HMAC 알고리즘에 대한
소스코드 활용 매뉴얼

2020. 12.

KISA 한국인터넷진흥원

제·개정 이력

순번	제·개정일	제·개정 내역
1	2020.12	"HMAC 알고리즘에 대한 소스코드 활용 매뉴얼" 발간

Contents

1. 개요

HMAC(Hash-based MAC)은 암호학적 해시함수와 비밀 키를 기반으로 하는 메시지 인증 코드(Message Authentication Code, MAC)이다. HMAC은 SHA-2, SHA-3와 같은 해시를 기반으로 값이 계산되며, 해시함수에 따라 HMAC-SHA-2, HMAC-SHA-3로 구분할 수 있다. HMAC은 데이터의 무결성 검증과 메시지 인증에 이용할 수 있다.

본 매뉴얼은 HMAC-SHA256를 다양한 언어 및 플랫폼에서 쉽게 활용할 수 있도록 C/C++, Java, ASP, PHP, JSP용으로 개발된 소스코드의 설명과 함께 사용 시 주의사항에 대해 설명한다.

2. HMAC 함수

가. HMAC 함수

MAC 함수는 해시함수의 암호학적 특성인 데이터 무결성 검증 이외에 메시지에 대한 인증 기능을 제공하는 암호학적 함수이다. MAC 함수는 메시지와 비밀 키를 입력받아 MAC 값을 생성한다. 송신자와 수신자는 동일한 MAC용 비밀 키를 공유한다. 송신자는 전송용 데이터를 MAC용 비밀 키로 MAC 값을 계산하고, 데이터와 그에 해당하는 MAC 값을 함께 수신자에게 전송한다. 수신자는 MAC용 비밀 키로 수신받은 데이터에 대한 MAC 값을 계산한 후 수신한 MAC 값과 비교하여 동일 여부를 판단한다. 이때 MAC 값이 동일하면, 데이터의 무결성이 유지되었다고 판단한다. 또한, 해당 MAC 값은 동일한 MAC용 비밀 키를 보유해야만 생성 가능하므로, 송신자로부터 전송되었다는 것을 인증할 수 있게 된다.

HMAC 함수는 해시함수를 기반으로 하여 MAC 값을 생성하며, 적용된 해시함수에 따라 HMAC-SHA1, HMAC-SHA2, HMAC-SHA3 등으로 구분된다. HMAC은 두 번의 해시함수가 수행되는 구조이다. 여기서, 비밀 키는 먼저 내부와 외부에서 사용되는 두 키를 생성하기 위해 사용된다. 첫 번째 단계에서 메시지와 내부 키로부터 파생된 내부 해시 값이 생성된다. 두 번째 단계에서 내부 해시 값과 외부 키로부터 생성된 해시 값이 HMAC 값으로 생성된다. 이를 수식으로 나타내면 다음과 같다.

$$HMAC(K,m) = H((K' \oplus opad) \| H((K' \oplus ipad) \| m))$$

$$K' = \begin{cases} H(K) & K\ is\ larger\ than\ block\ size \\ K & otherwise \end{cases}$$

여기서, 각 기호가 나타내는 의미는 다음과 같다.

H : 암호학적 해시함수 (SHA1, SHA2, SHA3 등)
m : 인증을 위한 메시지
K : 비밀 키
K' : 비밀 키 K로부터 파생된 해시함수 H의 블록길이의 키;
  -K가 블록길이보다 작다면, K에 대해 블록길이가 될 때까지 0으로 패딩;
  -K가 블록길이보다 크다면, $H(K)$에 대해 블록길이가 될 때까지 0으로 패딩;
$\|$: 연접
$\oplus$: 비트단위 XOR
$opad$: 반복된 0x5c으로 구성된 블록길이의 외부 패딩
$ipad$: 반복된 0x36으로 구성된 블록길이의 내부 패딩

3. 소스코드

본 장에서는 HMAC-SHA256 구현의 각 소스코드 구성물에 대해 설명한다. 본 매뉴얼에서는 C/C++, Java, JSP, ASP, PHP 구현환경을 다루기 때문에 각 환경별 소스코드 구성 및 각 소스코드 구성요소에 대한 간략한 설명을 제시한다.

각 운영환경에서 구현 소스의 정확성을 위해서는 KISA 암호모듈검증/검증대상 암호알고리즘 테스트벡터(HMAC-SHA256 테스트벡터)를 활용하였다[1].

HMAC-SHA256 소스코드는 다음과 같이 구성되어 있다.

- HMAC-SHA256 의 C/C++ 소스코드
- HMAC-SHA256 의 Java 소스코드
- HMAC-SHA256 의 JSP 소스코드
- HMAC-SHA256 의 ASP 소스코드
- HMAC-SHA256 의 PHP 소스코드

	동작환경
C/C++	Windows
	Linux
Java	Windows
	Linux
JSP	Apache Tomcat 9.0.19
ASP	Windows10/IIS(Version 10.0.17763.1) .NET v4.5
PHP	PHP Version 7.2.10

< 소스코드 별 동작환경 >

모든 소스코드는 32비트/64비트 운영체제에서 모두 동작한다.

1) https://seed.kisa.or.kr/kisa/kcmvp/EgovVerification.do

4. 응용 프로그램

Windows, Linux 등 다양한 운영체제 환경에서 응용 프로그램을 개발과 함께 암호화를 적용할 수 있도록 기본적으로 많이 사용하는 C/C++ 및 Java를 기반으로 HMAC-SHA256 소스코드를 개발하였다.

가. C/C++

프로젝트 생성 및 소스 추가, 빌드 등 배포되는 소스코드를 이용하여 암호화를 실행하는 방법에 대해서는 Microsoft 社의 Visual studio 2019를 활용하여 설명하도록 한다.

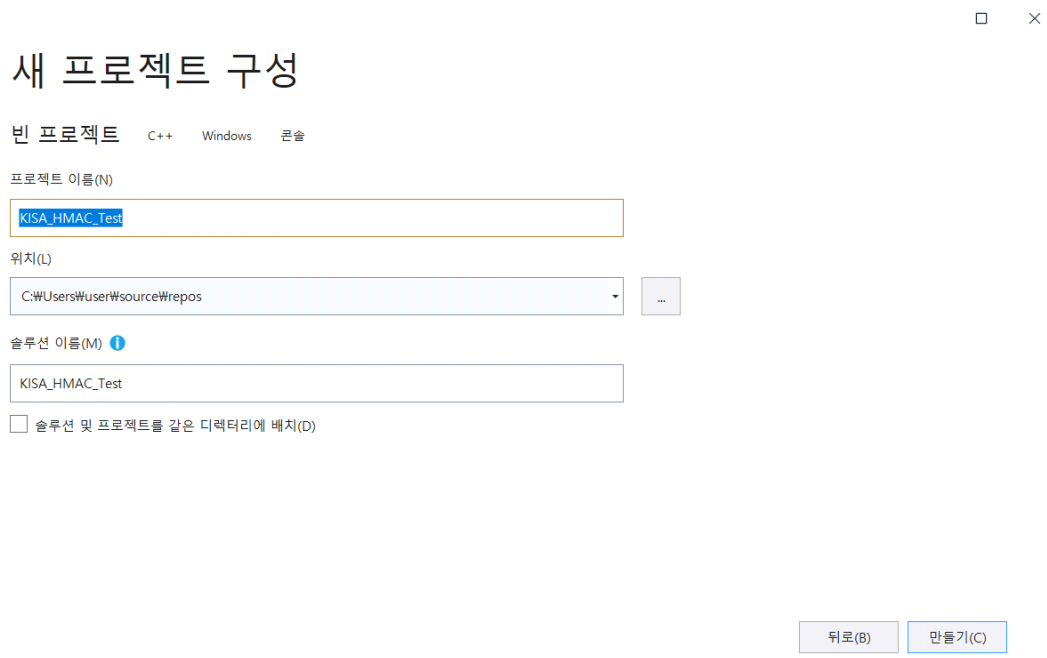
1) Windows 운영체제 환경에서 C/C++ 테스트 소스 실행

Windows 운영체제 환경에서 C/C++ 소스코드를 개발할 경우 다양한 IDE를 활용하여 개발할 수 있다. 본 절에서는 Microsoft 社에서 배포하고 있는 Visual Studio 2019을 활용하여 소스코드를 사용하는 방법을 설명한다. Visual Studio 2019가 아닌 다른 IDE를 활용하여 개발하는 내용은 생략하나, 이와 유사한 방법을 활용하여 개발할 수 있다.

Visual Studio 2019

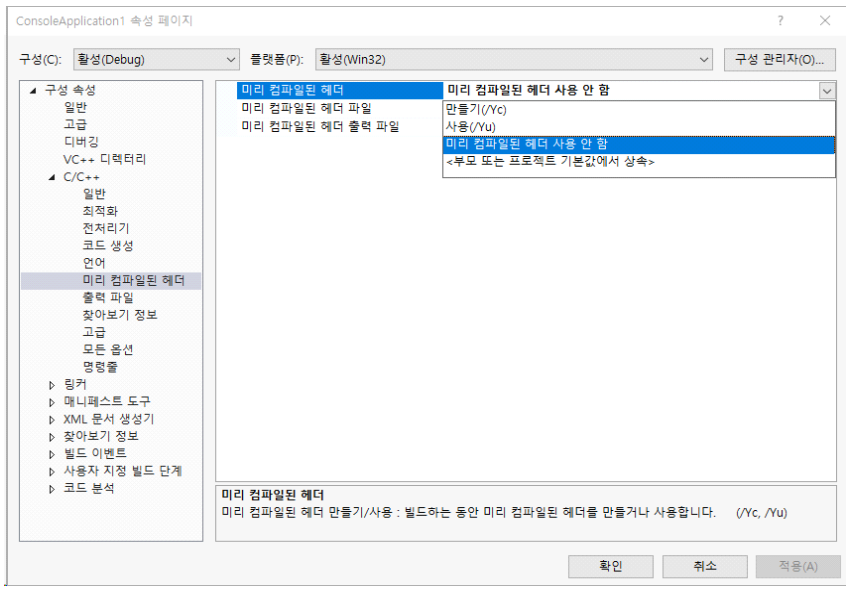
Visual Studio를 실행하여 [파일]에서 [새로만들기]->[프로젝트]를 클릭하여 "콘솔 앱"을 선택한다. 이름에는 프로젝트명을 입력하고 위치에는 생성시키고자 하는 곳의 위치를 지정한다.

생성된 프로젝트는 미리 컴파일된 헤더를 사용하도록 구성된 기본 파일들도 함께 생성된다. 개발환경에 따라 미리 컴파일된 헤더를 사용할 경우와 그렇지 않은 경우는 상이하므로 적절한 환경 설정을 해야 한다.



본 설명에서는 기본 생성된 파일을 사용하지 않고 배포된 파일을 사용하는 것을 기준으로 설명한다. 배포된 파일을 추가하기 위해서는 "솔루션 탐색기"에서 프로젝트를 우클릭한 뒤 [추가]->[기존 항목]을 선택한 후, 배포된 소스코드와 헤더파일을 추가한다.

프로젝트 속성 창이 열리면 좌측 트리에서 [C/C++]->[미리 컴파일된 헤더]을 클릭한다. [미리 컴파일된 헤더]에서 "미리 컴파일된 헤더 사용 안 함"을 선택한다.



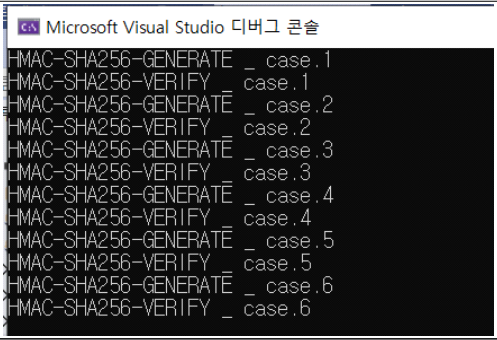
위와 같이 설정이 완료된 이후, 프로젝트를 빌드 한 뒤 테스트 소스를 실행한다.

<table>
<tr>
<td>테스트코드 구성</td>
<td>

```
5   void test_hmac_sha256()
6   {
7       unsigned char msg[1024] = { 0, }, key[1024] = { 0, }, output[32] = { 0, }, hmac[32];
8       unsigned int msgLen = 0, keyLen = 0, outputLen = 0, ret = 0;
9       //void HMAC_SHA256(const u8* message, u32 mlen, const u8* key, u32 klen, u8 hmac[SHA256_DIGEST_VALUELEN]
10
11      printf("HMAC-SHA256-GENERATE _ case.1\n");
12      keyLen = asc2hex(key, "C6F1D667A50AAEBA5A200A0A7CC24FFBB24984426AB8ABACCEE75162F3E1646B");
13      msgLen = asc2hex(msg, "548A457280851ECA0F5476AFDAC102CF6C7DBE09B3083D74FBD03DA31E9D7F27"
14          "F42CD656111A7D4BB005AD2EEAED6FB62CE0B0EBE7D6933189DA0B82AD6AA8FB8E21B19AC293744625"
15          "79DA0F130E3EB8DAB87F726EEB54EB5F4AE087091087ED0BAFFFC6FAB7AAC156F823DBBCEB17DD5E4E"
16          "5626B10F29AA656BE73B9A57C308");
17      outputLen = asc2hex(output, "96C37F36CA0DEA3B2B3E60F1F6CDF79CFF72CA2A43A091C8105AE882A690EF2F");
18
19      HMAC_SHA256(msg, msgLen, key, keyLen, hmac);
20
21      assert(memcmp(hmac, output, outputLen) == 0);
22
23      printf("HMAC-SHA256-VERIFY _ case.1\n");
24
25      ret = Verify_HMAC_SHA256(msg, msgLen, key, keyLen, hmac);
26
27      assert(ret == 0);
```

- 사전에 정의된 Known answer 값을 output에 저장한 후, HMAC의 결과인 hmac과 비교 수행
- assert() 함수를 사용하여 일치하지 않을 경우 오류 메시지 출력

</td>
</tr>
<tr>
<td>실행화면</td>
<td>

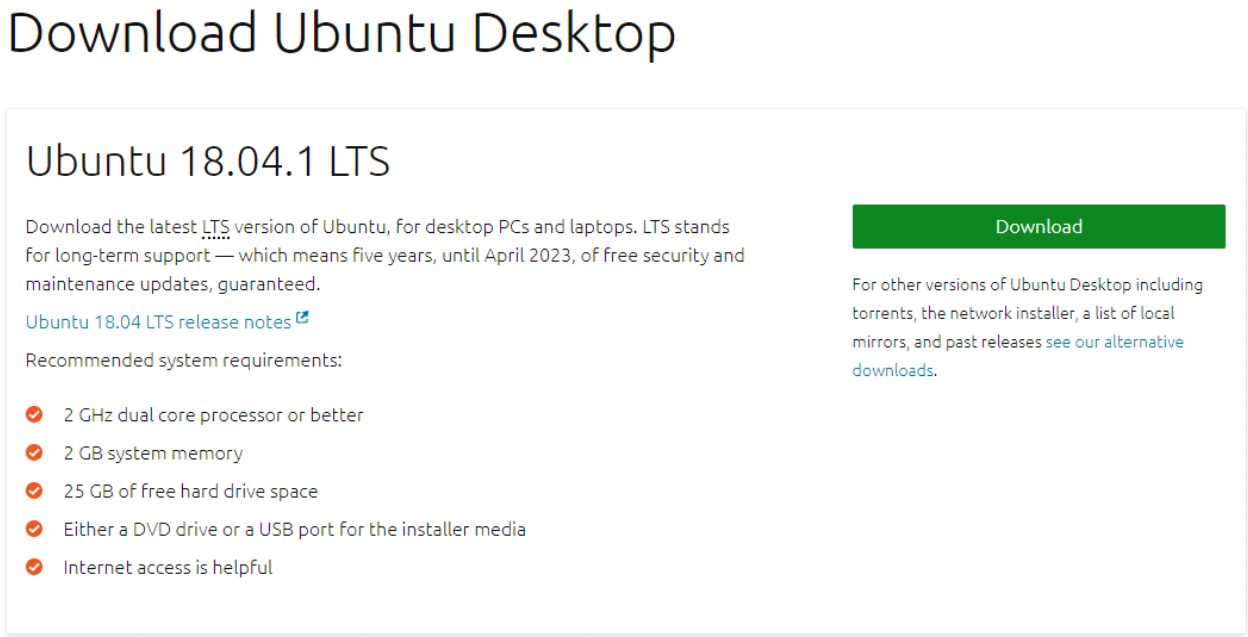


- HMAC 출력값과 사전에 정의된 Known answer 값을 비교하여 값이 다를 경우에만 오류 메시지 출력

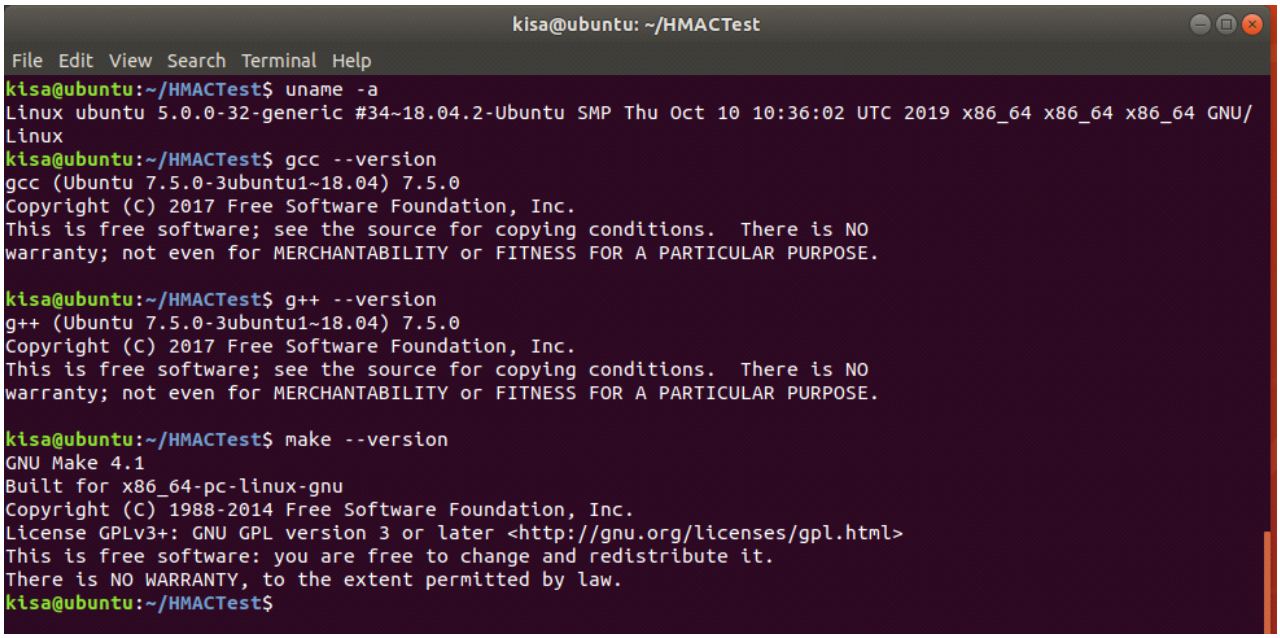
</td>
</tr>
</table>

2) Linux HMAC 테스트 소스 실행

Linux 운영체제 환경에서 C/C++ 소스코드를 개발하는 방법 역시 다양하다. Windows와 달리 Linux 에서는 운영체제에 기본 개발환경이 갖춰진 경우가 일반적이다. 따라서 각 Linux 운영체제의 터미널 또는 콘솔에서 간단한 명령어로도 C/C++ 소스코드를 활용할 수 있다. 본 절에서는 Linux 배포판 중 하나인 Ubuntu 18.04 환경에서 소스코드를 활용하는 방법을 설명한다.



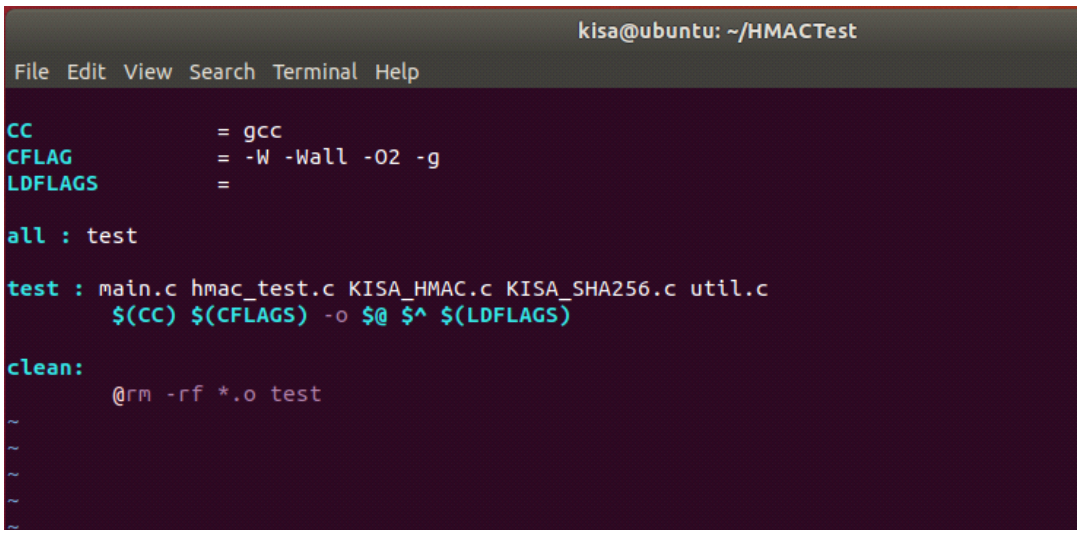
Ubuntu 18.04 버전을 내려받은 후 설치하면 아래와 같이 환경 정보를 확인할 수 있다.



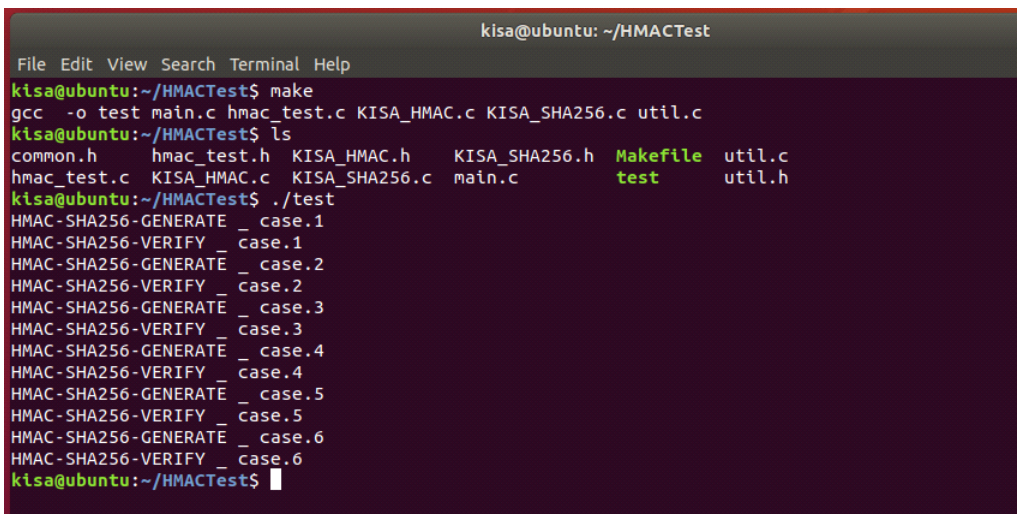
간혹, minimal version을 설치한 경우 위와 같은 정보를 확인할 수 없다. 그런 경우 아래 명령어를 통해 개발환경을 구축할 수 있다.

: $ sudo apt install build-essential –y

위 명령어를 통해 위 개발환경이 구축된 뒤 "gcc" 또는 "g++" 명령어를 활용하여 배포된 소스코드를 활용할 수 있다. Makefile을 작성하여 "make" 명령어로 손쉽게 개발할 수 있다. 아래는 Makefile의 예이다.



"make" 명령어를 통해 실행파일을 생성한다. 다음은 make를 수행한 결과와 make를 통해 생성된 test 실행파일을 통해 HMAC의 정상동작을 확인한 결과이다.



3) C/C++ 코드 구성

소스파일	내용
KISA_SHA256.h KISA_SHA256.c	SHA-256 해시함수 구현

KISA_HMAC.h KISA_HMAC.c	HMAC_SHA256 구현
util.h util.c	유틸리티 함수(데이터 형식 변환, 출력 등)
hmac_test.h hmac_test.c	HMAC 테스트 함수 구현
main.c	메인함수 구현(테스트 함수 호출)

< 윈도우즈/리눅스 환경에서 C/C++ 기반 HMAC 참조 구현 >

KISA_HMAC.c와 KISA_HMAC.h 파일이 HMAC 기능을 제공하는 소스코드이며, 이는 내부적으로 KISA_SHA256.c/KISA_SHA256.h를 이용한다. hmac_test.c는 HMAC 구현정확성을 테스트하기 위한 테스트벡터 정의 및 테스트 코드를 담고 있다.

4) HMAC 소스코드 인터페이스

HMAC 소스코드는 최상위 API HMAC_SHA256 함수를 통해 해시값을 생성한다. 내부적으로 KISA_SHA256.c/KISA_SHA256.h에서 제공하는 SHA256_Init(), SHA256_Process(), SHA256_Close() 함수를 이용하여 MAC 값을 생성한다.

void HMAC_SHA256(const u8 *message, u32 mlen,
const u8 *key, u32 klen,
u8 hmac[SHA256_DIGEST_VALUELEN])

매개변수 :

- message MAC을 생성할 원본 메시지
- mlen 원본 메시지의 바이트 길이
- key MAC을 생성할 때 사용하는 키
- klen key의 바이트 길이
- hmac 생성된 HMAC 값

반환값 :

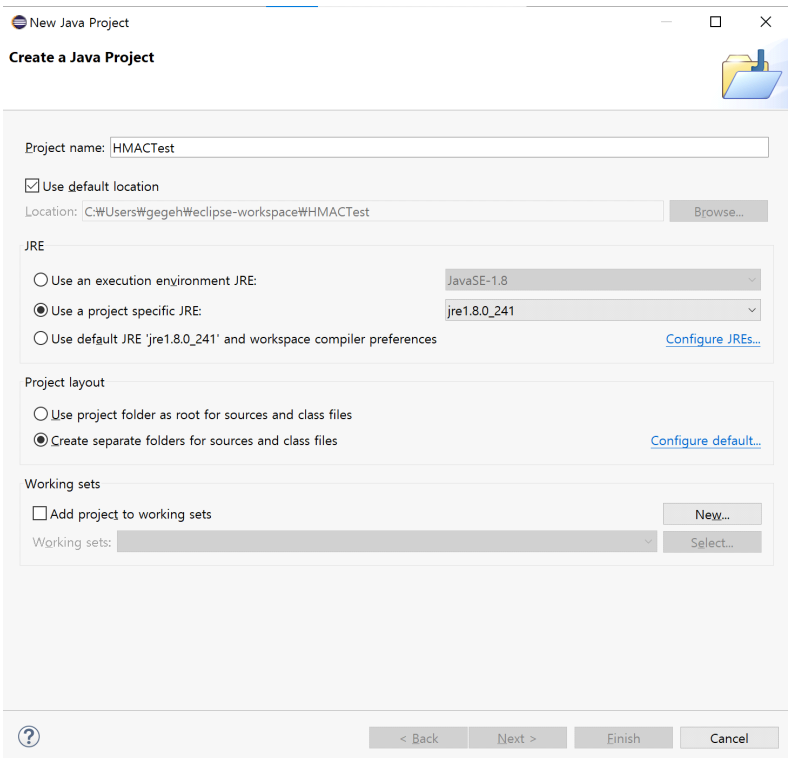
- 없음

나. Java

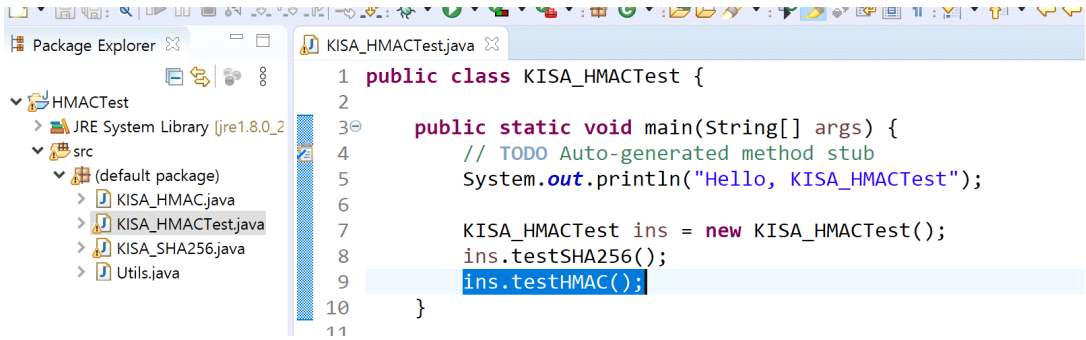
1) Windows HMAC 테스트 소스 실행

Windows 운영체제 환경에서 Java 소스코드를 개발하는데에는 다양한 IDE를 활용하여 개발할 수 있다. 본 절에서는 Eclipse PHOTON 무료 배포버전을 활용하여 소스코드를 사용하는 방법을 설명한다. Java는 기본적으로 개발환경에 Java JDK를 설치하여 개발환경을 구축해야 하나, 이에 대한 방법은 생략한다.

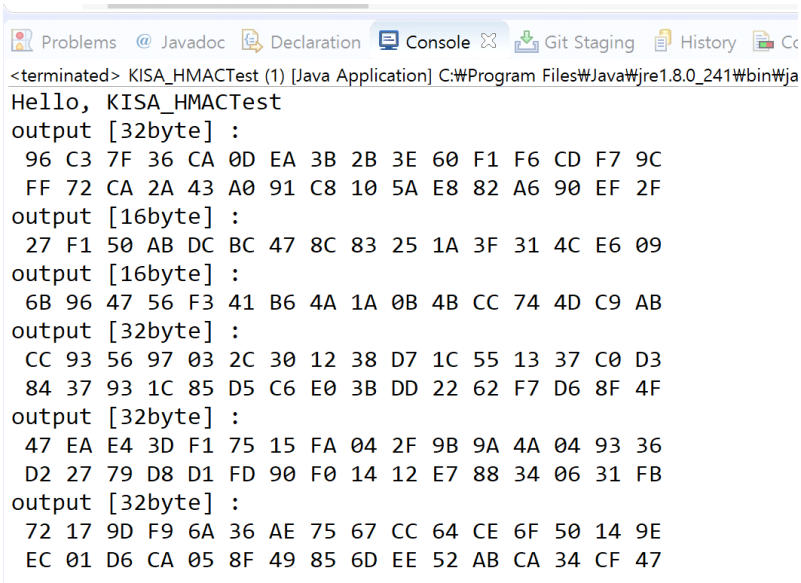
Eclipse Photon을 실행하여 [File]에서 [New]->[Java Project]를 클릭하여 프로젝트명을 기입하고 개발환경에 적절한 Java JRE를 선택하여 프로젝트를 생성한다.



생성된 프로젝트는 기본 파일을 함께 생성된다. 개발환경에 따라 필요한 파일을 사용하도록 설정한다. 본 설명에서는 기본 생성된 파일을 생성하지 않고 배포된 파일을 사용하는 것을 기준으로 설명한다. Eclipse에서 "Package Explorer"의 src에 배포된 소스코드를 추가한다. 이후 개발환경에 맞춰서 개발을 진행한다.

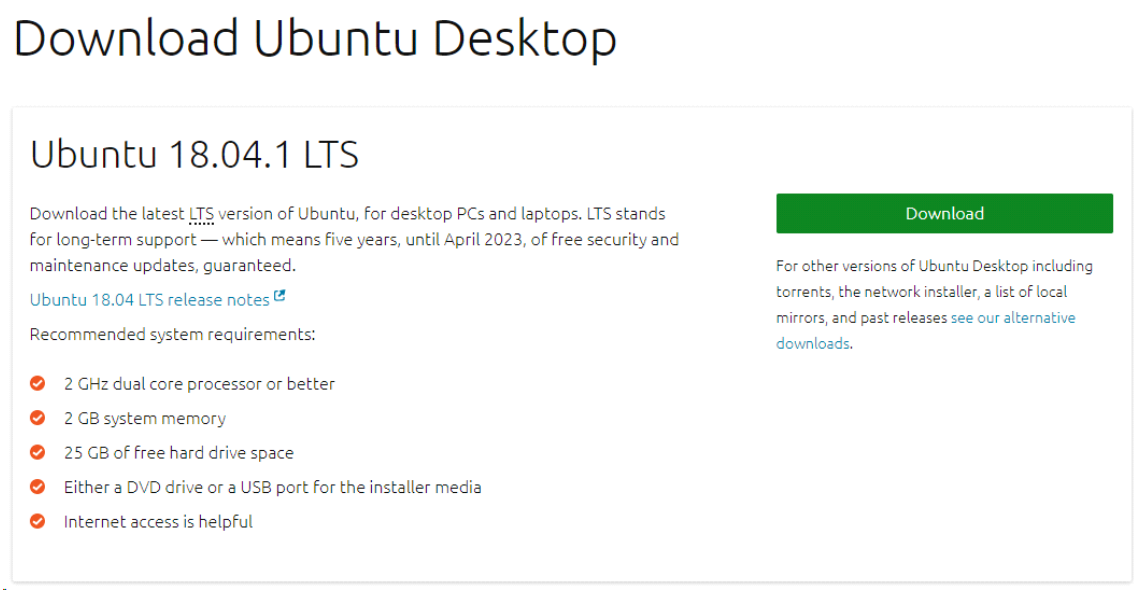


이클립스에서 추가한 소스코드를 빌드한 후 실행한다.

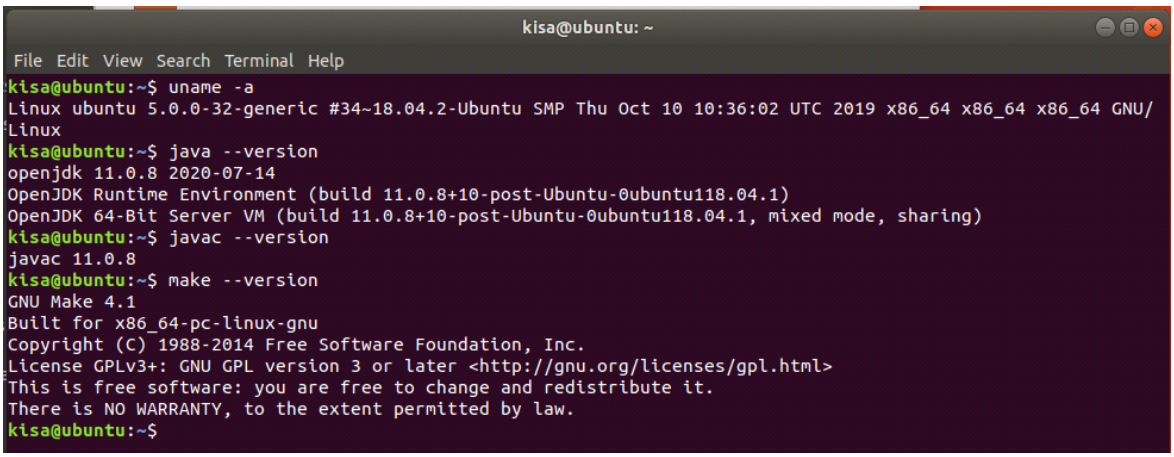


2) Linux HMAC 테스트 소스 실행

Linux 운영체제 환경에서 C/C++ 소스코드를 개발하는 방법 역시 다양하다. Windows와 달리 Linux에서는 운영체제에 기본 개발환경이 갖춰진 경우가 일반적이다. 따라서 각 Linux 운영체제의 터미널 또는 콘솔에서 간단한 명령어로도 C/C++ 소스코드를 활용할 수 있다. 본 절에서는 Linux 배포판 중 하나인 Ubuntu 18.04 환경에서 소스코드를 활용하는 방법을 설명한다.



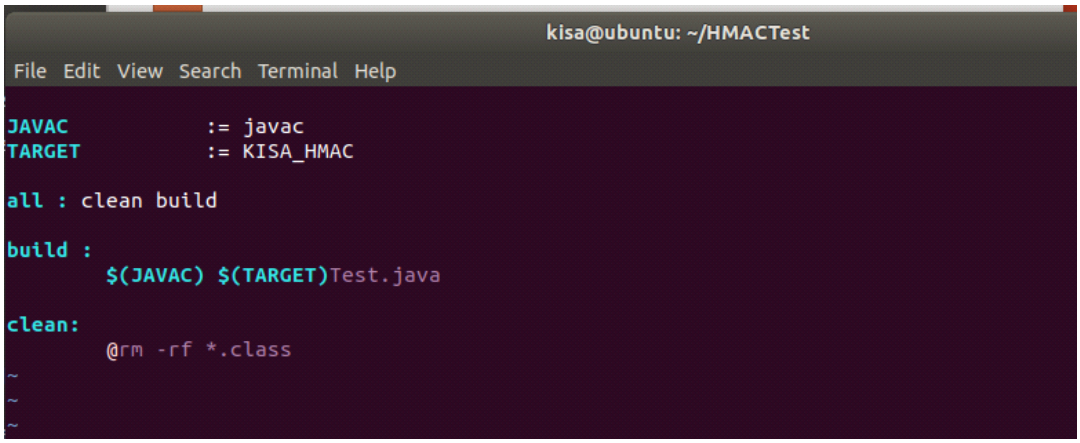
Ubuntu 18.04 버전을 내려받은 후 설치하면 아래와 같이 환경 정보를 확인할 수 있다.



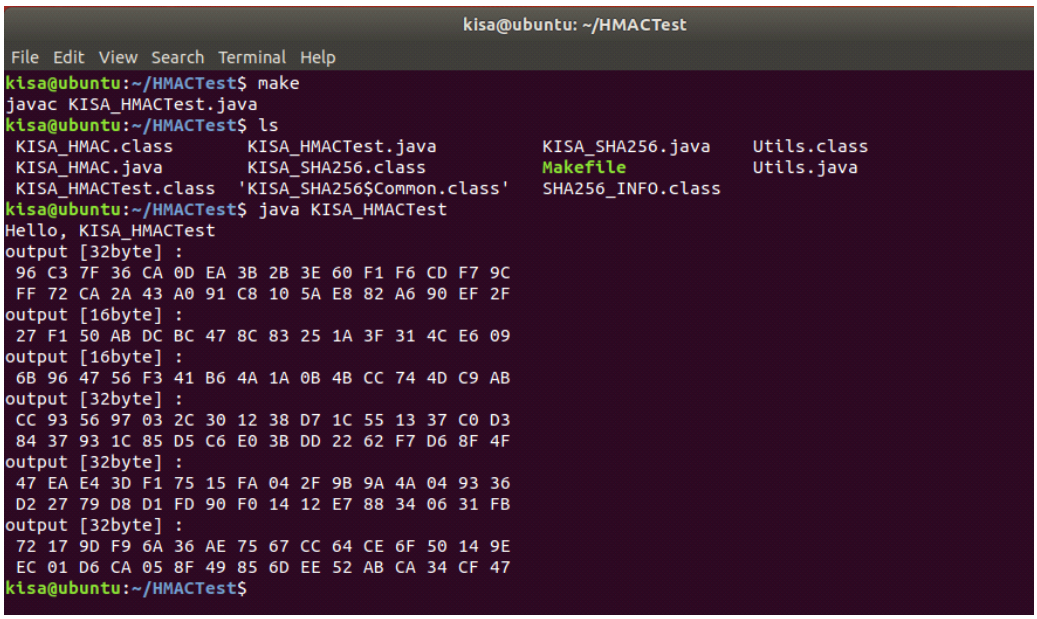
Ubuntu 18.04에서는 기본적으로 Java 개발환경이 구축되어 있지 않다. 하지만 openJDK는 Ubuntu의 패키지 설치로 간편하게 설치 가능하다.

$ sudo apt install default-jdk default-jre –y

위 명령어를 통해 Java 개발환경이 구축된 뒤 "javac" 명령어를 활용하여 배포된 소스코드를 활용할 수 있다. Makefile을 작성하여 "make" 명령어로 손쉽게 개발할 수 있다. 아래는 Makefile의 예이다.



"make" 명령어를 통해 Java로 컴파일된 결과물을 얻을 수 있으며, 다음은 make를 통해 생성된 "KISAHMACTest.class"를 실행한 하여 정상동작을 확인한 결과이다.



3) Java 코드 구성

소스파일	내용
KISA_SHA256.java	SHA-256 해시 메소드 구현
KISA_HMAC.java	HMAC_SHA256 구현
Utils.java	유틸리티 (데이터 형식 변환, 출력 등)
KISA_HMACTest.java	테스트 코드

< 윈도우즈/리눅스 환경에서 Java 기반 HMAC 참조 구현 >

HMAC 소스코드는 개발시 KISA_HMAC.java를 발췌하여 편리하게 HMAC를 호출하여 사용할 수 있

는 구조이며, 사용방법은 KISA_HMACTest.java참고하면 된다.

4) HMAC 소스코드 인터페이스

HMAC 소스코드는 최상위 API HMAC_SHA256_Transform 메소드를 통해 해시값을 생성한다.

public static void HMAC_SHA256_Transform(byte[] output, byte[] key, int keyLen, byte[] input, int inputLen)
매개변수 : - output 생성된 HMAC 값 - key MAC을 생성할 때 사용하는 키 - klen key의 바이트 길이 - input MAC을 생성할 원본 메시지 - inputLen 원본 메시지의 바이트 길이 **반환값 :** - 없음

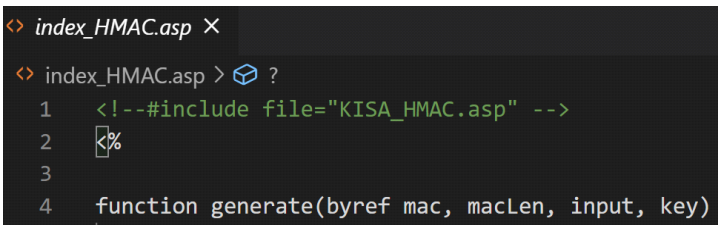
5. 웹 프로그램

데이터베이스, 홈페이지 등 다양한 Web Service 환경에서 전송·저장되는 구간의 암호화를 적용할 수 있도록 국산암호 소스코드를 개발하였다. 기본적으로 많이 사용하는 ASP, JSP, PHP를 기반으로 한다.

가. ASP

소스 활용 등 배포되는 소스코드를 이용하여 암호화를 실행하는 방법에 대해서는 간단한 에디터 상태에서 설명하도록 한다. 단, 웹 개발환경 구축에 대한 설명은 생략하기로 한다.

ASP로 개발된 웹 프로그램 소스코드에 배포된 파일을 포함시킨다.



KISA_HMAC.asp를 포함시킨 페이지를 브라우저에서 로드하면 다음과 같이 실행된다.

<키
(KEY)> "C6F1D667A50AAEBA5A200A0A7CC24FFBB24984426AB8ABACCEE75162F3E1646B"
:
<평문>
:
<인증
값 길이 "32"
>:
<MAC> "96C37F36CA0DEA3B2B3E60F1F6CDF79CFF72CA2A43A091C8105AE882A690EF2F"
:

1) ASP 코드 구성

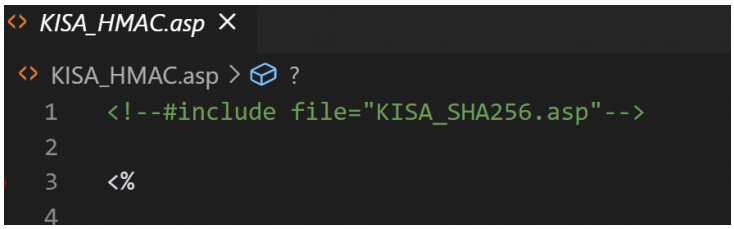
파일명	내용
KISA_SHA256.asp	SHA256 함수 구현(ASP)
KISA_HMAC.asp	SHA256 기반 HMAC 함수 구현
index_HMAC.asp	HMAC-SHA256 검증용 페이지 구현

< 윈도우즈 환경에서 ASP 기반 HMAC 참조 구현 >

HMAC-SHA256 소스코드는 개발시 KISA_HMAC.asp를 발췌하여 편리하게 HMAC을 호출하여 사용할 수 있는 구조이다. 다만, HMAC은 SHA256을 기반으로 동작하기 때문에 같은 경로에 KISA_SHA256.asp를 위치시켜야 한다. 사용방법은 index_HMAC.asp를 참고하면 된다.

2) HMAC-SHA256 소스코드 인터페이스

HMAC-SHA256 소스코드는 SHA256 기반으로 동작하므로 다음과 같이 KISA_SHA256.asp를 포함시켜야 한다.



HMAC-SHA256은 먼저 키 길이를 확인하여 SHA256의 블록길이인 512바이트보다 크다면, 키에 대해 SHA256을 수행한 해시값에 대해 0으로 패딩하여 512바이트로 맞춰준다. 키 길이가 512바이트보다 작다면, 0으로 패딩하여 512바이트로 맞춰준다. 이후, message을 활용하여 두 번의 SHA256을 수행하여 나온 출력 값을 HMAC으로 사용한다.

public Function HMAC_SHA256(message, mlen, key, klen, byref hmac)
매개변수 : - message 대상 메시지 - mlen 메시지에 대한 길이(바이트 단위) - key MAC 계산용 키 - klen 키에 대한 길이(바이트 단위) - hmac 출력될 hmac 값 **반환값 :** - 없음 **해당 함수는 SHA256 기반으로 작성된 코드이므로 32바이트의 HMAC이 출력됨**

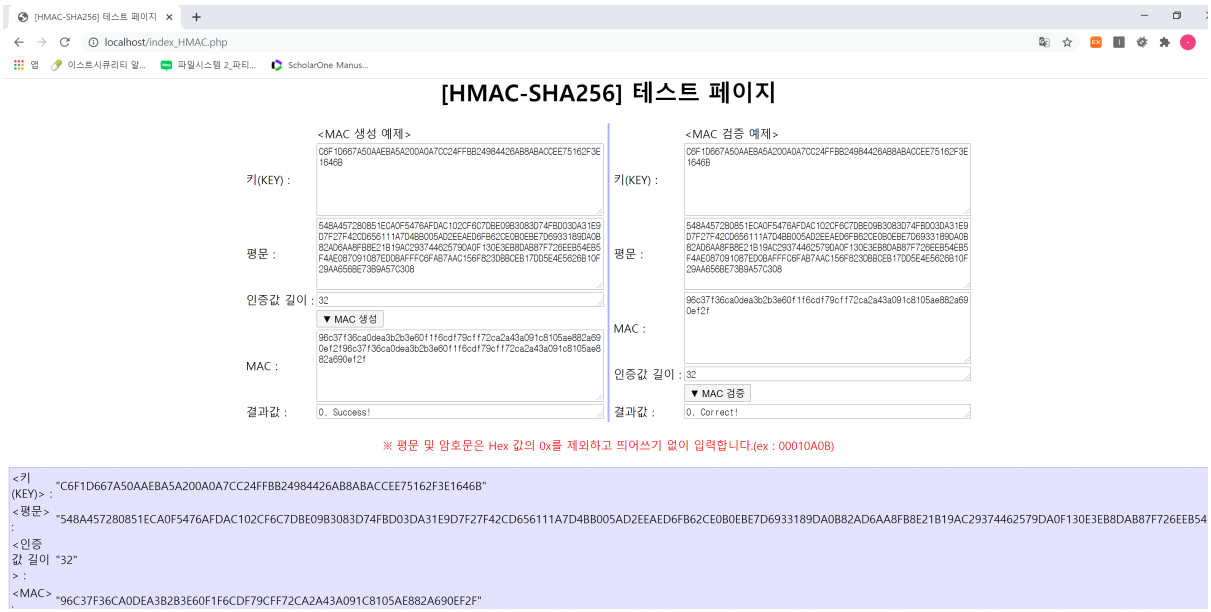
나. PHP

소스 활용 등 배포되는 소스코드를 이용하여 암호화를 실행하는 방법에 대해서는 간단한 에디터 상태에서 설명하도록 한다. 단, 웹 개발환경 구축에 대한 설명은 생략하기로 한다.

PHP로 개발된 웹 프로그램 소스코드에 배포된 파일을 포함시킨다.

```php
<?php
require_once ('KISA_HMAC.php');
```

KISA_HMAC.php를 포함시킨 페이지를 브라우저에서 로드하면 다음과 같이 실행된다.



1) PHP 코드 구성

파일명	내용
KISA_SHA256.php	SHA256 함수 구현(PHP)
KISA_HMAC.php	SHA256 기반 HMAC 함수 구현
index_HMAC.php	HMAC-SHA256 검증용 페이지 구현

< 윈도우즈 환경에서 PHP 기반 HMAC 참조 구현 >

HMAC-SHA256 소스코드는 개발시 KISA_HMAC.php를 발췌하여 편리하게 HMAC을 호출하여 사용할 수 있는 구조이다. 다만, HMAC은 SHA256을 기반으로 동작하기 때문에 같은 경로에 KISA_SHA256.php를 위치시켜야 한다. 사용방법은 index_HMAC.php를 참고하면 된다.

2) HMAC-SHA256 소스코드 인터페이스

HMAC-SHA256 소스코드는 SHA256 기반으로 동작하므로 다음과 같이 KISA_SHA256.php를 포함시켜야 한다.

```php
<?
require_once ('KISA_SHA256.php');

class HMAC{
```

HMAC-SHA256은 먼저 키 길이를 확인하여 SHA256의 블록길이인 512바이트보다 크다면, 키에 대해 SHA256을 수행한 해시값에 대해 0으로 패딩하여 512바이트로 맞춰준다. 키 길이가 512바이트보다 작다면, 0으로 패딩하여 512바이트로 맞춰준다. 이후, message을 활용하여 두 번의 SHA256을 수행하여 나온 출력 값을 HMAC으로 사용한다.

static function HMAC_SHA256($message, $mlen, $key, $klen, &$hmac)
매개변수 : - message 대상 메시지 - mlen 메시지에 대한 길이(바이트 단위) - key MAC 계산용 키 - klen 키에 대한 길이(바이트 단위) - hmac 출력될 hmac 값 **반환값 :** - 없음 **해당 함수는 SHA256 기반으로 작성된 코드이므로 32바이트의 HMAC이 출력됨**

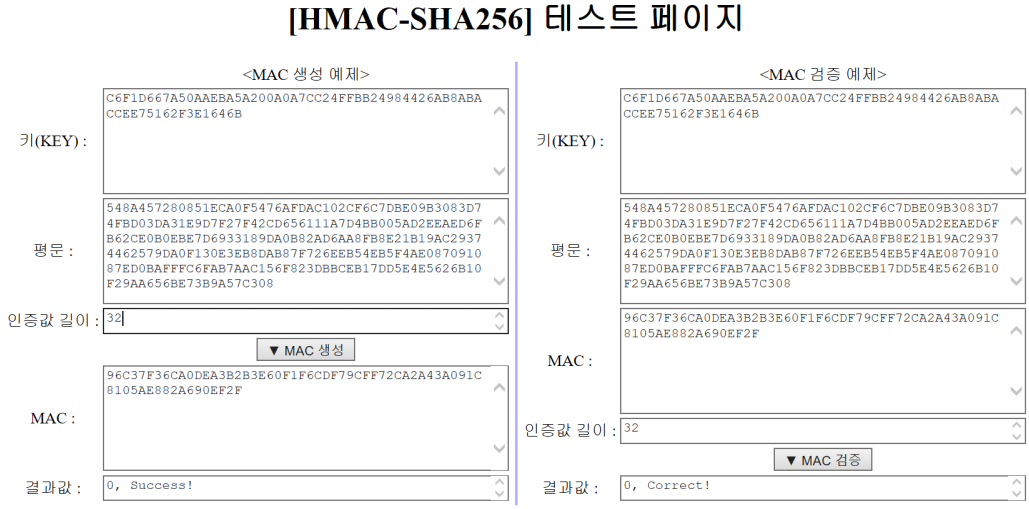
다. JSP

소스 활용 등 배포되는 소스코드를 이용하여 암호화를 실행하는 방법에 대해서는 간단한 에디터 상태에서 설명하도록 한다. 단, 웹 개발환경 구축에 대한 설명은 생략하기로 한다.

JSP로 개발된 웹 프로그램 소스코드에 배포된 파일을 포함시킨다.

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ include file="KISA_HMAC_SHA256.jsp" %>
<%!
```

KISA_HMAC_SHA256.jsp를 포함시킨 페이지를 브라우저에서 로드하면 다음과 같이 실행된다.



※ 평문 및 암호문은 Hex 값의 0x를 제외하고 띄어쓰기 없이 입력합니다.(ex : 00010A0B)

<키(KEY)> : C6F1D667A50AAEBA5A200A0A7CC24FFBB24984426AB8ABACCEE75162F3E1646B
<평문> : 548A457280851ECA0F5476AFDAC102CF6C7DBE09B3083D74FBD03DA31E9D7F27F42CD656111A7D4BB005AD2EEAED6FB62CE0B0EBE7D6933189DA0B82AD6AA8FB8E
<인증값 길이> : 32
<MAC> : 96C37F36CA0DEA3B2B3E60F1F6CDF79CFF72CA2A43A091C8105AE882A690EF2F

1) JSP 코드 구성

소스파일	내용
KISA_SHA256.jsp	SHA-256 해시함수 구현
KISA_HMAC_SHA256.jsp	HMAC_SHA256 구현
Utils.jsp	유틸리티 함수(데이터 형식 변환, 출력 등)
KISA_HMAC_SHA256_index.jsp	테스트를 위한 메인 페이지

< 윈도우즈 환경에서 JSP 기반 HMAC 참조 구현 >

HMAC-SHA256 소스코드는 개발시 KISA_HMAC.jsp를 발췌하여 편리하게 HMAC을 호출하여 사용할 수 있는 구조이다. 다만, HMAC은 SHA256을 기반으로 동작하기 때문에 같은 경로에 KISA_SHA256.php를 위치시켜야 한다. 사용방법은 KISA_HMAC_SHA256_index.jsp를 참고하면 된다.

2) HMAC-SHA256 소스코드 인터페이스

HMAC-SHA256 소스코드는 최상위 API HMAC_SHA256_Transform 메소드를 통해 해시값을 생성한다.

public static void HMAC_SHA256_Transform(byte[] output, byte[] key, int keyLen, byte[] input, int inputLen)
매개변수 : - output 생성된 HMAC 값 - key MAC을 생성할 때 사용하는 키 - klen key의 바이트 길이 - input MAC을 생성할 원본 메시지 - inputLen 원본 메시지의 바이트 길이 **반환값 :** - 없음

6. 참조구현값

본 장에서는 HMAC-SHA256에 대한 구현적합성 실험을 위한 참조구현값(Test Vectors)을 제공한다.

HMAC 함수는 MAC 값을 생성하기 위해 메시지(M)과 키(KEY)가 필요하다. 여기서, 사용자로부터 받은 메시지에 대한 MAC 값(MAC)을 생성한다.

가. HMAC-SHA256 참조구현값

< 데이터 1 >

메시지(M)	548A457280851ECA0F5476AFDAC102CF6C7DBE09B3083D74FBD03DA31E9D7F27F42CD656111A7D4BB005AD2EEAED6FB62CE0B0EBE7D6933189DA0B82AD6AA8FB8E21B19AC29374462579DA0F130E3EB8DAB87F726EEB54EB5F4AE087091087ED0BAFFFC6FAB7AAC156F823DBBCEB17DD5E4E5626B10F29AA656BE73B9A57C308
키(KEY)	C6F1D667A50AAEBA5A200A0A7CC24FFBB24984426AB8ABACCEE75162F3E1646B
MAC 값(MAC)	96C37F36CA0DEA3B2B3E60F1F6CDF79CFF72CA2A43A091C8105AE882A690EF2F

< 데이터 2 >

메시지(M)	49B993F89E1E755D8C3CAA5133FC84B288D4B63206A3AE59A1DC25CEFCE7F4D2DBC4290DDBF25A8D618F390CD0C06971FF53909AEAA3AE59A7BCADBC9CC03992F08AD12A3F901E5E920E84D08E61F874EBB0114F28D2617E7D6C0579125A7B996E51B4D832C26AD90701B428D5A6D8C2363460D82AF870D00C34568DC47D63F0
키(KEY)	511FE863204DC8C72BCCB0194F4DA02EA0EA5B8E1609BA7783844525C8070451
MAC 값(MAC)	27F150ABDCBC478C83251A3F314CE609

< 데이터 3 >

메시지(M)	4905C90565FD39E95E6261AD9E3F2D6085CB0A871648401CF02B82D6807DC3AB76814C9475970C900F602FEBF023F2C05970B5BDD103512617EAAD1C5AEB8C20AC20A9D3B5E406FFA4F381DB6BCCA88B2C5ACEE7EED45F7D95E270D911019A04D528094A8BACD04006617F067802F8CAE5D27E8DD7840F2A5458F4940D88965F
키(KEY)	3F62F99E1C4EE604A8B0F0990D58B163C624A6BD56DD82573A5CC87E1CF1989E
MAC 값(MAC)	6B964756F341B64A1A0B4BCC744DC9AB

< 데이터 4 >

메시지(M)	2BC8406115A03A2CC3AC13E802D5244F3449F1B0FD6671F03FD9A9D04907806D4BACCE0D0DC1B618E1A43665D773FCB72FB87B88144C3F0587757D599952EF2EDD79DE711DDE81564CF7859DBC1DED1391361C5B769130EE8E119261AD8ECBE7C9789A16ABBD43EEEAED29171A9D9F6A4109FCB38AFDEDB53EF5106FEF83BE9E
키(KEY)	96860CE5CCB1678808C9F49DD77CBFD2B6A0554EC9161EA9D0F5ED3C56E4D69C
MAC 값(MAC)	CC935697032C301238D71C551337C0D38437931C85D5C6E03BDD2262F7D68F4F

< 데이터 5 >

메시지(M)	AB1AD9E68F172869F747994EDF0B1E91E3DA5C10F72C029258965A07C3361E12E4518CC4113BBCAB3877320B1620B06BAA874D53EA1ECB55B495264A074269DE73C6C54FCEEEE47129D77BF602561FECBC9659874C03FC213467B67FB553E65640C1AD0D2FF748C5B2AF9D970C74131CFF4FA73384A33DFEC056332E34313C81
키(KEY)	9B45670F9C5B1A57E657D940F023F4B393852A3BBFFF07086DF47039D40B7C5BE0D9852CE0BA7FA57FFF8938F54FC2577E475E350B64F9D038232A2F7F665CE8
MAC 값(MAC)	47EAE43DF17515FA042F9B9A4A049336D22779D8D1FD90F01412E788340631FB

< 데이터 6 >

메시지(M)	71E8105DD735E6E3712904D8DF3232B84536CC3EFB79146F85C0CCDBC5B705206F9032AFBEED80D365006AEFC9F087076D7FA781ADEB1F2DB011618498D8D02DFBCDD2C6970B26E0415E784B1FA83295178396CD87270A8C378FB2378CD7447EFEE91828CB587F13ED8C56764A20DB6A2C88BFA9F0F77C95027FE87C097B0A1A
키(KEY)	E4845C209CBCD1A7927EE49E5C50155698B2F6E496FD84255191B700A42F58EBBAA413B1C7A8CD72F50538EBA2FCA4F7C8E41856BC14A1AE1C03EACF02AE0D3B7E3AA9E644DB353AA1F4935EB480E8B8DAB0ACB16AD5FDB8A3813989E2E9CBAF188E45477C1DB0087D931F80FBA48BA0B65EDA7B070A414BCE53CCB5ABAA3B4E
MAC 값(MAC)	72179DF96A36AE7567CC64CE6F50149EEC01D6CA058F49856DEE52ABCA34CF47