基于 SM9 的 JWT 强身份认证方案

罗娇燕,左黎明,陈艺琳,郝　恬
(华东交通大学 理学院,江西 南昌 330013)

摘　要:随着网络信息技术的快速发展,身份认证的应用范围也在不断扩大。其中,JWT(JSON Web Token)作为基于 Token 的身份认证技术,被广泛应用于 Web 应用程序和 API 领域,以实现简单、可靠的身份验证和安全通信。然而,开发人员对于 JWT 标准和技术细节理解不够深入,导致该技术在实践中经常出现各种安全漏洞。文中分析了近年来出现的有关 JWT 技术的安全问题,包括“none”算法绕过、敏感信息泄露、算法混淆攻击和密钥穷举攻击等,并针对这些问题提出了一种基于国密 SM9 的 JWT 强身份认证方案。该方案使用 SM9 公钥密码算法对 JWT 进行签名和验证,结合基于时间戳和随机数的验证机制,以提高算法的安全性和可靠性。最后对该方案进行安全性分析,结果表明该方案实现方法相对简单,能够有效地防御各种常见的 JWT 安全漏洞,同时具有良好的安全性和易用性,为 JWT 技术的安全应用提供了一种高效可靠的解决方法。

关键词:身份认证;JWT;SM9;数字签名;授权
中图分类号:TP309.7　　**文献标识码**:A　　**文章编号**:1673-629X(2024)03-0110-08
doi:10.3969/j.issn.1673-629X.2024.03.017

JWT Strong Identity Authentication Scheme Based on SM9

LUO Jiao-yan,ZUO Li-ming,CHEN Yi-lin,HAO Tian
(School of Science,East China Jiaotong University,Nanchang 330013,China)

Abstract:With the rapid development of network information technology,the application scope of identity authentication is continuously expanding. JSON Web Token (JWT),as a token-based identity authentication technology,has been widely used in web applications and API fields to achieve simple and reliable identity verification and secure communication. However,insufficient understanding of JWT standards and technical details among developers often leads to various security vulnerabilities in practice. We analyze security issues related to JWT technology that have emerged in recent years,including "none" algorithm bypass,sensitive information leakage,algorithm obfuscation attacks,and key enumeration attacks. To address these issues,a JWT strong authentication scheme based on China's national encryption standard SM9 is proposed. This scheme employs the SM9 public key cryptography algorithm for JWT signing and verification,and combines a verification mechanism based on timestamps and random numbers to enhance the security and reliability of the algorithm. A security analysis of the proposed scheme indicates that its implementation method is relatively simple,and it is effective in preventing various common JWT security vulnerabilities. The scheme also exhibits good security and usability,providing an efficient and reliable solution for secure application of JWT technology.

Key words:identity authentication;JSON Web Token;SM9;digital signature;authorization

0　引　言

随着互联网应用的普及和数据交换的增加,安全问题已成为互联网应用开发中不可忽视的挑战[1]。传统的基于 session-cookie 模式的有状态身份认证方式不仅面临着 CSRF 等安全问题,而且在可扩展性和跨域方面也存在缺陷[2]。针对上述问题,基于 Token 身份认证机制提出了一种无状态、轻量级、可扩展和支持跨域的身份认证方案。根据这些优点,Token 身份认证机制在 API 接口中得到了广泛应用,其中以 JWT 技术为代表[3]。

JWT 技术是由 RFC 7519[4]定义的一套开放标准,具有紧凑且安全的特点,可用于执行身份认证和信息交换。但由于 JWT 标准的复杂性以及开发者对技术细节和加密算法的理解不够透彻,为开发带来便利的同时也为应用程序的安全性埋下了隐患。2015 年 Tim McLean[5]发现,有些类库在实现 JWT 规范的过程

收稿日期:2023-04-09　　**修回日期**:2023-08-10
基金项目:江西省教育科技项目(GJJ200626,GJJ210625)
作者简介:罗娇燕(2001-),女,硕士研究生,研究方向为信息安全;通讯作者:左黎明(1981-),男,副教授,硕士,硕导,CCF 会员(E20-0013632M),研究方向为信息安全、非线性系统。

中会产生一些漏洞;例如利用“none”算法或者直接在交互过程中删除签名均可以绕过签名验证过程。2016 年,安全研究人员 Sjoerd Langkemper[6] 提出一种针对签名算法的攻击,将 RSA 和 HMAC 算法混淆使用来绕过服务器的身份认证。2021 年,Apache 发布的安全公告中,公开了 Apache ShenYu 中的身份验证绕过漏洞(编号:CVE-2021-37580);由于产品对客户端提交的 token 进行解析的同时没有进行校验,导致攻击者可绕过认证,直接进入系统后台。2022 年 Khaled Nassar[7] 在 github 上披露了 Oracle JavaSE 数字签名验证机制的漏洞利用代码(编号:CVE-2022-21449),该漏洞是由于开发人员在实现椭圆曲线数字签名算法(ECDSA)时未充分考虑签名验证机制所导致的,攻击者可以伪造签名从而绕过身份验证。同年,nodejs 的开源基础库 JsonWebToken 被曝存在远程代码执行漏洞(编号:CVE-2022-23529),攻击者可以通过该漏洞远程执行恶意代码,导致系统崩溃、数据泄露、账户被劫持等风险,从而给用户的隐私和财产带来巨大威胁。此外,由于该库每月在 NPM 上下载量超过 3 600 万次,应用于超过 22 000 个开源项目,其中包括微软、Twilio、Salesforce、Intuit、Box、IBM、Docusign、Slack、SAP 等知名公司创建的开源项目,这也意味着该漏洞的影响范围极其广泛。

JWT 中存在的安全问题,例如“none”算法绕过(漏洞编号:CVE-2018-1000531、CVE-2022-23540)、敏感信息泄露(漏洞编号:CVE-2019-7644、CVE-2022-23541)、密钥穷举攻击、算法混淆攻击(漏洞编号:CVE-2016-10555)等,主要是由于开发人员对于 JWT 技术理解程度的不够深入而引发的[8]。基于上述安全问题,该文提出了一种基于国密 SM9 的 JWT 强身份认证方案,利用国密 SM9 算法对认证参数进行数字签名,实现对用户身份的安全认证。

1　JWT 身份认证及其攻击

1.1　JWT 技术

消息保护技术在应用层中很少单独使用,通常融合在整个安全技术体系之中,而 JWT 就是代表性的消息保护技术[9]。JWT 可有效保证多方通信中的信息安全,在无需频繁调用资源服务器或者数据库的情况下,JWT 就可验证后续客户端请求,可适用于分布式站点的单点登录场景[10-11]。该文主要基于 JWT 的身份认证功能进行分析和优化。图 1 展示了基于 JWT 的身份认证流程[12]。首先,客户端使用登录凭证(比如用户名和密码)请求授权服务器;凭据验证通过后,授权服务器利用密钥生成一个 JWT 令牌返回客户端。然后,客户端携带此 JWT 令牌请求资源服务器上受保护的资源;资源服务器从 JWT 中解析用户请求并进行处理。

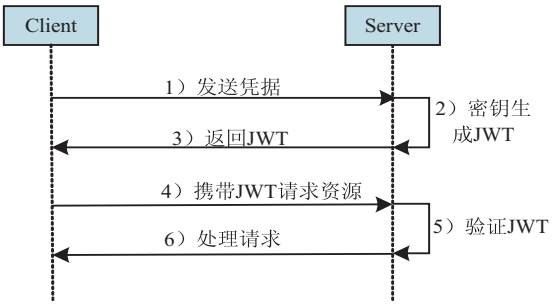

图 1　JWT 工作流程

JWT 是一个由三部分组成的字符串,分别是头部(Header)、载荷(Payload)和签名(Signature),点(“.”)号作为相邻部分的分隔符。其中头部和载荷部分本身是 JSON 格式的数据,可以利用 Base64url 算法对内容进行解码,解码后的内容如下所示。

```
Headers = {
  "typ": "JWT",
  "alg": "HS256"
}
Payload = {
  "iss": "Token Builder",
  "iat": 1680163910,
  "exp": 1680163970,
  "sub": "tom@ webgoat. org",
  "user name": "Tom",
  "Email": "tom@ webgoat. org",
  "admin": "true"
}
SIGNATURE =HMACSHA256{
base64UrlEncode(header) + "." + base64UrlEncode(payload),
密钥
}
```

头部定义令牌类型以及所采用的加密算法,HS256 代表该 JWT 令牌的签名算法为 HMACSHA256。载荷部分主要是对实体(通常是用户实体)和附加数据的声明,声明通常分为三类,分别是注册声明、公共声明和私有声明。图中的 iss(发布者)、iat(发布时间)、exp(到期时间)、sub(主题)等字段都是属于注册声明,并非强制性使用的;公共声明和私有声明开发者可根据实际交互情况进行设置。签名部分是利用算法对头部和载荷部分进行签名,防止数据被篡改[4];在头部声明的签名算法不同,签名的过程也是不同的。

1.2　针对基于 JWT 身份认证的攻击

JWT 是 API 技术中消息传递的重要形式,各个 API 服务提供方在 OAuth 2.0 和 OpenID Connect 使用它在各方之间交换信息[9]。尽管 JWT 的使用范围很

广,但常常因为提供方的实现不规范产生了一些安全问题。JWT 由三部分组成,该文针对各部分产生的漏洞进行分析,并对一些典型漏洞进行研究。

(1)针对头部的攻击。

“none”算法绕过。

JWT 的头部通过设置 typ 参数和 alg 参数分别标明了 Token 类型和签名算法。在 RFC 7518[13]标准中,定义了 JWT 可以使用一系列算法进行签名,也可以不使用算法进行签名,即将 alg 参数的值设置为 none。如图 2 所示,在 JWT 中设置“none”算法时,在签名部分将不使用任何算法进行签名,即签名部分为空字符串,后端服务器不执行签名验证,此时前两部分符合规范的任何 token 都是有效的。“none”算法最初的目的是为了方便开发者在开发环境中进行调试,如果在生产环境中也使用该功能,攻击者便可以伪造任意用户的 token 进行身份认证[5]。

```
eyJraWQiOiJhOTc3ZjhiNC1kMTdkLTQ4ZTAtYmY2MC0
yYjk0MTkyYjQxNjciLCJhbGciOiJSUzI1NiJ9.eyJpc3MiOi
Jwb3J0c3dpZ2dlcilslnN1Yil6IndpZW5lcilslmV4cCl6MT
Y3ODgwMjUzNH0.
```

图 2 基于“none”算法生成的 JWT

(2)针对载荷的攻击。

当 JWT 用于身份认证时,通常需要包含用户的相关信息,而载荷部分便是用于声明用户实体及其要发送的数据信息,是 JWT 不可缺少的一部分[4]。根据身份认证所需要的信息生成一个 JSON 对象,利用 Base64url 算法对其进行编码,编码后的字符串便是 JWT 的第二部分。按照 RFC4648[14]的定义,Base64url 算法是基于 Base64 算法而形成的一种加密方式,对内容进行了简单编码,无法保证传输过程中信息的机密性。因此可以直接利用 Base64url 算法对 JWT 中的载荷部分进行解码,如图 3 所示,从解码后的内容可以获取身份认证过程所包含的敏感信息。通过载荷部分泄露的敏感信息,推断身份认证的参数及其含义,再结合前面提到的“none”算法漏洞,攻击者可以直接接管账号或提取权限[9]。

```
Headers = {
  "alg": "HS256",
  "typ": "JWT"
}

Payload = {
  "username": "Lisi",
  " idCard": "360102199003071754",
  " sub": "Lisi@qq.com",
  "exp": 1681567389,
  " admin": "true",
  "iat": 1678802589,
  "password": "123456",
  "Email": "Lisi@qq.com"
}

Signature = "81wT_wmzN8LjmzGvS8EHppHR_Dxo6C6Ba1FAlrEU4Hk"
```

图 3 基于载荷部分的敏感信息泄露漏洞

(3)针对签名的攻击。

①算法混淆攻击。

签名部分主要是根据头部的 alg 参数设置的算法类型对 Base64url 算法编码后的头部和载荷两部分内容进行签名,从而保证了 JWT 在数据传输过程的不可篡改性。RFC 7518[13]标准声明了对称算法和非对称算法均可以作为 JWT 的签名算法,由于有些开发者的技术实现不规范,提供了与算法无关的签名验证方法 verify( )。算法伪码如下:该方法只是根据头部(Header)的 alg 参数值确定签名的算法类型,而不验证提供的 Token 加密算法,将导致身份认证服务器验证 JWT 签名的算法与开发人员预期的算法不相同,从而产生对称和非对称算法混淆漏洞。

```
publicKey = <public-key-of-server>;
token = request.getCookie("session");
verify(token, publicKey);
……
function verify(token, secretOrPublicKey){
    algorithm = token.getAlgHeader();
    if(algorithm == "RS256"){
        // Use the provided key as an RSA public key
    } else if (algorithm == "HS256"){
        // Use the provided key as an HMAC secret key
} }
```

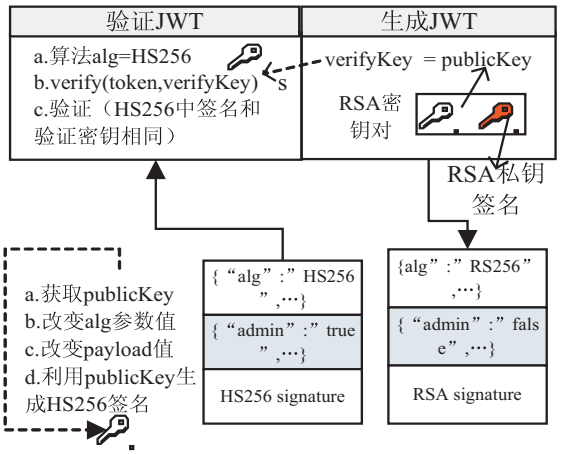

图 4 基于签名部分的算法混淆漏洞流程

产生算法混淆漏洞的流程如图 4 所示。当开发人员调用 verify( )方法但仅使用非对称算法(如 RS256)作为 JWT 的签名算法,即在身份认证过程中,认证服务器使用私钥对数据签名,使用公钥进行签名验证[15]。由于公钥对所有人可见,攻击者获取 RS256 公钥并将其作为 HS256 算法的签名密钥,同时将头部的 alg 参数值设为“HS256”,然后对修改后头部和载荷部分结合 HS256 算法进行签名生成 JWT 令牌。在身份认证过程中,认证服务器将使用 HS256 算法和相同的公钥去验证签名。

②密钥穷举攻击。

密钥穷举攻击(Brute Force Attack)是通过暴力破解的方式对所有可能的密钥值进行尝试来达到破解加密数据的目的的一种攻击方式。当开发者在使用某些签名算法(比如 HS256)时,可将任意长度的字符串作为签名密钥生成签名,此时若使用一个长度较短的弱密钥,将会导致密钥穷举攻击。图 5 为利用 python 暴力破解脚本破解弱密钥签名的 JWT 令牌的运行结果。

```
exp/JWT/bruteJWT.py
JWT token is:eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
.eyJpYXQiOjE2Nzk2MTc2ODIsImV4cCI6MTY3OTcwNDA4Miw
idXNlcm5hbWUiOiJhZG1pbiJ9
._wDgP3iqPer72zXKMpN9m77iydUgZpmójIIihpjiuXg
the key is [ 123456 ]
```

图 5　python 破解弱密钥签名的 JWT 令牌

2　基于国密 SM9 的 JWT 强身份认证方案

2.1　国密 SM9 算法简介

SM9 标识密码算法[16]是利用椭圆曲线对实现的基于标识的密码算法,算法的安全性主要是建立在有关双线性对问题难解性的基础之上。SM9 算法主要由三部分组成,分别是数字签名算法、密钥交换协议、密钥封装机制和公钥加密算法[17]。该文主要使用 SM9 算法中的数字签名部分,主要分为三个步骤:密钥产生、生成签名和签名验证。

参数说明见文献[17],下面主要描述 SM9 数字签名算法的各个步骤。

(1)密钥产生。

KGC(Key Generation Center,密钥生成中心)[16]产生随机数 $ks \in [1, N-1]$ 作为签名主私钥,$P_{pub} = [ks]P_2$ 作为签名主公钥,则签名主密钥对为(s, P_{pub}),P_{pub} 公开,ks 由 KGC 保存。而用户签名密钥对为(P_{pub-A}, d_A),其中用户签名公钥 $P_{pub-A} = [H_1(ID_A \| hid)]P + P_{pub}$,用户签名私钥 $d_A = [s \cdot (H_1(ID_A \| hid) + s)^{-1}]P_1$.

(2)生成签名。

授权服务器根据用户 A 的签名私钥 d_A 运用 SM9 数字签名算法对消息 M 进行计算,生成数字签名(h, S),签名生成流程如表 1 所示。

表 1　生成数字签名的算法流程

	SM9 数字签名生成算法
输入:	系统参数、消息 M 和签名私钥 d_A
输出:	数字签名 (h,S)
A1:	计算群 G_T 中的元素 $g = e(P_1, P_{pub})$
A2:	产生随机数 $r \in [1, N-1]$
A3:	计算群 G_T 中的元素 $w = g^r$
A4:	计算整数 $h = H_2(M \| w, N)$
A5:	计算整数 $l = (r-h) \bmod N$,若 $l = 0$ 则返回 A2
A6:	计算群 G_1 中的元素 $S = [l]d_A$
A7:	计算消息 M 的数字签名 (h,S)

(3)签名验证。

资源服务器根据 SM9 数字签名算法对用户 A 发送的消息 M' 和数字签名 (h', S') 进行验证,签名验证过程中需要进行的步骤如表 2 所示。

表 2　验证数字签名的算法流程

	SM9 数字签名验证算法
输入:	系统参数、消息 M'、用户标识 ID_A 和数字签名 (h', S')
输出:	签名验证成功/失败
B1:	计算 $S' \in G_1$ 是否成立,若不成立则验证失败
B2:	计算 $h' \in [1, N-1]$ 是否成立,若不成立则验证失败
B3:	计算群 G_T 中的元素 $g = e(P_1, P_{pub})$
B4:	计算群 G_T 中的元素 $t = g^{h'}$
B5:	计算整数 $h_1 = H_1(ID_A \| hid, N)$
B6:	计算群 G_2 中的元素 $P = [h_1]P_2 + P_{pub}$
B7:	计算群 G_T 中的元素 $u = e(S', P)$
B8:	计算群 G_T 中的元素 $w' = u \cdot t$
B9:	计算整数 $h_2 = H_2(M' \| w', N)$
B10:	检验 $h_2 = h$ 是否成立,成立则验证成功,反之验证失败

2.2　基于 SM9 的 JWT 强身份认证方案的令牌生成过程

基于 SM9 的 JWT 强身份认证方案提出了一种与传统的 JWT 技术不同的令牌生成方式,生成过程如图 6 所示。头部(Header)和载荷(Payload)部分内容分别为 C_{header} 和 $C_{payload}$,从 Payload 中获取发行时间 Val_{iat} 和用户 Alice 的身份标识 Uid_{Alice},将 Uid_{Alice} 作为国密 SM9 数字签名算法的用户标识,Val_{iat} 作为新鲜因子。

(1)通过国密 SM3 密码杂凑算法 H_{sm3} 将头部和载荷部分均映射成长度为 M 的数字串,即分别计算 $S_h = H_{sm3}(C_{header}, Val_{iat})$ 和 $S_p = H_{sm3}(C_{payload}, Val_{iat})$;

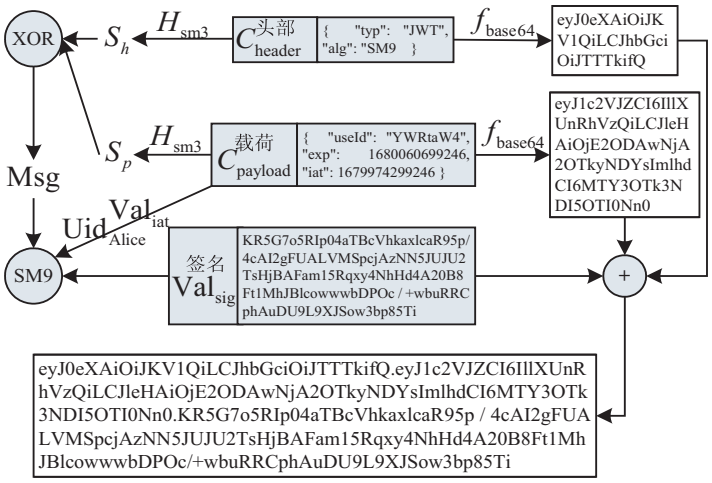

图 6 JWT 令牌生成过程

(2)计算消息 $Msg = S_h \oplus S_p$;

(3)利用国密 SM9 数字签名算法对 Msg 进行签名,运算结果 Val_{sig} 作为令牌 JwtToken 的签名部分;

(4)将 C_{header} 和 $C_{payload}$ 依次作为自变量 x 代入公式 $y = f_{base64}(x)$ 进行计算,其中 $x = \{C_{header}, C_{payload}\}$,运算结果为 Val_{header} 和 $Val_{payload}$;

(5)最终,将三部分结合起来得到 JWT 令牌,即 $JwtToken = Val_{header} + Val_{payload} + Val_{sig}$ 。

2.3 强身份认证方案工作原理

基于 JWT 的身份认证方案可应用的场景有 Web 应用[18]、s 桌面应用[19]、移动应用[19]和嵌入式应用[20],该文提出的基于 SM9 的 JWT 强身份认证方案也可以在这些场景中实现对用户的身份认证。身份认证流程如图 7 所示,主要包括用户 Alice、客户端 C、授权服务器 S_a 和资源服务器 S_r 四个节点;KGC 生成 S_a 的签名主私钥 ks 和签名主公钥 P_{pub} , S_a 保存主私钥 ks,公开主公钥 P_{pub} 。

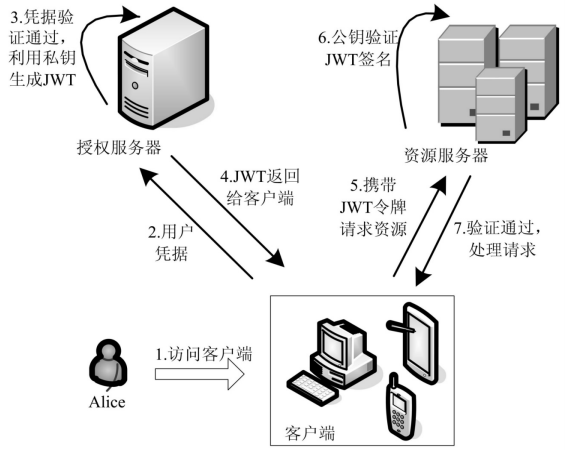

图 7 基于 JWT 的强身份认证方案工作流程

(1)用户 Alice 访问客户端 C。

(2)Alice 通过客户端 C 向授权服务器 S_a 提交用户凭据 T_{Alice} 。

(3)授权服务器 S_a 验证凭据 T_{Alice} ,验证通过后,生成成功状态码,并利用主私钥 ks 和身份标识 Uid_{Alice} 生成签名私钥 d_{Alice} ,然后结合国密 SM9 算法生成令牌 JwtToken ;若验证不通过,则授权服务器 S_a 将会生成失败状态码。

(4)授权服务器 S_a 将状态码和 JWT 令牌返回给客户端 C。

(5)客户端 C 携带 JWT 令牌向资源服务器 S_r 请求相关资源。

(6)资源服务器 S_r 根据主公钥 P_{pub} 和身份标识 Uid_{Alice} 生成 Alice 的签名公钥 $P_{pub-Alice}$,利用 $P_{pub-Alice}$ 验证令牌 JwtToken 。

(7)若资源服务器 S_r 验证通过,则处理请求并返回对应的服务器资源信息;反之,则返回处理失败状态码。

3 安全性分析与比较

3.1 数据完整性

授权服务器对客户端发送的用户凭据验证通过后,利用国密 SM9 数字签名算法生成令牌 JwtToken 的签名部分。如果在交互过程中,攻击者截获数据包,并对令牌 JwtToken 内容进行了篡改,那么相对应的签名部分也会发生变化,从而无法验证签名。因此,该方案可以保证数据完整性,防止数据在传输过程中被篡改。

3.2 不可抵赖性

授权服务器首先对令牌 JwtToken 的前两部分进行运算,然后利用私钥对运算结果进行签名生成第三部分。客户端可利用授权服务器发布的公钥对签名进行验证,从而确定签名来源的正确性;同时资源服务器也可利用公钥对令牌 JwtToken 进行验证,验证通过后再对用户请求进行处理。因此,该方案可以实现授权服务器的不可抵赖性,从而保证了数据的可靠性和真实性。

3.3　抗重放攻击

重放攻击(Replay Attacks)[21]又称回放攻击,指的是攻击者发送一个服务器已经接收过的包,从而实现欺骗服务器的目的。该方案选取时间戳作为新鲜因子嵌入在待签名的消息中,服务器在验证签名时,首先会验证令牌 JwtToken 的新鲜性,如果请求的时间不在有效时间范围内,即令牌 JwtToken 已经失效了,将会跳转到登录页面,重放数据包如图 8 所示。加入时间戳作为新鲜因子,有效预防了重放攻击,保证了数据的唯一性和失效性。

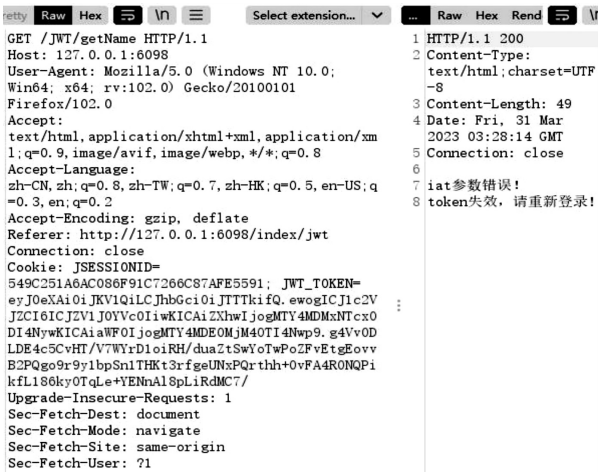

图 8　抗重放攻击测试数据包

3.4　可抵抗针对头部(Header)的“none”算法绕过攻击

该文在前面提到将头部 alg 参数设置为“none”时,后端服务器将不执行验证签名的过程,从而导致客户端携带无签名的 Token 也可进行资源访问。该文提出的认证方案中签名认证过程是基于代码中定义的国密 SM9 算法进行认证,与 alg 参数无关;同时当 alg 参数值不等于 SM9 时,令牌 JwtToken 将会失效。图 9 为模拟客户端携带“none”算法的 JWT 向服务器请求资源的过程。

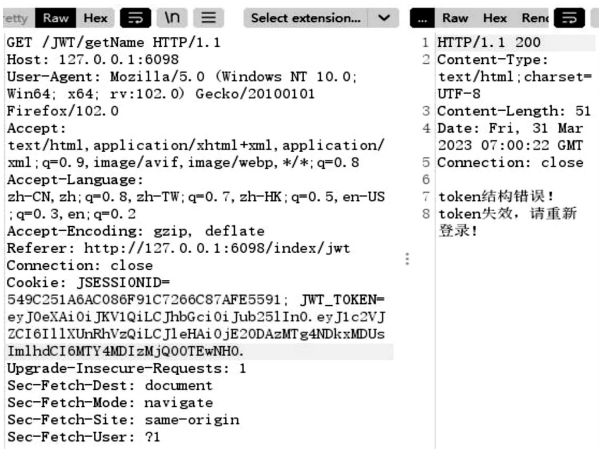

图 9　可抵抗针对头部(Header)的“none”算法绕过攻击测试数据包

3.5　可抵抗针对载荷(Payload)的敏感信息泄露攻击

JWT 在载荷(Payload)部分存放于用户实体的相关信息进行身份认证[4],如果开发者在声明用户实体的过程中直接明文存储相关信息,将会导致敏感信息泄露攻击。该文提出的认证方案针对载荷(Payload)部分所需要声明的用户实体信息均进行了编码操作,并且不将用户敏感信息存放在令牌 JwtToken 中,保证了用户信息的安全性,防止敏感信息泄露漏洞的出现。图 10 为模拟攻击者截获令牌 JwtToken 后解密的内容。

```
Headers = {
  "typ": "JWT",
  "alg": "SM9"
}

Payload = {
  "useId": "YwRtaw4",
  "exp": 1680060699246,
  "iat": 1679974299246
}

Signature = "KR5G7o5RIp04aTBcVhkaxIcaR95p_4cAI2gFUALVMSpcjA"
```

图 10　解密令牌 JwtToken

3.6　可抵抗针对签名(Signature)的算法混淆攻击

该文提出的认证方案中授权服务器利用私钥生成签名,客户端和资源服务器可以利用公钥进行签名验证,并且签名的生成和验证过程都是基于国密 SM9 算法,与 alg 参数无关。不过当 alg 参数值不等于 SM9 时,令牌 JwtToken 将被判定为无效令牌。

3.7　可抵抗针对签名(Signature)的密钥穷举攻击

该文提出的认证方案中的签名是国密 SM9 算法生成的,该算法是用椭圆曲线实现的基于标识的数字签名算法,具有很高的安全性和抗暴力破解能力。并且用户私钥是根据大整数类型的主私钥和用户标识生成的,假设攻击者获得了用户标识,也不可能通过猜测的方式获取用户私钥,从而无法破解加密数据,预防了密钥穷举攻击。

4　实验仿真与安全性比较

4.1　安全性比较

选用国内外应用了 JWT 技术实现了身份验证和授权的平台[22-23]进行安全性分析与对比,分析结果如表 3 所示。

表 3　安全性对比分析

	方案 1	方案 2	方案 3
数据完整性	√	√	√
不可抵赖性	√		√
抗重放攻击	√	√	√
抵抗“none”算法	√	√	√
抵抗敏感信息泄露		√	√
抵抗算法混淆攻击	√	√	√
抵抗密钥穷举攻击		√	√

方案 1:Microsoft Azure Active Directory;

方案 2:百度数据开放平台;

方案 3:文中方案。

4.2 方案实现与仿真

该文在 Windows 10 64 位操作系统下,使用基于 JDK 1.8 的环境和 IntelliJ IDEA 2021.2.1 开发平台,实现了一个基于 B/S 架构的身份认证方案,其中服务器端采用了 SpringMVC 和 Mybatis 框架进行开发,方案的核心代码如下:

服务器端根据用户 Alice 身份标识生成数字签名:

```
// 用户标识
String id_A = userId;
//初始化 Header 和 Payload 部分
String orgHeader = headerJson.to String();
String orgPayload = bodyJson.to String();
// 生成待签名的消息 msg
String msg = getMsg(orgHeader,orgPayload,iatBytes);
// Header 和 Payload 分别进行 base64url 编码
String header = base64URLEn(orgHeader);
String payload = base64URLEn(orgPayload);
// 对消息 msg 进行签名
String signatrue = sm9JwtUtils.sm9_sign(kgc,sm9,P_pub,ks,id_A,msg);
// 生成最终的 JWT
String JwtToken = joinWithDot(header,payload,signatrue);
```

服务器端验证签名:

```
// JWT 令牌解析
String header = base64URLDe(tokenParts[0]);
String payload = base64URLDe(tokenParts[1]); String signature = tokenParts[2];
// 解析 payload 部分
JSONObject payloadJson = new JSONObject(payload);
// 获取用户标识
String id_A = payloadJson.get("useId").to String();
// 获取 iat 字段的值,并转换为字节数组
byte[] iatBytes = ByteBuffer.allocate(Long.BYTES).putLong(payloadJson.optLong("iat")).array();
// 生成消息 msg
String msg = getMsg(header,payload,iatBytes);
//验证签名
boolean flag = sm9JwtUtils.sm9_verify(kgc,sm9,P_pub,ks,id_A,msg,signature);
```

(1)用户 Alice 在客户端通过登录页面向授权服务器端提交用户凭据,验证通过后,授权服务器端根据用户标识 Uid_{Alice} 和主私钥 ks 生成令牌 JwtToken ,并返回给客户端,数据包如图 11 所示。

(2)客户端携带令牌 JwtToken 向资源服务器请求资源,资源服务器解析传递令牌 JwtToken ,获取用户标识 Uid_{Alice} ,并计算消息 Msg ,然后再根据授权服务器公开的主公钥 P_{pub} 和 Uid_{Alice} 验证令牌 JwtToken 中的签名部分 Val_{sig} 。如果验证通过,资源服务器将根据 Uid_{Alice} 处理客户端请求,并返回对应的资源,资源请求交互过程的数据包如图 12 所示。

```
HTTP/1.1 302
Set-Cookie:
JWT_TOKEN=eyJ0eXAiOiJKV1QiLCJhbGciOiJTTTkifQ.eyJ1c2VJZCI6IllXUnRhVzQiLCJleH
AiOjE2ODAwMTMyMjA5NzQsImlhdCI6MTY3OTkyNjgyMDk3NH0.DK2jKPn2HQMUvyzjGipP/
W06PiNYjkp/
46751Az+W72wW+Gp2vltQTz6aQwY9bgHwgI5Im32sMRovpDKO2IX+6+QHBO57PhQG1QnngjTDy5
EaIfdZB2nE365EId7EPfY; Max-Age=86400; Expires=Tue, 28-Mar-2023 14:20:21
GMT; Path=/; HttpOnly
Location: http://127.0.0.1:6098/index
Content-Language: zh-CN
Content-Length: 0
Date: Mon, 27 Mar 2023 14:20:21 GMT
Keep-Alive: timeout=60
Connection: keep-alive
```

图 11 服务器端将 JWT 令牌给客户端的数据包

```
GET /JWT/getName HTTP/1.1
Host: 127.0.0.1:6098
Connection: keep-alive
Cache-Control: max-age=0
sec-ch-ua: "Google Chrome";v="111", "Not(A:Brand";v="8", "Chromium";v="111"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/111.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/
avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Cookie: JSESSIONID=C24FEE241183966BB4D71F98C9345AA7;
JWT_TOKEN=eyJ0eXAiOiJKV1QiLCJhbGciOiJTTTkifQ.eyJ1c2VJZCI6IllXUnRhVzQiLCJleH
AiOjE2ODAwMTMyMjA5NzQsImlhdCI6MTY3OTkyNjgyMDk3NH0.DK2jKPn2HQMUvyzjGipP/
W06PiNYjkp/
46751Az+W72wW+Gp2vltQTz6aQwY9bgHwgI5Im32sMRovpDKO2IX+6+QHBO57PhQG1QnngjTDy5
EaIfdZB2nE365EId7EPfY

HTTP/1.1 200
Content-Type: text/html;charset=UTF-8
Content-Length: 26
Date: Mon, 27 Mar 2023 14:21:44 GMT
Keep-Alive: timeout=60
Connection: keep-alive

......JWT.....admin
```

图 12 资源请求交互过程

5 结束语

针对 JWT 技术中存在的一些漏洞,该文提出了一种基于国密 SM9 算法的 JWT 强身份认证方案。与现有开放平台所使用的 JWT 技术进行安全性分析与比较,可知该认证方案具有数据完整性、不可抵赖性和抗重放攻击等特点,并且可抵抗“none”算法绕过、敏感信息泄露、算法混淆和密钥穷举等攻击。但实验表明该方案的运行效率不高,有待进一步完善。

参考文献:

[1] 杨泽霖,王基策,徐 斐,等. 远程办公系统安全综述[J]. 信息安全学报,2022,7(6):31-47.

[2] RAHMATULLOH A,GUNAWAN R,NURSUWARS F M S. Performance comparison of signed algorithms on JSON web token[J]. IOP Conference Series:Materials Science and Engineering. IOP Publishing,2019,550(1):012023.

[3] JÁNOKY L V,LEVENDOVSZKY J,EKLER P. An analysis on the revoking mechanisms for JSON web tokens[J]. International Journal of Distributed Sensor Networks,2018,14(9):15501477188.

[4] JONES B M,BRADLEY J,SAKIMURA N. JSON web token (JWT)[J]. Request For Comments,2015,7519:1-30.

[5] MCLEAN T. Critical vulnerabilities in JSON web token libraries[EB/OL]. 2015[2023-07-15]. https://www.chosenplaintext.ca/2015/03/31/jwt-algorithm-confusion.html.

[6] LANGKEMPER S. Attacking JWT authentication. [EB/OL]. 2016[2023-07-15]. https://www.sjoerdlangkemper.nl/2016/09/28/attacking-jwt-authentication/.

[7] NASSAR K. CVE-2022-21449 TLS PoC[EB/OL]. 2022[2023-07-15]. https://github.com/khalednassar/CVE-2022-21449-TLS-PoC.

[8] PUTRA A A, ALAM R, NUR A M. Stateless authentication with JSON web tokens using RSA-512 algorithm[J]. JURNAL INFOTEL,2019,11(2):36-42.

[9] 钱君生,杨　明,韦　巍. API 安全技术与实战[M]. 北京:机械工业出版社,2021.

[10] ETHELBERT O, MOGHADDAM F F, WIEDER P, et al. A JSON token-based authentication and access management schema for cloud SaaS applications[C]//2017 IEEE 5th international conference on future internet of things and cloud (FiCloud). Prague:IEEE,2017:47-53.

[11] 陈　佳. 一种基于 JWT 令牌认证的电力系统微服务认证授权方案[J]. 电工技术,2021(16):151-154.

[12] AHMED S, MAHMOOD Q. An authentication based scheme for applications using JSON web token[C]//2019 22nd international multitopic conference (INMIC). Islamabad: IEEE, 2019:1-6.

[13] JONES B M. JSON web algorithms (JWA)[J]. Request for Comments,2015,7518:1-69.

[14] JOSEFSSON S. The Base16, Base32, and Base64 data encodings[J]. Request For Comments,2006,4648:1-18.

[15] TANAEM, PENIDAS F. RESTFul web service untuk sistem pencatatan transaksi studi kasus PT. XYZ[J]. Jurnal Teknik Informatika dan Sistem Informasi,2016,2(1):1-10.

[16] GB/T 38635.1-2020,信息安全技术 SM9 标识密码算法第 1 部分:总则[S]. 北京:国家市场监督管理总局;国家标准化管理委员会,2020:1-62.

[17] 袁　峰,程朝辉. SM9 标识密码算法综述[J]. 信息安全研究,2016,2(11):1008-1027.

[18] FOTIOU N, SIRIS V A, POLYZOS G C. Capability-based access control for multi-tenant systems using OAuth 2.0 and verifiable credentials[C]//2021 international conference on computer communications and networks (ICCCN). Athens: IEEE,2021:1-9.

[19] SETIAWAN A, PURNAMASARI A I. Implementasi JSON web token berbasis algoritma sha-512 untuk otentikasi aplikasi batikKita[J]. Jurnal RESTI (Rekayasa Sistem Dan Teknologi Informasi),2020,4(6):1036-1045.

[20] SOLAPURKAR P. Building secure healthcare services using OAuth 2.0 and JSON web token in IOT cloud scenario[C]//2016 2nd international conference on contemporary computing and informatics (IC3I). Greater Noida:IEEE,2016:99-104.

[21] 肖斌斌,徐雨明. 基于双重验证的抗重放攻击方案[J]. 计算机工程,2017,43(5):115-120.

[22] Microsoft. Microsoft 标识平台概述[EB/OL]. (2023-05-08)[2023-07-15]. https://learn.microsoft.com/zh-cn/azure/active-directory/develop/v2-overview.

[23] 马　琳,宋俊德,宋美娜. 开放平台:运营模式与技术架构研究综述[J]. 电信科学,2012,28(6):125-140.