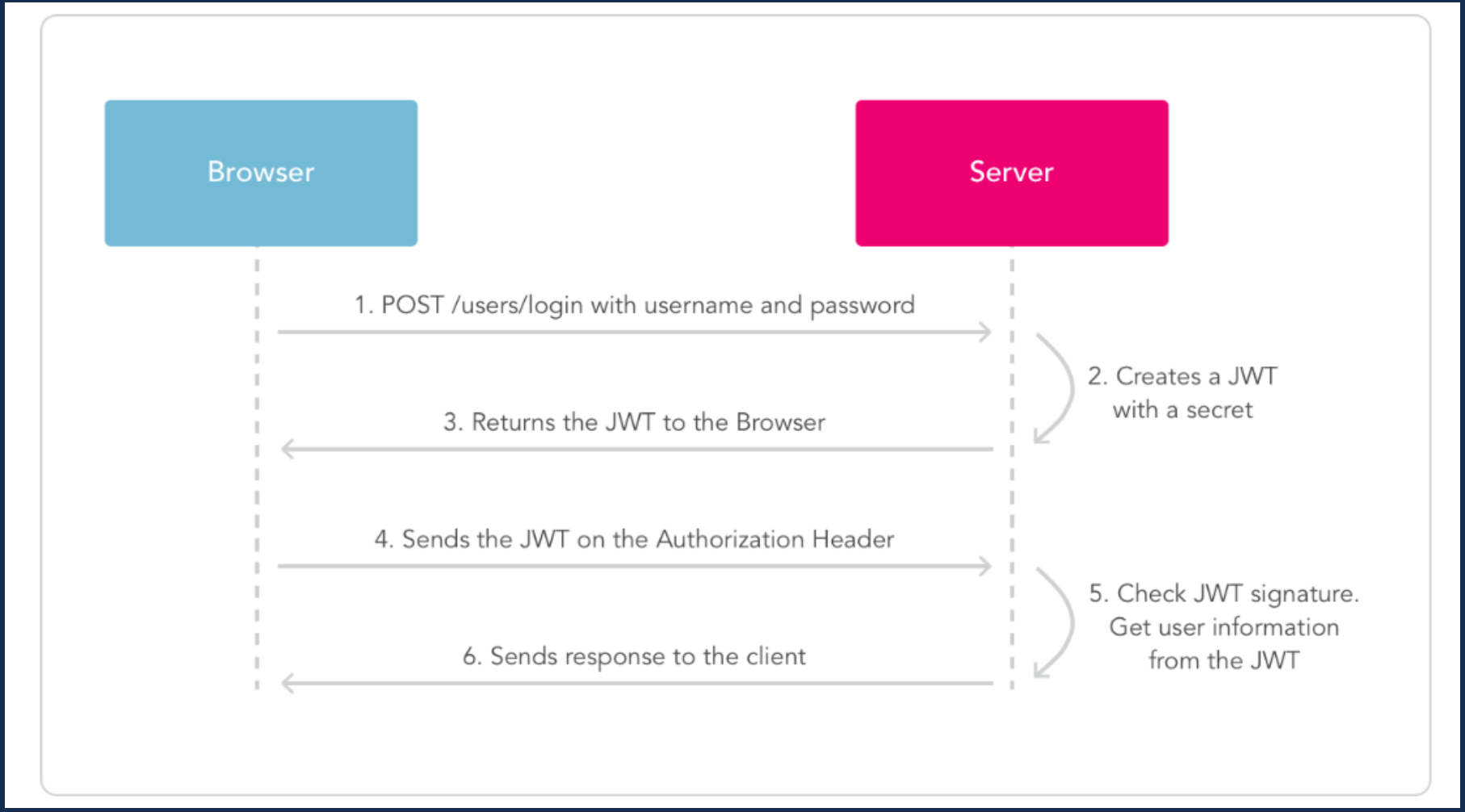
1 What is JWT ?

JWT(JSON Web Token) 是一种用于双方之间传递安全信息的简洁的、URL 安全的表述性声明规范。JWT 作为一个开放的标准（RFC 7519），定义了一种简洁的，自包含的方法用于通信双方之间以 JSON 对象的形式安全的传递信息。因为数字签名的存在，这些信息是可信的，JWT 可以使用 HMAC 算法或者是 RSA 的公私秘钥对进行签名。

- 简洁(Compact): 可以通过 URL，POST 参数或者在 HTTP header 发送，因为数据量小，传输速度也很快
- 自包含(Self-contained)：负载中包含了所需要的用户的所有信息，避免了多次查询数据库

2 JWT 通信过程


Browser
Server
1. POST /users/login with username and password
2. Creates a JWT with a secret
3. Returns the JWT to the Browser
4. Sends the JWT on the Authorization Header
5. Check JWT signature. Get user information from the JWT
6. Sends response to the client

3 JWT 的组成

- 3.1 encode

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

3 JWT 的组成

- 3.2 decode

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

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." + base64UrlEncode(payload),
    secret
)
```

4 JWT 的主要应用场景

- 4.1 身份认证

这种场景下，一旦用户完成了登陆，在接下来的每个请求中都包含一个包含 JWT 的 Token，可以用来验证用户身份以及对路由，服务和资源的访问进行权限验证。由于它的开销非常小，可以轻松的在不同的系统中传递，所有目前在单点登录（SSO）中比较广泛的使用了该技术。

跨域还是同域传递完全取决于你传递的方式：

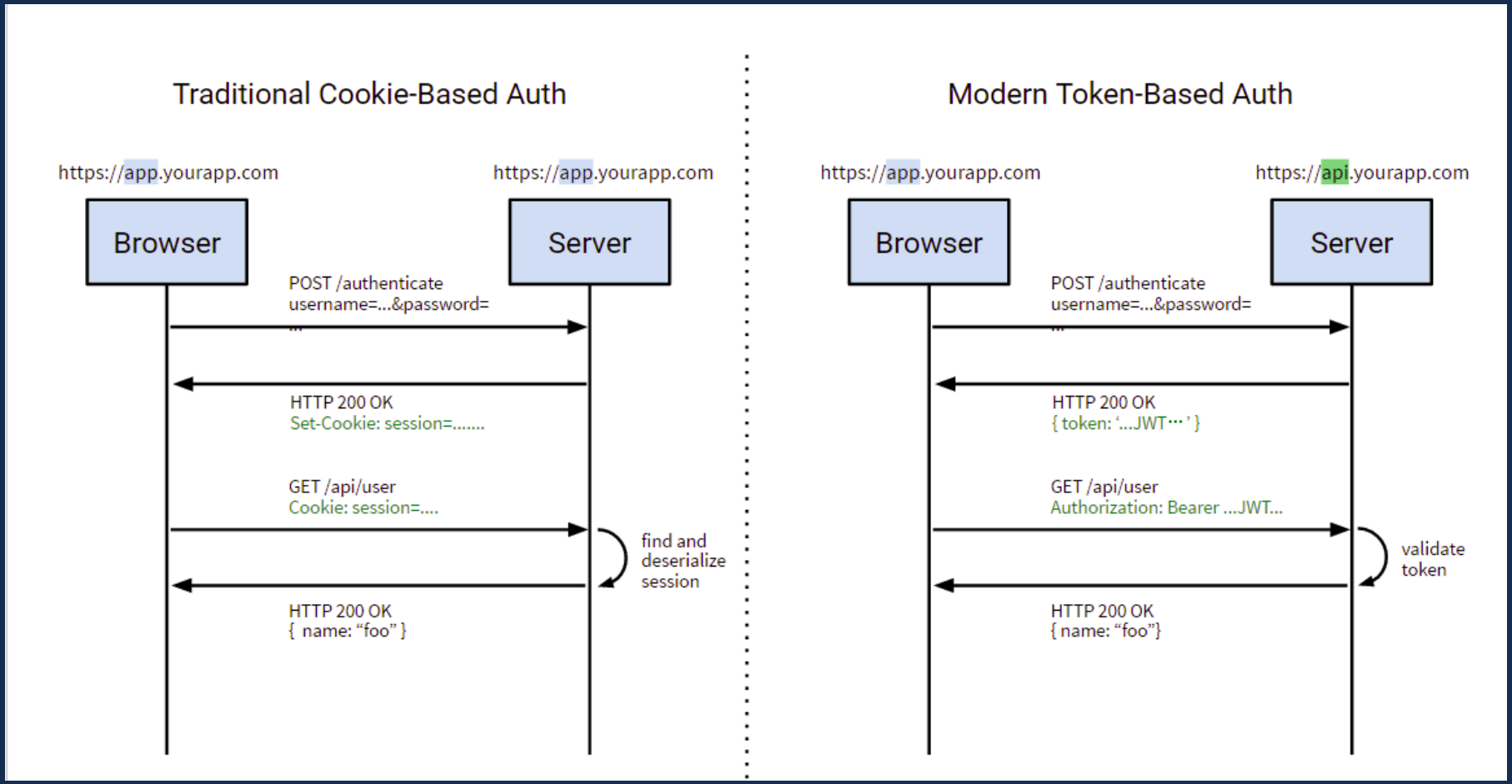
```
http header
Authorization: Bearer <token>
Cookie: token=Bearer <token>

URL query
https://www.aidandai.com?authorization=Bearer <token>
```

4 JWT 的主要应用场景

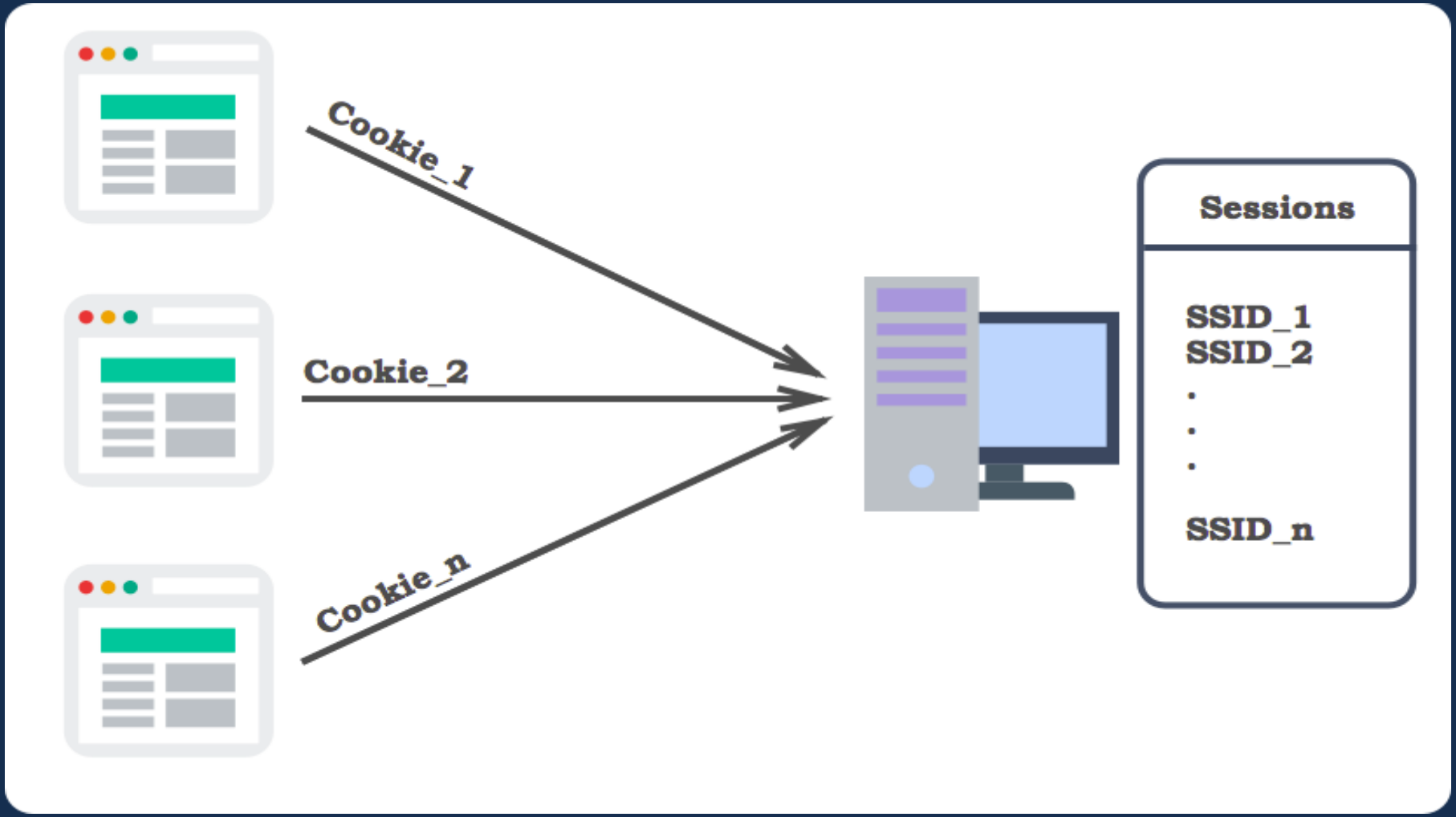
- 4.2 信息交换

JWT 是一个在双方间安全传递信息的好方法，因为它们能被签名。例如：我们可以使用公私秘钥对签名，这样就可以确保发送者以及他们发送的信息的真实性。此外，当签名使用头部和有效载荷计算时，还可以验证内容未被篡改。（这里不太了解，欢迎交流～）

5 常见用户认证机制


Traditional Cookie-Based Auth
https://app.yourapp.com
https://app.yourapp.com
Browser
Server
POST /authenticate
username=...&password=
...
HTTP 200 OK
Set-Cookie: session=.....
GET /api/user
Cookie: session=....
find and
deserialize
session
HTTP 200 OK
{ name: "foo" }
Modern Token-Based Auth
https://app.yourapp.com
https://api.yourapp.com
Browser
Server
POST /authenticate
username=...&password=
...
HTTP 200 OK
{ token: '...JWT··· ' }
GET /api/user
Authorization: Bearer ...JWT...
validate
token
HTTP 200 OK
{ name: "foo"}

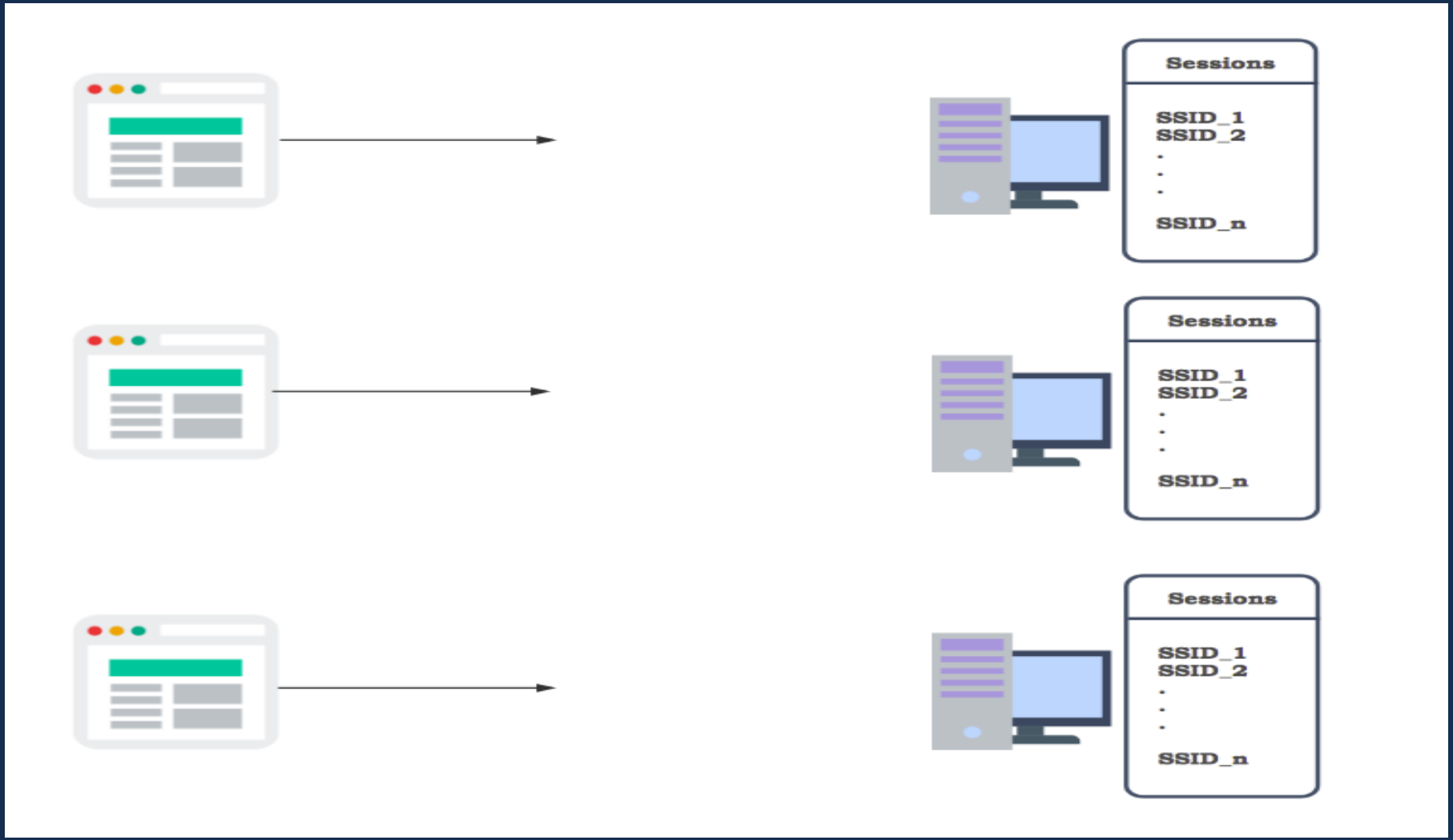
6 Cookie Auth 面临的问题



6 Cookie Auth 面临的问题

- 服务器端　Session 的存储方式
  - 内存、文件、数据库
- 单机部署　Session 造成的问题
  - 安全问题、高并发下的性能问题

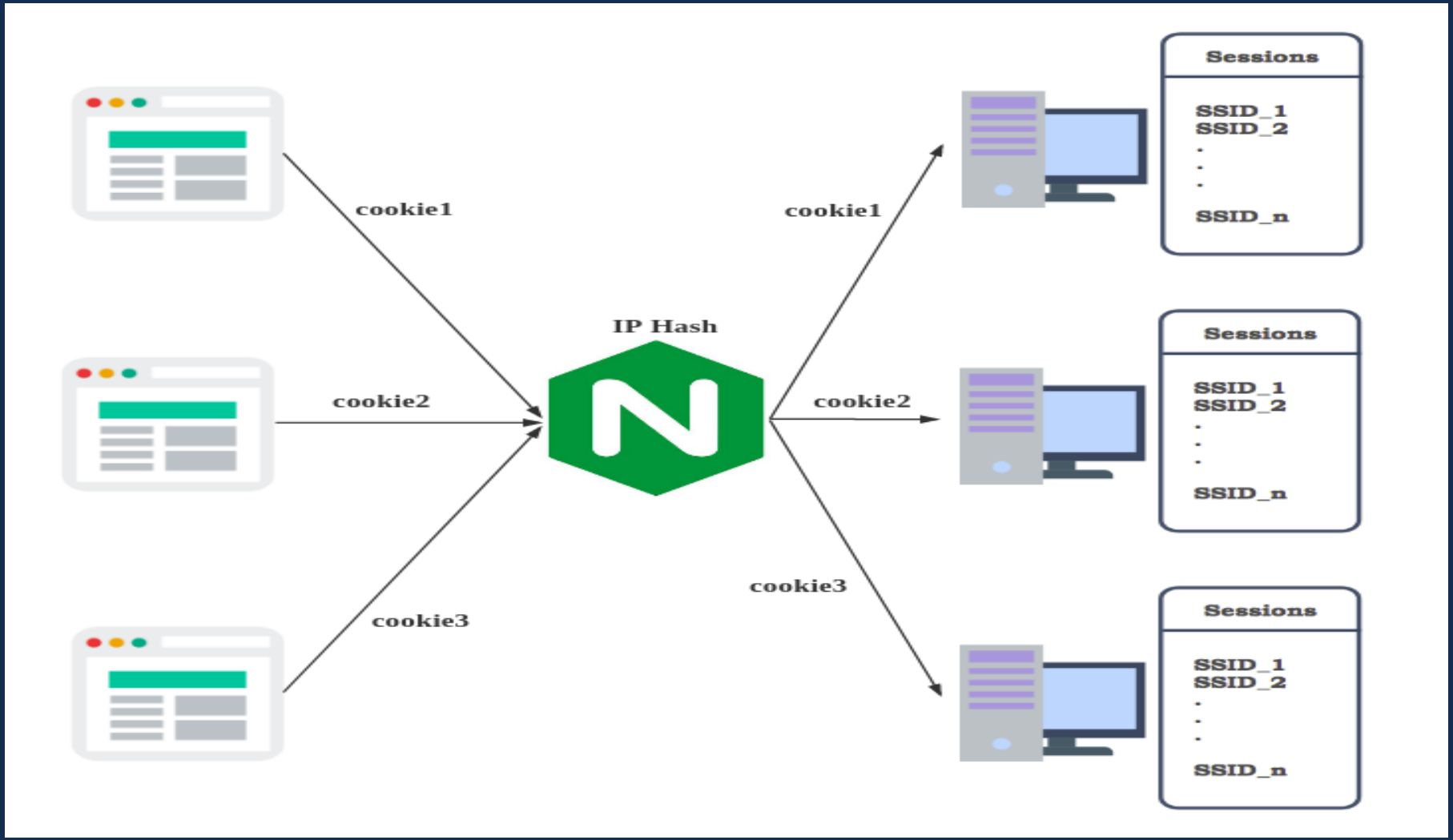
6 Cookie Auth 面临的问题



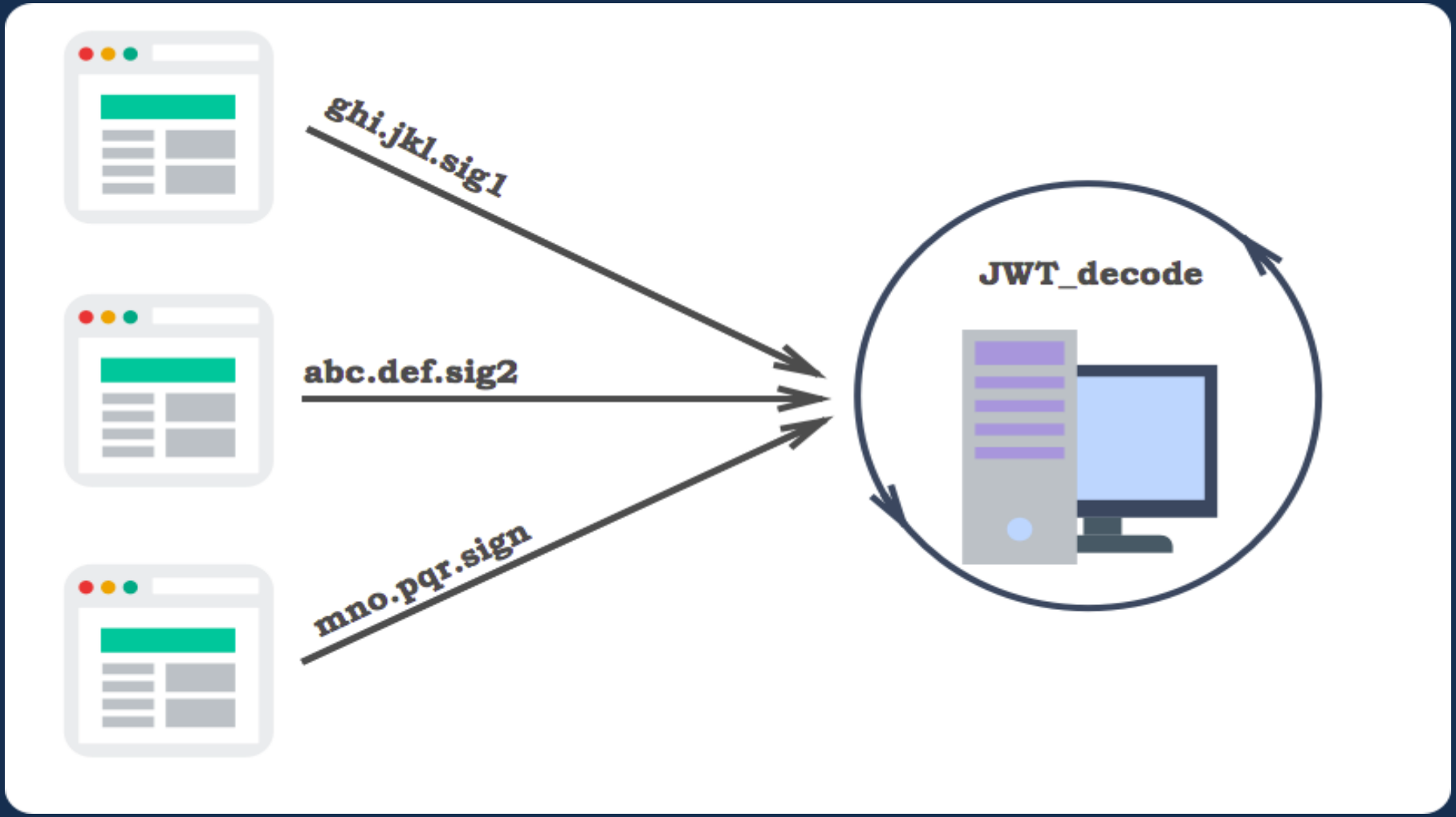
6 Cookie Auth 面临的问题

- 服务器端　Session 的存储方式
  - 文件（NFS）、数据库
- 多机部署(负载均衡)　Session 造成的问题
  - Session 共享、安全问题、高并发下的性能问题

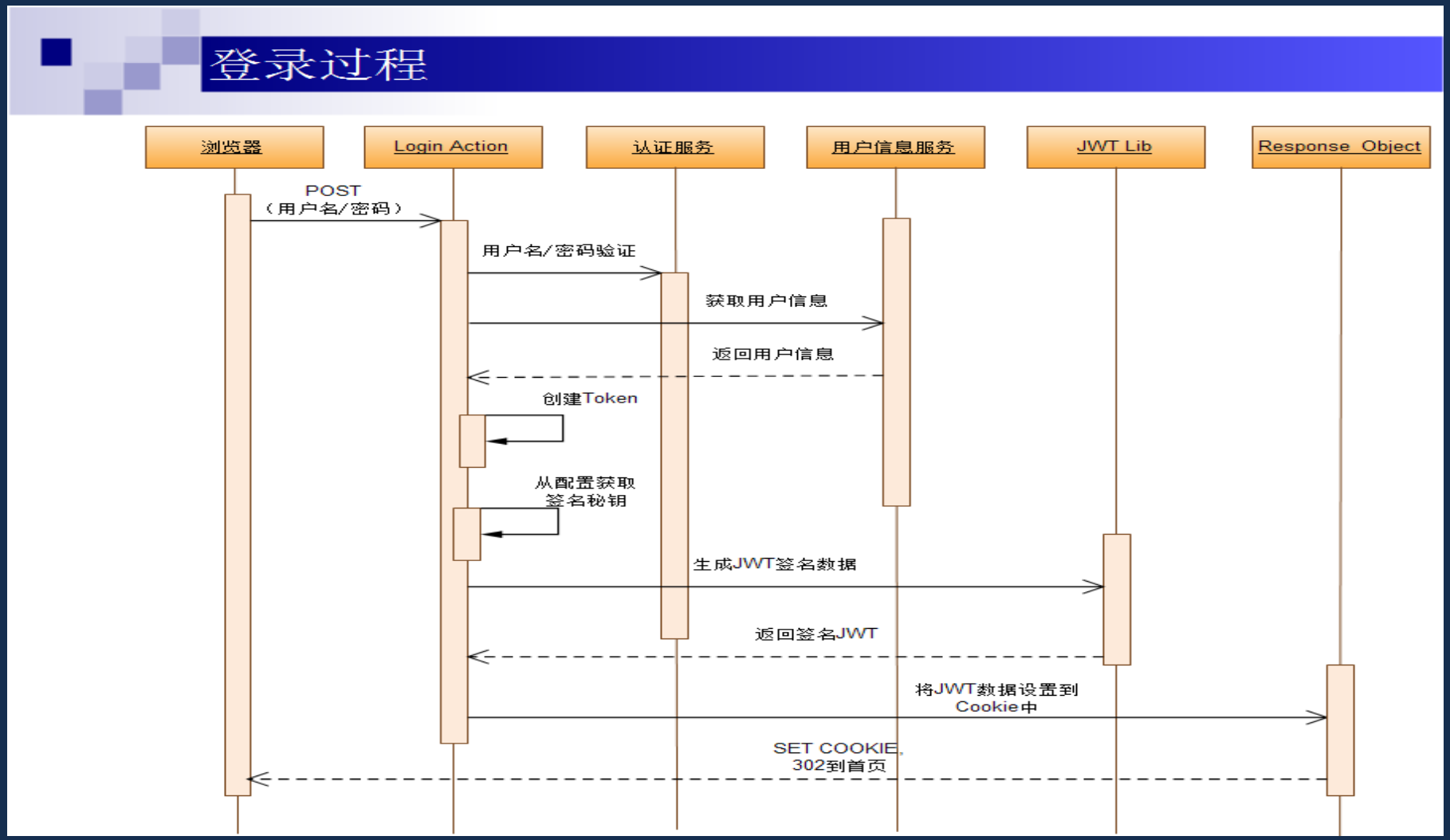
7 Cookie Auth 优化方案



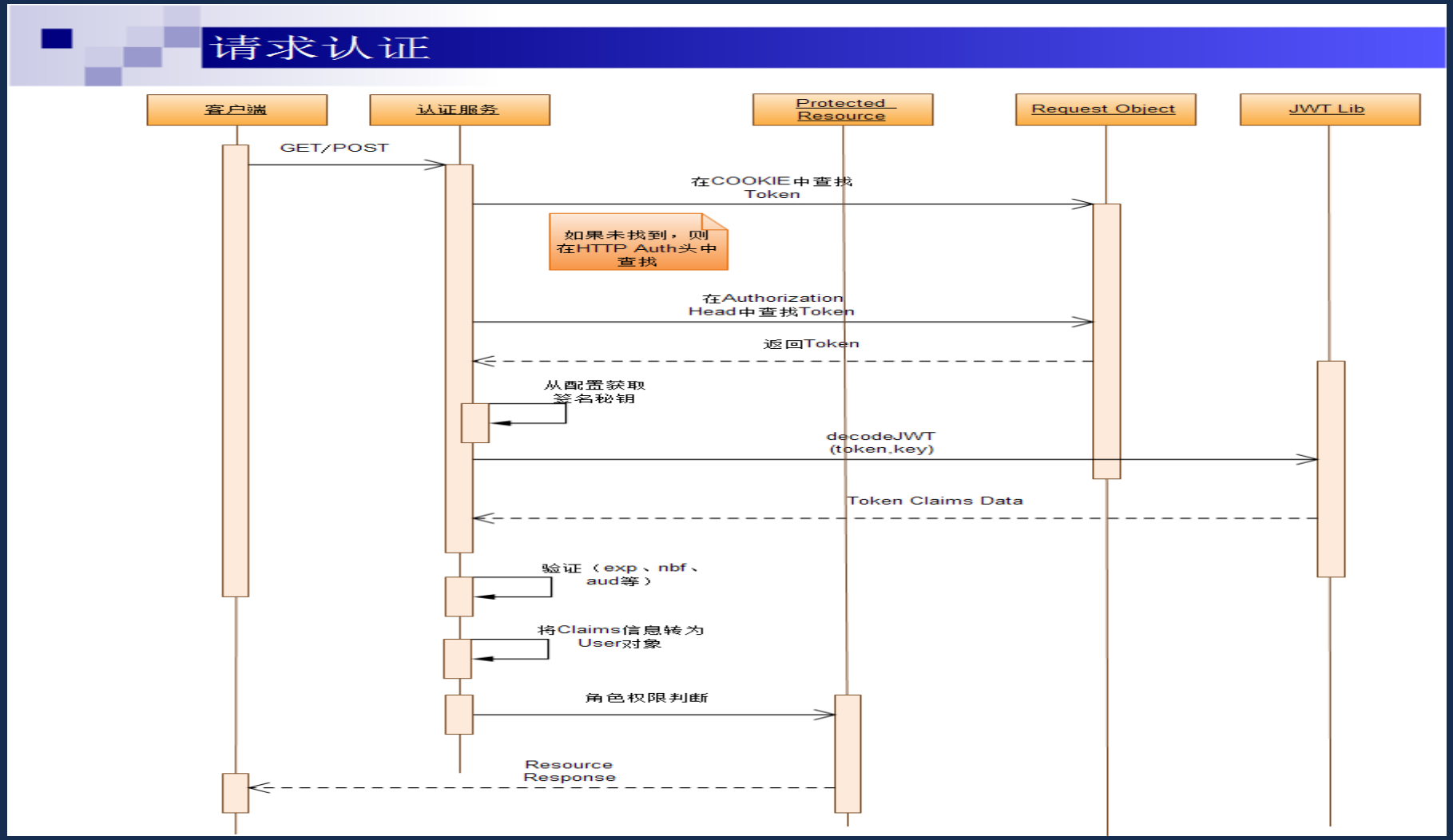
8 Token Auth 的优势



9 Token Auth with JWT 的登录认证



10 Token Auth with JWT 的请求认证



11 Node.js 中 JWT 的实现

- 11.1 encode

```
const base64Header = base64URLEncode(header)
const base64Payload = base64URLEncode(payload)

// 可以使用多种加密算法
const signature = HMACSHA256(
  `${base64Header}.${base64Payload}`,
    secret
)

const jwt = `${base64Header}.${base64Payload}.${signature}`
```

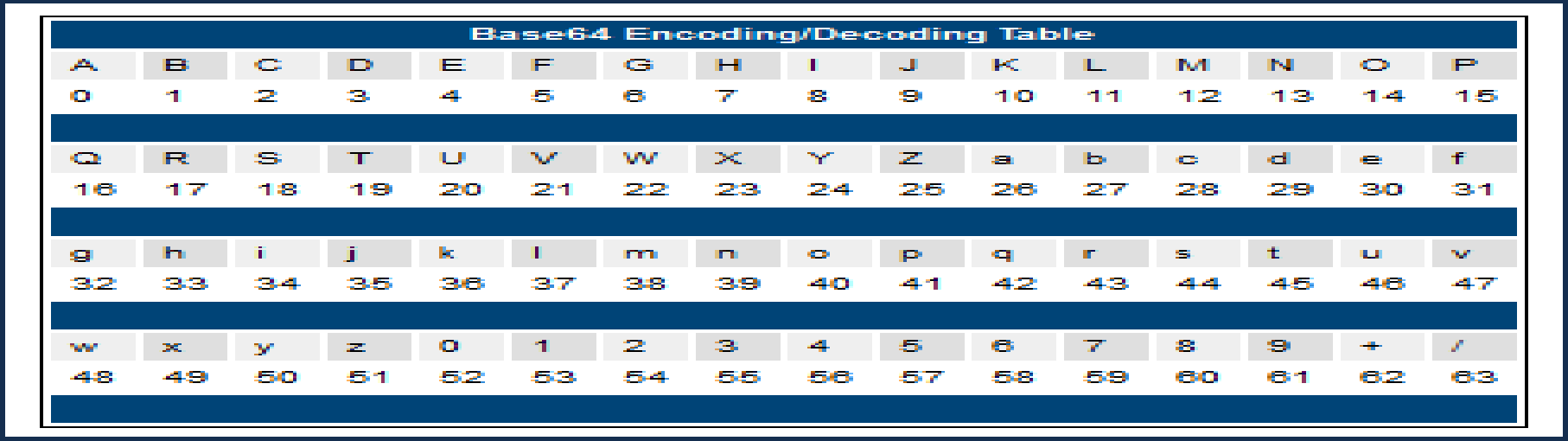
11 Node.js 中 JWT 的实现

- 11.2 decode

```
const result = jwt.decode(jwt, secret)

// verify

// decode
```

12 base64 加密原理

- 12.1 Base64 Encoding/Decoding Table



Base64 Encoding/Decoding Table

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

12 base64 加密原理

- 12.2 base64 加密例一

字母:	**T**	**o**	**m**	
ASCII:	84	111	109	
8bit字节:	01010100	01101111	01101101	
6bit字节:	010101	000110	111101	101101
十进制:	21	6	61	45
对应编码:	V	G	9	t

12 base64 加密原理

- 12.3 base64 加密例二

字母:	L	u	c	y				
ASCII:	76	117	99	121				
8bit字节:	01001100	01110101	01100011	01111001	00000000	00000000		
6bit字节:	010011	000111	010101	100011	011110	010000	000000	000000
十进制:	19	7	21	35	30	16	(异常)	(异常)
对应编码:	T	H	V	j	e	Q	=	=

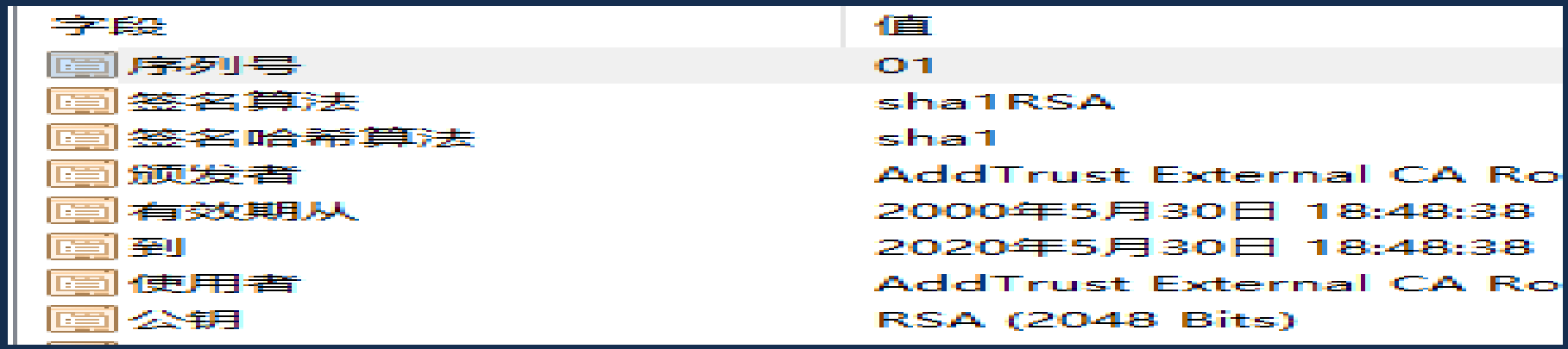
13 JWT 中常用的加密算法

- 13.1 密码学中的一些概念
- 对称加密
  - 秘钥
- 非对称加密
  - 秘钥对（公钥、私钥）
  - 公钥加密、私钥解密
  - 私钥生成数字签名、公钥验证数字签名（发送：多对一；客户端到服务器）
  - 私钥生成数字证书、公钥验证数字证书（广播：一对多；服务器到客户端）
- 数字签名、数字证书

13 JWT 中常用的加密算法

- 13.2 数字签名、数字证书

数字证书的作用是将 持有者的公钥 和 持有者的身份 绑定起来



字段	值
序列号	01
签名算法	sha1RSA
签名哈希算法	sha1
颁发者	AddTrust External CA Ro
有效期从	2000年5月30日 18:48:38
到	2020年5月30日 18:48:38
使用者	AddTrust External CA Ro
公钥	RSA (2048 Bits)

13 JWT 中常用的加密算法

- 13.3 Hmac 加密算法(带密钥的Hash函数)

```
const crypto = require('crypto')
const hash = crypto.createHmac('sha256', 'a secret')
    .update('some data to hash')
    .digest('hex')

//   7fd04df92f636fd450bc841c9418e5825c17f33ad9c87c518115a45971f7f77e
```

13 JWT 中常用的加密算法

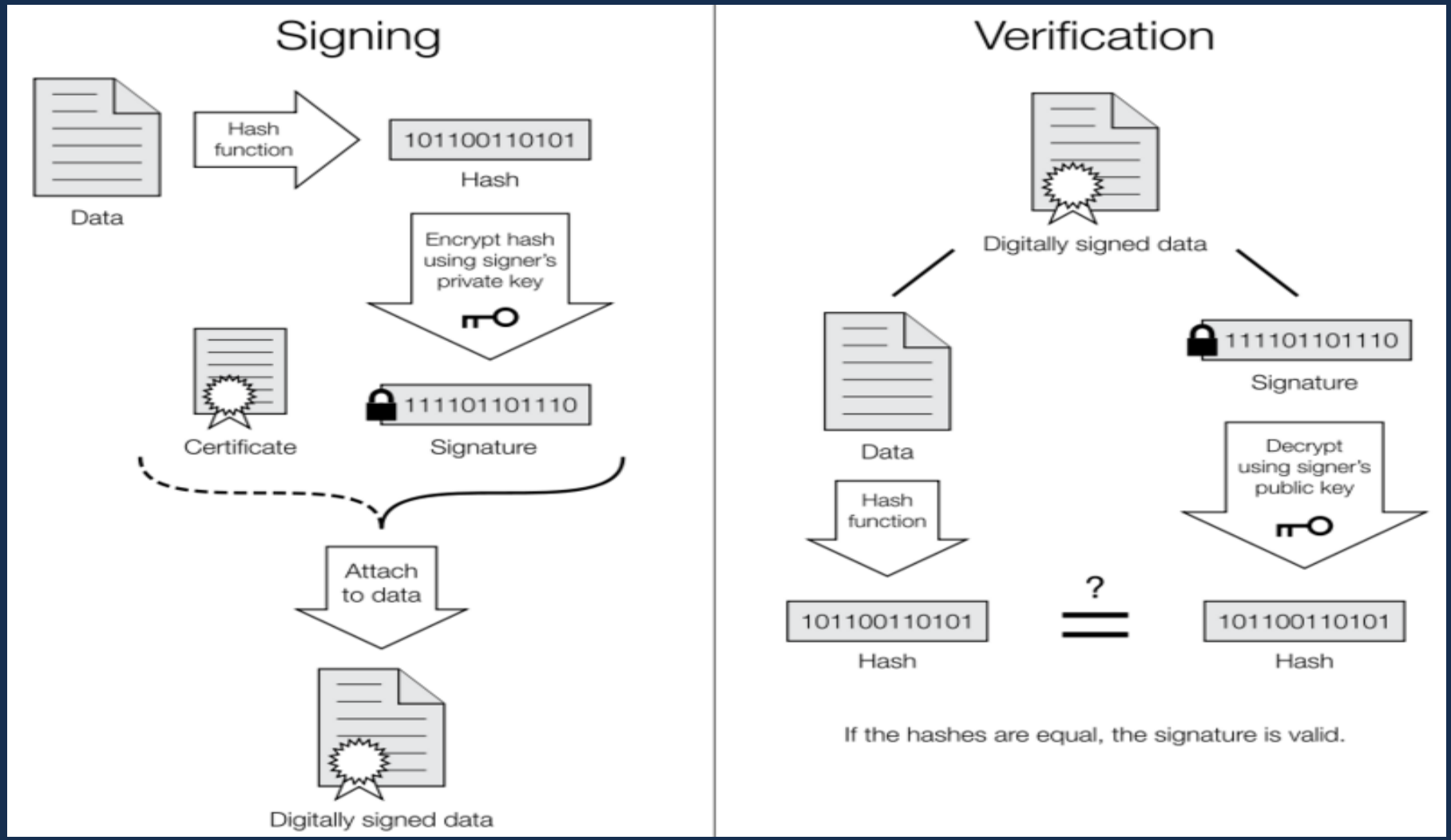
- 13.4 数字签名的制作和验证过程

「网站公钥」要保证可信度，需要通过 CA(Certification Authority) 认证。 通过将「网站公钥」用「CA 的私钥」签名生成「数字证书」，然后客户用「CA 的公钥」确认「数字证书」的完整性，里面会有 「网站公钥，该「网站公钥」就是可信的。然后客户在用「网站公钥」确认网站发送过来的信息的完整性。

```
Data + Signature + Digital Certificate
```

13 JWT 中常用的加密算法

- 13.5 数字签名的制作和验证过程



14 JWT 畅想 - 用户权限认证系统构想

需要在页面路由、API 接口或者资源服务器层验证用户权限。

- URL Request
  - Redirect
- Common Resource Request
  - 同域：同域下　cookie 存储　JWT 验证用户权限
  - 跨域：跨域时　query 带上 JWT 验证用户权限
- API Resource Request
  - LocalStorage 存储　JWT 验证用户权限
- Authentication
  - 资源数据表实现

15 基于 JWT 的 Token 认证的安全问题

- 用户名/密码验证过程的安全性（HTTPS）
- Cookie 的安全问题（HTTP-Only）
- Replay Attacks（时间戳）
- MITM（Man-In-The-Middle）Attacks（HTTPS）

16 参考资料

- Introduction to JSON Web Tokens
- JWT 简介
- JWT 在前后端分离中的应用与实践
- 手动实现一个 json web token
- 基于 Token 的 WEB 后台认证机制
- JSON Web Token (JWT)
- JWT: how to handle GET requests when user opens a new tab?
- JSON 网络令牌库中出现严重漏洞
- 阮一峰：数字签名是什么?
- 阮一峰：RSA算法原理（一）
- 阮一峰：RSA算法原理（二）
- 怎么保证「CA 的公钥」是真实的?

Thank You

- 自己造的轮子
  - https://github.com/AidanDai/node.jwt
  - https://www.npmjs.com/package/node.jwt