암호프로그래밍
CRYPTOGRAPHY PROGRAMMING

6. 패스워드기반 키생성

정보보호학과
이병천 교수

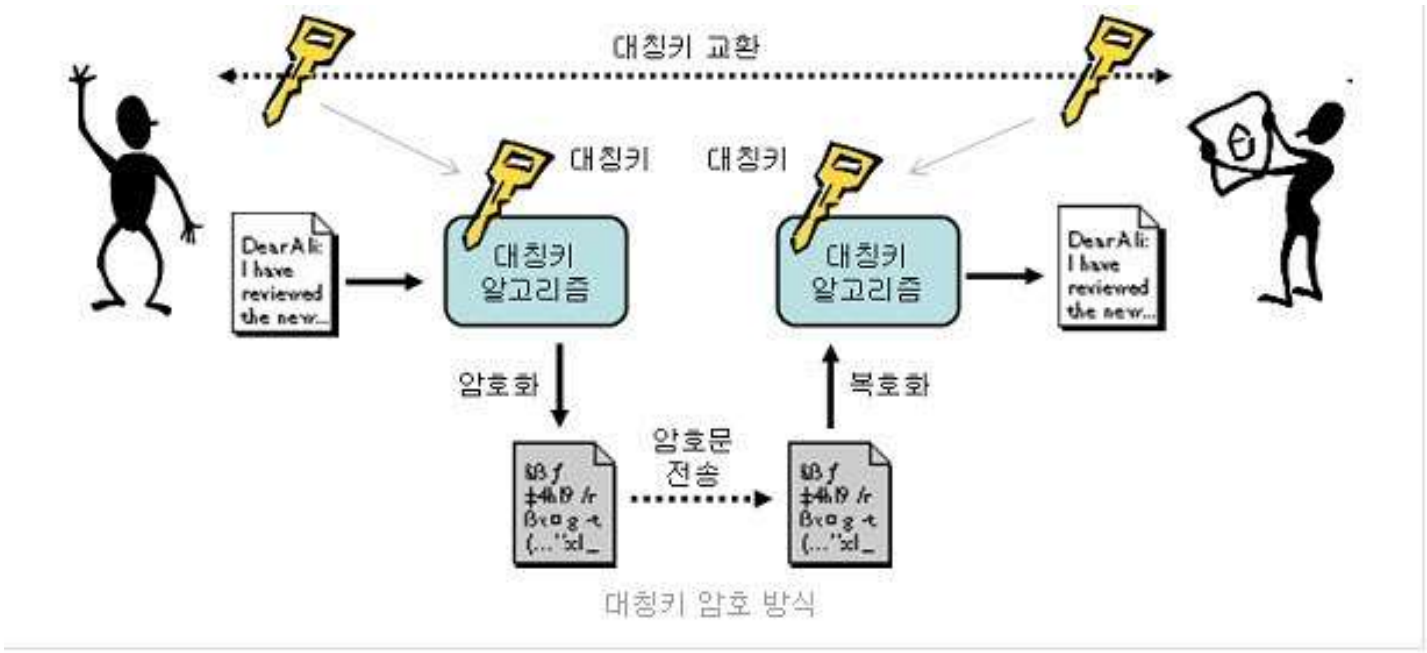
차례

6. 패스워드기반 키생성

1. 패스워드기반 키생성
2. 패스워드 해시 저장
3. 웹, 자바스크립트
4. 파이썬

1. 패스워드기반 키생성

- 암호알고리즘과 비밀키
  - 암호의 안전성은 비밀키의 안전함에 의존
  - 공격자가 예측할 수 없는 난수키를 사용하는 것이 중요



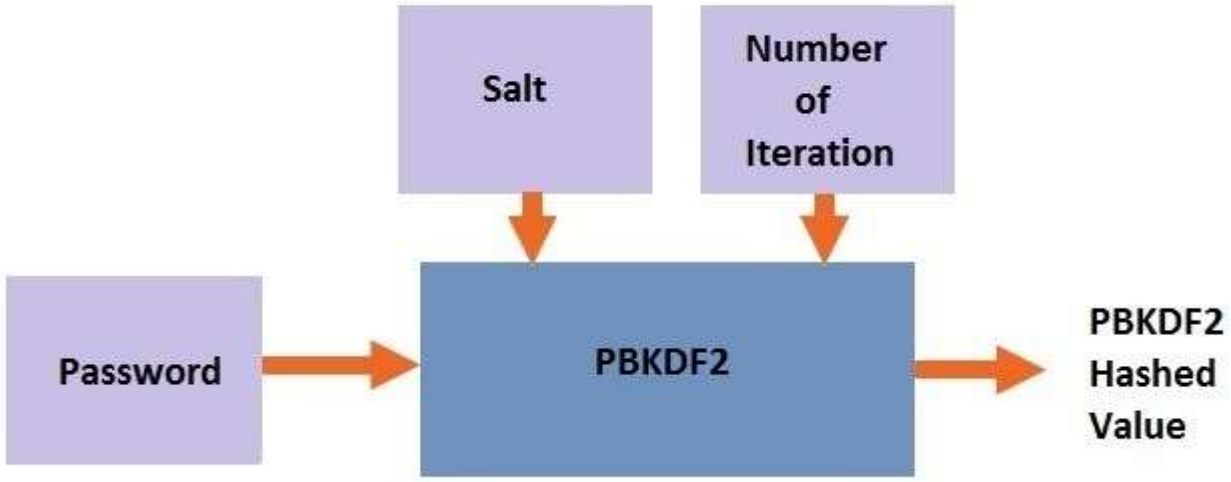
대칭키 암호 방식

패스워드기반 키생성

- 패스워드와 비밀키
  - 사용자가 입력하는 패스워드를 직접 암호알고리즘의 비밀키로 사용하는 것은 추측 가능한 키를 사용하게 되므로 위험
  - 무작위 대입공격, 사전공격 가능
  - 난수화된 비밀키를 사용해야 함
- 패스워드기반 암호화의 필요성
  - 사용자가 기억하는 패스워드를 키로 사용하려는 요구사항
  - 패스워드로부터 난수화된 비밀키를 생성하여 사용하는 방법이 필요

패스워드기반 키생성

- 패스워드 기반 키생성함수 (PBKDF2)
  - 패스워드로부터 난수화된 비밀키를 생성하는 방법
  - Password-Based Key Derivation Function v2
  - (1)사용자 입력 패스워드, (2)랜덤한 salt값, (3)반복횟수(iteration) 값을 이용하여 난수처럼 보이는 비밀키를 생성하여 사용
  - salt값과 반복횟수 값은 공격자의 사전공격을 어렵게 하는 중요한 요소



패스워드기반키생성 - PBKDF2

- 1. 사용자 입력 패스워드
  - 사용자가 입력하는 패스워드
  - 기억할 수 있는 정보로서 엔트로피가 높지 않음
- 2. 솔트(salt)
  - 난수로 생성하는 값
  - 공격자의 사전공격(dictionary attack)을 방지하기 위한 정보
  - 동일한 패스워드를 사용해도 솔트 때문에 다른 결과가 출력됨
- 3. 반복횟수(iteration)
  - 공격자의 공격비용을 증가시키기 위해 사용
  - 사용자는 한번만 계산하면 되지만 사전공격을 하는 공격자는 찾을때까지 반복계산 필요
- 4. 출력 키길이(dkLen)

DK = PBKDF2(Password, Salt, it, dkLen)

패스워드기반키생성 - PBKDF2

- PKCS#5 에 정의됨
  - password-based key-derivation function
  - forge.pkcs5.pbkdf2 객체 제공
  - 사용자입력 패스워드, 난수솔트, 반복횟수, 출력키길이 지정
  - 동기식/비동기식 함수 제공 – 반복횟수가 크면 계산시간이 길어져 다수 사용자 환경에서는 비동기식 계산 필요
  - AES의 16-byte (128비트) 키 생성 사례

```
// generate a password-based 16-byte key (동기식 키생성)
// note an optional message digest can be passed as the final parameter
var salt = forge.random.getBytesSync(128);
var derivedKey = forge.pkcs5.pbkdf2('password', salt, numIterations, 16);

// generate key asynchronously  (비동기식 키생성)
// note an optional message digest can be passed before the callback
forge.pkcs5.pbkdf2('password', salt, numIterations, 16, function(err, derivedKey) {
  // do something w/derivedKey
});
```

패스워드기반키생성 - PBKDF2

- 동기식 vs. 비동기식 프로그래밍

패스워드기반키생성 - PBKDF2

pbkdf2.js

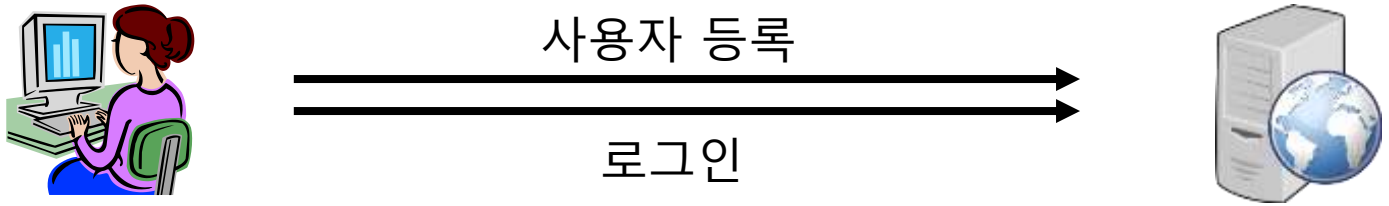
```
var forge = require('node-forge');
var salt;
var numIterations = 1000;

// generate a password-based 16-byte key
// note an optional message digest can be passed as the final parameter
salt = forge.random.getBytesSync(128);
var derivedKey = forge.pkcs5.pbkdf2('password', salt, numIterations, 16);
console.log('Derived key - sync: ', forge.util.bytesToHex(derivedKey));

// generate key asynchronously
// note an optional message digest can be passed before the callback
salt = forge.random.getBytesSync(128);
forge.pkcs5.pbkdf2('password', salt, numIterations, 32, function(err, derivedKey) {
  // do something w/derivedKey
  console.log('Derived key - async: ', forge.util.bytesToHex(derivedKey));
});
```

```
F:₩AppliedCrypto₩forge>node pbkdf2.js
Derived key - sync:  5f9c7c78b4915e42de4009a3d0a50961
Derived key - async:  c34131c94472106c86588f2b20c4e755b2651deed0f82ea65f532a7a4570b9d4

F:₩AppliedCrypto₩forge>node pbkdf2.js
Derived key - sync:  2468407e2b00b081844b9e02aa9378d2
Derived key - async:  a8bcc2d0c65f159048bbdcc994b540df68e6bde4a64318fadbde63852cb0285e

F:₩AppliedCrypto₩forge>node pbkdf2.js
Derived key - sync:  aab0daeeb5613f0c17a0014120d9154f
Derived key - async:  bb970bb17cf036d4fb90b2b859e3c5ea260bf51eea4dac825de4a6656facc288
```

난수 솔트를 이용하므로 동일한 패스워드에 대해 서로 다른 키가 생성됨

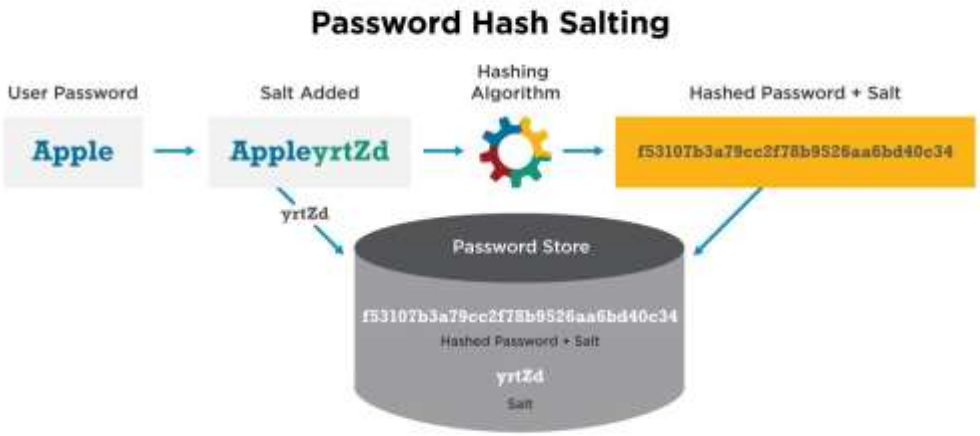
2. 패스워드 해시

- 패스워드 해시
  - 사용자의 로그인정보를 서버에 안전하게 저장하는 방법
  - 서버 관리자에 의한 사용자 패스워드 노출 방지
  - 서버 해킹시 해커에게 패스워드 노출 방지



패스워드 해시

- 서버에 저장되는 패스워드 해시




```
$2b$10$nOUIs5kJ7naTuTFkBy1veuK0kSxUFXfuaOKdOKf9xYT0KKIGSJwFa
 |  |  |                     |
 |  |  |                     hash-value = K0kSxUFXfuaOKdOKf9xYT0KKIGSJwFa
 |  |  |
 |  |  salt = nOUIs5kJ7naTuTFkBy1veu
 |  |
 |  cost-factor => 10 = 2^10 rounds
 |
 hash-algorithm identifier => 2b = BCrypt
```

패스워드 해시

- Bcrypt
  - password hashing function
  - https://www.npmjs.com/package/bcrypt
  - 패키지 설치
    - > npm install bcrypt

패스워드 해시

사용자 등록

```
bcrypt.hash(myPlaintextPassword, saltRounds, function(err, hash) {
    // Store hash in your password DB.
});
```

```
$2b$10$nOUIs5kJ7naTuTFkBy1veuK0kSxUFXfuaOKdOKf9xYT0KKIGSJwFa
 |  |  |                     |
 |  |  |                     hash-value = K0kSxUFXfuaOKdOKf9xYT0KKIGSJwFa
 |  |  |
 |  |  salt = nOUIs5kJ7naTuTFkBy1veu
 |  |
 |  cost-factor => 10 = 2^10 rounds
 |
 hash-algorithm identifier => 2b = BCrypt
```

패스워드 해시
DB 저장

입력 패스워드

사용자 로그인

```
// Load hash from your password DB.
bcrypt.compare(myPlaintextPassword, hash, function(err, result) {
    // result == true
});
```

패스워드 해시

비동기식

async (recommended)

```
const bcrypt = require('bcrypt');
const saltRounds = 10;
const myPlaintextPassword = 's0/₩/₩P4$$w0rD';
const someOtherPlaintextPassword = 'not_bacon';
```

To hash a password:

Technique 1 (generate a salt and hash on separate function calls):

```
bcrypt.genSalt(saltRounds, function(err, salt) {
    bcrypt.hash(myPlaintextPassword, salt, function(err, hash) {
        // Store hash in your password DB.
    });
});
```

Technique 2 (auto-gen a salt and hash):

```
bcrypt.hash(myPlaintextPassword, saltRounds, function(err, hash) {
    // Store hash in your password DB.
});
```

Note that both techniques achieve the same end-result.

To check a password:

```
// Load hash from your password DB.
bcrypt.compare(myPlaintextPassword, hash, function(err, result) {
    // result == true
});
bcrypt.compare(someOtherPlaintextPassword, hash, function(err, result) {
    // result == false
});
```

동기식

sync

```
const bcrypt = require('bcrypt');
const saltRounds = 10;
const myPlaintextPassword = 's0/₩/₩P4$$w0rD';
const someOtherPlaintextPassword = 'not_bacon';
```

To hash a password:

Technique 1 (generate a salt and hash on separate function calls):

```
const salt = bcrypt.genSaltSync(saltRounds);
const hash = bcrypt.hashSync(myPlaintextPassword, salt);
// Store hash in your password DB.
```

Technique 2 (auto-gen a salt and hash):

```
const hash = bcrypt.hashSync(myPlaintextPassword, saltRounds);
// Store hash in your password DB.
```

As with async, both techniques achieve the same end-result.

To check a password:

```
// Load hash from your password DB.
bcrypt.compareSync(myPlaintextPassword, hash); // true
bcrypt.compareSync(someOtherPlaintextPassword, hash); // false
```

패스워드 해시

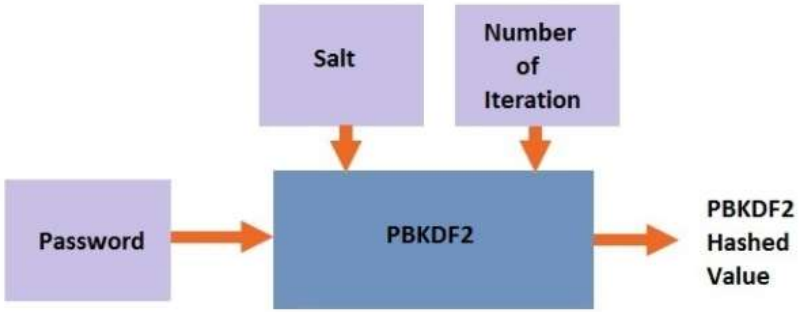
- bcrypt.js

```
const bcrypt = require("bcrypt");

const saltRounds = 10;
const myPassword = "hkjshkjdhfksdj";
const myPassword1 = "hkjshkjdhfksdj1";
let dbHash;

// 1. 사용자 등록
bcrypt.hash(myPassword, saltRounds, function (err, hash) {
  // Store hash in your password DB.
  console.log("PasswordHash: " + hash);
  dbHash = hash;

  // 2. 사용자 로그인
  bcrypt.compare(myPassword, dbHash, function (err, result) {
    // result == true
    console.log("Correct password: " + result);
  });
  bcrypt.compare(myPassword1, dbHash, function (err, result) {
    // result == false
    console.log("Incorrect password: " + result);
  });
});
```

3. PBKDF2 데모 페이지

패스워드기반 키생성 (PBKDF2)

사용자가 입력하는 패스워드를 직접 비밀키로 사용하는 것은 고정된 키를 사용하게 되어 사전공격 등의 방법이 가능하므로 보안성에 문제가 많다. 이를 해결하기 위하여 패스워드 기반 키생성함수(PBKDF2)를 이용하는데 (1)사용자 입력의 패스워드, (2)랜덤한 salt값, (3)반복횟수(iteration)값을 이용하여 난수처럼 보이는 암호키를 생성하여 사용한다. salt값과 반복횟수값은 공격자의 사전공격을 어렵게 하는 중요한 요소이다.



난수 Salt
난수 Salt 생성
랜덤 salt

반복횟수(Iteration)
100

패스워드
사용자입력패스워드

출력 키길이
16
바이트

생성된 키
PBKDF2 키생성

난수 Salt 생성
PBKDF2 키생성
초기화

4. 파이썬

- PBKDF2
  - https://pycryptodome.readthedocs.io/en/latest/src/protocol/kdf.html#pbkdf2

```
from Crypto.Protocol.KDF import PBKDF2
from Crypto.Hash import SHA512
from Crypto.Random import get_random_bytes

password = b'my super secret'
salt = get_random_bytes(16)
keys = PBKDF2(password, salt, 64, count=1000000, hmac_hash_module=SHA512)
key1 = keys[:32]
key2 = keys[32:]
```

파이썬 – pbkdf2


```
from Crypto.Protocol.KDF import PBKDF2
from Crypto.Hash import SHA512
from Crypto.Random import get_random_bytes

# password = b'my super secret sdfsdfsdf 한글'
message1 = "Hash Test 1..한글테스트..迫った妊婦たちてない 使度前必读 ."
password = message1.encode("utf-8")

salt = get_random_bytes(16)
keys = PBKDF2(password, salt, 64, count=1000000, hmac_hash_module=SHA512)
key1 = keys[:32]
key2 = keys[32:]

print('Password: {0}'.format(password))
print('Salt: {0}'.format(salt.hex()))
print('Keys: {0}'.format(keys.hex()))
print('Key1: {0}'.format(key1.hex()))
print('Key2: {0}'.format(key2.hex()))
```

파이썬 - bcrypt

- Bcrypt
  - https://pycryptodome.readthedocs.io/en/latest/src/protocol/kdf.html#bcrypt
  - 패스워드 저장

```
from base64 import b64encode
from Crypto.Hash import SHA256
from Crypto.Protocol.KDF import bcrypt

password = b"test"
b64pwd = b64encode(SHA256.new(password).digest())
bcrypt_hash = bcrypt(b64pwd, 12)
```

  - 패스워드 검증

```
from base64 import b64encode
from Crypto.Hash import SHA256
from Crypto.Protocol.KDF import bcrypt

password_to_test = b"test"
try:
    b64pwd = b64encode(SHA256.new(password).digest())
    bcrypt_check(b64pwd, bcrypt_hash)
except ValueError:
    print("Incorrect password")
```

파이썬 - bcrypt


```
from base64 import b64encode
from Crypto.Hash import SHA256
from Crypto.Protocol.KDF import bcrypt, bcrypt_check

# 사용자등록: Password Hash 생성
password = b"test"
b64pwd = b64encode(SHA256.new(password).digest())
bcrypt_hash = bcrypt(b64pwd, 12)
print('Password: {0}'.format(password))
print('Password Hash: {0}'.format(bcrypt_hash))

# 로그인: 패스워드 검증
password_to_test = b"test1"
try:
    b64pwd = b64encode(SHA256.new(password_to_test).digest())
    result = bcrypt_check(b64pwd, bcrypt_hash)
except ValueError:
    print("Incorrect password")
```