

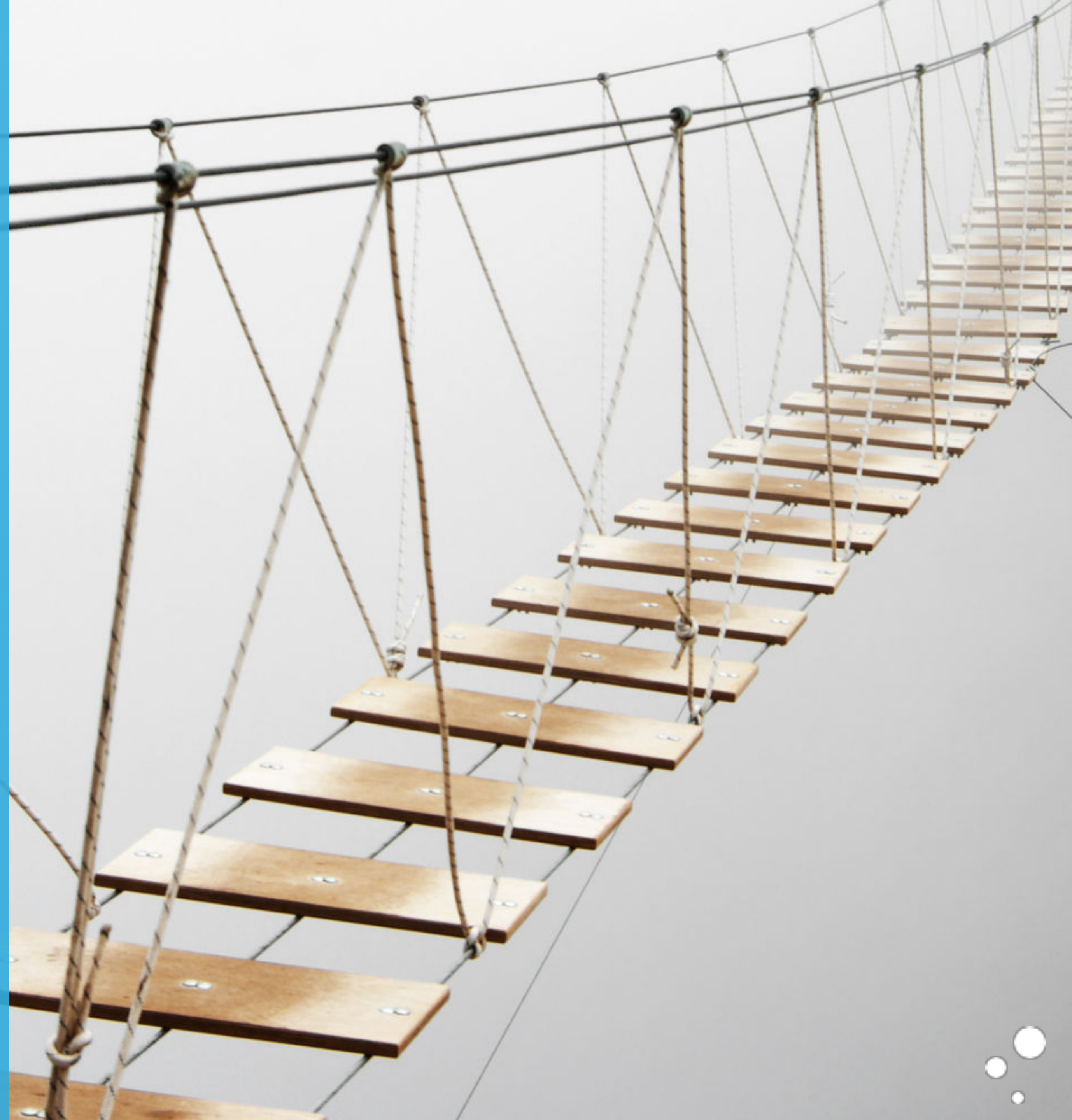


PentesterLab

JWAT ??

Attacking JSON WEB TOKENS...

Louis Nyffenegger
@PentesterLab
louis@pentesterlab.com





Security Engineer

- **Pentester/Code Reviewer/Security consultant/Security architect**
- **Run a website to help people learn security**



PentesterLab:

- **Platform to learn web security/penetration testing**
- **100% Hands-on**
- **Available for individuals (free and PRO) and enterprises**



Who uses JWT?



- A lot of people for OAuth
- A lot of people for sessions
- A lot of people to manage trust
- A lot of people for password reset
- A lot of people who care about being stateless and multi-datacenter architecture





- JOSE:
 - Javascript Object Signing and Encryption
 - Also the name of the working group
- JWT: JSON Web Token == “jot” Token
- JWE: JSON Web Encryption
- JWS: JSON Web Signature
- JWK: JSON Web Key
- JWA: JSON Web Algorithm



Crypto 101



Signature vs Encryption



Encryption gives you **confidentiality**

Signature gives you **integrity**



Multiple ways of signing



- With a secret using HMAC
- With a private key using RSA/EC/... (asymmetric)



Signing with a secret



Sign!



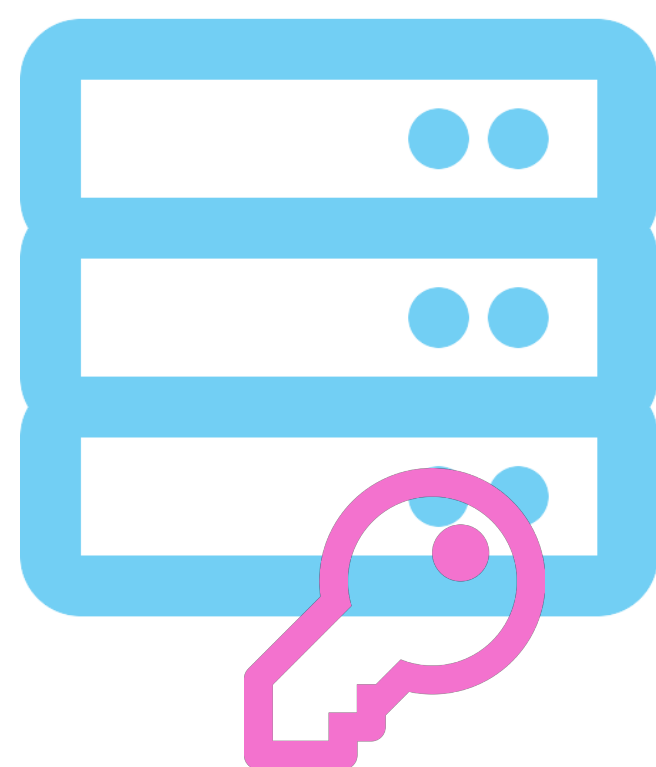
Verify!



Secret



Signing: asymmetric



Sign!



Verify!



Public



Private



THE JWT FORMAT



JavaScript Object Notation (JSON)



Human readable format to store or transmit objects

```
{  
  "firstname": "John",  
  "lastname": "Doe",  
  "age": 30,  
  "hobbies": ["security", "hacking", "lock picking"],  
  "address": {  
    "streetAddress": "1337 Hacker Street",  
    "city": "Hacker Town",  
    "country": "HackerLand"  
  }  
}
```



The Compact JWS Format



3 parts in a JSON Web Token:

Header

Payload

Signature



The Compact JWS Format



Separated by a dot

Header

■

Payload

■

Signature



The Compact JWS Format



Separated by a dot

```
eyJ0eXAiOiJK  
V1QiLCJhbGci  
OiJIUzI1NiJ9
```

■

```
eyJsb2dpci  
I6ImFkb  
WluIn0
```

■

```
FSfvCBAwypJ4abF6  
jFLmR7JgZhkW674  
Z8dIdAIRyt1E
```

eyJ = Base64 (' { " ')



The Compact JWS Format



Header and Payload are base64* encoded JSON

* urlsafe base64 encoding without padding

Base64 ({ ... })

■

Base64 ({ ... })

■

Base64 (...)

The signature is also base64 encoded



The Compact JWS Format: Encoding



Urlsafe base64 encoding without padding:

```
static string base64urlencode(byte [] arg)
{
    string s = Convert.ToBase64String(arg); // Regular base64 encoder
    s = s.Split('=')[0]; // Remove any trailing '='s
    s = s.Replace('+', '-'); // 62nd char of encoding
    s = s.Replace('/', '_'); // 63rd char of encoding
    return s;
}
```

*<https://tools.ietf.org/html/rfc7515#appendix-C>



The JWT Format: header



The header contains an algorithm “alg” attribute:

```
Base64 ( { "alg": "HS256",  
           "typ": "JWT" } )
```

■

...

■

...

To tell how the token was signed.

In this example HMAC with SHA256 was used



The JWT Format: Algorithms



A lot of different algorithms are supported*:

✓ None	✓ RS256	✓ ES256	✓ PS256
✓ HS256	✓ RS384	✓ ES384	✓ PS384
✓ HS384	✓ RS512	✓ ES512	✓ PS512
✓ HS512			

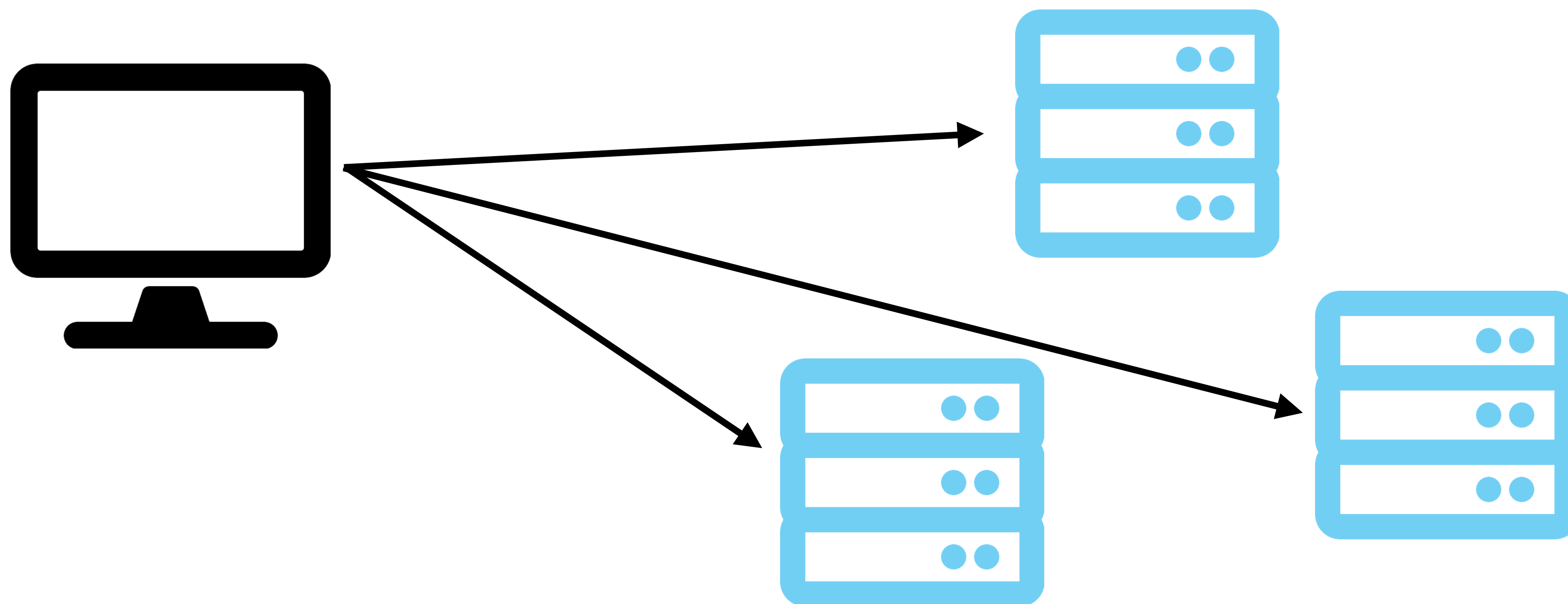
* <https://jwt.io/> covers most



The JWT Format: Algorithms



Scenario: one client talking to multiple services

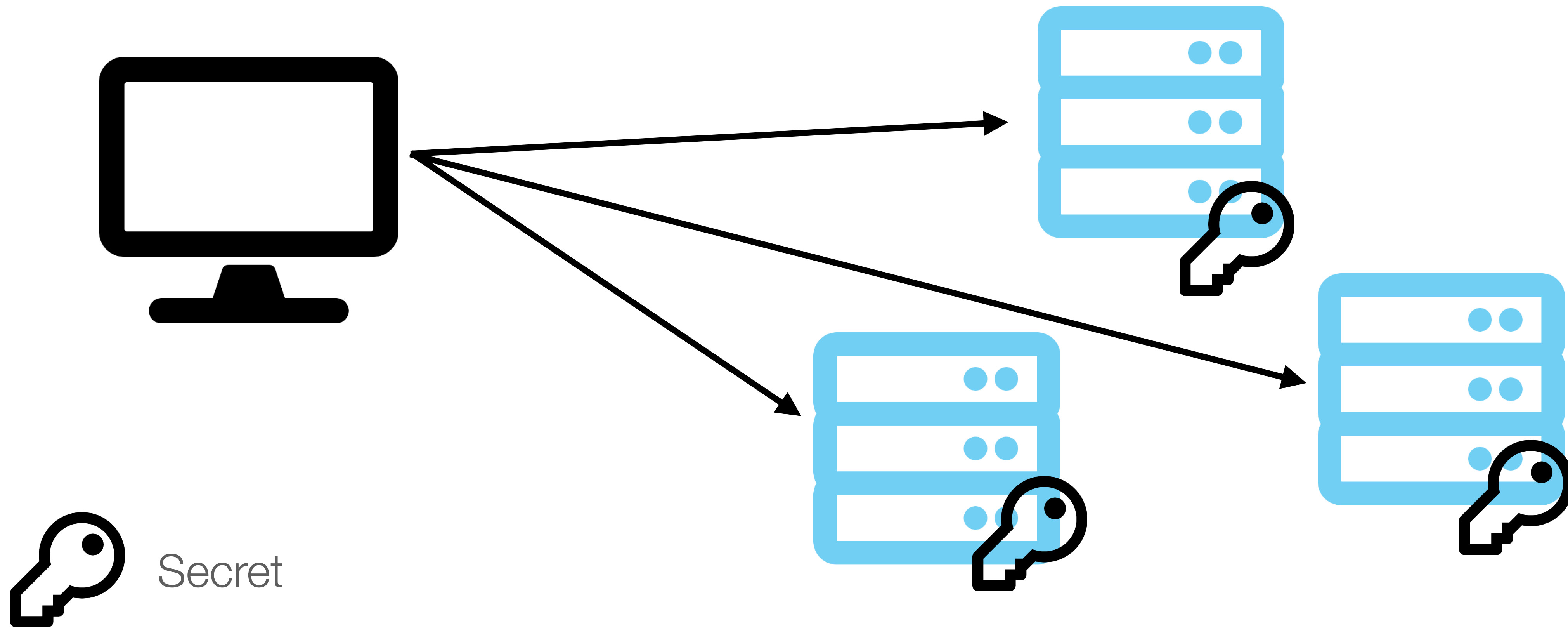


The JWT Format: Algorithms



HMAC: All services need to know the secret

- ✓ HS256
- ✓ HS384
- ✓ HS512

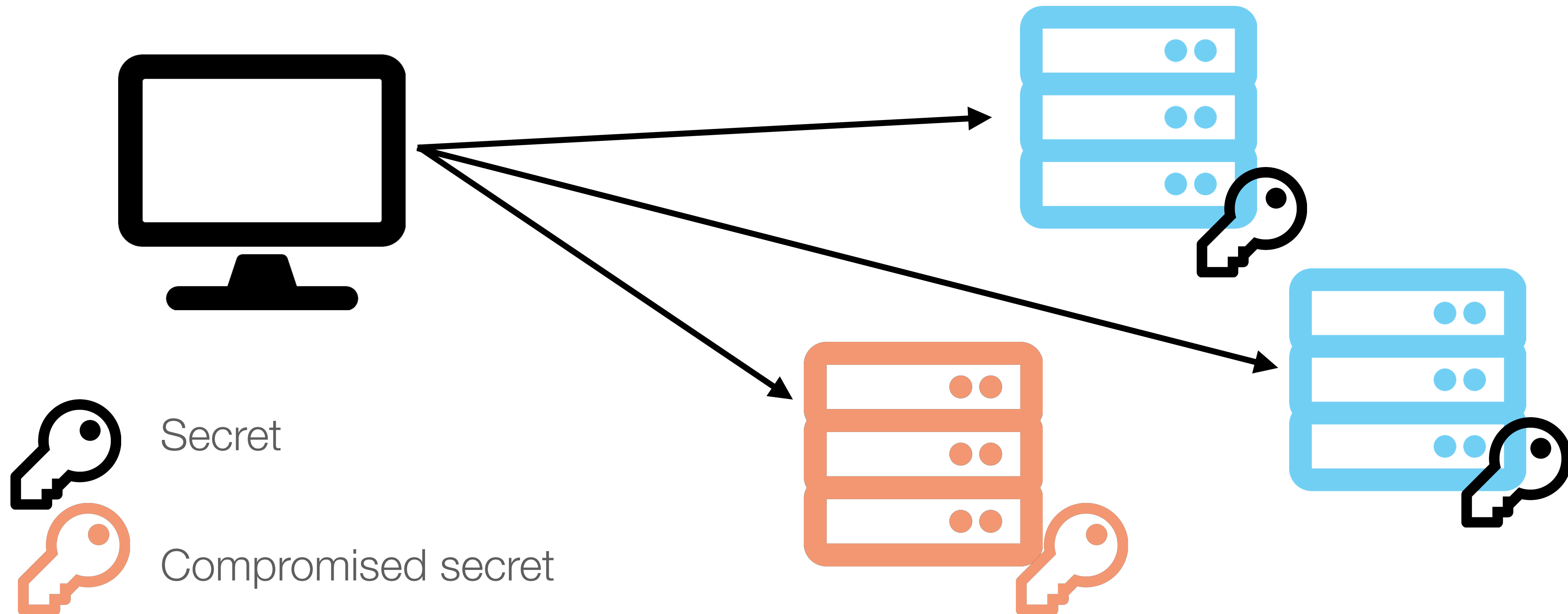


The JWT Format: Algorithms



HMAC: if one service gets compromised

- ✓ HS256
- ✓ HS384
- ✓ HS512



The JWT Format: Algorithms

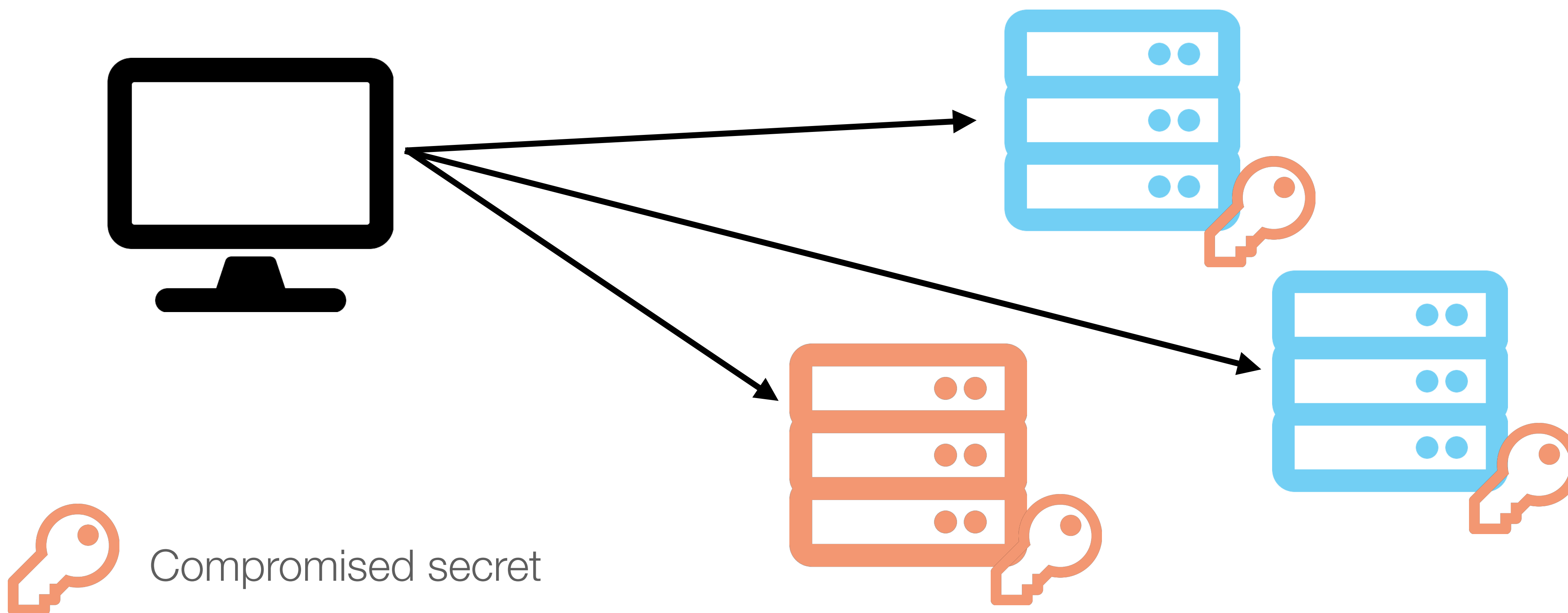


HMAC: the secret is compromised for all services

✓ HS256

✓ HS384

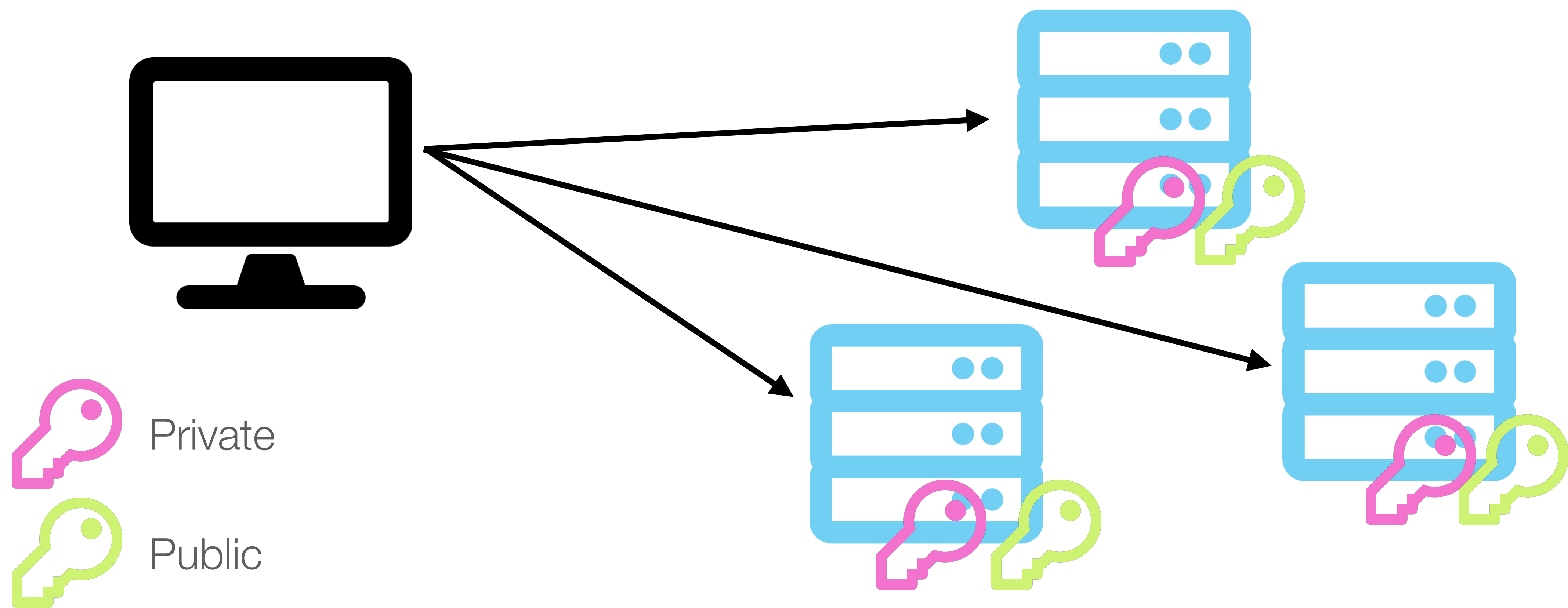
✓ HS512



The JWT Format: Asymmetric



Asymmetric: sharing the key

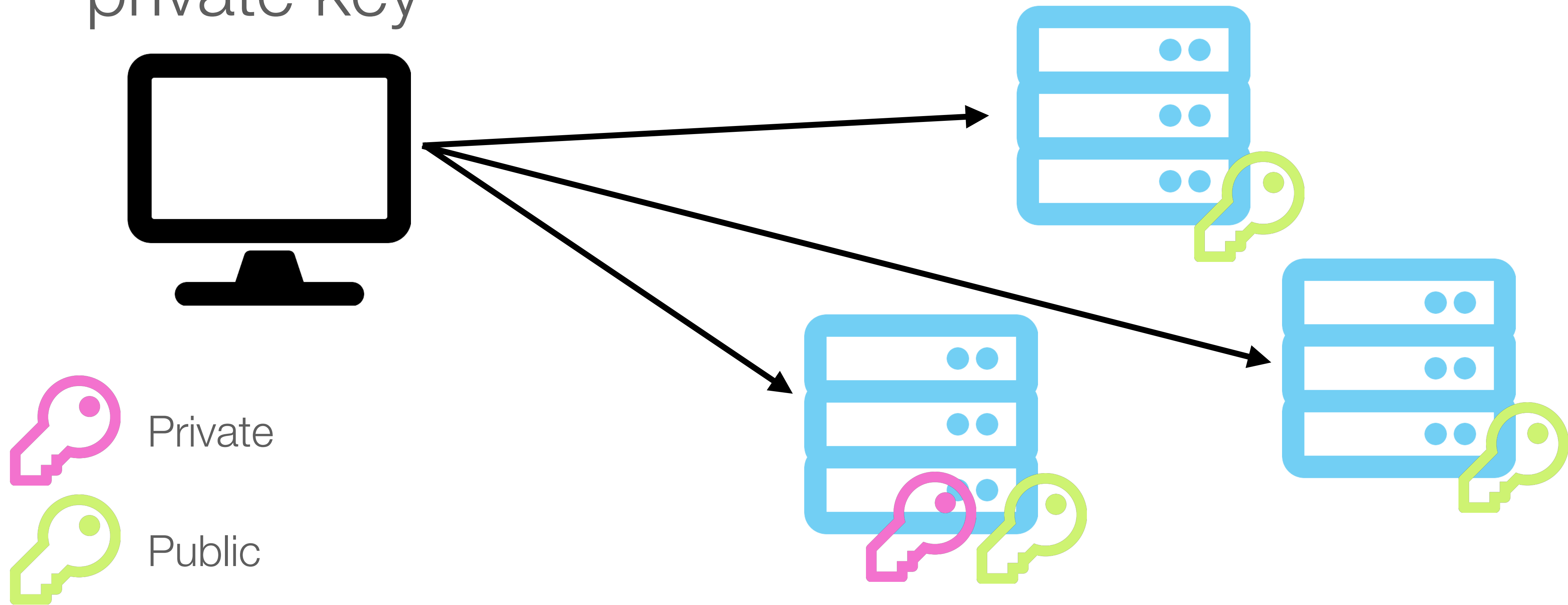


- ✓ RS256
- ✓ RS384
- ✓ RS512
- ✓ ES256
- ✓ ES384
- ✓ ES512
- ✓ PS256
- ✓ PS384
- ✓ PS512

The JWT Format: Asymmetric



Asymmetric: Only trusted services get the private key



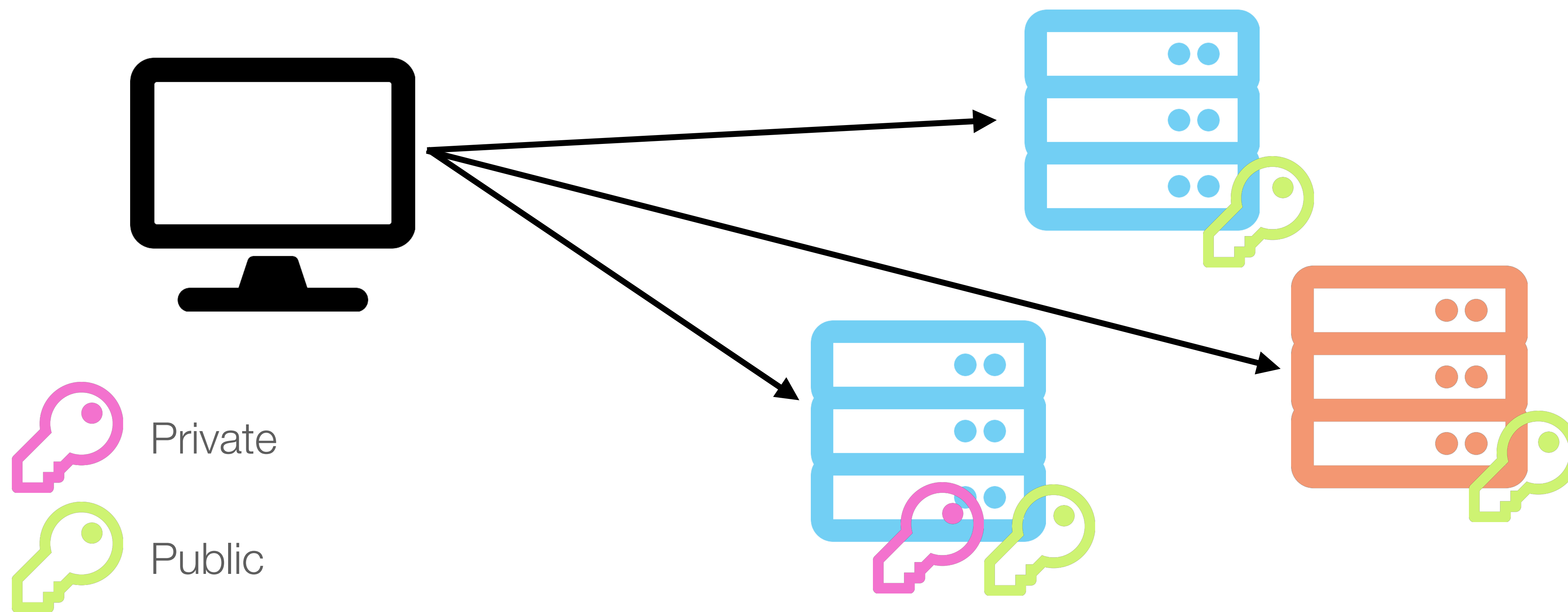
- ✓ RS256
- ✓ RS384
- ✓ RS512
- ✓ ES256
- ✓ ES384
- ✓ ES512
- ✓ PS256
- ✓ PS384
- ✓ PS512



The JWT Format: Asymmetric



Asymmetric: If one service gets compromised...

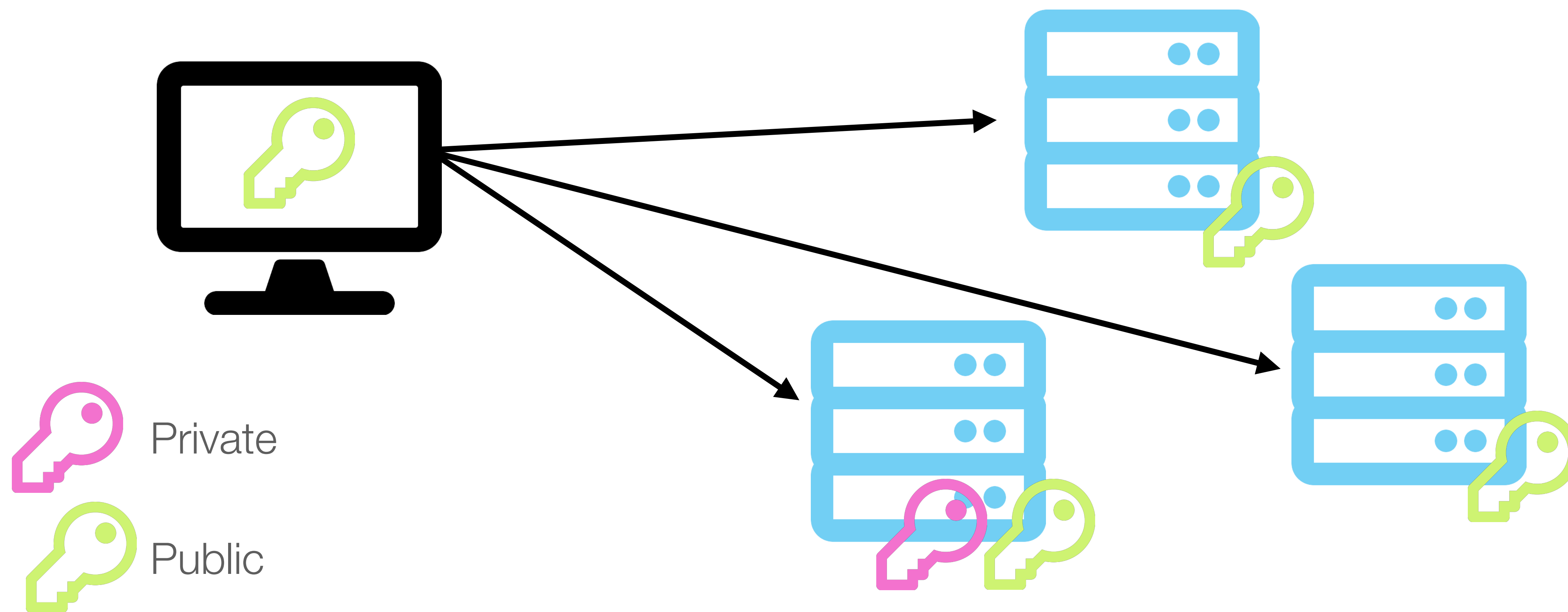


- ✓ RS256
- ✓ RS384
- ✓ RS512
- ✓ ES256
- ✓ ES384
- ✓ ES512
- ✓ PS256
- ✓ PS384
- ✓ PS512

The JWT Format: Asymmetric



Asymmetric: Even in the browser!

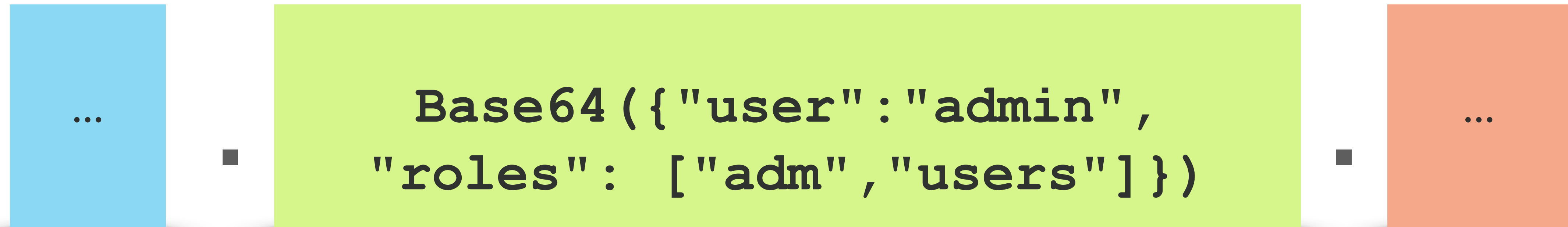


- ✓ RS256
- ✓ RS384
- ✓ RS512
- ✓ ES256
- ✓ ES384
- ✓ ES512
- ✓ PS256
- ✓ PS384
- ✓ PS512

The JWT Format: payload



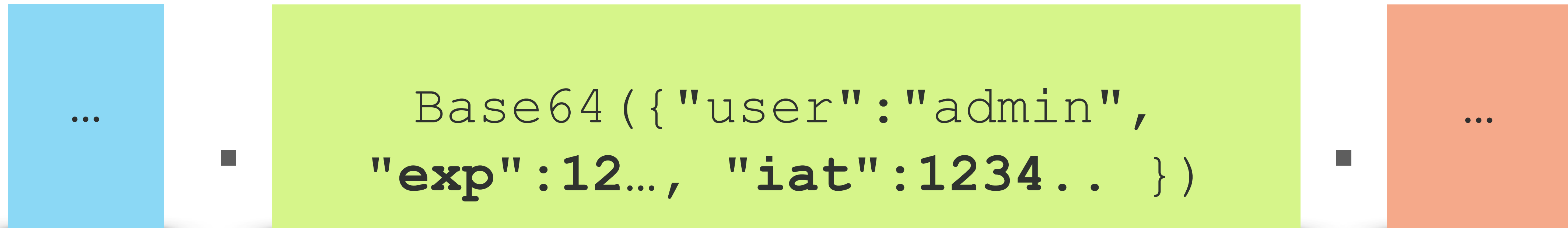
The payload may contain literally anything:



The JWT Format: payload



The payload **may** contain registered claims:





The payload **may** contain registered claims:

- “iss”: issuer
- “sub”: subject
- “aud”: audience
- “jti”: claim id
- “exp”: expiration time
- “nbf”: not before
- “iat”: issued at*

* useful for async processing



The JWT Format: creating a token



- Create the JSON header and base64 encode it
- Create the JSON payload and base64 encode it
- Concatenate with a dot the (encoded) header and payload
- **Sign the result (header+payload)**
- Base64 encode the signature
- Append a dot then the signature



The JWT Format: verifying a token



- Split the token in three parts based on the dots
- Base64 decode each part
- Parse the JSON for the header and payload
- Retrieve the algorithm from the header
- Verify the signature based on the algorithm
- Verify the claims





- Multiple systems can issue tokens
- A token can be used by multiple systems
- All these systems can use different libraries





When attacking JWT, your main goal is to bypass the signature mechanism



Not checking the signature



Not checking the signature



Some libraries provide two methods:

- decode <- don't use this one
- verify

Or just people forgetting to re-enforce the signature check after disabling it for some quick testing



Not checking the signature



Exploitation:

- Get a token
- Decode and tamper with the payload
- Profit



None algorithm



The None algorithm



Remember that slide?



None



RS256



ES256



PS256

Basically, don't sign the token

Used to be supported by default in few libraries



The None algorithm



Exploitation:

- Get a token
- Decode the header and change the algorithm to “None” (or “none”)
- Decode and tamper with the payload
- Keep or remove the signature
- Profit



Trivial Secret





The security of the signature relies on the strength of the secret

The secret can be cracked offline with just one valid token

Cracking is supported by hashcat



hashcat
@hashcat

Follow



Support added to crack JWT (JSON Web Token) with hashcat at 365MH/s on a single GTX1080:



```
$. /hashcat -m 16500 hash.txt -a 3 -w 3 ?a?a?a?a?a hash...
```

pastebin.com

11:06 AM - 21 Jan 2018



Trivial secret



<https://github.com/aichbauer/express-rest-api-boilerplate/blob/master/api/services/auth.service.js>

```
const jwt = require('jsonwebtoken');
```

```
const secret = process.env.NODE_ENV === 'production' ? process.env.JWT_SECRET : 'secret';
```





Exploitation:

- Get a token
- Brute force the secret until you get the same signature
- Tamper with the payload
- Re-sign the token using the secret



Algorithm confusion



Algorithm confusion



The sender controls the algorithm used

You can tell the receiver that the token has been signed using HMAC instead of RSA for example

With RSA, you sign with the private key and verify with the public key

With HMAC, you sign and verify with the same key

If you tell the receiver it's an HMAC and it verifies it with the public key (thinking it's RSA?)



Algorithm confusion



With RSA, you sign with the private key and verify with the public key

With HMAC, you sign and verify with the same key

You tell the receiver it's an HMAC (instead of RSA) and it verifies the signature using HMAC with the public key as the secret (thinking it's RSA):

You can sign the token with the public key





How to get the public key:

- Public key accessible in the javascript code
- Public key available in a mobile client
- Public key just available in the documentation.



Algorithm confusion



Exploitation:

- Get a token signed with RSA (you only have access to the public key)
- Decode the header and change the algorithm from RSA “RS256” to HMAC “HS256”
- Tamper with the payload
- Sign the token with the public RSA key



kid injection



Kid parameter



The **header** can contain a kid parameter:

- Key id (<https://tools.ietf.org/html/rfc7515#section-4.1.4>)
- Often used to retrieve a key from:
 - ✱ The filesystem
 - ✱ A Database

This is done prior to the verification of the signature

If the parameter is injectable, you can bypass the signature





Exploitation:

- Get a signed token containing a kid parameter
- Decode the header and change the kid with a SQL injection payload
- Tamper with the payload
- Sign the token using the return value from the SQL injection



CVE-2018-0114





JWS allows you to add a “jwk” attribute (JSON Web Key) to the header to tell the receiver what key was used to sign the token:

```
"jwk":  
  { "kty": "RSA",  
    "kid": "pentesterlab",  
    "use": "sig",  
    "n": "0wPQEuB9GAsYIFFQy54BnbpmI4oIenLYJYmLyVL8v9lDbT3NVIKWK4g4vOSOV2DoS3qVeedFOjx1CZnyr5k47D4gItocoESnJDMbZts8VyswOMMa4bX8zxhciQ4oCdpmpNZPlAq2nhkm7j60Jjj743vwm8GS7Tj0zAv0mCXl6wjmUIUZoHngWDLGnnvZBkiPBNq_W-RXqT4Om_4s_AEYwg-bRaHp64TwR66i8cVkpqngGx7CofvMoQ0p0QmnQYwl8CNHP3d7nd12BuyIFy_dcIe3b-0PIFkCn0YPqdq7jAKdvUBUGrKeyqd99-eu6LeQ1W3je07Fdbne-UPkEXssgDQ",  
    "e": "AQAB"  
  }
```



- Vulnerability in Cisco Node Jose
 - Node-Jose uses the embedded “jwk” key to check the signature
-
- Integrity bypass!





Exploitation:

- Get a token
- Decode and tamper with the payload
- Generate a RSA key
- Add "n" & "e" to the header and use RS256
- Sign the token with your RSA key



Conclusion



Recommendations



- ✓ Use strong keys and secrets
- ✓ Don't store them in your source code
- ✓ Make sure you have key rotation built-in



Recommendations



- ✓ Review the libraries you pick (KISS library)
- ✓ Make sure you check the signature
- ✓ Make sure your tokens expire
- ✓ Enforce the algorithm





- JWT are complex and kind of insecure by design
(make sure you check <https://github.com/paragonie/paseto>)
- JWT libraries introduce very interesting bugs
- Make sure you test for those if you write code, pentest or do bug bounties

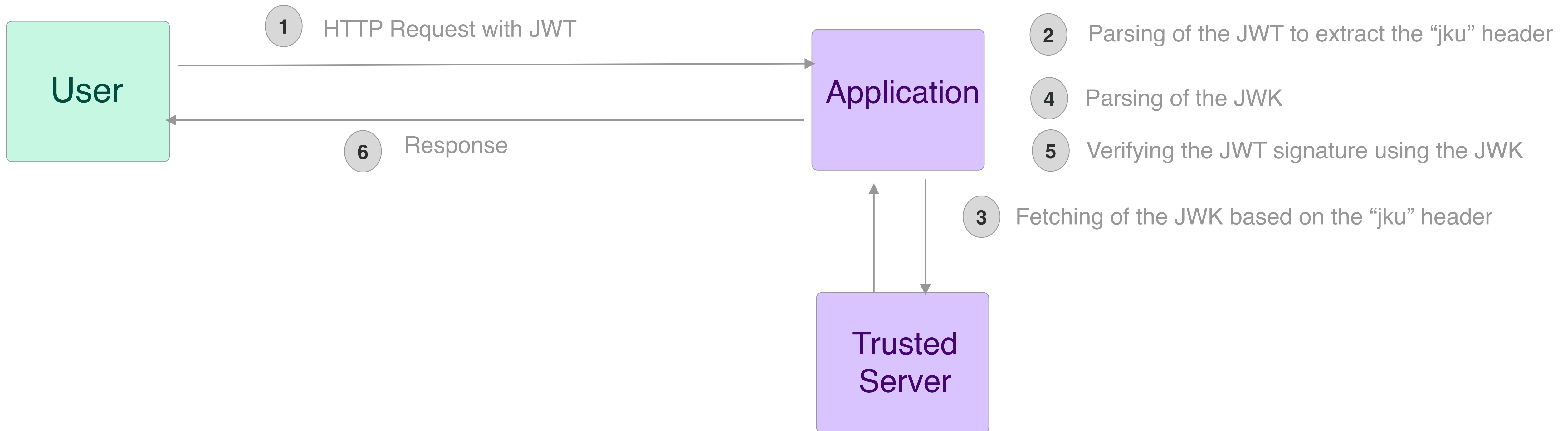




- If you read some of the JWS RFC, you probably learnt about jku and x5u parameter for the headers
- People are starting to use jku (JWK URL)



Back to the future





- Turns out filtering URL is incredibly hard (for people doing it)
 - ➡ Server Side Request Forgery ?
 - ➡ Authentication bypass
 - ➡ Chaining bugs together (Openredirect, header injection...)





PentesterLab

THANKS
FOR YOUR TIME !

Any questions?

louis@pentesterlab.com / PentesterLab.com / @PentesterLab

