

# SHA-256 해시함수에 대한 소스코드 활용 매뉴얼

2013.12.

## 제 · 개 정 이 력

순번	제 · 개정일	제 · 개정 내역	담당자
1	2013.12	“SHA-256 해시함수에 대한 소스코드 활용” 매뉴얼	김기문
2			
3			
4			

## < 목 차 >

1. 개요 .....	4
2. 해시 알고리즘 .....	4
3. 응용 프로그램 .....	5
3.1 C/C++ .....	5
3.1.1 프로젝트 생성 및 빌드 .....	5
3.1.2 소스코드 설명 .....	9
3.2 Java .....	11
3.2.1 프로젝트 생성 및 빌드 .....	11
3.2.2 소스코드 설명 .....	14
4. 웹 프로그램 .....	16
4.3 ASP .....	16
4.3.1 소스코드 추가 .....	16
4.3.2 소스코드 설명 .....	16
4.4 JSP .....	18
4.4.1 소스코드 추가 .....	18
4.4.2 소스코드 설명 .....	18
4.5 PHP .....	20
4.5.1 소스코드 추가 .....	20
4.5.2 소스코드 설명 .....	20
[부록] 참조구현값 .....	22

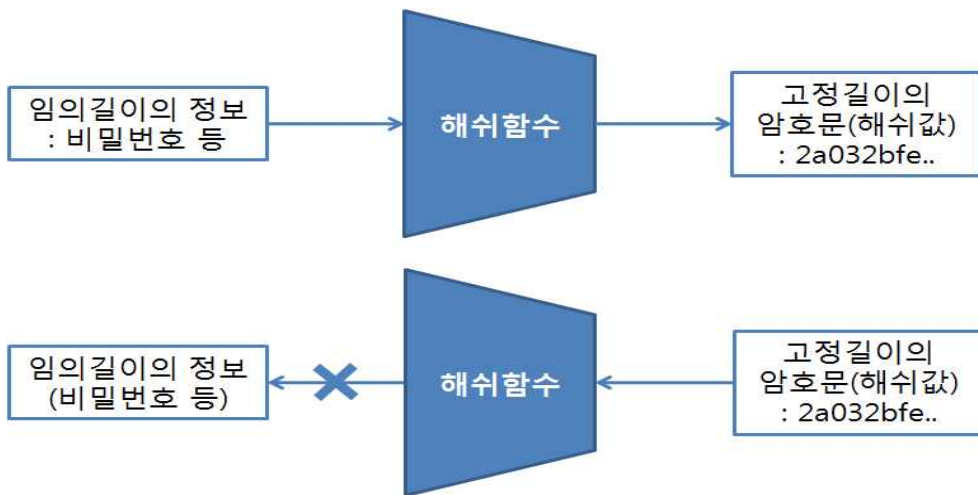
# 1. 개요

SHA(Secure Hash Algorithm)는 인터넷 뱅킹, 전자 서명, 메시지 인증 코드, 키 교환 알고리즘, 키 생성 알고리즘 등 다양한 분야의 보안 프로토콜에 사용이 된다. SHA 해시함수는 단방향 암호화로서 암호화 된 값을 복호화 하지 못한다.

본 매뉴얼은 SHA256 소스코드를 보다 쉽게 활용할 수 있도록 소스코드에 대한 설명과 함께 사용 시 주의사항을 다룬 매뉴얼이다.

# 2. 해시 알고리즘

해시함수는 임의길이의 정보를 입력으로 받아, 고정된 길이의 암호문(해시값)을 출력하는 암호기술로 암호화된 정보는 복호화가 불가능한 특징을 가지고 있다. 따라서 해시함수를 이용하면 아래와 같이 비밀번호를 입력하여 암호문(해시값)을 생성해 낼 수는 있지만, 암호문(해시값)을 가지고 원래의 비밀번호를 알아낼 수는 없다. 즉, 개인정보처리자도 시스템에 저장된 암호문(해시값)을 가지고 원래의 사용자 비밀번호를 알 수 없기 때문에 안전한 비밀번호 관리가 가능해 진다.



< 해시함수의 개념 >

### 3. 응용 프로그램

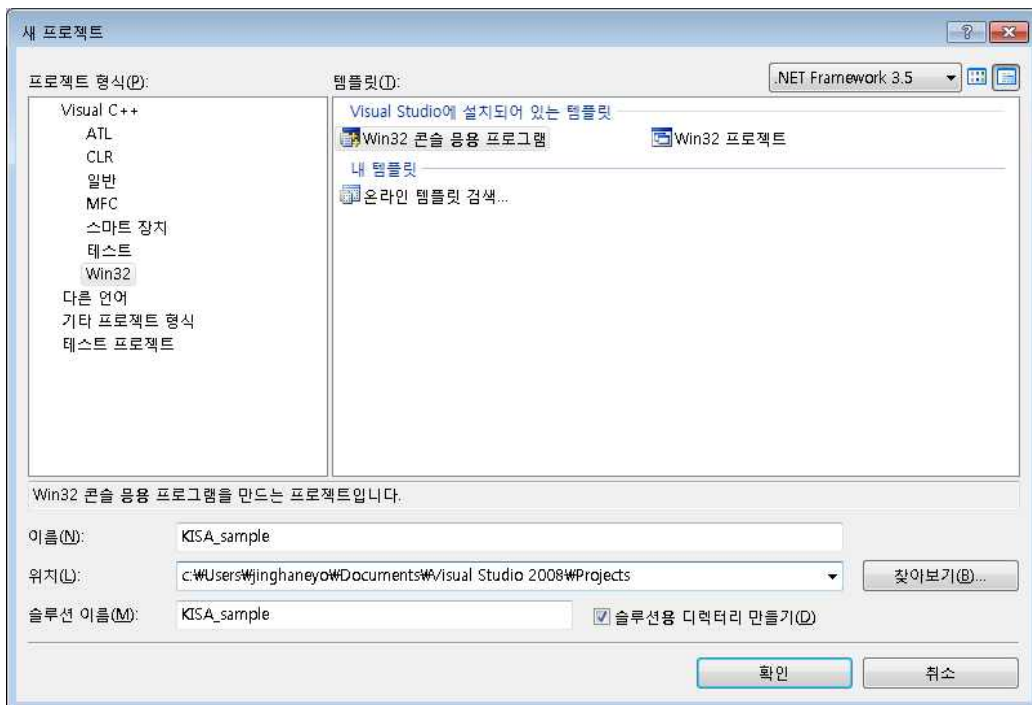
Windows, Linux 등 다양한 운영체제 환경에서 응용 프로그램을 개발과 함께 암호화를 적용할 수 있도록 해시함수에 대한 소스코드를 개발하였습니다. 기본적으로 많이 사용하는 C/C++ 및 Java에 대한 소스코드 설명이 제공된다.

#### 3.1. C/C++

프로젝트 생성 및 소스 추가, 빌드 등 배포되는 소스코드를 이용하여 해시를 실행하는 방법에 대해서는 Microsoft 社의 Visual studio 2008을 활용하여 설명하도록 한다.

##### 3.1.1 프로젝트 생성 및 빌드

Visual studio를 실행하여 “Win32 콘솔 응용 프로그램”을 선택한다. 이름에는 프로젝트 명을 기입하고 위치에는 생성시키고자 하는 곳의 위치를 지정하여 준다.

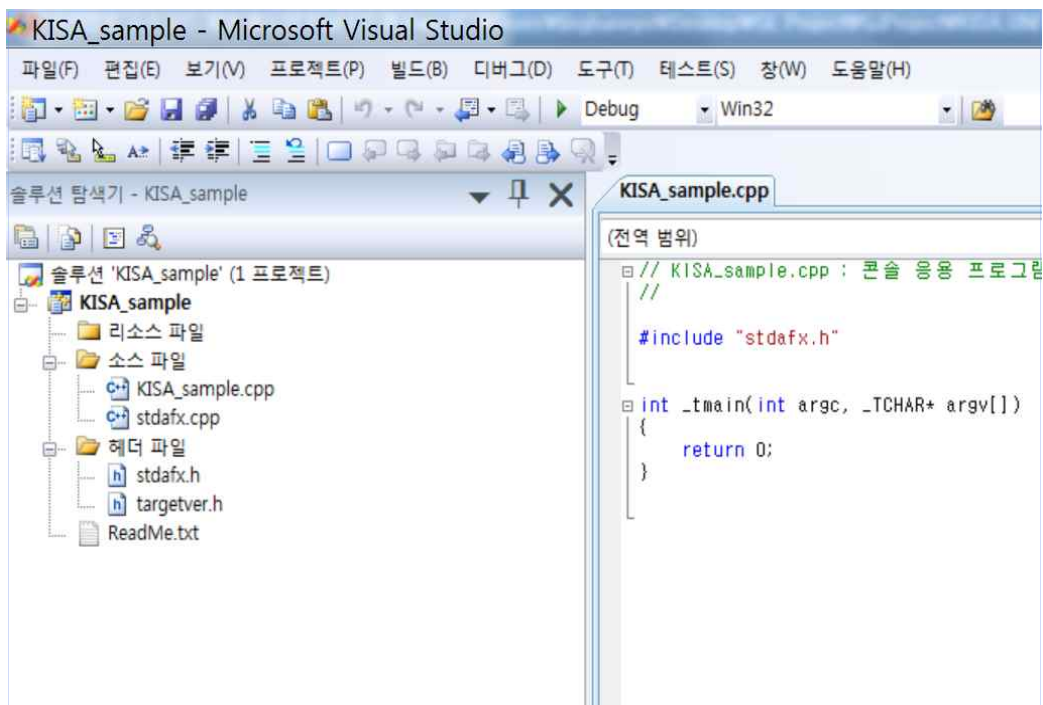


“콘솔 응용 프로그램”으로 선택한 후 마침 버튼을 누르면 콘솔 형태의 프로젝트가 생성된다. 미리 컴파일된 헤더를 체크해주면 `int _tmain( int argc, TCHAR *argv[] )`을 자동으로 생성시켜 준다.

빈 프로젝트는 말그대로 아무것도 없는 것으로 직접 헤더 파일과 소스파일을 추가하여 전부 작성하는 것이다. 여기서는 미리 컴파일된 헤더를 사용하여 프로젝트를 구성하도록 한다.

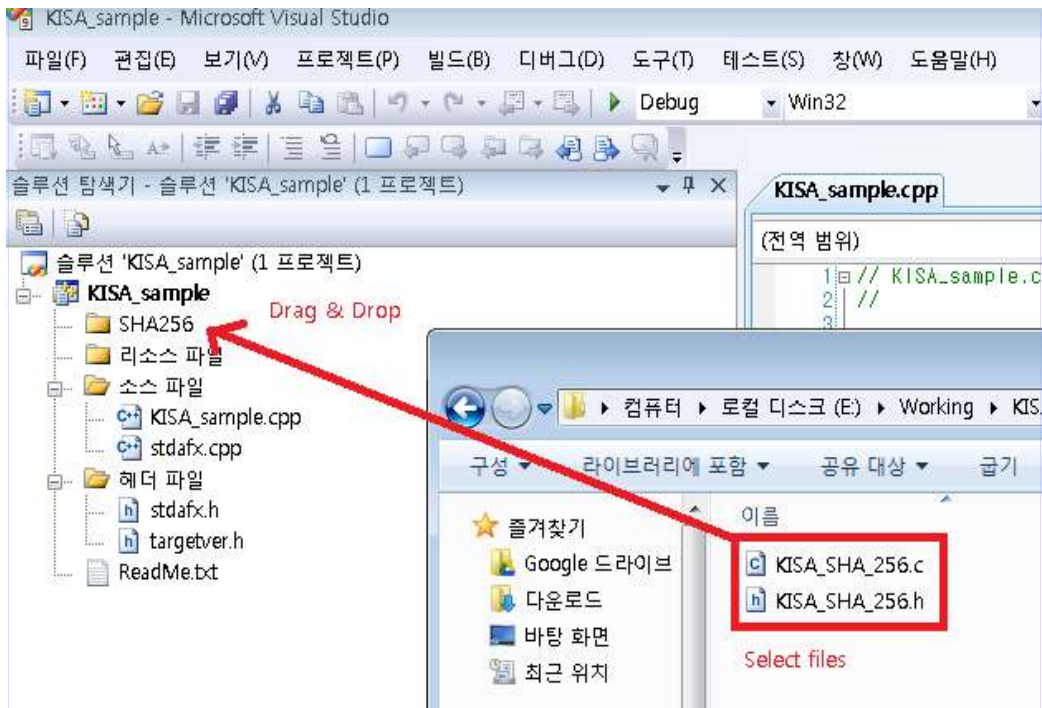


프로젝트 생성이 완료되면 “콘솔 응용 프로그램“을 작성할 수 있도록 기본적인 .h와 .cpp의 파일들을 생성된다. KISA\_sample.cpp의 \_tmain 함수 내에서 암호화/복호화 소스를 활용하도록 한다.

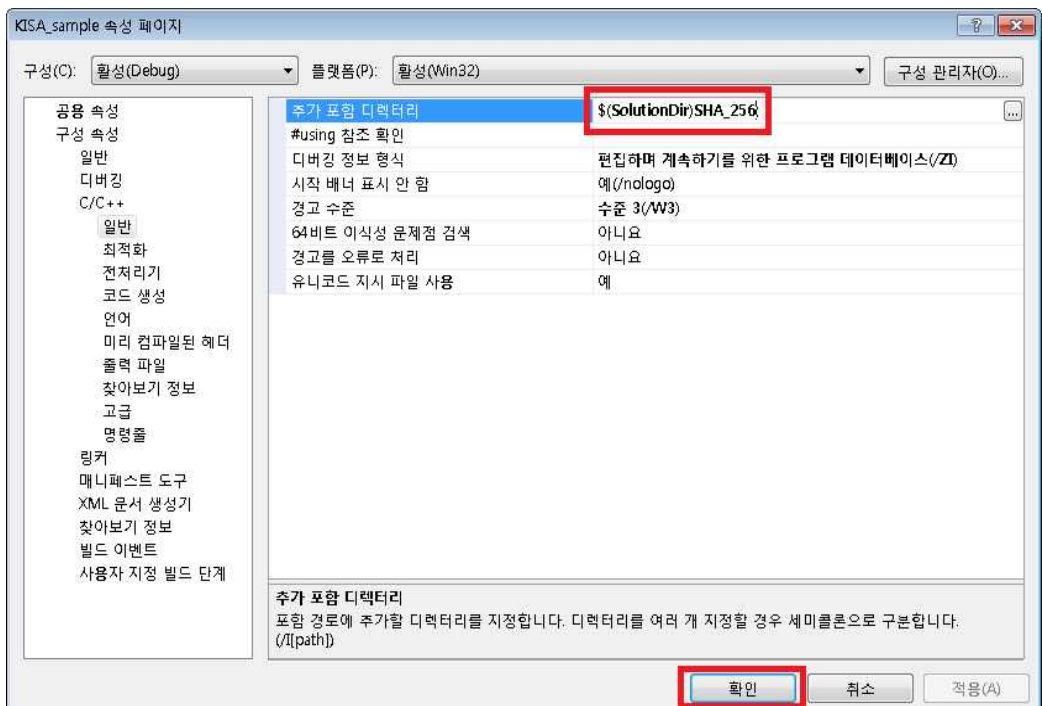


프로젝트 생성이 완료된 후, KISA\_sample 프로젝트 하위로 “SHA256“ 폴더를 추가하도록 한다.

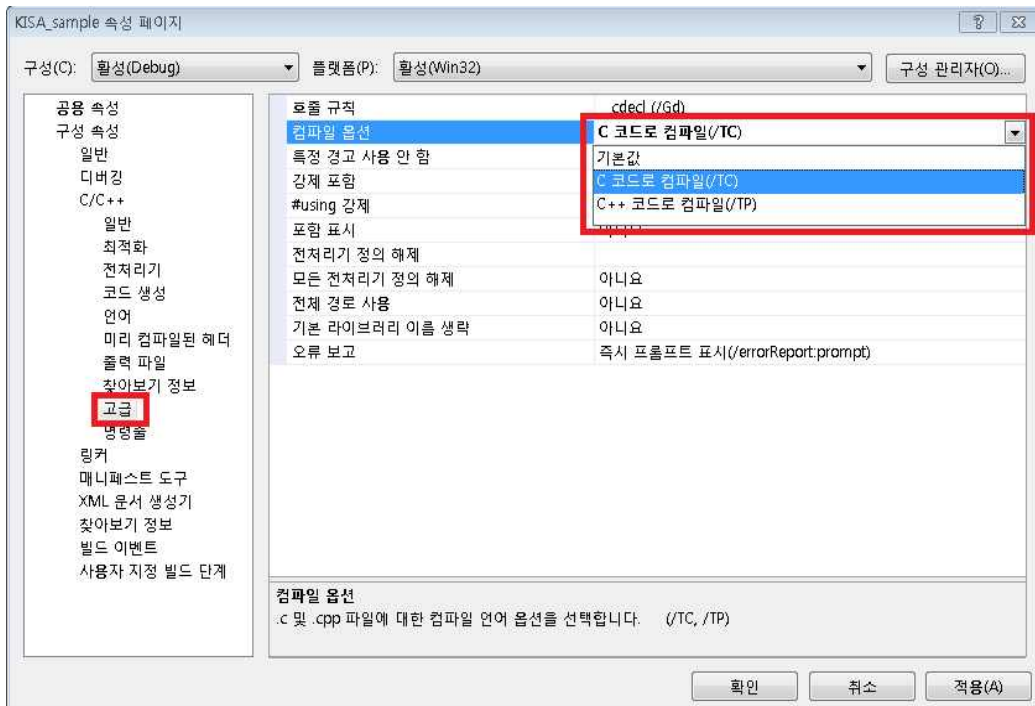
KISA에서 배포하는 소스코드 "KISA\_SHA256.c" 및 "KISA\_SHA256.h" 파일을 드래그 앤 드롭하여 해당 폴더로 복사한다.



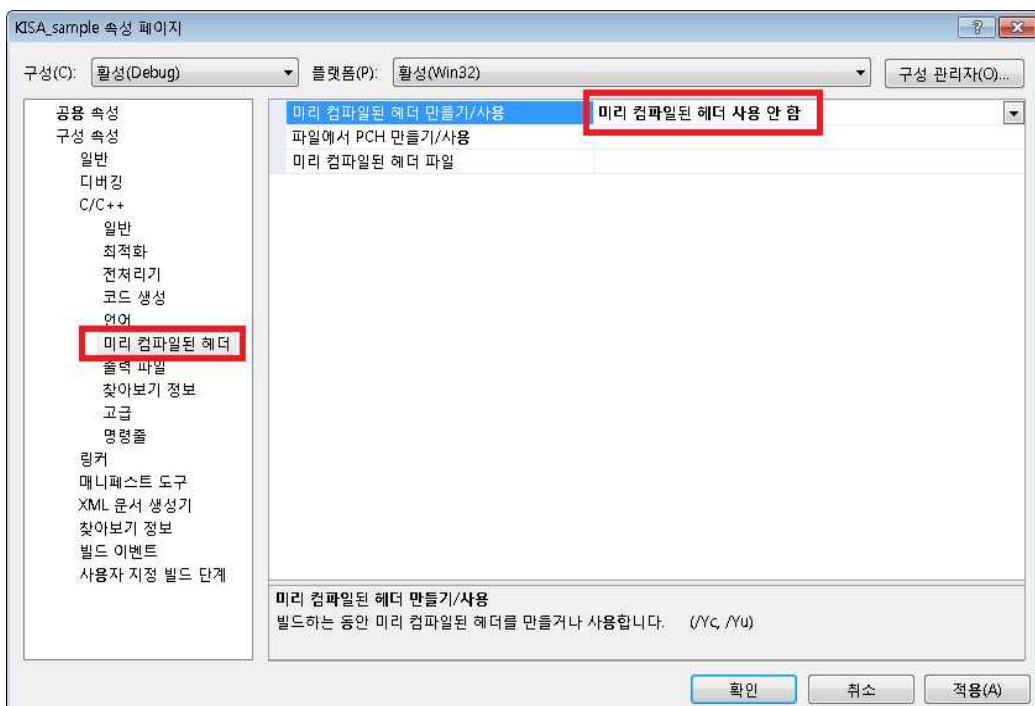
프로젝트 속성에서 C/C++의 추가 포함 디렉터리 항목에 포함 시킨 헤더와 소스 파일이 있는 폴더의 절대 경로 또는 상대 경로를 포함시켜야 한다. Visual studio에서 제공되는 매크로 함수를 이용한 상대 경로를 추가하였다. 매크로 관련해서는 도움말을 이용하고 여기서는 따로 설명을 하지 않는다. 여러 개의 폴더 또한 포함 시킬 수 있으며 구분자는 ";"로 하여 연속하여 기입 하면 된다.



또한, 고급에서 컴파일러 옵션을 "C 코드로 컴파일(/TC)"로 변경한다. C++로 해도 컴파일이 가능하나 배포되는 소스코드가 C이므로 C컴파일러로 구성하도록 한다.



마지막으로 .c 파일의 경우 “미리 컴파일된 헤더 사용 안 함”으로 설정하면 초기 빌드 할 수 있는 환경이 다 갖추어 졌다.





### 3.1.2 소스코드 설명

#### ☞ 함수 설명

```
void SHA256_Encrypt( IN const BYTE *pszPlainText, IN UINT pszMessage, OUT BYTE *pszDigest );
```

SHA256\_Init, SHA256\_Process, SHA256\_Close 함수를 내부적으로 모두 호출하여 간편한 인터페이스를 제공하기 위한 함수

#### 매개변수 :

pszMessage	입력 메시지의 포인터 변수
uPlainTextLen	입력 메시지의 바이트 길이
pszDigest	SHA-256 해시 값을 저장할 포인터 변수

#### 참 고 :

1. IN 사용자가 입력해야 할 매개변수를 의미 한다.
2. OUT 함수 호출 후 값이 채워지는 매개변수를 의미 한다.

```
void SHA256_Init( OUT SHA256_INFO *Info );
```

연쇄 변수와 길이 변수를 초기화 하는 함수

#### 매개변수 :

Info	SHA256_Process 호출 시 사용되는 구조체
------	------------------------------

#### 참 고 :

1. IN 사용자가 입력해야 할 매개변수를 의미 한다.
2. OUT 함수 호출 후 값이 채워지는 매개변수를 의미 한다.

```
void SHA256_Process( OUT SHA256_INFO *Info, IN const BYTE *pszMessage, IN UINT
uDataLen );
```

임의의 길이를 가지는 입력 메시지를 512비트 블록단위로 나누어 압축 함수를 호출하는 함수

**매개변수 :**

Info	SHA-256 구조체의 포인터 변수
pszMessage	입력 메시지의 포인터변수
uDataLen	입력 메시지의 바이트 길이

**참 고 :**

1. IN 사용자가 입력해야 할 매개변수를 의미 한다.
2. OUT 함수 호출 후 값이 채워지는 매개변수를 의미 한다.

```
void SHA256_Close( OUT SHA256_INFO *Info, OUT BYTE *pszDigest );
```

메시지 덧붙이기와 길이 덧붙이기를 수행한 후 마지막 메시지 블록을 가지고 압축 함수를 호출하는 함수

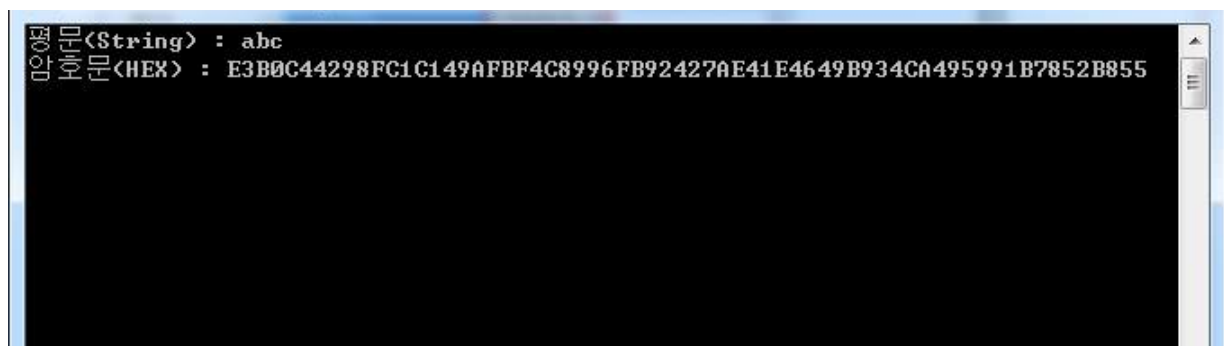
**매개변수 :**

Info	SHA-256 구조체의 포인터 변수
pszDigest	SHA-256 해시값을 저장할 포인터 변수

**참 고 :**

1. IN 사용자가 입력해야 할 매개변수를 의미 한다.
2. OUT 함수 호출 후 값이 채워지는 매개변수를 의미 한다.

☞ 테스트 화면

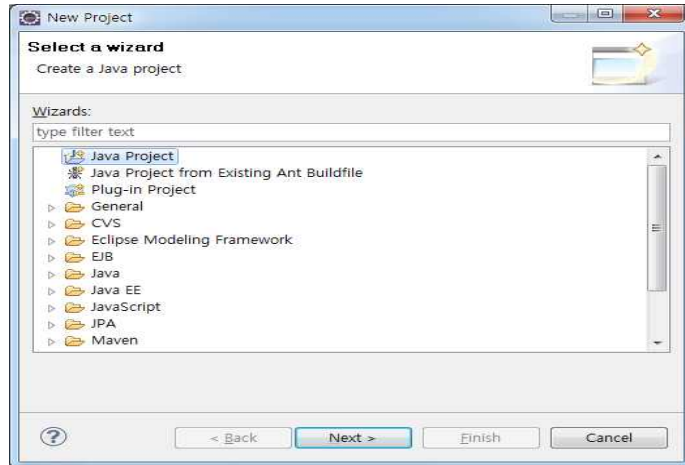


## 3.2. Java

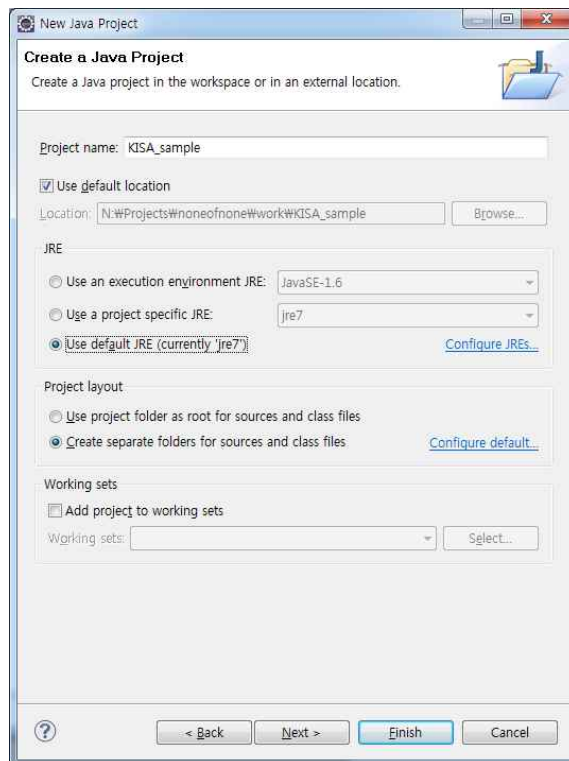
다음은 Java 형태의 프로그램을 하나 생성하여 헤더 및 소스와 파일 추가 하는 방법과 빌드하는 방법 등을 설명한다.

### 3.2.1 프로젝트 생성 및 빌드

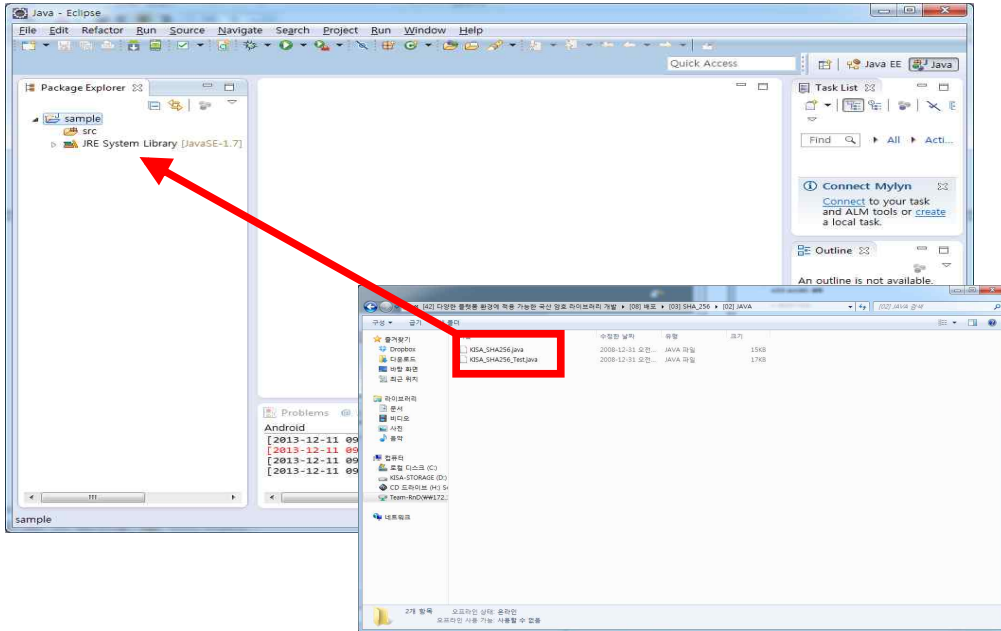
Eclipse를 실행하여 하기 이미지처럼 Java 프로젝트를 선택한다.



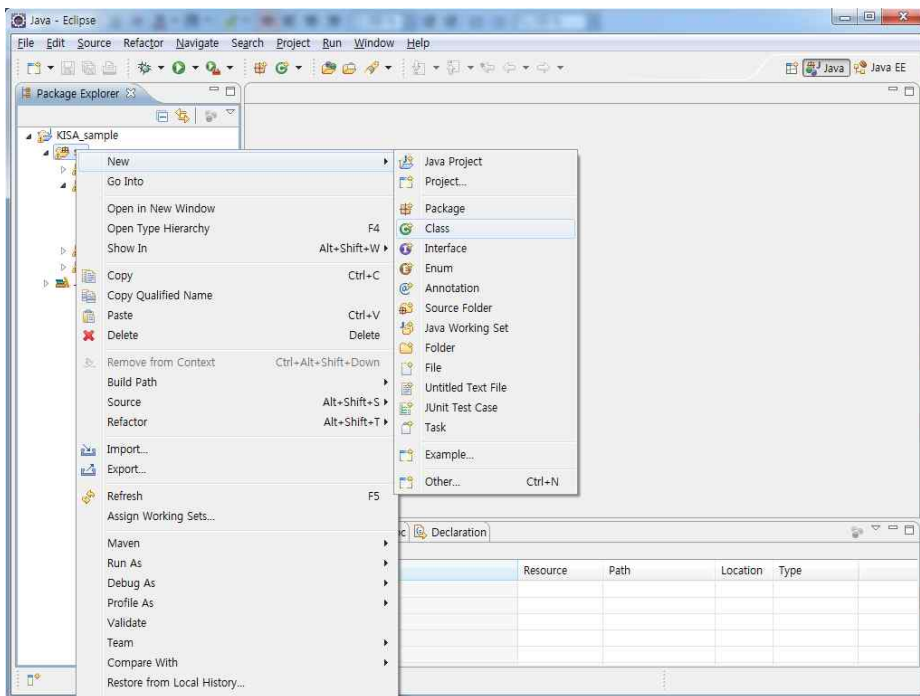
Next를 클릭하여 Project name에는 프로젝트 명을 기입한다. "Use default JRE" 를 선택 후 Finish 버튼을 눌러 프로젝트를 생성을 완료한다.



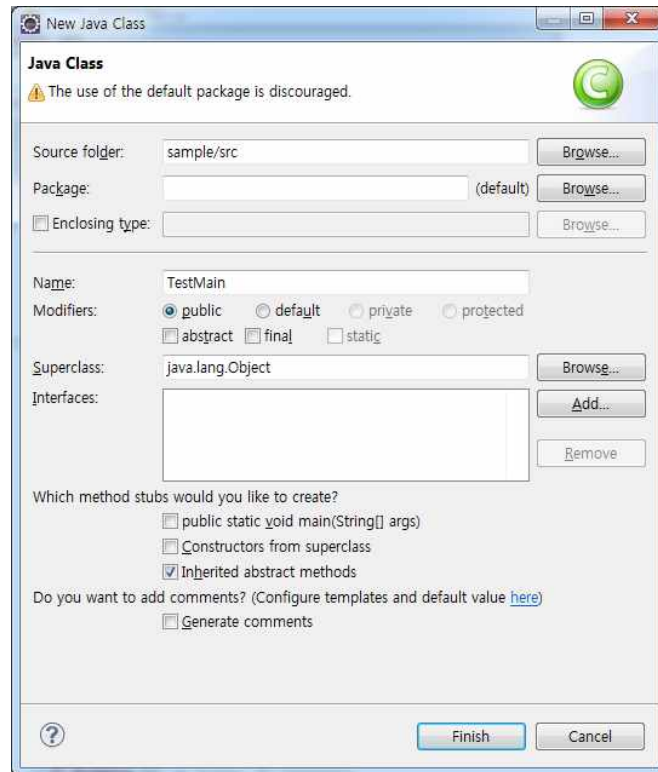
소스가 있는 폴더로 이동하여 프로젝트에 추가할 파일을 드래그&드롭으로 프로젝트로 이동시킨다. 이때 “Copy files and folders”를 선택하여 복사하여 준다.



테스트 클래스를 생성하여 배포 중인 소스를 사용할 수 있도록 구성한다. 먼저, src 폴더를 마우스로 우 클릭하여 New -> Class를 선택한다.

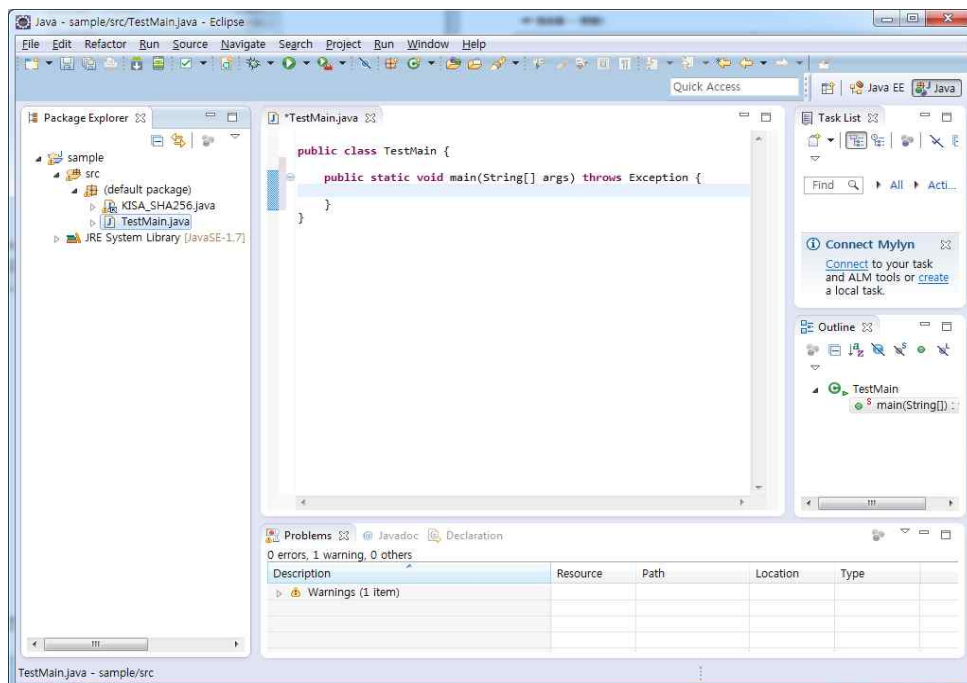


다음으로는 Package와 Name을 입력 후 Finish 버튼을 클릭하여 테스트 클래스를 생성한다.



생성된 빈 클래스 안에 아래와 같이 메인 함수를 추가 후 작업을 하면 된다.

```
public static void main(String[] args) throws Exception {  
}
```



### 3.2.2 소스코드 설명

#### ☞ 함수 설명

<b>public static void SHA256_Encrypt( byte[] pszMessage, int uPlainTextLen, byte[] pszDigest )</b>
SHA256_Init, SHA256_Process, SHA256_Close 함수를 내부적으로 모두 호출하여 간편한 인터페이스를 제공하기 위한 함수
<b>매개변수 :</b> pszMessage                    입력 메시지의 포인터변수 uPlainTextLen                입력 메시지의 바이트 길이 pszDigest                    SHA-256 해시 값을 저장할 포인터 변수

<b>public static void SHA256_Init( SHA256_INFO Info )</b>
연쇄 변수와 길이 변수를 초기화 하는 함수
<b>매개변수 :</b> Info                            SHA256_Process 호출 시 사용되는 구조체

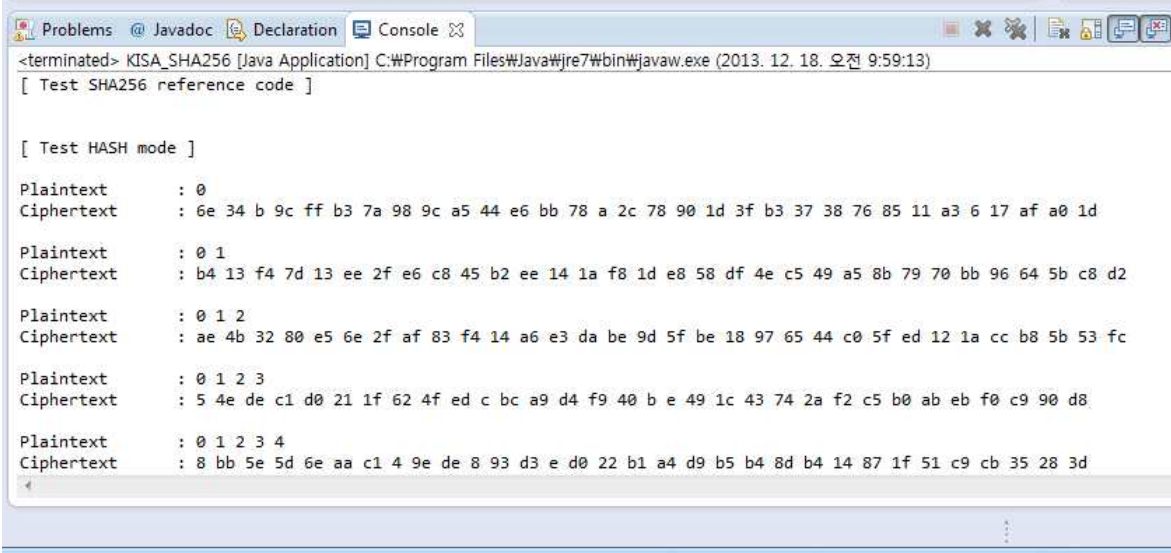
  

<b>public static void SHA256_Process( SHA256_INFO Info, byte[] pszMessage, int uDataLen )</b>
임의의 길이를 가지는 입력 메시지를 512비트 블록단위로 나누어 압축 함수를 호출하는 함수
<b>매개변수 :</b> Info                            SHA-256 구조체의 포인터 변수 pszMessage                    입력 메시지의 포인터변수 uDataLen                        입력 메시지의 바이트 길이

<b>public static void SHA256_Close( SHA256_INFO Info, byte[] pszDigest )</b>
메시지 덧붙이기와 길이 덧붙이기를 수행한 후 마지막 메시지 블록을 가지고 압축 함수를 호출하는 함수
<b>매개변수 :</b> Info                            SHA-256 구조체의 포인터 변수 pszDigest                      SHA-256 해시값을 저장할 포인터 변수

## 테스트 화면



The screenshot shows a Java IDE console window with the following content:

```
<terminated> KISA_SHA256 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (2013. 12. 18. 오전 9:59:13)
[ Test SHA256 reference code ]

[ Test HASH mode ]

Plaintext      : 0
Ciphertext     : 6e 34 b 9c ff b3 7a 98 9c a5 44 e6 bb 78 a 2c 78 90 1d 3f b3 37 38 76 85 11 a3 6 17 af a0 1d

Plaintext      : 0 1
Ciphertext     : b4 13 f4 7d 13 ee 2f e6 c8 45 b2 ee 14 1a f8 1d e8 58 df 4e c5 49 a5 8b 79 70 bb 96 64 5b c8 d2

Plaintext      : 0 1 2
Ciphertext     : ae 4b 32 80 e5 6e 2f af 83 f4 14 a6 e3 da be 9d 5f be 18 97 65 44 c0 5f ed 12 1a cc b8 5b 53 fc

Plaintext      : 0 1 2 3
Ciphertext     : 5 4e de c1 d0 21 1f 62 4f ed c bc a9 d4 f9 40 b e 49 1c 43 74 2a f2 c5 b0 ab eb f0 c9 90 d8

Plaintext      : 0 1 2 3 4
Ciphertext     : 8 bb 5e 5d 6e aa c1 4 9e de 8 93 d3 e d0 22 b1 a4 d9 b5 b4 8d b4 14 87 1f 51 c9 cb 35 28 3d
```

## 4. 웹 프로그램

Data Base, Homepage 등 다양한 Web Service 환경에서 전송·저장되는 구간의 암호화를 적용할 수 있도록 국산암호 소스코드를 개발하였습니다. 기본적으로 많이 사용하는 ASP, JSP, PHP를 기반으로 소스코드 등 활용방법에 대하여 설명한다.

### 4.1. ASP

소스 활용 등 배포되는 소스코드를 이용하여 해시함수를 실행하는 방법에 대해서는 간단한 에디터 상태에서 설명하도록 한다. 단, Windows Server 및 IIS 설정에 대한 설명은 생략하기로 한다.

#### 4.1.1 소스코드 추가

해시함수에 대한 소스코드가 작성된 파일을 사용하고자 하는 파일에 포함 시킨다.

```
<!--#include file="KISA_SHA256.asp"-->
<%response.Charset = "utf-8"%>
<%
Dim enc2

enc = request.form("ENC")
check = request.form("check")

If check = 1 then
    enc2 = test(enc)
End If

function test(str)
    test = SHA256_Encrypt(str)
end function

%>
```

#### 4.1.2 소스코드 설명

##### ☞ 함수 설명

#### Public Function SHA256\_Encrypt(sMessage)

SHA256\_Init, SHA256\_Process, SHA256\_Close 함수를 내부적으로 모두 호출하여 간편한 인터페이스를 제공하기 위한 함수

#### 매개변수 :

sMessage

입력 메시지 변수



### Public Function SHA256\_Init()

연쇄 변수와 길이 변수를 초기화 하는 함수

### Private Function SHA256\_Process(sMessage)

임의의 길이를 가지는 입력 메시지를 512비트 블록단위로 나누어 압축 함수를 호출하는 함수

매개변수 :

sMessage                    입력 메시지

### Public Function SHA256\_Close(HASH, M)

메시지 덧붙이기와 길이 덧붙이기를 수행한 후 마지막 메시지 블록을 가지고 압축 함수를 호출하는 함수

매개변수 :

HASH                        SHA256\_Init 호출하여 초기화된 변수

M                            SHA256\_Process 로 생성된 변수

☞ 테스트 페이지

## [SHA256] 테스트 페이지

<암호화 예제>

평문(String):

암호문(HEX):

## 4.2. JSP

소스 활용 등 배포되는 소스코드를 이용하여 해시함수를 실행하는 방법에 대해서는 간단한 에디터 상태에서 설명하도록 한다. 단, 운영체제 환경 및 Tomcat 설정에 대한 설명은 생략하기로 한다.

### 4.2.1 소스코드 추가

해시함수에 대한 소스코드가 작성된 파일을 사용하고자 하는 파일에 포함 시킨다.

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
pageEncoding="EUC-KR"%>
<%@ include file="KISA_SHA256.jsp" %>
<%!
public byte[] getBytes(String data) {
    String[] str = data.split(",");
    byte[] result = new byte[str.length];
    for(int i=0; i<result.length; i++) {
        result[i] = getHex(str[i]);
    }
    return result;
}

public String getString(byte[] data) {
    String result = "";
    for(int i=0; i<data.length; i++) {
        result = result + toHex(data[i]);
    }
    return result;
}

public byte getHex(String str) {
    str = str.trim();
    if(str.length() == 0)
        str = "00";
    else if(str.length() == 1)
        str = "0" + str;
}
```

### 4.2.2 소스코드 설명

#### ☞ 함수 설명

<code>public static void SHA256_Encrypt( byte[] pszMessage, int uPlainTextLen, byte[] pszDigest )</code>	
SHA256_Init, SHA256_Process, SHA256_Close 함수를 내부적으로 모두 호출하여 간편한 인터페이스를 제공하기 위한 함수	
<b>매개변수 :</b>	
pszMessage	입력 메시지의 포인터변수
uPlainTextLen	입력 메시지의 바이트 길이
pszDigest	SHA-256 해시 값을 저장할 포인터 변수

```
public static void SHA256_Init( SHA256_INFO Info )
```

연쇄 변수와 길이 변수를 초기화 하는 함수

매개변수 :

Info                                    SHA256\_Process 호출 시 사용되는 구조체

```
public static void SHA256_Process( SHA256_INFO Info, byte[] pszMessage, int uDataLen )
```

임의의 길이를 가지는 입력 메시지를 512비트 블록단위로 나누어 압축 함수를 호출하는 함수

매개변수 :

Info                                    SHA-256 구조체의 포인터 변수

pszMessage                            입력 메시지의 포인터변수

uDataLen                              입력 메시지의 바이트 길이

```
public static void SHA256_Close( SHA256_INFO Info, byte[] pszDigest )
```

메시지 덧붙이기와 길이 덧붙이기를 수행한 후 마지막 메시지 블록을 가지고 압축 함수를 호출하는 함수

매개변수 :

Info                                    SHA-256 구조체의 포인터 변수

pszDigest                              SHA-256 해시값을 저장할 포인터 변수

☞ 테스트 페이지

## [SHA256] 테스트 페이지

<암호화 예제>

평문(String):

암호문(HEX):

### 4.3. PHP

소스 활용 등 배포되는 소스코드를 이용하여 해시함수를 실행하는 방법에 대해서는 간단한 에디터 상태에서 설명하도록 한다. 단, Apache 서버 설정에 대한 설명은 생략하기로 한다.

#### 4.3.1 소스코드 추가

해시함수에 대한 소스코드가 작성된 파일을 사용하고자 하는 파일에 포함 시킨다.

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

function encrypt($str) {
    $planBytes = array_slice(unpack('c*', $str), 0);
    $ret = null;
    $bszChiperText = null;
    KISA_SEED_SHA256::SHA256_Encrypt($planBytes, count($planBytes), $bszChiperText);
    $r = count($bszChiperText);

    foreach($bszChiperText as $encryptedString) {
        $ret .= bin2hex(chr($encryptedString));
    }
    return $ret;
}

$enc = $_POST['ENC'];
$check = $_POST['check'];

if($enc != NULL || $check == 1)
{
    $g_bszPlainText = $enc;
    $enc2 = encrypt($g_bszPlainText);
}
```

#### 4.3.2 소스코드 설명

##### o SHA256

<b>static function SHA256_Encrypt( &amp;\$pszMessage, \$uPlainTextLen, &amp;\$pszDigest )</b>	
SHA256_Init, SHA256_Process, SHA256_Close 함수를 내부적으로 모두 호출하여 간편한 인터페이스를 제공하기 위한 함수	
<b>매개변수 :</b>	
\$pszMessage	입력 메시지의 포인터변수
\$uPlainTextLen	입력 메시지의 바이트 길이
\$pszDigest	SHA-256 해시 값을 저장할 포인터 변수

`static function SHA256_Init( &$Info )`

연쇄 변수와 길이 변수를 초기화 하는 함수

매개변수 :

\$Info                                    SHA256\_Process 호출 시 사용되는 구조체

`static function SHA256_Process(&$Info, &$pszMessage, $uDataLen)`

임의의 길이를 가지는 입력 메시지를 512비트 블록단위로 나누어 압축 함수를 호출하는 함수

매개변수 :

\$Info                                    SHA-256 구조체의 포인터 변수

\$pszMessage                            입력 메시지의 포인터변수

\$uDataLen                                입력 메시지의 바이트 길이

`static function SHA256_Close( &$Info, &$pszDigest )`

메시지 덧붙이기와 길이 덧붙이기를 수행한 후 마지막 메시지 블록을 가지고 압축 함수를 호출하는 함수

매개변수 :

\$Info                                    SHA-256 구조체의 포인터 변수

\$pszDigest                              SHA-256 해시값을 저장할 포인터 변수

☞ 테스트 페이지

## [SHA256] 테스트 페이지

<암호화 예제>

평문(String):

암호문(HEX):

## [부록] 참조구현값

본 SHA-256 소스코드 매뉴얼에서는 평문 메시지 2가지에 대하여 아래와 같이 참조구현값(Test Vectors)이 제공된다.

### [데이터 1]

Input Message		"abc"
Initial hash value	H[0]	6a09e667
	H[1]	bb67ae85
	H[2]	3c6ef372
	H[3]	a54ff53a
	H[4]	510e527f
	H[5]	9b05688c
	H[6]	1f83d9ab
	H[7]	5be0cd19
Digest		BA7816BF 8F01CFEA 414140DE 5DAE2223 B00361A3 96177A9C B410FF61 F20015AD

### [데이터 2]

Input Message		"abcdbcdecdefdefgefghfghighijhi jki jkl jklmklmnlmnomnopnopq"
Initial hash value	H[0]	6a09e667
	H[1]	bb67ae85
	H[2]	3c6ef372
	H[3]	a54ff53a
	H[4]	510e527f
	H[5]	9b05688c
	H[6]	1f83d9ab
	H[7]	5be0cd19
Digest		248D6A61 D20638B8 E5C02693 0C3E6039 A33CE459 64FF2167 F6ECEDD4 19DB06C1