟수 확인

- 소수 판정법으로는 Miller-Rabin 소수 판정법만이 허용되며 반복 횟수는 아래와 같음
  · $\alpha == 2048$ 비트인 경우 : 최소 56회 수행 (오류 확률 : 2^{-112})
  · $\alpha == 3072$ 비트인 경우 : 최소 64회 수행 (오류 확률 : 2^{-128})

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] "안전한 제로화" 참고

□ 난수발생기를 이용한 도메인 변수 생성 방법

도메인 변수 생성 알고리즘(*KCDSA_GenParam*)		고려사항
입력	– 소수 P의 비트 길이 α – 소수 Q의 비트 길이 β	①
출력	– 도메인 변수 P, Q, G	
1	$J = GenPrime(\alpha - \beta)$	②, ③
2	$Count = 0$	
3	$Count = Count + 1$	
4	if($Count > 2^{24}$) 단계 1로 이동	
5	$Q = GenPrime(\beta)$	②, ③
6	$P = 2JQ + 1$	
7	if($Len(P) > \alpha$) 단계 3으로 이동	
8	if($TestPrime(P) == False$) 단계 3으로 이동	③
9	$h = GenRandRange(1, P-1)$	②
10	$G = h^{2J} \bmod P$	
11	if($G == 1$) 단계 9로 이동	
12	도메인 변수 P, Q, G 출력	

① 도메인 변수 길이 확인

지원 가능한 도메인 변수 길이 확인
- $(\alpha, \beta) \in \{(2048, 224), (2048, 256), (3072, 256)\}$ 비트

② 검증대상 난수발생기 사용 확인

난수를 생성하는 경우, 검증대상 난수발생기를 사용해야 함

③ 소수 생성방법 및 Miller-Rabin 소수 판정법 반복 횟수 확인

- 소수 생성방법은 암호모듈 구현안내서(GVI Part 1) 9.1 참조
- 소수 판정법으로는 Miller-Rabin 소수 판정법만이 허용되며 반복 횟수는 아래와 같음
  - $\alpha == 2048$ 비트인 경우 : 최소 56회 수행 (오류 확률 : 2^{-112})
  - $\alpha == 3072$ 비트인 경우 : 최소 64회 수행 (오류 확률 : 2^{-128})

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] “안전한 제로화” 참고

□ 일방향 함수 PPGF를 이용한 키 생성 방법 (다수의 서명키 동시 생성 가능)

키 생성 알고리즘(*KCDSA_KeyGen*)		고려사항
입력	– 도메인 변수 P, Q, G – 생성할 키 개수 m – (Optional) 사용자 입력값 $userInput$	①
출력	– m개의 (서명키, 검증키) 쌍 (X_1, Y_1), (X_2, Y_2), … , (X_m, Y_m)	
1	$b = GenRandRange(Len(Q) - 1, As\ Large\ As\ Possible)$	②, ③
2	$tmp = GenRandom(b)$	③
3	$XKEY = StoI(tmp)$	
4	for i from 1 to m do	
5	$XSEED_i = StoI(PPGF(userInput, b))$ $XVAL = (XKEY + XSEED_i) \bmod 2^b$ $X_i = StoI(PPGF(ItoS(XVAL, b), b)) \bmod Q$ $tmp = StoI(PPGF(ItoS((X_i + XSEED_i) \bmod 2^b, b), b))$ $XKEY = (XKEY + tmp) \bmod 2^b$	
6	end for	
7	for i from 1 to m do	
8	$x = X_i^{-1} \bmod Q$ $Y_i = G^x \bmod P$	
9	end for	
10	m개의 키 쌍 (X_1, Y_1), (X_2, Y_2), … , (X_m, Y_m) 출력	

① 도메인 변수 확인

본 구현안내서의 절차에 준하여 또는 검증필 암호모듈을 사용해 생성된 도메인 변수인 경우, 별도 확인 불필요

도메인 변수 생성 방법이 불분명한 경우, 다음의 항목을 만족하는지 확인

- $(Len(P), Len(Q)) \in \{(2048, 224), (2048, 256), (3072, 256)\}$ 비트
- $1 < G < P$
- $Q \mid (P - 1)$ ※ $(P-1)$이 Q로 나누어 떨어짐
- $G^Q \bmod P == 1$

② 임의의 길이값 최대 범위

임의의 정수 $XKEY$를 생성하기 위한 길이 변수 b의 최대값은 시스템 허용 범위 내에서 임의의 값으로 선택 가능

③ 검증대상 난수발생기 사용 확인

난수를 생성하는 경우, 검증대상 난수발생기를 사용해야 함

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] "안전한 제로화" 참고

□ 난수발생기를 이용한 키 생성 방법

키 생성 알고리즘(*KCDSA_KeyGen*)		고려사항
입력	– 도메인 변수 P, Q, G	①
출력	– (서명키, 검증키) 쌍 (X, Y)	
1	$X = GenRandRange(2^{Len(Q)-8}, Q)$	②
2	$x = X^{-1} \bmod Q$	
3	$Y = G^x \bmod P$	
4	키 쌍 (X, Y) 출력	

① 도메인 변수 확인

본 구현안내서의 절차에 준하여 또는 검증필 암호모듈을 사용해 생성된 도메인 변수인 경우, 별도 확인 불필요

도메인 변수 생성 방법이 불분명한 경우, 다음의 항목을 만족하는지 확인
- $(Len(P), Len(Q)) \in \{(2048, 224), (2048, 256), (3072, 256)\}$ 비트
- $1 < G < P$
- $Q \mid (P-1)$ ※ $(P-1)$이 Q로 나누어 떨어짐
- $G^Q \bmod P == 1$

② 검증대상 난수발생기 사용 확인

난수를 생성하는 경우, 검증대상 난수발생기를 사용해야 함

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] "안전한 제로화" 참고

나. KCDSA 서명 생성

KCDSA 서명 생성 알고리즘(*KCDSA_Sign*)		고려사항
설정	– $Hash$: 내부 해시함수 (블록 길이 : $bLen$ 비트)	①
입력	– 도메인 변수 P, Q, G – (서명키, 검증키) 쌍 (X, Y) – 메시지 M	②, ③, ④
출력	– 전자서명값 (R, S)	
1	$k = GenRandRange(2^{Len(Q)-8}, Q)$	⑤
2	$w = G^k \bmod P$	
3	$R = LSB_{Len(Q)}(Hash(ItoS(w, Len(P))))$	
4	$z = Y \bmod 2^{bLen}$	
5	$h = LSB_{Len(Q)}(Hash(ItoS(z, bLen) \| M))$	
6	$e = StoI((R \oplus h)) \bmod Q$	
7	$s = X(k - e) \bmod Q$	
8	if $(s == 0)$ 단계 1로 이동	
9	$S = ItoS(s, Len(Q))$	
10	전자서명값 (R, S) 출력	

① 검증대상 해시함수 사용 확인

사용 가능한 해시함수 확인
- $Len(Q) == 224$인 경우, SHA2-224 및 SHA2-256 사용 가능
- $Len(Q) == 256$인 경우, SHA2-256 사용 가능

② 메시지 M 길이 확인

메시지 M은 해시함수의 입력 메시지 길이 조건을 충족해야 함
- SHA2-224, SHA2-256 : $Len(M) \leq 2^{64} - 513$ 비트

③ 도메인 변수 확인

본 구현안내서의 절차에 준하여 또는 검증필 암호모듈을 사용해 생성된 도메인 변수인 경우, 별도 확인 불필요

도메인 변수 생성 방법이 불분명한 경우, 다음의 항목을 만족하는지 확인
- $(Len(P), Len(Q)) \in \{(2048, 224), (2048, 256), (3072, 256)\}$ 비트
- $1 < G < P$
- $Q \mid (P - 1)$ ※ $(P-1)$이 Q로 나누어 떨어짐
- $G^Q \bmod P == 1$

④ 키 쌍 확인

본 구현안내서의 절차에 준하여 또는 검증필 암호모듈을 사용해 생성된 키 쌍인 경우, 별도 확인 불필요
키 쌍 생성 방법이 불분명한 경우, 다음의 항목을 만족하는지 확인 - $1 < X < Q,\ 1 < Y < P$ - $G^{X^{-1} \bmod Q} \bmod P == Y$

⑤ 검증대상 난수발생기 사용 확인

난수를 생성하는 경우, 검증대상 난수발생기를 사용해야 함

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] “안전한 제로화” 참고

다. KCDSA 서명 검증

KCDSA 서명 검증 알고리즘(*KCDSA_Verify*)		고려사항
설정	– $Hash$: 내부 해시함수 (블록 길이 : $bLen$ 비트)	①
입력	– 도메인 변수 P, Q, G – 검증키 Y – 메시지 M – 전자서명값 (R, S)	②, ③, ④
출력	– 서명검증 결과 $Valid$ or $Invalid$	
1	$\mathrm{if}(Len(R) != Len(Q))$ $Invalid$ 출력	
2	$s = StoI(S)$	
3	$\mathrm{if}(s \geq Q)$ $Invalid$ 출력	
4	$z = Y \bmod 2^{bLen}$	
5	$h = LSB_{Len(Q)}(Hash(ItoS(z, bLen) \| M))$	
6	$e = StoI((R \oplus h)) \bmod Q$	
7	$w = Y^s G^e \bmod P$	
7	$r = LSB_{Len(Q)}(Hash(ItoS(w, Len(P))))$	
8	$\mathrm{if}(r == R)$ $Valid$ 출력 $else$ $Invalid$ 출력	

① 검증대상 해시함수 사용 확인

사용 가능한 해시함수 확인
- $Len(Q) == 224$인 경우, SHA2-224 및 SHA2-256 사용 가능
- $Len(Q) == 256$인 경우, SHA2-256 사용 가능

② 메시지 M 길이 확인

메시지 M은 해시함수의 입력 메시지 길이 조건을 충족해야 함
- SHA2-224, SHA2-256 : $Len(M) \leq 2^{64} - 513$ 비트

③ 도메인 변수 확인

본 구현안내서의 절차에 준하여 또는 검증필 암호모듈을 사용해 생성된 도메인 변수인 경우, 별도 확인 불필요
도메인 변수 생성 방법이 불분명한 경우, 다음의 항목을 만족하는지 확인 - $(Len(P), Len(Q)) \in \{(2048, 224), (2048, 256), (3072, 256)\}$ 비트 - $1 < G < P$ - $Q \mid (P-1)$ ※ $(P-1)$이 Q로 나누어 떨어짐 - $G^Q \bmod P == 1$

④ 검증키 확인

본 구현안내서의 절차에 준하여 또는 검증필 암호모듈을 사용해 생성된 검증키인 경우, 별도 확인 불필요
키 쌍 생성 방법이 불분명한 경우, 최소한 다음의 항목을 만족하는지 확인 - $1 < Y < P$

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] “안전한 제로화” 참고

11장 타원곡선 기반 전자서명

타원곡선 기반 전자서명

1 범위

❑ 본 문서에서는 ISO/IEC 14888-3에 기술된 타원곡선 기반 전자서명 알고리즘(ECDSA) 및 한국형 인증서 기반 타원곡선 전자서명 알고리즘(EC-KCDSA)를 구현할 경우의 고려사항을 기술한다.

적용알고리즘	구분	검증대상 타원곡선		해시함수
ECDSA/ EC-KCDSA	Prime Field	112비트 안전성	P-224	SHA2-224/256
		128비트 안전성	P-256	SHA2-256
	Binary Field	112비트 안전성	B-233 K-233	SHA2-224/256
		128비트 안전성	B-283 K-283	SHA2-256

2 관련표준

- [KS X ISO/IEC 14888-3] 부가형 디지털 서명 - 제3부: 이산대수 기반 메커니즘 (2021)
- [KS X ISO/IEC 15946-1] 타원곡선에 기반한 암호 기술 - 제1부: 일반 (2024)
- [KS X ISO/IEC 15946-5] 타원곡선에 기초한 암호기법 - 제5부: 타원곡선 생성 (2024)
- [TTAK.KO-12.0015/R3] 부가형 전자 서명 방식 표준 - 제3부: 타원 곡선을 이용한 한국형 인증서 기반 전자서명 알고리즘 (EC-KCDSA) (2016)
- [ISO/IEC 14888-3] Digital signatures with appendix Part 3: Descrete logarithm based mechanisms (2018)
- [ISO/IEC 15946-1] Cryptographic techniques based on elliptic curves Part 1: General (2016)
- [ISO/IEC 15946-5] Cryptographic techniques based on elliptic curves Part 5: Elliptic curve generation (2022)
- [FIPS 186-5] Digital Signature Standard (DSS) (2023)

3 기호

기 호	의 미
$GF(q)$	q개의 원소를 갖는 유한체
$\#(GF(q))$	타원곡선 E의 위수로 무한원점 O를 포함한 타원곡선 E의 모든 원소의 개수
O	타원곡선의 무한원점으로 타원곡선 가환군에서의 항등원임
$Len(X)$	비트열 X의 비트 길이

기 호	의 미
$GenRandRange(A, B)$	A보다 크고 B보다 작은 임의의 난수값을 생성하는 함수 (A 초과 ~ B 미만)
$GenRandom(len)$	len비트 길이의 난수 비트열을 생성하는 함수
$\lceil n \rceil$	n보다 크거나 같은 정수 중에서 최소값
$\lfloor n \rfloor$	n보다 작거나 같은 정수 중에서 최대값
$Min(A, B)$	A, B 둘 중 작은 값
$MSB_i(X)$	주어진 비트열 X의 상위(왼쪽) i개 비트열
$LSB_i(X)$	주어진 비트열 X의 하위(오른쪽) i개 비트열
$Hash(X)$	입력된 임의 길이의 메시지 X에 대해 고정된 길이의 해시값을 출력하는 해시함수
$StoI(X)$	비트열 X를 양의 정수값으로 변환하는 함수
$ItoS(N, len)$	양의 정수값 N을 len길이의 비트열로 변환하는 함수

4 타원곡선 기반 전자서명

- 타원곡선 기반 전자서명은 타원곡선 상의 이산대수 문제의 어려움에 그 안전성의 기초를 두고 있으며, ECDSA와 EC-KCDSA가 검증대상 암호알고리즘으로 승인되어 있다.
- ECDSA와 EC-KCDSA는 부가형 전자서명 알고리즘으로서 임의의 길이를 갖는 메시지에 대한 전자서명을 생성 및 검증할 수 있다.
- ECDSA와 EC-KCDSA는 키 생성, 전자서명 생성, 그리고 전자서명 검증으로 구성되며, 권고된 도메인 변수만을 허용한다.

5 알고리즘 명세 및 구현 시 고려사항

가. 권고 도메인 변수 사용

■ 타원곡선 기반 전자서명에서는 권고된 도메인 변수의 사용만을 허용한다.

■ 권고 커브 형태는 다음과 같다.

권고 커브 형태	
Prime 커브 (P-224/256)	$E: y^2 \equiv x^3 - 3x + b$
Binary 커브 (B-233/283)	$E: y^2 + xy \equiv x^3 + x^2 + b$
Koblitz 커브 (K-233/283)	$E_a: y^2 + xy \equiv x^3 + ax^2 + 1,\ a = 0$

■ 도메인 변수는 다음과 같이 구성된다.

도메인 변수 구성요소	내용
q	Prime 커브 : 유한체 $GF(q)$에 대하여 $q = p$ (p는 소수) Binary, Koblitz 커브 : 유한체 $GF(q)$에 대하여 $q = 2^m$ (m은 소수)
$a,\ b$	타원곡선 방정식의 계수
G	기저점(Base point)으로 불리는 타원곡선의 특정 점($G_x,\ G_y$)
n	기저점 G의 위수(order)
h	타원곡선 위수의 여인자(cofactor)로 $h = \#E(GF(q))/n$로 정의

■ 타원곡선 연산 시 연산효율성을 위하여 다양한 좌표계가 사용될 수 있으며, 각 좌표계에 따른 무한원점은 다음과 같이 정의된다.

- Affine 좌표계 : (0, 0), (0, 1)
- Projective 좌표계 : (0, 1, 0)
- Jacobian 좌표계 : (1, 1, 0)
- Lopez-Dahab 좌표계 : (1, 0, 0)

■ 권고 커브별 도메인 변수 참조값

- Curve P-224

(D:Decimal, H:Hexadecimal)

도메인 변수		값
p (D)		26959946667150639794667015087019630673557916260026308143510066298881
b (H)		b4050a85 0c04b3ab f5413256 5044b0b7 d7bfd8ba 270b3943 2355ffb4
G	G_x (H)	b70e0cbd 6bb4bf7f 321390b9 4a03c1d3 56c21122 343280d6 115c1d21
	G_y (H)	bd376388 b5f723fb 4c22dfe6 cd4375a0 5a074764 44d58199 85007e34
n (D)		26959946667150639794667015087019625940457807714424391721682722368061

- Curve P-256 (D:Decimal, H:Hexadecimal)

도메인 변수		값
p (D)		115792089210356248762697446949407573530086143415290314195533631308867097853951
b (H)		5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b
G	G_x (H)	6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0 f4a13945 d898c296
	G_y (H)	4fe342e2 fe1a7f9b 8ee7eb4a 7c0f9e16 2bce3357 6b315ece cbb64068 37bf51f5
n (D)		115792089210356248762697446949407573529996955224135760342422259061068512044369

- Curve B-233 (D:Decimal, H:Hexadecimal)

도메인 변수		값 (Polynomial Basis)
b (H)		066 647ede6c 332c7f8c 0923bb58 213b333b 20e9ce42 81fe115f 7d8f90ad
G	G_x (H)	0fa c9dfcbac 8313bb21 39f1bb75 5fef65bc 391f8b36 f8f8eb73 71fd558b
	G_y (H)	100 6a08a419 03350678 e58528be bf8a0bef f867a7ca 36716f7e 01f81052
n (D)		6901746346790563787434755862277025555839812737345013555379383634485463

- Curve B-283 (D:Decimal, H:Hexadecimal)

도메인 변수		값 (Polynomial Basis)
b (H)		27b680a c8b8596d a5a4af8a 19a0303f ca97fd76 45309fa2 a581485a f6263e31 3b79a2f5
G	G_x (H)	5f93925 8db7dd90 e1934f8c 70b0dfec 2eed25b8 557eac9c 80e2e198 f8cdbecd 86b12053
	G_y (H)	3676854 fe24141c b98fe6d4 b20d02b4 516ff702 350eddb0 826779c8 13f0df45 be8112f4
n (D)		7770675568902916283677847627294075626569625924376904889109196526770044277787378692871

- Curve K-233 (D:Decimal, H:Hexadecimal)

도메인 변수		값 (Polynomial Basis)
a (D)		0
G	G_x (H)	172 32ba853a 7e731af1 29f22ff4 149563a4 19c26bf5 0a4c9d6e efad6126
	G_y (H)	1db 537dece8 19b7f70f 555a67c4 27a8cd9b f18aeb9b 56e0c110 56fae6a3
n (D)		34508731733952818937173779311385127605709409888622521263280870247413343

- Curve K-283 (D:Decimal, H:Hexadecimal)

도메인 변수		값 (Polynomial Basis)
a (D)		0
G	G_x (H)	503213f 78ca4488 3f1a3b81 62f188e5 53cd265f 23c1567a 16876913 b0c2ac24 58492836
	G_y (H)	1ccda38 0f1c9e31 8d90f95d 07e5426f e87e45c0 e8184698 e4596236 4e341161 77dd2259
n (D)		38853377844514581418389238136470378132848117337930613242958749975298158297044226038 73

나. ECDSA

□ 키 생성

키 생성 알고리즘(*ECDSA_GenKey*)		고려사항
입력	– 타원곡선 도메인 변수 $(q,\ a,\ b,\ G,\ n,\ h)$	①
출력	– (서명키, 검증키) 쌍 $(d,\ Q)$	
1	$d = GenRandRange(2^{Len(n)-(Len(n)\bmod 8)-8}, n)$	②
2	$Q = dG$	
3	키 쌍 $(d,\ Q)$ 출력	

① 도메인 변수 확인

권고 도메인 변수 여부 확인
- 타원곡선 도메인 변수$(q,\ a,\ b,\ G,\ n,\ h)$가 P-224/256, B-233/283, K-233/283 도메인 변수 중 하나와 일치해야 함

② 검증대상 난수발생기 사용 확인

난수를 생성하는 경우, 검증대상 난수발생기를 사용해야 함

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] “안전한 제로화” 참고

□ ECDSA 서명 생성

ECDSA 서명 생성 알고리즘(*ECDSA_Sign*)		고려사항
설정	– 타원곡선 도메인 변수 $(q,\ a,\ b,\ G,\ n,\ h)$ – $Hash$: 내부 해시함수 (출력 길이 : $hLen$ 비트)	①, ②
입력	– 서명키 d – 메시지 M	③, ④
출력	– 전자서명값 $(R,\ S)$	
1	$z = StoI(MSB_{Min(hLen, Len(n))}(Hash(M)))$	
2	$k = GenRandRange(2^{Len(n) - (Len(n) \bmod 8) - 8}, n)$	⑤
3	$(x, y) = kG$	
4	$r = x \bmod n$	
5	$\text{if}(r == 0)$ 단계 2로 이동	
6	$s = k^{-1}(z + rd) \bmod n$	
7	$\text{if}(s == 0)$ 단계 2로 이동	
8	$R = ItoS(r, Len(n))$	
9	$S = ItoS(s, Len(n))$	
10	전자서명값 $(R,\ S)$ 출력	

① 타원곡선 도메인 변수 확인

권고 도메인 변수 여부 확인
- 타원곡선 도메인 변수$(q,\ a,\ b,\ G,\ n,\ h)$가 P-224/256, B-233/283, K-233/283 도메인 변수 중 하나와 일치해야 함

② 검증대상 해시함수 사용 확인

사용 가능한 해시함수 확인
- P-256, B-283, K-283인 경우, SHA2-256 사용 가능
- P-224, B-233, K-233인 경우, SHA2-224 및 SHA2-256 사용 가능

③ 서명키 확인

본 구현안내서의 절차에 준하여 또는 검증필 암호모듈을 사용해 생성된 서명키인 경우, 별도 확인 불필요

서명키 생성 방법이 불분명한 경우, 최소한 다음의 항목을 만족하는지 확인
- $1 < d < n$

④ 메시지 M 길이 확인

메시지 M은 해시함수의 입력 메시지 길이 조건을 충족해야 함
- SHA2-224, SHA2-256 : $Len(M) \leq 2^{64} - 1$ 비트

⑤ 검증대상 난수발생기 사용 확인

난수를 생성하는 경우, 검증대상 난수발생기를 사용해야 함

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] “안전한 제로화” 참고

□ ECDSA 서명 검증

ECDSA 서명 검증 알고리즘(*ECDSA_Verify*)		고려사항
설정	– 타원곡선 도메인 변수 $(q,\ a,\ b,\ G,\ n,\ h)$ – $Hash$: 내부 해시함수 (출력 길이 : $hLen$ 비트)	①, ②
입력	– 검증키 Q – 메시지 M – 전자서명값 $(R,\ S)$	③, ④
출력	– 서명검증 결과 $Valid$ or $Invalid$	
1	$r = StoI(R)$	
2	$s = StoI(S)$	
3	$\text{if}(r \geq n)\ \text{or}\ (s \geq n)$ $Invalid$ 출력	
4	$z = StoI(MSB_{Min(hLen,\ Len(n))}(Hash(M)))$	
5	$u_1 = zs^{-1} \bmod n$	
6	$u_2 = rs^{-1} \bmod n$	
7	$(x, y) = u_1 G + u_2 Q$	
8	$\text{if}((x, y) == O)$ $Invalid$ 출력	
9	$\text{if}(r == (x \bmod n))$ $Valid$ 출력 $else$ $Invalid$ 출력	

① 타원곡선 도메인 변수 확인

권고 도메인 변수 여부 확인
- 타원곡선 도메인 변수$(q,\ a,\ b,\ G,\ n,\ h)$가 P-224/256, B-233/283, K-233/283 도메인 변수 중 하나와 일치해야 함

② 검증대상 해시함수 사용 확인

사용 가능한 해시함수 확인
- P-256, B-283, K-283인 경우, SHA2-256 사용 가능
- P-224, B-233, K-233인 경우, SHA2-224 및 SHA2-256 사용 가능

③ 검증키 확인

본 구현안내서의 절차에 준하여 또는 검증필 암호모듈을 사용해 생성된 검증키인 경우, 별도 확인 불필요
검증키 생성 방법이 불분명한 경우, 최소한 다음의 항목을 만족하는지 확인 - Q가 무한원점 O가 아닌지 확인 - Q의 좌표인 Q_x와 Q_y가 0이 아닌 $GF(q)$의 원소인지 확인 - Q가 타원곡선 위의 점인지 확인 - $nQ == O$ 임을 확인

④ 메시지 M 길이 확인

메시지 M은 해시함수의 입력 메시지 길이 조건을 충족해야 함 - SHA2-224, SHA2-256 : $Len(M) \leq 2^{64} - 1$ 비트

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] “안전한 제로화” 참고

나. EC-KCDSA

□ 키 생성

키 생성 알고리즘(*ECKCDSA_GenKey*)		고려사항
입력	– 타원곡선 도메인 변수 $(q,\ a,\ b,\ G,\ n,\ h)$	①
출력	– (서명키, 검증키) 쌍 $(d,\ Q)$	
1	$d = GenRandRange(2^{Len(n) - (Len(n) \bmod 8) - 8}, n)$	②
2	$Q = d^{-1}G$	
3	키 쌍 $(d,\ Q)$ 출력	

① 도메인 변수 확인

권고 도메인 변수 여부 확인
- 타원곡선 도메인 변수$(q,\ a,\ b,\ G,\ n,\ h)$가 P-224/256, B-233/283, K-233/283 도메인 변수 중 하나와 일치해야 함

② 검증대상 난수발생기 사용 확인

난수를 생성하는 경우, 검증대상 난수발생기를 사용해야 함

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] "안전한 제로화" 참고

□ EC-KCDSA 서명 생성

EC-KCDSA 서명 생성 알고리즘(*ECKCDSA_Sign*)		고려사항
설정	– 타원곡선 도메인 변수 (q, a, b, G, n, h) – $Hash$: 내부 해시함수 (블록 길이 : $bLen$ 비트, 출력 길이 : $hLen$ 비트)	①, ②
입력	– (서명키, 검증키) 쌍 (d, Q) – 메시지 M	③, ④
출력	– 전자서명값 (R, S)	
1	$k = GenRandRange(2^{Len(n)-(Len(n) \bmod 8)-8}, n)$	⑤
2	$(x, y) = kG$	
3	$LH = Min(hLen, 8 \times \lceil \log_{256} n \rceil)$	
4	$R = LSB_{LH}(Hash(x))$	
5	$C_Q = MSB_{bLen}(Q_x \| Q_y)$	
6	$v = LSB_{LH}(Hash(C_Q \| M))$	
7	$e = StoI((R \oplus v)) \bmod n$	
8	$t = d(k-e) \bmod n$	
9	if $(t == 0)$ 단계 1로 이동	
10	$S = ItoS(t, Len(n))$	
11	전자서명값 (R, S) 출력	

① 타원곡선 도메인 변수 확인

권고 도메인 변수 여부 확인
- 타원곡선 도메인 변수(q, a, b, G, n, h)가 P-224/256, B-233/283, K-233/283 도메인 변수 중 하나와 일치해야 함

② 검증대상 해시함수 사용 확인

사용 가능한 해시함수 확인
- P-256, B-283, K-283인 경우, SHA2-256 사용 가능
- P-224, B-233, K-233인 경우, SHA2-224 및 SHA2-256 사용 가능

③ 키 쌍 확인

본 구현안내서의 절차에 준하여 또는 검증필 암호모듈을 사용해 생성된 키 쌍인 경우, 별도 확인 불필요
키 쌍 생성 방법이 불분명한 경우, 다음의 항목을 만족하는지 확인 - $1 < d < n$ - $Len(d) == \lfloor \log_2 n \rfloor$ - $d^{-1}G == Q$

④ 메시지 M 길이 확인

메시지 M은 해시함수의 입력 메시지 길이 조건을 충족해야 함 - SHA2-224, SHA2-256 : $Len(M) \leq 2^{64} - 513$ 비트

⑤ 검증대상 난수발생기 사용 확인

난수를 생성하는 경우, 검증대상 난수발생기를 사용해야 함

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] “안전한 제로화” 참고

□ EC-KCDSA 서명 검증

EC-KCDSA 서명 검증 알고리즘(*ECKCDSA_Verify*)		고려사항
설정	– 타원곡선 도메인 변수 (q, a, b, G, n, h) – $Hash$: 내부 해시함수 (블록 길이 : $bLen$ 비트, 출력 길이 : $hLen$ 비트)	①, ②
입력	– 검증키 Q – 메시지 M – 전자서명값 (R, S)	③, ④
출력	– 서명검증 결과 $Valid$ or $Invalid$	
1	$LH = Min(hLen, 8 \times \lceil \log_{256} n \rceil)$	
2	if $(Len(R) != LH)$ $Invalid$ 출력	
3	$s = StoI(S)$	
4	if $(s < 1)$ or $(s \geq n)$ $Invalid$ 출력	
5	$C_Q = MSB_{bLen}(Q_x \| Q_y)$	
6	$v = LSB_{LH}(Hash(C_Q \| M))$	
7	$e = StoI((R \oplus v)) \bmod n$	
8	$(x, y) = sQ + eG$	
9	if $(R == LSB_{LH}(Hash(x))$ $Valid$ 출력 $else$ $Invalid$ 출력	

① 타원곡선 도메인 변수 확인

권고 도메인 변수 여부 확인
- 타원곡선 도메인 변수(q, a, b, G, n, h)가 P-224/256, B-233/283, K-233/283 도메인 변수 중 하나와 일치해야 함

② 검증대상 해시함수 사용 확인

사용 가능한 해시함수 확인
- P-256, B-283, K-283인 경우, SHA2-256 사용 가능
- P-224, B-233, K-233인 경우, SHA2-224 및 SHA2-256 사용 가능

③ 검증키 확인

본 구현안내서의 절차에 준하여 또는 검증필 암호모듈을 사용해 생성된 검증키인 경우, 별도 확인 불필요
검증키 생성 방법이 불분명한 경우, 최소한 다음의 항목을 만족하는지 확인 - Q가 무한원점 O가 아닌지 확인 - Q의 좌표인 Q_x와 Q_y가 0이 아닌 $GF(q)$의 원소인지 확인 - Q가 타원곡선 위의 점인지 확인 - $nQ == O$ 임을 확인

④ 메시지 M 길이 확인

메시지 M은 해시함수의 입력 메시지 길이 조건을 충족해야 함 - SHA2-224, SHA2-256 : $Len(M) \leq 2^{64} - 513$ 비트

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] “안전한 제로화” 참고

GVI Part 2

Guide for Vendor Implementations

12장 이산대수 기반 키 설정

이산대수 기반 키 설정

1 범위

❑ 본 문서에서는 ISO/IEC 11770-3에 기술된 비대칭 기법을 이용한 키 관리 메커니즘의 부록 D.6 이산대수 기반 키 설정(DH)을 구현할 경우의 고려사항을 기술한다.

알고리즘	도메인 변수 길이(비트)
DH	$(P,\ Q) \in \{\ (2048,\ 224),\ (2048,\ 256),\ (3072,\ 256)\ \}$

2 관련표준

- [KS X ISO/IEC 11770-3] 키 관리 - 제3부: 비대칭 기법을 이용한 메커니즘 (2024)
- [TTAK.KO-12.0001/R4] 부가형 전자 서명 방식 표준 - 제2부: 한국형 인증서 기반 전자 서명 알고리즘 (KCDSA) (2016)
- [ISO/IEC 11770-3] Key management - Part 3: Mechanisms using asymmetric techniques (2021)

3 기호

기 호	의 미
$Len(X)$	비트열 X의 비트 길이
$GenPrime(X)$	X 비트 길이의 소수를 생성하는 함수
$TestPrime(X)$	Miller-Rabin 소수 판정법을 이용하여 입력값 X의 소수 여부를 판단하는 함수
$GenRandRange(A,B)$	A보다 크고 B보다 작은 임의의 난수값을 생성하는 함수 (A 초과 ~ B 미만)
$GenRandom(len)$	len비트 길이의 난수 비트열을 생성하는 함수
$Null$	Empty String
$\lfloor n \rfloor$	n보다 작거나 같은 정수 중에서 최대값
$[n]_s$	음이 아닌 정수 $n\,(n < 2^s)$의 s비트 이진수 표현
0^n	n길이의 연속된 0 비트열 (Ex. $0^5 = 00000,\ 0^3 = 000$)
1^n	n길이의 연속된 1 비트열 (Ex. $1^5 = 11111,\ 1^3 = 111$)
$LSB_i(X)$	주어진 비트열 X의 하위(오른쪽) i개 비트열
$Hash(X)$	입력된 임의 길이의 메시지 X에 대해 고정된 길이의 해시값을 출력하는 해시함수
$StoI(X)$	비트열 X를 양의 정수값으로 변환하는 함수
$ItoS(N,len)$	양의 정수값 N을 len길이의 비트열로 변환하는 함수

4 이산대수 기반 키 설정 (Key Agreement)

- 대칭키 암호알고리즘은 공개키 암호알고리즘에 비해 빠른 속도로 암·복호화를 수행하는 장점을 갖고 있다. 하지만 송·수신자가 동일한 키를 사용하여 암·복호화 과정을 수행하여야 하므로 키를 안전하게 공유하는 것이 필요하다.
- 이산대수 기반 키 설정은 공개키 암호화 방식을 이용하여 송·수신자간 비밀키를 공유하는 방식으로 이산대수 문제의 어려움에 따른 안전성에 기반한다.
- 이산대수 기반 키 설정은 도메인 변수 생성, 키 토큰 생성, 그리고 키 합의 과정으로 구성된다.

5 알고리즘 명세 및 구현 시 고려사항

가. 도메인 변수 생성

■ 도메인 변수 P, Q는 다음의 조건을 만족해야 한다.

- P : $2^{\alpha-1} < P < 2^{\alpha}$ 의 크기를 가지는 소수이며, $(P-1)/2Q$ 이 소수이거나 최소한 Q보다 큰 소수들의 곱으로 구성한다. 본 문서는 $J=(P-1)/2Q$ 가 소수인 경우를 다룬다.
- Q : $P-1$ 을 나누는 소수이며, $2^{\beta-1} < Q < 2^{\beta}$ 의 크기를 가진다.

■ 생성원 G는 다음의 조건을 만족해야 한다.

- G : 위수 Q인 부분군의 생성원이다. 즉, $G = g^{(P-1)/Q} \bmod P$ $(1 < g < P-1,\ G > 1)$을 만족하는 정수를 선택한다.

□ 일방향 함수 PPGF를 이용한 도메인 변수 생성 방법

일방향 함수(PPGF)		고려사항
설정	– $Hash$: 내부 해시함수 (출력 길이 : LH 비트)	①
입력	– 비트열 $Seed$ – 출력 길이(비트) n	
출력	– 길이 n인 비트열 U	
1	$k = \lfloor (n-1)/LH \rfloor$	
2	$r = n \bmod LH$	
3	$U = Null$	
4	for i from 0 to $(k-1)$ do	
5	$U = Hash(Seed \| [i]_8) \| U$	
6	end for	
7	$U = LSB_r(Hash(Seed \| [k]_8)) \| U$	

도메인 변수 생성 알고리즘(*DH_GenParam*)		고려사항
입력	– 소수 P의 비트 길이 α – 소수 Q의 비트 길이 β	②
출력	– 도메인 변수 P, Q, G – 증거값 J, $Seed$, $Count$, h	
1	$Seed = GenRandom(\beta)$	③
2	$U = PPGF(Seed, \alpha - \beta - 4)$	
3	$tmpJ = (1 \| 0^3 \| U) \vee (0^{\alpha-\beta-1} \| 1)$	
4	$J = StoI(tmpJ)$	
5	if($TestPrime(J) == False$) 단계 1로 이동	④
6	$Count = 0$	
7	$Count = Count + 1$	
8	if($Count > 2^{24}$) 단계 1로 이동	
9	$U = PPGF(Seed \| [Count]_{32}, \beta)$	
10	$tmpQ = U \vee (1 \| 0^{\beta-2} \| 1)$	
11	$Q = StoI(tmpQ)$	
12	$P = 2JQ + 1$	
13	if($Len(P) > \alpha$) 단계 7로 이동	
14	if($TestPrime(Q) == False$) 단계 7로 이동	④
15	if($TestPrime(P) == False$) 단계 7로 이동	④
16	$h = GenRandRange(1, P-1)$	③
17	$G = h^{2J} \bmod P$	
18	if($G == 1$) 단계 16으로 이동	
19	도메인 변수 P, Q, G와 증거값 J, $Seed$, $Count$, h 출력	

① 검증대상 해시함수 사용 확인

사용 가능한 해시함수 확인
- $Len(Q) == 224$인 경우, SHA2-224 및 SHA2-256 사용 가능
- $Len(Q) == 256$인 경우, SHA2-256 사용 가능

② 도메인 변수 길이 확인

지원 가능한 도메인 변수 길이 확인
- $(\alpha, \beta) \in \{(2048, 224), (2048, 256), (3072, 256)\}$ 비트

③ 검증대상 난수발생기 사용 확인

난수를 생성하는 경우, 검증대상 난수발생기를 사용해야 함

④ Miller-Rabin 소수 판정법 반복 횟수 확인

- 소수 판정법으로는 Miller-Rabin 소수 판정법만이 허용되며 반복 횟수는 아래와 같음
  - $\alpha == 2048$ 비트인 경우 : 최소 56회 수행 (오류 확률 : 2^{-112})
  - $\alpha == 3072$ 비트인 경우 : 최소 64회 수행 (오류 확률 : 2^{-128})

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] "안전한 제로화" 참고

□ 난수발생기를 이용한 도메인 변수 생성 방법

도메인 변수 생성 알고리즘(*KCDSA_GenParam*)		고려사항
입력	- 소수 P의 비트 길이 α - 소수 Q의 비트 길이 β	①
출력	- 도메인 변수 P, Q, G	
1	$J = GenPrime(\alpha - \beta)$	②, ③
2	$Count = 0$	
3	$Count = Count + 1$	
4	if($Count > 2^{24}$) 단계 1로 이동	
5	$Q = GenPrime(\beta)$	②, ③
6	$P = 2JQ + 1$	
7	if($Len(P) > \alpha$) 단계 3으로 이동	
8	if($TestPrime(P) == False$) 단계 3으로 이동	③
9	$h = GenRandRange(1, P-1)$	②
10	$G = h^{2J} \bmod P$	
11	if($G == 1$) 단계 9로 이동	
12	도메인 변수 P, Q, G 출력	

① 도메인 변수 길이 확인

지원 가능한 도메인 변수 길이 확인
- $(\alpha, \beta) \in \{(2048, 224), (2048, 256), (3072, 256)\}$ 비트

② 검증대상 난수발생기 사용 확인

난수를 생성하는 경우, 검증대상 난수발생기를 사용해야 함

③ 소수 생성방법 및 Miller-Rabin 소수 판정법 반복 횟수 확인

- 소수 생성방법은 암호모듈 구현안내서(GVI Part 1) 9.1 참조
- 소수 판정법으로는 Miller-Rabin 소수 판정법만이 허용되며 반복 횟수는 아래와 같음
  - $\alpha == 2048$ 비트인 경우 : 최소 56회 수행 (오류 확률 : 2^{-112})
  - $\alpha == 3072$ 비트인 경우 : 최소 64회 수행 (오류 확률 : 2^{-128})

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] “안전한 제로화” 참고

나. 키 토큰 생성

■ 구성원 A는 랜덤한 비밀 값 r_A ($r_A \in \{2, \cdots, Q-2\}$)를 생성하고 키 토큰 $KT_A (= G^{r_A} \bmod P)$를 계산하여 구성원 B에게 전송한다.

■ 구성원 B는 랜덤한 비밀 값 r_B ($r_B \in \{2, \cdots, Q-2\}$)를 생성하고 키 토큰 $KT_B (= G^{r_B} \bmod P)$를 계산하여 구성원 A에게 전송한다.

키 토큰 생성 알고리즘(*DH_GenToken*)		고려사항
입력	– 도메인 변수 P, Q, G	①
출력	– 개인키(비밀값) rd – 공개키(키 토큰) KT	
1	$rd = GenRandRange(1, Q-1)$	②
2	$KT = G^{rd} \bmod P$	
3	개인키 rd, 키 토큰 KT 출력	

① 도메인 변수 확인

본 구현안내서의 절차에 준하여 또는 검증필 암호모듈을 사용해 생성된 도메인 변수인 경우, 별도 확인 불필요
도메인 변수 생성 방법이 불분명한 경우, 다음의 항목을 만족하는지 확인 – $(Len(P), Len(Q)) \in \{(2048, 224), (2048, 256), (3072, 256)\}$ 비트 – $1 < G < P$ – $Q \mid (P-1)$ ※ $(P-1)$이 Q로 나누어 떨어짐 – $G^Q \bmod P == 1$

② 검증대상 난수발생기 사용 확인

난수를 생성하는 경우, 검증대상 난수발생기를 사용해야 함

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행 ※ [부록] "안전한 제로화" 참고

다. 키 합의

■ 구성원 A는 다음과 같이 공유키를 계산한다.

$$K_{AB} = KT_B{}^{r_A} = (G^{r_B})^{r_A} = G^{r_A r_B} \bmod P$$

■ 구성원 B는 다음과 같이 공유키를 계산한다.

$$K_{AB} = KT_A{}^{r_B} = (G^{r_A})^{r_B} = G^{r_A r_B} \bmod P$$

키 합의 알고리즘(*DH_ComputeSharedSecret*)		고려사항
입력	– 도메인 변수 P, Q, G – 개인키(비밀값) rd – 상대 구성원의 키 토큰 KT	①
출력	– 공유키 SS	
1	if $(KT \geq P)$ or $(KT \leq 1)$ Error 출력	②
2	if $(KT^Q \bmod P != 1)$ Error 출력	②
3	$s = KT^{rd} \bmod P$	
4	$SS = ItoS(s, Len(P))$	
5	공유키 SS 출력	

① 도메인 변수 확인

본 구현안내서의 절차에 준하여 또는 검증필 암호모듈을 사용해 생성된 도메인 변수인 경우, 별도 확인 불필요

도메인 변수 생성 방법이 불분명한 경우, 다음의 항목을 만족하는지 확인

- $(Len(P), Len(Q)) \in \{(2048, 224), (2048, 256), (3072, 256)\}$ 비트
- $1 < G < P$
- $Q \mid (P-1)$ ※ $(P-1)$이 Q로 나누어 떨어짐
- $G^Q \bmod P == 1$

② 키 토큰 유효성 확인

전달받은 키 토큰 KT가 유효한 형태인지 확인

※ 구성원 간 인증이 선행된 경우 유효성 확인은 불필요함

※ 상대 구성원을 확인할 수 없는 경우, 키 토큰에 대한 유효성 검사 수행

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] “안전한 제로화” 참고

13장 타원곡선 기반 키 설정

타원곡선 기반 키 설정

1 범위

❑ 본 문서에서는 ISO/IEC 11770-3에 기술된 비대칭 기법을 이용한 키 관리 메커니즘의 부록 E.7 타원곡선 기반 키 설정(ECDH)을 구현할 경우의 고려사항을 기술한다.

구분	검증대상 타원곡선	
	112비트 안전성	128비트 안전성
Prime Field	P-224	P-256
Binary Field	B-233	B-283
	K-233	K-283

❑ 타원곡선 기반 키 설정을 구현 시, 검증대상 키 설정에서는 다음의 여인자 곱셈 방법을 사용한다.

	Cofactor Multiplication($j=\#E/n$)	Compatibility(l)	
Option 1	○	×	$(l=1)$

2 관련표준

- [KS X ISO/IEC 11770-3] 키 관리 – 제3부: 비대칭 기법을 이용한 메커니즘 (2024)
- [KS X ISO/IEC 15946-1] 타원곡선에 기반한 암호 기술 – 제1부: 일반 (2024)
- [KS X ISO/IEC 15946-5] 타원곡선에 기초한 암호기법 – 제5부: 타원곡선 생성 (2024)
- [ISO/IEC 11770-3] Key management Part 3: Mechanisms using asymmetric techniques (2021)
- [ISO/IEC 15946-1] Cryptographic techniques based on elliptic curves Part 1: General (2016)
- [ISO/IEC 15946-5] Cryptographic techniques based on elliptic curves Part 5: Elliptic curve generation (2022)
- [FIPS 186-5] Digital Signature Standard (DSS) (2023)

3 기호

기 호	의 미
$GF(q)$	q개의 원소를 갖는 유한체
$\#(GF(q))$	타원곡선 E의 위수로 무한원점 O를 포함한 타원곡선 E의 모든 원소의 개수
O	타원곡선의 무한원점으로 타원곡선 가환군에서의 항등원임
$Len(X)$	비트열 X의 비트 길이
$GenRandRange(A, B)$	A보다 크고 B보다 작은 임의의 난수값을 생성하는 함수 (A 초과 ~ B 미만)
$GenRandom(len)$	len비트 길이의 난수 비트열을 생성하는 함수
$MSB_i(X)$	주어진 비트열 X의 상위(왼쪽) i개 비트열
$LSB_i(X)$	주어진 비트열 X의 하위(오른쪽) i개 비트열
$StoI(X)$	비트열 X를 양의 정수값으로 변환하는 함수
$ItoS(N, len)$	양의 정수값 N을 len길이의 비트열로 변환하는 함수

4 타원곡선 기반 키 설정

- 대칭키 암호알고리즘은 공개키 암호알고리즘에 비해 빠른 속도로 암·복호화를 수행하는 장점을 갖고 있다. 하지만 송·수신자가 동일한 키를 사용하여 암·복호화 과정을 수행하여야 하므로 키를 안전하게 공유하는 것이 필요하다.
- 타원곡선 기반 키 설정은 공개키 암호화 방식을 이용하여 송·수신자간 비밀키를 공유하는 방식으로 타원곡선 상 이산대수 문제의 어려움에 따른 안전성에 기반한다.
- 타원곡선 기반 키 설정은 키 토큰 생성과 키 합의 과정으로 구성된다.

5 알고리즘 명세 및 구현 시 고려사항

가. 권고 도메인 변수 사용

■ 타원곡선 기반 전자서명에서는 권고된 도메인 변수의 사용만을 허용한다.

■ 권고 커브 형태는 다음과 같다.

권고 커브 형태	
Prime 커브 (P-224/256)	$E: y^2 \equiv x^3 - 3x + b$ (cofactor $h=1$, compatibility $l=1$)
Binary 커브 (B-233/283)	$E: y^2 + xy \equiv x^3 + x^2 + b$ (cofactor $h=2$, compatibility $l=1$)
Koblitz 커브 (K-233/283)	$E_a: y^2 + xy \equiv x^3 + ax^2 + 1,\ a=0$ (cofactor $h=4$, compatibility $l=1$)

■ 도메인 변수는 다음과 같이 구성된다.

도메인 변수 구성요소	내용
q	Prime 커브 : 유한체 $GF(q)$에 대하여 $q=p$ (p는 소수) Binary, Koblitz 커브 : 유한체 $GF(q)$에 대하여 $q=2^m$ (m은 소수)
$a,\ b$	타원곡선 방정식의 계수
G	기저점(Base point)으로 불리는 타원곡선의 특정 점($G_x,\ G_y$)
n	기저점 G의 위수(order)
h	타원곡선 위수의 여인자(cofactor)로 $h = \#E(GF(q))/n$로 정의

■ 타원곡선 연산 시 연산효율성을 위하여 다양한 좌표계가 사용될 수 있으며, 각 좌표계에 따른 무한원점은 다음과 같이 정의된다.

- Affine 좌표계 : (0, 0), (0, 1)
- Projective 좌표계 : (0, 1, 0)
- Jacobian 좌표계 : (1, 1, 0)
- Lopez-Dahab 좌표계 : (1, 0, 0)

■ 권고 커브별 도메인 변수 참조값

- Curve P-224

(D:Decimal, H:Hexadecimal)

도메인 변수		값
p (D)		26959946667150639794667015087019630673557916260026308143510066298881
b (H)		b4050a85 0c04b3ab f5413256 5044b0b7 d7bfd8ba 270b3943 2355ffb4
G	G_x (H)	b70e0cbd 6bb4bf7f 321390b9 4a03c1d3 56c21122 343280d6 115c1d21
	G_y (H)	bd376388 b5f723fb 4c22dfe6 cd4375a0 5a074764 44d58199 85007e34
n (D)		26959946667150639794667015087019625940457807714424391721682722368061

- Curve P-256 (D:Decimal, H:Hexadecimal)

도메인 변수		값
p (D)		115792089210356248762697446949407573530086143415290314195533631308867097853951
b (H)		5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b
G	G_x (H)	6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0 f4a13945 d898c296
	G_y (H)	4fe342e2 fe1a7f9b 8ee7eb4a 7c0f9e16 2bce3357 6b315ece cbb64068 37bf51f5
n (D)		115792089210356248762697446949407573529996955224135760342422259061068512044369

- Curve B-233 (D:Decimal, H:Hexadecimal)

도메인 변수		값 (Polynomial Basis)
b (H)		066 647ede6c 332c7f8c 0923bb58 213b333b 20e9ce42 81fe115f 7d8f90ad
G	G_x (H)	0fa c9dfcbac 8313bb21 39f1bb75 5fef65bc 391f8b36 f8f8eb73 71fd558b
	G_y (H)	100 6a08a419 03350678 e58528be bf8a0bef f867a7ca 36716f7e 01f81052
n (D)		6901746346790563787434755862277025555839812737345013555379383634485463

- Curve B-283 (D:Decimal, H:Hexadecimal)

도메인 변수		값 (Polynomial Basis)
b (H)		27b680a c8b8596d a5a4af8a 19a0303f ca97fd76 45309fa2 a581485a f6263e31 3b79a2f5
G	G_x (H)	5f93925 8db7dd90 e1934f8c 70b0dfec 2eed25b8 557eac9c 80e2e198 f8cdbecd 86b12053
	G_y (H)	3676854 fe24141c b98fe6d4 b20d02b4 516ff702 350eddb0 826779c8 13f0df45 be8112f4
n (D)		7770675568902916283677847627294075626569625924376904889109196526770044277787378692871

- Curve K-233 (D:Decimal, H:Hexadecimal)

도메인 변수		값 (Polynomial Basis)
a (D)		0
G	G_x (H)	172 32ba853a 7e731af1 29f22ff4 149563a4 19c26bf5 0a4c9d6e efad6126
	G_y (H)	1db 537dece8 19b7f70f 555a67c4 27a8cd9b f18aeb9b 56e0c110 56fae6a3
n (D)		34508731733952818937173779311385127605709409888622521263280870247413343

- Curve K-283 (D:Decimal, H:Hexadecimal)

도메인 변수		값 (Polynomial Basis)
a (D)		0
G	G_x (H)	503213f 78ca4488 3f1a3b81 62f188e5 53cd265f 23c1567a 16876913 b0c2ac24 58492836
	G_y (H)	1ccda38 0f1c9e31 8d90f95d 07e5426f e87e45c0 e8184698 e4596236 4e341161 77dd2259
n (D)		38853377844514581418389238136470378132848117337930613242958749975298158297044226038 73

나. 키 토큰 생성

■ 구성원 A는 랜덤한 비밀 값 r_A ($r_A \in \{2, \cdots, n-2\}$)를 생성하고 키 토큰 $KT_A (= r_A G)$를 계산하여 구성원 B에게 전송한다.

■ 구성원 B는 랜덤한 비밀 값 r_B ($r_B \in \{2, \cdots, n-2\}$)를 생성하고 키 토큰 $KT_B (= r_B G)$를 계산하여 구성원 A에게 전송한다.

키 토큰 생성 알고리즘(*ECDH_GenToken*)		고려사항
입력	– 타원곡선 도메인 변수 (q, a, b, G, n, h)	①
출력	– 개인키(비밀값) rd – 공개키(키 토큰) KT	
1	$rd = GenRandRange(1, n-1)$	②
2	$KT = rdG$	
3	개인키 rd, 키 토큰 KT 출력	

① 도메인 변수 확인

권고 도메인 변수 여부 확인
- 타원곡선 도메인 변수(q, a, b, G, n, h)가 P-224/256, B-233/283, K-233/283 도메인 변수 중 하나와 일치해야 함

② 검증대상 난수발생기 사용 확인

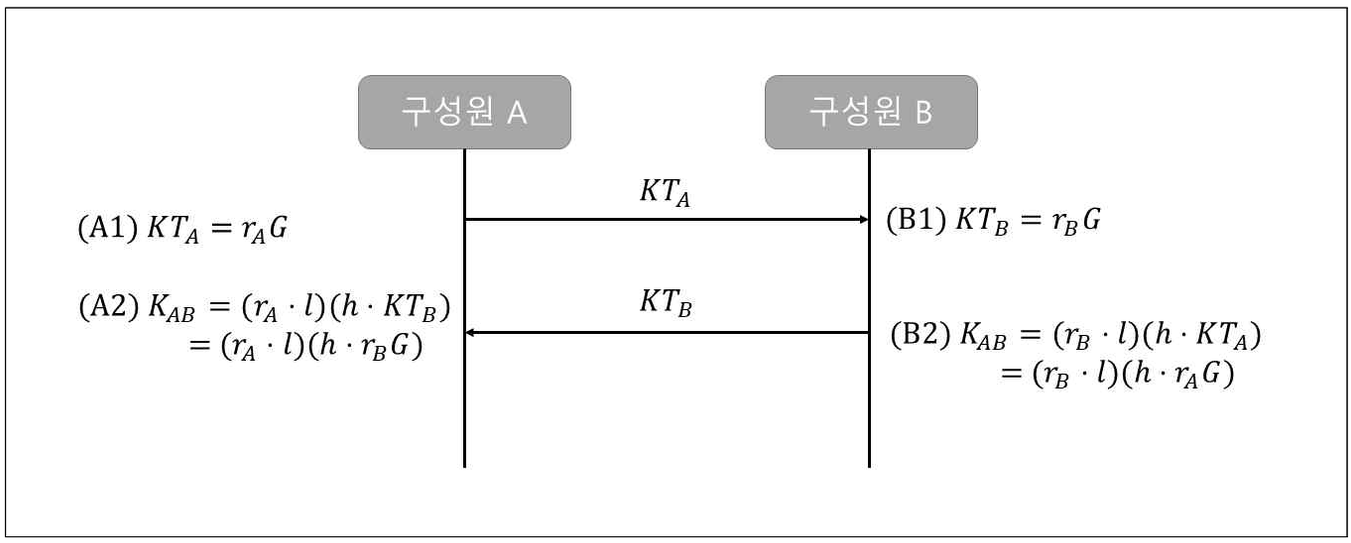
난수를 생성하는 경우, 검증대상 난수발생기를 사용해야 함

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] “안전한 제로화” 참고

다. 키 합의



■ 구성원 A는 다음과 같이 공유키를 계산한다.

$$K_{AB} = (r_A \bullet l)(h\,KT_B)$$

■ 구성원 B는 다음과 같이 공유키를 계산한다.

$$K_{AB} = (r_B \bullet l)(h\,KT_A)$$

키 합의 알고리즘(*ECDH_ComputeSharedSecret*)		고려사항
입력	– 타원곡선 도메인 변수 $(q,\ a,\ b,\ G,\ n,\ h)$ – 개인키(비밀값) rd – 상대 구성원의 키 토큰 KT	①
출력	– 공유키 SS	
1	if$(KT is\ Invalid)$ Error 출력	②
2	$s = (rd \bullet l)(h\,KT)$	
3	if$(s == O)$ Error 출력	
4	$SS = I2OS(s, Len(n))$	
5	공유키 SS 출력	

① 도메인 변수 확인

권고 도메인 변수 여부 확인
– 타원곡선 도메인 변수$(q,\ a,\ b,\ G,\ n,\ h)$가 P-224/256, B-233/283, K-233/283 도메인 변수 중 하나와 일치해야 함

② 키 토큰 유효성 확인

전달받은 키 토큰 KT가 유효한 형태인지 확인
- KT가 무한원점 O가 아닌지 확인
- KT의 좌표인 KT_x와 KT_y가 0이 아닌 $GF(q)$의 원소인지 확인
- KT가 타원곡선 위의 점인지 확인
- $nKT == O$ 임을 확인

※ 구성원 간 인증이 선행된 경우 유효성 확인은 불필요함

※ 상대 구성원을 확인할 수 없는 경우, 키 토큰에 대한 유효성 검사 수행

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] “안전한 제로화” 참고

14장 패스워드 기반 키 유도

패스워드 기반 키 유도

1 범위

❑ 본 문서에서는 TTAK.KO-12.0334에 명세된 패스워드 기반 키 유도 알고리즘인 PBKDF를 구현할 경우의 고려사항을 기술한다.

구분	알고리즘	
의사 난수 함수 (PRF)	HMAC	SHA2-224/256/384/512
		LSH-224/256/384/512 LSH-512_224/512_256
		SHA3-224/256/384/512

2 관련표준

- [TTAK.KO-12.0334-Part1] 패스워드 기반 키 유도 함수 - 제1부: 일반 (2018)
- [TTAK.KO-12.0334-Part2] 패스워드 기반 키 유도 함수 - 제2부: 해시 함수 SHA-2 (2018)
- [TTAK.KO-12.0334-Part3] 패스워드 기반 키 유도 함수 - 제3부: 해시 함수 LSH (2018)
- [TTAK.KO-12.0334-Part4] 패스워드 기반 키 유도 함수 - 제4부: 해시 함수 SHA-3 (2019)
- [RFC 8018] PKCS #5: Password-Based Cryptography Specification Version 2.1 (2017)

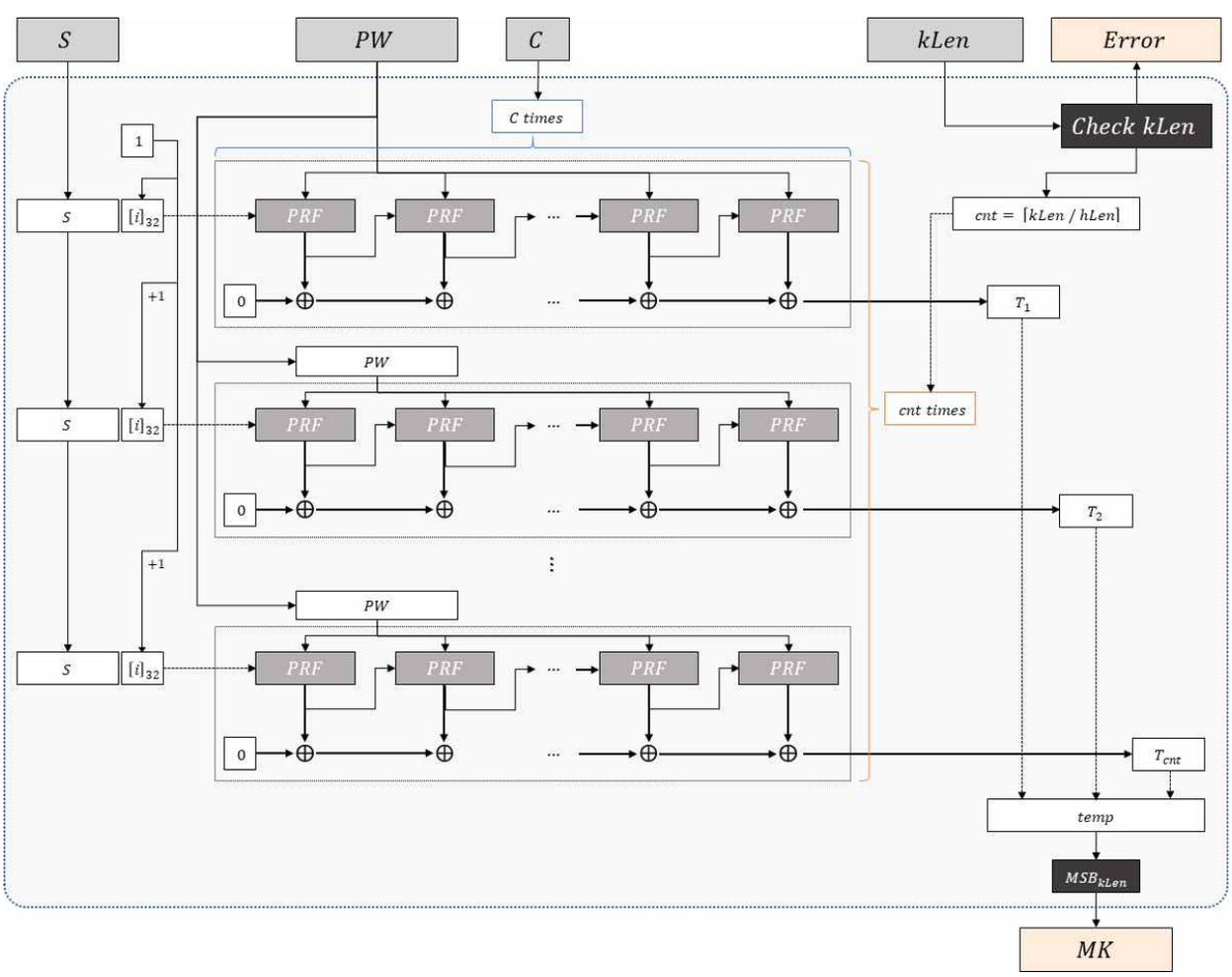
3 기호

기 호	의 미
$PRF(K, D)$	비밀키 K와 메시지 D를 이용하여 의사 난수 비트열을 생성하는 의사 난수 함수
$MSB_i(X)$	주어진 비트열 X의 상위(왼쪽) i개 비트열
$\lceil n \rceil$	n보다 크거나 같은 정수 중에서 최소값
$[n]_s$	음이 아닌 정수 $n (n < 2^s)$의 s비트 이진수 표현

4 패스워드 기반 키 유도

- 패스워드 기반 키 유도 함수(PBKDF)는 패스워드와 부가 데이터를 입력으로 받아 암호 알고리즘 동작에 사용될 수 있는 키 요소를 생성하는 함수이며, 키 요소가 가져야 하는 길이에 따라 의사 난수 함수를 반복 호출한다.
- PBKDF는 기반이 되는 의사 난수 함수와 반복 횟수에 의해 정의된다.
- PBKDF의 입력은 패스워드, 솔트, 유도되는 키의 길이 정보를 포함한다.
- 반복 횟수의 경우 필수로 요구되는 값은 1,000회이지만, 알고리즘 동작환경의 허용 가능한 범위 내에서 최대한 많은 반복 횟수(100,000회 이상)를 사용할 것을 권고한다.

5 알고리즘 명세 및 구현 시 고려사항



패스워드 기반 키 유도(*PBKDF*)		고려사항
설정	– PRF : 내부 의사 난수 함수 (출력 길이 : $hLen$ 비트)	①
입력	– 패스워드/패스프레이즈 PW – 솔트 S – 반복 횟수 C – 출력값의 비트 길이 $kLen$	②, ③, ④
출력	– 유도된 키 비트열 MK	
1	if$(kLen > ((2^{32}-1) \times hLen))$ Error 출력	
2	$cnt = \lceil kLen / hLen \rceil$	
3	$r = kLen - ((cnt-1) \times hLen)$	
4	for i from 1 to cnt do	
5	$T_i = 0$ $U_0 = S \| [i]_{32}$	
6	for j from 1 to C do	
7	$U_j = PRF(PW, U_{j-1})$ $T_i = T_i \oplus U_j$	
8	end for	
9	end for	
10	$MK = T_1 \| T_2 \| ... \| MSB_r(T_{cnt})$	
11	유도된 키 비트열 MK 출력	

① 검증대상 의사 난수 함수 사용 확인

의사 난수 함수로 검증대상 해시함수 기반 메시지 인증코드 사용 확인
- HMAC_SHA2 인 경우 : SHA2-224/256/384/512
- HMAC_LSH 인 경우 : LSH-224/256/384/512, LSH-512_224/512_256
- HMAC_SHA3 인 경우 : SHA3-224/256/384/512

② 패스워드/패스프레이즈 길이 확인

입력된 패스워드/패스프레이즈의 길이는 최소 9자리 이상이어야 함

③ 솔트 길이 및 구성 방법 확인

솔트는 전체 또는 일부가 난수발생기로 생성되어야 하며, 난수발생기로 생성된 부분의 길이는 128 비트 이상이어야 함

※ 솔트는 패드워드 사전공격(Dictionary Attack)을 방지하기 위한 목적으로 사용되는 파라미터임

④ 반복 횟수 확인

반복 횟수로 1,000 이상의 값이 입력되는지 확인

※ 알고리즘 동작환경의 허용 가능한 범위에서 최대한 많은 반복 횟수 사용(100,000회 이상) 권고

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행

※ [부록] “안전한 제로화” 참고

GVI Part 2

Guide for Vendor Implementations

15장 의사난수 함수 기반 키 유도

의사난수 함수 기반 키 유도

1 범위

❑ 본 문서에서는 TTAK.KO-12.0272 및 TTAK.KO-12.0333에 명세된 의사난수 함수 기반 키 유도 알고리즘인 KBKDF를 구현할 경우의 고려사항을 기술한다.

구분	알고리즘	
의사 난수 함수 (PRF)	CMAC	ARIA, SEED, LEA, AES, HIGHT
	HMAC	SHA2-224/256/384/512
		LSH-224/256/384/512 LSH-512_224/512_256
		SHA3-224/256/384/512

2 관련표준

- [TTAK.KO-12.0333-Part1] HMAC 기반 키 유도 함수 – 제1부: 일반 (2018)
- [TTAK.KO-12.0333-Part2] HMAC 기반 키 유도 함수 – 제2부: 해시 함수 SHA-2 (2018)
- [TTAK.KO-12.0333-Part3] HMAC 기반 키 유도 함수 – 제3부: 해시 함수 LSH (2018)
- [TTAK.KO-12.0333-Part4] HMAC 기반 키 유도 함수 – 제4부: 해시 함수 SHA-3 (2019)
- [TTAK.KO-12.0272] 블록 암호 기반 키 유도 함수 (2015)
- [ISO/IEC 11770-6] Key management Part 6: Key derivation (2016)

3 기호

기 호	의 미
$Len(X)$	비트열 X의 비트 길이
$PRF(K, D)$	비밀키 K와 메시지 D를 이용하여 의사 난수 비트열을 생성하는 의사 난수 함수
$Null$	Empty String
0^n	n길이의 연속된 0 비트열 (Ex. $0^5 = 00000$, $0^3 = 000$)
1^n	n길이의 연속된 1 비트열 (Ex. $1^5 = 11111$, $1^3 = 111$)
$\lceil n \rceil$	n보다 크거나 같은 정수 중에서 최소값
$[n]_s$	음이 아닌 정수 $n\,(n < 2^s)$의 s비트 이진수 표현
$MSB_i(X)$	주어진 비트열 X의 상위(왼쪽) i개 비트열
$\{X\}$	데이터 X가 선택적 입력임을 표시

4 의사난수 함수 기반 키 유도

- 의사난수 함수 기반 키 유도 함수(KBKDF)는 키와 부가 데이터를 입력으로 받아 암호알고리즘 동작에 사용될 수 있는 키 요소를 생성하는 함수이며, 키 요소가 가져야 하는 길이에 따라 의사 난수 함수를 반복 호출한다.
- 의사 난수 함수를 반복 호출하는 방법을 반복 모드라고 부르며, 이 반복 모드는 카운터 모드, 피드백 모드, 더블 파이프라인 모드로 구분된다.
- 반복 과정에서, 입력된 키는 키 유도 키로 사용되고, 입력 데이터는 반복 변수와 고정된 입력 데이터의 문자열로 구성된다.
- 반복 모드에 따라, 반복 변수는 카운터나 이전 단계에서의 의사 난수 함수 출력, 또는 둘의 결합이 될 수 있으며, 더블파이프라인 반복 모드의 경우에는 첫 번째 파이프라인으로부터의 출력이 될 수 있다.

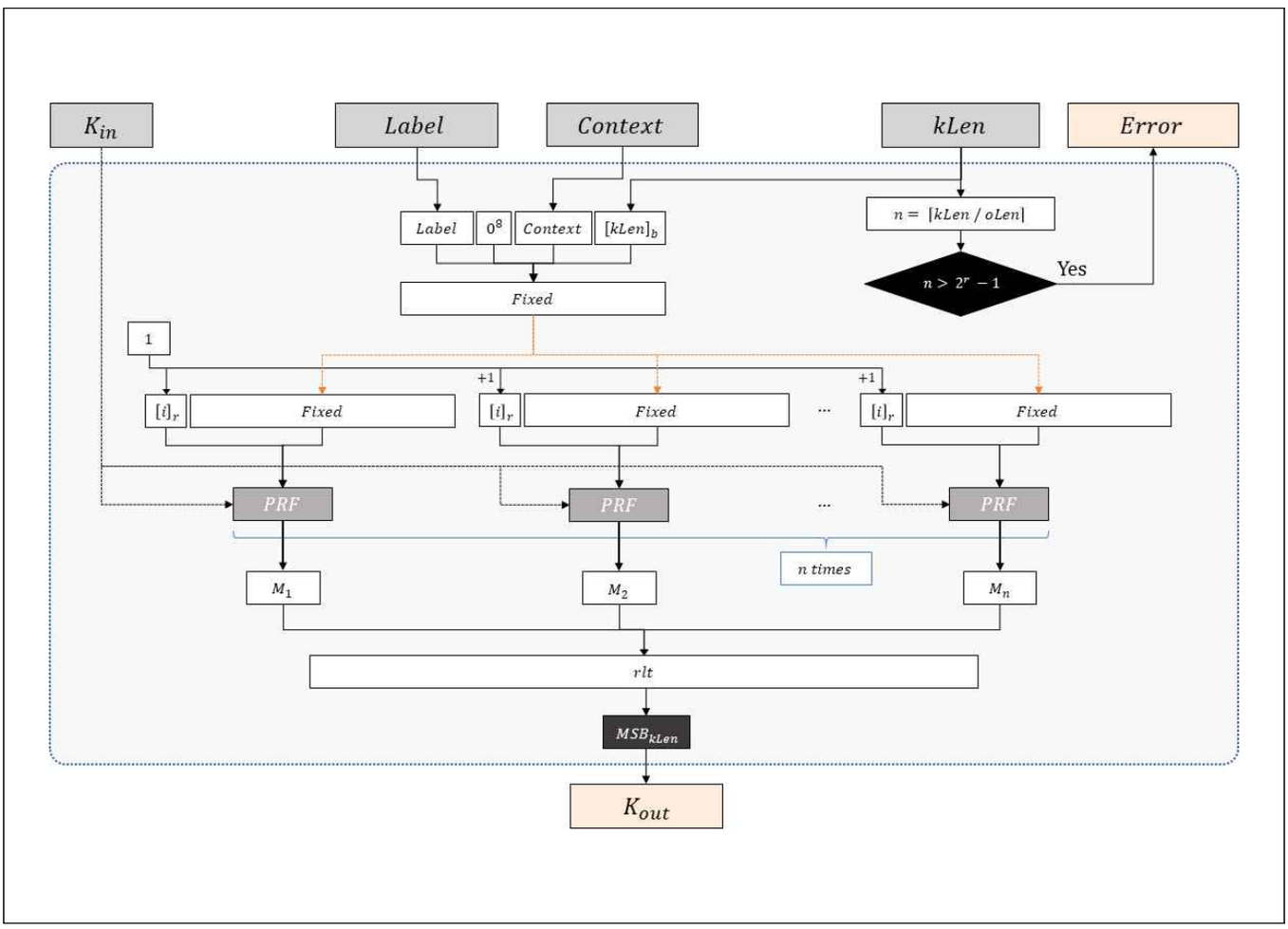
5 알고리즘 명세 및 구현 시 고려사항

가. 카운터(CTR) 모드

▸카운터 모드에서 의사 난수 함수의 출력은 카운터를 반복 변수로 사용하여 계산된다.

▸각 반복 과정에서 고정된 입력 데이터 및 반복 변수는 다음과 같다.

- 반복 변수 : $[i]_r$
- 고정된 입력 데이터 : $Label \| 0^8 \| Context \| [kLen]_r$



카운터 모드 키 유도 함수(*KBKDF_CTR*)		고려사항
설정	– PRF : 내부 의사 난수 함수 (출력 길이 : $oLen$ 비트) – r : 2진수로 표현할 카운터 값의 비트 길이 – b : 2진수로 표현할 정보의 비트 길이	①, ②
입력	– 입력 암호키 K_{in} – (키 목적) 레이블 $Label$ – (키 정보) 컨텍스트 $Context$ – 출력값의 비트 길이 $kLen$	③, ④
출력	– 유도된 키 비트열 K_{out}	
1	$n = \lceil kLen / oLen \rceil$	
2	if $(n > 2^r - 1)$ Error 출력	
3	$rlt = Null$	
4	for i from 1 to n do	
5	$M_i = PRF(K_{in}, [i]_r \| Label \| 0^8 \| Context \| [kLen]_b)$ $rlt = rlt \| M_i$	⑤
6	end for	
7	$K_{out} = MSB_{kLen}(rlt)$	
8	유도된 키 비트열 K_{out} 출력	

① 검증대상 의사 난수 함수 사용 확인

의사 난수 함수로 검증대상 메시지 인증코드 사용 확인
- CMAC-ARIA, CMAC-SEED, CMAC-LEA, CMAC-AES, CMAC-HIGHT
- HMAC-SHA2, HMAC-LSH, HMAC-SHA3

② r 길이 범위 확인

지원 가능한 길이 범위 확인
- $1 \leq r \leq 32$

③ 암호키 길이 확인

의사 난수 함수별 지원 가능한 키 길이 확인
- CMAC-ARIA, CMAC-LEA, CMAC-AES의 암호키 길이 : 128/192/256 비트
- CMAC-SEED, CMAC-HIGHT의 암호키 길이 : 128 비트
- HMAC 기반 메시지 인증코드 암호키 길이 : $oLen \leq Len(K_{in})$

④ $kLen$ 길이 범위 확인

지원 가능한 길이 범위 확인
- $kLen < 2^b$ 비트

⑤ 키 제어 이슈 관련 고려사항

PRF로 CMAC을 사용할 경우 발생할 수 있는 키 제어 관련 보안 이슈를 고려하여, 고정 데이터를 다음과 같이 구성할 수 있음
- $Label \| 0^8 \| Context \| [kLen]_b \| M_0$,

$$M_0 = PRF(K_{in}, Label \| 0^8 \| Context \| [kLen]_b)$$

[공통] 제로화 수행

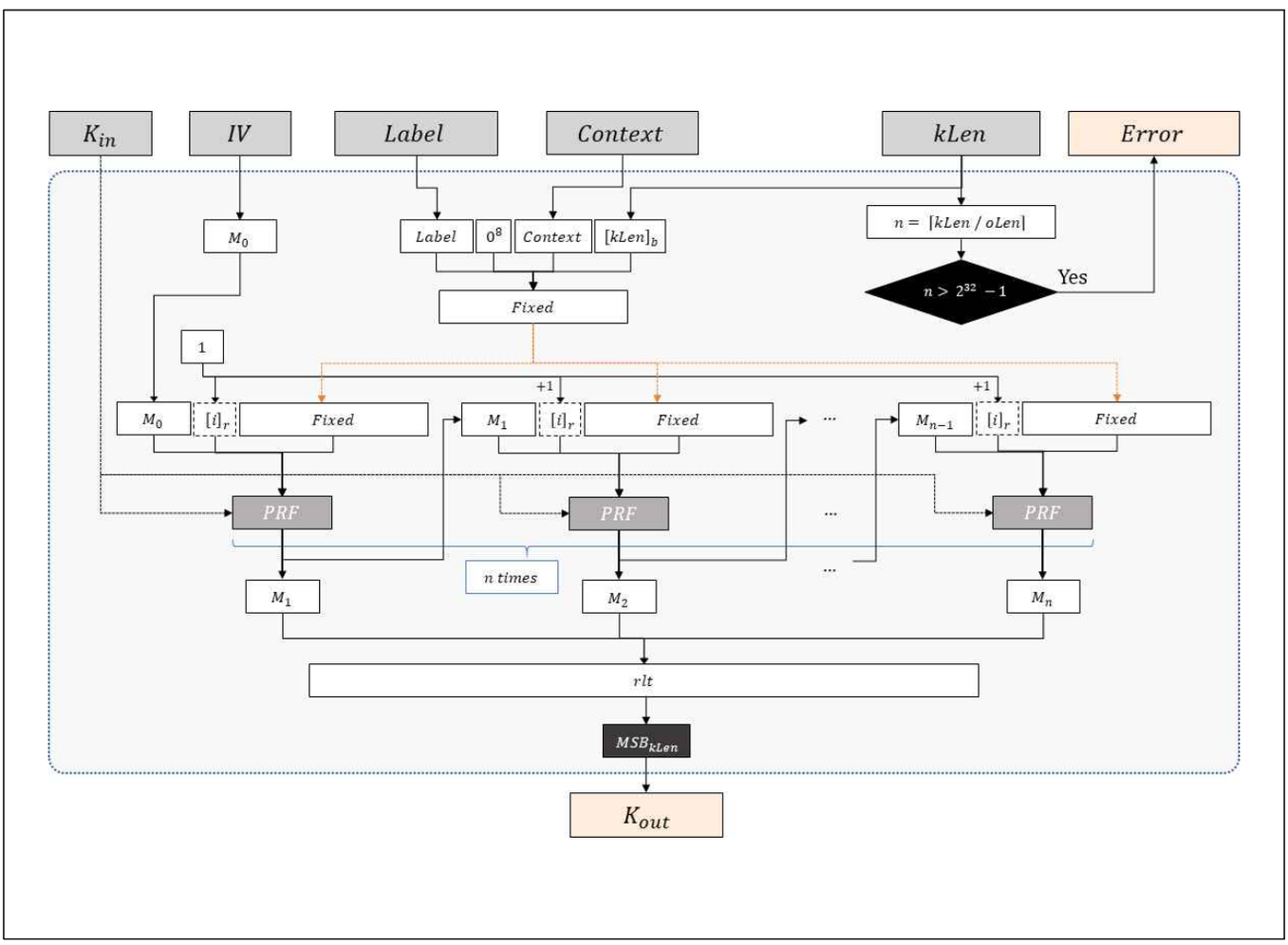
사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] “안전한 제로화” 참고

나. 피드백(FB) 모드

▸ 피드백 모드에서 의사 난수 함수의 출력은 이전 반복의 결과와 (선택적으로) 카운터를 반복 변수로 사용하여 계산된다.

▸ 각 반복 과정에서 고정된 입력 데이터 및 반복 변수는 다음과 같다.

- 반복 변수 : $M_{i-1}\{\,\|\,[i]_r\,\}$
- 고정된 입력 데이터 : $Label \| 0^8 \| Context \| [kLen]_b$



피드백 모드 키 유도 함수(*KBKDF_FB*)		고려사항
설정	– PRF : 내부 의사 난수 함수 (출력 길이 : $oLen$ 비트) – r : 2진수로 표현할 카운터 값의 비트 길이 – b : 2진수로 표현할 정보의 비트 길이 – $ctrFlag$: 카운터 사용 여부	①, ②
입력	– 입력 암호키 K_{in} – 초기값 IV – (키 목적) 레이블 $Label$ – (키 정보) 컨텍스트 $Context$ – 출력값의 비트 길이 $kLen$	③, ④
출력	– 유도된 키 비트열 K_{out}	
1	$n = \lceil kLen / oLen \rceil$	
2	if$(n > 2^{32}-1)$ Error 출력	
3	$rlt = Null$	
4	$M_0 = IV$	⑤
5	for i from 1 to n do	
6	if$(ctrFlag == True)$ $M_i = PRF(K_{in}, M_{i-1} \| [i]_r \| Label \| 0^8 \| Context \| [kLen]_b)$ $else$ $M_i = PRF(K_{in}, M_{i-1} \| Label \| 0^8 \| Context \| [kLen]_b)$	⑤
7	$rlt = rlt \| M_i$	
8	end for	
9	$K_{out} = MSB_{kLen}(rlt)$	
10	유도된 키 비트열 K_{out} 출력	

① 검증대상 의사 난수 함수 사용 확인

의사 난수 함수로 검증대상 메시지 인증코드 사용 확인
- CMAC-ARIA, CMAC-SEED, CMAC-LEA, CMAC-AES, CMAC-HIGHT
- HMAC-SHA2, HMAC-LSH, HMAC-SHA3

② r 길이 범위 확인

지원 가능한 길이 범위 확인
- $1 \leq r \leq 32$

③ 암호키 길이 확인

의사 난수 함수별 지원 가능한 키 길이 확인
- CMAC-ARIA, CMAC-LEA, CMAC-AES의 암호키 길이 : 128/192/256 비트
- CMAC-SEED, CMAC-HIGHT의 암호키 길이 : 128 비트
- HMAC 기반 메시지 인증코드 암호키 길이 : $oLen \leq Len(K_{in})$

④ $kLen$ 길이 범위 확인

지원 가능한 길이 범위 확인
- $kLen < 2^b$ 비트

⑤ 키 제어 이슈 관련 고려사항

PRF로 CMAC을 사용할 경우 발생할 수 있는 키 제어 관련 보안 이슈를 고려하여, IV를 입력받지 않고 다음과 같이 구성할 수 있음
- $IV = PRF(K_{in}, Label \| 0^8 \| Context \| [kLen]_b)$

※ 이 경우, $ctrFlag$를 $True$로 설정하여 카운터 값을 반드시 사용해야 함

[공통] 제로화 수행

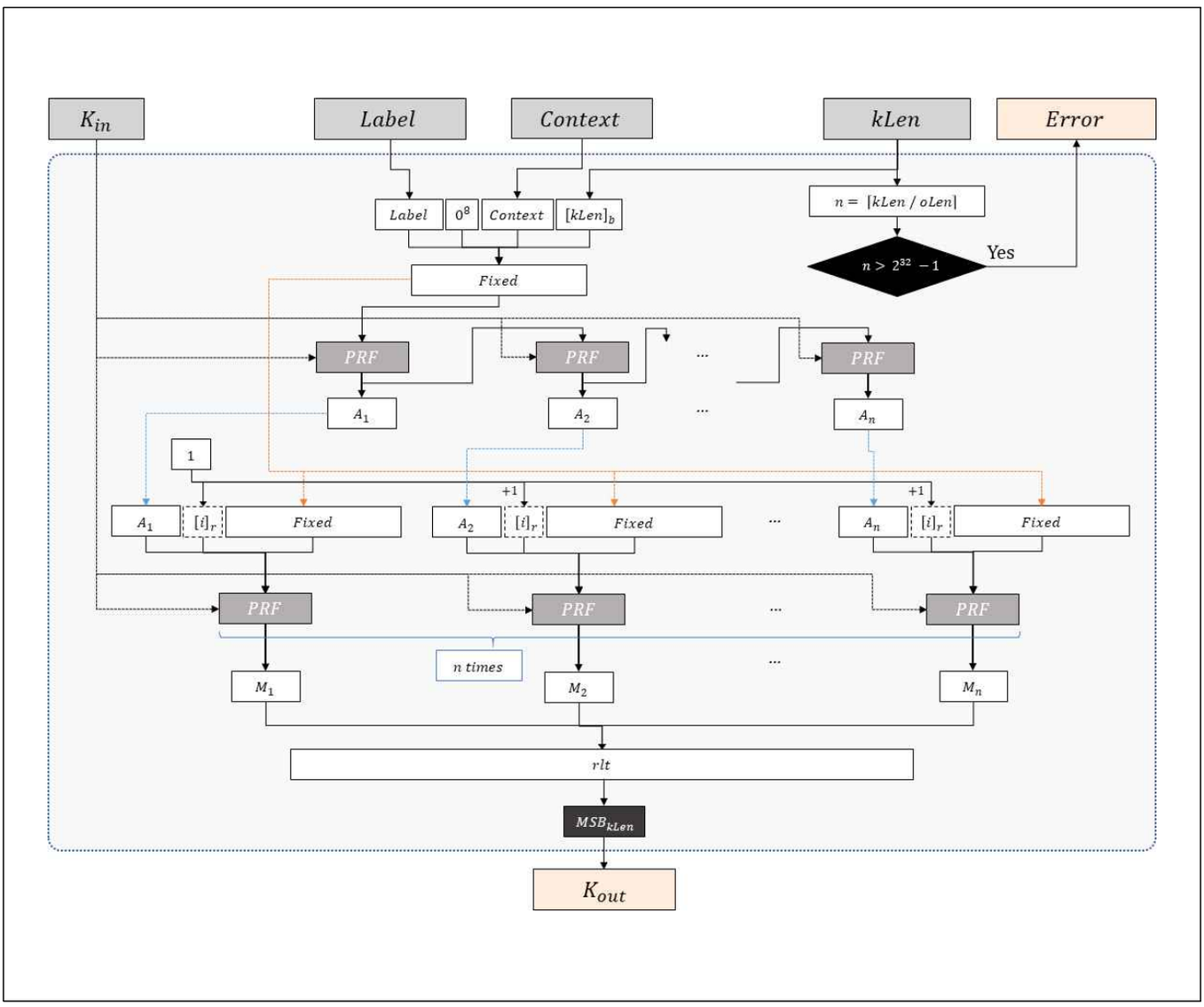
사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] “안전한 제로화” 참고

다. 더블 파이프라인(DP) 모드

▸더블 파이프라인 모드는 의사 난수 함수를 두 개의 파이프라인에서 반복하도록 정의한다. 첫 번째 파이프라인에서 비트열 A가 생성되고, A의 각 블록은 두 번째 파이프라인에서 해당하는 의사 난수 함수의 입력값으로 사용된다.

▸각 반복 과정에서 고정된 입력 데이터와 초기값, 반복 변수는 다음과 같다.

- 초기값 A_0 : $Label \| 0^8 \| Context \| [kLen]_b$
- 반복 변수 : $A_i \{ \| [i]_r \}$
- 고정된 입력 데이터 : $Label \| 0^8 \| Context \| [kLen]_b$



더블 파이프라인 모드 키 유도 함수(*KBKDF_DP*)		고려사항
설정	– PRF : 내부 의사 난수 함수 (출력 길이 : $oLen$ 비트) – r : 2진수로 표현할 카운터 값의 비트 길이 – b : 2진수로 표현할 정보의 비트 길이 – $ctrFlag$: 카운터 사용 여부	①, ②
입력	– 입력 암호키 K_{in} – (키 목적) 레이블 $Label$ – (키 정보) 컨텍스트 $Context$ – 출력값의 비트 길이 $kLen$	③, ④
출력	– 유도된 키 비트열 K_{out}	
1	$n = \lceil kLen / oLen \rceil$	
2	if $(n > 2^{32}-1)$ Error 출력	
3	$rlt = Null$	
4	$A_0 = Label \| 0^8 \| Context \| [kLen]_b$	
5	for i from 1 to n do	
6	$A_i = PRF(K_{in}, A_{i-1})$	
7	if $(ctrFlag == True)$ $M_i = PRF(K_{in}, A_i \| [i]_r \| Label \| 0^8 \| Context \| [kLen]_b)$ $else$ $M_i = PRF(K_{in}, A_i \| Label \| 0^8 \| Context \| [kLen]_b)$	⑤
8	$rlt = rlt \| M_i$	
9	end for	
10	$K_{out} = MSB_{kLen}(rlt)$	
11	유도된 키 비트열 K_{out} 출력	

① 검증대상 의사 난수 함수 사용 확인

의사 난수 함수로 검증대상 메시지 인증코드 사용 확인
- CMAC-ARIA, CMAC-SEED, CMAC-LEA, CMAC-AES, CMAC-HIGHT
- HMAC-SHA2, HMAC-LSH, HMAC-SHA3

② r 길이 범위 확인

지원 가능한 길이 범위 확인
- $1 \leq r \leq 32$

③ 암호키 길이 확인

의사 난수 함수별 지원 가능한 키 길이 확인
- CMAC-ARIA, CMAC-LEA, CMAC-AES의 암호키 길이 : 128/192/256 비트
- CMAC-SEED, CMAC-HIGHT의 암호키 길이 : 128 비트
- HMAC 기반 메시지 인증코드 암호키 길이 : $oLen \leq Len(K_{in})$

④ $kLen$ 길이 범위 확인

지원 가능한 길이 범위 확인
- $kLen < 2^b$ 비트

⑤ 키 제어 이슈 관련 고려사항

PRF로 CMAC을 사용할 경우 발생할 수 있는 키 제어 관련 보안 이슈를 고려하여, 다음과 같이 구성할 수 있음
- $ctrFlag$를 $True$로 설정하여 카운터 값을 반드시 사용함

[공통] 제로화 수행

사용이 끝난 중요보안매개변수(SSP)에 대한 제로화 수행
※ [부록] “안전한 제로화” 참고

부록

부 록

1 안전한 제로화

가. 최적화로 인한 생략방지

□ 컴파일러 최적화 옵션으로 인한 제로화 코드 생략방지

- 제로화 코드는 사용된 메모리의 값을 초기화한 후 메모리 자원을 반납하는 절차이기 때문에 컴파일러 최적화 옵션에 따라 생략되는 경우가 존재함
- 제로화 코드가 컴파일러 옵션에 의해 생략되는 것을 방지하기 위하여 다음의 방법 사용 가능

대응방법	내용
Fake 코드이용	제로화를 수행하는 API에 연산의 결과에 영향을 주지 않고 추가 연산만을 수행하는 코드를 삽입하여 컴파일러 최적화로 인한 코드 생략 방지
Volatile 키워드 사용	volatile은 컴파일러 옵션 최적화의 영향을 받지 않도록 하는 키워드이기 때문에 변수 선언 또는 제로화 API 구현 시 volatile 키워드 적용
SecureZeroMemory	윈도우 환경에서는 SecureZeroMemory API를 제공하여 코드생략 방지 (내부적으로 volatile 키워드를 이용하고 있음)

나. 컨텍스트 제로화

□ 컨텍스트 포인터 제로화 시 다중 제로화 필요

- 암호관련 API 수행 시 필요한 인자를 컨텍스트 형태로 전달하는 경우가 있으며 컨텍스트에는 다음과 같이 포인터 변수들이 포함되기도 함

컨텍스트 정의

```
typedef struct _KCMVP_BIGNUM_{
    int sign;                   // 부호
    unsigned int numOfLimb;     // 큰수 저장에 사용된 워드 개수
    unsigned int* limb;         // 큰수를 저장한 메모리 포인터
}KCMVP_BIGNUM
typedef struct _KCMVP_RSA_PrivateKey_{
    KCMVP_BIGNUM  *n;             // 모듈러스 값
    KCMVP_BIGNUM  *d;             // 개인키 d값
    KCMVP_BIGNUM  *p;             // 개인키 p값
    KCMVP_BIGNUM  *q;             // 개인키 q값
} KCMVP_RSA_PrivateKey;
```

– 다음은 컨텍스트 내부에 포함된 포인터 변수들이 참조하는 메모리 제로화를 누락한 경우임

잘못된 제로화

```
BOOL KCMVP_RSA_PrivateKey_Zeroize(KCMVP_RSA_PrivateKey* privKey){
    if(priv Key != NULL) {
        SecureZeroMemory(privKey, sizeof(KCMVP_RSA_PrivateKey));
        free(privKey);
    }
    ...
}
```

– 다음은 컨텍스트에 포함된 포인터 변수들이 참조하는 메모리들을 확인하여 올바르게 제로화하고 있는 경우임

올바른 제로화

```
BOOL KCMVP_RSA_PrivateKey_Zeroize(KCMVP_RSA_PrivateKey* privKey){
    if(priv Key != NULL){
        if(privKey->n != NULL){
            KCMVP_BIGNUM_zeroize(privKey->n);
            free(privKey->n);
        }
        if(privKey->d != NULL){
            KCMVP_BIGNUM_zeroize(privKey->d);
            free(privKey->d);
        }
        if(privKey->p != NULL){
            KCMVP_BIGNUM_zeroize(privKey->p);
            free(privKey->p);
        }
        if(privKey->q != NULL){
            KCMVP_BIGNUM_zeroize(privKey->q);
            free(privKey->q);
        }
    }
    ...
}

BOOL KCMVP_BIGNUM_zeroize(KCMVP_BIGNUM* bNum){
    ...
    if(bNum→limb != NULL){
        SecureZeroMemory(bNum→limb, WORDS_FOR_2048);
        free(bNum→limb);
    }
    ...
}
```

다. 제로화 종류

□ 절차적 제로화

- 하나의 API 내부에서 사용이 끝난 중요보안매개변수를 제로화
  ※ 평문의 마지막 블록에 대한 암호화를 수행하는 Encrypt_Final()에서 암호화에 사용된 라운드키 제로화
- API 외부에서 API 내부로 복사되어 사용된 데이터에 대한 제로화
- API 내부에서 생성되어 사용된 데이터를 API 반환 전 제로화

절차적 제로화 예시–난수발생기 인스턴스 생성 후

```
int KCMVP_drbg_instantiate(DRBG_CTX *dctx, const unsigned char *pers, size_t perslen){
    // entropy : 엔트로피를 수집하는 내부 변수
    size_t entlen = 0;
    unsigned char *entropy = NULL;
     ...
    // 사용이 끝난 엔트로피를 수집하는 내부 변수를 제로화함
    KCMVP_cleanup_entropy(dctx, entropy, entlen);
    ...
}
```

절차적 제로화 예시–대칭키 암호화 수행 후

```
// 대칭키 암호화 API
int Encrypt_Final(unsinged char* CT, int* sizeofCT, unsigned char* PT, int sizeofPT,
                  unsigned char* MK, int sizeofMK){
    int ret = TRUE;
    unsigned char RoundKeys[numofRounds][sizeofKey] = {{0x00, }, };

    // 라운드키 생성(키스케쥴)
    keySchedule(RoundKeys, MK, sizeofMK);

    // 암호화 수행(내부 암호화 API인 _Encrypt() 사용)
    ret = _Encrypt(RoundKeys, CT, sizeofCT, PT, sizeofPT);

    // 라운드 키에 대한 제로화(절차적 제로화)
    SecureZeroMemory(RoundKeys, sizeof(RoundKeys));

    return ret;
}
```

□ 운영적 제로화

- 명시적 제로화 API를 호출함으로써, 메모리에 로드된 중요보안매개변수를 제로화
- Task 단위의 제로화(예 : 파일 암호화 완료 후, 마스터키 제로화)
- 암호모듈 종료 시, 제로화 API를 이용하여 중요보안매개변수 제로화

운영적 제로화 예시–암호모듈 종료 시 난수발생기 내부상태 제로화

```
// 암호모듈 종료 API
int KCMVP_Module_Exit(KCMVP_Module_State* state){
    int ret = TRUE;

    // 난수발생기 상태 제로화
    if(state->DRBG_State != NULL)
    {
        // 난수발생기 제로화 API를 통한 제로화 수행
        ret = DRBG_zeroization(state->DRBG_State);
    }

    // 추가적으로 암호모듈 내부에 할당된 핵심보안매개변수에 대하여 운영적 제로화 수행

    return ret;
}
```

운영적 제로화 예시-Task 종료 시 제로화

```
// 암호모듈은 동적으로 링크되었다고 가정함
int fileEncrypt(char* originalFileName, char* encryptedFileName)
{
    int ret = TRUE;
    unsigned char MK[sizeofKey] = {0x00, };
    unsigned char PT[blockSize] = {0x00, }, CT[blockSize] = {0x00, };

    FILE* fpInput = NULL, fpOutput = NULL;

    // 암호화할 파일 열기
    fpInput = fopen(originalFileName, "r");
    if(fpInput == NULL) {ret = FALSE; goto err;}

    // 암호화된 결과를 저장할 파일 열기
    fpOutput = fopen(encryptedFileName, "w");
    if(fpOutput == NULL) {ret = FALSE; goto err;}

    // 난수발생기를 이용하여 암호화용 마스터키 생성
    DRBG_generateKey(MK, sizeofKey);

    // 파일을 읽으며 암호화를 수행
    // 암호화된 데이터는 encryptedFileName에 저장됨
    while(!feof(originalFileName))
    {
        ...;
    }

    // 파일 암호화 Task가 완료된 후, 제로화 수행
    symmetricKey_Zeroization(MK, sizeofKey);
err:
    return ret;
}
```

2 안전한 패딩 방법

ECB, CBC 모드는 평문 블록을 암/복호화 알고리즘의 입력으로 사용하기 때문에, 평문 블록의 길이가 블록암호 알고리즘의 1블록 길이(blocksize)의 양의 정수배가 되도록 덧붙이기 방법이 적용되어야 한다.
본 덧붙이기 방법은 ISO/IEC 국제표준 및 PKCS에서 사용되는 방법으로, 적용되는 시스템에 맞는 방법을 선택하여 사용함을 권고한다. 패딩 방법의 사용 예는 16진법 바이트 단위로 표기한다.

가. 패딩 방법

□ 패딩 방법 1

- 평문의 길이가 (blocksize × t + m) 비트 (0 ≤ m < blocksize, 0 ≤ t)일 때, m이 0이 아닐 경우, 평문의 길이가 blocksize 비트의 양의 정수배가 되도록 평문의 끝에 비트 '1'을 추가한 후 (blocksize − m - 1)개의 '0' 비트를 덧붙인다. 또한 m이 0인 경우에는 패딩 방법이 사용됨을 표기하기 위해, 추가적인 blocksize 비트 '10…00' 블록을 추가한다.

```
예)  평문       ( 48 비트) :  4F 52 49 54 48 4D

     적용 결과 (128 비트) :  4F 52 49 54 48 4D 80 00 00 00 00 00 00 00 00 00


     평문       (128 비트) :  53 45 45 44 41 4C 47 A8 3E D1 80 F1 29 DC 4A 78

     적용 결과 (256 비트) :  53 45 45 44 41 4C 47 A8 3E D1 80 F1 29 DC 4A 78
                             80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

□ 패딩 방법 2

- 본 패딩 방법은 바이트 단위로만 적용 가능하다. 평문의 길이가 (blocksize × t + m) 바이트 (0 ≤ m < blocksize, 0 ≤ t)일 때, m이 0이 아닐 경우, 평문의 길이가 blocksize 바이트의 양의 정수배가 되도록 평문의 끝에 패딩이 필요한 바이트 수 (blocksize - m)을 덧붙인다. m이 0인 경우에는 패딩 방법이 사용됨을 표기하기 위해, 추가적인 blocksize 바이트 블록을 추가한다.

```
예)  평문       ( 48 비트) :  4F 52 49 54 48 4D

     적용 결과 (128 비트) :  4F 52 49 54 48 4D 0A 0A 0A 0A 0A 0A 0A 0A 0A 0A


     평문       (128 비트) :  53 45 45 44 41 4C 47 A8 3E D1 AF 07 4A 73 12 2C

     적용 결과 (256 비트) :  53 45 45 44 41 4C 47 A8 3E D1 AF 07 4A 73 12 2C
                             10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
```

나. 안전성 고려사항

– 상기 패딩 방법 이외에도 여러 가지 패딩 방법이 다양한 표준 및 암호 응용 프로토콜 규격에 제시되어 있다. 그러나 CBC 모드를 사용하는 다양한 암호 응용 프로토콜에 대한 패딩 오라클 공격(Padding Oracle Attack) 결과, 현재 알려진 대다수의 패딩 방법이 공격에 활용될 수 있는 것으로 밝혀진 바 있다[2]. 패딩 오라클 공격은 공격자가 CBC 모드로 암호화된 데이터로부터 패딩이 올바르게 적용되었는지 여부를 확인할 수 있다는 가정에 기반하며, 이를 활용하여 암호키 없이 평문 데이터를 알 수 있다. 따라서 데이터 암호화를 위해 CBC 모드를 사용할 경우, 패딩 오라클 공격의 여지를 없애도록 구현하는 것이 필요하다. 또한 암호문에 대한 MAC 계산을 추가하여 공격자의 유효한 암호문 생성을 방지해야 한다.

2) Serge Vaudenay. Security flaws induced by CBC padding – Applications to SSL, IPSEC, WTLS... Advances in Cryptology – EUROCRYPT 2002. Lecture Notes in Computer Science, vol. 2332, pp. 534-546, 2002

GVI Part 2

Guide for Vendor Implementations

암호모듈 구현안내서

Part 2 | 검증대상 암호알고리즘 구현안내서

GVI 2025. 10.

국가정보원 NATIONAL INTELLIGENCE SERVICE

NSR 국가보안기술연구소