암호프로그래밍
CRYPTOGRAPHY PROGRAMMING

9. 전자서명

정보보호학과
이병천 교수

차례

9. 전자서명

1. 전자서명
2. 웹, 자바스크립트
3. 파이썬

1. 전자서명이란?

- 서명의 변천

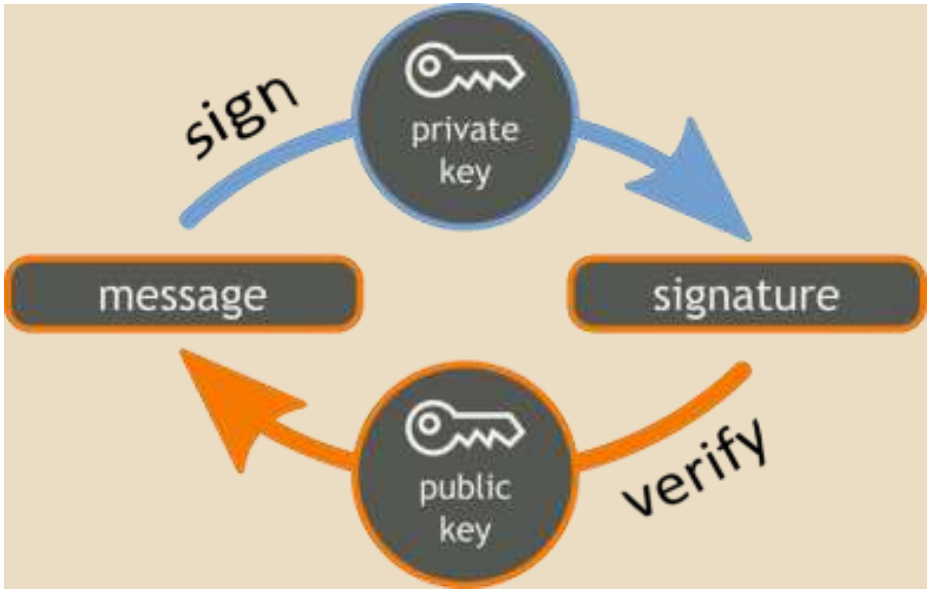
전통적인 서명

전자문서에 대한 서명

전자서명이란? 암호기술을 이용하여 전자문서에 대한 서명기능을 구현한 것

전자서명이란?

- 전자서명

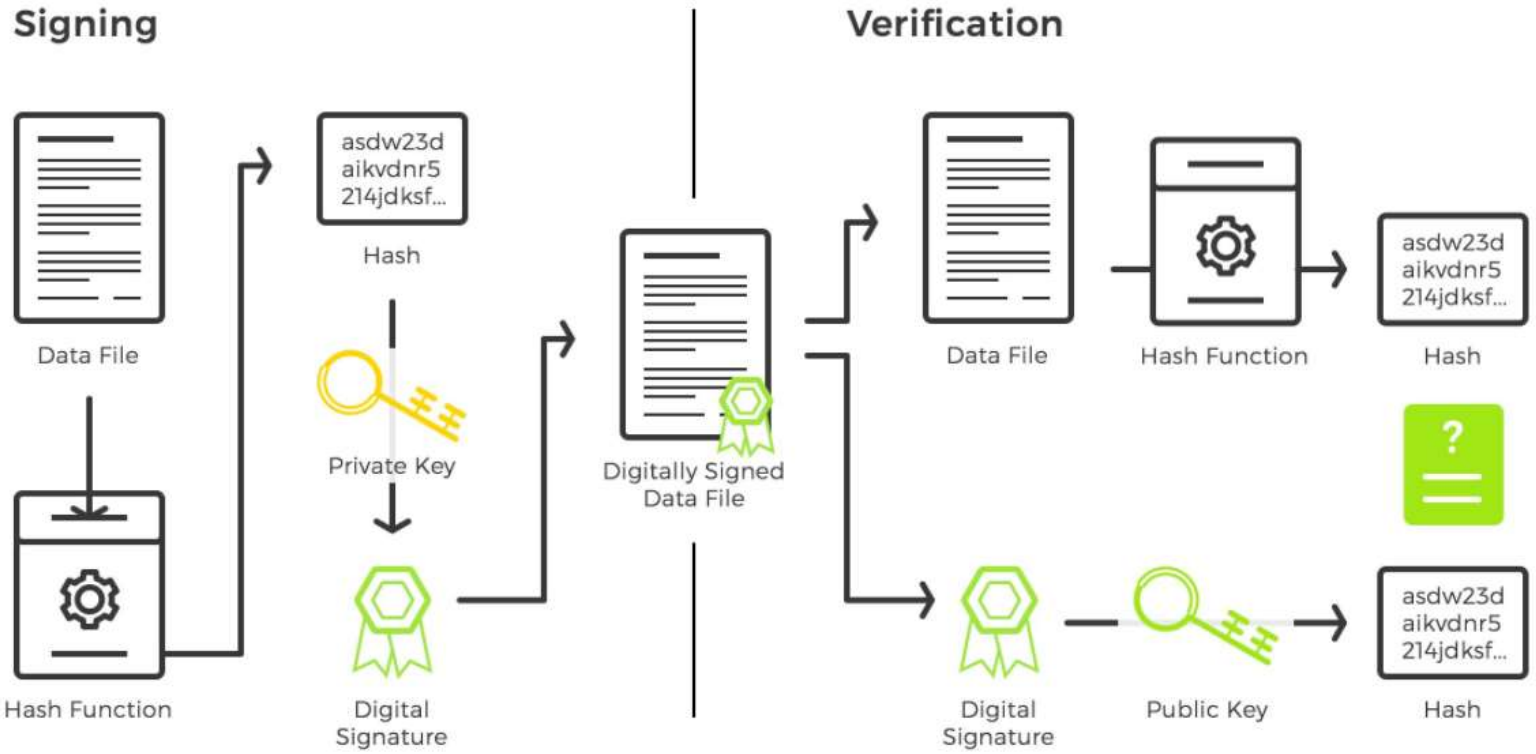


전자서명은 개인키를 가진 사용자만이 생성 가능
(부인방지 기능 제공)

공개키를 가진 누구나 서명의 유효성을 검증 가능

전자서명이란?

- 전자서명



해시값에 대한 서명을 문서에 대한 서명으로 인정함
해시함수의 역상저항성, 충돌저항성 특징을 이용

전자서명법

전자서명의 효력을 법적으로 인정

전자서명법

[시행 2013.3.23] [법률 제11690호, 2013.3.23, 타법개정]

미래창조과학부 (정보보호정책과) 02-2110-2929

제1장 총칙

제1조(목적) 이 법은 전자문서의 안전성과 신뢰성을 확보하고 그 이용을 활성화하기 위하여 전자서명에 관한 기본적인 사항을 정함으로써 국가사회의 정보화를 촉진하고 국민생활의 편익을 증진함을 목적으로 한다.

제2조(정의) 이 법에서 사용하는 용어의 정의는 다음과 같다.

1. "전자문서"라 함은 정보처리시스템에 의하여 전자적 형태로 작성되어 송신 또는 수신되거나 저장된 정보를 말한다.
2. "전자서명"이라 함은 서명자를 확인하고 서명자가 당해 전자문서에 서명을 하였음을 나타내는데 이용하기 위하여 당해 전자문서에 첨부되거나 논리적으로 결합된 전자적 형태의 정보를 말한다.
3. "공인전자서명"이라 함은 다음 각목의 요건을 갖추고 공인인증서에 기초한 전자서명을 말한다.
   가. 전자서명생성정보가 가입자에게 유일하게 속할 것
   나. 서명 당시 가입자가 전자서명생성정보를 지배·관리하고 있을 것
   다. 전자서명이 있은 후에 당해 전자서명에 대한 변경여부를 확인할 수 있을 것
   라. 전자서명이 있은 후에 당해 전자문서의 변경여부를 확인할 수 있을 것
4. "전자서명생성정보"라 함은 전자서명을 생성하기 위하여 이용하는 전자적 정보를 말한다.
5. "전자서명검증정보"라 함은 전자서명을 검증하기 위하여 이용하는 전자적 정보를 말한다.
6. "인증"이라 함은 전자서명생성정보가 가입자에게 유일하게 속한다는 사실을 확인하고 이를 증명하는 행위를 말한다.

제3조(전자서명의 효력 등) ①다른 법령에서 문서 또는 서면에 서명, 서명날인 또는 기명날인을 요하는 경우 전자문서에 공인전자서명이 있는 때에는 이를 충족한 것으로 본다. <개정 2001.12.31>

②공인전자서명이 있는 경우에는 당해 전자서명이 서명자의 서명, 서명날인 또는 기명날인이고, 당해 전자문서가 전자서명된 후 그 내용이 변경되지 아니하였다고 추정한다.<개정 2001.12.31>

③공인전자서명외의 전자서명은 당사자간의 약정에 따른 서명, 서명날인 또는 기명날인으로서의 효력을 가진다.

<신설 2001.12.31>

[제목개정 2001.12.31]

전자서명의 요구조건

위조불가	개인키를 소유한 서명자만이 서명문 생성 가능
변경불가	서명된 문서의 내용 변경 불가
서명자인증	개인키를 소유한 자가 전자서명의 행위자임
재사용불가	A문서의 전자서명을 B문서의 전자서명으로 사용 불가
부인방지	서명자는 전자서명 후에 행위에 대한 부인 불가

전자서명의 활용

- 사용자 인증
  - 로그인 등에서 사용자 확인을 위해 사용
  - 로그인 메시지를 사용자의 개인키로 서명하여 전송, 서버에서 사용자의 인증서(공개키)로 검증
  - 인터넷뱅킹, 전자정부서비스 로그인

- 메시지 인증
  - 메시지에 전자서명을 부가하여 사용자의 서명 사실을 증명
  - 수신자는 사용자의 공개키로 서명을 검증
  - 부인방지 기능 제공

전자서명

- 수기서명과 전자서명의 비교

항목	수기서명	디지털 서명
서명 결과	고정	변화
디지털 복제	어려움	쉬움
서명 과정	간단	수학적 연산
법적 효력	있음	있음
위조	가능	불가능
서명 도구	필기구	컴퓨터
보조 수단	불필요	필요 (해쉬함수)

전자서명 알고리즘

- RSA
- ElGamal
- DSA
  - NIST에서 1991년 미국전자서명 표준으로 제안
- Nyberg-Rueppel
- KCDSA
  - 한국의 전자서명 표준
- ECDSA
  - 타원곡선기반 DSA 전자서명
- GQ
- Schnorr

RSA 전자서명

- RSA 전자서명은 RSA 공개키 암호화의 반대 연산
  - 서명생성(sign) : 개인키 이용
  - 서명검증(verify) : 공개키 이용
- 전자서명은 해쉬함수와 함께 사용
  - 전자서명은 공개키암호 연산을 사용하므로 속도가 느림.
  - 메시지 전체에 대해 서명연산을 하지 않고 메시지의 해쉬값에 대해 서명연산을 1회만 수행
  - 계산량과 통신량을 크게 줄일 수 있음
- 서명자의 신분 인증?
  - 인증기관이 발행한 인증서를 이용하여 서명자의 신분 인증

RSA 전자서명 데모

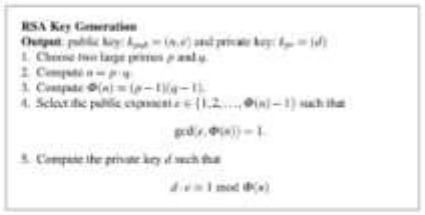
- https://kjur.github.io/jsrsasign/sample-rsasign.html

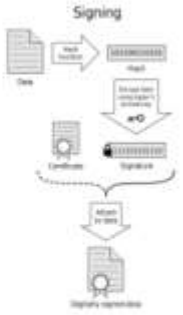
RSA 전자서명 데모

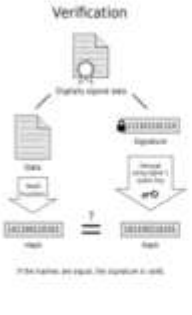
http://cris.joongbu.ac.kr/course/2019-1/wp/crypto/rsasig.html

RSA 전자서명

RSA는 공개키 암호시스템의 하나로, 암호화뿐만 아니라 전자서명이 가능한 최초의 알고리즘으로 알려져 있다. RSA가 갖는 전자서명 기능은 인증을 요구하는 전자 상거래 등에 RSA의 광범위한 활용을 가능하게 하였다. 1978년 로널드 라이베스트(Ron Rivest), 아디 샤미르(Adi Shamir), 레너드 애들먼(Leonard Adleman)의 연구에 의해 체계화되었으며, RSA라는 이름은 이들 3명의 이름 앞글자를 딴 것이다. 이 세 발명자는 이 공로로 2002년 튜링상을 수상했다.







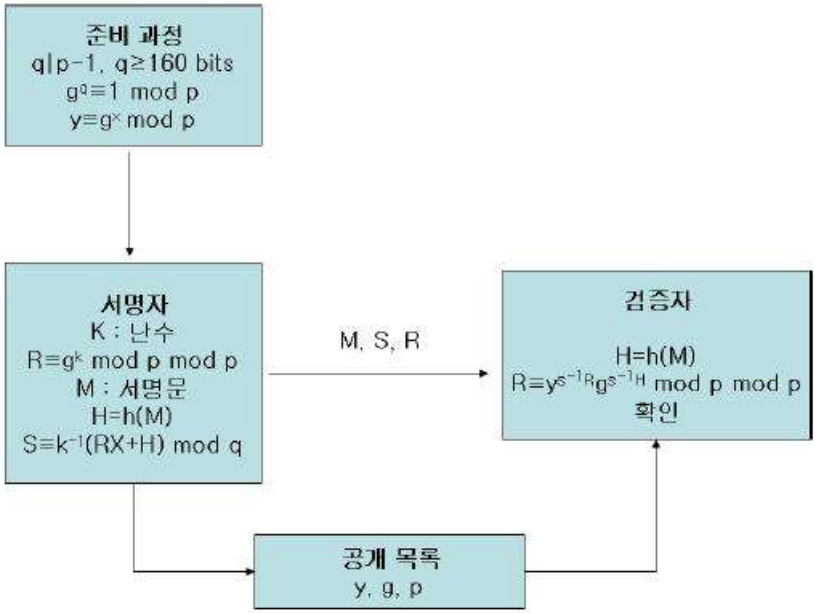
Key Length 1024 난수 키생성 (송신자, 서명자)

송신자 공개키

송신자 개인키

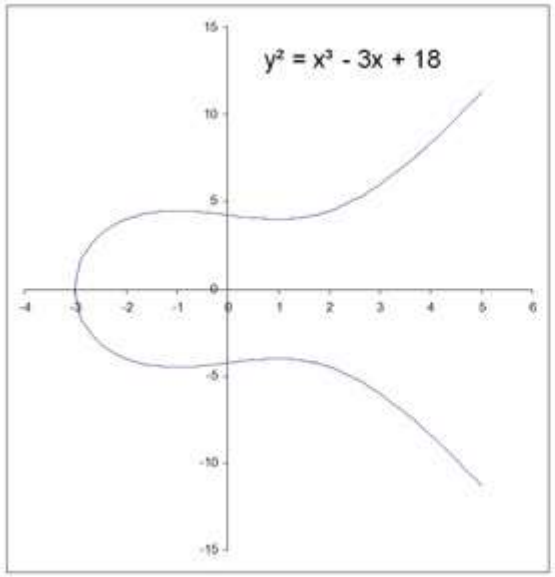
DSA (Digital Signature Algorithm)

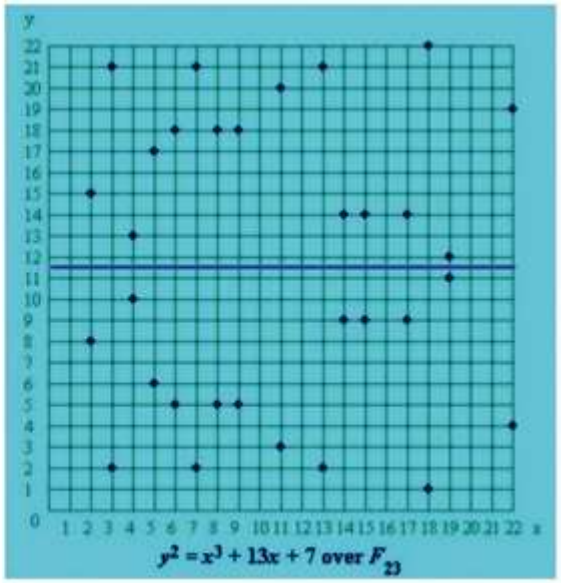
- 1991년 NIST에서는 DSS에서 사용할 DSA을 제안.
- DSS는 전자서명의 표준 이름이고 이 표준안에서 사용되는 알고리즘이 DSA임. (미국 연방 표준 서명 알고리즘) □
- 이산대수문제의 어려움을 안정성의 바탕으로 삼음
  - Y=g^x mod p 에서 Y, g, p을 공개하더라도 x를 계산하기 어렵다는 가정 □



ECDSA

- 타원곡선 전자서명 알고리즘(Elliptic Curve Digital Signature Algorithm)





2. 웹, 자바스크립트에서의 전자서명

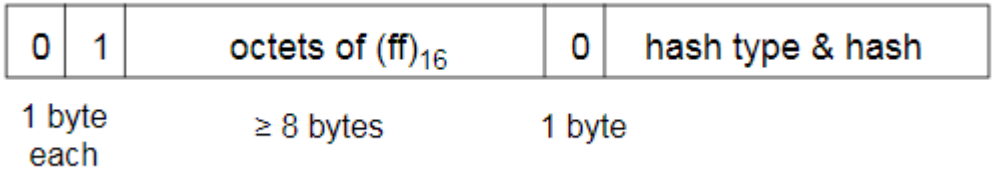
- node-forge에서의 전자서명
  - 서명 생성(개인키 이용): privateKey.sign()
  - 서명 검증(공개키 이용): publicKey.verify()

- 난수화 전자서명의 필요성
  - RSA 전자서명은 난수요소를 포함시키지 않으면 같은 메시지에 대해 항상 동일한 서명값을 출력.
  - 공격자가 동일한 메시지에 대한 서명임을 인지 가능

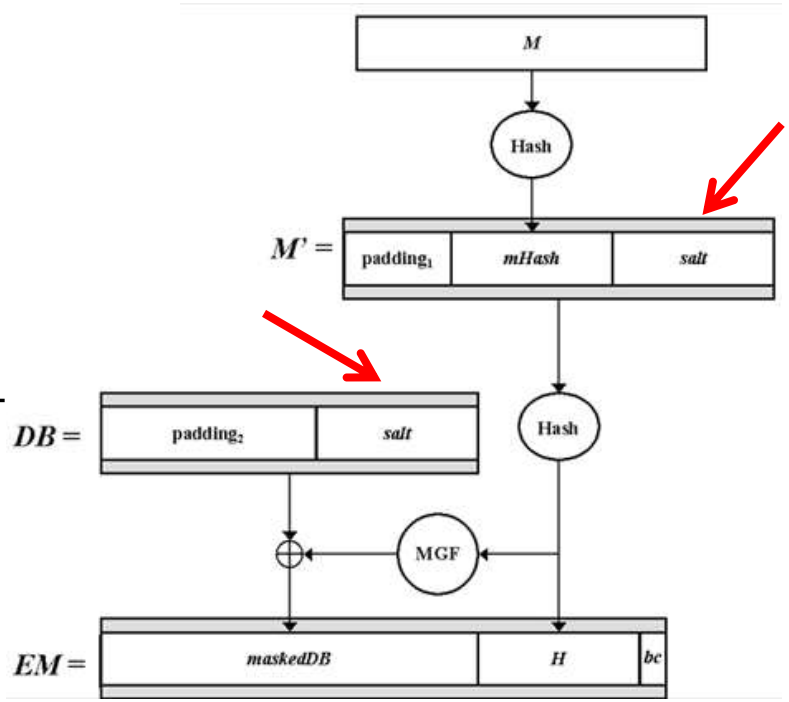
- RSA 전자서명 알고리즘 종류
  - RSASSA PKCS#1 v1.5
  - RSASSA-PSS

난수화된 RSA 전자서명

- RSASSA PKCS#1 v1.5
  - PKCS#1 v1.5 패딩으로 난수요소 추가한 후 전자서명



- RSA/PSS
  - Probabilistic Signature Scheme
  - Salt를 이용한 난수화 패딩 이용
  - Salt를 수신자에게 안전하게 전달



RSA 전자서명

1. 키생성

```
var forge = require('node-forge');
var plaintext = "Hello world hello world";

var rsa = forge.pki.rsa;

// 1. RSA 키생성
var keypair = rsa.generateKeyPair({bits: 1024, e: 0x10001}); // e = 65537
var publicKey = keypair.publicKey;
var privateKey = keypair.privateKey;
console.log('Public key: ₩n'+forge.pki.publicKeyToPem(publicKey));
console.log('Private key: ₩n'+forge.pki.publicKeyToPem(privateKey));
console.log();
```

```
Public key:
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCxK9wEh+UEfjGLc3GUd0DxEH7u
kuGCW26KztWkUIjXdBamm3DOdznOx/FRJEe6zBOJV2yLSJkfogJTff40LJzctIAf
ONPsFolsyf/23ki39+q1poa58R8cyeQgZ3hJmWIE8HGcZ3xXScA8E/F5W7xPlKCN
jO7lA3KoZLCkO3irpwIDAQAB
-----END PUBLIC KEY-----

Private key:
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCxK9wEh+UEfjGLc3GUd0DxEH7u
kuGCW26KztWkUIjXdBamm3DOdznOx/FRJEe6zBOJV2yLSJkfogJTff40LJzctIAf
ONPsFolsyf/23ki39+q1poa58R8cyeQgZ3hJmWIE8HGcZ3xXScA8E/F5W7xPlKCN
jO7lA3KoZLCkO3irpwIDAQAB
-----END PUBLIC KEY-----
```

RSA 전자서명

2. RSASSA PKCS#1 v1.5 전자서명

```
// 2. RSA 전자서명. 디폴트는 RSASSA PKCS#1 v1.5
console.log("RSASSA PKCS#1 v1.5");
// sign data with a private key and output DigestInfo DER-encoded bytes
// (defaults to RSASSA PKCS#1 v1.5)
var md = forge.md.sha1.create();
md.update(plaintext, 'utf8');
var signature = privateKey.sign(md);
console.log('Signature: '+forge.util.bytesToHex(signature));
// verify data with a public key
// (defaults to RSASSA PKCS#1 v1.5)
var verified = publicKey.verify(md.digest().bytes(), signature);
console.log('Verification: '+verified);
console.log();
```

개인키를 이용한 전자서명 생성

공개키를 이용한 전자서명 검증

```
RSASSA PKCS#1 v1.5
Signature:
7a62f0caee05a5cdadbe4d62c5a28889ab99883523b2665360a60fa5ddcec36ccdfb73f44
ea8ea0d715316ffbfa92a1491779742c1b708dc2468b56318b6679bc34b9fe9ce8171a72
87c64fd87381ccedb4bceacc808adfb0a828b4ed1e5a1edb610867f747a382c4d68a83ff5
6b8c273e8de3bda1d2263683152f3a2192bc5f
Verification: true
```

RSA 전자서명

3. RSASSA-PSS 전자서명

```
// 3. RSA 전자서명. RSASSA-PSS 방식
console.log("RSASSA-PSS");
// sign data using RSASSA-PSS where PSS uses a SHA-1 hash, a SHA-1 based
// masking function MGF1, and a 20 byte salt
var md = forge.md.sha1.create();
md.update(plaintext, 'utf8');
var pss = forge.pss.create({
  md: forge.md.sha1.create(),
  mgf: forge.mgf.mgf1.create(forge.md.sha1.create()),
  saltLength: 20
  // optionally pass 'prng' with a custom PRNG implementation
  // optionalls pass 'salt' with a forge.util.ByteBuffer w/custom salt
});
var signature = privateKey.sign(md, pss);
console.log('Signature: '+forge.util.bytesToHex(signature));
// verify RSASSA-PSS signature
var pss = forge.pss.create({
  md: forge.md.sha1.create(),
  mgf: forge.mgf.mgf1.create(forge.md.sha1.create()),
  saltLength: 20
  // optionally pass 'prng' with a custom PRNG implementation
});
var md = forge.md.sha1.create();
md.update(plaintext, 'utf8');
var verified = publicKey.verify(md.digest().getBytes(), signature, pss);
console.log('Verification: '+verified);
```

```
RSASSA-PSS
Signature:
2675fde10ddf80c03a5f458d479f38371519
b50aa7e32bc791c3136dcbb320e78ca0ccde
08e1e198fcdb7dce94bd5a9487fa65bb109d
9d3c5999ae7ed99df4e1fd12bd8b1a9a0b1
6ceb4d1b5a2650896cb2cda8b187a538647
9fe8920258fd5ba6931107bd3428e812764
b9a86c25917f5e3144acc428b22837a899fa
2b756b1
Verification: true
```

ECDSA 전자서명

- **EdDSA**
  - **Edwards-curve Digital Signature Algorithm**
  - 에드워드 커브 타원곡선 전자서명 알고리즘
- ED25519
  - EdDSA signature scheme using SHA-512 (SHA-2) and Curve25519

ECDSA 전자서명


```
const forge = require("node-forge");
var ed25519 = forge.pki.ed25519;

// generate a random ED25519 keypair
var keypair = ed25519.generateKeyPair();
var privateKey = keypair.privateKey;
var publicKey = keypair.publicKey;

// sign a UTF-8 message
var signature1 = ed25519.sign({
  message: "test",
  encoding: "utf8",
  privateKey: privateKey,
});

// sign a message passed as a buffer
var signature2 = ed25519.sign({
  message: Buffer.from("test", "utf8"),
  privateKey: privateKey,
});

// sign a message digest (shorter "message" == better performance)
var md = forge.md.sha256.create();
md.update("test", "utf8");
var signature3 = ed25519.sign({
  md: md,
  privateKey: privateKey,
});
```

```
// verify a signature on a UTF-8 message
var verified1 = ed25519.verify({
  message: "test",
  encoding: "utf8",
  signature: signature1,
  publicKey: publicKey,
});

// sign a message passed as a buffer
var verified2 = ed25519.verify({
  message: Buffer.from("test", "utf8"),
  signature: signature2,
  publicKey: publicKey,
});

// verify a signature on a message digest
var md = forge.md.sha256.create();
md.update("test", "utf8");
var verified3 = ed25519.verify({
  md: md,
  signature: signature3,
  publicKey: publicKey,
});

console.log("Signature1: " + forge.util.bytesToHex(signature1));
console.log("Verified1: " + verified1);
console.log("Signature2: " + forge.util.bytesToHex(signature2));
console.log("Verified2: " + verified2);
console.log("Signature3: " + forge.util.bytesToHex(signature3));
console.log("Verified3: " + verified3);
```

3. 파이썬에서의 전자서명

- Crypto.Signature.pkcs1_15
- Crypto.Signature.pss

- 서명 생성
  - h = SHA256.new(messageUtf8)
  - signature = pkcs1_15.new(key).sign(h)
- 서명 검증
  - h = SHA256.new(messageUtf8)
  - pkcs1_15.new(key.publickey()).verify(h, signature)

RSA 전자서명

1. RSA 키생성

```
from Crypto.Signature import pkcs1_15
from Crypto.Hash import SHA256
from Crypto.PublicKey import RSA
from base64 import b64encode, b64decode
from Crypto.Signature import pss
from Crypto import Random

# 1. RSA 키쌍 생성
key = RSA.generate(2048)  # bit수를 파라메터로 입력

# 공개키 암호의 키는 오랜 기간 사용하는 것을 목표로 하며 파일로 저장하여 사용

# RSA 개인키를 파일로 저장
private_key = key.export_key()              # 키쌍에서 개인키 추출 (PEM 형식)
file_out = open("privateKey.pem", "wb")
file_out.write(private_key)
file_out.close()
print("RSA 개인키: ₩n", private_key)
print()

# RSA 공개키를 파일로 저장
# RSA 공개키는 일반적으로 인증서에 포함되어 배포됨. 타인에게 공개할 수 있음.
public_key = key.publickey().export_key()  # 키쌍에서 공개키 생성 및 추출 (PEM 형식)
file_out = open("publicKey.pem", "wb")
file_out.write(public_key)
file_out.close()
print("RSA 공개키: ₩n", public_key)
print()
```

RSA 전자서명

2. RSA PKCS#1 v1.5 전자서명

```
# 서명할 메시지
message = 'To be signed. 한글메시지.【速報】大阪府で過去最多1260人の感染確認. 央视评副处长体验送外卖累瘫街头.'
messageUtf8 = message.encode("utf-8")

# 2. RSA 전자서명: PKCS#1 v1.5
# 서명 생성
h = SHA256.new(messageUtf8)
signature = pkcs1_15.new(key).sign(h)
sig_b64 = b64encode(signature).decode('utf-8')
print("RSA 전자서명 (PKCS#1 v1.5): ₩n", sig_b64)
print()

# 서명 검증
h = SHA256.new(messageUtf8)
try:
    pkcs1_15.new(key.publickey()).verify(h, signature)
    print("The signature is valid.")
except (ValueError, TypeError):
    print("The signature is not valid.")
print()
```

개인키를 이용한 전자서명 생성

공개키를 이용한 전자서명 검증

RSA 전자서명

3. RSA-PSS 전자서명

```
# 3. RSA 전자서명: PSS
# 서명 생성
h = SHA256.new(messageUtf8)
signature = pss.new(key).sign(h)
sig_b64 = b64encode(signature).decode('utf-8')
print("RSA 전자서명 (PSS): ₩n", sig_b64)
print()

# 서명 검증
h = SHA256.new(messageUtf8)
try:
    pss.new(key.publickey()).verify(h, signature)
    print("The signature is authentic.")
except (ValueError, TypeError):
    print("The signature is not authentic.")
```

개인키를 이용한
전자서명 생성

공개키를 이용한
전자서명 검증