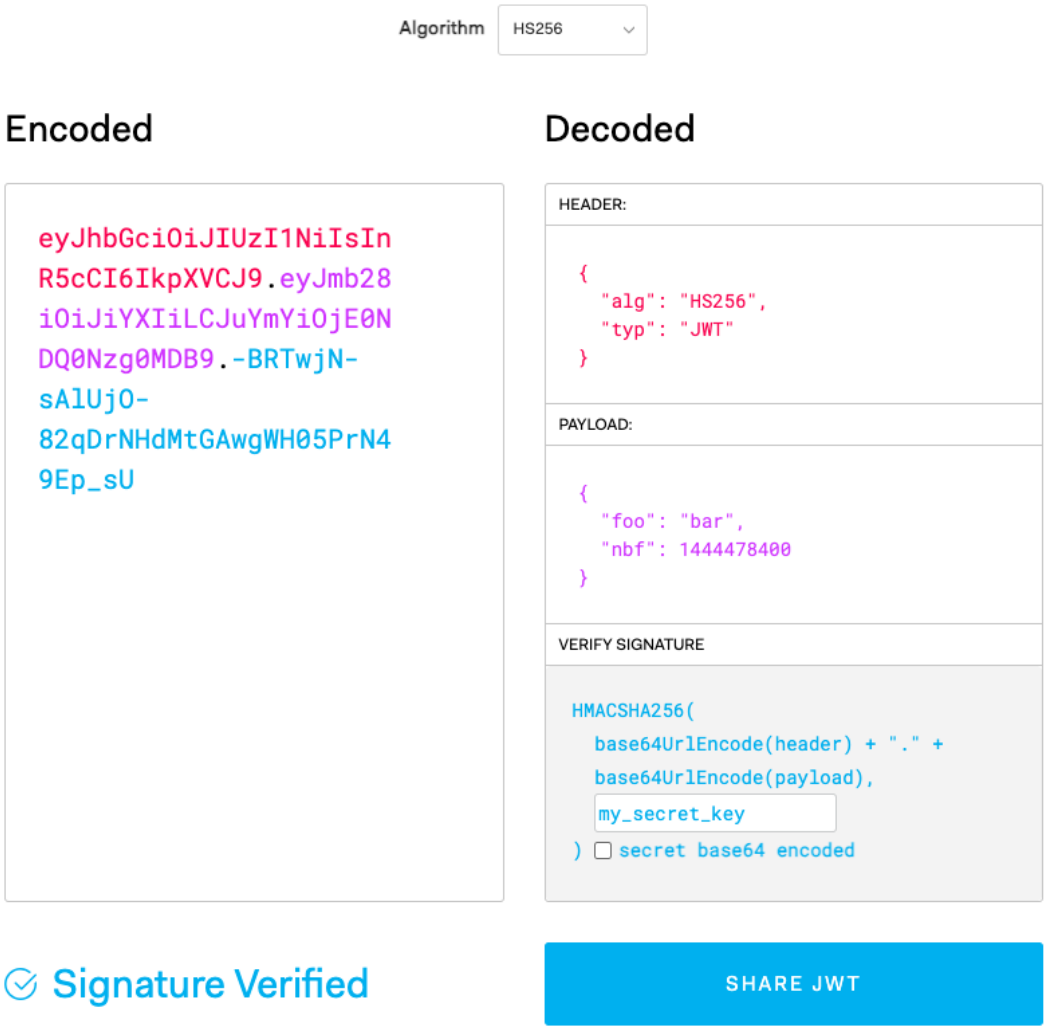
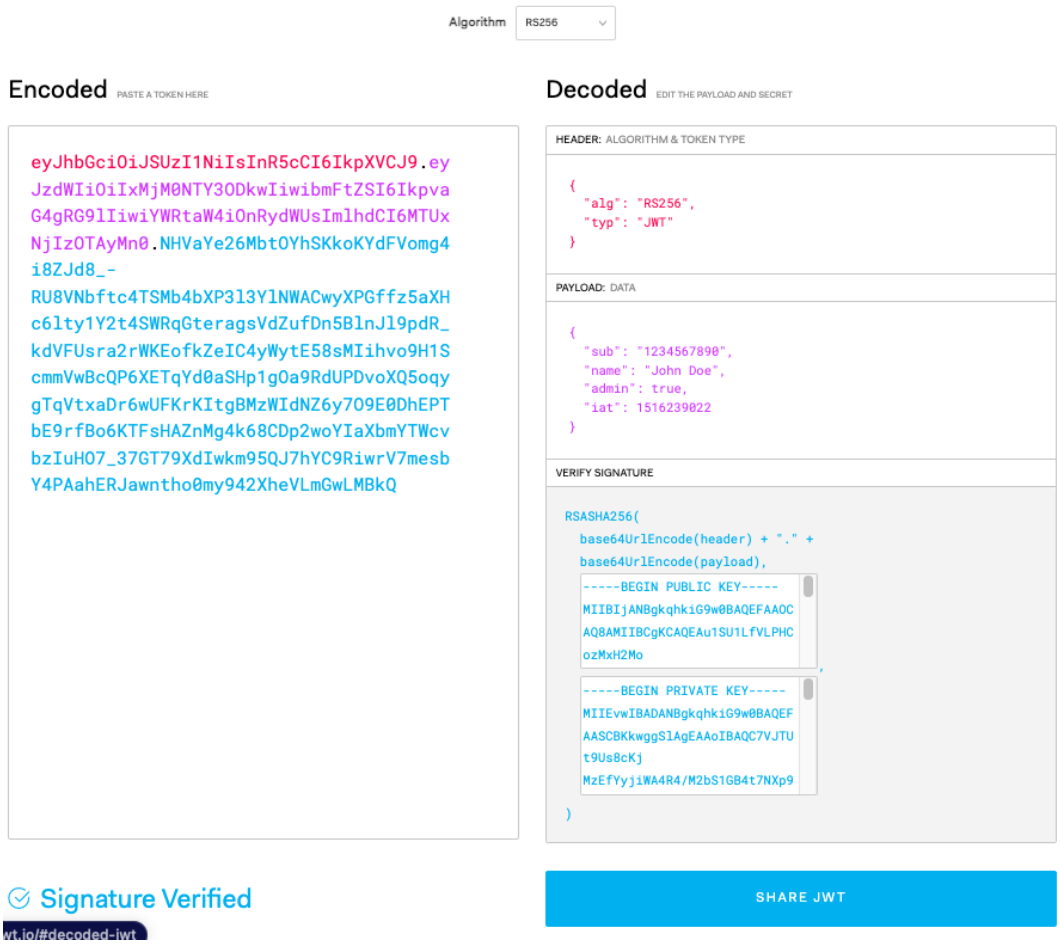
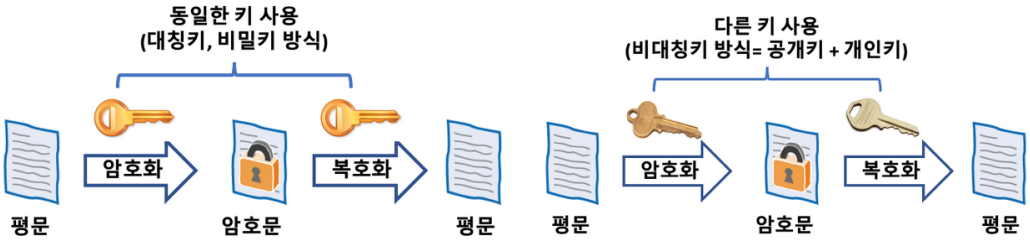
[CS 지식] JWT

- JSON Web Tokens - jwt.io
- ## 왜 관심을 가지게 되었나?
  - 모바일 게임의 경우 간단한 게임은 웹 서버를 통해 데이터를 주고 받게 되는데, 관련 사용자의 정보를 어떻게 판단 할지에 대해 리서치 진행을 하다, 일반적으로 많이 사용하는 JWT에 대해 알게 됐고, 관련 기술을 사용하는게 적합하다고 판단이 들었기 때문에 알아보게 됨.
- ## JWT (JSON Web Tokens)
  - **공식 사이트 내용 발췌**
    - https://jwt.io/introduction 자세한 내용은 해당 사이트 참고
    - **JWT란?**
      - JSON 웹 토큰(JWT)은 당사자 간에 정보를 JSON 객체로 안전하게 전송하기 위한 간결하고 독립적인 방법을 정의하는 개방형 표준(RFC 7519)입니다. 이 정보는 디지털 서명이 되어 있기 때문에 검증되고 신뢰할 수 있습니다. JWT는 비밀(HMAC 알고리즘 사용) 또는 RSA 또는 ECDSA를 사용하는 공개/개인 키 쌍을 사용하여 서명할 수 있습니다.
      - JWT를 암호화하여 당사자 간의 비밀을 유지할 수도 있지만, 여기서는 서명된 토큰에 초점을 맞추겠습니다. 서명된 토큰은 토큰에 포함된 클레임의 무결성을 확인할 수 있는 반면, 암호화된 토큰은 다른 당사자로부터 이러한 클레임을 숨깁니다. 공개/개인 키 쌍을 사용하여 토큰에 서명하는 경우, 서명은 개인 키를 보유한 당사자만이 토큰에 서명했음을 증명하기도 합니다.
    - **JSON 웹 토큰은 언제 사용해야 하나요?**
      - 다음은 JSON 웹 토큰이 유용한 몇 가지 시나리오입니다:
        - **인증:** 이것은 JWT를 사용하는 가장 일반적인 시나리오입니다. 사용자가 로그인하면 이후의 각 요청에 JWT가 포함되어 사용자가 해당 토큰으로 허용된 경로, 서비스 및 리소스에 액세스할 수 있습니다. 통합 인증은 오버헤드가 적고 여러 도메인에서 쉽게 사용할 수 있기 때문에 오늘날 널리 사용되는 기능입니다.
        - **정보 교환:** JSON 웹 토큰은 당사자 간에 정보를 안전하게 전송하는 좋은 방법입니다. 예를 들어 공개/개인 키 쌍을 사용하여 JWT에 서명할 수 있으므로 발신자가 자신이 말한 사람인지 확인할 수 있습니다. 또한 서명은 헤더와 페이로드를 사용하여 계산되므로 콘텐츠가 변조되지 않았는지 확인할 수도 있습니다.
  - **요약**
    - JWT는 JSON Web Token의 약자로, 정보를 안전하게 전달하기 위한 암호화된 토큰입니다.
    - 간단히 말해, 유용한 작업(예: 인증)을 수행하는 서명된 JSON 객체입니다. 일반적으로 Oauth2의 무기명 토큰에 사용됩니다.
      - 무기명 토큰: 특정 유저나 클라이언트를 식별하는데 사용 되는 토큰 이지만, 해당 사용자나 클라이언트의 식별 정보가 없이 발급된 토큰
        - e.g. 게임에서 유저는 로그인할 때 서버로부터 무기명 토큰을 받을 수 있습니다. 이 토큰은 특정 게임 클라이언트를 식별하긴 하지만, 그 유저의 이름, 이메일 주소, 또는 다른 개인 정보를 포함하지 않습니다.
  - **간단한 설명**

- 

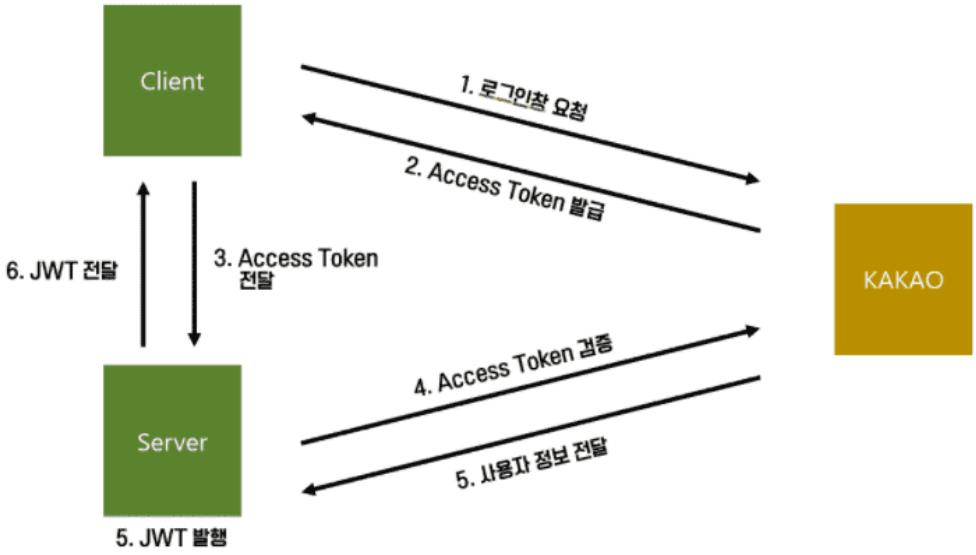
  - Header: 암호화 방식 기재.
  - Payload: 실제 전송하려는 데이터 전송. base64 인코딩된 값으로 별도 암호화 되지 않아 중요한 정보는 노출하면 안됨
    - e.g: uuid나 id, userkey 노출 가능 (해당 값을 통해 유저 검색 하기 위함). 비밀번호. 개인 정보 노출 x
  - Signature: 서명. 암호화된 값으로 해당 값으로 정상적인 값인지에 대한 유효성 검증 가능.
- **친절한 디버거 사이트**
  - 

- 

  - 공식 사이트에서 jwt의 설명 및 디버거등을 제공 한다.
  - RSA의 public, private 키는 RSA 키 온라인 생성기를 통해 생성함
- **JWT 알고리즘: HMAC (대칭) vs RSA (비대칭)**
  - https://en.wikipedia.org/wiki/HMAC
  - https://en.wikipedia.org/wiki/RSA_(cryptosystem)
  - 

구현 방법에 대한 고민

- **구글 로그인 OAuth 구현 시 토큰에 대한 검증**
  - 구글 로그인 이후 토큰을 발급 받게 되는데 일반적으로 id_token으로 검증을 하게 됨.
  - id_token의 경우 jwt 형태로 제공하고 있고, RSA 방식의 알고리즘을 지원하고 있어 공개키로 해당 값의 유효성을 검증할 수 있다.
    - 서버(예: 구글)를 통해 검증(엔드포인트 방식)은 서버 부하로 권장되지 않는 것으로 보인다.
    - 구글이 private key 가지고 있고.. 나는 public key로 해당 값 검증.
- **소셜 로그인 구현 방법 예**



- - 소셜 로그인 모두 종료 이후에는 서버의 jwt 영역이 됨. 액세스 토큰, 리프레시 토큰 등.
- **Question?**
  - 액세스 토큰을 항상 전달하고 있는데, 해당 액세스 토큰을 검증하는 방법은?
  - 소셜 로그인 이후 해당 값을 그대로 사용하지 않고, (구글에서 발급된 액세스 토큰 등) 별도의 jwt 토큰을 사용하는게 옳은 방법인가?
    - 구글 로그인을 한 이후에 구글의 서비스를 사용하지 않는다면.. 굳이 구글의 액세스 토큰을 사용할 일은 없어 보임.
- **간단한 JWT 토큰 검증 예**
  -

// TokenAuthMiddleware 토큰의 유효성 검사를 한다. func TokenAuthMiddleware(c *gin.Context) { h := token.MyCustomHeader{}

```
if err := c.ShouldBindHeader(&h); err != nil {
    c.JSON(200, err)
    c.AbortWithError(401, fmt.Errorf("header bind error"))
    return
}

tokenString := h.Authorization
if len(tokenString) == 0 {
    c.JSON(500, gin.H{
        "msg": "not token",
    })
    c.AbortWithError(401, fmt.Errorf("not token error"))
    return
}

tk, err := token.ParseTokenString(tokenString)
if valErr, ok := err.(*jwt.ValidationError); ok {
    if valErr.Errors&jwt.ValidationErrorExpired != 0 {
        c.JSON(400, gin.H{
```

```
            "status": 10,
            "reason": "token expired",
            "error": valErr.Error(),
        })
        c.Abort()
        return
    }
}

if err != nil || tk == nil || !tk.Valid {
    c.JSON(400, gin.H{"error": err.Error()})
    c.Abort()
    return
}

if claims, ok := tk.Claims.(*token.MyCustomClaims); ok {
    fmt.Println("claims", claims.UUID, claims.StandardClaims.ExpiresAt)
} else {
    c.Abort()
    return
}

// Pass on to the next-in-chain
c.Next()
}
```

- ```go
// 실행 코드
func main() {
    r := gin.Default()

    v1 := r.Group("/v1")
    {
        v1.POST("/zone", api.GetZone)
        v1.POST("/join", api.Join)

        v1.Use(middleware.TokenAuthMiddleware)
        v1.POST("/login", api.Login)
    }

    r.Run() // listen and serve on 0.0.0.0:8080 (for windows "localhost:8080")
}
```
- 미들웨어 만들어두고, 로그인 이후 모든 검증에서 시행하려고 했던 방대한 꿈이 있었으나.. 사장 됨

## 결론

- 모바일 게임 웹 서버에서 인증을 구축하는 부분은 JWT가 일반적이다.
```

    - 간단한 모바일 게임에서도 인증 자체에 대해 상당히 많은 고민을 해야 한다.
    - 소규모로 제작하는 게임의 경우 직접 서버를 구축하는 것 보다는 서비스를 이용하는 편이 좋을 것 같다? (인증 및 인가 처리를 별도로 신경 쓸 필요가 없다.)
        - [[[CS 지식] BaaS & GBaaS]]
- ## 참고
    - golang-jwt Github