Research & Technique

JWT 서명키 유출이 초래하는 인증 위협과 리스크 대응 전략

■ 서론

JWT 는 인증에 필요한 정보를 하나의 토큰에 담아 전달하는 방식이다. 서버는 이 토큰을 통해 사용자를 확인하므로, 사용자 상태를 별도로 저장하지 않아도 인증을 처리할 수 있다. 이러한 특징 때문에 JWT 는 규모가 큰 서비스나 여러 서버로 구성된 환경에서 널리 사용된다.

JWT 기반 인증에서는 서버가 요청을 받을 때, 전달된 토큰이 정상적으로 생성된 것인지를 확인한다. 이때 서버가 확인하는 핵심 요소는 토큰의 서명이다.

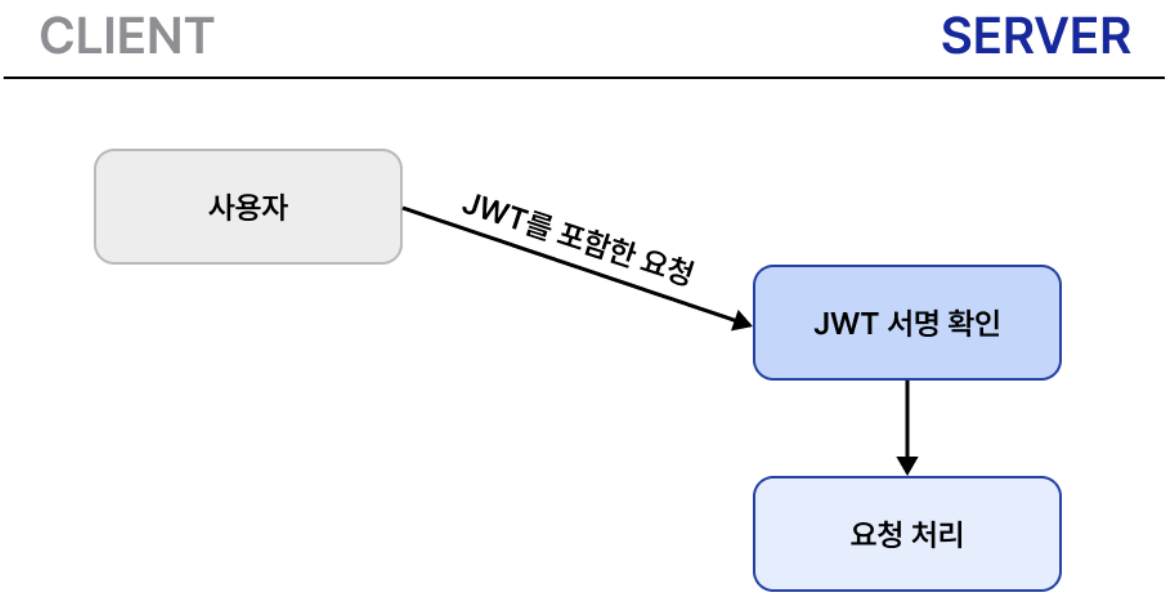


그림 1. JWT 기반 인증

이 구조에서는 서명 작성에 사용되는 서명키 하나에 인증의 신뢰가 집중된다. 따라서 서명키가 외부에 노출되거나 관리가 부실할 경우, 인증 체계 전체에 심각한 문제가 발생할 수 있다. 본 보고서는 JWT 의 이러한 특성을 바탕으로, 서명키 관리가 소홀할 때 어떤 문제가 발생하는지를 실제 사례를 통해 살펴보고자 한다.

■ JWT

1) JWT 란?

JWT 는 서버가 요청을 받을 때마다 “이 요청이 인증된 사용자로부터 온 정상 요청인지”, 그리고 해당 사용자가 “이 기능을 수행할 권한이 있는지”를 판단하기 위해 사용하는 토큰이다.

JWT 는 주로 로그인 성공 시 발급되어 이후 요청에 함께 전달되며, 서버는 이를 통해 사용자 식별과 권한 확인을 수행한다. 따라서 JWT 는 단순 로그인 상태 유지뿐 아니라 API 접근 제어, 서비스 간 인증(SSO)처럼 요청의 정당성을 증명해야 하는 환경에서 폭넓게 활용된다.

기존에는 서버가 “누가 어떤 상태인지”를 서버 내부에 저장해 두는 세션 방식을 통해, 요청이 올 때마다 저장된 정보를 조회하는 방식이 일반적이었다. 반면 JWT 방식은 서버가 상태를 계속 들고 있기보다는, 필요한 증명 정보를 토큰으로 묶어 사용자가 들고 다니게 한다. 이처럼 JWT 는 서버가 로그인 상태를 서버 메모리/DB 에 저장하지 않는 Stateless 구조를 전제로 하며, 요청이 올 때마다 토큰의 서명 검증 결과로 사용자를 확인한다.

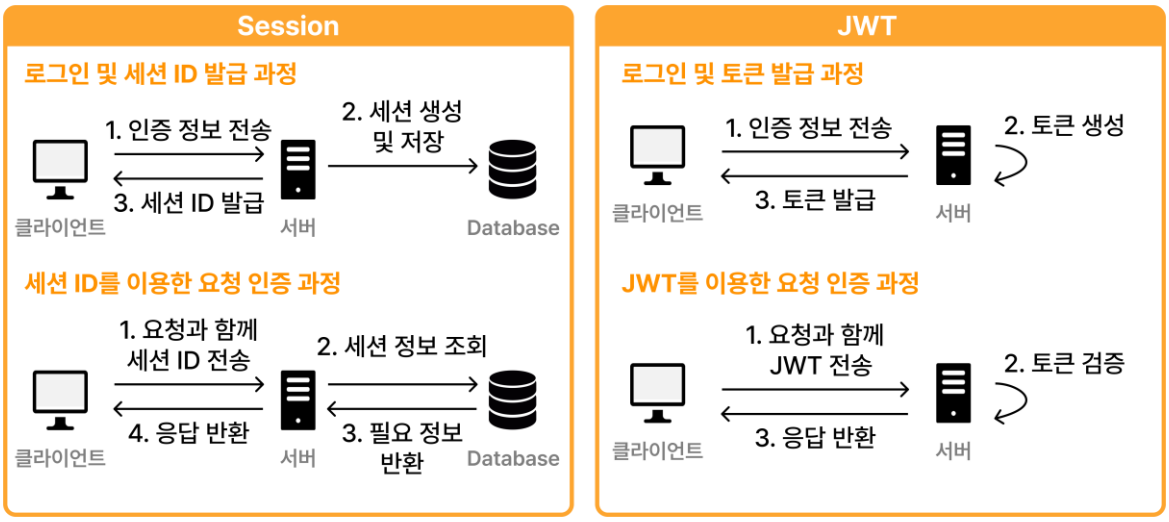


그림 2. 세션 및 JWT 발급 및 인증

2) JWT 구성 요소

JWT 는 점(.)으로 구분되는 세 개의 구성 요소로 이루어지며, 일반적으로 Header.Payload.Signature 형태로 표현된다. 각 구성 요소는 사람이 직접 읽기 위한 형식이 아닌, 데이터 전송 간 효율성을 위해 Base64Url 방식으로 인코딩된 문자열이다.

JWT 예시
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkVRU1QiLCJhZG1pbiI6dHJ1ZSwiaWF0IjoxNTE2MjM5MDIyLCJodHRwczovL3d3dy5za3NoaWVsZHVzLmNvbS8iOnRydWUsInVzZXJuYW1lIjoiRVFTVCBMYWIifQ.PYq5fttdbnGoTpSsLiSI0rSBB49hsc1mAP85jG1wiGA

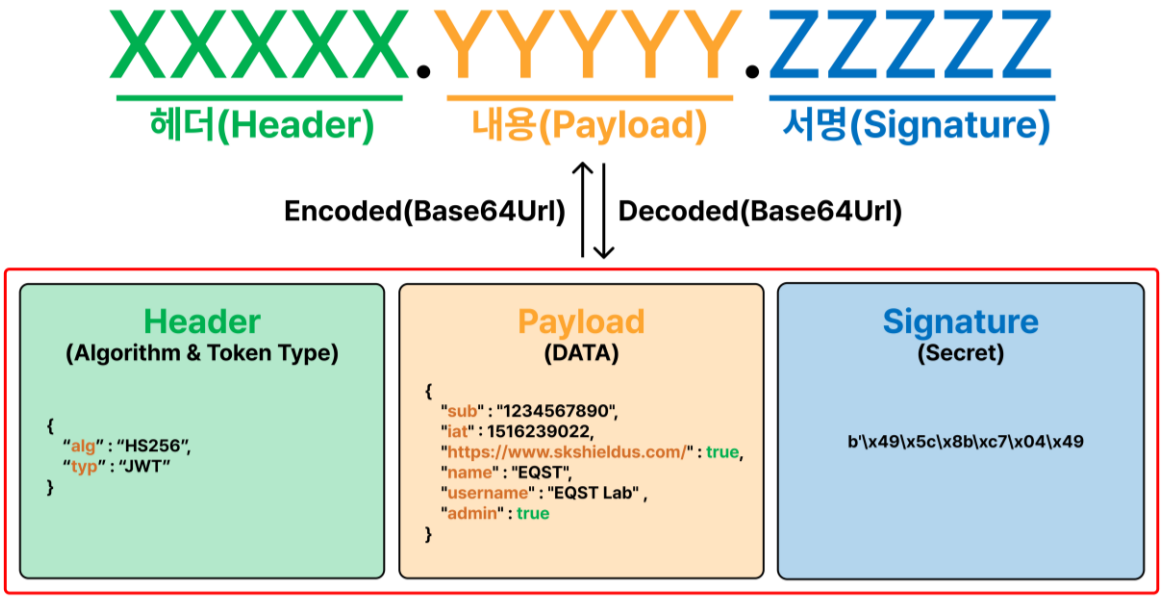


그림 3. JWT 구조

① Header(헤더)

헤더는 서명 알고리즘과 토큰 타입 정보를 포함하고 있다. 서버는 토큰을 검증할 때, 미리 허용해 둔 알고리즘 목록과 헤더에 명시된 알고리즘을 비교한 후, 일치하는 경우에만 해당 알고리즘을 사용해 시그니처(Signature)를 검증한다.

```
{
  "alg": "HS256",  // 서명 알고리즘
  "typ": "JWT"     // 토큰 타입
}
```

그림 4. JWT 헤더

② Payload(페이로드)

페이로드에는 토큰에 담을 실제 정보들이 포함되며, 이 정보의 각 항목을 클레임(Claim)이라 한다. JWT 표준(RFC 7519)에서는 클레임의 특성과 사용 범위에 따라 등록된 클레임, 공개 클레임, 비공개 클레임으로 구분된다.

```
{
  "sub": "1234567890",
  "iat": 1516239022,
  "https://www.skshieldus.com/": true,
  "name": "EQST",
  "team": "EQST Lab",
  "admin": true
}
```

그림 5. 디코딩 된 전체 페이로드

- **등록된 클레임(Registered Claim)**

등록된 클레임은 JWT 표준에서 의미와 용도가 사전에 정의된 클레임으로, 토큰 발급자, 유효 기간, 대상 등을 표현하기 위해 사용된다. 필수 항목은 아니지만, 토큰의 유효성 판단과 서비스 이용에 활용되기 때문에 실제 서비스 환경에서 사용되는 경우가 많다.

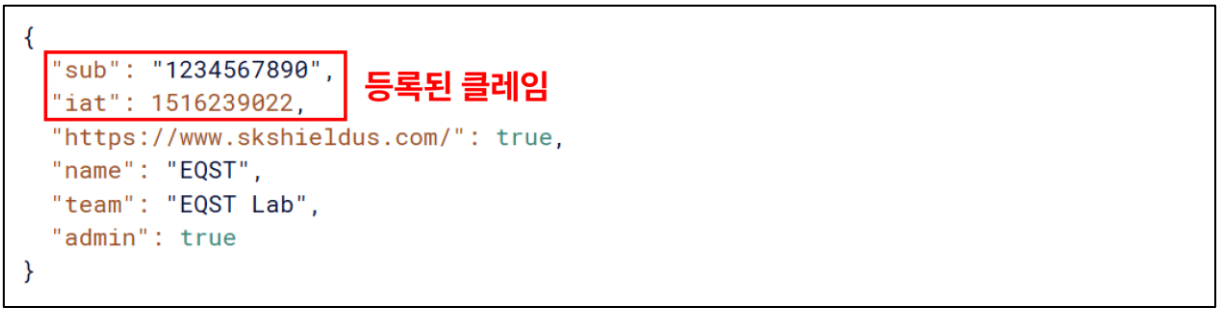


그림 6. 페이로드 내 등록된 클레임

JWT 표준에서 정의한 등록된 클레임 종류는 다음과 같다.

약어	클레임명(Full Name)	설명
iss	Issuer	토큰 발급자
sub	Subject	사용자 고유 식별자
aud	Audience	토큰 수신자
exp	Expiration Time	토큰 만료 시간(Unix Time)
nbf	Not Before	토큰 활성 시작 시간
iat	Issued At	토큰 발급 시간
jti	JWT ID	토큰 고유 식별자

표 1. 등록된 클레임 종류

• **공개 클레임(Public Claim)**

공개 클레임은 여러 서비스가 같은 JWT 를 함께 사용할 때, 서로 다른 서비스가 같은 이름의 클레임을 써서 충돌하는 일을 막기 위해 사용된다. 이를 위해 클레임 이름을 “https://www.skshieldus.com/”처럼 URI 형태로 길게 만들어 어느 조직 또는 서비스의 정의인지를 구분한다. 즉, 클레임 이름을 URI 형식으로 정의한 경우 해당 클레임은 공개 클레임으로 분류된다.

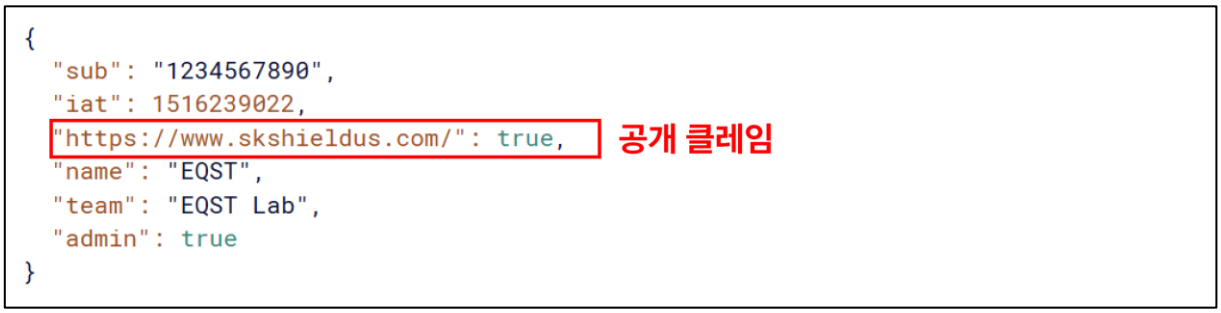


그림 7. 페이로드 내 공개 클레임

• **비공개 클레임(Private Claim)**

비공개 클레임은 JWT 표준에 의해 의미가 정의되어 있지 않아, 클레임 이름과 값 모두 서비스 요구사항에 따라 자유롭게 설계할 수 있다. 이로 인해 서버의 인증 및 인가 로직에 필요한 정보를 서비스 특성에 맞게 직접 포함하는 용도로 주로 활용된다.

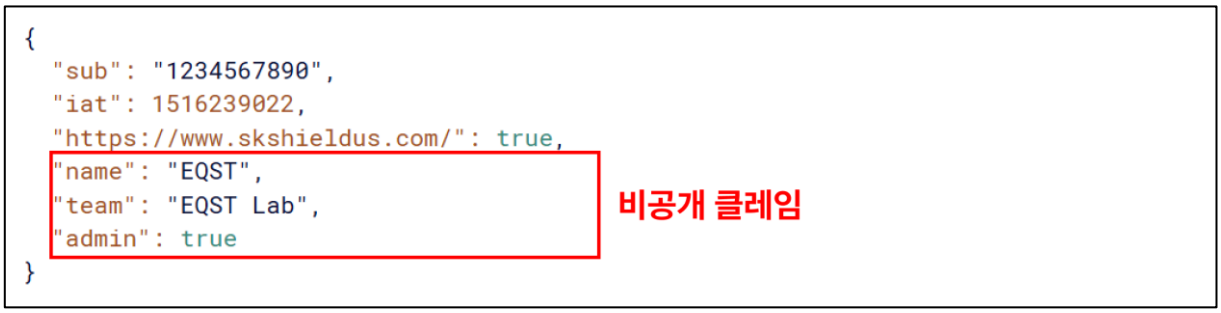


그림 8. 페이로드 내 비공개 클레임

③ Signature(시그니처)

시그니처는 JWT 가 전송 과정에서 위·변조되지 않았음을 확인하고, 서버가 신뢰하는 발급 주체가 생성한 토큰인지 판단하기 위한 서명 값이다. JWT 의 헤더와 페이로드는 누구나 쉽게 디코딩 해 내용을 확인할 수 있으므로, 토큰의 신뢰 여부는 시그니처 검증 결과에 의해 결정된다.

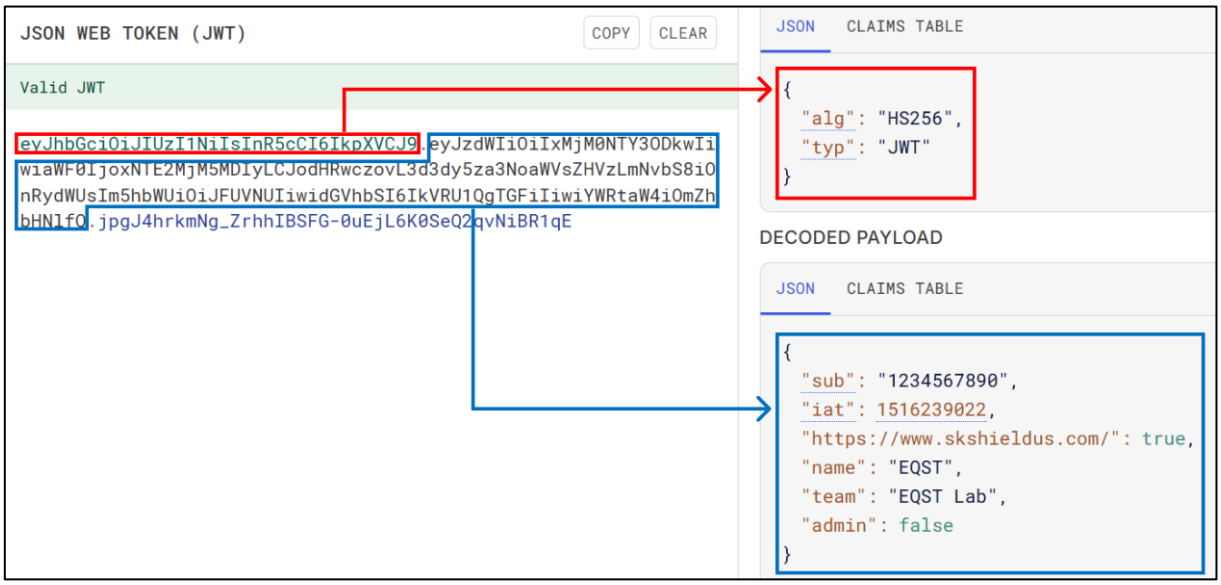


그림 9. 디코딩 된 JWT

시그니처는 JWT 의 헤더와 페이로드를 각각 Base64Url 인코딩한 뒤, 두 값을 점(.)으로 연결한 문자열을 해시 함수의 입력값 중 하나로 사용해 생성된다. 따라서 헤더나 페이로드가 조금이라도 변경되면 입력값이 달라지고, 서버의 검증 과정에서 시그니처가 일치하지 않아 토큰이 거부된다.

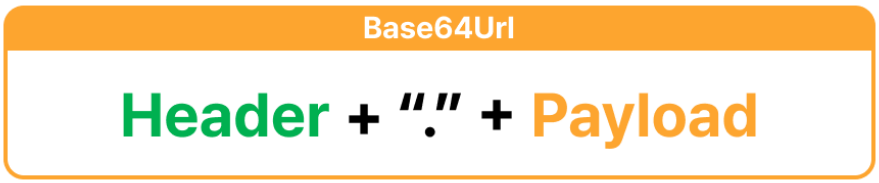


그림 10. 시그니처 입력값

이때 서버가 입력값이 바뀌었는지를 판단할 때 핵심이 되는 성질이 해시 함수의 특성이다. 해시 함수는 임의 길이의 입력 데이터를 고정 길이의 값으로 변환하며, 입력값이 조금만 바뀌어도 결과값이 완전히 달라진다. 그래서 토큰 내용이 변경되면 입력값이 달라지고, 결과적으로 시그니처 검증도 실패하게 된다.

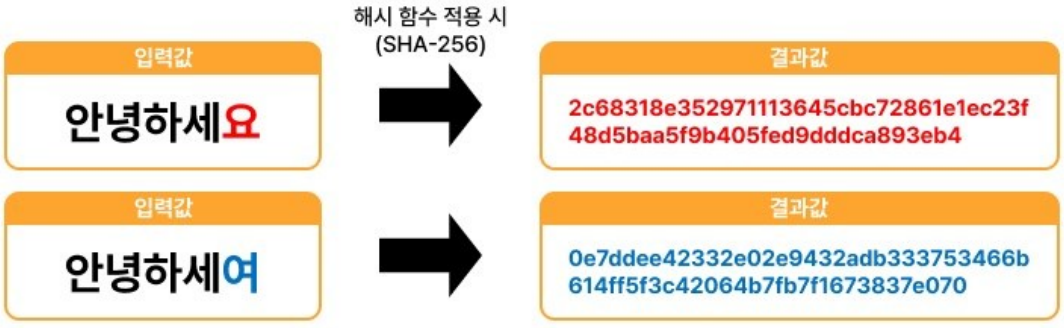


그림 11. 해시 예시

하지만 해시만으로는 인증을 성립시킬 수 없다. 해시는 입력값만 알면 누구나 같은 결과를 계산할 수 있으므로, 공격자도 페이로드를 바꾼 뒤 그에 맞는 해시 값을 다시 계산해 붙일 수 있기 때문이다. 따라서 JWT 는 단순 해시가 아니라, 키를 사용하는 서명 알고리즘으로 시그니처를 생성한다. 즉 "서버가 가진 키로만 만들 수 있는 값"을 시그니처로 사용함으로써, 토큰이 변조되지 않았을 뿐 아니라 서버가 신뢰하는 키로 서명된 토큰인지까지 확인한다.

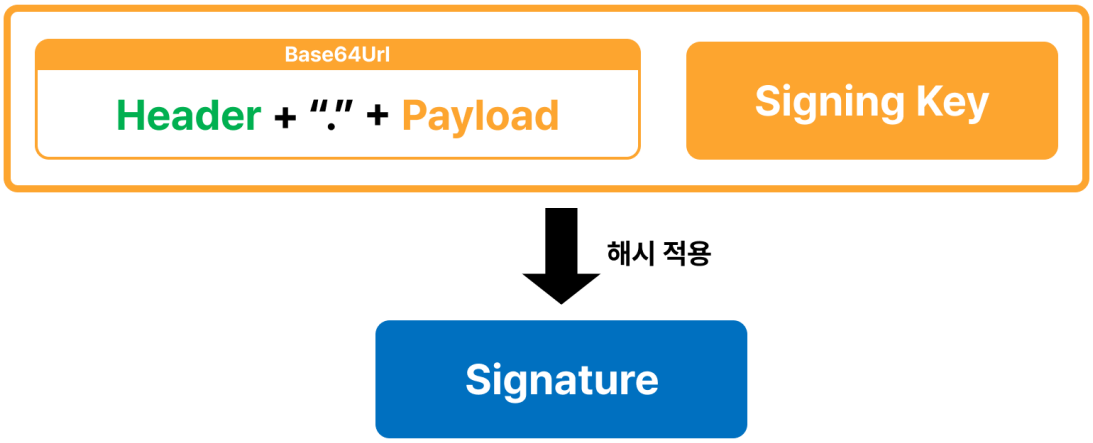


그림 12. JWT 시그니처 생성 과정

그렇기에 JWT 의 신뢰는 시그니처 생성 시 사용되는 키가 안전하게 보호된다는 전제에서만 유지된다. 서명키가 유출되는 등의 이유로 신뢰를 잃게 될 경우, JWT 인증 체계가 흔들릴 수 있으며, 이를 이어지는 내용에서 실제 사례와 시나리오를 통해 설명한다.

■ 서명키 유출로 인한 피해 사례

앞서 살펴본 것처럼 JWT 는 시그니처 검증 결과로 신뢰 여부를 판단한다. 이 때문에 시그니처를 생성하는 서명키의 보호와 관리가 보안의 핵심이 된다. 아래 사례들은 JWT 인증 기반 구조에서 서명키 관리에 실패했을 때 어떤 문제가 발생하는지 보여준다.

1) 쿠팡 사용자 개인정보 유출(2025) – API 인증 우회로 인한 대량 개인정보 유출

2025 년 쿠팡에서는 내부 인증 체계에 사용되던 JWT 서명키 관리 문제와 연관된 대규모 개인정보 유출 사고가 발생했다. 유출 규모는 약 3,370 만 건으로 알려졌다. 언론 보도에 따르면, 퇴사자가 재직 중 취득한 서명키가 악용되었고, 퇴사 이후 서명키 폐기 및 갱신 등의 관리가 미흡했던 점이 지적됐다.

공격자는 JWT 의 헤더와 페이로드를 임의로 구성한 뒤, 취득한 서명키로 서명을 생성함으로써 서버 측 검증을 통과하는 토큰을 만들어낼 수 있었고, 개인정보 조회 API의 인증을 우회하여 다수 계정의 개인정보에 접근할 수 있었다.

해당 접근은 단기간의 일회성 시도가 아닌 일정 기간 지속적으로 발생한 것으로 알려졌으며, 공격이 종료되기 전까지 위조된 JWT 를 이용한 비정상적인 API 호출이 즉각적으로 차단되지 않았다. 이 사건은 JWT 인증 구조에서 서명키가 유출될 경우, 서버가 해당 요청을 정상 사용자 요청과 구분하기 어렵다는 점을 실제 사고를 통해 보여준 사례이다.

2) Microsoft Storm-0558(2023) – 위조 토큰을 통한 메일함 열람

2023 년 Microsoft 는 “Storm-0558”이라는 중국계 위협 그룹이 Outlook 및 Exchange Online 서비스에 접근하기 위해 위조된 인증 토큰을 사용한 사실을 공식 발표했다. Microsoft 의 후속 보고에 따르면, 공격자는 Microsoft 개인 계정(MSA)에서 로그인 이후 발급되는 인증 토큰에 사용되던 서명키를 획득했고, 이를 이용해 정상 사용자로 로그인한 것처럼 보이는 인증 토큰을 직접 생성했다.

서비스는 해당 토큰을 정상적인 인증 요청으로 인식했기 때문에, 공격자는 특정 사용자의 권한으로 메일 조회 API 에 접근할 수 있었다. 이 사건은 사용자 인증 정보를 서버 세션이 아닌 서명된 토큰에 담아 처리하던 구조에서, 그 토큰을 생성하는 키가 유출되며 인증 체계 전체가 무력화된 사례이다.

이는 JWT 를 대표로 하는 stateless 토큰 인증 환경에서 서명키 하나가 인증 신뢰의 중심에 있으며, 해당 키가 노출될 경우 인증 전반이 붕괴될 수 있음을 보여준다.

3) Solorigate/Golden SAML[1] (2020) – SAML 토큰 위조를 통한 SSO 인증 우회

2020 년 공개된 SolarWinds 공급망 침해(Solorigate) 대응 과정에서, Microsoft 는 공격자가 일부 피해 조직의 ADFS[2] 를 장악한 뒤 ADFS 의 토큰 서명 인증서(개인키)를 탈취해 SAML 로그인 토큰을 위조하고 정상 인증처럼 통과시키는 행위를 확인했다고 안내했다. 공격자는 이 인증서를 이용해 임의 사용자로 발급된 것처럼 보이는 SAML 토큰을 생성하고 서명할 수 있으며, 서비스는 서명이 유효한 것으로 인식해 이를 정상 로그인 결과로 받아들이게 된다.

이 사례는 JWT 가 아니라 SAML 형식 토큰에서 발생한 문제이다. 그러나 서명된 토큰을 신뢰하는 구조에서는, 토큰 형식을 불문하고 서명을 생성하고 검증하는 데 사용되는 키(서명키/서명 인증서) 관리가 중요하다는 것을 보여준다.

■ 유출된 서명키를 통한 JWT 위조

앞선 사례들은 서명키가 유출되면 공격자가 토큰을 직접 만들어 인증을 우회할 수 있음을 보여준다. 서버가 서명 검증을 신뢰의 핵심 기준으로 삼는 구조에서는, 외부에서 생성된 토큰이라도 동일한 키로 서명되면 정상 토큰과 구분하기 어렵다. 이 때문에 앞선 사례에서도 침해 사실이 즉시 인지되지 않았으며, 사후 조사 및 분석을 통해 확인되었다. 본 챕터에서는 이를 JWT 검증 기준과 위조 과정의 관점에서 설명한다.

서버는 일반적으로 서명 검증을 1 차 신뢰 기준으로 삼아 토큰의 위·변조 여부를 판단한다.

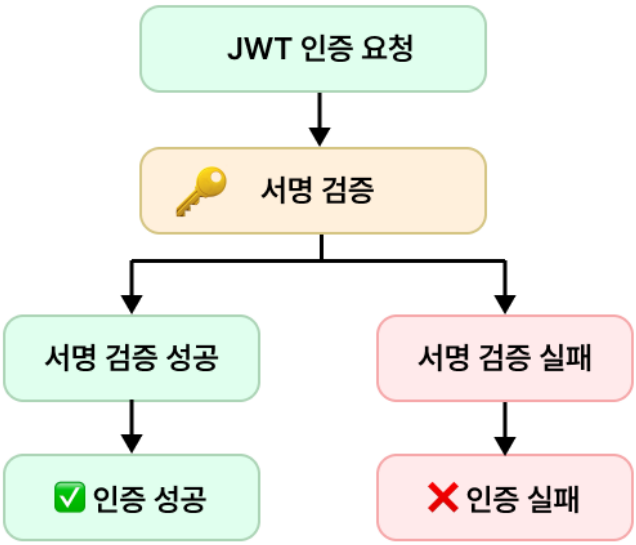


그림 13. JWT 검증 기준

[1] SAML (Security Assertion Markup Language): SSO 환경에서 인증 결과를 XML 기반 토큰으로 전달하는 표준

[2] ADFS (Active Directory Federation Service): 회사 계정으로 한번 로그인하면, 그 사실을 증명하는 토큰을 발급해 여러 서비스에 자동 로그인되게 해주는 윈도우 서버 기능

서명키가 유출된 상황에서는 공격자도 동일한 방식으로 서명값을 생성할 수 있게 된다. 이로 인해 JWT 기반 인증 환경에서는, 서명키 보유자가 곧 토큰 발급자와 동일한 권한을 갖게 된다. 정상적인 인증 서버에서 발급된 토큰과, 외부에서 동일한 서명키로 생성된 토큰을 구분할 수 없기 때문이다.

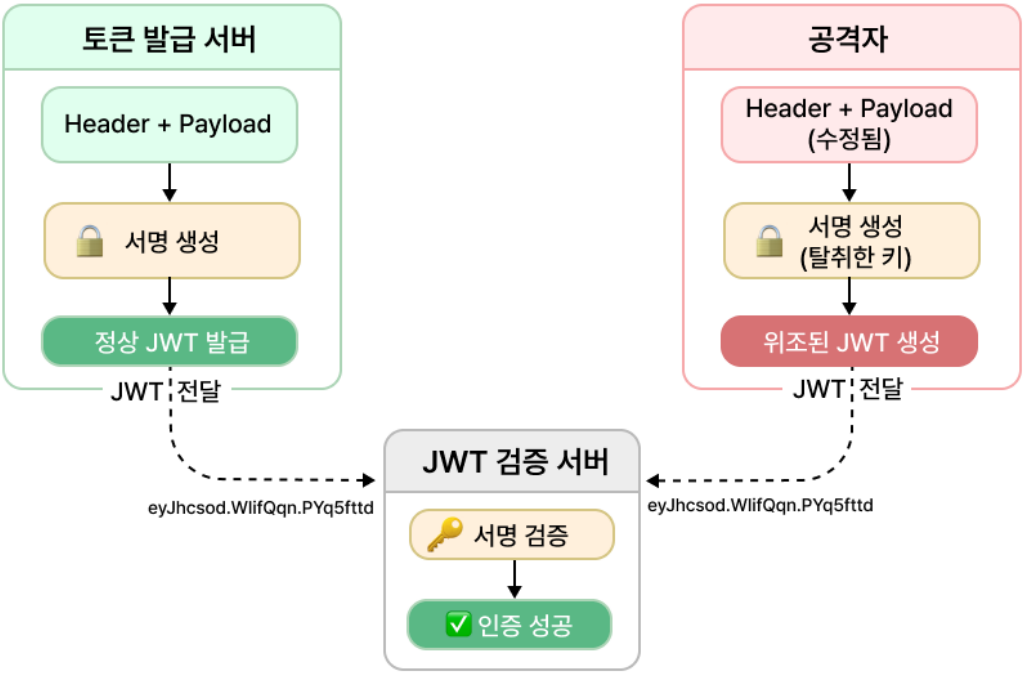


그림 14. 서명키를 이용한 JWT 위조(생성)과정

특히 JWT 의 페이로드는 암호화되지 않은 상태로 전달되므로, 토큰 구조를 파악하거나 내부 클레임 값을 확인할 수 있다.

```
{
  "sub": "1234567890",
  "iat": 1516239022,
  "https://www.skshieldus.com/": true,
  "name": "EQST",
  "team": "EQST Lab",
  "admin": false
}
```

그림 15. JWT 페이로드

공격자가 서명키를 확보한 경우, 기존 토큰의 페이로드를 참고하여 사용자 식별자 또는 권한 정보를 변경한 뒤, 동일한 키로 서명된 새로운 JWT 생성이 가능하다.

그림 16. 페이로드가 수정된 JWT 생성

이처럼 JWT 기반 인증은 서명 검증을 신뢰 기준으로 삼기 때문에, 서명키가 외부에 노출되면 동일 서명을 통한 토큰 위조 시도가 가능한 환경이 된다.

■ 공격 시나리오

JWT 가 공격자에 의해 위조될 경우, 다양한 문제들이 발생할 수 있다. 본 챕터에서는 그 예시를 설명한다.

1) API 인증 우회

쇼핑몰 서비스에서 개인정보 조회, 주문 내역 확인처럼 사용자별 데이터를 반환하는 API 는 요청에 포함된 JWT 로 요청자를 식별하는 경우가 많다. 실제로 아래 요청 헤더에서 확인할 수 있듯, 개인정보 조회 및 수정 API 호출 시 쿠키에 JWT 형태의 토큰이 포함되어 전달된다.

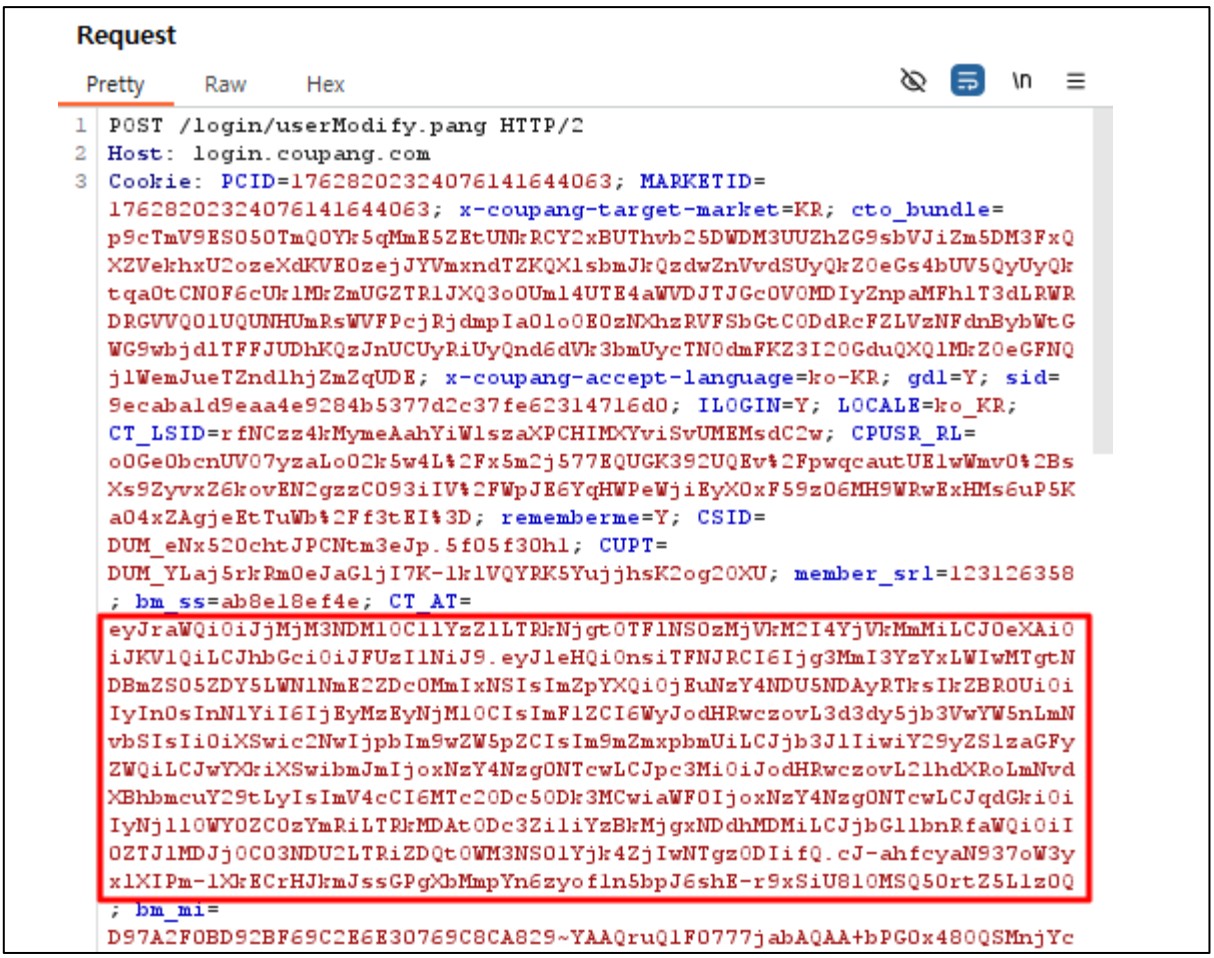


그림 17. 개인정보확인 API 요청에 존재하는 JWT

해당 JWT 를 디코딩하면 sub 와 같은 사용자 식별 정보가 포함되어 있음을 확인할 수 있다. 일부 구현에서는 서버가 요청 처리 과정에서 이 sub 값을 조회 대상 사용자 식별에 직접 사용한다.

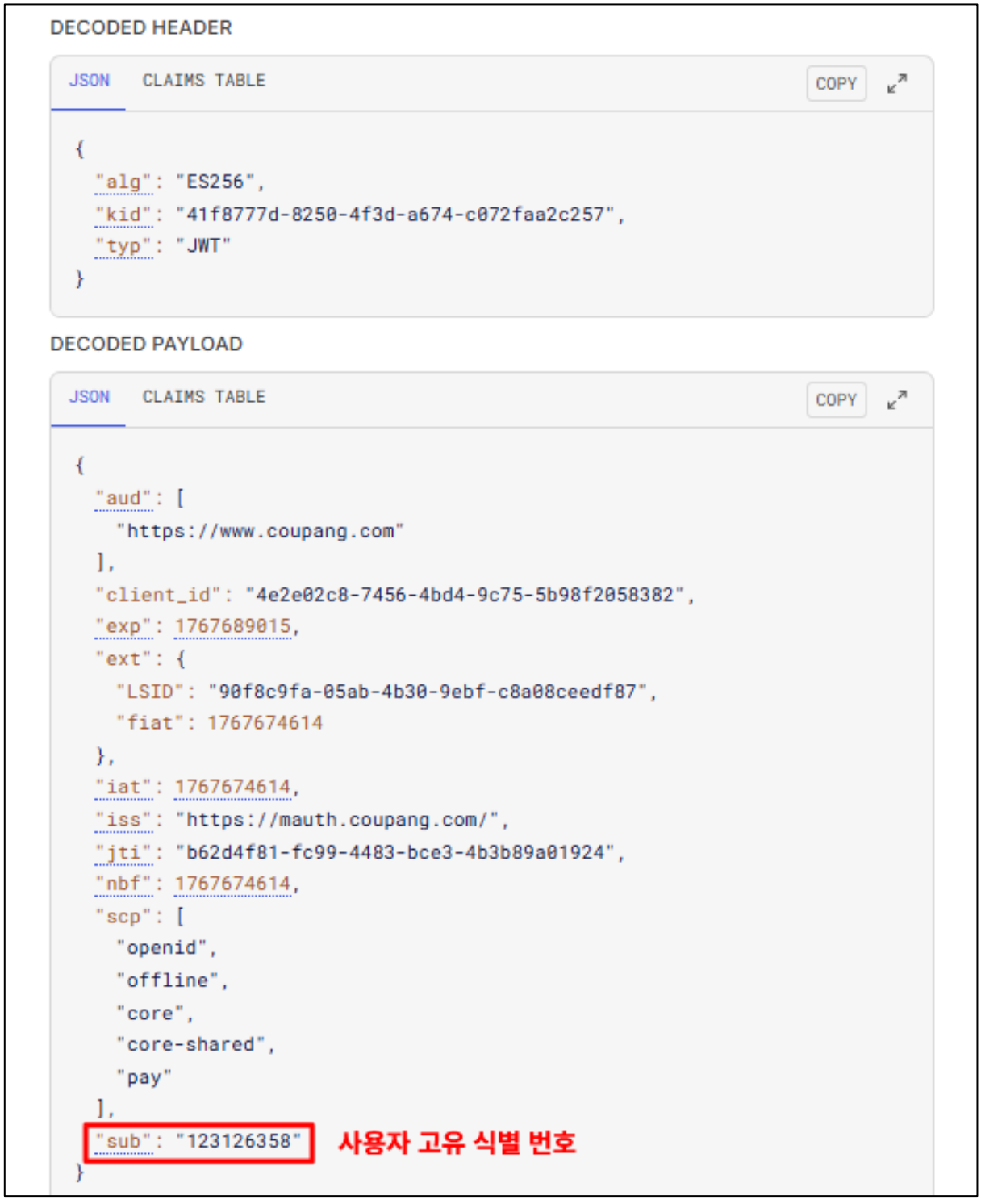
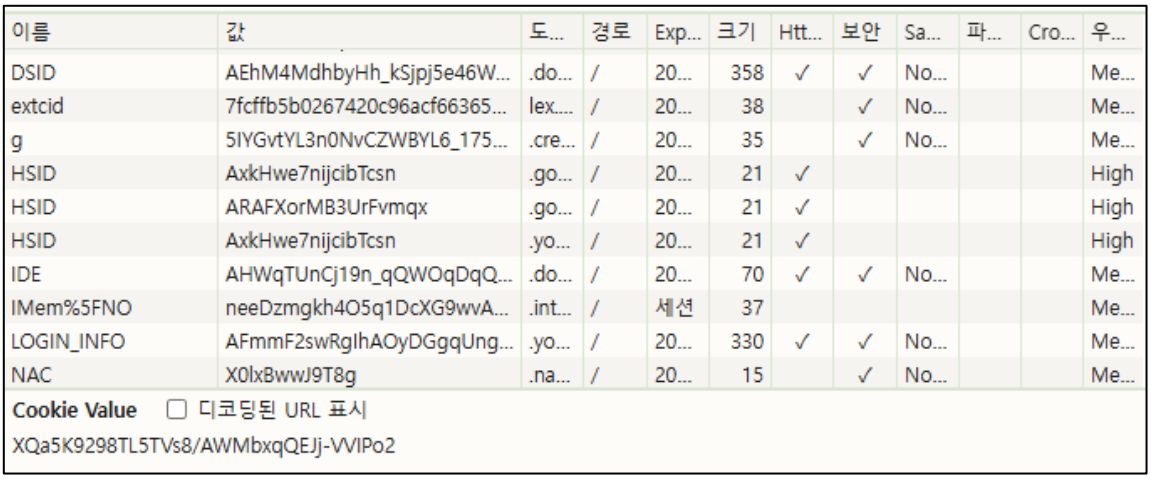


그림 18. JWT 디코딩 결과

따라서 서명키가 유출된 상황에서는 공격자가 sub 값을 다른 사용자 식별자로 변경한 뒤, 유효한 서명을 가진 JWT 를 새로 생성하여 동일 API 를 호출할 수 있다. 이러한 방식은 자동화와 결합되면 연속적인 사용자 식별자 대입을 통해 대량 조회로 확대될 수 있어, 개인정보 유출 규모가 급격히 커질 수 있다.

2) 로그인 우회

일부 서비스는 로그인 성공 후 JWT 를 쿠키에 저장하고, 이후 요청에서는 해당 토큰의 유효성만으로 로그인 상태를 판단한다. 로그인 전에는 아래와 같이 JWT 가 존재하지 않는다.



이름	값	도...	경로	Exp...	크기	Htt...	보안	Sa...	파...	Cro...	우...
DSID	AEhM4MdhbyHh_kSjpj5e46W...	.do...	/	20...	358	✓	✓	No...			Me...
extcid	7fcffb5b0267420c96acf66365...	lex....	/	20...	38		✓	No...			Me...
g	5IYGvtYL3n0NvCZWBYL6_175...	.cre...	/	20...	35		✓	No...			Me...
HSID	AxkHwe7nijcibTcsn	.go...	/	20...	21	✓					High
HSID	ARAFXorMB3UrFvmqx	.go...	/	20...	21	✓					High
HSID	AxkHwe7nijcibTcsn	.yo...	/	20...	21	✓					High
IDE	AHWqTUnCj19n_qQWOqDqQ...	.do...	/	20...	70	✓	✓	No...			Me...
IMem%5FNO	neeDzmgkh4O5q1DcXG9wvA...	.int...	/	세션	37						Me...
LOGIN_INFO	AFmmF2swRgIhAOyDGgqUng...	.yo...	/	20...	330	✓	✓	No...			Me...
NAC	X0lxBwwJ9T8g	.na...	/	20...	15		✓	No...			Me...

Cookie Value ☐ 디코딩된 URL 표시

XQa5K9298TL5TVs8/AWMbxqQEJj-VVIPo2

그림 19. 로그인 전 쿠키

로그인 후에는 id_token 이라는 JWT 가 새로 생성되어 쿠키에 저장되는 것을 확인할 수 있다.

이름	값	도...	경로	Exp...	크기	Htt...	보안	Sa...	파...	Cro...	우...
DSID	AEhM4MdhbyHh_kSjpj5e46W...	.do...	/	20...	358	✓	✓	No...			Me...
extcid	7fcffb5b0267420c96acf66365...	lex....	/	20...	38		✓	No...			Me...
g	5IYGvtYL3n0NvCZWBYL6_175...	.cre...	/	20...	35		✓	No...			Me...
HSID	AxkHwe7nijcibTcsn	.go...	/	20...	21	✓					High
HSID	ARAFXorMB3UrFvmqx	.go...	/	20...	21	✓					High
HSID	AxkHwe7nijcibTcsn	.yo...	/	20...	21	✓					High
id_token	eyJraWQiOiJfTmVvUDZlX1Nl...	.int...	/	세션	1032	✓	✓	Lax			Me...
IDE	AHWqTUnCj19n_qQWOqDqQ...	.do...	/	20...	70	✓	✓	No...			Me...
IMem%5FNO	neeDzmgkh4O5q1DcXG9wvA...	.int...	/	세션	37						Me...
interparkSNO	Vxv7t0P%2F85qVE2iXyuxx9p0...	.int...	/	세션	60		✓	Lax			Me...

Cookie Value ☐ 디코딩된 URL 표시

eyJraWQiOiJfTmVvUDZlX1NlWkxkVnpQZk9NbUVRb09CUE9uR3ZNZmdQUUNZc0l6ZFJBIiwiYWxnIjoiUlMyNTYifQ.eyJhY2Nlc3Nf
dG9rZW4iOiJXeGtnd3g4akxtZ3hrWGE5WmlmWVVkS1BCZzJcL0F1aXVmRjR3d1VMbVBkVVdhRlhyV1hBTURGMWJqN0N4SlN3R
yIsInN1YiI6IjdYXC9RNlhRXC9jZExVYVdtZURlek9uZz09IiwiYXVkIjoidGlja2V0LXBjIiwicmVmcmVzaF90b2tlbiI6IkpsbDZJbmtlUzNLa
XdkTEh3WTl4SjdxcE1EOUkzOG1BbjhtenJmTHFaWng3RVh5SzdHYlBPYlVZRGQycDdsdUEiLCJpc3MiOiJodHRwczpcL1wvYWNjb3
VudHMuaW50ZXJwYXJrLmNvbSIsInJlbWVtYmVyX21lIjpmYWxzZSwiZXhwIjoxNzY4NTMyMjUxLCJpc19ub2xfY29ubmVjdGVkIjp0
nVlLCJpYXQiOjE3Njg1MjY4NTEsInZlcnNpb24iOiIyLjAiLCJtbm8iOiJuZWVEem1na2g0TzVxMURjWEc5d3ZBPT0iLCJzaWQiOiJWeH
Y3dDBQXC84NXFWRTJpWHl1eHg5cDBwM1NnRFVjb3JuOVk1ZDBicVljWT0ifQ.o9mos9k12ZFyrnEkm7Ls3xug3p3VG3-dm9B4DY
47J1rLNkJODAUxC1B8PileajHQD5WUlMDQNFUmCSr0xM5vBKnP5ZWa3FDZSA_o0e31J4yEGP5QSYMK7_zc_LE2ez25UX3QDdoq
dlMpMxKDfq2y3KGfYhJj39-Id-jqlf-_40PgefhIfeZCbAbCZyfHPPMyTZ-EmclCS2MjWUyf6thOztla7t-e3YBNaLZooP7etlS50_0yD-LC
RmRh38mb6ADd8Fz7mGrB768GXvQ32LnBc9Xk7rAF4gDyNhUZXTFokcxGnd9yPe69AFAJMKAh8vYDyfqjCgCiqAu-yPG5AiThA

그림 20. 로그인 후 JWT 토큰이 생성된 모습

이 id_token 를 디코딩해보면 sub/iat/exp 같은 기본 클레임 외에도 access_token, refresh_token, sid 처럼 로그인 상태를 구성하고, 유지하기 위한 값이 함께 포함된 것을 확인할 수 있다. 즉 이 토큰은 서비스가 로그인 상태를 판단하고 유지하는 데 사용하는 세션 대체 수단으로 동작할 수 있다.

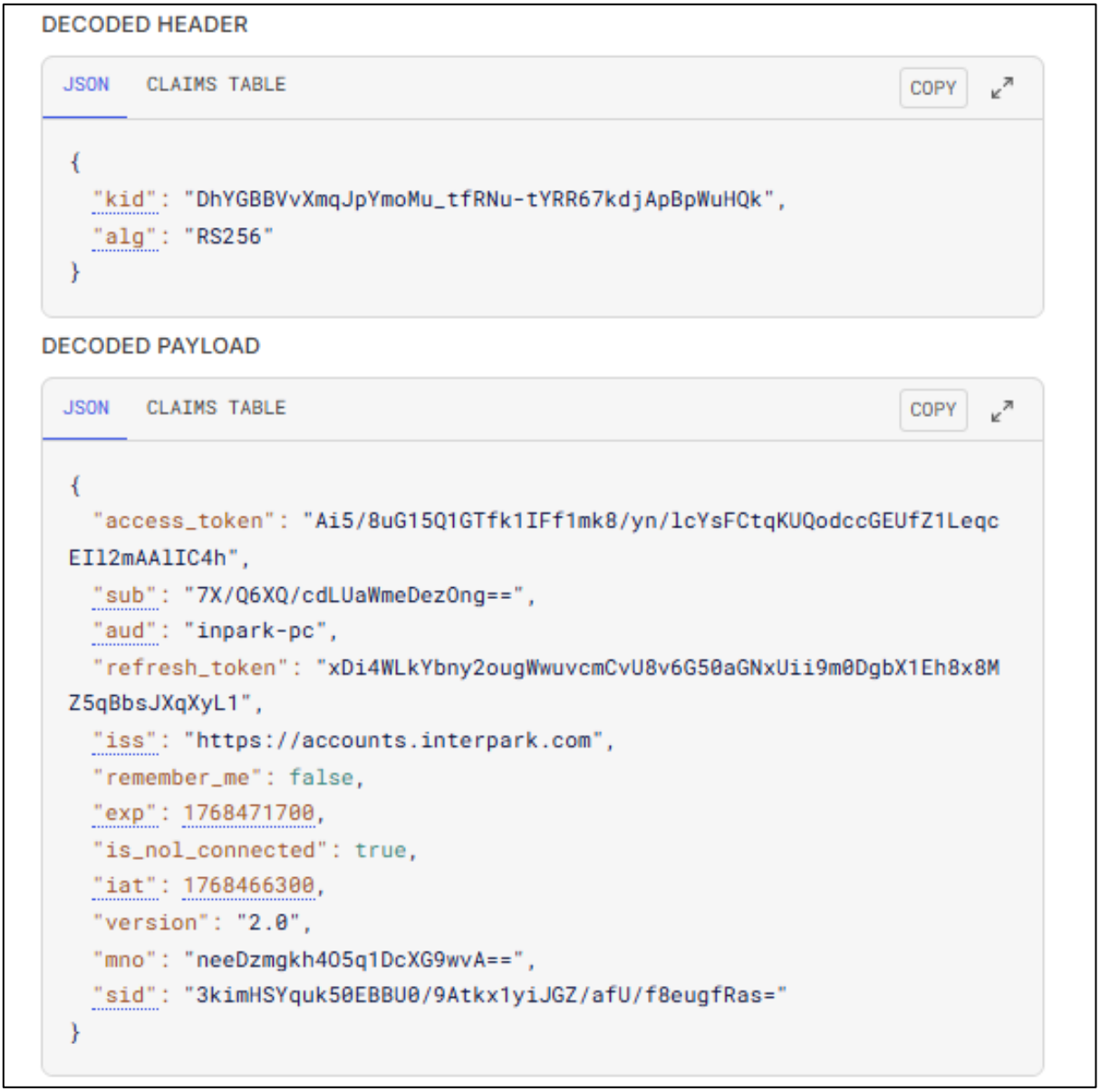


그림 21. JWT 디코딩 결과

서명키가 유출되면 공격자는 로그인 절차 없이도 유효한 id_token 을 생성해 주입할 수 있으며, 서비스는 이를 정상 로그인 결과로 인식한다. 즉, 로그인을 하지 않았음에도 로그인 한 것처럼 행동하는 것이 가능해진다.

3) 연계 로그인 우회

SSO(연계 로그인) 환경에서는 사용자가 서비스에 직접 ID/PW 입력하는 대신, 외부 인증 시스템이 로그인 결과를 토큰 형태로 발급하고 서비스는 이를 검증해 로그인 처리한다. 이때 로그인 결과 토큰이 JWT 형태로 전달되는 경우가 많으며, 서비스는 토큰의 서명 유효성 및 기본 클레임을 확인해 "정상 발급된 로그인 토큰"인지 판단한다.

만약 공격자가 SSO 토큰에 사용되는 서명키를 확보하거나, 토큰 검증 로직의 허점을 이용해 서명 검증을 우회할 수 있다면, 정상 인증 과정을 거치지 않고도 특정 사용자로 로그인한 것처럼 보이는 토큰을 만들어 서비스에 제출할 수 있다. 서비스는 서명이 유효한 토큰을 정상 로그인 결과로 받아들이기 때문에, 결과적으로 공격자는 SSO 로 연결된 계정의 권한으로 서비스에 접근하는 것이 가능해진다.

특히 SSO 는 여러 서비스가 동일한 인증 체계를 공유하는 구조인 경우가 많아, 하나의 서명키가 침해되면 단일 서비스에 그치지 않고 연동된 다른 서비스까지 영향이 확대된다. 이처럼 연계 로그인 환경에서의 JWT 위조는 "개별 서비스의 인증 우회"를 넘어 "인증 체계 전반의 신뢰 붕괴"로 이어질 수 있다.

4) 권한 상승

JWT 기반 인증 환경에서는 구현 방식에 따라 사용자 권한 정보가 페이로드의 클레임으로 포함될 수 있다. 예를 들어 일반 사용자와 관리자 계정을 구분하기 위해 role, isAdmin, authLevel 등의 클레임이 사용될 수 있으며, 서비스는 이후 요청에서 해당 값을 기준으로 접근 가능한 기능을 판단한다.

이처럼 권한 정보가 JWT 에 포함되는 구조에서는, 서명키가 유출될 경우 공격자가 기존 토큰의 페이로드를 수정해 권한 관련 클레임 값을 임의로 변경한 뒤, 유효한 서명으로 다시 JWT 를 생성할 수 있다. 예를 들어 role 값을 "ADMIN"으로 설정하거나 권한 레벨 값을 상향 조정한 토큰을 만들어 관리자 전용 API 를 호출하면, 관리자 페이지 접근, 사용자 계정 관리, 설정 변경, 로그 조회 등 권한 기능이 악용될 수 있다.

이러한 권한 상승은 단일 계정 침해를 넘어 서비스 운영 전반에 영향을 미칠 수 있다는 점에서 위험도가 크다. 특히 관리자 권한 범위가 넓을수록 데이터 변조, 대량 정보 유출, 서비스 장애 유발 등 2 차 피해로 확대될 수 있다.

■ 대응방안

이처럼 JWT 기반 인증 환경에서는 서명키 하나에 인증의 신뢰가 집중되며, 해당 키가 유출될 경우 토큰 형식이나 알고리즘과 무관하게 인증 체계 전반이 영향을 받는다. 이처럼 서명키에 대한 의존도가 높은 구조에서는, 서명키를 파일이나 환경변수, 문서 등에 평문으로 보관하거나 특정 관리자만 알고 있는 방식으로 운영할 경우 키 유출 위험을 완전히 배제하기 어렵다.

따라서 JWT 보안의 방향성은 서명키가 사람이나 애플리케이션 실행 환경에 노출되지 않도록 분리하여 보호하고, 침해 상황에서도 인증 피해를 최소화할 수 있는 구조를 마련하는 데 있다.

이를 위해 서명키 관리 관점에서 서명키 분리, 서명 권한 통제, 유출 가정 피해 제한 순으로 대응 방안을 정리한다.

1) 서명키 분리

본 챕터에서는 서명키 관리의 첫 단계로, 서명·검증 분리(대칭키/비대칭키)와 서명키 격리 운용을 통해 서명키가 사용되는 범위를 최소화하는 방안을 제시한다.

- **서명과 검증 분리**

JWT 기반 인증에서는 서버가 요청을 받을 때, 전달된 토큰이 정상적으로 발급된 것인지를 서명 검증 결과 하나로 판단한다. 즉, 서버는 "이 토큰이 어떤 키로 서명되었는가"만을 기준으로 인증 여부를 결정하며, 이 과정에서 사용되는 서명키는 토큰의 신뢰성을 결정하는 핵심 요소가 된다.

구분	서명 알고리즘	토큰 생성	토큰 검증
대칭키	HMAC-SHA256(HS256)	비밀키(Secret Key)	
비대칭키	RSA-SHA256(RS256)	개인키(Private Key)	공개키(Public Key)

표 2. 키 방식별 대표 JWT 서명 알고리즘

대칭키 기반 방식인 HMAC-SHA256(HS256)은 하나의 비밀키로 토큰 생성과 검증을 모두 수행한다. 구현이 단순하고 처리 속도가 빠르기 때문에, 단일 서비스나 비교적 단순한 인증 구조에서는 HS256 방식이 널리 사용되어 왔다. 그러나 인증 요청을 처리하는 모든 서버가 동일한 비밀키를 보유해야 하므로, 서비스가 확장될수록 키 배포 범위가 넓어지고 노출면이 커진다.

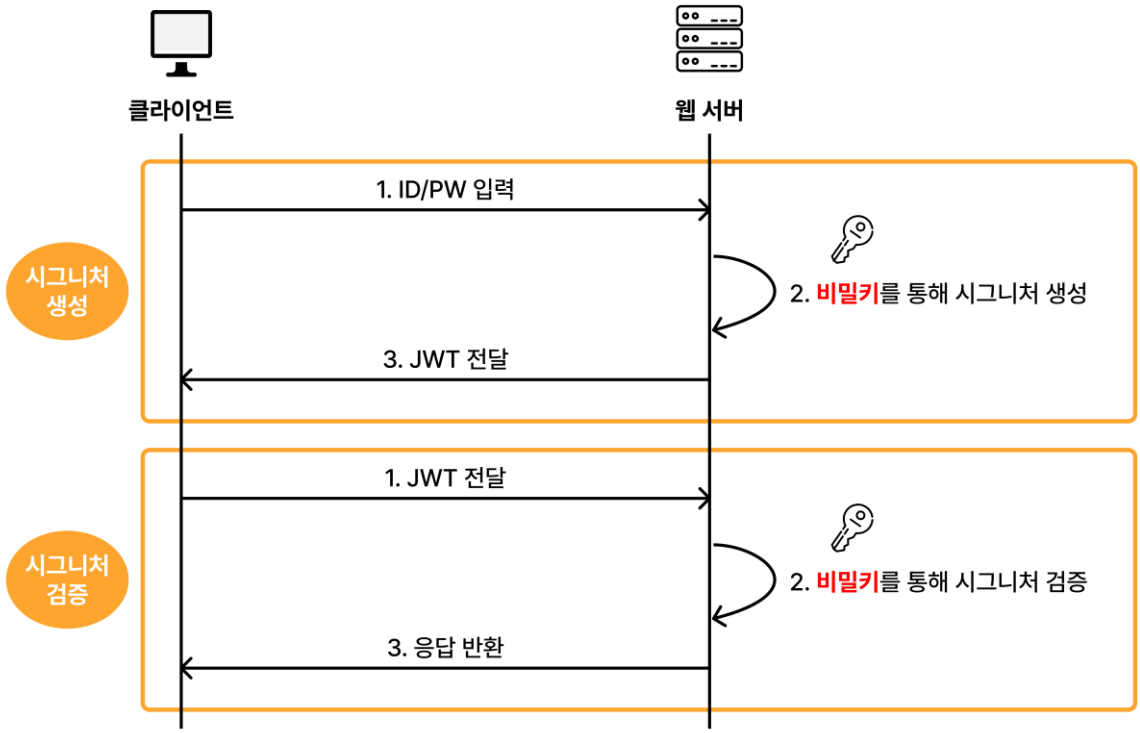


그림 22. 대칭키 기반 시그니처 생성 및 검증 과정

이 구조에서는 토큰을 검증하는 웹/API 서버가 비밀키를 직접 보유해야 하므로, 사용자 요청이 도달하는 영역 자체가 서명키 노출 지점이 된다. 따라서 서버가 침해되거나 키가 노출될 경우 공격자는 동일한 비밀키로 유효한 서명을 생성해 JWT 를 위조할 수 있다.

비대칭키 기반 방식인 RSA-SHA256(RS256)은 이러한 구조적 한계를 보완하기 위해, 토큰 생성에 사용되는 개인키와 검증에 사용되는 공개키를 분리한다. 개인키는 제한된 환경에서만 관리되며, 각 서비스 서버에는 토큰 검증에 필요한 공개키만 배포된다. 이로 인해 인증 요청을 처리하는 서버 수가 증가하더라도, 토큰을 생성할 수 있는 권한은 개인키를 보유한 서버로 제한된다.

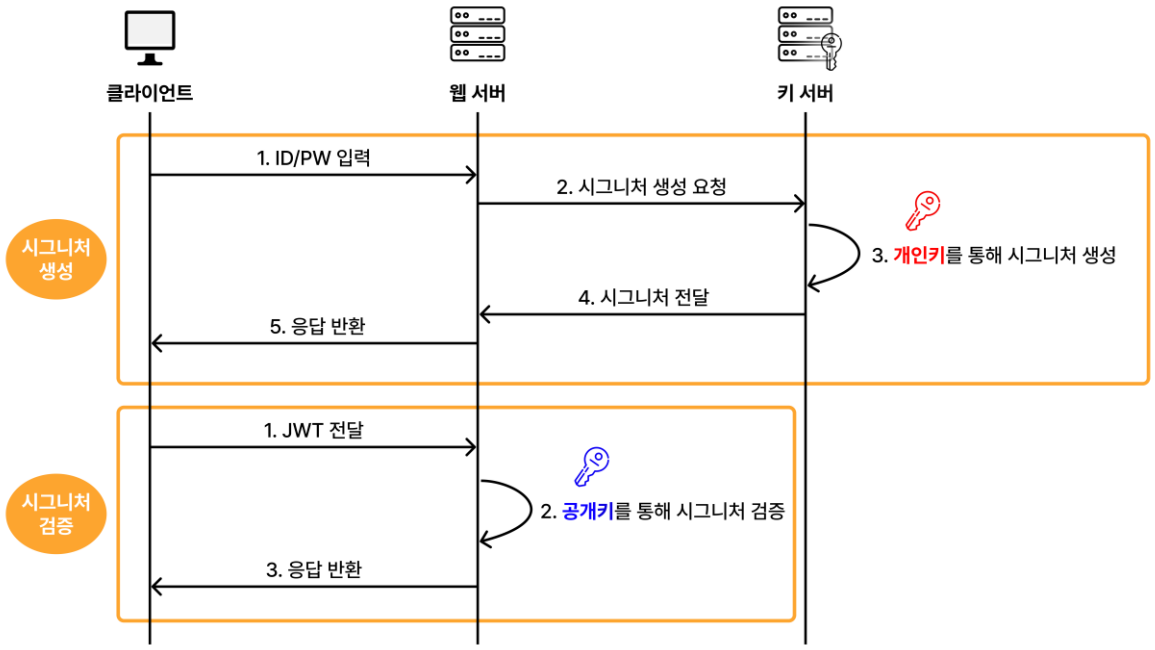


그림 23. 비대칭키 기반 시그니처 생성 및 검증 과정

이러한 구조에서는 검증 서버에 배포된 공개키가 노출되더라도 새로운 JWT 를 생성할 수 없다. 공개키는 토큰의 유효성을 판단하는 검증 용도로만 사용되며, 해당 키 자체로는 시그니처를 만들 수 없기 때문이다. 결과적으로 서명키가 포함된 환경과 검증 환경이 분리되어, 검증 서버의 키 노출이 토큰 위조로 이어지지 않는다.

또한 시그니처 검증은 서비스 규모와 관계없이 성능과 보안에 직접적인 영향을 미친다. 대칭키 기반 구조는 토큰 생성과 검증이 동일한 서버에 집중되어 트래픽 증가 시 성능 저하가 발생할 수 있으며, 침해 시 인증 위험이 확산된다. 반면 비대칭키 기반 구조는 시그니처 생성과 검증을 분리하여 서버 부하를 분산하고 생성 권한을 제한할 수 있다. 이러한 이유로 JWT 기반 인증에서는 비대칭키 기반 서명 방식이 구조적으로 더 안전한 선택으로 권장된다.

앞서 설명한 서명·검증 분리 구조는 서명키를 최소 범위에서만 사용하도록 제한하는 것을 목표로 하며, 이후 설명하는 HSM 과 같은 보안 장치는 이러한 구조에서 서명키를 보호하기 위한 수단으로 활용된다.

- **서명키 격리 운용**

JWT 환경에서 서명키 보호의 핵심은 키가 사람과 애플리케이션 실행 영역에 얼마나 노출되는지에 달려 있다. 서명키가 환경 변수, 설정 파일, 배포 산출물, 운영 문서 등에 포함되면 내부자 오남용이나 계정 탈취, 퇴사자 리스크 같은 현실적인 변수를 통해 키 유출 가능성이 생긴다. 따라서 서명키는 사람이 확인하거나 복사할 수 있는 값으로 취급하지 않고, 필요한 경우에만 제한된 방식으로 사용되도록 격리하는 운영 구조가 필요하다.

이를 위해 서명키 운영은 다음과 같은 방향으로 설계되어야 한다.

설계 방향	설명
중앙 관리 및 사람 개입 최소화	서명키를 코드/서버 설정에 포함하지 않고 중앙에서 관리하며, 배포·운영 과정에서 사람이 키 값을 직접 취급하는 절차를 최소화
키 미보관	애플리케이션이 키 값을 보관하지 않고 서명 대상 데이터만 전달하여 서명 연산을 요청하고 결과(시그니처)만 수신
사용 경로 제한 및 추적	서명키를 사용할 수 있는 주체와 경로를 제한해 노출 접점을 줄이고, 오·남용 탐지를 위해 서명 요청 이력을 남김

표 3. 서명키 노출 최소화를 위한 운영 원칙

이와 같은 키 노출 최소화 원칙을 기술적으로 쉽게 구현할 수 있도록 돕는 수단이 HSM(Hardware Security Module)이다. HSM 은 JWT 시그니처 생성 및 검증에 사용되는 중요한 키를 사람이나 애플리케이션이 직접 확인할 수 없도록 보호하기 위해 설계된 하드웨어 기반 보안 장치이다. 키는 HSM 내부에서만 생성·저장·사용되며, 외부 시스템은 키의 실제 값을 알지 못한 채 시그니처 생성과 같은 서명 연산을 요청하고 그 결과만을 전달받는다.

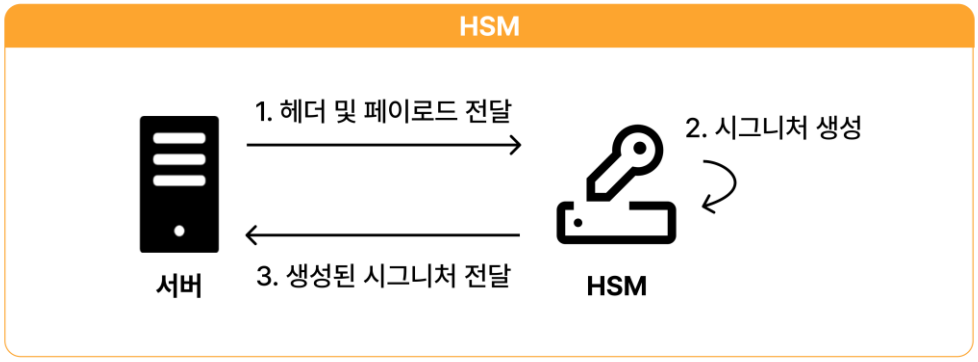


그림 24. HSM 사용 시 시그니처 생성 흐름

HSM 은 기술적으로 대칭키 방식에서 토큰 생성과 검증 모두에 사용될 수 있다. 그러나 모든 인증 요청에 HSM 을 연동하는 구조는 성능과 확장성 측면에서 운영 부담이 있다. 비대칭키 방식은 서명과 검증을 분리할 수 있어, HSM 을 서명키(개인키) 보호와 시그니처 생성에만 활용할 수 있다. 따라서 JWT 기반 인증 환경에서는 비대칭키 방식을 전제로, HSM 을 서명 전용으로 활용하는 방식을 권장한다.

HSM 은 목적에 따라 내장형, 네트워크형, 클라우드 기반 서비스 등으로 구분된다. 서버 간 인증이나 대규모 서비스 환경에서는 중앙에서 서명 연산을 처리할 수 있는 네트워크형 또는 클라우드 서비스 형태의 HSM 이 주로 사용된다. 반면 USB 형태의 HSM 은 주로 공인 인증서와 같은 개인 인증 용도로 활용되며, JWT 서명키 보호 목적에는 일반적으로 적합하지 않다.

구분	설명
내장형 HSM	서버 내부 슬롯(PCIe 등)에 직접 장착하는 카드 형태의 장치
네트워크형 HSM	독립된 전용 장비 형태로 네트워크를 통해 암호 연산 요청을 처리
USB형 HSM	휴대 가능한 USB 형태로 주로 개인 인증 및 서명 용도로 사용
칩형 HSM	반도체 칩 내부에 암호 연산 기능을 구현 (IoT 기기 등에 내장)
클라우드 HSM 서비스	클라우드 사업자(AWS, Azure 등)가 가상화된 환경에서 제공하는 HSM 서비스

표 4. HSM 종류

하지만 서명키 자체를 보호하더라도, 어떤 시스템이 서명 연산을 요청할 수 있는지에 대한 통제가 이루어지지 않으면 서명키가 유출된 것과 유사한 수준의 피해로 이어질 수 있다. 이러한 한계를 보완하기 위해, 다음에서는 서명 권한을 통제·추적하는 관리 체계를 중심으로 대응 방안을 살펴본다.

2) 서명 권한 통제

다음으로 서명 연산 요청 권한을 정책으로 제한하고 이력을 추적하는 통제 방안을 제시한다.

- **KMS 기반 서명 권한 관리**

HSM 을 통해 서명키의 외부 노출을 구조적으로 차단하더라도, 서명 연산을 요청할 수 있는 권한이 적절히 통제되지 않으면 보안 사고로 이어질 수 있다. 즉, 키 자체를 보호하는 것뿐 아니라, 누가 언제 어떤 목적으로 서명할 수 있는지에 대한 관리 역시 중요하다. 이러한 역할을 담당하는 것이 KMS(Key Management System)이다.

KMS 는 여러 애플리케이션과 서비스에서 사용하는 암호키를 중앙에서 관리하며, 키 생성부터 저장, 사용, 로테이션, 폐기까지의 전 과정을 정책 기반으로 통제한다. 이를 통해 관리자는 어떤 시스템이 어떤 키로 언제 서명 연산을 요청했는지를 확인할 수 있고, 서명 연산(키 사용) 권한을 서비스 또는 역할 단위로 세밀하게 제한할 수 있다. 또한 감사 로그와 모니터링 기능을 통해 비정상적인 서명 요청을 탐지하고 대응할 수 있다.

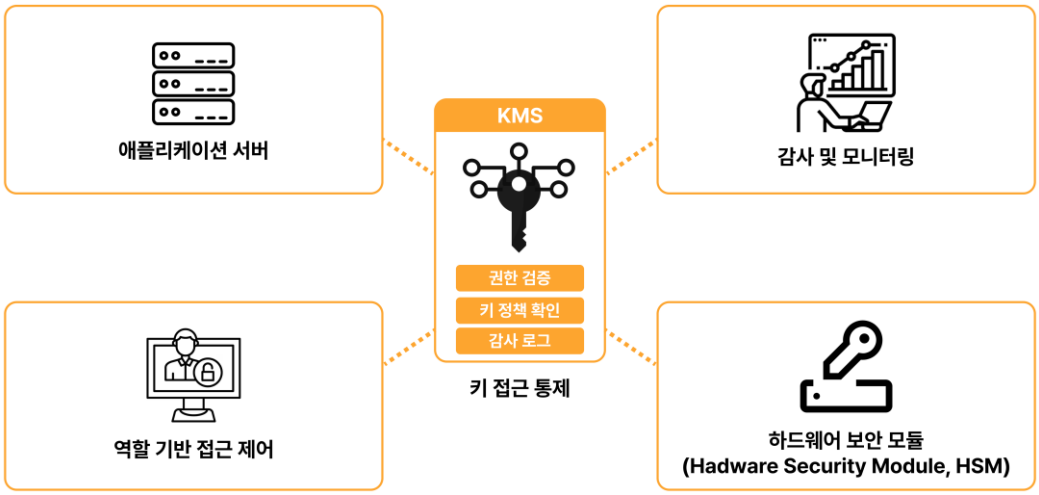


그림 25. KMS 기반 중앙 키 관리 아키텍처

주요 클라우드 서비스에서는 KMS 를 서비스 형태로 제공한다. 클라우드 KMS 는 IAM 기반 접근 제어, 자동 로테이션, 감사 로그 연동 등을 기본적으로 지원하며, 필요에 따라 클라우드 HSM 과 연계해 서명키를 하드웨어 수준에서 보호할 수 있다. 이 경우 애플리케이션은 KMS 를 통해 서명 요청을 수행하며, 실제 서명키는 클라우드 제공자의 보호된 환경 내에 유지된다.

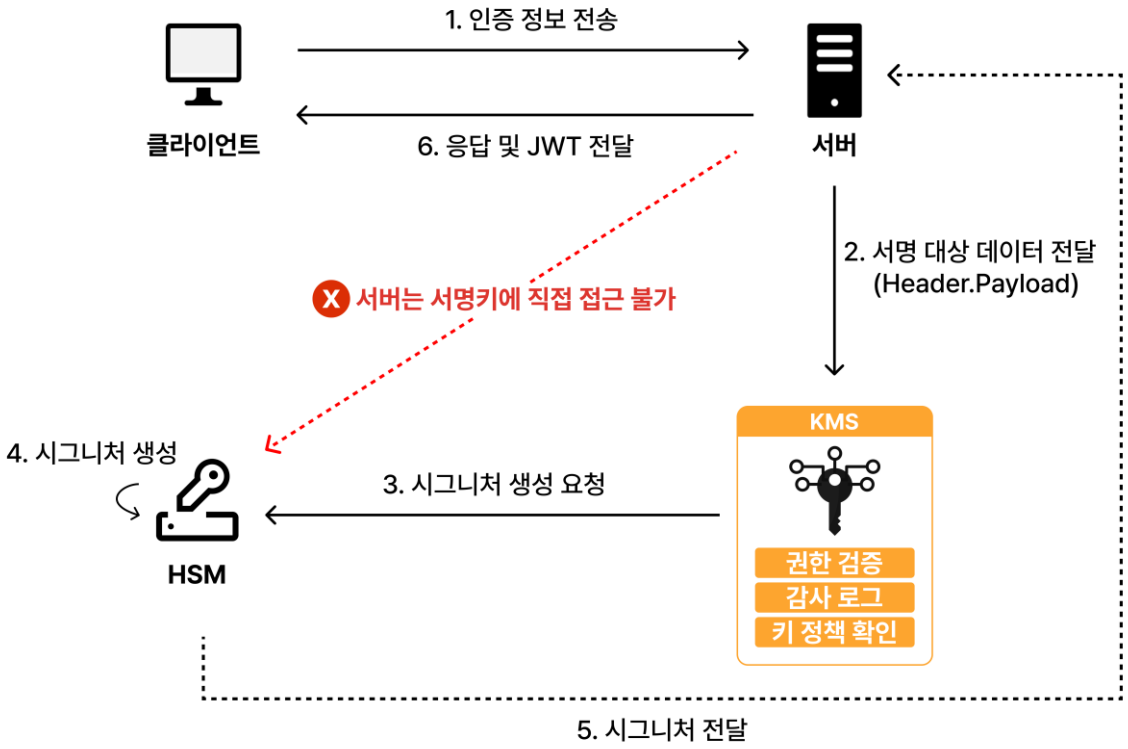


그림 26. KMS 와 HSM 을 통한 JWT 시그니처 생성 흐름

반면 금융권, 공공기관, 폐쇄망 환경과 같이 외부 클라우드 서비스 사용이 제한되거나 자체 보안 가이드라인을 따라야 하는 조직에서는, 동일한 개념을 온프레미스 환경에서 직접 구현해야 한다. 이 경우에도 대응의 핵심은 서명키를 중앙에서 관리하고, 사용 권한과 이력을 통제하는 구조를 갖추는 것이다.

구현 방식은 조직의 인프라와 보안 요구 수준에 따라 달라질 수 있지만, 서명 요청 권한의 분리, 사용 이력 기록과 감사, 주기적인 키 로테이션은 온프레미스 환경에서도 공통적으로 고려되어야 할 핵심 요소이다. 특히 키 로테이션은 동일 키의 장기 사용을 피함으로써 키 노출 시 영향 기간을 제한하고, 사고 대응 시 키 교체를 통해 피해 확산을 신속히 차단할 수 있게 한다.

이러한 구조를 기반으로 HSM 과 KMS 를 함께 적용하면 서명키 유출로 인한 위험을 효과적으로 완화할 수 있다. 그러나 보안 사고 가능성을 완전히 배제할 수는 없으므로, 다음으로는 이를 전제로 인증 피해의 확산을 최소화하기 위한 설계적 보완 방안을 살펴본다.

3) 서명키 유출 대비 피해 제한

마지막으로 서명키 유출 가능성을 전제로, sub 클레임 설계와 중요 기능 추가 검증을 통해 피해 확산 범위를 제한하는 완화 방안을 제시한다.

- **sub 클레임 설계**

앞선 대응 방안들은 서명키를 보호하고 관리하는 데 초점을 맞추고 있지만, 현실적으로 키 유출 가능성을 완전히 배제하는 것은 어렵다. 따라서 JWT 보안에서는 키 보호와 더불어, 서명키가 유출되었을 때 피해가 어디까지 확산될 수 있는지를 제한하는 설계적 보완 역시 함께 고려되어야 한다.

JWT 의 sub 클레임은 토큰이 어떤 사용자를 나타내는지를 식별하기 위한 값으로, 서버는 시그니처 검증이 완료된 토큰에 대해 sub 값을 기준으로 사용자 계정을 식별한다. 이때 sub 값이 내부 사용자 ID 와 같이 순차적이거나 의미를 가지는 정수값으로 구성되어 있을 경우, 서명키 유출 시 구조적인 위험이 발생할 수 있다. 공격자는 유효한 서명키로 토큰을 생성할 수 있는 상황에서 sub 값을 변경하며 다른 사용자로 가장한 JWT 를 반복적으로 만들어낼 수 있기 때문이다.

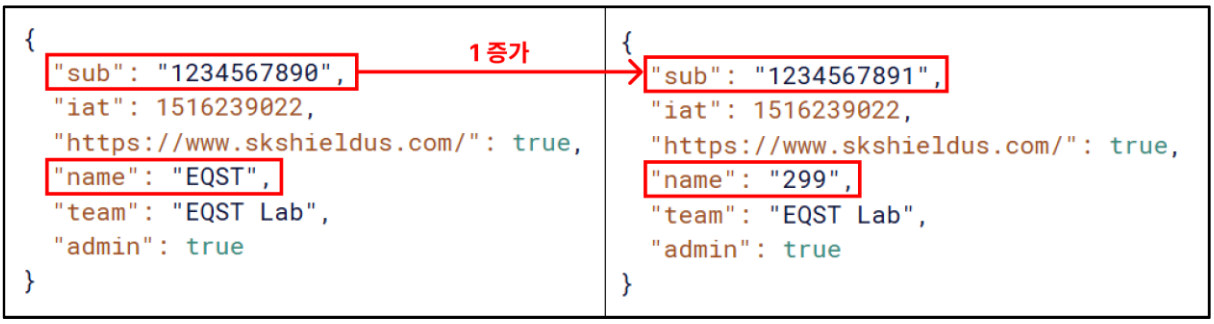


그림 27. sub 클레임 예시

이러한 구조에서는 개별 계정 침해를 넘어, 자동화된 방식으로 다수 계정을 순차적으로 시도하는 공격으로 확산될 가능성이 높아진다. 즉, sub 값의 설계 방식이 서명키 유출 시 피해 범위를 결정짓는 요소로 작용하게 된다.

이를 완화하기 위해 sub 클레임에는 외부에 노출되더라도 사용자 정보를 직접적으로 추정하기 어려운 비의미적 식별자를 사용하는 것이 바람직하다. UUID 와 같은 값은 사용자 식별 기능은 유지하면서도, 값의 연속성이나 범위를 기반으로 한 추측을 어렵게 만들어 자동화된 계정 전환 공격의 난이도를 높인다.

비의미적 식별자 적용은 서명키 유출을 근본적으로 차단하는 대응책은 아니지만, 침해 발생 시 공격자의 행위를 제한하고 피해 확산 속도를 늦추는 현실적인 보완 전략이다. JWT 표준에서 sub 클레임은 토큰 주체를 식별하기 위한 등록 클레임으로 정의되어 있으므로, 해당 역할을 전제로 외부 노출을 가정한 안전한 값 설계가 필요하다.

• **중요 기능 추가 검증**

서명키 보호와 sub 클레임 설계를 통해 토큰 위조와 계정 확산 위험을 완화하더라도, JWT 기반 인증 구조에서는 유효한 토큰을 보유한 요청을 정상 인증으로 처리한다는 특성 자체를 완전히 제거할 수는 없다. 따라서 키 유출이나 토큰 탈취가 발생한 상황을 전제로, 토큰 하나로 모든 기능 접근을 허용하지 않는 구조적 보완이 필요하다.

실제 서비스 환경에서는 개인정보 조회, 계정 정보 변경, 결제·환불과 같은 민감한 기능에 접근할 때, 로그인 상태와 별도로 비밀번호 재입력과 같은 추가 검증 절차를 요구하는 경우가 많다. 이는 인증 토큰의 신뢰 범위를 제한함으로써 단일 토큰 탈취로 인한 피해를 줄이기 위한 설계이다.

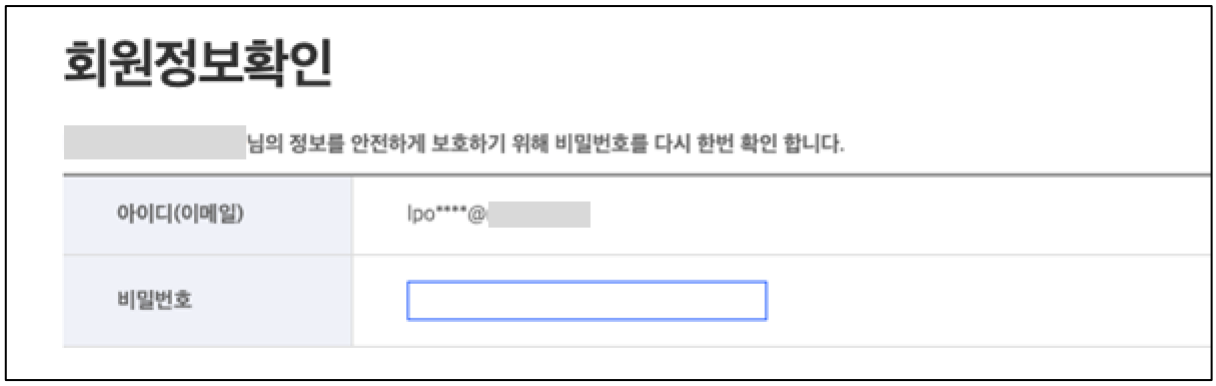


그림 28. 개인정보 페이지 접근 시

이와 같은 추가 검증은 JWT 의 서명 검증 이후에 수행되며, 토큰이 유효하더라도 특정 행위에 대해서는 사용자 재확인 또는 별도의 인증 단계를 거치도록 구성 가능하다. 이를 통해 공격자가 유효한 JWT 를 확보하더라도, 민감한 API 나 핵심 기능에 대한 직접적인 악용을 어렵게 만들 수 있다.

결과적으로 JWT 기반 인증 환경에서는 모든 기능을 동일한 신뢰 수준으로 처리하기보다, 기능의 중요도에 따라 인증 강도를 차등 적용하는 구조가 현실적인 보완 전략이 될 수 있다. 이는 서명키 유출이라는 최악의 상황에서도 서비스 전반의 피해를 제한하기 위한 추가적인 방어 계층으로 작용한다.

■ 마무리

JWT 환경에서 서명키가 유출되면 공격자는 정상 사용자와 구분하기 어려운 토큰을 생성해 사용할 수 있으며, 탐지가 지연될 경우 피해가 빠르게 확대될 수 있다. 따라서 JWT 보안의 핵심은 서명 알고리즘 선택 이전에 서명키를 어떻게 보호하고, 통제하는가에 있다.

특히 “관리자나 내부 인력은 알고 있어도 된다”는 운영 방식은 다양한 운영·접근 경로를 통해 키가 노출될 여지를 만든다. 따라서 서명키는 사람이 직접 보관·공유하는 대상이 아니라, 격리된 환경에서 필요 시에만 제한된 권한으로 사용되도록 설계되어야 한다. 이러한 원칙이 지켜질 때, 침해 상황에서도 인증 체계 전반으로 피해가 확산되는 위험을 실질적으로 줄일 수 있다.

■ 참고 사이트

• Microsoft Security (Storm-0558): https://www.microsoft.com/en-us/security/blog/2023/07/14/analysis-of-storm-0558-techniques-for-unauthorized-email-access/
• Microsoft (Golden SAML): https://www.microsoft.com/en-us/msrc/blog/2020/12/customer-guidance-on-recent-nation-state-cyber-attacks
• MITRE ATT&CK (Golden SAML): https://attack.mitre.org/techniques/T1606/002/
• RFC7519: https://datatracker.ietf.org/doc/html/rfc7519
• JWT
-https://www.jwt.io/introduction#what-is-json-web-token-structure
-https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html
-https://learn.microsoft.com/en-us/entra/identity-platform/id-token-claims-reference
• AWS KMS: https://docs.aws.amazon.com/kms/latest/developerguide/overview.html
• AWS CloudHSM: https://docs.aws.amazon.com/cloudhsm/latest/userguide/introduction.html
• Azure key-vault: https://learn.microsoft.com/en-us/azure/key-vault/general/basic-concepts
• Coupang:
-https://pages.coupang.com/f/160047
-https://news.coupang.com/archives/58857/
-https://v.daum.net/v/20251202212010142