11차 강의
자바웹프로그래밍

강사 : 최도현

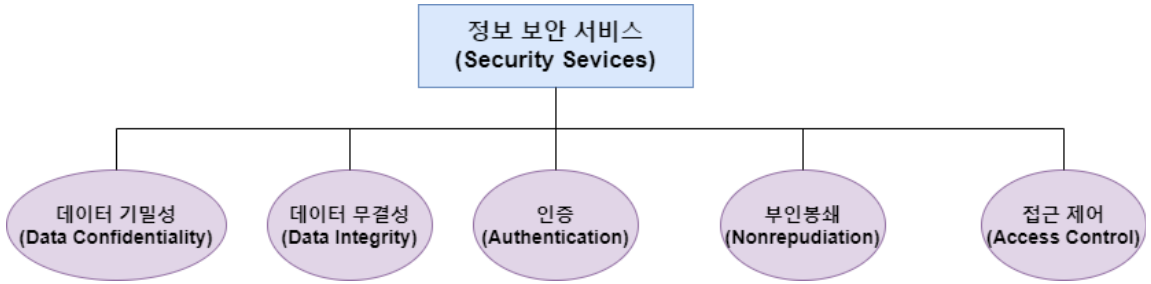
오늘의 할일 - 1

- 주 별 기술 트렌드 분석

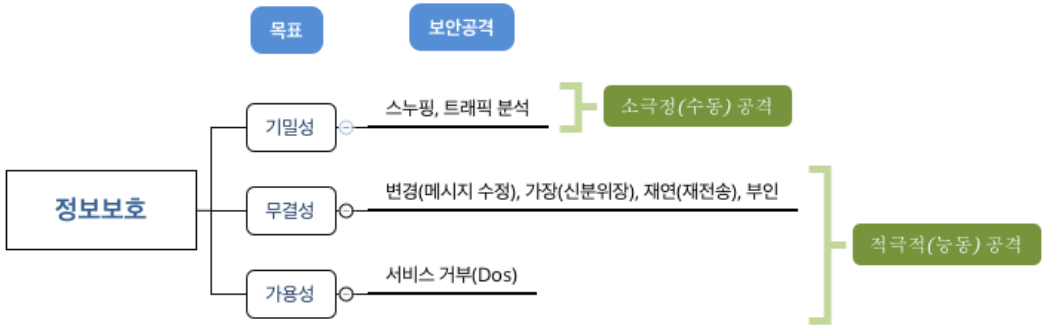
웹 보안 – 암호화 & 보안 토큰

암호화

- 참고 : OSI 7 계층 표준
  - 세부 동작 방식을 계층 분리
    - 각 계층이 통신을 위한 진입로
  - 암호화의 필요성 : 계층 보호
    - HTML 폼 전송 데이터 : 분석 가능
    - 모든 계층이 안전 X
  - 각 계층마다 적절한 보안 기술 적용 필요
- 보안의 필수 요소
  - 기밀성은 제3자가 데이터를 분석 x
  - 오늘 실습에서 **암호화**



정보보호 3대 요소



보안 토큰

- JWT (JSON Web Token) 보안이란?
  - JSON 형식의 보안 토큰
    - 사용자 인증, 확장성과 빠른 인증 제공
  - OAuth2.0, 로그인 세션 등에서 사용됨

- 내부 구조(3개 구성)
  - Header (헤더)
    - 토큰 타입(JWT), 서명 알고리즘 포함
  - Payload (페이로드)
    - 사용자 정보와 클레임(claim) 포함
  - Signature (서명)
    - 헤더 + 페이로드를 비밀 키로 서명
    - 위변조 여부 확인에 사용

전용 토큰을 생성하고 검증

Encoded	Decoded	Parts

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
.eyJpc3MiOiIiLCJpYXQiOjE2NzYyMTc5NT
AsImV4cCI6MTcwNzc1Mzk1MCwiYXVkIjoi
YWthbWFpLWJsb2ciLCJzdWIiOiIiLCJjb21
wYW55IjoiQWthbWFpIiwidXNlciI6IkFrYW1
haS1yZWFkZXIiLCJhZG1pbiI6Im5vIn0.kM
Pz3Z7BSlBTJKijD8[illegible]
oQwJO6I
	{
	"typ": "JWT",
	"alg": "HS256"
	}
		Header

	{
	"iss": "",
	"iat": 1676217950
	"sub": "",
	"company": "Akamai",
	"user": "Akamai-reader",
	"admin": "no"
	}
		Payload

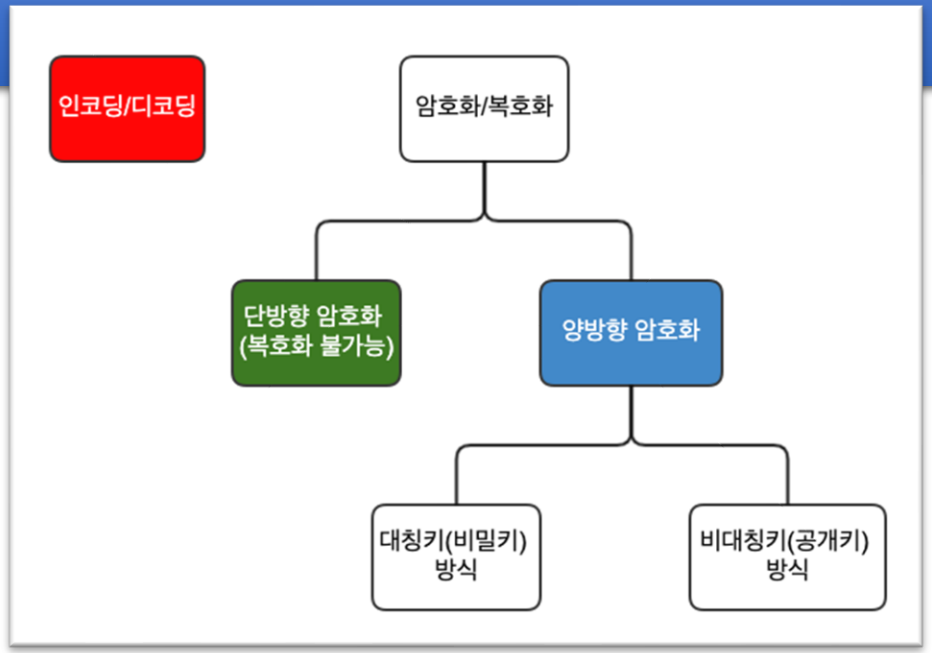
	HMACSHA256(base64Encode(header) + "." + base64Encode(payload), secret_key)	Signature

실습 : 추가 보안 토큰 구현

• 데이터 저장 - 암호화

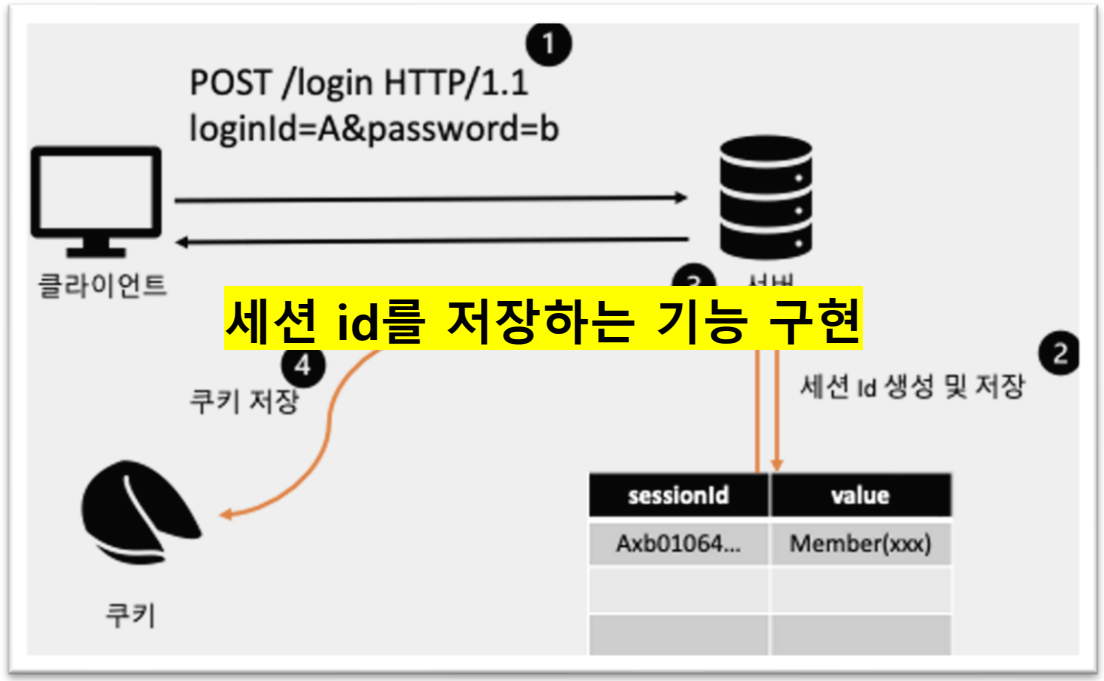
데이터 저장 – 암호화(대칭키:비밀키)

안전한 통신 – 보안 토큰



지난주 내용 살펴보기

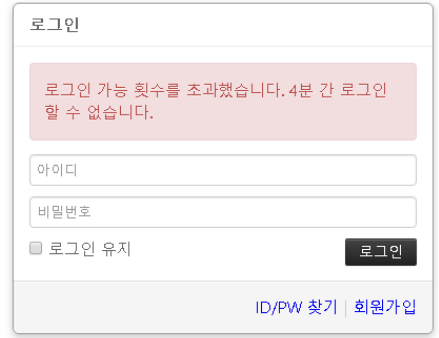
- 데이터 저장 – 세션
  - 세션을 통한 로그인 저장 및 상태 확인



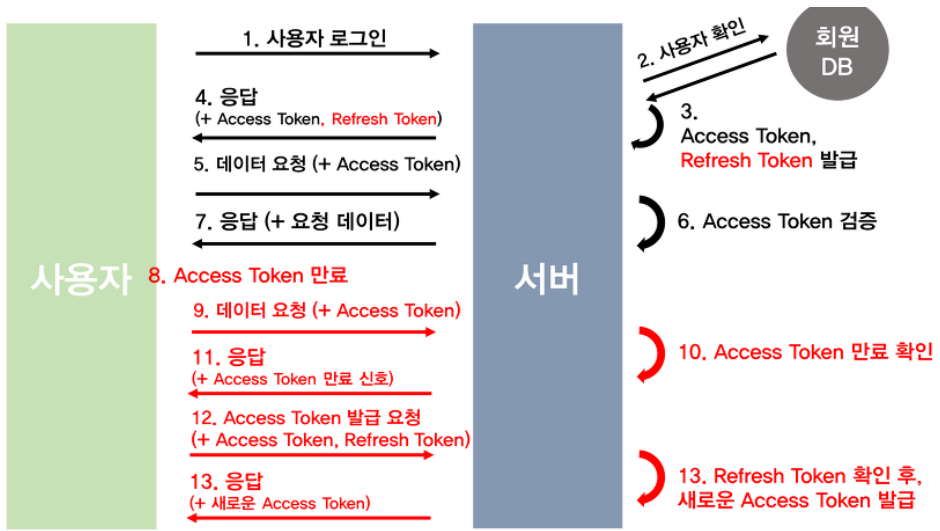
항목	태그 이름/설명
getElementById를 대체할 수 있는 이 함수는?	
세션 스토리지 객체의 이름은?	
세션 스토리지에 저장된 모든 정보를 삭제하는 함수는?	
기존 쿠키와 세션의 저장 유지하는 방법에 차이점은?	
세션 스토리지의 자료구조의 구성은?	
세션 스토리지는 쿠키에 비해 안전한가?	

데이터 저장 – 세션(암호화/복호화)

- 로그인 기능 강화하자. 안전한 데이터 저장?
  - 주요 보안 기능들 예)
    - 세션 토큰 기능 추가
      - JSP, PHP 등 서버사이드 언어로 세션 구현(PASS)
    - 세션 로그인 시간 설정
      - 로그인 시간/횟수 제한, 로그인 시간(요청) 지연
  - 안전한 데이터 교환 기능
    - **암호화/복호화 처리** 구현해보자.
    - 추가 : 보안 토큰을 구현해보자.

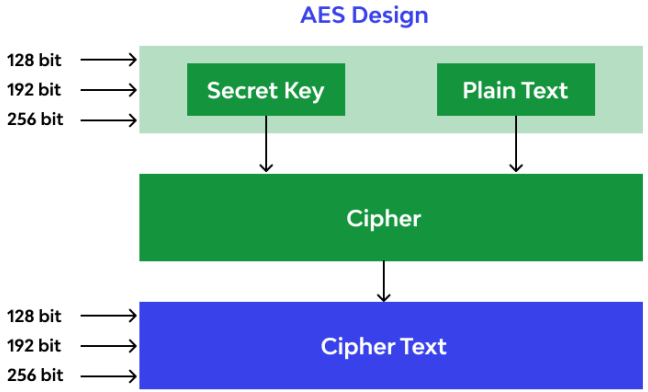


암호화/복호화 및 보안 토큰 구현하기!

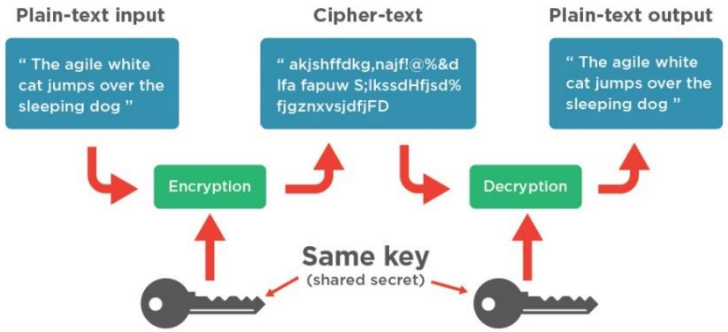


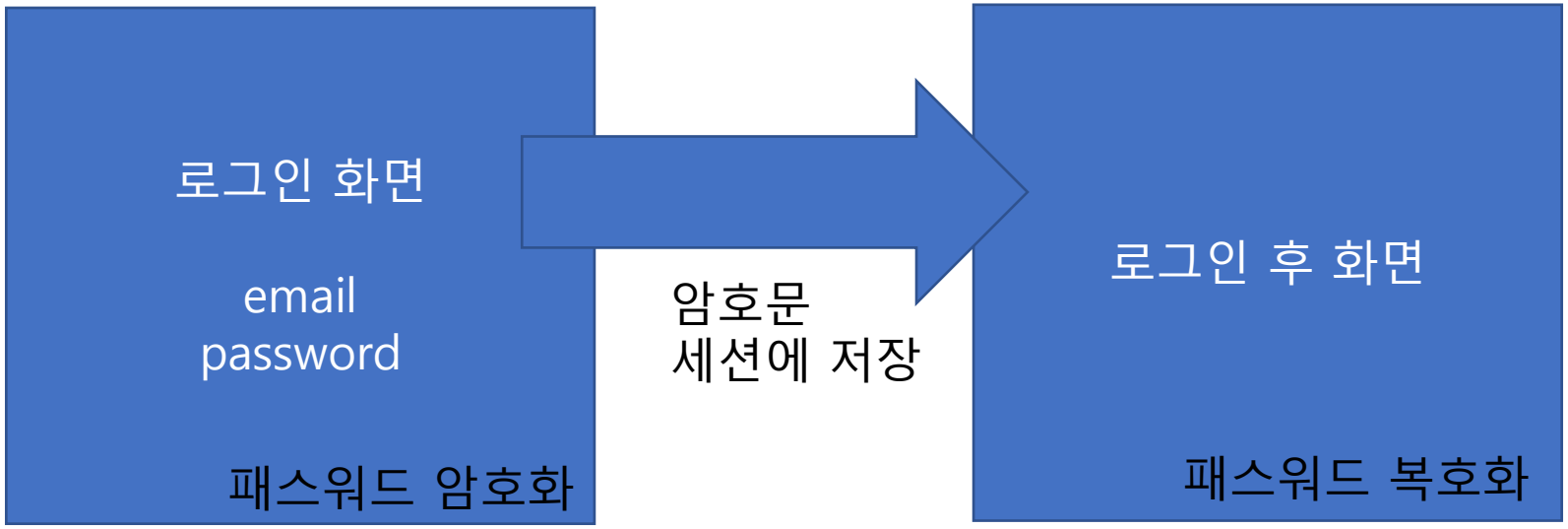
데이터 저장 – 세션(암호화/복호화)

- 기존 자바스크립트 연동
  - index_login.html를 수정한다.
    - login.html과 같이 헤더에 login.js을 연동한다.
  - 암호화/복호화의 동작과정 – AES 256 적용



대칭키 : 양방향 같은 키를 사용





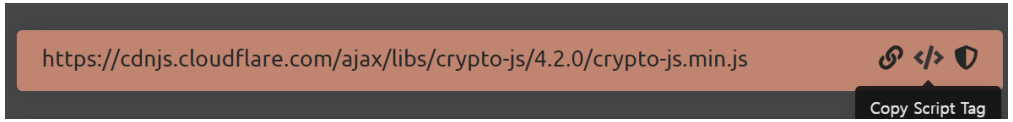
login.html **index_login.html**

데이터 저장 – 세션(암호화/복호화)

- crypto-js 라이브러리 활용
  - login.html, index_login.html를 수정한다.
    - 참고 : https://cdnjs.com/libraries/crypto-js
    - 웹 사이트에 가서 직접 복사하자.

  - 참고 : AES 암호 알고리즘
    - Advanced Encryption Standard
      - 소스 오픈 : 내부 동작 알아도 분석 X
    - 미국, 국제표준, 대칭키, 블록 암호

  - 두 파일 모두 head에 연동한다
    - 앞서 연동한 login.js 다음에 삽입한다. (defer)



공개키(더 안전) > 대칭키

구분	공공기관	민간부문 (법인·단체·개인)
대칭키 암호 알고리즘	SEED, LEA, HIGHT, ARIA	SEED HIGHT ARIA-128/192/256 AES-128/192/256 Camelia-128/192/256 등
공개키 암호 알고리즘 (메시지 암·복호화)	RSAES-OAEP	RSA RSAES-OAEP 등
일방향 암호 알고리즘	SHA-224/256/384/512	SHA-224/256/384/512 Whirlpool 등

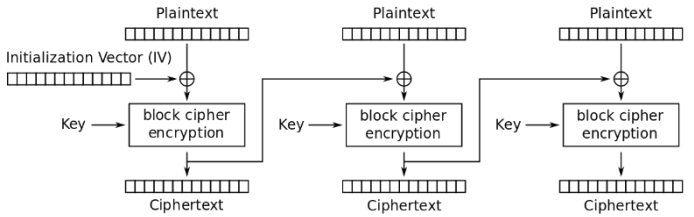
데이터 저장 – 세션(암호화/복호화)

- Js 폴더에 crypto.js를 추가한다.
  - Head에 연동한다, 암/복호화 함수를 추가한다.
    - 참고 : 양방향 동일한 키, 기본 CBC 모드로 동작한다.

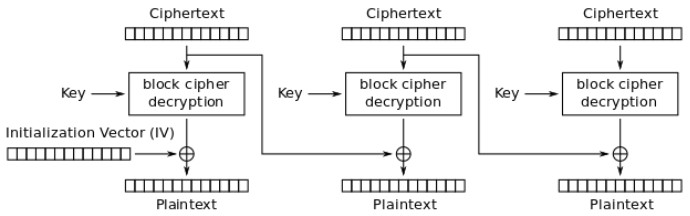
용도	대칭 키 블록 운영모드	금융권 이용분야	구체적 예시
기밀성	ECB, CBC, CFB, OFB, CTR, XTS-AES	거래내용 암호화 등	SEED128-CBC
인증	CMAC	신용카드 결제 등	AES, SEED128-CMAC
기밀성/인증	CCM, GCM, KW, KWP	교통카드 지불 등	AES, SEED128-CCM

기본 : IV 랜덤 정수가 입력됨

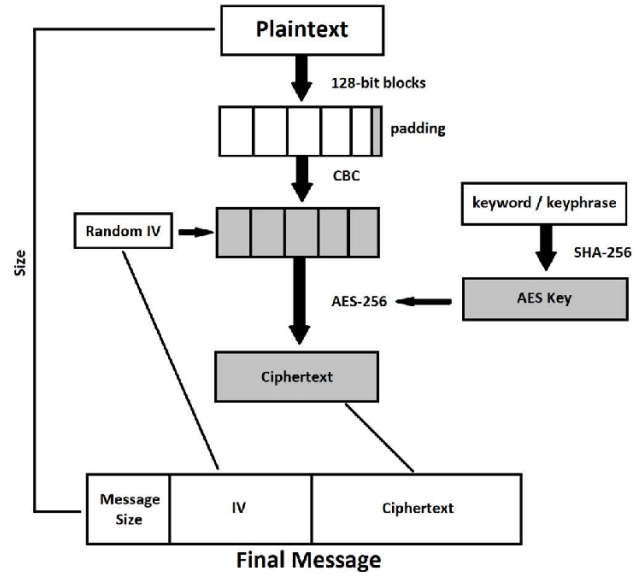
```
function encodeByAES256(key, data){ //
    const cipher = CryptoJS.AES.encrypt(data, CryptoJS.enc.Utf8.parse(key), {
        iv: CryptoJS.enc.Utf8.parse(""), // IV 초기화 벡터
        padding: CryptoJS.pad.Pkcs7, // 패딩
        mode: CryptoJS.mode.CBC // 운영 모드
    });
    return cipher.toString();
}

function decodeByAES256(key, data){
    const cipher = CryptoJS.AES.decrypt(data, CryptoJS.enc.Utf8.parse(key), {
        iv: CryptoJS.enc.Utf8.parse(""),
        padding: CryptoJS.pad.Pkcs7,
        mode: CryptoJS.mode.CBC
    });
    return cipher.toString(CryptoJS.enc.Utf8);
}
```



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption



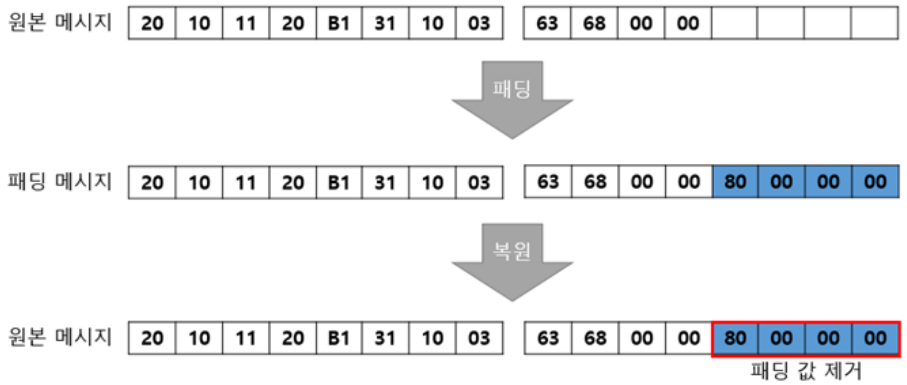
데이터 저장 – 세션(암호화/복호화)

- crypto.js에 패스워드 보안 처리를 추가한다.
  - 암호 알고리즘 : 임시 키를 사용, 패딩 및 인코딩 처리
    - 참고 : 블록 암호이다. (블록 크기의 배수로 분리)
    - 참고 : 키는 사전에 교환 필요

Key Size	Possible combinations
1-bit	2
2-bit	4
4-bit	16
8-bit	256
16-bit	65536
32-bit	4.2×10^{9}
56-bit (DES)	7.2×10^{16}
64-bit	1.8×10^{19}
128-bit (AES)	3.4×10^{38}
192-bit (AES)	6.2×10^{57}
256-bit (AES)	1.1×10^{77}

```
function encrypt_text(password){
    const k = "key"; // 클라이언트 키
    const rk = k.padEnd(32, " "); // AES256은 key 길이가 32
    const b = password;
    const eb = this.encodeByAES256(rk, b); // 실제 암호화
    return eb;
    console.log(eb);
}

function decrypt_text(){
    const k = "key"; // 서버의 키
    const rk = k.padEnd(32, " "); // AES256은 key 길이가 32
    const eb = session_get();
    const b = this.decodeByAES256(rk, eb); // 실제 복호화
    console.log(b);
}
```

키는 크면 클수록 안전하다.

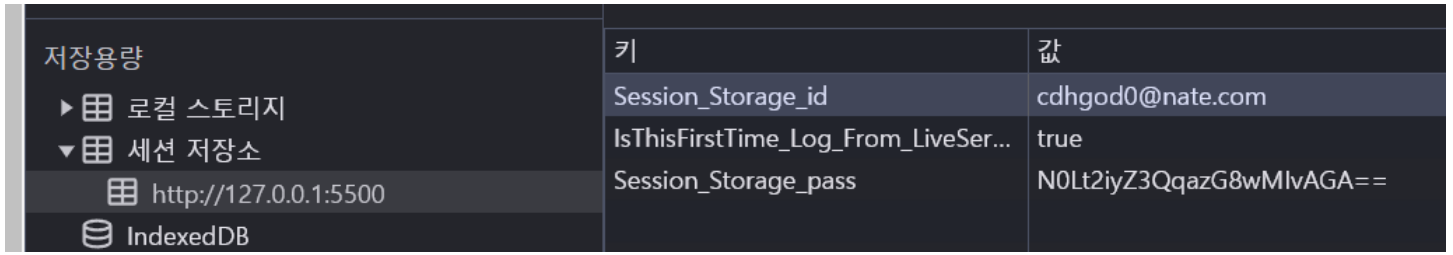


데이터 저장 – 세션(암호화/복호화)

- login.js의 세션 set 함수를 수정한다.
  - 패스워드를 세션에 추가 저장
    - 세션 이름 추가되어 id, pass로 이름 수정됨, 기존 코드는 주석처리

```
function session_set() { //세션 저장
  let session_id = document.querySelector("#typeEmailX"); // DOM 트리에서 ID 검색
  let session_pass = document.querySelector("#typePasswordX"); // DOM 트리에서 pass 검색
  if (sessionStorage) {
      let en_text = encrypt_text(session_pass.value);
      sessionStorage.setItem("Session_Storage_id", session_id.value);
      sessionStorage.setItem("Session_Storage_pass", en_text);

  } else {
      alert("로컬 스토리지 지원 x");
  }
}
```

입력한 패스워드를 암호화 한다.



  - 암호화된 데이터 저장
    - 다음 페이지....

데이터 저장 – 세션(암호화/복호화)

- Login.js의 복호화 함수를 추가한다.
  - 세션의 암호화된 값을 복호화 수행

```
function init_logined(){
  if(sessionStorage){
      decrypt_text(); // 복호화 함수
  }
  else{
    alert("세션 스토리지 지원 x");
  }
}
```

- index_login.html을 수정한다.
  - 페이지 로딩시에 복호화 수행 함수 호출

```
<body style="background-color: black;" onload="init_logined();">
```

  - 복호화 된 값 콘솔창 확인
    - 참고 : login.js, session.js 연결 확인

```
function session_get() { //세션 읽기
  if (sessionStorage) {
     return sessionStorage.getItem("Session_Storage_pass");
  } else {
      alert("세션 스토리지 지원 x");
  }
}
```

기존 아이디, 패스의 이름이 변경 되었다. 수정해주자.

```
function session_check() { //세션 검사
  if (sessionStorage.getItem("Session_Storage_id")) {
      alert("이미 로그인 되었습니다.");
      location.href='../login/index_login.html'; // 로그인된 페이지로 이동
  }
}
```

응용 문제 풀기 – NOW!!!!

- 세션 암호화 관련 문제
  - 웹 브라우저 내장 라이브러리
    - Web Crypto API 활용하기
      - 참고 : https 전용이지만 수행 가능
  - AES-256-GCM 대칭 암호 알고리즘 구현
    - Crypto2.js를 생성하고 암/복호화 함수를 정의한다.
      - 로그인, 로그인 후 페이지에 연결한다.
  - 주요 기능
    - 세션에 Session_Storage_pass2로 저장
    - 로그인 후 복호화도 구현

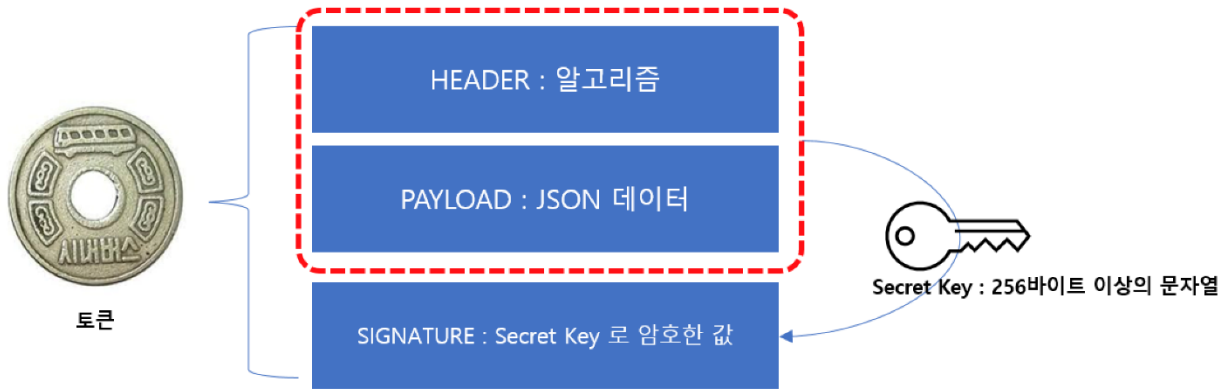
```
Async function() {
    Await ...
}
```

- 실습 결과 확인 – Q / A

키	값
IsThisFirstTime_Log_From_LiveServer	true
Session_Storage_id	cjjcjddjjd@nate.com
Session_Storage_pass	nKVaeWFHscsU5+Qi:X0Xvuo6m/H8W+bNmwFh/kjZ2U..
Session_Storage_pass2	3RhucgfGx5K9d8Xt:xSEtzNC//JUZ+t0tLXmHJ5gaDebma.

• 안전한 인증 – 보안 토큰

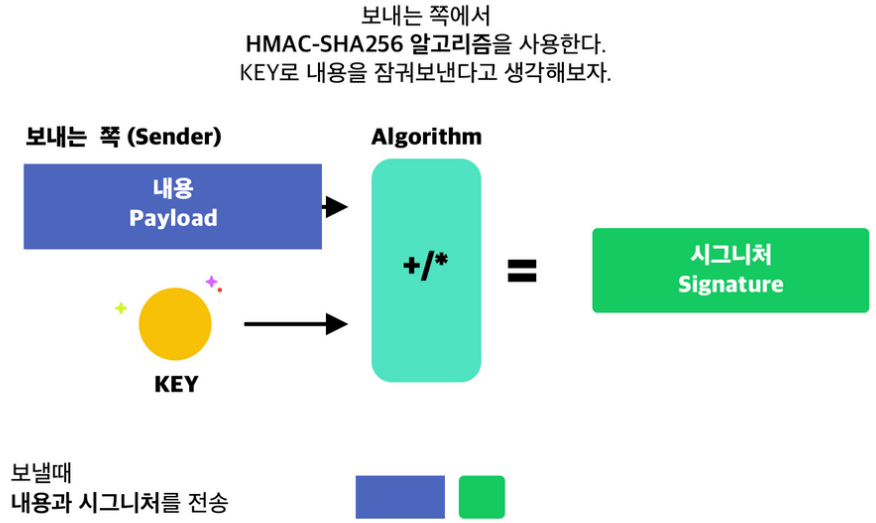
안전한 인증 – JWT



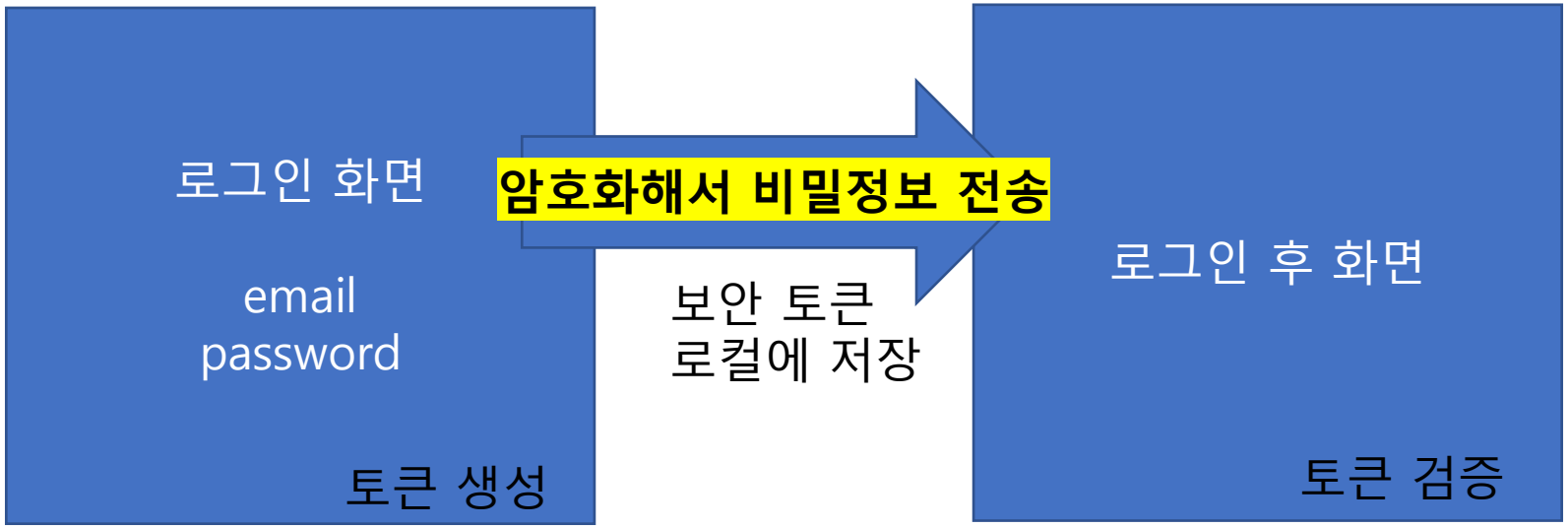
안전한 인증 – JWT

- 앞서 구현한 암호화/복호화
  - 주요 목적 : 데이터를 보호 저장

- JWT는? 로그인 후 상태 확인(지속 인증)
  - 로컬 저장, 보안 토큰 검증



해시 함수 : 단방향 생성, 복원 X



안전한 인증 – JWT

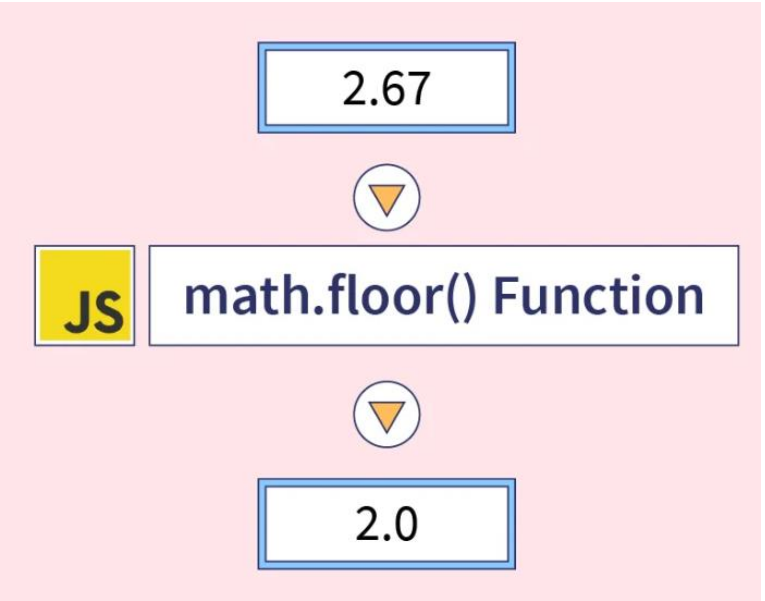
- 로그인을 수행하면 토큰 생성
  - Js 폴더에 login.js 파일을 수정한다.
    - check_input() 함수를 수정한다. 최상위 변수 추가

```
// 전역 변수 추가, 맨 위 위치
 const idsave_check = document.getElementById('idSaveCheck');
    const payload = {
        id: emailValue,
        exp: Math.floor(Date.now() / 1000) + 3600  // 1시간 (3600초)
    };
    const jwtToken = generateJWT(payload);
```

      - 세션 생성 이후 토큰을 로컬에 저장
        - 식별자 : 이메일

```
        console.log('이메일:', emailValue);
        console.log('비밀번호:', passwordValue);
        session_set(); // 세션 생성
        localStorage.setItem('jwt_token', jwtToken);
        loginForm.submit();
```

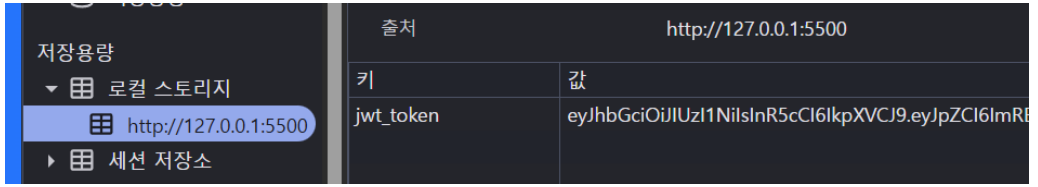
토큰 유지 시간 : 1시간



아래쪽 정수 내림 – 소수점 버림

안전한 인증 – JWT

F12 → 로컬 스토리지 확인 가능



- 새로운 자바스크립트 추가
  - Js 폴더에 jwt_token.js 파일을 추가한다.
    - login.html에 jwt_token.js를 head에 연동한다.
    - 참고 : 텍스트 전송에 적합한 BASE64(바이너리→ASCII)

```
// JWT 비밀 키 (실제 운영 환경에서는 복잡한 키 사용 필수)
const JWT_SECRET = "your_secret_key_here";

function generateJWT(payload) {
    // 1. 헤더 생성 및 Base64 인코딩
    const header = { alg: "HS256", typ: "JWT" };
    const encodedHeader = btoa(JSON.stringify(header));
    // 2. 페이로드 Base64 인코딩
    const encodedPayload = btoa(JSON.stringify(payload)); // JSON 형태로 변환 후 인코딩
    // 3. 서명 생성 (HMAC-SHA256 알고리즘 사용)
    const signature = CryptoJS.HmacSHA256(`${encodedHeader}.${encodedPayload}`, JWT_SECRET);
    const encodedSignature = CryptoJS.enc.Base64.stringify(signature);
    // 4. 최종 토큰 조합
    return `${encodedHeader}.${encodedPayload}.${encodedSignature}`;
}
```

Base64 Encoding Table

Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

인코딩 자체 : 보안과 관련 X

안전한 인증 – JWT

- jwt_token.js에 내용을 추가한다.
  - 참고 : JSON 문자열 형태로 변환

세션 + 토큰 조합으로 보안 강화

```
function verifyJWT(token) { // 토큰 검증
    try {
        // 1. 토큰을 헤더, 페이로드, 서명으로 분할
        const parts = token.split('.');
        if (parts.length !== 3) return null; // 형식 오류 체크

        const [encodedHeader, encodedPayload, encodedSignature] = parts;
        // 2. 서명 재계산 및 비교
        const signature = CryptoJS.HmacSHA256(`${encodedHeader}.${encodedPayload}`, JWT_SECRET);
        const calculatedSignature = CryptoJS.enc.Base64.stringify(signature);
        if (calculatedSignature !== encodedSignature) return null; // 서명 불일치
        // 3. 페이로드 파싱 및 만료 시간 검증
        const payload = JSON.parse(atob(encodedPayload)); // 디코딩 후 해석
        if (payload.exp < Math.floor(Date.now() / 1000)) { // 밀리초 단위
            console.log('보안 토큰이 만료되었습니다');
            return null;
        }
        return payload; // 검증 성공
    } catch (error) {
        return null; // 파싱 오류 또는 기타 예외 처리
    }
}
```

Encoded	Decoded	Parts

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
.eyJpc3MiOiIiLCJpYXQiOjE2NzYyMTc5NT
AsImV4cCI6MTcwNzc1Mzk1MCwiYXVkIjoi
YWthbWFpLWJsb2ciLCJzdWIiOiIiLCJjb21
wYW55IjoiQWthbWFpIiwidXNlciI6IkFrYW1
haS1yZWFkZXIiLCJhZG1pbiI6Im5vIn0.kM
Pz3Z7BSlBTJKijD8bcrpzTZejX7VCZ77w5
oQwJO6I
	{
	"typ": "JWT",
	"alg": "HS256"
	}
		Header

	{
	"iss": "",
	"iat": 1676217950,
	"exp": 1707753950,
	"aud": "akamai-blog",
	"sub": "",
	"company": "Akamai",
	"user": "Akamai-reader",
	"admin": "no"
	}
		Payload

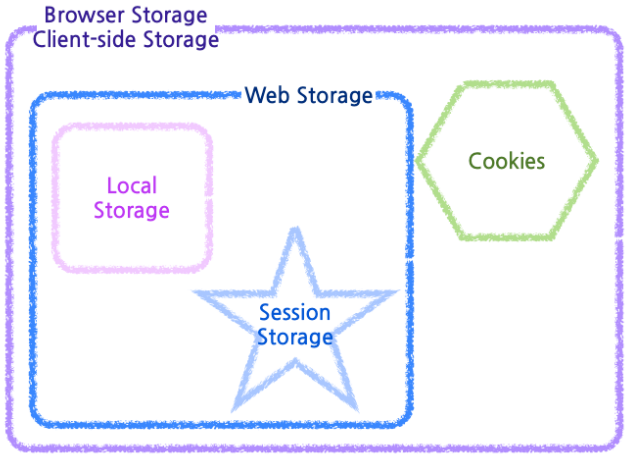
	HMACSHA256(base64Encode(header) + "." + base64Encode(payload), secret_key)	Signature

헤드, 페이로드, 서명으로 구성

안전한 인증 – JWT

- jwt_token.js에 내용을 추가한다.
  - 참고 : 로컬 스토리지로부터 토큰 확인

```
function isAuthenticated() { // 사용자 인증 상태 확인
    const token = localStorage.getItem('jwt_token');
    if (!token) return false; // 토큰 없음
    const payload = verifyJWT(token);
    console.log(payload);
    return !!payload; // 페이로드 유무로 인증 상태 판단
}

function checkAuth() { // 인증 검사 수행
    const authenticated = isAuthenticated(); // 한 번만 검증 호출

    if (authenticated) {
        alert('정상적으로 토큰이 검증되었습니다.');
    } else {
        alert('토큰 검증 에러!! 인증되지 않은 접근입니다.');
        window.location.href = '../login/login.html'; // 로그인 페이지 이동
    }
}
```



쿠키보다 로컬이 안전하다.

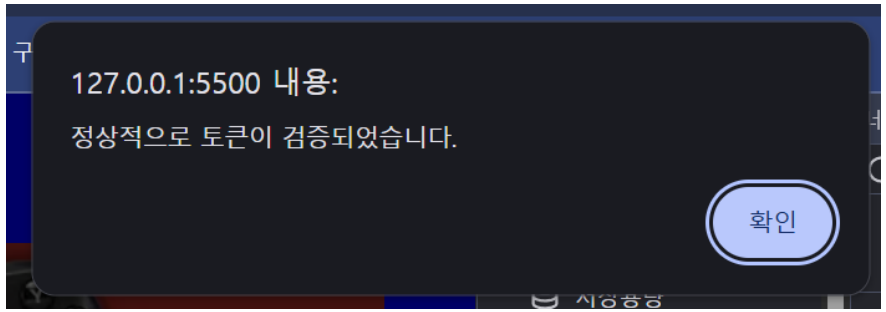
	쿠키 Cookie	로컬 스토리지 Local Storage	세션 스토리지 Session Storage
매번 요청마다 서버로 전송?	O	X	X
용량제한	4KB	모바일: 2.5MB 데스크탑: 5MB ~ 10MB	모바일: 2.5MB 데스크탑: 5MB ~ 10MB
어떻게 얻나요	document.cookie	window.localStorage	window.sessionStorage
영구적인가?	유효기간이 있음	사용자가 지우지 않는 한 영구적	윈도우나 브라우저 탭을 닫으면 제거
저장방식	Key-value	Key-value	Key-value

참고: https://www.zerocho.com/category/HTML&DOM/post/5918515b1ed39f00182d3048

안전한 인증 – JWT

- index_login.html를 수정한다.
  - 로그인 후 페이지 로딩과 동시에 토큰 검증
    - 참고 : jwt_token.js를 head에 연동한다.
  - Body 태그 속성으로 함수 호출
    - 참고 : 2개 이상 함수 호출 가능

```
<body style="background-color: blue;" onload="checkAuth(); init_logined();">
```



  - 로그인을 수행해보자.
  - F12 → 콘솔
    - 토큰 값 : 이메일, 타임 스탬프 확인 가능

- 11주차 연습문제

로그 아웃 기능 추가

로그 아웃 처리

- 로그 아웃 기능 추가
  - JWT 토큰(jwt_token)을 삭제하는 함수 구현하기
    - 기존에는 로그아웃과 동시에 쿠키, 세션을 삭제

  - 로그인 이후
    - 로컬스토리지의 토큰을 삭제하자.
    - 참고 : .removeItem 메소드를 활용한다.

- 실습 결과 확인 – Q / A

Q & A

- 다음주 할일
  - LOL 웹 사이트 구현(JS)
    - 로그인 창 : 세션 암호화

- 무엇이든 물어보세요!