



Seoul
Software
ACademy

웹 개발자 부트캠프 과정

SeSAC x CODINGOn

With. 팀 뽀빠

암호화 이론

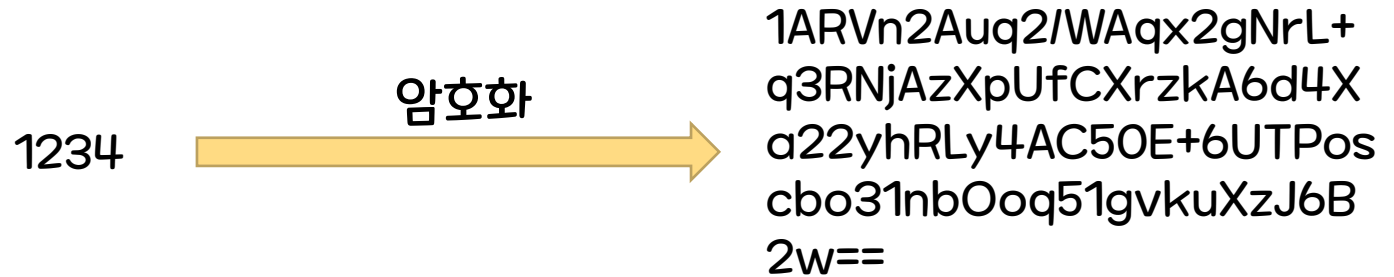
현재 DB

- DB에 pw 값이 그대로 저장
- DB가 해킹당한다면? 🤖

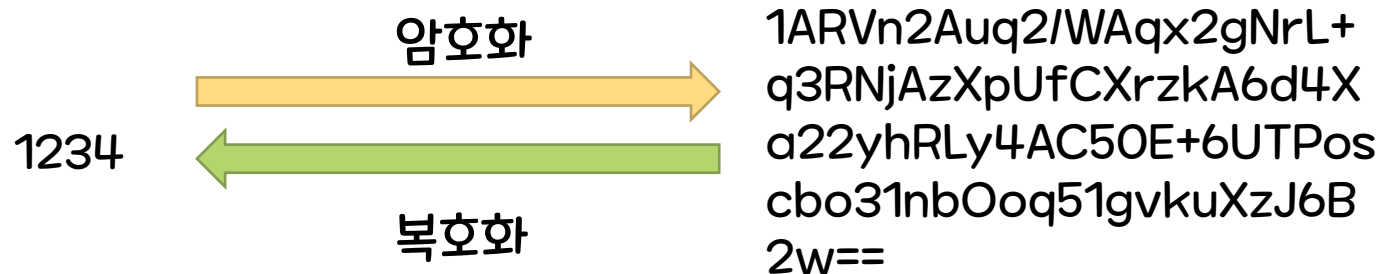
	id	pw	name
▶	1		1
	2		2
	dvadva	ks3ah	송하나
	hanjo	jk4	한조
	hong1234	8o4	홍길동
	jungkrat	4if	정크랫
	power70	qx8s	변사또
	sexysung	8awjko	성춘향
	widowmaker	3ewifh3	위도우
●	NULL	NULL	NULL

암호화 종류

단방향 암호화



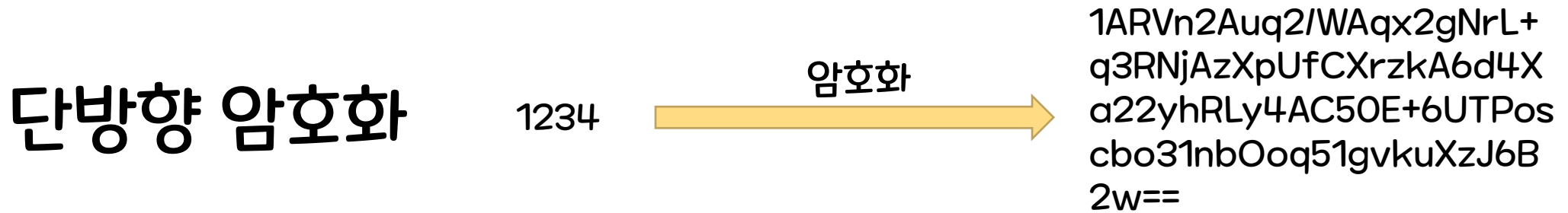
양방향 암호화



암호화 종류

암호화종류				
종류	암호화	복호화	암호화방식	알고리즘
단방향	가능	불가능	Hash	MD5, SHA-1, SHA-256
양방향	가능	가능	대칭키	AES, DES
			공개키	RSA, ECC

암호화 종류(단방향)

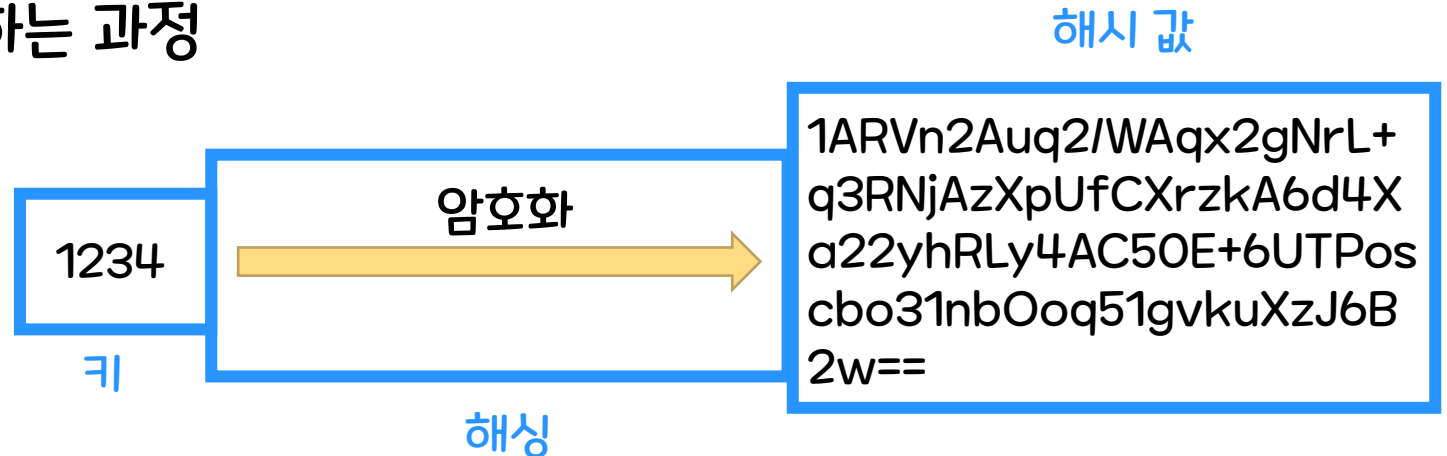


특징

- 데이터 무결성을 검증하는 데 주로 사용되며, **패스워드 저장** 등에서도 활용
- 단방향이므로 원본 데이터를 복원할 수 없음
- 동일한 데이터에 대해서는 항상 동일한 해시 값이 생성됨
 - 서로 다른 데이터에 대해서도 같은 해시 값이 나올 수 있음(충돌)
- 미세한 데이터 변화에도 해시 값은 완전히 다름
- 주로 해시 함수(MD5, SHA-1, SHA-256 등)를 사용하여 구현

해시(Hash)

- 해시(Hash) : 해시 함수에 의해 얻어지는 값
- 해시 함수 (Hash Function) = 해시 알고리즘
 - 임의의 크기의 데이터를 고정된 크기의 데이터로 변환하는 함수
 - 키(key) : 매핑 전 원래 데이터 값
 - 해시 값 (hash value) : 매핑 후 데이터 값
 - 해싱 (hashing) : 매핑하는 과정



해시함수(해시알고리즘)

SHA-256 (Secure Hash Algorithm 256-bit)

- 256비트 해시 값, 충돌 저항성 및 보안성 높음
- 많은 애플리케이션에서 데이터 무결성 검증이나 디지털 서명에 사용

SHA-512 (Secure Hash Algorithm 512-bit)

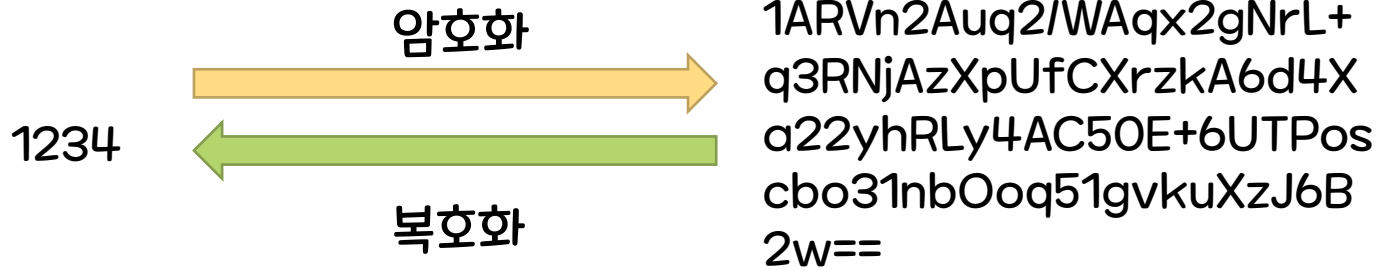
- 512비트 해시 값, 충돌 저항성 및 보안성 높음
- 더 큰 해시 값을 생성하기 때문에 좀 더 안전한 알고리즘

사용하지 않음

- MD5 (Message Digest Algorithm 5): 128비트 해시 값, 충돌 저항성 약함
- SHA-1 (Secure Hash Algorithm 1): 160비트 해시 값, 충돌 저항성 약함

암호화 종류(양방향)

양방향 암호화



특징

- 데이터의 기밀성을 유지하거나 **안전한 통신을 위해 사용**
- 공개키와 대칭키 암호화를 조합하여 데이터 보안을 유지하면서 처리 속도를 향상
- HTTPS와 같은 프로토콜로 클라이언트-서버 통신을 보호하여 안전한 웹 통신을 제공

대칭키 암호화 알고리즘

대칭키 사용: 암호화와 복호화에 동일한 키를 사용하므로 키 관리가 중요!!!

- **AES(Advanced Encryption Standard)**
 - 다양한 키(비밀 값)과 블록(암호화할 데이터)을 제공
 - 종류: AES-128, AES-192, AES-256
 - 빠른 처리 속도, 다양한 플랫폼에서 구현이 용이
 - ECB, CBC, CFB, OFB, CTR 등 다양한 운용 모드를 통해 데이터 블록을 처리
- DES (Data Encryption Standard)
 - 초기에 개발된 56비트 키와 64비트 블록 크기를 가지는 대칭키
 - Triple DES: 3DES는 DES를 여러 번 적용하여 암호화하므로 보안성은 높아지지만, 처리 속도가 느려짐

AES 블록 암호화 운용 모드

- ECB (Electronic Codebook):
 - 각 데이터 블록을 독립적으로 암호화
 - 같은 입력 블록에 대해 항상 같은 암호문이 생성되기 때문에 패턴이 누출될 수 있음
 - 보안성이 낮아 주로 간단한 암호화 작업에서 사용되며, 중복되는 패턴이 없는 데이터에 적용될 때 더 안전하게 사용
- CBC (Cipher Block Chaining):
 - 이전 블록의 암호문과 현재 블록의 평문을 XOR 연산하여 암호화
 - 초기 블록은 초기화 벡터(IV)로 암호화
 - 이전 블록의 암호문이 현재 블록에 영향을 주기 때문에 같은 평문에 대해 다른 암호문이 생성되어 패턴이 누출되지 않음

공개키 암호화 알고리즘

두 개의 키: 공개키와 개인키라는 두 개의 키 쌍을 사용

- **RSA(Rivest-Shamir-Adleman)**
 - 종류: RSA-1024, RSA-2048, RSA-3072, RSA-4096
- **ECC(Elliptic Curve Cryptography): 타원 곡선을 기반**
 - 종류: ECC-192, ECC-256, ECC-384

특징

- 공개키는 다른 사람과 공유되며, 개인키는 오직 소유자만 알고 있어야 함
- ECC는 작은 키 길이로도 강력한 보안성을 제공하며, RSA는 대칭키 암호화보다 복잡하지만 효율적으로 사용

암호화 실습

Crypto

- 암호화 알고리즘이 모여 있는 패키지



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

Crypto 암호화

- `createHash(algorithm)`
 - 지정한 해시 알고리즘을 사용하여 해시 객체를 생성
- `pbkdf2Sync(password, salt, iterations, keylen, digest)`
 - 비밀번호 기반 키 도출 함수를 동기적으로 실행
- `createCipheriv(algorithm, key, iv) / createDecipheriv(algorithm, key, iv)`
 - 대칭키 암호화와 복호화를 위한 객체를 생성

Crypto 암호화

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

const createHashedPassword = (password) => {
  return crypto.createHash("sha512").update(password).digest("base64");
};
```



해시를 만들기 위해 사용하는 함수로 parameter로는 사용할 알고리즘 이름이 들어간다.

Crypto 암호화

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

const createHashedPassword = (password) => {
  return crypto.createHash("sha512").update(password).digest("base64");
};
```

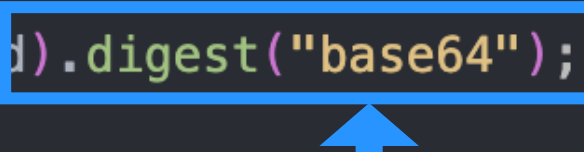


password 문자열을 전달합니다. 이 데이터가 해시 함수에 입력
으로 사용

Crypto 암호화

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

const createHashedPassword = (password) => {
  return crypto.createHash("sha512").update(password).digest("base64");
};
```

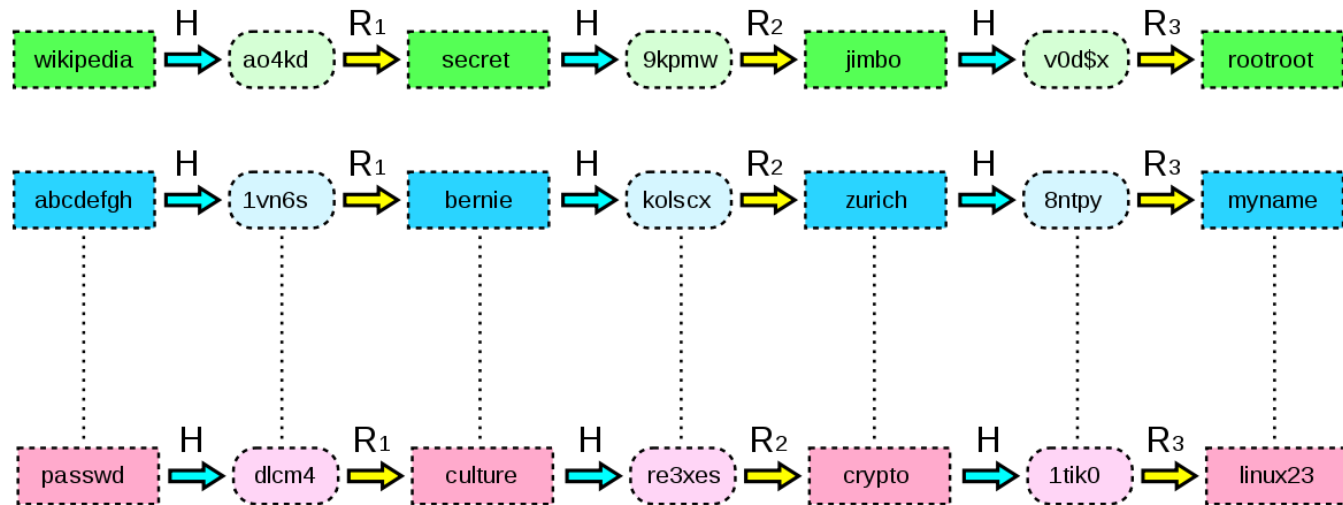


해시를 만들 때 인코딩 방식을 설정한다.

- base64 : 64개의 가능한 문자로 데이터를 인코딩하는 방식
64개의 문자 중 62개는 영문 대소문자와 10진수 숫자로 구성,
나머지 두 개의 문자는 인코딩과 디코딩 과정에서 추가적인 문자로 사용
- hex: 16진수 인코딩
16진수? 각 숫자는 0-9까지는 숫자로, 10-15까지는 a부터 f까지의 문자로 구성

해시함수 한계! 레인보우 테이블

레인보우 테이블(Rainbow Table)?
해시함수를 사용해 만들어낼 수 있는
값들을 대량으로 저장해놓은 표



해시 함수를 사용하여 암호화된 비밀번호를 빠르게 역추적하여 원본 비밀번호를 찾는 공격 기법

암호화 보완법

1) salt

- 입력한 값에 salt라는 특정 값을 붙여 변형시키는 것

2) 해시 함수 반복

- 해시 함수를 여러 번 돌려 본래의 값을 예측하기 어렵게 만드는 것

Crypto 암호화

- 비밀번호 기반 키 도출 함수
(PBKDF: Password-Based Key Derivation Function) : 버퍼반환
- 주로 사용자 비밀번호를 저장할 때 사용

```
const salt = crypto.randomBytes(16).toString('base64') // 솔트 생성
const iterations = 100000; // 반복 횟수
const keylen = 64; // 생성할 키의 길이
const digest = 'sha512'; // 해시 알고리즘

const createPbkdf = (password) => {
  return crypto.pbkdf2Sync(password, salt, iterations, keylen, digest).toString('base64');
};
```

Crypto 검증

```
const verifyPassword = (password, salt, dbPassword) => {  
  const compare = crypto.pbkdf2Sync(password, salt, iterations, keylen, digest).toString('base64');  
  if (compare === dbPassword) return true;  
  return false;  
}
```

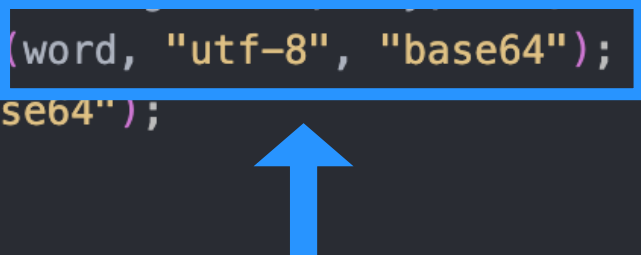
Crypto 는 단방향 알고리즘이기 때문에 복호화가 불가능

즉, 입력한 값과 동일한 알고리즘을 이용해 다시 암호화를 해 비교한다.

Crypto 암호화

```
const algorithm = "aes-256-cbc"; //알고리즘
const key = crypto.randomBytes(32); // 256비트 키
const iv = crypto.randomBytes(16); // 초기화 벡터


const cipherEncrypt = (word) => {
  const cipher = crypto.createCipheriv(algorithm, key, iv); //암호화 객체 생성
  let encryptedData = cipher.update(word, "utf-8", "base64"); //암호화할 데이터 처리
  encryptedData += cipher.final("base64");
  return encryptedData;
};
```



암호화 원본 데이터, 입력 인코딩, 출력 인코딩

Crypto 복호화

```
const decipher = (encryptedData) => {  
  const decipher = crypto.createDecipheriv(algorithm, key, iv);  
  let decryptedData = decipher.update(encryptedData, "base64", "utf-8");  
  decryptedData += decipher.final("utf-8");  
  console.log("Decrypted:", decryptedData);  
  return decryptedData;  
};
```



암호화 데이터, 입력 인코딩, 출력 인코딩

Bcrypt

- 비밀번호를 암호화하는 알고리즘 중 하나
- Blowfish 암호를 기반으로 설계된 암호화 함수
- 현재까지도 사용 중인 가장 강력한 매커니즘임과 동시에 해싱이 느리고 비용이 많이 든다.
- 강력한 보안이 필요할 때 적합

Bcrypt 암호화

```
npm install bcrypt
```

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



Bcrypt 암호화

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

const salt = 10; // 암호화에 사용할 salt의 수준을 설정합니다. 정수사용

const bcryptPassword = (password) => {
  return bcrypt.hashSync(password, salt);
};

const comparePassword = (password, dbPassword) => {
  return bcrypt.compareSync(password, dbPassword);
};
```