데이터 암호화 (Data Encryption)

목 차

암호화(Encryption)

- 허가된 이 외에는 읽을 수 없도록 정보를 부호화 하는 것
- 부호화하는 방식을 암호화 알고리즘이라 함

암호 알고리즘의 구분

- 키의 갯수 :
- 1 key: 비밀키(대칭키)
- 2 key: 공개키(비대칭키)
- 데이터의 처리 단위: 스트림(Stream), 블록(Block)
- 원본 복호화 여부: 단방향, 양방향

해시 함수(Hash Function)

- 임의의 길이의 입력을 고정된 길이의 출력으로 바꾸는 함수
- 동일한 입력에 대해서는 동일한 출력이 보장
- Hash Table 기반의 빠른 검색이 필요한 경우 등에 많이 사용
- 해시 함수는 빠른 연산속도를 특징으로 하며 출력값은 digest 라고 부름.
- h(M) = H 로 표현
- h() 는 해시 함수, M 는 입력 값, H = 해시 값을 의미
- 다른 입력이지만 동일한 출력이 발생할 수 있으며 이를 충돌 (Collision) 이라고 함
- 충돌이 적어야 좋은 알고리즘

Cryptographic Hash Function

-암호용 해시 함수는 매핑된 해싱 값만을 알아가지고는 원래 입력 값을 알아내기 힘들다는 사실에 의해 사용될 수 있다. 또한 전송된 데이터의 무결성을 확인해주는 데 사용되기도 하는데, 메시지가 누구에게서 온 것인지 입증해주는 HMAC를 구성하는 블록으로 사용된다. 해시 함수는 결정론적으로 작동해야 하며, 따라서 두 해시 값이 다르다면 그 해시값에 대한 원래 데이터도 달라야 한다. (역은 성립하지 않는다) 해시 함수의 질은 입력 영역에서의 해시 충돌 확률로 결정되는데, 해시 충돌의 확률이 높을수록 서로 다른 데이터를 구별하기 어려워지고 검색하는 비용이 증가하게 된다.

- 일반 해시 함수에 아래 3가지 특성을 만족하는 함수

역상 저항성(preimage resistance)

-주어진 해시 값에 대해, 그 해시 값을 생성하는 입력값을 찾는 것이 계산상 어렵다.
제 1 역상 공격에 대해 안전해야 한다. 이 성질은 일방향함수와 연관되어 있다.
- H(M) = H 에서 H 값인 'aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d'
를 알아도 입력인 M 값 'hello' 를 계산하기는 어려움

제 2 역상 저항성(second preimage resistance)

입력 값에 대해, 그 입력의 해시 값을 바꾸지 않으면서 입력을 변경하는 것이 계산상
불가능 2역상 공격에 대해 안전해야 한다.
- h(M) = H 에서 H 값인 aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d'
- M 인 'hello'를 알 경우 동일한 출력을 내는 M1 을 찾
는 문제

충돌 저항성(collision resistance)

해시 충돌에 대해 안전해야 한다. 같은 해시 값을 생성하는 두 개의 입력값을 찾는 것이 계산상 어려워야 한다.
동일한 출력 값을 내는 임의의 M1, M2 을 찾는 문제
충돌 저항성에 따라 Hash 의 안정성은 길이의 반정도로 판단
(SHA1 = 80bit, SHA256 = 128bit)

-해시 충돌

해시 충돌이란 해시 함수가 서로 다른 두 개의 입력값에 대해 동일한 출력값을 내는 상황을 의미한다.
해시 함수가 무한한 가짓수의 입력값을 받아 유한한 가짓수의 출력값을 생성하는 경우,
비둘기집 원리에 의해 해시 충돌은 항상 존재한다.
해시 충돌은 해시 함수를 이용한 자료구조나 알고리즘의 효율성을 떨어뜨리며,
따라서 해시 함수는 해시 충돌이 자주 발생하지 않도록 구성되어야 한다.
암호학적 해시 함수의 경우 해시 함수의 안전성을 깨뜨리는 충돌 공격이 가능할 수 있기 때문에
의도적인 해시 충돌을 만드는 것이 어렵도록 만들어야 한다.
해시 충돌을 일으키는 임의의 두 값을 찾는 과정을 '충돌 공격(collision attack)'이라고 한다.
또한 주어진 값에 대해 그 값과 해시 충돌을 일으키는 값을 찾는 것은 역상 공격이라고 한다.
암호 공격라는 측면에서 보면 역상 공격은 충돌 공격 보다 더 심각한 공격이 될 것이다.

해시 알고리즘 종류

알고리즘	출력 비트 수	내부 상태 크기[c 1]	블록 크기	Length size	Word size	라운드 수	수)[c 2]		
							충돌	2차 역상	역상
GOST	256	256	256	256	32	256	2^{105}	2^{192}	2^{192}
HAVAL	256/224/192/160/128	256	1,024	64	32	160/128/96	가능		
MD2	128	384	128	-	32	864	$2^{63.3}$		2^{73}
MD4	128	128	512	64	32	48	3	2^{64}	$2^{78.4}$
MD5	128	128	512	64	32	64	$2^{20.96}$		$2^{123.4}$
PANAMA	256	8,736	256	-	32	-	가능		
RadioGatún	Up to 608/1,216 (19 words)	58 words	3 words	-	1–64	-	2^{352} 또는 2^{704}		
RIPEMD	128	128	512	64	32	48	2^{18}		
RIPEMD-128/256	128/256	128/256	512	64	32	64			
RIPEMD-160	160	160	512	64	32	80	2^{51}:48		
RIPEMD-320	320	320	512	64	32	80			
SHA-0	160	160	512	64	32	80	$2^{33.6}$		
SHA-1	160	160	512	64	32	80	2^{51}		
SHA-256/224	256/224	256	512	64	32	64	$2^{28.5}$:24		$2^{248.4}$:42
SHA-512/384	512/384	512	1,024	128	64	80	$2^{32.5}$:24		$2^{494.6}$:42
Tiger(2)-192/160/128	192/160/128	192	512	64	64	24	2^{62}:19		$2^{184.3}$
WHIRLPOOL	512	512	512	256	8	10	2^{120}:4.5		

- MD5(128bit)는 매우 취약하고 SHA1(160bit) 은 사용하지 않는 것을 권고
- SHA1 이 당장 문제되는 것은 아니고 전자 서명등 장기 보존하는 데이터 사용에만 권고하지 않음
- Session에만 사용하거나 파일 checksum, git commit hash 등에 사용은 아무 문제없음

대칭키 알고리즘(Symmetric Algorithm)

- 하나의 키를 사용하여 데이터를 암/복호화
- 암/복호화 Key는 대칭키 또는 비밀키(Secret Key) 라고 부름
- 암/복호화 속도가 빠르고 구현이 용이
- 블록(Block) 암호 알고리즘과 스트림(Stream) 암호 알고리즘으로 나눠짐

스트림 암호 알고리즘

- 대칭키를 만든 후에 Bit 단위로 XOR 연산으로 암호화
- RC4, AS/2 등의 알고리즘이 있음
- 속도는 빠르지만 실무에서는 거의 사용하지 않으므로 몰라도 됨.!

블록 암호 알고리즘

- 대칭키 알고리즘 중 암/복호화시 데이터를 블록 단위로 처리하는 알고리즘
- 이제는 쓰면 안 되는 DES, 현재 미국 표준인 AES, 국내 표준 SEED등이 있음
- 블록 크기와 Key 길이는 알고리즘마다 다름
- SEED 는 128 bit(16 byte), AES 는 128, 192, 256 bit Key 길이 지원
- 일본이 만든 Camellia 는 SSL/TLS에서 많이 사용됨
- AES 는 전 세계 공모를 통해 선정 - 벨기에의 암호학자 2명이 제출한 Rijndael 이 채택

보편적인 알고리즘 사용종류

단방향 : 재사용이 불가능 (비밀번호)
양방향 : (재사용 가능 (주소, 이메일)
단방향 암호는 bcrypt가 제일 보편화 되어있다.
양방향 암호는 AES 와 RSA등이 보편화 되어있다.

Bcrypt Algorithm

bcrypt is a password hashing function designed by Niels Provos and David Mazières, based on the Blowfish cipher, and presented at USENIX in 1999.[1] Besides incorporating a salt to protect against rainbow table attacks, bcrypt is an adaptive function: over time, the iteration count can be increased to make it slower, so it remains resistant to brute-force search attacks even with increasing computation power.

Rainbow Attack

- 역상 저항성을 우회해서 입력 값을 찾기 위한 Brute-force attack 의 일종
- 다양한 문자열을 조합하여 해시값을 계산한 rainbow table 작성
- 획득한 해시값을 table 과 비교하여 원본 메시지 추출
- 입력 값에 임의의 값(Salt) 를 추가하여 hash 를 계산하면 방지

암호화 관련 라이브러리 추가

```
<dependencies>
    <!-- 세큐리티 -->
    <!-- https://mvnrepository.com/artifact/org.springframework.security/spring-security-core -->
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-core</artifactId>
        <version>5.0.8.RELEASE</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.springframework.security/spring-security-web -->
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-web</artifactId>
        <version>5.0.8.RELEASE</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.springframework.security/spring-security-config -->
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-config</artifactId>
        <version>5.0.8.RELEASE</version>
    </dependency>

    <!-- Spring -->
    <dependency>
        <groupId>org.springframework</groupId>
```

컨트롤러에 **Dependency Injection**

MemberController.java

```
37
38      // 회원 가입 post
39      @RequestMapping(value = "/register", method = RequestMethod.POST)
40      public String postRegister(MemberVO vo) throws Exception {
41          logger.info("post resister");
42
43          String inputPass = vo.getUserPass();
44          String pass = passEncoder.encode(inputPass);
45          vo.setUserPass(pass);
46
47          service.register(vo);
48
49          return "redirect:/";
50      }
51
```

@Autowired BCryptPasswordEncoder passEncoder; 추가 작성 ----

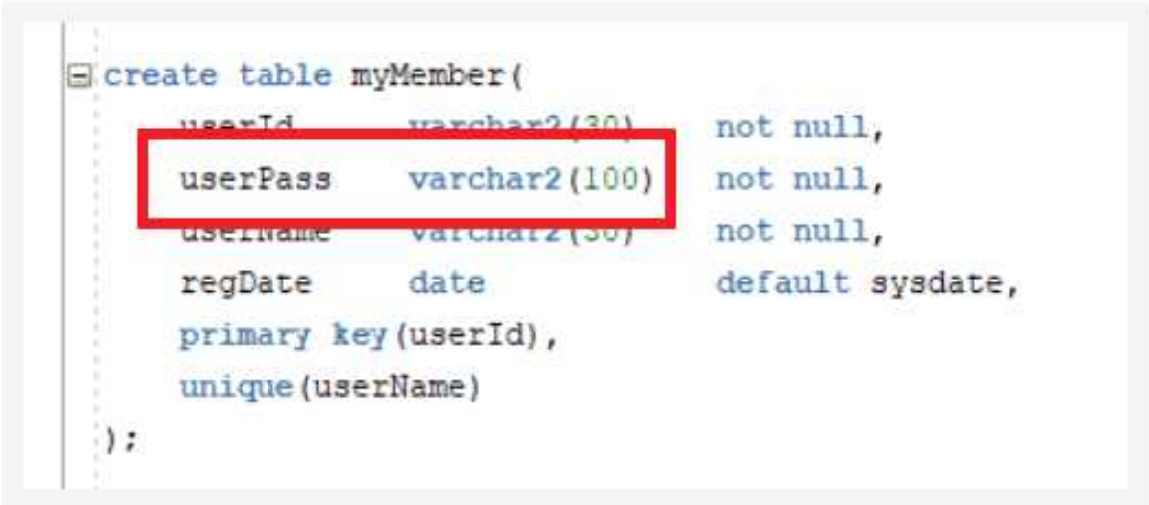
```
select * from myMember;
```

질의 결과 ×

SQL | 인출된 모든 행: 2(0초)

	USERID	USERPASS	USERNAME	REGDATE
1	user01	1111	user01	18/10/01
2	test01	$2a$10$jjP9H8U8eoah4zl4//erpO2eNqrd0jeuIRA0VZqrEeVKjvss3FBLW	test01	18/10/12

1111이 아닌 암호화로 저장 되어있음



```
create table myMember(
    userId      varchar2(30)    not null,
    userPass    varchar2(100)   not null,
    userName    varchar2(30)    not null,
    regDate     date            default sysdate,
    primary key(userId),
    unique(userName)
);
```

로그인 실패 ->저장된 암호는 **Encryption** 되어 있기 때문에 **DB**에는 **21312 4232352562kkfj23**식으로 저장되어있음.

때문에 복호화를 시켜서 **service.login(vo)** 로 인해 입력한 아이디와 매칭되는 값을 데이터에서 찾아 리턴고 **passEncoder.matches()** 메서드는 입력한 패스워드와 저장된 패스워드를 비교, 같다면 **true** 틀리면 **false**를 리턴

적절한 암호 알고리즘 선택법

- 현대의 암호 알고리즘은 암호화 방법이 모두 투명하게 공개
- 개발한 암호학자들은 해독시 현상금 지불
- 혹독한 검증을 거쳐 살아 남은 게 현대의 암호 알고리즘
- 스스로 만든 알고리즘은 전혀 검증이 되지 않았으므로
많이 쓰는 알고리즘 선택
- 대칭키는 AES, 공개키는 RSA 등 많이 사용되고 구현물이 많
은 알고리즘 사용
- 대칭키 운영 모드로 ECB 는 절대 피할 것
- Hash 함수와 암호화 key 길이는 데이터의 보존 연한을
고려하여 선택

사례별 암호화 적용방법

- 암호화는 시스템과 데이터가 털리는 최악의 상황에 대비한 최후의 방어책
- 보안 취약점 패치/업데이트, 방화벽, SELinux, 침입탐지,
Secure Coding 등의 보안 대책을 먼저 적용해야 ,단방향 암호화(사용자 암호)
- Hash 함수보다는 Bcrypt 등의 password 해싱 전용 함수 사용
- 꼭 Hash 를 직접 사용해야 한다면 그래도 Bcyrpt
- 그래도 고집을 꺽지 못하겠다면 2가지 반드시 반영
- Rainbow attack 을 방지하기 위한 Salt 첨가 및 관리
- Brute-force attack 을 방지하기 위한 적절한 Key stretching
양방향 암호화(사용자만 해독 가능)
- Lastpass 같이 사용자의 데이터를 보관하는 서비스
- 고객의 신뢰 확보를 위해서는 서비스 제공자는 고객 데이터
를 풀 수 있으면 안 됨
- 사용자만 풀 수 있도록 key 와 IV 는 PBKDF2 로 생성
- 알고리즘은 AES256-CBC 사용
- Brute-Force Attack 을 방지하기 위해 Key stretching 용
Iteration count 는 최소 몇 천번 이상 수행
양방향 암호화(서비스 제공자만 해독 가능)
- 고객의 신용 카드 정보, 계좌 정보등을 보관할 경우 서비스
제공자가 필요시 해독해서 사용해야 함
- 중요 데이터는 강력한 알고리즘 (AES256등) 을 사용하여 암호화
- 암호화에 사용한 Key 를 파일로 관리하면 위험

- HSM 을 도입하여 HSM 내부에 key 관리 및 암호화 연산 수행
- Cloud 환경일 경우 HSM Managed Service 사용

Transport layer 암호화

- Network 을 통해 상대방과 데이터를 주고 받을 경우
- 별도의 전용 프로토콜이 있는 곳(금융권등)이 아니면 무조건 HTTPS 를 사용
- 사용도 쉽고 보안도 견고(Web, WAS 기본 탑재, 방대한 구현 library..)
- 최신 버전의 TLS 와 알고리즘, HSTS(HTTP Strict Transport Security) 를 적용
- 자세한 내용은 https://lesstif.gitbooks.io/web-service-hardening/content/ssl-tlshttps.html#hsts http-strict-transport-security 참고

Application Layer 암호화

- HTTPS 로도 전송 계층의 암호화와 무결성이 보장
- 하지만 특정 업무 도메인의 경우 App layer 의 보안을 요구하는 경우가 있음
- 이런 용도를 위해 암호 메시지 규격(CMS; Cryptographic Message Syntax) 중 Enveloped Data 를 사용