MagicCrypto

Version 2.2.0

API 함수 설명서

Version 2.2

• 문서 개정 이력

버전	변경일	변경사유	변경 내용	작성자
1.0	2008.01.21	제정		장형도
1.1	2008.03.19	수정	난수생성 알고리즘 ID 수정	장형도
1.2	2008.04.11	수정	6.1.52 에러코드 수정	장형도
1.3	2008.04.14	추가	6.1.11 함수 추가	장형도
1.4	2008.06.09	수정	키출력 설명 수정	장형도
1.5	2009.03.13	추가	자가시험시 소프트웨어 무결성 시험 추가	장형도
1.6	2009.04.07	수정	1.2 제품소개 변경	장형도
1.7	2010.04.06	추가	3 알고리즘 ID 추가	장형도
2.0	2013.02.28	수정	2.2. 자가테스트	장형도
2.1	2014.06.20	추가	3 알고리즘 ID 추가 6.1.50 키유도 함수 추가	장형도
2.2	2019.10.25	추가	Hash, 대칭키 암복호화, 키유도 알고리즘 추가	장형도

• 목 차

1. 개요

1.1. 저작권

1.2. 소개

본 문서는 MagicCrypto 암호모듈을 사용하여 응용프로그램을 개발하는 개발자가 암/복호화, 전자서명 생성/검증 등과 같은 암호서비스를 원할히 이용할 수 있도록 각 서비스에 대한 예제와 각 함수에 설명을 서술한다. MagicCrypto 암호모듈은 Windows 및 Linux, 유닉스 환경과 안정성에 초점을 두어 개발되었으며, 일정 수준의 암호학적 지식을 가진 개발자를 대상으로 하고 있다.

2. Magic Crypto 기능

2.1. 버전관리

MagicCrypto 암호모듈의 라이브러리 모듈 및 문서 파일에 대한 버전 관리는 ㈜드림시큐리티에 의해 이루어진다.
설치된 암호모듈에 대한 버전은 MC_GetVersion 함수를 통하여 확인할 수 있다. (5.2.1 참조)

2.2. 자가 검증

관리자에 의한 초기 설치 시점과 개발자가 필요로 하는 시점에 MagicCrypto 암호모듈이 올바르게 작동하는지 확인하기 위해 자가 테스트를 수행한다.

자가 테스트(전원인가 시험, 조건부 시험) 시 수행하는 항목은 다음과 같다.

자가시험	시험	비고
동작전 자가시험	암호 알고리즘 시험	검증대상 해시 KAT
		검증대상 HMAC KAT
		난수발생기 KAT 시험
	핵심 기능 시험	잡음원 건전성 시험
	암호모듈 소프트웨어 무결성 검증 시험	
조건부 자가시험	암호알고리즘 시험	전자서명 생성 및 검증 시험
		대칭키 암/복호화 시험
		비대칭키 암/복호화 시험
		키설정, 키유도 알고리즘 시험

	핵심기능 시험	키 쌍 일치시험
		잡음원 건전성 시험
주기적 자가시험	암호 알고리즘 시험	검증대상 해시 KAT
		검증대상 HMAC KAT
		난수발생기 KAT 시험
		전자서명 생성 및 검증 시험
		대칭키 암/복호화 시험
		비대칭키 암/복호화 시험
		키설정, 키유도 알고리즘 시험
	핵심기능 시험	키 쌍 일치시험
		잡음원 건전성 시험
	암호모듈 소프트웨어 무결성 검증 시험	

암호모듈의 무결성 검증은 동작 전 자가시험과 자가시험 함수 API 호출을 통하여 수행할 수 있으며, 적재를 시작하면 동작 전 자가시험을 통하여 암호모듈의 무결성 검증을 자동으로 수행하게 된다.
동작 전 자가시험 시 무결성 검증은 검증모드에서 지원하는 HMAC (SHA-256)을 이용하며 암호학적 난수 생성기에서 생성된 무결성 검증키를 이용하여 무결성 검증을 수행한다.
암호모듈은 해당 플랫폼의 시스템 환경변수에서 지정하고 있는 경로에서 검색하여 암호모듈 파일(MagicCrypto.{dll,so,sl,a,dylib})을 찾아내어 무결성 검증시험의 입력으로 한다. 시스템 환경변수는 운영환경에 따라 각각 다음과 같다.

운영환경	**환경변수**
Windows	PATH

SunOS, (embedded) Linux, QNX, Tizen, Android, QNX	LD_LIBRARY_PATH
HP-UX	SHLIB_PATH
IBMAIX	LIBPATH
MacOS	DYLD_LIBRARY_PATH

환경변수에 지정된 경로에는 반드시 동일한 이름의 암호모듈을 하나만 설치해야 한다. 환경변수의 경로에 암호모듈이 2 개 이상 설치 된 경우, 환경변수 설정 우선순위에 따라 예상과는 다른 암호모듈이 호출되어 오동작 할 수 있다.
무결성 검증이 실패할 경우 상태출력 인터페이스 함수의 리턴값을 통하여 MC_FAIL 을 출력하고 정상 상태인 경우 MC_OK 를 출력한다.

2.2.1. Android 에서의 사용

2.2.1.1. 패키지 경로 획득

```
File libDir = new File(this.getApplicationInfo().nativeLibraryDir);
String libPath = libDir.getAbsolutePath()+ "/lib:/system/lib";
```

App 의 패키지 경로는 MainActivity 클래스에서 위와 같이 획득 할 수 있다.

2.2.1.1. 경로 지정

Android 환경에서 구동 할 경우 버전에 따라 다르게 동작한다. 6.0 미만의 버전에서는, 메모리에 적재되어 자동으로 초기화 되는 경우 (JNI_OnLoad 함수) 암호모듈을 읽어오지 못해 무결성 검증에 실패하게 된다. 정상적으로 암호모듈을 사용하기 위해서는 setenv 함수를 이용하여 암호모듈이 위치한 경로를 설정 한 후 명시적으로 MC_Initialize 를 호출 해야 한다.

```
char libpath[512];
```

```
strcpy(libpath, packagepath);
rv = setenv("LD_LIBRARY_PATH", libpath, 1);
```

Android 버전 6.0 이상의 버전에서는 암호모듈 경로를 자동으로 읽어오게 되어 위의 설정 없이도 정상 동작 한다.

2.2.1.2. 옵션 지정

Android Studio 개발 툴에서 암호모듈을 사용한 응용프로그램 개발 시 App 으로 패키징 되면서 무결성 값을 자동으로 strip 시키는 경우가 있어 $(PROJECT_HOME)/app/build.gradle 파일에 아래와 같이 옵션을 지정해 주어야 한다.

```
packagingOptions {
   doNotStrip “*/armeabi/libMagicCrypto.so”
   doNotStrip “*/arm64-v8a/libMagicCrypto.so”
   doNotStrip “*/armeabi-v7a/libMagicCrypto.so”
}
```

2.3. 초기화와 종료

초기화는 암호모듈을 사용할 수 있도록 관련 구조체와 내부 자원에 대한 초기화와 할당을 수행하며, 종료는 사용한 자원에 대한 반납을 수행하도록한다. (5.2.3, 5.2.4 참조)

2.4. 운영 모드 변환

운영 모드는 KCMV 와 NON-KCMV 를 구분하여 암호모듈을 사용할 수 있도록 설계되었다.
운영모드 변환은 MC_SetApiMode 함수, 운영모드 확인은 MC_GetApiMode 함수를 통하여 수행 할 수 있다. (5.2.5 참조)

2.5. 세션 관리

세션 단위의 암호연산을 수행하도록 세션을 열고 닫을 수 있다.(5.2.8, 5.2.9 참조)

2.6. 에러 처리

각각의 기능에 따른 에러와 파라미터 설정에 따른 에러의 출력을 지원한다. 사용자는 출력된 에러 값을 참고하여 사용상의 문제점을 확인하고 에러를 수정할 수 있다. (5.2.10, 5.2.11 참조)
에러 코드에 따른 값과 출력되는 스트링은 다음과 같다

2.7. 객체 관리

내부적으로 사용되는 객체(Object)의 생성, 파괴 등의 관리를 수행한다. (5.2.12~5.2.14 참조)

2.8. 서명 및 검증

공개키 알고리즘을 통한 전자서명과 서명에 대한 검증을 지원한다.
전자서명 및 검증을 위하여 RSA, KCDSA, ECDSA 와 같은 암호 알고리즘을 지원하고 있다. (5.2.15~5.2.22 참조)

2.9. 암호화 및 복호화

대칭키 및 공개키 알고리즘을 통한 암호화와 복호화를 지원한다. (5.2.23~5.2.31 참조)

2.10. 해쉬 및 MAC

해쉬 알고리즘을 통한 데이터의 해쉬값 생성과 HMAC 를 지원한다. (5.2.32~5.2.43 참조)

2.11. 난수 및 키 관리

난수의 생성 및 공개키 및 개인키 쌍 생성을 지원한다. (7.1.44~7.1.47 참조)
난수생성기 부분에 사용자의 입력을 컨텍스트의 파라미터에 받아 엔트로피를 높이도록 기능하고 있다.
공개키와 개인키의 생성은 표준에 따라 안전하게 생성하고 있다.

3. 암호모듈 운영모드

MagicCrypto 암호모듈은 두가지 운영모드를 지원하고 있다. 암호검증제도에 적합한 암호알고리즘을 전용으로 사용할 수 있게 해주는 KCMV(MC_MODE_ KCMV 로 정의됨) 모드와 임의의 암호알고리즘을 사용할 수 있는 NON_KCMV(MC_MODE_ NON_KCMV) 모드이다.
KCMV 모드의 경우 안전한 알고리즘을 사용하도록 정의되어 있으므로 사용자의 알고리즘의 선택이 용이하다.

3.1. 검증대상 (KCMV)모드

MagicCrypto 암호모듈을 초기 설치하면 기본으로 KCMV 모드로 운영되도록 되어 있다. NON_KCMV 모드로 설정되어 있는 경우 MC_SetApiMode 함수를 사용하여 운영모드를 변경할 수 있다. (6.1.10 참조)
KCMV 모드에서 지원하는 암호알고리즘 정의는 다음과 같다.

3.1.1. 해쉬 알고리즘

구분	암호알고리즘 정의	설명
해쉬	MC_ALGID_SHA256	SHA256 알고리즘
	MC_ALGID_SHA224	SHA224 알고리즘
	MC_ALGID_SHA384	SHA384 알고리즘
	MC_ALGID_SHA512	SHA512 알고리즘
	MC_ALGID_LSH224	LSH224 알고리즘
	MC_ALGID_LSH256	LSH256 알고리즘
	MC_ALGID_LSH384	LSH384 알고리즘
	MC_ALGID_LSH512	LSH512 알고리즘

3.1.2. 서명 알고리즘

구분	암호알고리즘 정의	설명

서명	MC_ALGID_SHA256WithRSAPSS	SHA256 with RSA-PSS 알고리즘
	MC_ALGID_SHA256WithKCDSA	SHA256 with KCDSA 알고리즘
	MC_ALGID_SHA256WithECDSA_P224_12	SHA256 with ECDSA Prime224 알고리즘
	MC_ALGID_SHA256WithECDSA_B233_10	SHA256 with ECDSA Binary233 알고리즘
	MC_ALGID_SHA256WithECDSA_B233_11	SHA256 with ECDSA Binary233 알고리즘
	MC_ALGID_SHA256WithECDSA_P256_r1	SHA256 with ECDSA Prime256 알고리즘
	MC_ALGID_SHA256WithECDSA_B283_r1	SHA256 with ECDSABinary283r 알고리즘
	MC_ALGID_SHA256WithECDSA_B283_k1	SHA256 with ECDSABinary283k 알고리즘

3.1.3. 공개키 암호화 알고리즘

구분	암호알고리즘 정의	설명
공개키 암호화	MC_ALGID_SHA256WithRSAOAEP	SHA256 with RSA-OAEP 암호화 알고리즘

3.1.4. 대칭키 암호화 알고리즘

구분	암호알고리즘 정의	설명
대칭키 암호화	MC_ALGID_SEED	SEED 알고리즘
	MC_ALGID_ARIA_128BITKEY	128bit 키를 사용하는 ARIA 알고리즘
	MC_ALGID_ARIA_192BITKEY	192bit 키를 사용하는 ARIA 알고리즘
	MC_ALGID_ARIA_256BITKEY	256bit 키를 사용하는 ARIA 알고리즘
	MC_ALGID_LEA_128BITKEY	128bit 키를 사용하는 LEA 알고리즘
	MC_ALGID_LEA_192BITKEY	192bit 키를 사용하는 LEA 알고리즘
	MC_ALGID_LEA_256BITKEY	256bit 키를 사용하는 LEA 알고리즘
	MC_ALGID_HIGHT	HIGHT 알고리즘

3.1.5. MAC 알고리즘

구분	암호알고리즘 정의	설명
MAC	MC_ALGID_SHA256_HMAC	SHA256 HMAC 알고리즘

	MC_ALGID_SHA384_HMAC	SHA384 HMAC 알고리즘
	MC_ALGID_SHA512_HMAC	SHA512 HMAC 알고리즘
Cipher Mac	MC_ALGID_SEED_GMAC	SEED GMAC 알고리즘
	MC_ALGID_ARIA_128BITKEY_GMAC	128bit 키를 사용하는 ARIA GMAC 알고리즘
	MC_ALGID_ARIA_192BITKEY_GMAC	192bit 키를 사용하는 ARIA GMAC 알고리즘
	MC_ALGID_ARIA_256BITKEY_GMAC	256bit 키를 사용하는 ARIA GMAC 알고리즘
	MC_ALGID_LEA_128BITKEY_GMAC	128bit 키를 사용하는 LEA GMAC 알고리즘
	MC_ALGID_LEA_192BITKEY_GMAC	192bit 키를 사용하는 LEA GMAC 알고리즘
	MC_ALGID_LEA_256BITKEY_GMAC	256bit 키를 사용하는 LEA GMAC 알고리즘

3.1.6. 키 설정 알고리즘

구분	암호알고리즘 정의	설명
키 설정	MC_ALGID_DH	DH 키설정 알고리즘
	MC_ALGID_ECDH224P10	ECDH (P-224) 키설정 알고리즘
	MC_ALGID_ECDH_P256_r1	ECDH (P-256) 키설정 알고리즘

3.1.7. 키 유도 알고리즘

구분	암호알고리즘 정의	설명
키 유도	MC_ALGID_SHA256WithPBKDF2	PKCS#5 PBKDF2 키 유도 알고리즘
	MC_ALGID_SHA256WithKBKDF_CTR	HMac Counter 모드 키 유도 알고리즘

3.1.8. 난수 생성 알고리즘

구분	암호알고리즘 정의	설명
난수	MC_ALGID_SHA256DRBG	ISO/IEC 18031 Hash DRBG 난수 생성기

3.1.9. 키 생성 알고리즘

구분	암호알고리즘 정의	설명
키쌍 생성	MC_ALGID_RSA2048	RSA 용 2048bit 키 생성 알고리즘
	MC_ALGID_RSA3072	RSA 용 3072bit 키 생성 알고리즘
	MC_ALGID_RSA2048P2V2	RSA v2 용 2048bit 키 생성 알고리즘
	MC_ALGID_RSA3072P2V2	RSA v2 용 3072bit 키 생성 알고리즘
	MC_ALGID_KCDSAWithParam	파라미터를 사용한 KCDSA 키생성 알고리즘
	MC_ALGID_KCDSA2048	KCDSA 용 2048bit 키 생성 알고리즘
	MC_ALGID_DHWithParam	파라미터를 사용한 DH 키생성 알고리즘
	MC_ALGID_DH2048	DH 용 2048bit 키 생성 알고리즘
	MC_ALGID_EC_P224_12	ECC (Prime-224) 키 생성 알고리즘
	MC_ALGID_EC_B233_10	ECC (Binary233 #10) 키 생성 알고리즘
	MC_ALGID_EC_B233_11	ECC (Binary233 #11) 키 생성 알고리즘
	MC_ALGID_EC_P256_r1	ECC (Prime-256) 키 생성 알고리즘
	MC_ALGID_EC_B283_r1	ECC (Binary-283 r1) 키 생성 알고리즘
	MC_ALGID_EC_B283_k1	ECC (Binary-283 k1) 키 생성 알고리즘

3.2. 비 검증 대상(NON-KCMV) 모드

KCMV 의 대상이 아닌 임의의 암호알고리즘을 포함하는 모듈을 사용하고자 하는 경우 사용자는 NON_KCMV 모드로 설정하면 추가적인 알고리즘의 사용을 할 수 있다. 모드의 변경은 MC_SetApiMode 함수를 사용하여 수행할 수 있다. (6.1.10 참조)
NON_KCMV 모드에서 지원하는 암호알고리즘 정의는 다음과 같다.

3.2.1. 해쉬 알고리즘

구분	암호알고리즘 정의	설명
해쉬	MC_ALGID_MD5	MD5 알고리즘
	MC_ALGID_SHA1	SHA1 알고리즘
	MC_ALGID_SHA256	SHA256 알고리즘

	MC_ALGID_SHA224	SHA224 알고리즘
	MC_ALGID_SHA384	SHA384 알고리즘
	MC_ALGID_SHA512	SHA512 알고리즘
	MC_ALGID_LSH224	LSH224 알고리즘
	MC_ALGID_LSH256	LSH256 알고리즘
	MC_ALGID_LSH384	LSH384 알고리즘
	MC_ALGID_LSH512	LSH512 알고리즘
	MC_ALGID_HAS160	HAS160 알고리즘

3.2.2. 서명 알고리즘

구분	암호알고리즘 정의	설명
서명	MC_ALGID_MD5WithRSA	MD5 with RSA 알고리즘
	MC_ALGID_SHA1WithRSA	SHA1 with RSA 알고리즘
	MC_ALGID_SHA256WithRSA	SHA256 with RSA 알고리즘
	MC_ALGID_SHA384WithRSA	SHA384 with RSA 알고리즘
	MC_ALGID_SHA512WithRSA	SHA512 with RSA 알고리즘
	MC_ALGID_SHA1WithRSAPSS	SHA1 with RSA-PSS 알고리즘
	MC_ALGID_SHA256WithRSAPSS	SHA256 with RSA-PSS 알고리즘
	MC_ALGID_SHA384WithRSAPSS	SHA384 with RSA-PSS 알고리즘
	MC_ALGID_SHA512WithRSAPSS	SHA512 with RSA-PSS 알고리즘
	MC_ALGID_SHA1WithKCDSA	SHA1 with KCDSA 알고리즘
	MC_ALGID_SHA256WithKCDSA	SHA256 with KCDSA 알고리즘
	MC_ALGID_HAS160WithKCDSA	HAS160 with KCDSA 알고리즘
	MC_ALGID_SHA1WithECDSA	SHA1 with ECDSA 알고리즘
	MC_ALGID_SHA1WithECDSA_B163_3	SHA1 with ECDSA 알고리즘(FIPS K-163)
	MC_ALGID_SHA1WithECDSA_B163_5	SHA1 with ECDSA 알고리즘(WTLS C-165)
	MC_ALGID_SHA256WithECDSA_P224_12	SHA256 with ECDSA 알고리즘(P-224)
	MC_ALGID_SHA256WithECDSA_B233_10	SHA256 with ECDSA 알고리즘(B-233_10)
	MC_ALGID_SHA256WithECDSA_B233_11	SHA256 with ECDSA 알고리즘(B-233_11)
	MC_ALGID_SHA256WithECDSA_P256_r1	SHA256 with ECDSA 알고리즘(P-256r1)
	MC_ALGID_SHA384WithECDSA_P256_r1	SHA384 with ECDSA 알고리즘(P-256r1)
	MC_ALGID_SHA512WithECDSA_P256_r1	SHA512 with ECDSA 알고리즘(P-256r1)

	MC_ALGID_SHA256WithECDSA_P256_k1	SHA256 with ECDSA 알고리즘(P-256k1)
	MC_ALGID_SHA384WithECDSA_P256_k1	SHA384 with ECDSA 알고리즘(P-256k1)
	MC_ALGID_SHA512WithECDSA_P256_k1	SHA512 with ECDSA 알고리즘(P-256k1)
	MC_ALGID_SHA256WithECDSA_B283_r1	SHA256 with ECDSA 알고리즘(B-283r1)
	MC_ALGID_SHA384WithECDSA_B283_r1	SHA384 with ECDSA 알고리즘(B-283r1)
	MC_ALGID_SHA512WithECDSA_B283_r1	SHA512 with ECDSA 알고리즘(B-283r1)
	MC_ALGID_SHA256WithECDSA_B283_k1	SHA256 with ECDSA 알고리즘(B-283k1)
	MC_ALGID_SHA384WithECDSA_B283_k1	SHA384 with ECDSA 알고리즘(B-283k1)
	MC_ALGID_SHA512WithECDSA_B283_k1	SHA512 with ECDSA 알고리즘(B-283k1)
	MC_ALGID_SHA256WithECDSA_P384_r1	SHA256 with ECDSA 알고리즘(P-384r1)
	MC_ALGID_SHA384WithECDSA_P384_r1	SHA384 with ECDSA 알고리즘(P-384r1)
	MC_ALGID_SHA512WithECDSA_P384_r1	SHA512 with ECDSA 알고리즘(P-384r1)
	MC_ALGID_SHA256WithECDSA_P521_r1	SHA256 with ECDSA 알고리즘(P-521r1)
	MC_ALGID_SHA384WithECDSA_P521_r1	SHA384 with ECDSA 알고리즘(P-521r1)
	MC_ALGID_SHA512WithECDSA_P521_r1	SHA512 with ECDSA 알고리즘(P-521r1)

3.2.3. 공개키 암호화 알고리즘

구분	암호알고리즘 정의	설명
공개키 암호화	MC_ALGID_RSA	RSA 암호화 알고리즘
	MC_ALGID_SHA1WithRSAOAEP	SHA1 with RSA-OAEP 암호화 알고리즘
	MC_ALGID_SHA256WithRSAOAEP	SHA256 with RSA-OAEP 암호화 알고리즘

3.2.4. 대칭키 암호화 알고리즘

구분	암호알고리즘 정의	설명
대칭키 암호화	MC_ALGID_3DES_3KEY	3Key 3DES 알고리즘
	MC_ALGID_DES	DES 알고리즘
	MC_ALGID_SEED	SEED 알고리즘
	MC_ALGID_ARIA_128BITKEY	128bit 키를 사용하는 ARIA 알고리즘
	MC_ALGID_ARIA_192BITKEY	192bit 키를 사용하는 ARIA 알고리즘
	MC_ALGID_ARIA_256BITKEY	256bit 키를 사용하는 ARIA 알고리즘

	MC_ALGID_AES_128BITKEY	128bit 키를 사용하는 AES 알고리즘
	MC_ALGID_AES_192BITKEY	192bit 키를 사용하는 AES 알고리즘
	MC_ALGID_AES_256BITKEY	256bit 키를 사용하는 AES 알고리즘
	MC_ALGID_LEA_128BITKEY	128bit 키를 사용하는 LEA 알고리즘
	MC_ALGID_LEA_192BITKEY	192bit 키를 사용하는 LEA 알고리즘
	MC_ALGID_LEA_256BITKEY	256bit 키를 사용하는 LEA 알고리즘
	MC_ALGID_RC2_40BITKEY	40bit 키를 사용하는 RC2 알고리즘
	MC_ALGID_RC2_128BITKEY	128bit 키를 사용하는 RC2 알고리즘
	MC_ALGID_RC2_192BITKEY	192bit 키를 사용하는 RC2 알고리즘
	MC_ALGID_RC2_256BITKEY	256bit 키를 사용하는 RC2 알고리즘
	MC_ALGID_RC4	RC4 알고리즘
	MC_ALGID_HIGHT	HIGHT 알고리즘

3.2.5. MAC 알고리즘

구분	암호알고리즘 정의	설명
MAC	MC_ALGID_MD5_HMAC	MD5 HMAC 알고리즘
	MC_ALGID_SHA1_HMAC	SHA1 HMAC 알고리즘
	MC_ALGID_SHA256_HMAC	SHA256 HMAC 알고리즘
	MC_ALGID_SHA384_HMAC	SHA384 HMAC 알고리즘
	MC_ALGID_SHA512_HMAC	SHA512 HMAC 알고리즘
	MC_ALGID_DESCBC_MAC	DES CBC MAC 알고리즘
Cipher Mac	MC_ALGID_SEED_GMAC	SEED GMAC 알고리즘
	MC_ALGID_ARIA_128BITKEY_GMAC	128bit 키를 사용하는 ARIA GMAC 알고리즘
	MC_ALGID_ARIA_192BITKEY_GMAC	128bit 키를 사용하는 ARIA GMAC 알고리즘
	MC_ALGID_ARIA_256BITKEY_GMAC	128bit 키를 사용하는 ARIA GMAC 알고리즘
	MC_ALGID_AES_128BITKEY_GMAC	128bit 키를 사용하는 ARIA GMAC 알고리즘
	MC_ALGID_AES_192BITKEY_GMAC	128bit 키를 사용하는 ARIA GMAC 알고리즘
	MC_ALGID_AES_256BITKEY_GMAC	128bit 키를 사용하는 ARIA GMAC 알고리즘
	MC_ALGID_LEA_128BITKEY_GMAC	128bit 키를 사용하는 ARIA GMAC 알고리즘
	MC_ALGID_LEA_192BITKEY_GMAC	128bit 키를 사용하는 ARIA GMAC 알고리즘
	MC_ALGID_LEA_256BITKEY_GMAC	128bit 키를 사용하는 ARIA GMAC 알고리즘

3.2.6. 키 설정 알고리즘

구분	암호알고리즘 정의	설명
키 설정	MC_ALGID_DH	DH 키설정 알고리즘
	MC_ALGID_ECDH_P224_12	ECDH (P-224) 키설정 알고리즘
	MC_ALGID_ECDH_P256_r1	ECDH (P-256) 키설정 알고리즘

3.2.7. 난수 생성 알고리즘

구분	암호알고리즘 정의	설명
난수	MC_ALGID_SHA256DRBG	ISO/IEC 18031 Hash DRBG 난수 생성기

3.2.8. 키 생성 알고리즘

구분	암호알고리즘 정의	설명
키 생성	MC_ALGID_RSA512	RSA 용 512bit 키 생성 알고리즘
	MC_ALGID_RSA1024	RSA 용 1024bit 키 생성 알고리즘
	MC_ALGID_RSA2048	RSA 용 2048bit 키 생성 알고리즘
	MC_ALGID_RSA3072	RSA 용 3072bit 키 생성 알고리즘
	MC_ALGID_RSA4096	RSA 용 4096bit 키 생성 알고리즘
	MC_ALGID_RSA1024P2V2	RSA v2 용 1024bit 키 생성 알고리즘
	MC_ALGID_RSA2048P2V2	RSA v2 용 2048bit 키 생성 알고리즘
	MC_ALGID_RSA3072P2V2	RSA v2 용 3072bit 키 생성 알고리즘
	MC_ALGID_RSA4096P2V2	RSA v2 용 4096bit 키 생성 알고리즘
	MC_ALGID_KCDSAWithParam	파라미터를 사용한 KCDSA 키생성 알고리즘
	MC_ALGID_KCDSA1024	KCDSA 용 1024bit 키 생성 알고리즘
	MC_ALGID_KCDSA2048	KCDSA 용 2048bit 키 생성 알고리즘
	MC_ALGID_EC_B163_3	FIPS K-163 커브 키 생성 알고리즘
	MC_ALGID_EC_B163_5	WTLS C-165 커브 키 생성 알고리즘
	MC_ALGID_EC_P224_12	Prime 224 #12 커브 키 생성 알고리즘
	MC_ALGID_EC_B233_10	Binary 233 #10 커브 키 생성 알고리즘

	MC_ALGID_EC_B233_11	Binary 233 #11 커브 키 생성 알고리즘
	MC_ALGID_EC_P256_r1	Prime256 r1 커브 키 생성 알고리즘
	MC_ALGID_EC_P256_k1	Prime256 k1 커브 키 생성 알고리즘
	MC_ALGID_EC_B283_r1	Binary283 r1 커브 키 생성 알고리즘
	MC_ALGID_EC_B283_k1	Binary283 k1 커브 키 생성 알고리즘
	MC_ALGID_EC_P384_r1	Prime 384r1 커브 키 생성 알고리즘
	MC_ALGID_EC_P521_r1	Prime 521 r1 커브 키 생성 알고리즘
	MC_ALGID_DHWithParam	파라미터를 사용한 DH 키생성 알고리즘
	MC_ALGID_DH1024	DH 용 1024bit 키 생성 알고리즘
	MC_ALGID_DH2048	DH 용 2048bit 키 생성 알고리즘

4. 암/복호화 운영모드 및 패딩방식 설정

4.1. 운영모드 설정

MagicCrypto 암호모듈은 다양한 암/복호화 운영모드를 지원한다. 암호모듈이 초기화 되면 기본적으로 CBC 모드를 사용한 암/복호화를 수행할 수 있는 상태로 되며, MC_SetOption 함수를 이용하여 ECB 모드, CBC 모드 등 필요에 따라 암호운영모드를 설정할 수 있다.

구분	운영모드 ID	설명
운영모드 설정	MC_ALGMODE_CBC	CBC 모드 설정
	MC_ALGMODE_ECB	ECB 모드 설정
	MC_ALGMODE_CTR	CTR 모드 설정
	MC_ALGMODE_GCM	GCM 모드 설정
	MC_ALGMODE_CCM	CCM 모드 설정

4.2. 패딩방식 설정

MagicCrypto 암호모듈은 다양한 메시지 패딩방식을 지원한다. 암호모듈이 초기화 되면 기본적으로 PKCS#5 패딩을 사용할 수 있는 상태로 되며, MC_SetOption 함수를 이용하여 SSL 패딩, One 패딩 등 필요에 따라 메시지 패딩방식을 설정할 수 있다.

구분	운영모드 ID	설명
패딩방식 설정	MC_PADTYPE_PKCS5	PKCS#5 방식의 패딩 설정
	MC_PADTYPE_PKCS5	
	MC_PADTYPE_SSL	SSL 방식의 패딩 설정
	MC_PADTYPE_WTLS	
	MC_PADTYPE_NONE	패딩을 하지 않음
	MC_PADTYPE_ONE	0x80, 0x00, … , 0x00 을 패딩
	MC_PADTYPE_X923	부족한 만큼 0x00 채우고 마지막 바이트만 padding 된 바이트 수 기록

5. 암호모듈 사용예제

5.1. 해쉬

입출력 데이터는 기본적으로 16 진수 바이너리 데이터를 이용하여 메시지와 해쉬값을 입출력하며 콘솔을 통하여 입력 받는 경우 16 진수로 변환해 주어야 한다.

```
void Digest()
{
	MC_RV rv;
	MC_ALGORITHM mcAlg = {MC_ALGID_NONE, NULL, 0};
	MC_HSESSION hSession = 0;
	MC_UCHAR data[4096], out[64];
	MC_UINT datalen, outlen;
	MC_CHAR ch;

	// 세션 초기화
	rv = MC_OpenSession(&hSession);
	if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 1;}

	// API 운영모드 설정
	// 기본설정값은 MC_MODE _KCMV
	// MC_MODE_NON_KCMV, MC_MODE _KCMV
	rv = MC_SetApiMode(hSession, MC_MODE _KCMV);
	if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 2;}

	// 지원 알고리즘
	// MC_ALGID_MD5, MC_ALGID_SHA1, MC_ALGID_SHA256, MC_ALGID_HAS160
	mcAlg.mcAlgId = MC_ALGID_SHA256;

	// 메시지 입력
	inputData("Message : ", data, datalen);
	// 스트링으로 입력받아 16 진수 바이너리 데이터로 처리한다.
	datalen = processData(data);

	outlen = sizeof(out);
	memset(out, 0, outlen);

	// 해쉬 초기화, 알고리즘 설정
	rv = MC_DigestInit(hSession, &mcAlg);
	if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

	// 메시지 해쉬
	rv = MC_Digest(hSession, data, datalen, out, &outlen);
	if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

	// 해쉬값 출력
```

```
        printf("Digest(%d) : ₩n", outlen);
        printOutput(out, outlen);

        // 세션 종료
        MC_CloseSession(hSession);
}
```

5.2. 메시지 인증

입출력 데이터는 기본적으로 16 진수 바이너리 데이터를 이용하여 메시지와 해쉬값을 입출력하며 콘솔을 통하여 입력 받는 경우 16 진수로 변환해 주어야 한다.
인증키는 MC_CreateObject 를 이용하여 사용자의 입력값으로 인증키를 설정하거나, MC_GenerateKey 를 이용하여 인증키를 생성한다.

```
void Mac()
{
        MC_RV rv;
        MC_ALGORITHM mcAlg = { MC_ALGID_SHA256_HMAC, NULL, 0};
        MC_HOBJECT hKey = 0;
        MC_HSESSION hSession = 0;
        MC_UCHAR data[4096], out[64], key[256];
        MC_UINT datalen, outlen, keylen;
        MC_CHAR ch;

        // 세션 초기화
        rv = MC_OpenSession(&hSession);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 1;}

        // 인증키 입력 (스트링)
        inputData("MAC key [Gen]: ", key, keylen);
        // 스트링을 바이너리로 변환
        keylen = processData(key);
        if(keylen == 0)
                // 입력이 없으면 키 생성
                MC_GenerateKey(hSession, &mcAlg, &hKey);
        Else
                // 인증키 설정
                MC_CreateObject(hSession, key, keylen, &hKey);

        // 메시지 입력 (스트링)
        inputData("Message : ", data, datalen);
        // 스트링을 바이너리로 변환
        datalen = processData(data);

        outlen = sizeof(out);
        memset(out, 0, outlen);
```

```
        // 메시지 인증 초기화
        rv = MC_CreateMacInit(hSession, &mcAlg, hKey);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

        // 메시지 인증값 계산
        rv = MC_CreateMac(hSession, data, datalen, out, &outlen);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

        // 메시지 인증값 출력
        printf("MAC(%d) : ₩n", outlen);
        printOutput(out, outlen);

        // 메시지 인증값 검증 초기화
        rv = MC_VerifyMacInit(hSession, &mcAlg, hKey);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

        // 메시지 인증값 검증
        rv = MC_VerifyMac(hSession, data, datalen, out, outlen);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}
        else printf("Verify Success₩n");

        // 인증키 제로화
        MC_DestroyObject(hSession, hKey);
        // 세션 종료
        MC_CloseSession(hSession);
}
```

5.3. 전자서명

입출력 데이터는 기본적으로 16 진수 바이너리 데이터를 이용하여 메시지와 해쉬값을 입출력하며 콘솔을 통하여 입력 받는 경우 16 진수로 변환해 주어야 한다.

키쌍은 MC_CreateObject 를 이용하여 사용자의 입력값으로 개인키 및 공개키를 설정하거나, MC_GenerateKeyPair 를 이용하여 개인키, 공개키쌍을 생성한다. 개인키, 공개키쌍을 설정하는 경우 MC_CreateObject 를 사용하여 5.8 절의 데이터 구조를 따라 개인키 세션 오브젝트 핸들, 공개키 세션 오브젝트 핸들로 설정할 수 있다.

```
void Sign()
{
        MC_RV rv;
        MC_ALGORITHM mcAlg = {0,0,0}, mcKeyAlg = {0,0,0};
        MC_HOBJECT hPubkey = 0, hPrikey = 0;
        MC_HSESSION hSession = 0;
        MC_UCHAR data[4096], out[512], key[1024], param[1024], tmp[1024];
```

```
        MC_UINT datalen, outlen, keylen, paramlen, tmplen;
        MC_CHAR ch;

        // 세션 초기화
        rv = MC_OpenSession(&hSession);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 1;}

        // 알고리즘 설정
        // MC_ALGID_SHA1WithRSAPSS, MC_ALGID_SHA1WithRSA
        // MC_ALGID_SHA1WithECDSA163B3, MC_ALGID_SHA1WithECDSA163B5
        // MC_ALGID_SHA1WithKCDSA, MC_ALGID_SHA256WithKCDSA
        mcAlg.mcAlgId = MC_ALGID_SHA256WithKCDSA;

        // 메시지 입력 (스트링)
        inputData("Message : ", data, datalen);
        // 스트링을 바이너리로 변환
        datalen = processData(data);

        // 개인키 입력 (스트링)
        inputData("Private key [Gen]: ", key, keylen);
        // 스트링을 바이너리로 변환
        keylen = processData(key);
        if(keylen == 0)
        {
                // 개인키 입력이 없으면 키쌍 생성
                if(mcAlg.mcAlgId == MC_ALGID_SHA1WithRSAPSS) mcKeyAlg.mcAlgId
= MC_ALGID_RSA1024;
                else    if(mcAlg.mcAlgId    ==    MC_ALGID_SHA1WithECDSA163B3)
mcKeyAlg.mcAlgId = MC_ALGID_ECC163B3;
                else    if(mcAlg.mcAlgId    ==    MC_ALGID_SHA1WithECDSA163B5)
mcKeyAlg.mcAlgId = MC_ALGID_ECC163B5;
                else if(mcAlg.mcAlgId == MC_ALGID_SHA256WithKCDSA) {
                        // KCDSA 의 경우 파라메터 설정
                        mcKeyAlg.mcAlgId = MC_ALGID_KCDSAWithParam;
                        paramlen                =                HexToBin(param,
"3082011f0281810094 0f2886f849956805be4d218f448ab5aa0877beb6a54c783676dcca
03aae5f9ff6de10b13c0c30902053910a333d6c07a7a1652ffd3ff7576f8c1943f42950b968
b889a31cef1184d1d17fde6a54b03c303155d4e3b7eaa4ed8605388c2a94ebd824cb0d0fb
a7f6c8583daad73e5e07ec4198a7f8499daf04474d590460cda3021500f16ef4500e045f30
ca486848b02e9a0ebdbcb44b028181008c7c44bbcf8c1cd086c7be59f4aa45e07e53c3430
3e7bbee8e6006729ef2ff7728e693afb66f00ce7dcdad5f5ba1d4b9d93b03553d2f2cecf34e
c084a8f5fc6ef44c3119bec4baff8fa989af3e87034dfd61e4e9ceab6331d62847824479592
4c25781ecf0f8ee5ce8118b8f69a321ccfec99629c067f9ba3225282abcf2e32a");
                        mcKeyAlg.pParam = param;
                        mcKeyAlg.nParam = paramlen;
                }
                // 키쌍 생성
                rv = MC_GenerateKeyPair(hSession, &mcKeyAlg, &hPubkey, &hPrikey);
                if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}
        }
        Else
                // 개인키 설정
```

```
            MC_CreateObject(hSession, key, keylen, &hPrikey);

        outlen = sizeof(out);
        memset(out, 0, outlen);

        // 전자서명 생성 초기화
        rv = MC_SignInit(hSession, &mcAlg, hPrikey);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

        // 전자서명 생성
        rv = MC_Sign(hSession, data, datalen, out, &outlen);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

        // 전자서명 값 출력
        printf("Sign(%02d) : ₩n", outlen);
        printOutput(out, outlen);

        printf("New sign value for verify [y/N]? "); ch = getchar();
        if(!memicmp(&ch, (char*)"y", 1))
        {
            inputData("Sign : ", out, outlen);
            outlen = processData(out);
        }

        // 공개키 입력 (스트링)
        inputData("Public key [Gen]: ", key, keylen);
        // 스트링을 바이너리로 변환
        keylen = processData(key);
        if(keylen != 0)
            // 키쌍 생성하지 않을 경우 공개키 설정
            MC_CreateObject(hSession, key, keylen, &hPubkey);

        // KCDSA 의 경우 파라메터 설정
        if(mcAlg.mcAlgId == MC_ALGID_SHA1WithKCDSA)
        {
            inputData("Parameter [Gen]: ", tmp, tmplen);
            tmplen = processData(tmp);
            if(tmplen != 0) {
                mcAlg.pParam = tmp;
                mcAlg.nParam = tmplen;
            }
            else {
                mcAlg.pParam = mcKeyAlg.pParam;
                mcAlg.nParam = mcKeyAlg.nParam;
            }
        }

        // 전자서명 검증 초기화
        rv = MC_VerifyInit(hSession, &mcAlg, hPubkey);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

        // 전자서명 검증
```

```
        rv = MC_Verify(hSession, data, datalen, out, outlen);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}
        else printf("Verify Success₩n");

        // 공개키 제로화
        MC_DestroyObject(hSession, hPubkey);
        // 개인키 제로화
        MC_DestroyObject(hSession, hPrikey);
        // 세션 종료
        MC_CloseSession(hSession);
}
```

5.4. 비밀키 암호화

입출력 데이터는 기본적으로 16 진수 바이너리 데이터를 이용하여 메시지와 해쉬값을 입출력하며 콘솔을 통하여 입력 받는 경우 16 진수로 변환해 주어야 한다.
비밀키는 MC_CreateObject 를 이용하여 사용자의 입력값으로 비밀키를 설정하거나, MC_GenerateKey 를 이용하여 비밀키를 생성한다.

```
void Encrypt()
{
        MC_RV rv;
        MC_ALGORITHM mcAlg = {0,0,0};
        MC_HOBJECT hKey = 0;
        MC_HSESSION hSession = 0;
        MC_ALGMODE algMode = MC_ALGMODE_CBC;
        MC_PADTYPE algPad = MC_PADTYPE_PKCS5;
        MC_UCHAR data[4096], cipher[4112], plain[4096], key[32], iv[32];
        MC_UINT datalen, cipherlen, plainlen, keylen = 32, ivlen = 32;
        MC_ALGORITHM prng = {MC_ALGID_SHA256DRBG, (MC_UCHAR*)"random",
6};
        MC_CHAR ch;

        // 세션 초기화
        rv = MC_OpenSession(&hSession);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 1;}

        // 암호모듈 운영모드 설정
        rv = MC_SetApiMode(hSession, gApimode);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 2;}

        // 지원 알고리즘
        // MC_ALGID_SEED, MC_ALGID_ARIA_128BITKEY
        // MC_ALGID_ARIA_192BITKEY, MC_ALGID_ARIA_256BITKEY
        // MC_ALGID_3DES_3KEY
```

```
mcAlg.mcAlgId = MC_ALGID_SEED;

// 암,복호화 운영모드 선택 (CBC/ECB/CTR/GCM/CCM)
printf("₩n1. MC_ALGMODE_CBC₩n");
printf("2. MC_ALGMODE_ECB₩n");
printf("3. MC_ALGMODE_CTR₩n");
printf("4. MC_ALGMODE_GCM₩n");
printf("5. MC_ALGMODE_CCM₩n");

printf("Select [1]: "); ch = getchar();
if(!memicmp(&ch, (char*)"2", 1)) algMode = MC_ALGMODE_ECB;
else algMode = MC_ALGMODE_CBC;

// 패딩방식 선택 (PKCS5, SSL, ONE)
printf("₩n1. MC_PADTYPE_PKCS5₩n");
printf("2. MC_PADTYPE_SSL₩n");
printf("3. MC_PADTYPE_ONE₩n");
printf("Select [1]: "); ch = getchar();
if(!memicmp(&ch, (char*)"2", 1)) algPad = MC_PADTYPE_SSL;
else if(!memicmp(&ch, (char*)"3", 1)) algPad = MC_PADTYPE_ONE;
else algPad = MC_PADTYPE_PKCS5;

// 암,복호화 운영모드와 패딩방식 설정
rv = MC_SetOption(hSession, algMode, algPad);
if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 2;}

// 평문 메시지 입력
inputData("Message : ", data, datalen);
datalen = processData(data);

// 비밀키 입력
inputData("Key [Gen]: ", key, keylen);
keylen = processData(key);
if(keylen == 0)
        // 비밀키 입력이 없는 경우 비밀키 생성
        MC_GenerateKey(hSession, &mcAlg, &hKey);
else
        // 비밀키 설정
        MC_CreateObject(hSession, key, keylen, &hKey);

// CBC 모드인 경우 초기벡터 설정
mcAlg.pParam = 0;
mcAlg.nParam = 0;
if(algMode != MC_ALGMODE_ECB)
{
        // 초기벡터 입력
        inputData("Iv [Gen]: ", iv, ivlen);
        ivlen = processData(iv);
        if(ivlen == 0)
        {
                if(mcAlg.mcAlgId == MC_ALGID_3DES_3KEY) ivlen = 8;
                else ivlen = 16;
```

```
                        // 초기벡터 생성
                        MC_GenerateRandom(hSession, &prng, iv, ivlen);
                }
                // 알고리즘 파라메터에 초기벡터 설정
                mcAlg.pParam = iv;
                mcAlg.nParam = ivlen;
        }

        cipherlen = sizeof(cipher);
        memset(cipher, 0, cipherlen);

        // 암호화 초기화
        rv = MC_EncryptInit(hSession, &mcAlg, hKey);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

        // 암호화
        rv = MC_Encrypt(hSession, data, datalen, cipher, &cipherlen);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

        printf("Ciphertext(%d) : ₩n", cipherlen);
        printOutput(cipher, cipherlen);

        printf("New ciphertext for decrypt [y/N]? "); ch = getchar();
        if(!memicmp(&ch, (char*)"y", 1))
        {
                inputData("Ciphertext : ", cipher, cipherlen);
                cipherlen = processData(cipher);
        }

        plainlen = sizeof(plain);
        memset(plain, 0, plainlen);

        // 복호화 초기화
        rv = MC_DecryptInit(hSession, &mcAlg, hKey);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

        // 복호화
        rv = MC_Decrypt(hSession, cipher, cipherlen, plain, &plainlen);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

        printf("Plaintext(%d) : ₩n", plainlen);
        printOutput(plain, plainlen);

        // 비밀키 제로화
        MC_DestroyObject(hSession, hKey);
        // 세션 종료
        MC_CloseSession(hSession);
}
```

5.5. 공개키 암호화

입출력 데이터는 기본적으로 16 진수 바이너리 데이터를 이용하여 메시지와 해쉬값을 입출력하며 콘솔을 통하여 입력 받는 경우 16 진수로 변환해 주어야 한다.
키쌍은 MC_CreateObject 를 이용하여 사용자의 입력값으로 공개키 및 개인키를 설정하거나, MC_GenerateKeyPair 를 이용하여 키쌍을 생성한다. 개인키, 공개키쌍을 설정하는 경우 MC_CreateObject 를 사용하여 5.8 절의 데이터 구조를 따라 개인키 세션 오브젝트 핸들, 공개키 세션 오브젝트 핸들로 설정할 수 있다

```
void AsEncrypt()
{
	MC_RV rv;
	MC_ALGORITHM mcAlg = {0,0,0}, mcKeyAlg = {0,0,0};
	MC_HOBJECT hPubkey = 0, hPrikey = 0;
	MC_HSESSION hSession = 0;
	MC_UCHAR data[512], cipher[512], plain[512], key[512];
	MC_UINT datalen, cipherlen, plainlen, keylen;
	MC_CHAR ch;

	rv = MC_OpenSession(&hSession);
	if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 1;}

	rv = MC_SetApiMode(hSession, gApimode);
	if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 2;}

	// 알고리즘 설정
	// MC_ALGID_SHA256WithRSAOAEP, MC_ALGID_RSA
	mcAlg.mcAlgId = MC_ALGID_SHA256WithRSAOAEP;

	// 메시지 입력
	inputData("Message : ", data, datalen);
	datalen = processData(data);

	// 공개키 입력
	inputData("Public key [Gen]: ", key, keylen);
	keylen = processData(key);
	if(keylen == 0)
	{
		// 공개키 입력이 없는 경우 키쌍 생성
		if(mcAlg.mcAlgId == MC_ALGID_SHA1WithRSAOAEP)
mcKeyAlg.mcAlgId = MC_ALGID_RSA1024;
		else if(mcAlg.mcAlgId == MC_ALGID_RSA) mcKeyAlg.mcAlgId =
MC_ALGID_RSA1024;
		MC_GenerateKeyPair(hSession, &mcKeyAlg, &hPubkey, &hPrikey);
```

```
        }
        else
                // 공개키 생성
                MC_CreateObject(hSession, key, keylen, &hPubkey);

        cipherlen = sizeof(cipher);
        memset(cipher, 0, cipherlen);

        // 공개키 암호화 초기화
        rv = MC_EncryptInit(hSession, &mcAlg, hPubkey);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

        // 공개키 암호화
        rv = MC_Encrypt(hSession, data, datalen, cipher, &cipherlen);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

        printf("Ciphertext(%d) : ₩n", cipherlen);
        printOutput(cipher, cipherlen);

        printf("New Ciphertext for decrypt [y/N]? "); ch = getchar();
        if(!memicmp(&ch, (char*)"y", 1))
        {
                inputData("Ciphertext : ", cipher, cipherlen);
                cipherlen = processData(cipher);
        }

        // 개인키 입력
        inputData("Private key [Gen]: ", key, keylen);
        keylen = processData(key);
        if(keylen != 0)
                MC_CreateObject(hSession, key, keylen, &hPrikey);

        plainlen = sizeof(plain);
        memset(plain, 0, plainlen);

        // 개인키 복호화 초기화
        rv = MC_DecryptInit(hSession, &mcAlg, hPrikey);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

        // 개인키 복호화
        rv = MC_Decrypt(hSession, cipher, cipherlen, plain, &plainlen);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

        printf("Plaintext(%d) : ₩n", plainlen);
        printOutput(plain, plainlen);

        // 공개키 제로화
        MC_DestroyObject(hSession, hPubkey);
        // 개인키 제로화
        MC_DestroyObject(hSession, hPrikey);
        // 세션 종료
        MC_CloseSession(hSession);
```

```
}
```

5.6. 키쌍 생성

개인키, 공개키쌍 생성은 암호알고리즘에 따라 생성하며 ASN.1 형식으로 키오브젝트핸들을 이용하여 출력한다.
생성된 키쌍을 이용하기 위하여 키오브젝트 핸들을 직접 이용하거나, 키를 출력하기 위하여 MC_GetObjectValue 함수를 통하여 생성된 개인키, 공개키를 출력할 수 있다.

```
void GenKeyPair()
{
	MC_RV rv;
	MC_ALGORITHM mcAlg = {0,0,0};
	MC_HOBJECT hPubkey = 0, hPrikey = 0;
	MC_HSESSION hSession = 0;
	MC_UCHAR param[1024], tmp[2048];
	MC_UINT paramlen, tmplen;
	MC_CHAR ch;

	// 세션 초기화
	rv = MC_OpenSession(&hSession);
	if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 1;}

	// 지원 알고리즘
	//MC_ALGID_RSA1024, MC_ALGID_RSA2048
	//MC_ALGID_KCDSA1024, MC_ALGID_KCDSA2048
	//MC_ALGID_ECC163B3, MC_ALGID_ECC163B5

	mcAlg.mcAlgId = MC_ALGID_RSA1024;

	// KCDSA 의 경우 알고리즘 파라메터 설정 가능
	if(mcAlg.mcAlgId == MC_ALGID_KCDSA1024 || mcAlg.mcAlgId ==
MC_ALGID_KCDSA2048)
	{
		inputData("Parameter [Gen]: ", param, paramlen);
		paramlen = processData(param);
		if(paramlen != 0)
			mcAlg.mcAlgId = MC_ALGID_KCDSAWithParam;
		mcAlg.pParam = param;
		mcAlg.nParam = paramlen;
	}

	// 개인키, 공개키쌍 생성
	rv = MC_GenerateKeyPair(hSession, &mcAlg, &hPubkey, &hPrikey);
	if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}
```

```
        // 개인키 출력
        // 할당된 출력버퍼에 키오브젝트 핸들의 데이터를 출력한다.
        // 출력 버퍼의 크기가 작을경우 에러를 발생한다.
        memset(tmp, 0x00, sizeof(tmp)); tmplen = sizeof(tmp);
        MC_GetObjectValue(hSession, hPrikey, tmp, &tmplen);
        printf("PrivateKey(%d) : ₩n", tmplen);
        printOutput(tmp, tmplen);

        // 공개키 출력
        memset(tmp, 0x00, sizeof(tmp)); tmplen = sizeof(tmp);
        MC_GetObjectValue(hSession, hPubkey, tmp, &tmplen);
        printf("PublicKey(%d) : ₩n", tmplen);
        printOutput(tmp, tmplen);

        // 개인키, 공개키 제로화
        MC_DestroyObject(hSession, hPubkey);
        MC_DestroyObject(hSession, hPrikey);
        // 세션 종료
        MC_CloseSession(hSession);
}
```

5.7. 키 객체

5.7.1. 키 객체 생성

암호모듈 운영시 비밀키, 인증키, 개인키, 공개키 등의 키 설정이 필요한 경우 MC_CreateObject 함수를 이용하여 키를 설정 할 수 있다.
각각의 키는 암호화 되어 있지 않은 키이며, 비밀키 및 인증키는 해당 알고리즘의 길이에 맞는 키를 설정하여야 하고, 개인키 및 공개키는 각 알고리즘 별 5.9 절의 데이터 구조에 따라 키를 구성한 후 설정하여야 한다.

```
void SetKey()
{
        MC_RV rv;
        MC_ALGORITHM mcAlg = {0,0,0}, mcKeyAlg = {0,0,0};
        MC_HOBJECT hPubkey = 0, hPrikey = 0;
        MC_HSESSION hSession = 0;
        MC_UCHAR data[4096], out[512], key[1024], param[1024], tmp[1024];
        MC_UINT datalen, outlen, keylen, paramlen, tmplen;
        MC_CHAR ch;

        // 세션 초기화
        rv = MC_OpenSession(&hSession);
```

```
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 1;}

        // 개인키 입력 (스트링)
        inputData("Private key [Gen]: ", key, keylen);
        // 스트링을 바이너리로 변환
        keylen = processData(key);

        // 개인키 설정
        MC_CreateObject(hSession, key, keylen, &hPrikey);

        // 개인키 버퍼 제로화
        memset(key, 0x55, keylen);
        memset(key, 0x00, keylen);

        // 암호서비스 이용
        ...

        // 키 제로화
        MC_DestroyObject(hSession, hPrikey);
        // 세션 종료
        MC_CloseSession(hSession);
}
```

암호화된 비밀키, 인증키, 개인키, 공개키 등의 키를 암호모듈에 주입하는 경우 MC_Unwrap 함수를 이용하여 키를 설정 할 수 있다. 암호화된 키를 사용하여 키 세션오브젝트 핸들을 반환한다.

```
void UnwrapKey()
{
        MC_RV rv = MC_OK;
        MC_ALGORITHM mcAlg = {MC_ALGID_SHA1WithSEED, NULL, 0};
        MC_HSESSION hSession = 0;
        MC_HOBJECT hWrappingKey = 0, hKey = 0;
        MC_UCHAR WrappedKey[1024];
        MC_INT nWrappedKey = 1024;

        // 세션 초기화
        rv = MC_OpenSession(&hSession);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 1;}

        // Unwrap 용 키 설정
        rv = MC_CreateObject(hSession, (MC_UCHAR*)"password", 8, &hWrappingKey);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 1;}

        // 키 Unwrap
        rv = MC_UnwrapKey(hSession, &mcAlg, hWrappingKey, WrappedKey,
nWrappedKey, &hKey);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 1;}

        // 키 핸들 제로화
```

```
        MC_DestroyObject(hSession, hWrappingKey);
        MC_DestroyObject(hSession, hKey);

        // 세션 종료
        MC_CloseSession(hSession);
}
```

5.7.2. 키 확인

암호모듈 운영시 비밀키, 인증키, 개인키, 공개키 등의 키에 대한 출력이 필요한 경우 MC_GetObjectValue 를 이용하여 키오브젝트 핸들로부터 키를 획득할 수 있다.

```
void GenKeyPair()
{
        MC_RV rv;
        MC_ALGORITHM mcAlg = {0,0,0};
        MC_HOBJECT hPubkey = 0, hPrikey = 0;
        MC_HSESSION hSession = 0;
        MC_UCHAR param[1024], tmp[2048];
        MC_UINT paramlen, tmplen;
        MC_CHAR ch;

        // 세션 초기화
        rv = MC_OpenSession(&hSession);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 1;}

        // 알고리즘 설정
        mcAlg.mcAlgId = MC_ALGID_RSA1024;

        // 개인키, 공개키쌍 생성
        rv = MC_GenerateKeyPair(hSession, &mcAlg, &hPubkey, &hPrikey);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); break;}

        // 개인키 출력
        // 할당된 출력버퍼에 키오브젝트 핸들의 데이터를 출력한다.
        // 출력 버퍼의 크기가 작을경우 에러를 발생한다.
        memset(tmp, 0x00, sizeof(tmp)); tmplen = sizeof(tmp);
        MC_GetObjectValue(hSession, hPrikey, tmp, &tmplen);
        printf("PrivateKey(%d) : ₩n", tmplen);
        printOutput(tmp, tmplen);

        // 공개키 출력
        memset(tmp, 0x00, sizeof(tmp)); tmplen = sizeof(tmp);
        MC_GetObjectValue(hSession, hPubkey, tmp, &tmplen);
        printf("PublicKey(%d) : ₩n", tmplen);
        printOutput(tmp, tmplen);
```

```
        // 개인키, 공개키 제로화
        MC_DestroyObject(hSession, hPubkey);
        MC_DestroyObject(hSession, hPrikey);
        // 세션 종료
        MC_CloseSession(hSession);
}
```

암호화된 비밀키, 인증키, 개인키, 공개키 등의 키를 암호모듈로부터 출력하는 경우 MC_Wrap 함수를 이용하여 키를 출력 할 수 있다. 키 세션오브젝트로부터 암호화된 키를 반환한다.

```
void WrapKey()
{
        MC_RV rv = MC_OK;
        MC_ALGORITHM mcAlg = {MC_ALGID_SHA1WithSEED, NULL, 0};
        MC_HSESSION hSession = 0;
        MC_HOBJECT hWrappingKey = 0, hKey = 0, hKey2 = 0;
        MC_UCHAR WrappedKey[1024];
        MC_INT nWrappedKey = 1024;

        // 세션 초기화
        rv = MC_OpenSession(&hSession);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 1;}

        // Wrap 하기 위한 암호 설정
        rv = MC_CreateObject(hSession, (MC_UCHAR*)"password", 8, &hWrappingKey);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 1;}

        // hKey 에 대하여 Wrap 키를 생성
        rv = MC_WrapKey(hSession, &mcAlg, hWrappingKey, hKey, WrappedKey,
&nWrappedKey);
        if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 1;}

        // 키 핸들 제로화
        MC_DestroyObject(hSession, hWrappingKey);
        MC_DestroyObject(hSession, hKey);

        // 세션 종료
        MC_CloseSession(hSession);
}
```

5.8. 난수 생성

입출력 데이터는 기본적으로 16 진수 바이너리 데이터를 이용하여 메시지와

해쉬값을 입출력하며 콘솔을 통하여 입력 받는 경우 16 진수로 변환해 주어야 한다.
알고리즘 파라메터를 이용하여 사용자의 추가적인 씨앗값을 입력 할 수 있다.

```
void GenRandom()
{
	MC_RV rv = MC_OK;
	MC_ALGORITHM mcAlg = {MC_ALGID_SHA256DRBG, NULL, 0};
	MC_HSESSION hSession = 0;
	MC_UCHAR Random[16][32];    /* 20 byte random */
	MC_INT RandNum = 16, RandBytes = 32;
	MC_INT i, t1 = 0, t2 = 0;

	// 세션 초기화
	rv = MC_OpenSession(&hSession);
	if(rv != MC_OK) {printf("%s₩n", MC_GetErrorString(rv)); return 1;}

	for(i=0; i<RandNum; i++)
	{
		// RandBytes 의 난수 생성
		rv = MC_GenerateRandom(hSession, &mcAlg, Random[i], RandBytes);
	}

	/* 중복 확인 */
	for(i=0; i<RandNum; i++)
	{
		printf("random%02d(%02d) : ₩n", i+1, RandBytes);
		printOutput(Random[i], RandBytes);
	}

	// 세션 종료
	MC_CloseSession(hSession);
}
```

5.9. 데이터 구조

RSA, KCDSA, ECDSA 등의 비대칭키 알고리즘은 개인키와 공개키의 두 종류의 키를 한 쌍으로 하며 MagicCrypto 암호모듈에서 사용하는 개인키, 공개키는 모두 ASN.1 형식으로 사용된다.
또한 RSA, KCDSA, ECDSA 전자서명 데이터 및 KCDSA 공개 파라메터도 ASN.1 형식으로 사용된다.

5.9.1. RSA 개인키

RSA 개인키는 아래와 같은 구조를 갖는다.

```
RSAPrivateKey ::= SEQUENCE {
        version Version,
        modulus INTEGER,                -- n
        publicExponent INTEGER,         -- e
        privateExponent INTEGER,        -- d
        prime1 INTEGER,                 -- p
        prime2 INTEGER,                 -- q
        exponent1 INTEGER,              -- d mod (p-1)
        exponent2 INTEGER,              -- d mod (q-1)
        coefficient INTEGER,            -- (inverse of q) mod p
}
```

5.9.2. RSA 공개키

RSA 공개키는 아래와 같은 구조를 갖는다.

```
RSAPublicKey ::= SEQUENCE {
        modulus INTEGER,                -- n
        publicExponent INTEGER          -- e
}
```

5.9.3. RSA 전자서명

```
RSA-Sig-Value ::= M^d (mod n)           -- Length = |modulus|
```

5.9.4. KCDSA 개인키

KCDSA 개인키는 아래와 같은 구조를 갖는다.

```
KCDSAPrivateKey ::= SEQUENCE {
        Version INTEGER,                -- version(0)
        P INTEGER,                      -- odd prime P=2JQ+ 1
        Q INTEGER,                      -- odd prime
        G INTEGER,                      -- generator of order Q
        Y INTEGER,                      -- public key
        X INTEGER                       -- private key
}
```

5.9.5. KCDSA 공개키

KCDSA 공개키는 아래와 같은 구조를 갖는다.

```
KCDSAPublicKey ::= INTEGER              -- public key Y
```

5.9.6. KCDSA 전자서명

KCDSA 전자서명 데이터는 아래와 같은 구조를 갖는다.

```
KCDSASignatureValue ::= SEQUENCE {
        r BIT STRING,
        s INTEGER
}
```

5.9.7. KCDSA 도메인 파라메터

KCDSA 도메인 파라미터는 아래와 같은 구조를 갖는다.

```
KCDSAParameter ::= SEQUENCE {
        P INTEGER,                      -- odd prime P=2JQ+ 1
        Q INTEGER,                      -- odd prime
        G INTEGER,                      -- generator of order Q
}
```

5.9.8. ECDSA 개인키

ECDSA 개인키는 아래와 같은 구조를 갖는다.

```
ECPrivateKey ::= SEQUENCE {
        Version INTEGER { ecPrivkeyVer1(1) }
        privateKey OCTET STRING,
}
```

5.9.9. ECDSA 공개키

ECDSA 공개키 아래와 같은 구조를 갖는다.

```
ECPublicKey ::= ECPoint                    -- Uncompressed form
```

5.9.10. ECDSA 전자서명

ECDSA 전자서명 데이터는 아래와 같은 구조를 갖는다.

```
ECDSA-Sig-Value ::= SEQUENCE {
        r INTEGER,
        s INTEGER
}
```

6. 암호모듈 상세 명세

6.1. MagicCrypto API 함수 명세

6.1.1. MC_GetVersion

암호모듈의 버전을 구한다. 버전은 major, minor, release 의 3 개 항목으로 구성되어있다. 버전이 표시하는 형식은 major.minor.release 와 같이 점(.)으로 구분하여 표시한다.

```
MCAPI MC_RV MC_GetVersion (
        MC_VERSION *pVersion
);
```

6.1.1.1. 파라메터

pVersion　　[출력] 암호모듈의 버전 구조체

6.1.1.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError 또는 MC_GetErrorStrng 을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.1.3. 예제

```
MC_VERSION mcVer;
MC_GetVersion(&mcVer);
printf("%s V%d.%d.%d ₩n", mcVer.name, mcVer.major, mcVer.minor, mcVer.release);
```

6.1.2. MC_GetStatus

암호모듈의 상태를 확인 한다.

```
MCAPI MC_RV MC_GetStatus (
        MC_UIMT *pFlag
);
```

6.1.2.1. 파라메터

pFlag [출력] 암호모듈의 상태

플래그 값	값	설명
MC_STAT_NONE	0x00000000	암호모듈이 시작 되지 않음
MC_STAT_INITIALIZED	0x10000000	암호모듈 초기화 완료 플래그
MC_STAT_FATAL	0x20000000	심각한 에러 상태 플래그

6.1.2.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError 또는 MC_GetErrorStrng 을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.2.3. 예제

```
MC_UINT flag;
MC_GetStatus(&flag);
printf(" MC_GetState rv = %d, flag = 0x%08x ₩n", rv, flag);

if (flag & MC_STAT_INITIALIZED) {
        printf(" MC State is normal . ₩n");
} else if (flag & MC_STAT_FATAL) {
```

```
        printf(" MC State is fatal error ! ₩n");
} else if (flag & MC_STAT_NONE) {
        printf(" MC State is not initialized. ₩n");
} else {
        printf(" MC State is unknown. ₩n");
}
```

6.1.3. MC_Initialize

암호모듈을 초기화한다. 전역환경변수 gpApi 의 메모리를 할당하고 초기화한다.
초기화에 성공하면 내부적으로 플래그를 설정하여 MC_OpenSession 시 초기화가 정상적으로 수행되었는지 검사한다.

```
MCAPI MC_RV MC_Initialize (
        MC_VOID *pInitArgs
);
```

6.1.3.1. 파라메터

pInitArgs　　[입력] 현재 사용하지 않는 변수, NULL 입력

6.1.3.2. 리턴값

성공 시 MC_OK, 실패 시 MC_FAIL.

6.1.3.3. 예제

```
MC_Initialize(NULL);
```

6.1.3.4. 참고

MC_Finalize

6.1.4. MC_Finalize

암호모듈을 종료한다. 암호모듈의 환경변수 pApi 의 메모리를 해제하고, 제로화 한다.

```
MCAPI MC_RV MC_Finalize ();
```

6.1.4.1. 파라메터

없음

6.1.4.2. 리턴값

성공시 MC_OK, 실패시 에러코드.

6.1.4.3. 예제

```
MC_Initialize(NULL);
…
MC_Finalize();
```

6.1.4.4. 참고

MC_Initialize

6.1.5. MC_Selftest

암호모듈의 암호 서비스에 대한 자가시험을 한다. 자가시험은 알고리즘시험, KAT 시험, 난수시험, 키쌍생성시험등을 포함한다.

```
MCAPI MC_RV MC_Selftest ();
```

6.1.5.1. 파라메터

없음.

6.1.5.2. 리턴값

성공시 MC_OK, 실패시 MC_FAIL.

6.1.5.3. 예제

```
MC_Selftest();
```

6.1.6. MC_GetErrorString

리턴값을 참조하여 에러 정보를 가져 온다. 리턴값은 스트링으로 반환되며 printf 등의 함수를 이용하여 가공할 수 있다.

```
MCAPI MC_STR MC_GetErrorString (
        MC_RV nRv
);
```

6.1.6.1. 파라메터

nRv [입력] 리턴 값, 에러코드

6.1.6.2. 리턴값

에러 스트링

6.1.6.3. 예제

```
MC_API *pApi = NULL;
MC_APIMODE mcApiMode = MC_MODE_NON_KCMV;
MC_Initialize(&pApi, NULL);
rv = MC_SetApiMode(pApi, mcApiMode);
```

```
if(rv != MC_OK) printf("ERROR: %s₩n", MC_GetErrorString(rv));
```

6.1.7. MC_OpenSession

암호모듈 초기화 이후, 암호 세션을 초기화 한다. 세션 환경변수의 메모리를 할당하고 이에 대한 핸들을 반환한다.
암호세션을 초기화 하면 암호모듈에서 제공하는 암호화, 전자서명 등의 암호서비스를 이용할 수 있는 상태가 된다.

```
MCAPI MC_RV MC_OpenSession (
        MC_HSESSION *phSession
);
```

6.1.7.1. 파라메터

phSession　[출력] 세션 핸들

6.1.7.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.7.3. 예제

```
MC_HSESSION hSession;
MC_Initialize(NULL);
MC_OpenSession(&hSession);
```

6.1.7.4. 참고

MC_Initialize, MC_CloseSession, MC_GetError, MC_GetErrorString

6.1.8. MC_CloseSession

세션을 종료한다. 세션 핸들과 연결된 세션 환경변수의 할당된 메모리를 해제하고, 제로화 한다.

```
MCAPI MC_RV MC_CloseSession (
        MC_HSESSION hSession
);
```

6.1.8.1. 파라메터

hSession　　[입력] 세션 핸들

6.1.8.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_ GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.8.3. 예제

```
MC_HSESSION hSession;
MC_Initialize(NULL);
MC_OpenSession(&hSession);
…
MC_CloseSession(hSession);
```

6.1.8.4. 참고

MC_Initialize, MC_OpenSession, MC_GetError, MC_ GetErrorString

6.1.9. MC_CopySession

세션을 복사한다.

```
MCAPI MC_RV MC_CopySession (
        MC_HSESSION hSession, MC_HSESSION *phSession
);
```

6.1.9.1. 파라메터

hSession [입력] 세션 핸들
phSession [출력] 복사 된 세션 핸들

6.1.9.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.9.3. 예제

```
MC_HSESSION hSession, hSession2;
MC_Initialize(NULL);
MC_OpenSession(&hSession);
MC_CopySession(hSession, &hSession2);
...
MC_CloseSession(hSession);
```

6.1.9.4. 참고

MC_Initialize, MC_OpenSession, MC_GetError, MC_ GetErrorString

6.1.10. MC_SetApiMode

암호모듈의 운영모드를 설정한다.
암호검증대상 알고리즘 사용모드와 모든 알고리즘을 사용할 수 있는 모드의 설정이 가능하다. 검증대상알고리즘을 사용하기 위하여 MC_MODE_KCMV를 설정하고, 비검증대상알고리즘을 포함한 모든 알고리즘을 사용하기 위해

서는 MC_MODE_NON_KCMV 를 설정한다.

```
MCAPI MC_RV MC_SetApiMode (
        MC_HSESSION hSession,
        MC_APIMODE mcApiMode
);
```

6.1.10.1. 파라메터

hSession [입력] 세션 핸들
mcApiMode [입력] 모드 (MC_MODE_KCMV, MC_MODE_NON_KCMV)

6.1.10.2. 리턴값

성공시 MC_OK, 실패시 에러코드.

6.1.10.3. 예제

```
MC_APIMODE mcApiMode = MC_MODE_NON_KCMV;
MC_Initialize(NULL);
MC_OpenSession(&hSession);
MC_SetApiMode(hSession, mcApiMode);
```

6.1.10.4. 참고

MC_OpenSession

6.1.11. MC_GetApiMode

암호모듈의 현재의 운영모드를 확인한다. 함수의 출력 파라메터를 통해 검증모드일 경우 MC_MODE_KCMV, 비검증모드 일 경우 MC_MODE_NON_KCMV 를 반환한다.

```
MCAPI MC_RV MC_SetApiMode (
        MC_HSESSION hSession,
        MC_APIMODE *pmcApiMode
```

```
);
```

6.1.11.1. 파라메터

hSession [입력] 세션 핸들
pmcApiMode [출력] 모드 (MC_MODE_KCMV, MC_MODE_NON_KCMV)

6.1.11.2. 리턴값

성공시 MC_OK, 실패시 에러코드.

6.1.11.3. 예제

```
MC_APIMODE mcApiMode;
MC_Initialize(NULL);
MC_OpenSession(&hSession);
MC_GetApiMode(hSession, &mcApiMode);
```

6.1.11.4. 참고

MC_OpenSession

6.1.12. MC_SetLog

암호모듈의 로그 정보를 기록할 수 있는 로그 함수, 레벨 및 로그 파일의 정보를 설정한다.
로그 함수는 사용자가 직접 작성해야 한다. 로그함수의 프로토타입은 mcapi_type.h 에서 찾을 수 있다. 로그레벨은 기본값 0 일 때 암호모듈의 로그정보를 표시하지 않으며, 3 일때에는 모든 로그정보를 표시하게 된다.

```
MCAPI MC_RV MC_SetLog (
        MC_CALLBACK cbLog,
        MC_UINT nLevel,
        MC_STR pszLogPath
);
```

6.1.12.1. 파라메터

cbLog [입력] 사용자 콜백 로그 함수, 예제 참조
nLevel [입력] 디버깅 레벨 (0~3)
pszLogPath [입력] 로그를 출력할 파일 경로

6.1.12.2. 리턴값

성공시 MC_OK, 실패시 에러코드.

6.1.12.3. 예제

```
/* path : 로그파일경로, fmt : 출력형식지정 */
extern int cbLog(const char *path, const char *fmt, ...);
...
MC_Initialize(NULL);
MC_SetLog(cbLog, 2, "./mcapi.log");
...
```

6.1.12.4. 참고

MC_Initialize

6.1.13. MC_GetError

세션 핸들로부터 에러 정보를 가져 온다.

```
MCAPI MC_STR MC_GetError (
        MC_HSESSION hSession
);
```

6.1.13.1. 파라메터

hSession [입력] 세션 핸들

6.1.13.2. 리턴값

에러 스트링

6.1.13.3. 예제

```
rv = MC_DigestInit(hSession, &mcAlg);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.13.4. 참고

MC_OpenSession

6.1.14. MC_GetErrorStack

에러가 발생했을 경우 해당 세션의 세션 핸들로부터 에러가 난 곳의 함수 스택, 줄번호, 에러내용을 가져온다.

```
MCAPI MC_STR MC_GetErrorStack (
        MC_HSESSION hSession
);
```

6.1.14.1. 파라메터

hSession [입력] 세션 핸들

6.1.14.2. 리턴값

에러 위치를 나타내는 문자열 (에러 파일, 함수, 줄번호)

6.1.14.3. 예제

```
rv = MC_DigestInit(hSession, &mcAlg);
if(rv != MC_OK) printf("ERROR: %s₩n", MC_GetErrorStack(hSession));
```

6.1.14.4. 참고

MC_OpenSession

6.1.15. MC_CreateObject

사용자의 데이터를 이용하여, 암호 세션의 오브젝트를 생성한다.

```
MCAPI MC_RV MC_CreateObject (
        MC_HSESSION hSession,
        MC_UCHAR *pData,
        MC_UINT nLength,
        MC_HOBJECT *phObject
);
```

6.1.15.1. 파라메터

hSession	[입력] 세션 핸들
pData	[입력] 데이터 포인터
nLength	[입력] 데이터 길이
phObject	[출력] 오브젝트 핸들

6.1.15.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.15.3. 예제

```
MC_HSESSION hSession;
MC_HOBJECT hKey;
MC_UCHAR data[10] = "a1b2c3d4";
MC_OpenSession(pApi,&hSession);
MC_CreateObject(hSession, data, strlen(data), &hKey);
```

6.1.15.4. 참고

MC_OpenSession, MC_DestroyObject, MC_GetObjectValue

6.1.16. MC_DestroyObject

암호 세션에서 생성된 세션 오브젝트를 제거한다. 메모리를 해제하고 제로화를 수행한다.

```
MCAPI MC_RV MC_DestroyObject (
        MC_HSESSION hSession,
        MC_HOBJECT hObject
);
```

6.1.16.1. 파라메터

hSession [입력] 세션 핸들
hObject [입력] 오브젝트 핸들

6.1.16.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.16.3. 예제

```
MC_HSESSION hSession;
MC_HOBJECT hKey;
MC_UCHAR data[10] = "a1b2c3d4";
MC_OpenSession(pApi,&hSession);
MC_CreateObject(hSession, data, strlen(data), &hKey);
…
MC_DestroyObject(hSession, hKey);
```

6.1.16.4. 참고

MC_OpenSession, MC_CreateObject, MC_GetObjectValue

6.1.17. MC_GetObjectValue

세션 오브젝트의 값을 가져온다. 이때, 값이 저장되는 pData 는 pnLength 를 최대크기로 메모리가 할당 된것으로 간주하며, 이때 할당된 메모리 크기인 pnLength 를 초기값으로 하여 pData 의 크기를 반환한다.

```
MCAPI MC_RV MC_GetObjectValue (
        MC_HSESSION hSession,
        MC_HOBJECT hObject,
        MC_UCHAR *pData,
        MC_UINT *pnLength
);
```

6.1.17.1. 파라메터

hSession	[입력] 세션 핸들
hObject	[입력] 오브젝트 핸들
pData	[출력] 데이터 포인터
pnLength	[출력] 데이터 길이

6.1.17.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.17.3. 예제

```
MC_UCHAR data[1024];
MC_UINT len = sizeof(data);
memset(data, 0x00, sizeof(data));
…
```

```
MC_GetObjectValue(hSession, hPubkey, data, &len);
```

6.1.17.4. 참고

MC_OpenSession, MC_CreateObject

6.1.18. MC_SignInit

전자서명 생성을 초기화 한다. 세션 정보를 초기화 하고, 알고리즘과 개인키를 설정한다. 개인키는 사용자가 쉽게 변경할 수 없는 세션 오브젝트핸들로 설정하며, 개인키의 형식은 각 알고리즘에 따른 ASN.1 표기 형식을 따르며, 필요시 알고리즘의 파라메터를 설정한다.

```
MCAPI MC_RV MC_SignInit (
        MC_HSESSION hSession,
        MC_ALGORITHM *pAlg,
        MC_HOBJECT hPriKey
);
```

6.1.18.1. 파라메터

hSession	[입력] 세션 핸들
pAlg	[입력] 알고리즘 포인터
hPriKey	[입력] 개인키 오브젝트 핸들

6.1.18.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.18.3. 예제

```
MC_ALGORITHM mcAlg = {MC_ALGID_SHA1WithRSAPSS, NULL, 0};
MC_OpenSession(pApi, &hSession);
```

```
rv = MC_SignInit(hSession, &mcAlg, hPrikey);
if(rv != MC_OK) printf("ERROR: %s (%d)Wn", MC_GetErrorString(rv), rv);
```

6.1.18.4. 참고

MC_OpenSession, MC_Sign, MC_SignUpdate

6.1.19. MC_Sign

단일 메시지에 대한 전자서명을 생성한다. 메시지 pData 에 대한 전자서명을 수행하고 pSign 으로 전자서명의 결과를 표시한다. 전자서명은 각 알고리즘에 따라 ASN.1 형식으로 표시된다. pnSignLen 은 pSign 데이터의 최대 메모리 할당 크기를 초기값으로 하여 전자서명이 완료된 후에 전자서명 데이터의 길이를 반환한다.
전자서명 생성 종료시 세션 환경변수가 참조하는 메모리는 제로화 된다.

```
MCAPI MC_RV MC_Sign (
        MC_HSESSION hSession,
        MC_UCHAR *pData,
        MC_UINT nDataLen,
        MC_UCHAR *pSign,
        MC_UINT *pnSignLen
);
```

6.1.19.1. 파라메터

hSession	[입력] 세션 핸들
pData	[입력] 데이터 포인터
nDataLen	[입력] 데이터 길이
pSign	[출력] 전자서명 데이터 포인터
pnSignLen	[출력] 전자서명 데이터 길이

6.1.19.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내

용을 확인할 수 있다.

6.1.19.3. 예제

```
MC_UCHAR *data, *sign;
MC_UINT dataLen, signLen;
…
MC_OpenSession(pApi, &hSession);
MC_SignInit(hSession, &mcAlg, hPrikey);
signLen = 1024; /* maximum size of sign */
rv = MC_Sign(hSession, data, dataLen, sign, &signLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
…
MC_CloseSession(hSession);
```

6.1.19.4. 참고

MC_SignInit, MC_CloseSession

6.1.20. MC_SignUpdate

메시지를 한번에 입력하기 어려운 경우 또는 메시지가 여러 부분으로 구성되어 있는 경우, 전자서명을 생성하기 위한 메시지의 일부를 입력한다.

```
MCAPI MC_RV MC_SignUpdate (
        MC_HSESSION hSession,
        MC_UCHAR *pData,
        MC_UINT nDataLen
);
```

6.1.20.1. 파라메터

hSession	[입력] 세션 핸들
pData	[입력] 데이터 포인터
nDataLen	[입력] 데이터 길이

6.1.20.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.20.3. 예제

```
MC_UCHAR *data;
MC_UINT dataLen;
…
MC_OpenSession(pApi, &hSession);
MC_SignInit(hSession, &mcAlg, hPrikey);
rv = MC_SignUpdate(hSession, data, dataLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.20.4. 참고

MC_SignInit, MC_SignFinal

6.1.21. MC_SignFinal

MC_SignUpdate 를 통하여 입력받은 메시지에 대한 전자서명을 생성한다. pnSignLen 은 pSign 의 최대 메모리 할당 크기를 초기값으로 하여 서명생성이 종료된 후 서명 데이터의 크기를 반환한다. 반환되는 데이터의 크기가 할당된 메모리의 크기보다 크면 에러를 리턴하게 된다.
전자서명 생성 종료시 세션 환경변수가 참조하는 메모리는 제로화 된다.

```
MCAPI MC_RV MC_SignFinal (
        MC_HSESSION hSession,
        MC_UCHAR *pSign,
        MC_UINT *pnSignLen
);
```

6.1.21.1. 파라메터

hSession	[입력] 세션 핸들
pSign	[출력] 전자서명 데이터 포인터
pnSignLen	[출력] 전자서명 데이터 길이

6.1.21.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.21.3. 예제

```
MC_UCHAR *data, *sign;
MC_UINT dataLen, signLen;
...
MC_OpenSession(pApi, &hSession);
MC_SignInit(hSession, &mcAlg, hPrikey);
signLen = 1024; /* maximum size of sign */
MC_SignUpdate(hSession, data, dataLen);
rv = MC_SignFinal(hSession, sign, &signLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
...
MC_CloseSession(hSession);
```

6.1.21.4. 참고

MC_SignUpdate, MC_CloseSession

6.1.22. MC_VerifyInit

전자서명 검증을 초기화 한다. 세션 정보를 초기화 하고, 알고리즘과 공개키를 설정한다. 공개키는 알고리즘에 따라 ASN.1 형식으로 표현되며, 오브

젝트핸들을 통하여 관리한다.

```
MCAPI MC_RV MC_VerifyInit (
        MC_HSESSION hSession,
        MC_ALGORITHM *pAlg,
        MC_HOBJECT hPubKey
);
```

6.1.22.1. 파라메터

hSession	[입력] 세션 핸들
pAlg	[입력] 알고리즘 포인터
hPubKey	[입력] 공개키 오브젝트 핸들

6.1.22.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.22.3. 예제

```
MC_ALGORITHM mcAlg = {MC_ALGID_SHA1WithRSAPSS, NULL, 0};
MC_OpenSession(pApi, &hSession);
rv = MC_VerifyInit(hSession, &mcAlg, hPubkey);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.22.4. 참고

MC_OpenSession, MC_Verify, MC_VerifyUpdate

6.1.23. MC_Verify

단일 메시지에 대한 전자서명을 검증한다. 메시지와 서명값을 입력하면 초기화 단계에서 입력된 공개키 연산을 수행하여 전자서명 검증 결과를 반환

한다.
전자서명 검증 종료시 세션 환경변수가 참조하는 메모리는 제로화 된다.

```
MCAPI MC_RV MC_Verify (
        MC_HSESSION hSession,
        MC_UCHAR *pData,
        MC_UINT nDataLen,
        MC_UCHAR *pSign,
        MC_UINT nSignLen
);
```

6.1.23.1. 파라메터

hSession	[입력] 세션 핸들
pData	[입력] 데이터 포인터
nDataLen	[입력] 데이터 길이
pSign	[입력] 전자서명 데이터 포인터
nSignLen	[입력] 전자서명 데이터 길이

6.1.23.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.23.3. 예제

```
MC_UCHAR *data, *sign;
MC_UINT dataLen, signLen;
…
MC_OpenSession(pApi, &hSession);
MC_VerifyInit(hSession, &mcAlg, hPubkey);
rv = MC_Verify(hSession, data, dataLen, sign, signLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
…
MC_CloseSession(hSession);
```

6.1.23.4. 참고

MC_VerifyInit, MC_CloseSession

6.1.24. MC_VerifyUpdate

메시지를 한번에 입력하기 어려운 경우 또는 메시지가 여러 부분으로 구성되어 있는 경우, 전자서명을 검증하기 위한 메시지의 일부를 입력한다.

```
MCAPI MC_RV MC_VerifyUpdate (
        MC_HSESSION hSession,
        MC_UCHAR *pData,
        MC_UINT nDataLen
);
```

6.1.24.1. 파라메터

hSession	[입력] 세션 핸들
pData	[입력] 데이터 포인터
nDataLen	[입력] 데이터 길이

6.1.24.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 에러코드와 내용을 확인할 수 있다.

6.1.24.3. 예제

```
MC_UCHAR *data;
MC_UINT dataLen;
...
MC_OpenSession(pApi, &hSession);
MC_VerifyInit(hSession, &mcAlg, hPubkey);
rv = MC_VerifyUpdate(hSession, data, dataLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.24.4. 참고

MC_VerifyInit, MC_VerifyFinal

6.1.25. MC_VerifyFinal

MC_VerifyUpdate 를 통하여 입력받은 메시지와 전자서명값을 입력받아 전자서명을 검증한다.
전자서명 검증 종료시 세션 환경변수가 참조하는 메모리는 제로화 된다.

```
MCAPI MC_RV MC_VerifyFinal (
        MC_HSESSION hSession,
        MC_UCHAR *pSign,
        MC_UINT nSignLen
);
```

6.1.25.1. 파라메터

hSession	[입력] 세션 핸들
pSign	[입력] 전자서명 데이터 포인터
nSignLen	[입력] 전자서명 데이터 길이

6.1.25.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.25.3. 예제

```
MC_UCHAR *data, *sign;
MC_UINT dataLen, signLen;
...
MC_OpenSession(pApi, &hSession);
```

```
MC_VerifyInit(hSession, &mcAlg, hPubkey);
MC_VerifyUpdate(hSession, data, dataLen);
rv = MC_VerifyFinal(hSession, sign, signLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
...
MC_CloseSession(hSession);
```

6.1.25.4. 참고

MC_VerifyUpdate, MC_CloseSession

6.1.26. MC_SetOption

대칭키 알고리즘의 운영모드와 패딩타입을 설정한다. 대칭키 운영모드는 ECB/CBC/CTR 모드, 메시지 인증을 위한 GCM/CCM 모드를 지원하며 추가적인 모드에 대한 확장이 가능하다. 메시지 패딩타입은 PKCS#5 와 SSL 패딩, 1 과 1 이후에 0 을 대칭키 블록 크기에 맞게 패딩하는 ONE_PADDING 을 지원하며 향후 확장이 용이하게 설계되어 있다.

```
MCAPI MC_RV MC_SetOption (
        MC_HSESSION hSession,
        MC_ALGMODE mcAlgMode,
        MC_PADTYPE mcPadType
);
```

6.1.26.1. 파라메터

hSession [입력] 세션 핸들
mcAlgMode [입력] 대칭키 알고리즘 운영모드
mcPadType [입력] 패딩 타입

6.1.26.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내

용을 확인할 수 있다.

6.1.26.3. 예제

```
MC_ALGMODE mode = MC_ALGMODE_CBC;
MC_PADTYPE pad = MC_PADTYPE_PKCS5;
rv = MC_SetOption(hSession, mode, pad);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.26.4. 참고

MC_OpenSession, MC_Encrypt, MC_EncryptFinal, MC_Decrypt, MC_DecryptFinal

6.1.27. MC_EncryptInit

암호화를 초기화 한다. 세션 정보를 초기화 하고, 알고리즘과 비밀키를 설정한다. 초기벡터를 필요로 하는 암호화 운영모드의 경우, 초기벡터는 알고리즘 파라메터를 이용하여 설정한다.

```
MCAPI MC_RV MC_EncryptInit (
        MC_HSESSION hSession,
        MC_ALGORITHM *pAlg,
        MC_HOBJECT hKey
);
```

6.1.27.1. 파라메터

hSession	[입력] 세션 핸들
pAlg	[입력] 알고리즘 포인터
hKey	[입력] 비밀키 오브젝트 핸들

6.1.27.2. 리턴값

성공시 MC_OK, 실패시 에러코드.

실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.27.3. 예제

```
MC_ALGORITHM mcAlg = {MC_ALGID_SEED, NULL, 0};
MC_ALGORITHM mcPrng = {MC_ALGID_ANSIX962RNG, NULL, 0};
MC_HOBJECT hKey;
MC_UCHAR iv[16];
MC_GenerateKey(hSession, &mcAlg, &hKey);
MC_GenerateRandom(hSession, &mcPrng, iv, sizeof(iv));
mcAlg.pParam = iv;
mcAlg.nParam = sizeof(iv);
rv = MC_EncryptInit(hSession, &mcAlg, hKey);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.27.4. 참고

MC_OpenSession, MC_GenerateKey, MC_GenerateRandom, MC_Encrypt

6.1.28. MC_Encrypt

단일 메시지에 대한 암호화를 수행한다. pnCipherLen 는 pCipher 의 최대 메모리 할당 크기로 암호문의 크기가 할당된 메모리 크기보다 큰 경우 에러를 발생한다. 암호화를 수행하고 pnCipherLen 는 암호문의 길이를 반환한다.
암호화 종료시 세션 환경변수가 참조하는 메모리는 제로화 된다.

```
MCAPI MC_RV MC_Encrypt (
        MC_HSESSION hSession,
        MC_UCHAR *pData,
        MC_UINT nDataLen,
        MC_UCHAR *pCipher,
        MC_UINT *pnCipherLen
);
```

6.1.28.1. 파라메터

hSession	[입력] 세션 핸들
pData	[입력] 데이터 포인터
nDataLen	[입력] 데이터 길이
pCipher	[출력] 암호문의 포인터
pnCipherLen	[출력] 암호문 길이

6.1.28.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.28.3. 예제

```
MC_UCHAR *data, *cipher;
MC_UINT dataLen, cipherLen;
...
MC_EncryptInit(hSession, &mcAlg, hKey);
cipherLen = 1024; /* maximum size of cipher */
rv = MC_Encrypt(hSession, data, dataLen, cipher, &cipherLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
...
MC_CloseSession(hSession);
```

6.1.28.4. 참고

MC_EncryptInit, MC_CloseSession

6.1.29. MC_EncryptUpdate

메시지를 한번에 입력하기 어려운 경우 또는 메시지가 여러 부분으로 구성되어 있는 경우, 암호화를 하기 위한 메시지의 일부를 입력한다.

pnCipherLen 는 pCipher 의 최대 메모리 할당 크기를 입력받아 암호문의 길이를 반환한다.

```
MCAPI MC_RV MC_EncryptUpdate (
        MC_HSESSION hSession,
        MC_UCHAR *pData,
        MC_UINT nDataLen,
        MC_UCHAR *pCipher,
        MC_UINT *pnCipherLen
);
```

6.1.29.1. 파라메터

hSession [입력] 세션 핸들
pData [입력] 데이터 포인터
nDataLen [입력] 데이터 길이
pCipher [출력] 암호문의 포인터
pnCipherLen [출력] 암호문 길이

6.1.29.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.29.3. 예제

```
MC_UCHAR *data, *cipher;
MC_UINT dataLen, cipherLen;
...
MC_EncryptInit(hSession, &mcAlg, hKey);
cipherLen = 1024; /* maximum size of cipher */
rv = MC_EncryptUpdate(hSession, data, dataLen, cipher, &cipherLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.29.4. 참고

MC_EncryptInit, MC_EncryptFinal

6.1.30. MC_EncryptFinal

메지지의 마지막 부분에 대한 암호화를 수행하고 암호화를 종료한다. 암호화 종료시 세션 환경변수가 참조하는 메모리는 제로화 된다.
pnCipherLen 는 pCipher 의 최대 메모리 할당 크기를 입력받아 암호문의 길이를 반환한다.
암호화 종료시 세션 환경변수가 참조하는 메모리는 제로화 된다.

```
MCAPI MC_RV MC_EncryptFinal (
        MC_HSESSION hSession,
        MC_UCHAR *pCipher,
        MC_UINT *pnCipherLen
);
```

6.1.30.1. 파라메터

hSession [입력] 세션 핸들
pCipher [출력] 암호문의 포인터
pnCipherLen [출력] 암호문 길이

6.1.30.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.30.3. 예제

```
MC_UCHAR *data, *cipher;
MC_UINT dataLen, cipherLen;
...
MC_EncryptInit(hSession, &mcAlg, hKey);
```

```
MC_EncryptUpdate(hSession, data, dataLen, cipher, &cipherLen);
cipherLen = 1024; /* maximum size of cipher */
rv = MC_EncryptFinal(hSession, cipher, &cipherLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
...
MC_CloseSession(hSession);
```

6.1.30.4. 참고

MC_EncryptUpdate, MC_CloseSession

6.1.31. MC_DecryptInit

복호화를 초기화 한다. 세션 정보를 초기화 하고, 알고리즘과 비밀키를 설정한다. 초기벡터를 필요로 하는 암호화 운영모드의 경우, 초기벡터는 알고리즘 파라메터를 이용하여 설정한다.

```
MCAPI MC_RV MC_DecryptInit (
        MC_HSESSION hSession,
        MC_ALGORITHM *pAlg,
        MC_HOBJECT hKey
);
```

6.1.31.1. 파라메터

hSession	[입력] 세션 핸들
pAlg	[입력] 알고리즘 포인터
hKey	[입력] 비밀키 오브젝트 핸들

6.1.31.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.31.3. 예제

```
MC_ALGORITHM mcAlg = {MC_ALGID_SEED, NULL, 0};
MC_HOBJECT hKey;
MC_UCHAR iv[16];
MC_CreateObject(hSession, key, keyLen, &hKey);
mcAlg.pParam = iv;
mcAlg.nParam = sizeof(iv);
rv = MC_DecryptInit(hSession, &mcAlg, hKey);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.31.4. 참고

MC_OpenSession, MC_Decrypt

6.1.32. MC_Decrypt

단일 메시지에 대한 복호화를 수행한다. pnDataLen 는 pData 의 최대 메모리 할당 크기로 복호문의 크기가 할당된 메모리 크기보다 큰 경우 에러를 발생한다. 복호화를 수행하고 pnDataLen 는 복호문의 길이를 반환한다.
복호화 종료시 세션 환경변수가 참조하는 메모리는 제로화 된다.

```
MCAPI MC_RV MC_Decrypt (
        MC_HSESSION hSession,
        MC_UCHAR *pCipher,
        MC_UINT nCipherLen,
        MC_UCHAR *pData,
        MC_UINT *pnDataLen
);
```

6.1.32.1. 파라메터

hSession	[입력] 세션 핸들
pCipher	[입력] 암호문의 포인터
pnCipherLen	[입력] 암호문 길이
pData	[출력] 복호문 포인터

nDataLen [출력] 복호문 길이

6.1.32.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.32.3. 예제

```
MC_UCHAR *data, *cipher;
MC_UINT dataLen, cipherLen;
...
MC_DecryptInit(hSession, &mcAlg, hKey);
dataLen = 1024; /* maximum size of data */
rv = MC_Decrypt(hSession, cipher, cipherLen, data, &dataLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
...
MC_CloseSession(hSession);
```

6.1.32.4. 참고

MC_DecryptInit, MC_CloseSession

6.1.33. MC_DecryptUpdate

메시지를 한번에 입력하기 어려운 경우 또는 메시지가 여러 부분으로 구성되어 있는 경우, 복호화를 하기 위한 암호문의 일부를 입력한다. pnDataLen 은 pData 의 최대 메모리 할당 크기를 입력받아 복호문의 길이를 반환한다.

```
MCAPI MC_RV MC_DecryptUpdate (
        MC_HSESSION hSession,
        MC_UCHAR *pCipher,
        MC_UINT nCipherLen,
```

```
        MC_UCHAR *pData,
        MC_UINT *pnDataLen
);
```

6.1.33.1. 파라메터

hSession	[입력] 세션 핸들
pCipher	[입력] 암호문의 포인터
pnCipherLen	[입력] 암호문 길이
pData	[출력] 복호문 포인터
nDataLen	[출력] 복호문 길이

6.1.33.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.33.3. 예제

```
MC_UCHAR *data, *cipher;
MC_UINT dataLen, cipherLen;
...
MC_DecryptInit(hSession, &mcAlg, hKey);
dataLen = 1024; /* maximum size of data */
rv = MC_DecryptUpdate(hSession, cipher, cipherLen, data, &dataLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.33.4. 참고

MC_DecryptInit, MC_DecryptFinal

6.1.34. MC_DecryptFinal

메지지의 마지막 부분에 대한 복호화를 수행하고 복호화를 종료한다. 복호

화 종료시 세션 환경변수가 참조하는 메모리는 제로화 된다.
pnDataLen 은 pData 의 최대 메모리 할당 크기를 입력받아 복호문의 길이를 반환한다.
복호화 종료시 세션 환경변수가 참조하는 메모리는 제로화 된다.

```
MCAPI MC_RV MC_DecryptFinal (
        MC_HSESSION hSession,
        MC_UCHAR *pData,
        MC_UINT *pnDataLen
);
```

6.1.34.1. 파라메터

hSession [입력] 세션 핸들
pData [출력] 복호문 포인터
nDataLen [출력] 복호문 길이

6.1.34.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.34.3. 예제

```
MC_UCHAR *data, *cipher;
MC_UINT dataLen, cipherLen;
...
MC_DecryptInit(hSession, &mcAlg, hKey);
MC_DecryptUpdate(hSession, cipher, cipherLen, data, &dataLen);
dataLen = 1024; /* maximum size of data */
rv = MC_DecryptFinal(hSession, data, &dataLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
...
MC_CloseSession(hSession);
```

6.1.34.4. 참고

MC_DecryptUpdate, MC_CloseSession

6.1.35. MC_DigestInit

해쉬를 초기화 한다. 세션 정보를 초기화 하고, 알고리즘을 설정한다.

```
MCAPI MC_RV MC_DigestInit (
        MC_HSESSION hSession,
        MC_ALGORITHM *pAlg
);
```

6.1.35.1. 파라메터

hSession [입력] 세션 핸들
pAlg [입력] 알고리즘 포인터

6.1.35.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.35.3. 예제

```
MC_ALGORITHM mcAlg = {MC_ALGID_SHA1, NULL, 0};
MC_HSESSION hSession;
MC_OpenSession(pApi, &hSession);
rv = MC_DigestInit(hSession, &mcAlg);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.35.4. 참고

MC_OpenSession, MC_Digest

6.1.36. MC_Digest

단일 메시지에 대한 해쉬값을 구한다. pnDigestLen 은 pnDigest 의 최대 메모리 할당 크기를 입력받아 해쉬값의 길이를 반환한다.
메시지 해쉬 생성 종료시 세션 환경변수가 참조하는 메모리는 제로화 된다.

```
MCAPI MC_RV MC_Digest (
        MC_HSESSION hSession,
        MC_UCHAR *pData,
        MC_UINT nDataLen,
        MC_UCHAR *pDigest,
        MC_UINT *pnDigestLen
);
```

6.1.36.1. 파라메터

hSession	[입력] 세션 핸들
pData	[입력] 데이터 포인터
nDataLen	[입력] 데이터 길이
pDigest	[출력] 메시지 해쉬 포인터
pnDigestLen	[출력] 메시지 해쉬 길이

6.1.36.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.36.3. 예제

```
MC_UCHAR *data, *digest;
MC_UINT dataLen, digestLen;
...
MC_DigestInit(hSession, &mcAlg);
digestLen = 20; /* maximum size of digest */
```

```
rv = MC_Digest(hSession, data, dataLen, digest, &digestLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
...
MC_CloseSession(hSession);
```

6.1.36.4. 참고

MC_DigestInit, MC_CloseSession

6.1.37. MC_DigestUpdate

메시지를 한번에 입력하기 어려운 경우 또는 메시지가 여러 부분으로 구성되어 있는 경우, 해쉬값을 구하기 위한 메시지의 일부를 입력한다.

```
MCAPI MC_RV MC_DigestUpdate (
        MC_HSESSION hSession,
        MC_UCHAR *pData,
        MC_UINT nDataLen
);
```

6.1.37.1. 파라메터

hSession	[입력] 세션 핸들
pData	[입력] 데이터 포인터
nDataLen	[입력] 데이터 길이

6.1.37.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.37.3. 예제

```
MC_UCHAR *data;
```

```
MC_UINT dataLen;
...
MC_DigestInit(hSession, &mcAlg);
rv = MC_DigestUpdate(hSession, data, dataLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.37.4. 참고

MC_DigestInit, MC_DigestFinal

6.1.38. MC_DigestFinal

해쉬를 종료한다. pnDigestLen 은 pDigest 의 최대 메모리 할당 크기를 입력받아 해쉬값의 길이를 반환한다.
해쉬 종료시 세션 환경변수가 참조하는 메모리는 제로화 된다.

```
MCAPI MC_RV MC_DigestFinal (
        MC_HSESSION hSession,
        MC_UCHAR *pDigest,
        MC_UINT *pnDigestLen
);
```

6.1.38.1. 파라메터

hSession	[입력] 세션 핸들
pDigest	[출력] 메시지 해쉬 포인터
pnDigestLen	[출력] 메시지 해쉬 길이

6.1.38.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.38.3. 예제

```
MC_UCHAR *data, *digest;
MC_UINT dataLen, digestLen;
...
MC_DigestInit(hSession, &mcAlg);
MC_DigestUpdate(hSession, data, dataLen);
...
digestLen = 20; /* maximum size of digest */
rv = MC_DigestFinal(hSession, digest, &digestLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
...
MC_CloseSession(hSession);
```

6.1.38.4. 참고

MC_DigestUpdate, MC_CloseSession

6.1.39. MC_CreateMacInit

메시지 무결성값 생성를 초기화 한다. 세션 정보를 초기화 하고, 알고리즘과 비밀키를 설정한다.

```
MCAPI MC_RV MC_CreateMacInit (
        MC_HSESSION hSession,
        MC_ALGORITHM *pAlg,
        MC_HOBJECT hKey
);
```

6.1.39.1. 파라메터

hSession	[입력] 세션 핸들
pAlg	[입력] 알고리즘 포인터
hKey	[입력] 키 오브젝트 핸들

6.1.39.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.39.3. 예제

```
MC_ALGORITHM mcAlg = {MC_ALGID_SHA1_HMAC, NULL, 0};
MC_HOBJECT hKey;
MC_GenerateKey(hSession, &mcAlg, &hKey);
rv = MC_CreateMactInit(hSession, &mcAlg, hKey);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.39.4. 참고

MC_OpenSession, MC_GenerateKey, MC_GenerateRandom, MC_CreateObject, MC_CreateMac, MC_CreateMacUpdate

6.1.40. MC_CreateMac

단일 메시지에 대한 무결성값을 생성한다. pnMacLen 은 pMac 의 최대 메모리 할당 크기를 입력받아 무결성값의 길이를 반환한다.
메시지 인증값 생성 종료시 세션 환경변수가 참조하는 메모리는 제로화 된다.

```
MCAPI MC_RV MC_CreateMac (
        MC_HSESSION hSession,
        MC_UCHAR *pData,
        MC_UINT nDataLen,
        MC_UCHAR *pMac,
        MC_UINT *pnMacLen
);
```

6.1.40.1. 파라메터

hSession	[입력] 세션 핸들
pData	[입력] 데이터 포인터
nDataLen	[입력] 데이터 길이
pMac	[출력] 메시지 무결성값 포인터
pnMacLen	[출력] 메시지 무결성값 길이

6.1.40.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.40.3. 예제

```
MC_UCHAR *data, *mac;
MC_UINT dataLen, macLen;
MC_ALGORITHM mcAlg = {MC_ALGID_SHA1_HMAC, NULL, 0};
MC_HOBJECT hKey;
…
MC_GenerateKey(hSession, &mcAlg, &hKey);
MC_CreateMactInit(hSession, &mcAlg, hKey);
macLen = 20; /* maximum size of mac */
rv = MC_CreateMac(hSession, data, dataLen, mac, &macLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
…
MC_CloseSession(hSession);
```

6.1.40.4. 참고

MC_CreateMacInit, MC_CloseSession

6.1.41. MC_CreateMacUpdate

메시지를 한번에 입력하기 어려운 경우 또는 메시지가 여러 부분으로 구성되어 있는 경우, 메시지 무결성값을 생성하기 위한 메시지의 일부를 입력한다.

```
MCAPI MC_RV MC_CreateMacUpdate (
        MC_HSESSION hSession,
        MC_UCHAR *pData,
        MC_UINT nDataLen
);
```

6.1.41.1. 파라메터

hSession	[입력] 세션 핸들
pData	[입력] 데이터 포인터
nDataLen	[입력] 데이터 길이

6.1.41.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.41.3. 예제

```
MC_UCHAR *data;
MC_UINT dataLen;
…
MC_CreateMacInit(hSession, &mcAlg, hKey);
rv = MC_CreateMacUpdate(hSession, data, dataLen);
if(rv != MC_OK) printf("ERROR: %s (%d)Wn", MC_GetErrorString(rv), rv);
```

6.1.41.4. 참고

MC_CreateMacInit, MC_CreateMacFinal

6.1.42. MC_CreateMacFinal

무결성값 생성을 종료한다. pnMacLen 은 pMac 의 최대 메모리 할당 크기를 입력받아 해쉬값의 길이를 반환한다.
메시지 인증값 생성 종료시 세션 환경변수가 참조하는 메모리는 제로화 된다.

```
MCAPI MC_RV MC_CreateMacFinal (
        MC_HSESSION hSession,
        MC_UCHAR *pMac,
        MC_UINT *pnMacLen
);
```

6.1.42.1. 파라메터

hSession	[입력] 세션 핸들
pMac	[출력] 메시지 무결성값 포인터
pnMacLen	[출력] 메시지 무결성값 길이

6.1.42.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.42.3. 예제

```
MC_UCHAR *data, *mac;
MC_UINT dataLen, macLen;
...
MC_CreateMacInit(hSession, &mcAlg, hKey);
MC_CreateMacUpdate(hSession, data, dataLen);
...
macLen = 20; /* maximum size of mac */
```

```
rv = MC_CreateMacFinal(hSession, mac, &macLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
...
MC_CloseSession(hSession);
```

6.1.42.4. 참고

MC_CreateMacUpdate, MC_CloseSession

6.1.43. MC_VerifyMacInit

메시지 무결성값 검증을 초기화 한다. 세션 정보를 초기화 하고, 알고리즘과 비밀키를 설정한다.

```
MCAPI MC_RV MC_VerifyMacInit (
        MC_HSESSION hSession,
        MC_ALGORITHM *pAlg,
        MC_HOBJECT hKey
);
```

6.1.43.1. 파라메터

hSession	[입력] 세션 핸들
pAlg	[입력] 알고리즘 포인터
hKey	[입력] 키 오브젝트 핸들

6.1.43.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.43.3. 예제

```
MC_ALGORITHM mcAlg = {MC_ALGID_SHA1_HMAC, NULL, 0};
```

```
MC_HOBJECT hKey;
MC_CreateObject(hSession, key, keyLen, &hKey);
rv = MC_VerifyMactInit(hSession, &mcAlg, hKey);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.43.4. 참고

MC_OpenSession, MC_CreateObject, MC_VerifyMac, MC_VerifyMacUpdate

6.1.44. MC_VerifyMac

단일 메시지에 대한 무결성값을 검증한다. 메시지 인증값 검증 종료시 세션 환경변수가 참조하는 메모리는 제로화 된다.

```
MCAPI MC_RV MC_VerifyMac (
        MC_HSESSION hSession,
        MC_UCHAR *pData,
        MC_UINT nDataLen,
        MC_UCHAR *pMac,
        MC_UINT nMacLen
);
```

6.1.44.1. 파라메터

hSession	[입력] 세션 핸들
pData	[입력] 데이터 포인터
nDataLen	[입력] 데이터 길이
pMac	[입력] 메시지 무결성값 포인터
nMacLen	[입력] 메시지 무결성값 길이

6.1.44.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.44.3. 예제

```
MC_UCHAR *data, *mac;
MC_UINT dataLen, macLen;
MC_ALGORITHM mcAlg = {MC_ALGID_SHA1_HMAC, NULL, 0};
MC_HOBJECT hKey;
…
MC_VerifyMactInit(hSession, &mcAlg, hKey);
rv = MC_VerifyMac(hSession, data, dataLen, mac, macLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
…
MC_CloseSession(hSession);
```

6.1.44.4. 참고

MC_VerifyMacInit, MC_CloseSession

6.1.45. MC_VerifyMacUpdate

메시지를 한번에 입력하기 어려운 경우 또는 메시지가 여러 부분으로 구성되어 있는 경우, 메시지 무결성값을 검증하기 위한 메시지의 일부를 입력한다.

```
MCAPI MC_RV MC_VerifyMacUpdate (
        IN MC_HSESSION hSession,
        IN MC_UCHAR *pData,
        IN MC_UINT nDataLen
);
```

6.1.45.1. 파라메터

hSession	[입력] 세션 핸들
pData	[입력] 데이터 포인터
nDataLen	[입력] 데이터 길이

6.1.45.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.45.3. 예제

```
MC_UCHAR *data;
MC_UINT dataLen;
…
MC_VerifyMacInit(hSession, &mcAlg, hKey);
rv = MC_VerifyMacUpdate(hSession, data, dataLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.45.4. 참고

MC_VerifyMacInit, MC_VerifyMacFinal

6.1.46. MC_VerifyMacFinal

MC_VerifyMacUpdate 를 통하여 입력받은 메시지와 무결성값을 입력받아 메시지의 무결성을 검증한다.
메시지 인증값 검증 종료시 세션 환경변수가 참조하는 메모리는 제로화 된다.

```
MCAPI MC_RV MC_VerifyMacFinal (
        MC_HSESSION hSession,
        MC_UCHAR *pMac,
        MC_UINT nMacLen
);
```

6.1.46.1. 파라메터

hSession	[입력] 세션 핸들
pMac	[입력] 메시지 무결성값 포인터
nMacLen	[입력] 메시지 무결성값 길이

6.1.46.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.46.3. 예제

```
MC_UCHAR *data, *mac;
MC_UINT dataLen, macLen;
...
MC_VerifyMacInit(hSession, &mcAlg, hKey);
MC_VerifyMacUpdate(hSession, data, dataLen);
...
rv = MC_VerifyMacFinal(hSession, mac, macLen);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
...
MC_CloseSession(hSession);
```

6.1.46.4. 참고

MC_VerifyMacUpdate, MC_CloseSession

6.1.47. MC_GenerateRandom

nRandomLen 길이의 의사 난수(Pseudo random)를 생성한다.
pRandomData 는 nRandomLen 크기의 메모리 할당이 되어 있어야 한다.

```
MCAPI MC_RV MC_GenerateRandom (
        MC_HSESSION hSession,
        MC_ALGORITHM *pAlg,
        MC_UCHAR *pRandomData,
        MC_UINT nRandomLen
);
```

6.1.47.1. 파라메터

hSession [입력] 세션 핸들
pAlg [입력] 알고리즘 포인터
pRandomData [출력] 의사난수 포인터
nRandomLen [입력] 생성할 의사 난수 길이

6.1.47.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.47.3. 예제

```
MC_ALGORITHM mcAlg = {MC_ALGID_SHA256DRBG, NULL, 0};
MC_UCHAR prng[16];
rv = MC_GenerateRandom(hSession, &mcPrng, prng, sizeof(prng));
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.47.4. 참고

MC_OpenSession, MC_GenerateKey, MC_EncryptInit, MC_CreateMacInit

6.1.48. MC_GenerateKey

대칭키 알고리즘과 MAC 값 생성에 필요한 비밀키를 생성한다. 또한 KCDSA 도메인 파라메터를 생성할 수 있으며, KCDSA 도메인 파라메터는 13.2.2.4 항목의 ASN.1 형식으로 생성하여 phKey 로 반환한다. 이때 도메

인 파라메터의 값을 얻어오기 위하여 MC_GetObjectValue 함수를 이용할 수 있다.

```
MCAPI MC_RV MC_GenerateKey (
        IN MC_HSESSION hSession,
        IN MC_ALGORITHM *pAlg,
        OUT MC_HOBJECT *phKey
);
```

6.1.48.1. 파라메터

hSession	[입력] 세션 핸들
pAlg	[입력] 알고리즘 포인터
phKey	[출력] 키 오브젝트 핸들

6.1.48.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.48.3. 예제

```
MC_ALGORITHM mcAlg = {MC_ALGID_SEED, NULL, 0};
MC_HOBJECT hKey;
rv = MC_GenerateKey(hSession, &mcAlg, &hKey);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.48.4. 참고

MC_OpenSession, MC_EncryptInit, MC_SignInit

6.1.49. MC_GenerateKeyPair

비대칭키 알고리즘의 공개키와 개인키를 생성한다. 이때 생성되는 키쌍은

전자서명 생성 및 검증을 통하여 키쌍에 대한 검증이 가능하며 각 알고리즘에 따른 키쌍은 13.2 절의 ASN.1 키형식을 따른다.

```
MCAPI MC_RV MC_GenerateKeyPair (
        IN MC_HSESSION hSession,
        IN MC_ALGORITHM *pAlg,
        OUT MC_HOBJECT *phPubKey,
        OUT MC_HOBJECT *phPriKey
);
```

6.1.49.1. 파라메터

hSession	[입력] 세션 핸들
pAlg	[입력] 알고리즘 포인터
phPubKey	[출력] 공개키 오브젝트 핸들
phPriKey	[출력] 개인키 오브젝트 핸들

6.1.49.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.49.3. 예제

```
MC_HOBJECT hPubkey, hPrikey;
rv = MC_GenerateKeyPair(hSession, &mcAlg, &hPubkey, &hPrikey);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.49.4. 참고

MC_OpenSession, MC_SignInit, MC_EncryptInit

6.1.50. MC_WrapKey

WrapKey 함수는 서비스 함수로, 입력 받은 키를 검증대상 알고리즘을 이

용하여 CBC 모드로 암호화한다.
암호모듈에서 사용되는 비밀키, 개인키, 공개키, 인증키 등의 키 오브젝트핸들을 사용자 암호를 이용하여 암호화하여 암호모듈에서 응용프로그램으로 출력한다.

```
MCAPI MC_RV MC_WrapKey (
        IN MC_HSESSION hSession,
        IN MC_ALGORITHM *pAlg,
        IN MC_HOBJECT hWrappingKey,
        IN MC_HOBJECT hKey,
        OUT MC_UCHAR *pWrappedKey,
        OUT MC_UINT *pnWrappedKeyLen
);
```

6.1.50.1. 파라메터

hSession	[입력] 세션 핸들
pAlg	[입력] 알고리즘 포인터
hWrappingKey	[입력] 사용자 암호 오브젝트 핸들
hKey	[입력] 암호화 할 키 오브젝트 핸들
pWrappedKey	[출력] 암호화된 키 포인터
pnWrappedKeyLen	[출력] 암호화된 키 길이

6.1.50.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.50.3. 예제

```
MC_HOBJECT hWrappingKey, hKey;
rv = MC_WrapKey(hSession, &mcAlg, hWrappingKey, hKey,
                        WrappedKey, &nWrappedKey);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.50.4. 참고

MC_OpenSession, MC_UnwrapKey, MC_CloseSession

6.1.51. MC_UnwrapKey

WrapKey 함수는 서비스 함수로, 입력 받은 키를 검증대상 알고리즘을 이용해서 CBC 모드로 복호화한다.
암호화된 비밀키, 개인키, 공개키, 인증키 등의 키를 응용프로그램에서 암호모듈에 설정하기 위한 키오브젝트핸들을 생성한다.

```
MCAPI MC_RV MC_UnwrapKey (
        IN MC_HSESSION hSession,
        IN MC_ALGORITHM *pAlg,
        IN MC_HOBJECT hWrappingKey,
        IN MC_UCHAR *pWrappedKey,
        IN MC_UINT nWrappedKeyLen,
        OUT MC_HOBJECT *phKey
);
```

6.1.51.1. 파라메터

hSession	[입력] 세션 핸들
pAlg	[입력] 알고리즘 포인터
hWrappingKey	[입력] 사용자 암호 오브젝트 핸들
pWrappedKey	[입력] 암호화된 키 포인터
nWrappedKeyLen	[입력] 암호화된 키 길이
phKey	[출력] 키 오브젝트 핸들

6.1.51.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.51.3. 예제

```
MC_HOBJECT hWrappingKey, hKey;
rv = MC_UnwrapKey(hSession, &mcAlg, hWrappingKey,
                        WrappedKey, nWrappedKey, &hKey);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.51.4. 참고

MC_OpenSession, MC_WrapKey, MC_CloseSession

6.1.52. MC_DeriveKey

암호화된 비밀키, 개인키, 공개키, 인증키 등의 키를 응용프로그램에서 암호 모듈에 설정하기 위한 키오브젝트핸들을 생성한다.
알고리즘에 따라 키 유도 또는 키 설정 기능이 동작한다. 키 유도 알고리즘을 이용하여 생성 된 키는 키를 암호화 하는데에만 사용하기를 권장한다. (데이터 암복호화에는 사용하지 않음)

```
MCAPI MC_RV MC_UnwrapKey (
        IN MC_HSESSION hSession,
        IN MC_ALGORITHM *pAlg,
        IN MC_HOBJECT hPriKey,
        OUT MC_UCHAR *pDerivedKey,
        OUT MC_UINT *pnDerivedKeyLen
);
```

6.1.52.1. 파라메터

hSession [입력] 세션 핸들
pAlg [입력] 알고리즘 포인터

PBKDF 키 유도 알고리즘의 경우 아래와 같이 ASN.1 형식의 구조체를 생성하여 pAlg 의 파라미터로 입력 해야 한다.

```
pbkdfParam::= SEQUENCE {
  salt                  OctetString,
  iterationCounter      INTEGER
}
```

Hmac KDF 키 유도 알고리즘의 경우 아래와 같이 ASN.1 형식의 구조체를 생성하여 pAlg 의 파라미터로 입력 해야 한다.

```
HmacKdfParam::= SEQUENCE {
  label          OctetString,
  context        OctetString
}
```

hPriKey [입력] 사용자 개인키 오브젝트 핸들, pbkdf 의 경우 패스워드, hmac kdf 의 경우 키 입력
pDerivedKey [출력] 설정 또는 유도된 키 포인터
pnDerivedKeyLen [출력] 설정 또는 유도된 키 길이

6.1.52.2. 리턴값

성공시 MC_OK, 실패시 에러코드.
실패시 MC_GetError() 또는 MC_GetErrorString()을 통하여 에러코드와 내용을 확인할 수 있다.

6.1.52.3. 예제

```
MC_HOBJECT hKey;
rv = MC_Derive(hSession, &mcAlg, hKey, pDK, &nDK);
if(rv != MC_OK) printf("ERROR: %s (%d)₩n", MC_GetErrorString(rv), rv);
```

6.1.52.4. 참고

MC_OpenSession

6.2. 에러 코드

에러 코드 0x0001 (MC_FAIL) 의 에러 스트링은 GENERAL_ERROR 가 발생한 경우 심각한 에러로 암호모듈을 재설치 해야 한다.

에러코드	에러 스트링	설명
0x0000	OK	성공
0x0001	GENERAL_ERROR	심각한 에러 (무결성 검증 실패, 알고리즘

		시험 실패)
0x1001	NULL_DATA_POINTER	널 포인터 입력
0x1002	DATA_LENGTH	데이터 길이 에러
0x1003	MEMORY_ALLOC_FAILED	메모리 할당 실패
0x1004	MCAPI_NOT_INITIALIZED	암호모듈이 초기화 되어있지 않음
0x1005	MCAPI_ALREADY_INITIALIZED	암호모듈이 이미 초기화 되어있음
0x1006	CONTEXT_NOT_CREATED	환경변수가 초기화 되어있지 않음
0x1007	CONTEXT_ALREADY_CREATED	환경변수가 이미 초기화 되어있음
0x1008	INVALID_POINTER	잘못된 포인터
0x1009	INVALID_HANDLE	잘못된 핸들
0x100A	DATA_NOT_CREATED	오브젝트가 생성되어있지 않음
0x100B	DATA_CREATED	오브젝트가 이미 생성되어 있음
0x100C	UNSUPPORTED_ALGORITHM	지원하지 않는 알고리즘의 사용
0x100D	UNSUPPORTED_PADTYPE	지원하지 않는 패딩 타입
0x100E	UNSUPPORTED_MODE	지원하지 않는 대칭키 운영모드
0x100F	INIT_PROCESS_MISSING	초기화 단계가 생략 되었음
0x1010	UPDATE_PROCESS_MISSING	갱신 단계가 생략 되었음
0x1011	OTHER_PROCESS_RUNNING	중첩된 암호서비스의 사용
0x1012	SET_KEY	키 설정 에러
0x1013	SET_IV	초기벡터 설정 에러
0x1014	SET_PUBKEY	공개키 설정 에러
0x1015	SET_PRIKEY	개인키 설정 에러
0x1016	SET_PARAM	파라메터 설정 에러
0x1017	PROCESS_FAILED	암호서비스의 프로세스 진행 에러, 암호연산 에러
0x1018	ENCRYPTED_DATA_LENGTH	패딩되어있지 않는 평문의 암호화
0x1019	DECRYPT_FAILED	복호화 실패
0x101A	VERIFY_FAILED	메시지 전자서명 검증/무결성 검증 실패
0x101B	NOT_ENOUGH_BUFFER	출력 데이터의 출력버퍼 길이가 부족함
0x101C	MC_ERR_NOT_SESSION_OBJECT	세션 오브젝트가 아님