Timeless Timing Attacks

by
Tom Van Goethem & Mathy Vanhoef

DistriNet
KU LEUVEN
NYU | ABU DHABI

Hello!

Tom Van Goethem

Researcher at DistriNet -
KU Leuven, Belgium

Fanatic web & network
security enthousiast

Exploiter of side-channel attacks in
browsers & the Web platform

Mathy Vanhoef

Postdoctoral Researcher at
NYU Abu Dhabi
Soon: professor at KU Leuven

Interested in Wi-Fi security, software
security and applied crypto

Discovered KRACK attacks against
WPA2, RC4 NOMORE

Timing attacks...

```python
if secret condition:
  do_something()
# continue
```

```python
for el in arr:
  if check_secret_property(el):
    break
```

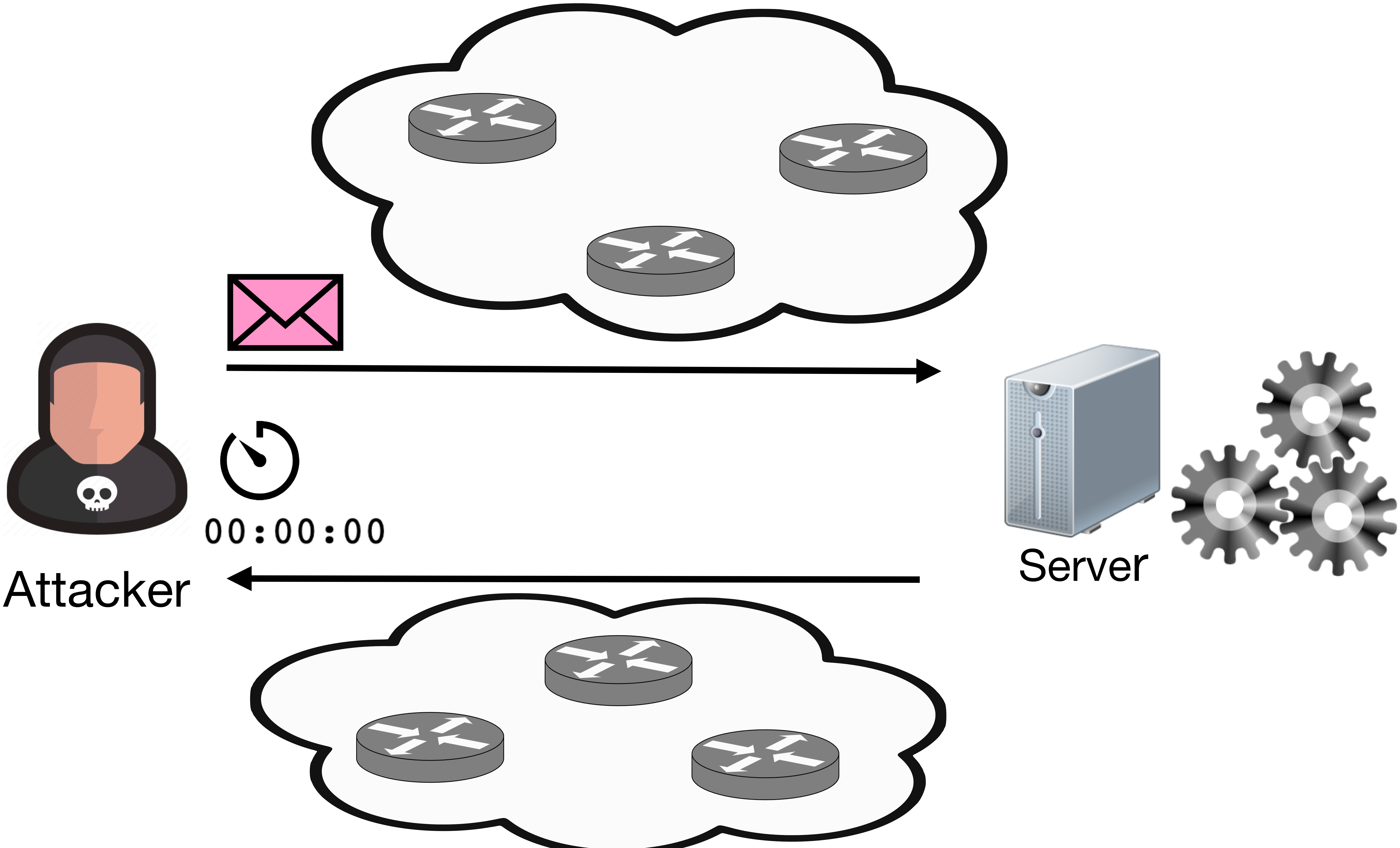
```python
if len(arr_with_secret_elements) > 0:
  do_something()
```

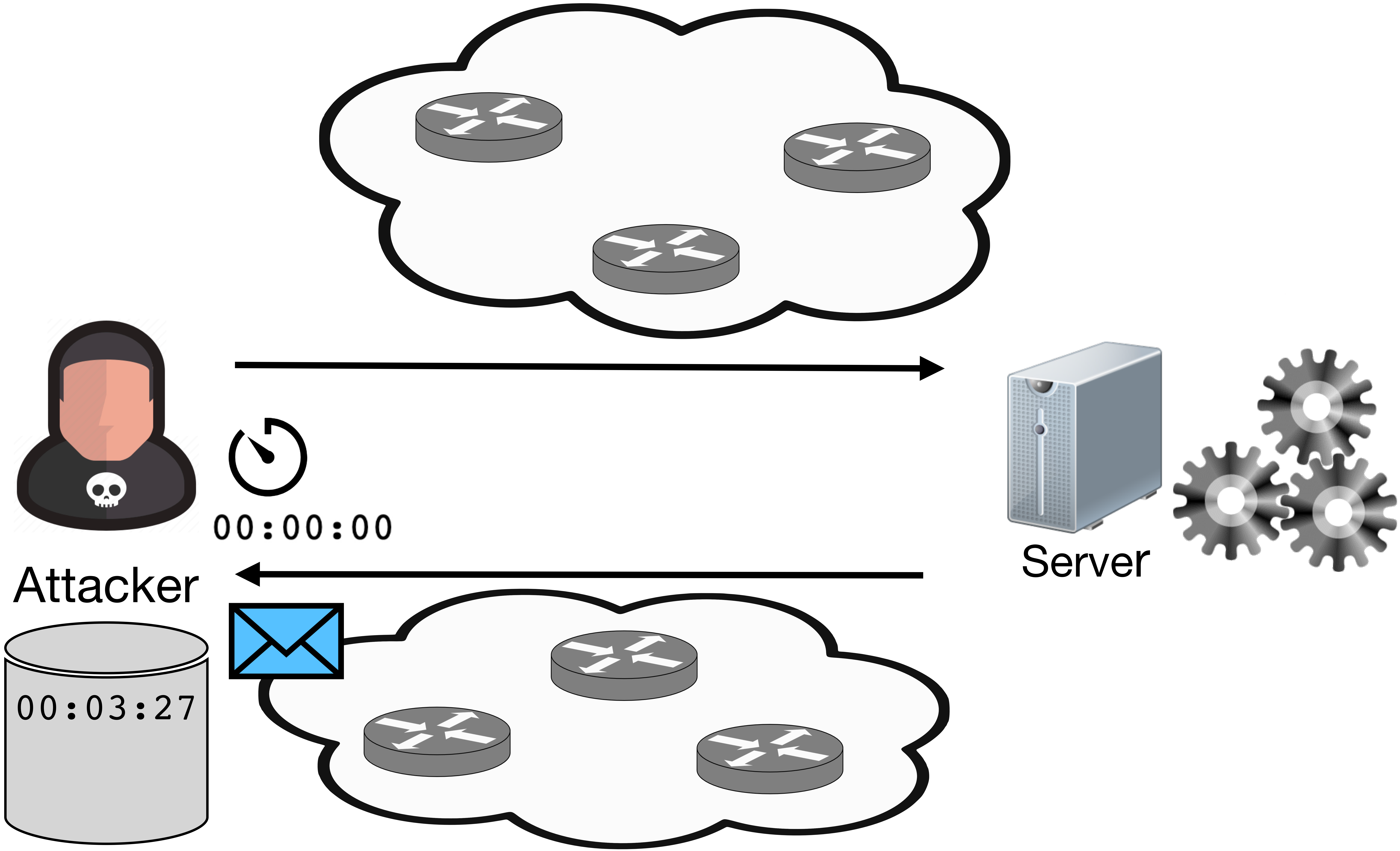
Remote Timing Attacks

- Step 1: attacker connects to target server
- Step 2: attacker sends a (large) number of requests to the server
- Step 3: for each request attacker measures time it takes to receive a response
- Step 4: attacker compares timing of 2 sets of requests (baseline vs target)
- Step 5: using statistical analysis, it is determined which request took longer
- Step 6: SUCCESS?

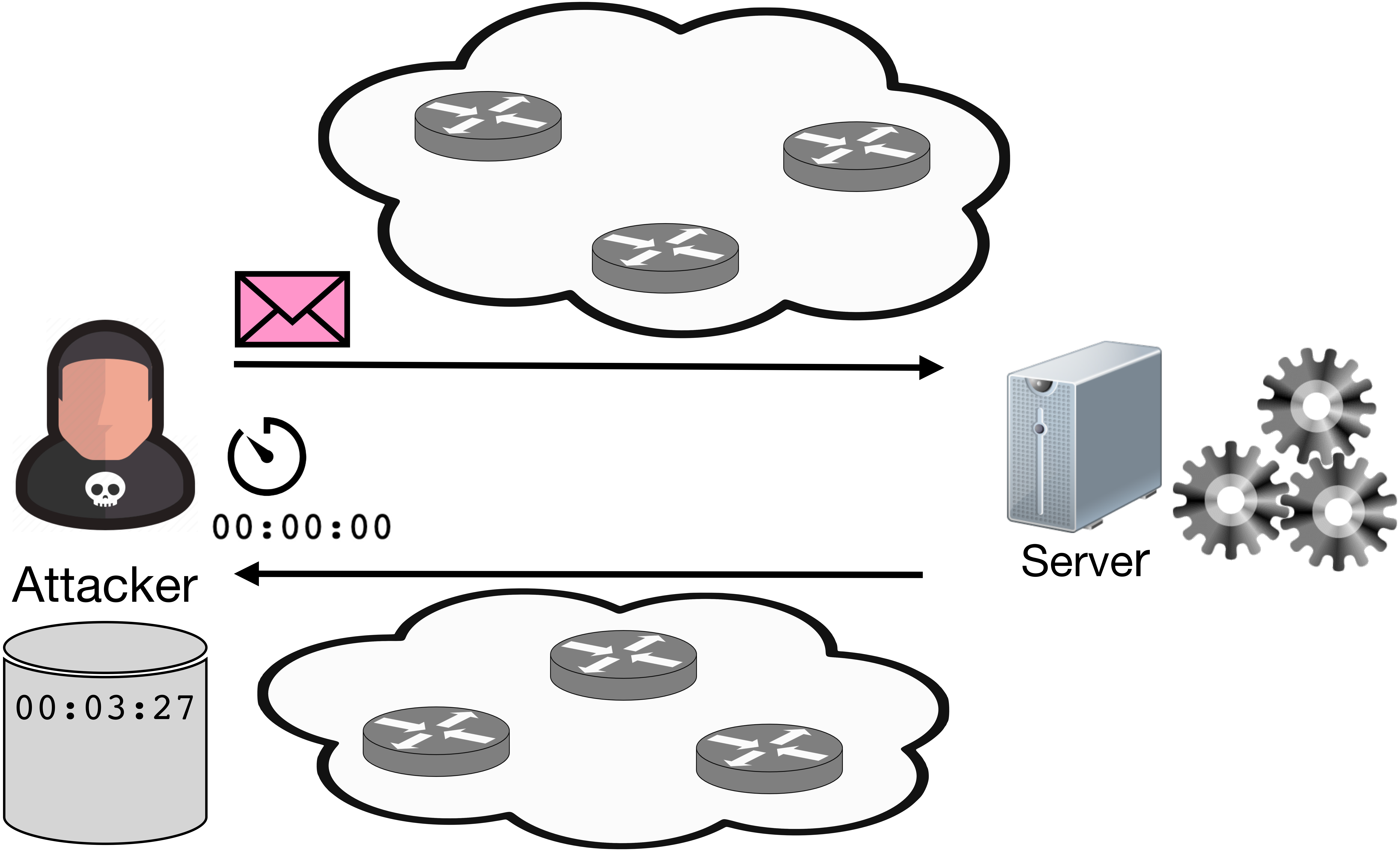
Remote Timing Attacks Success

- Performance of timing attacks is influenced by different aspects:
  - Network connection between attacker and server
    - higher **jitter** → worse performance
    - attacker could try to move closer to target, e.g. same cloud provider
  - Jitter is present on both **upstream and downstream** path
  - **Size of timing leak** determines if attack can be successful
    - Timing difference of 50ms is easier to detect than 5µs
  - Number of **measurements** (more → better performance)

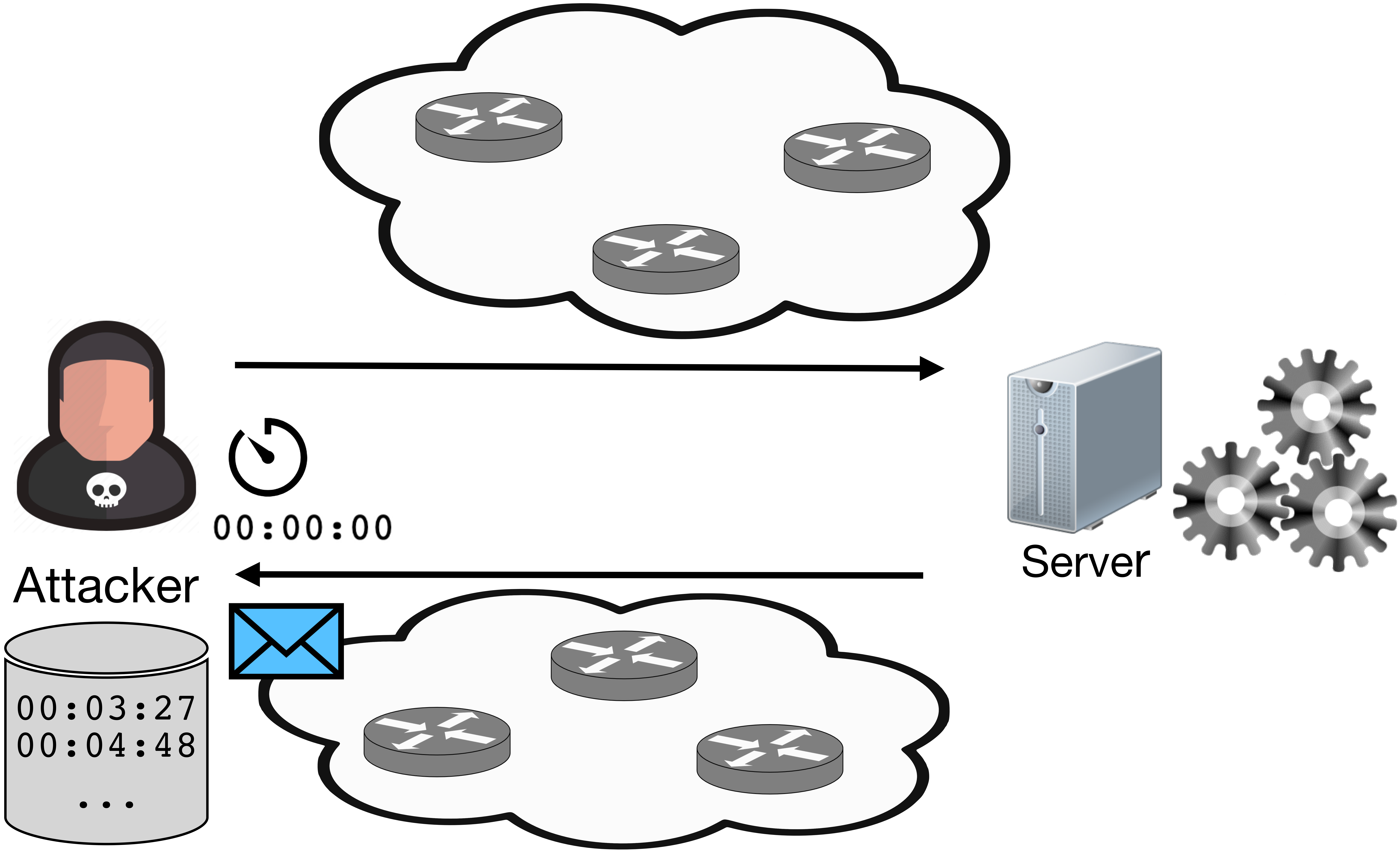

00:00:00
Attacker
Server



00:00:00
Attacker
00:03:27
Server



00:00:00
Attacker
00:03:27
Server


00:00:00
Attacker
Server
00:03:27
00:04:48
...

	EU	US	Asia
50µs	333	4,492	7,386
20µs	2,926	16,820	-
10µs	23,220	-	-
5µs	-	-	-

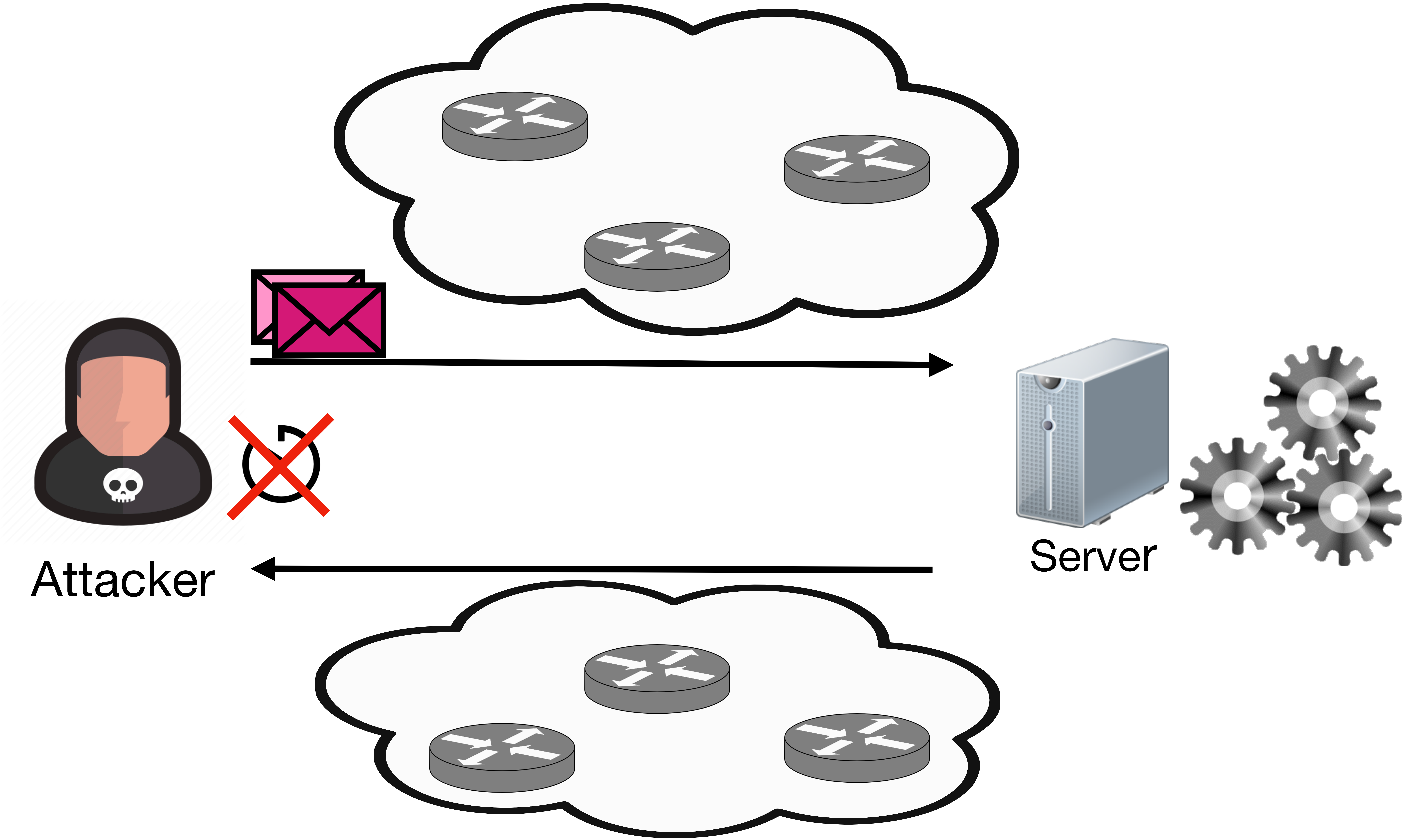
Number of requests required to determine timing difference (5-50µs) with 95% accuracy

based on measurements between university network and AWS
imposed maximum: 100,000

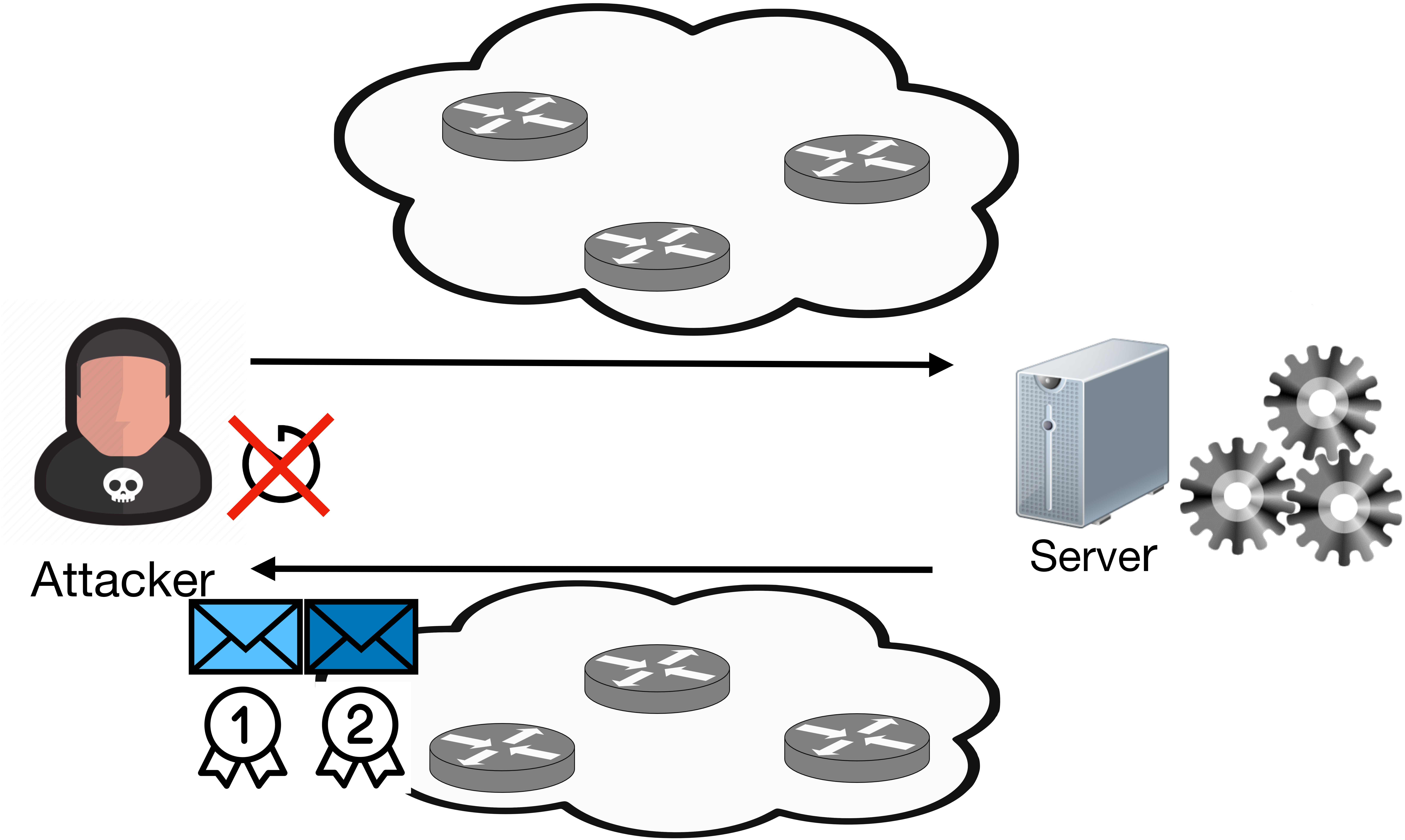
Timeless Timing Attacks

Timeless Timing Attacks

- Absolute response timing is unreliable, as it will always include jitter for every request
- Let’s get rid of the notion of time (hence timeless)
- Instead of relying on sequential timing measurements, we can **exploit concurrency** and only consider response order
  - => no absolute timing measurements!!
- Timeless timing attacks are **unaffected by network jitter**



Attacker
Server



Attacker
Server
1
2

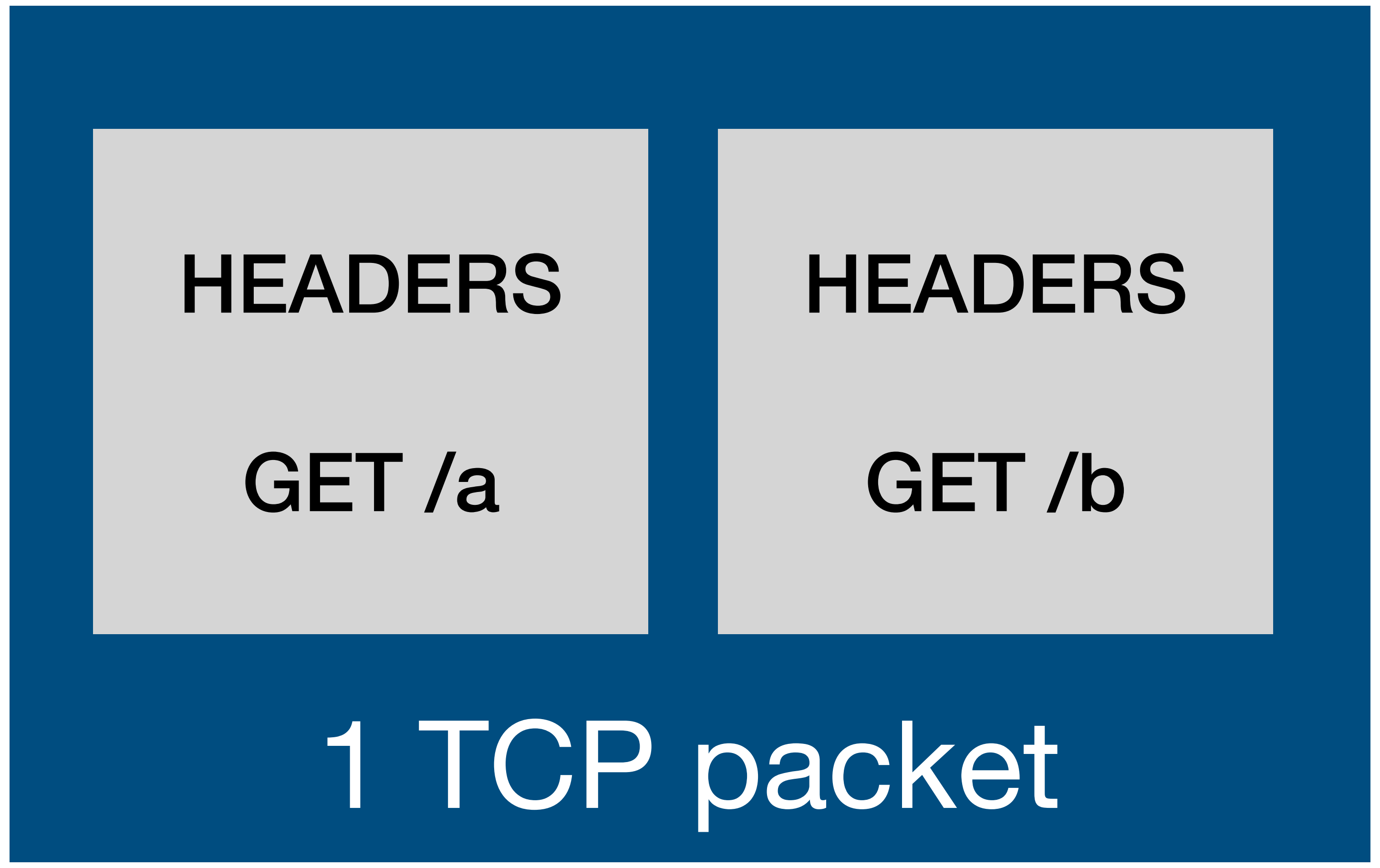
Timeless Timing Attacks: Requirements

1. Requests need to **arrive at the same time** at the server
2. Server needs to process requests **concurrently**
3. **Response order** needs to reflect difference in execution time

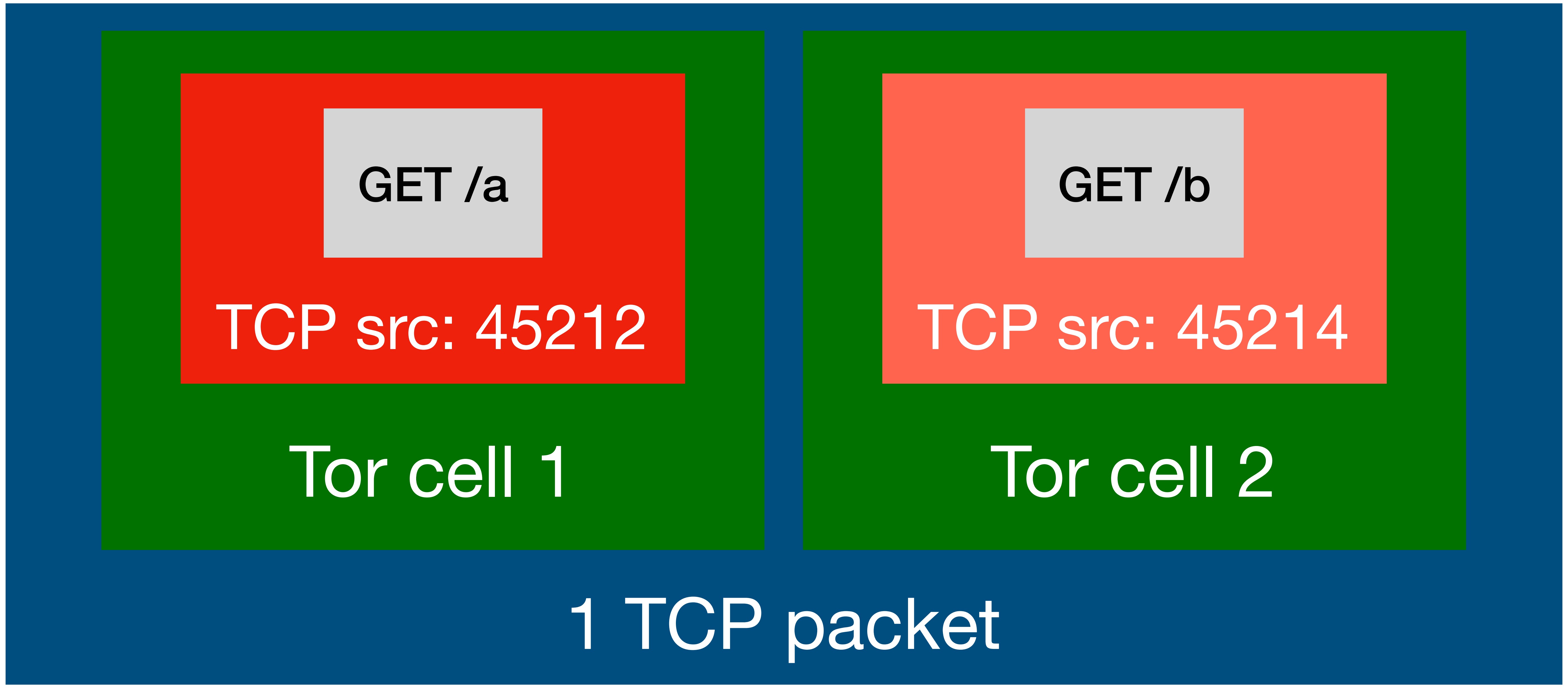
Requirement #1: simultaneous arrival

- Two options: multiplexing or encapsulation
- **Multiplexing**:
    - Needs to be supported by the protocol (e.g. HTTP/2 and HTTP/3 enable multiplexing, HTTP/1.1 does not)
    - A single packet can carry multiple requests that will be processed concurrently
- **Encapsulation:**
    - Another network protocol is responsible for encapsulating multiple streams (e.g. HTTP/1.1 over Tor or VPN)

HTTP/2
(multiplexing)



HTTP/1.1 + Tor
(encapsulation)



Requirement #2: concurrent execution

- Application-dependent; most can be executed in parallel
  possible exception: crypto operations that rely on sequential operations

Requirement #3: response order

- Most operations will generate response immediately after processing
- On TLS connections, response is decrypted in same order as it was encrypted on the server.
  TCP sequence numbers or (relative) TCP timestamps can also be used

How many requests/pairs are needed?

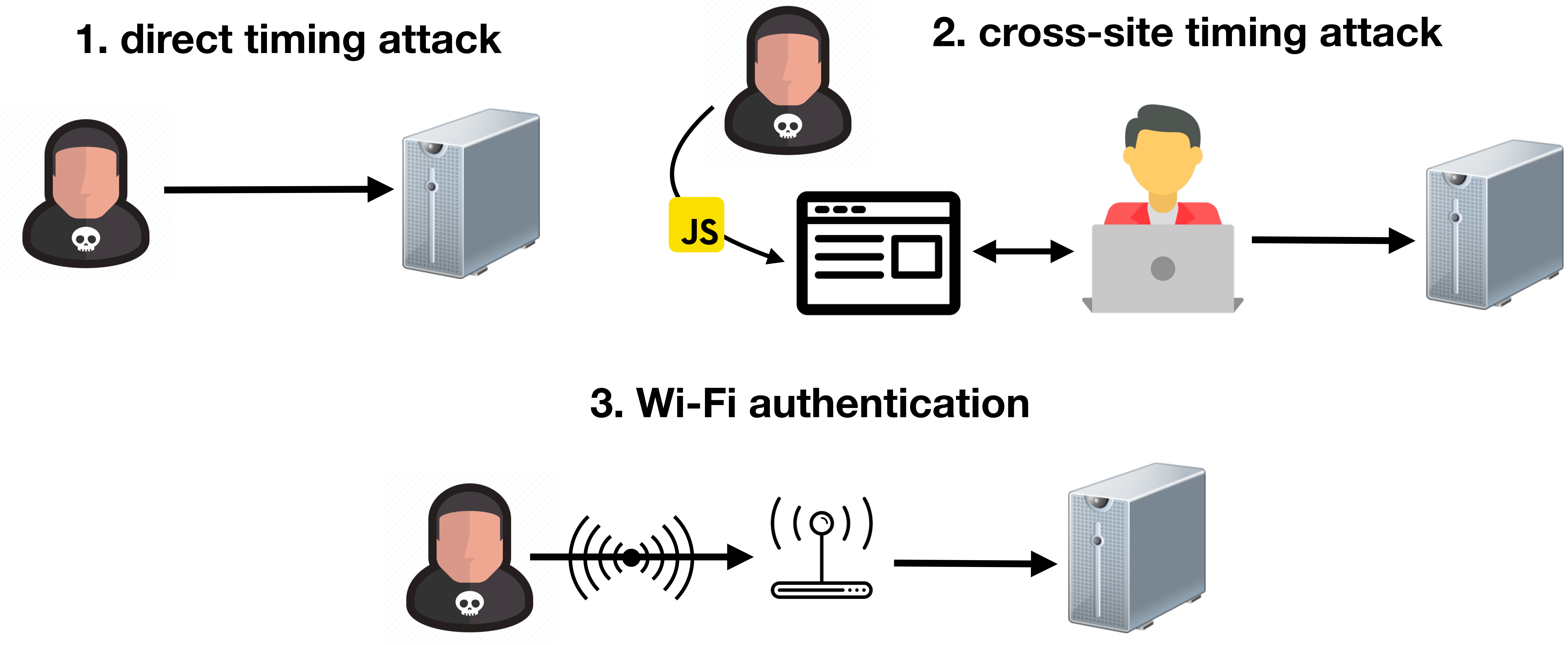
Sequential Timing Attacks

	EU	US	Asia	LAN	localhost
50µs	333	4,492	7,386	20	14
20µs	2,926	16,820	-	41	16
10µs	23,220	-	-	126	20
5µs	-	-	-	498	42
Smallest diff	10µs	20µs	50µs	150ns	150ns

Timeless Timing Attacks

	Internet (anywhere)
50µs	6
20µs	6
10µs	11
5µs	52
Smallest diff	100ns

Attack Scenarios



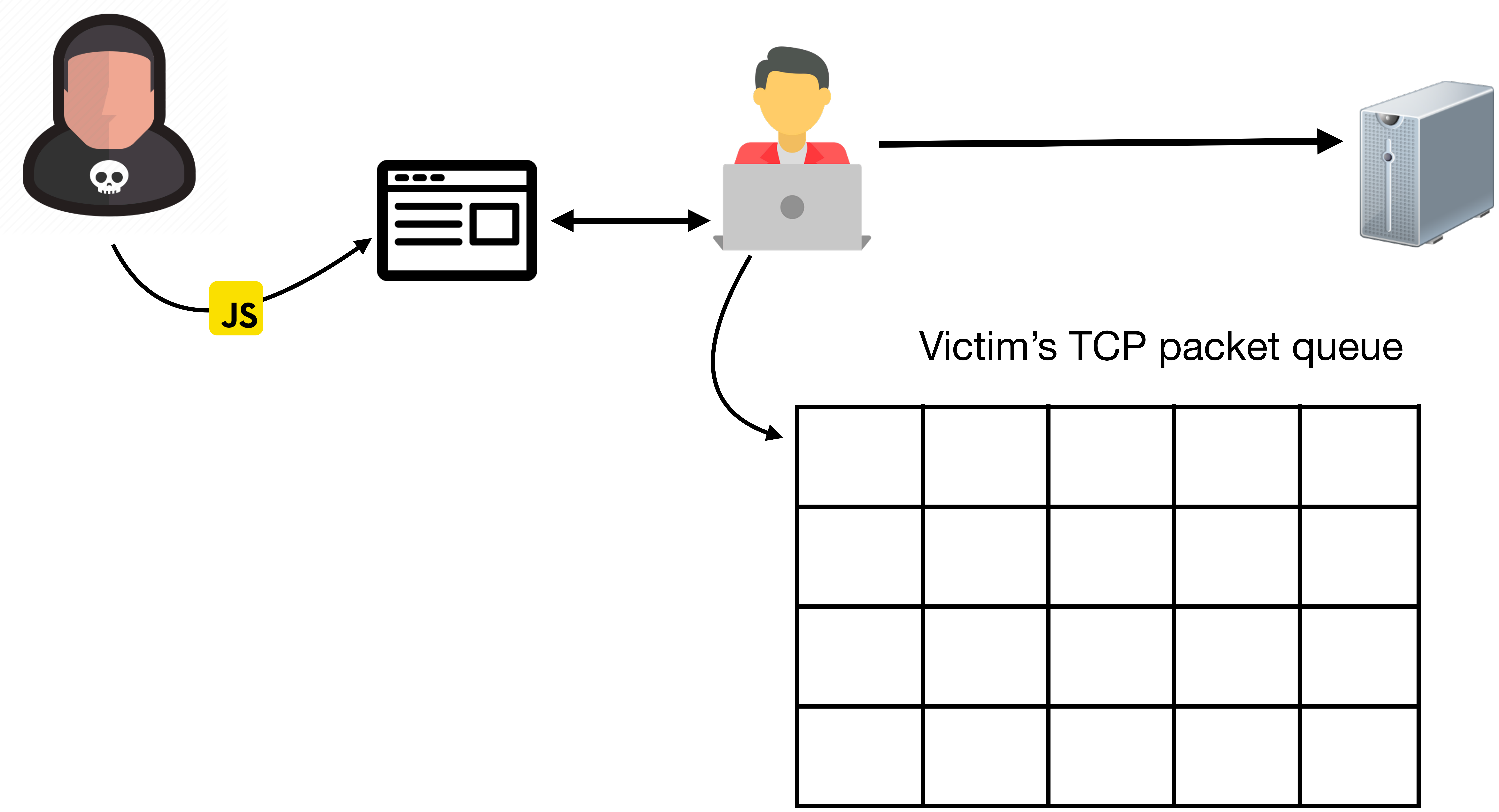
Cross-site Timing Attack

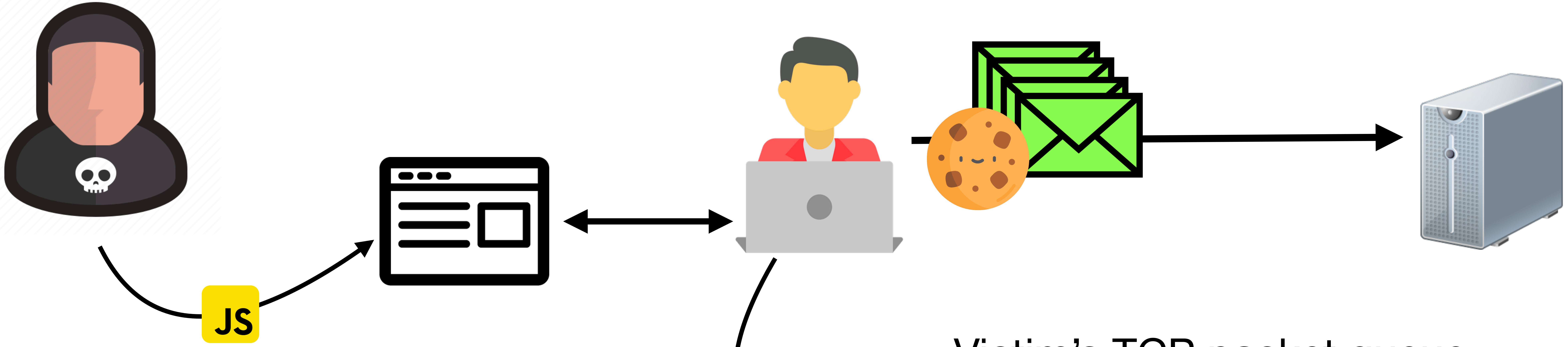
- Victim user lands on malicious website (by clicking a link, malicious advertisement, urgent need to look at cute animal videos, …)
- Attacker launches attack from JavaScript to trigger requests to targeted web server
- Victim's cookies are automatically included in request; request is processed using victim's authentication
- Attacker observes response order (e.g. via `fetch.then()`), and leaks sensitive information that victim shared with website
- Real-world example: abuse search function on HackerOne to leak information about private reports

Cross-site Timeless Timing Attack

- Attacker has no low-level control over network; browser chooses how to send request to kernel
- Need another technique to force 2 requests in single packet
- TCP congestion control to the rescue!!
- Congestion control prevents client from sending all packets at once
  needs ACK from server before sending more
- When following requests are queued, they are merged in single packet 👍

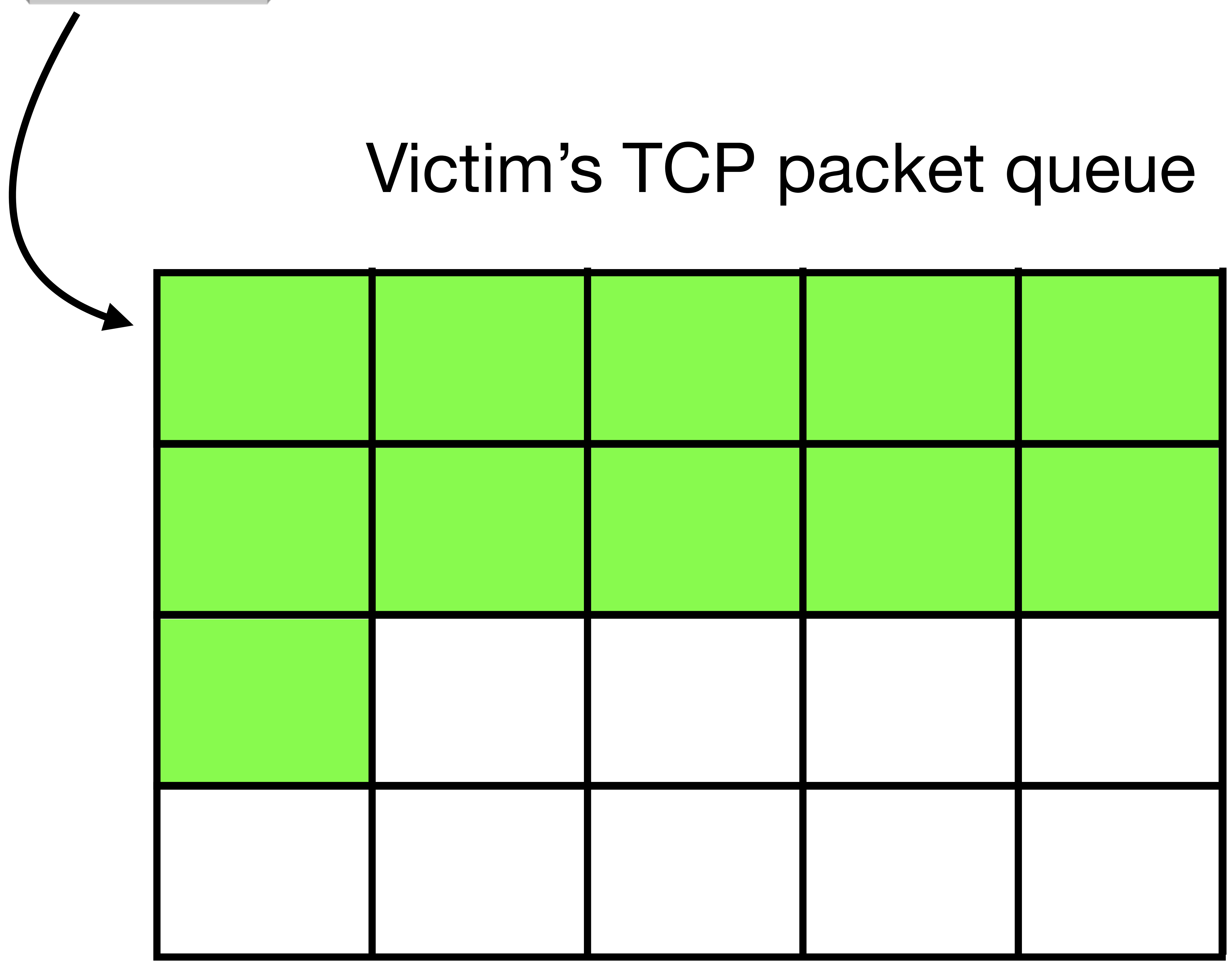
```
fetch(target_bogus_url, {
 "mode": "no-cors",
 "credentials": "include",
 "method": "POST",
 "body": veryLongString
});

fetch(target_baseline_url, {
 "mode": "no-cors",
 "credentials": "include"
});

fetch(target_alt_url, {
 "mode": "no-cors",
 "credentials": "include"
});
```

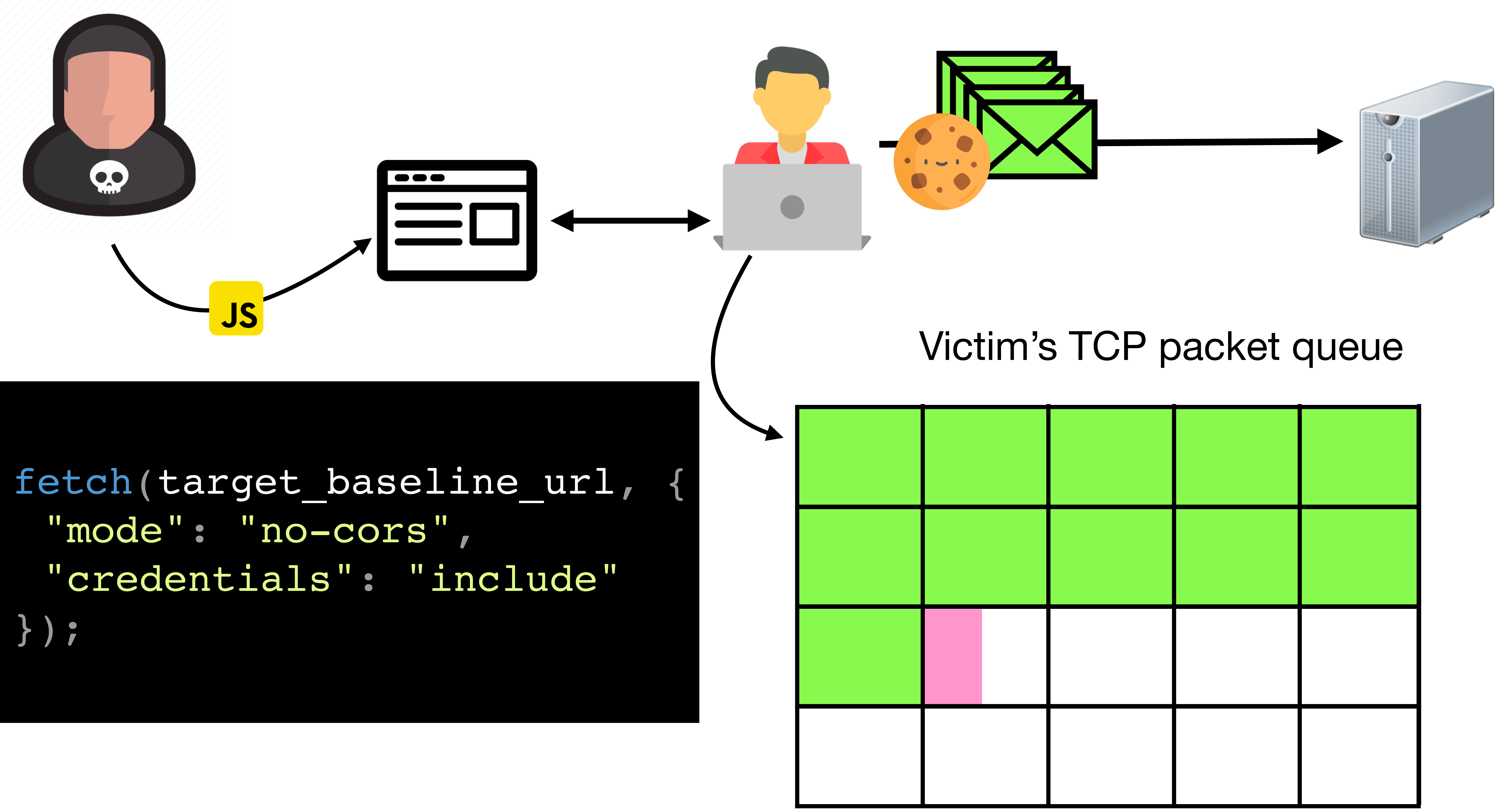

JS
Victim’s TCP packet queue



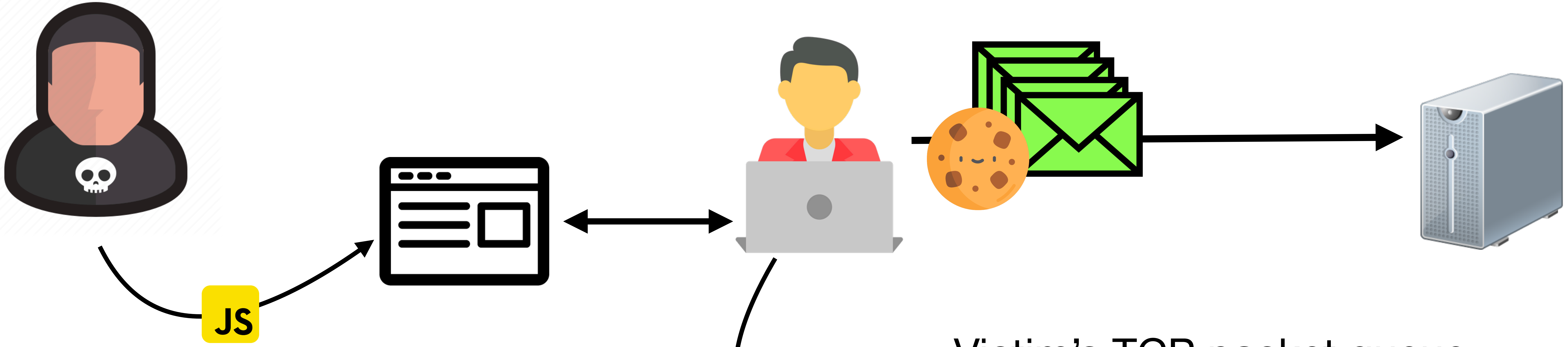

```
fetch(target_bogus_url, {
  "mode": "no-cors",
  "credentials": "include",
  "method": "POST",
  "body": veryLongString
});
```

Victim's TCP packet queue



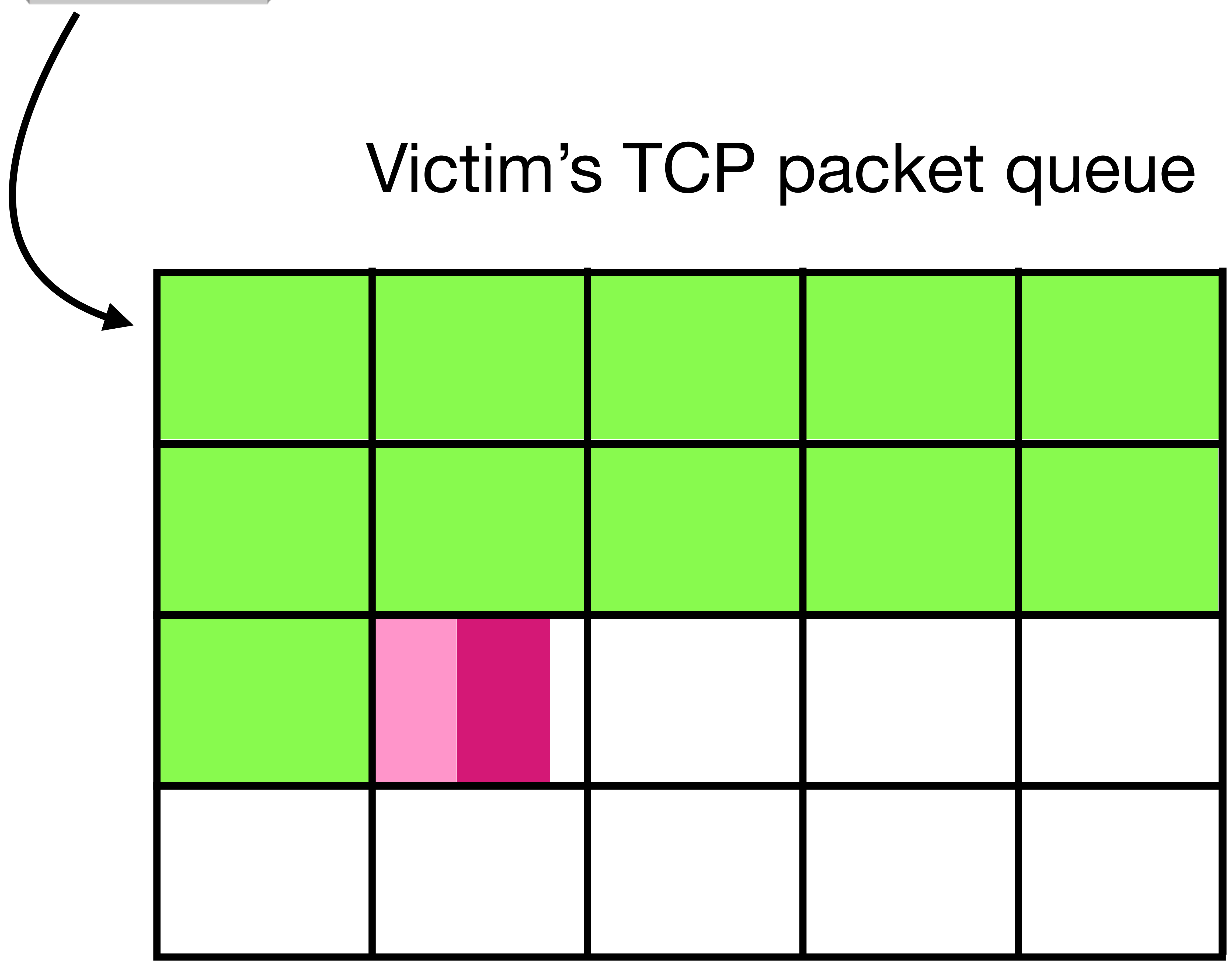


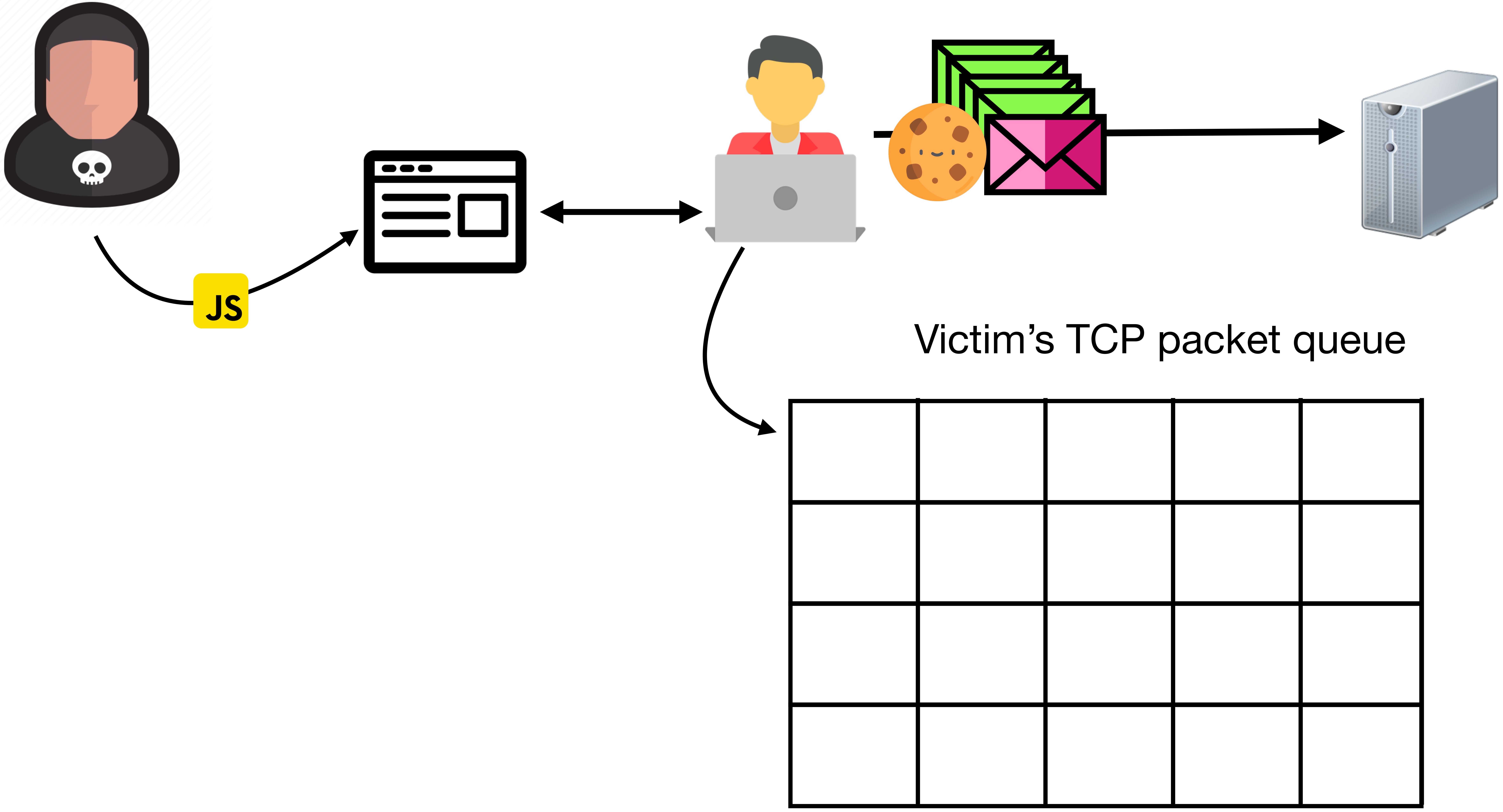

```
fetch(target_baseline_url, {
  "mode": "no-cors",
  "credentials": "include"
});
```




```javascript
fetch(target_alt_url, {
  "mode": "no-cors",
  "credentials": "include"
});
```

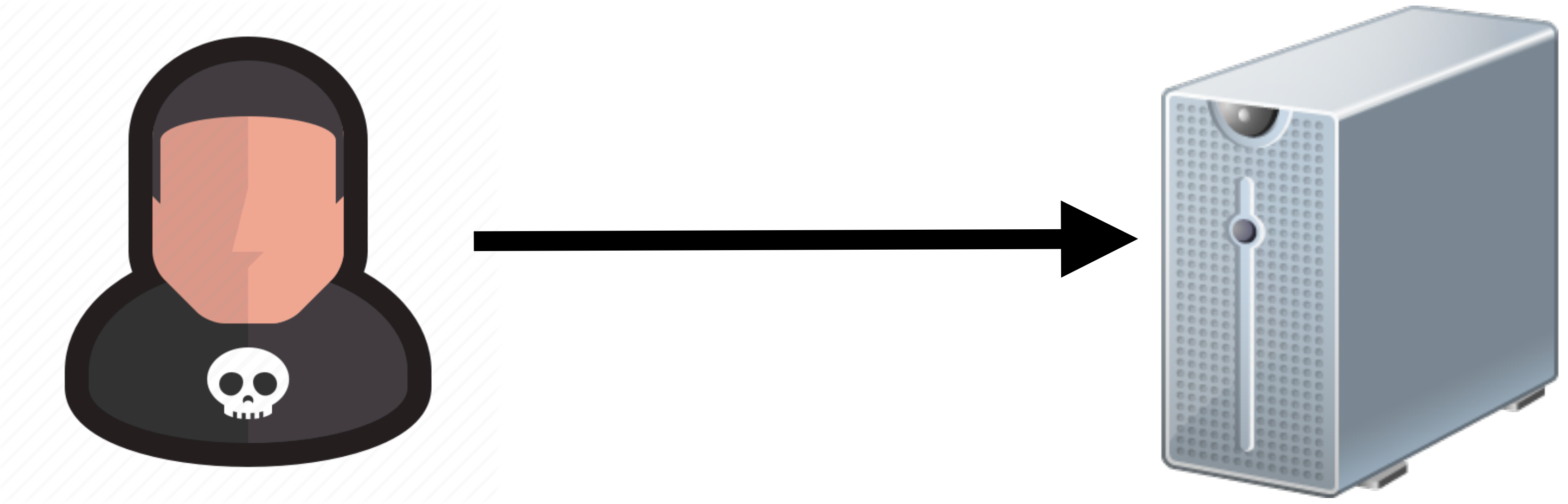
Victim's TCP packet queue



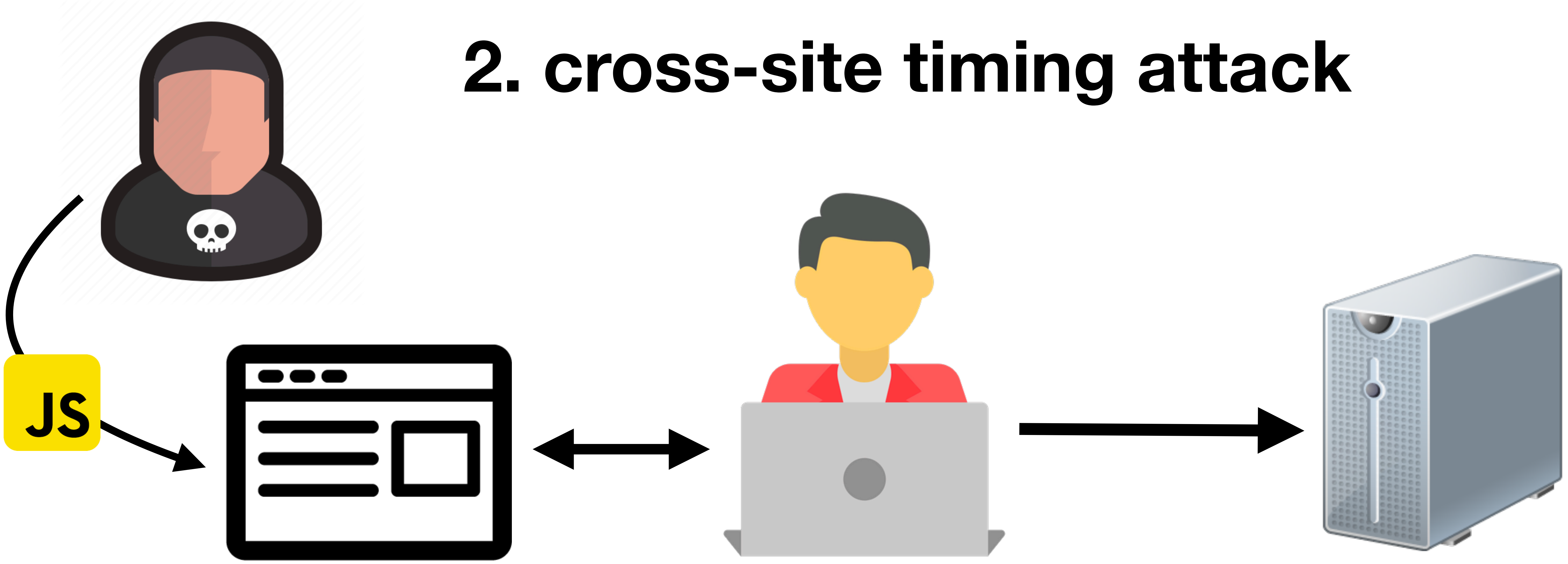

JS
Victim’s TCP packet queue

Attack Scenarios

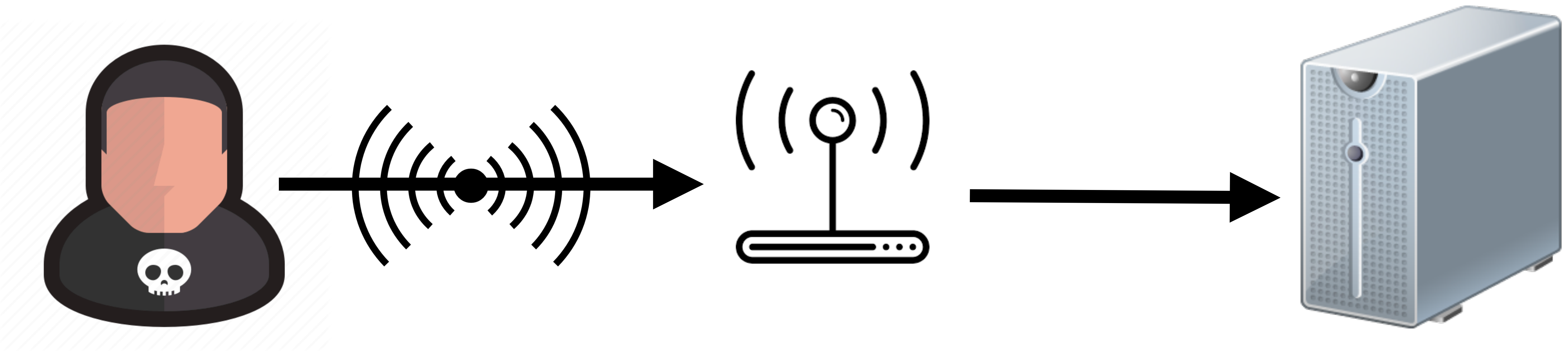
1. direct timing attack



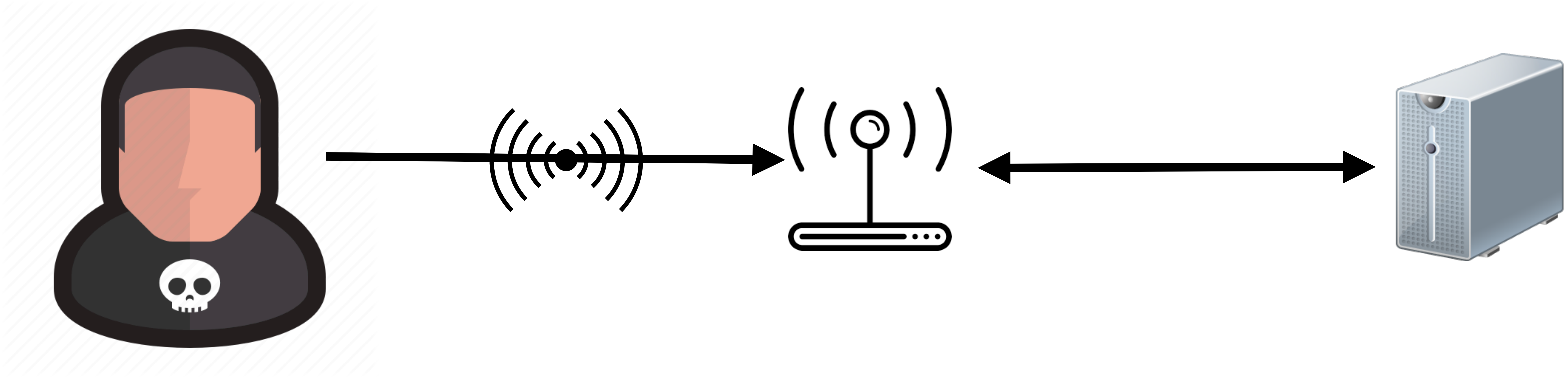
2. cross-site timing attack



3. Wi-Fi authentication

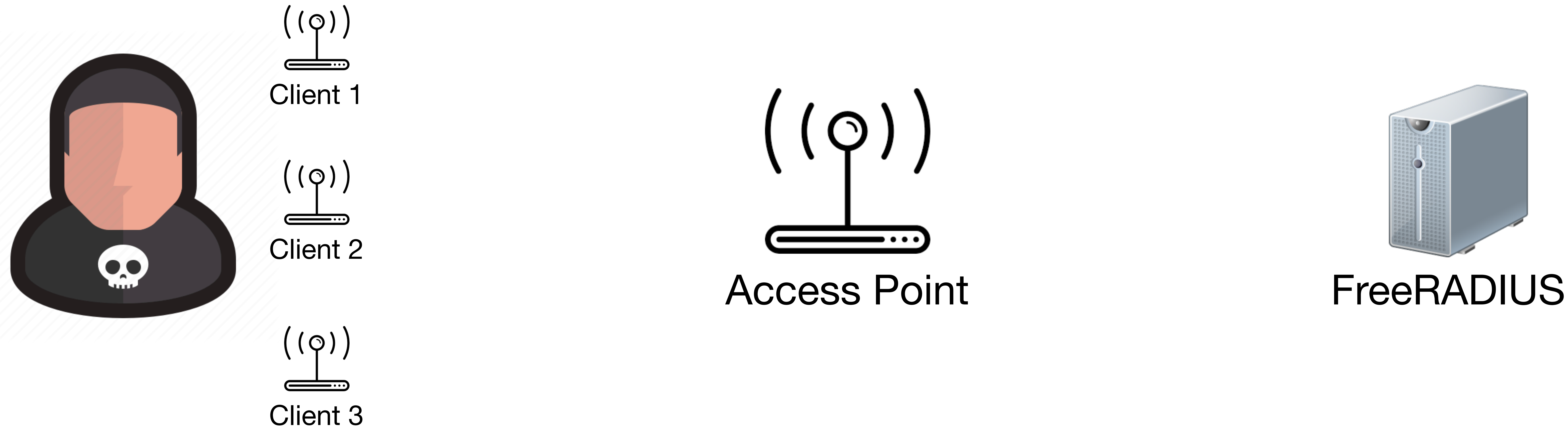


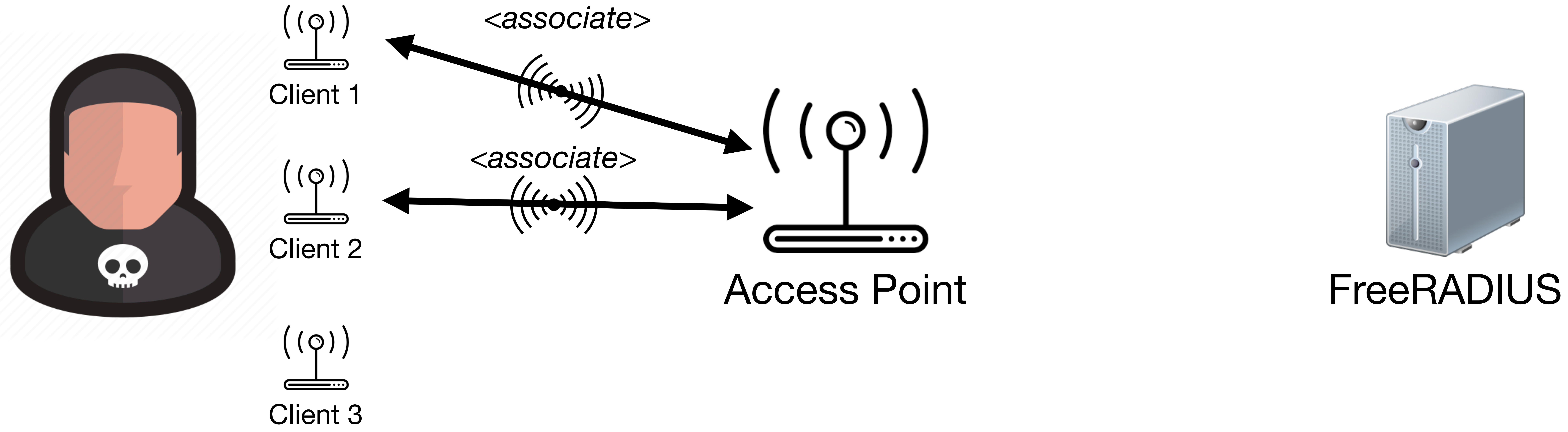
Exploiting Wi-Fi authentication (WPA2 w/ EAP-pwd)



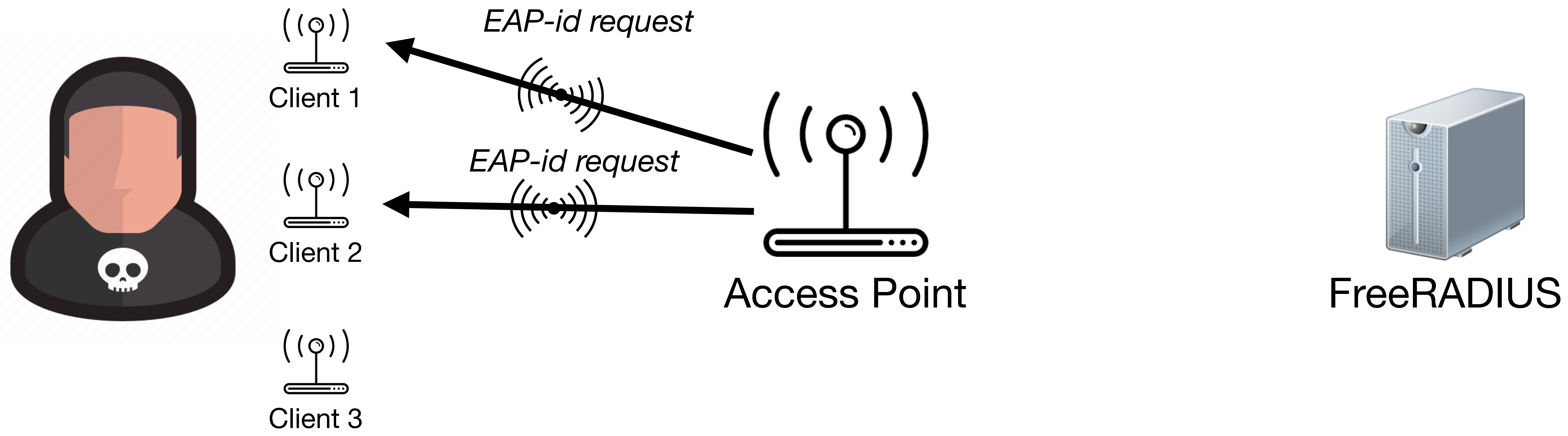
WPA2 & EAP-pwd

- WPA2 is one of the most widely used Wi-Fi protocols
- Authentication can be done using certificates (e.g. EAP-PEAP), or using passwords, relying on EAP-pwd
- Authentication happens between client and authentication server (e.g. FreeRADIUS), access point forwards messages
- Communication between AP and authentication server is typically protected using TLS
- EAP-pwd uses hash-to-curve to verify password
  - A timing leak was found! 😱
  - “Fortunately” small timing difference, so considered not possible to exploit 😁

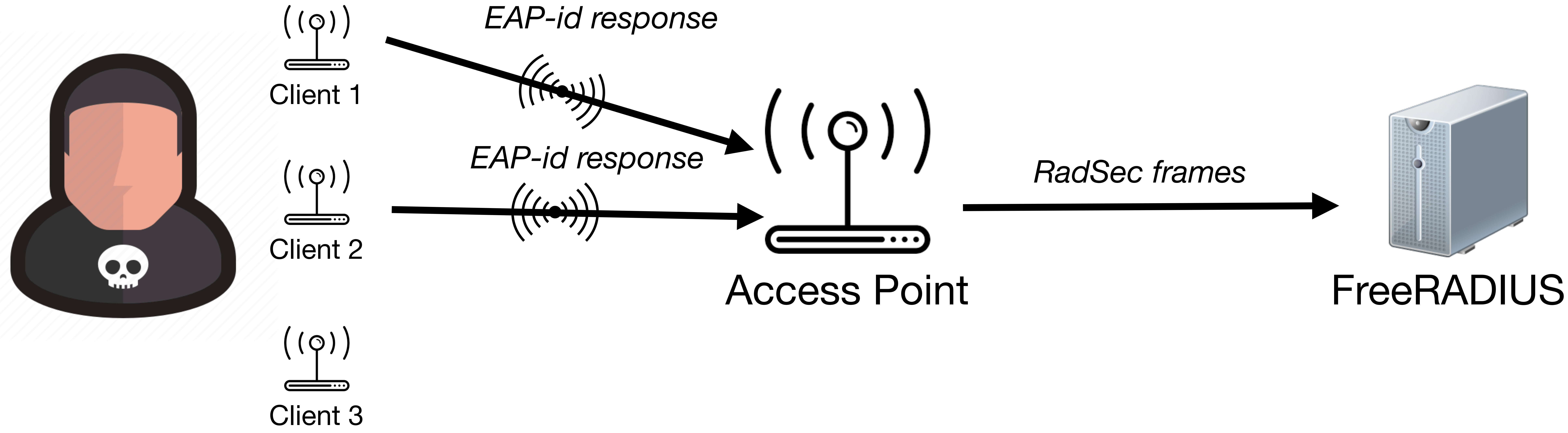

Client 1
Client 2
Client 3
Access Point
FreeRADIUS



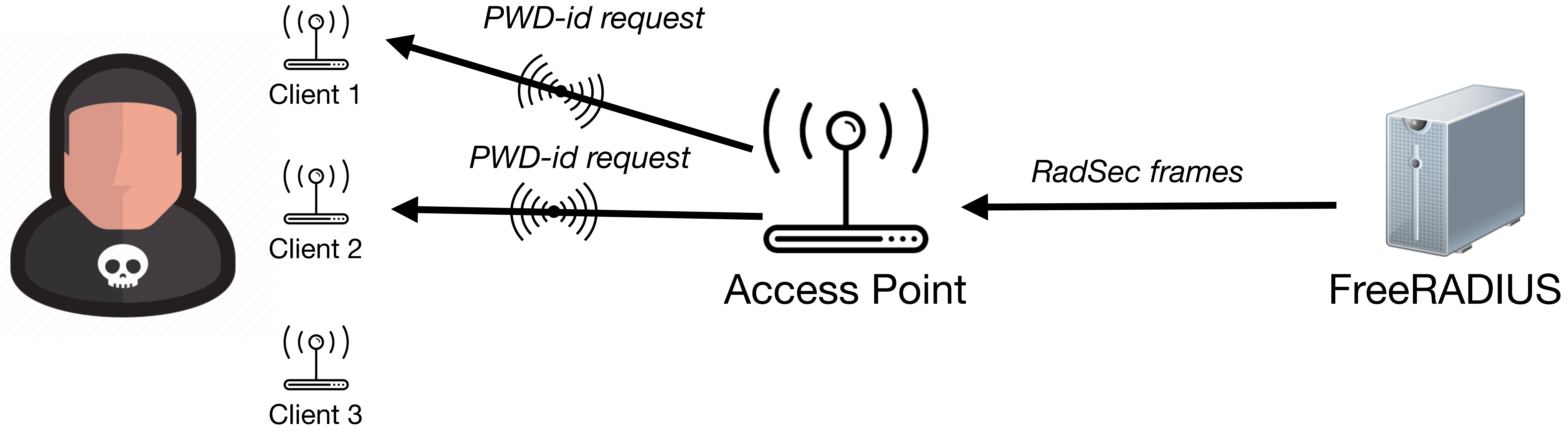
Client 1
<associate>
<associate>
Client 2
Client 3
Access Point
FreeRADIUS

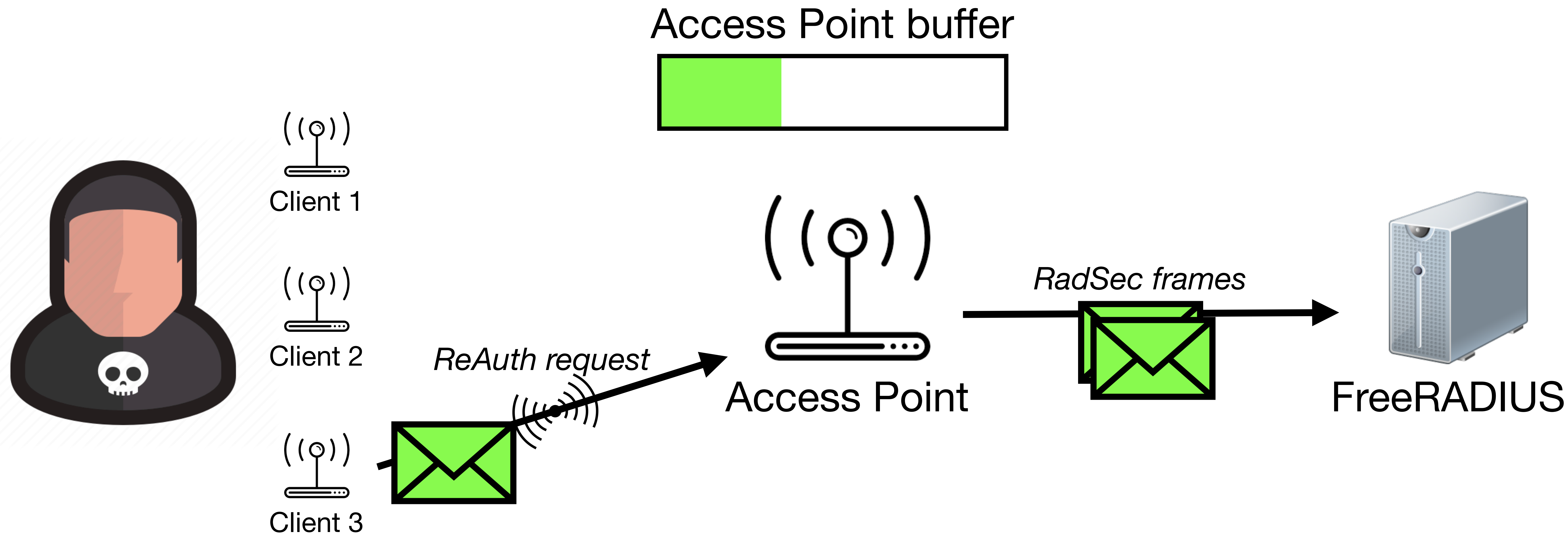


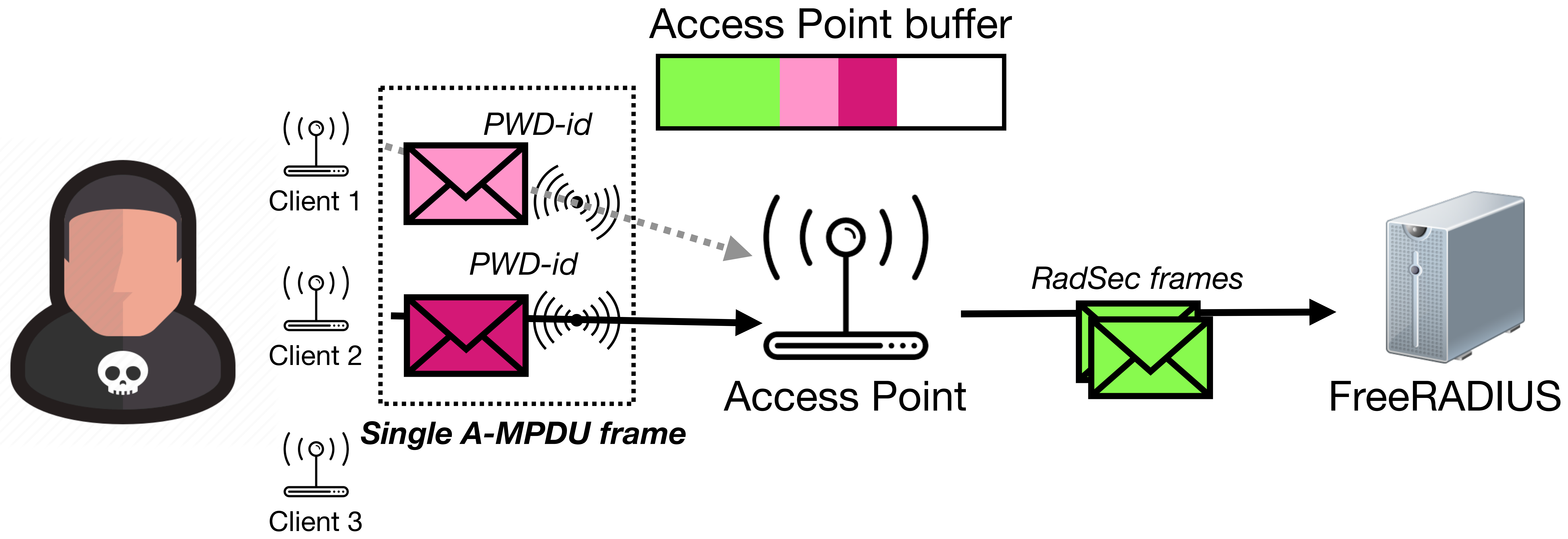
EAP-id request
Client 1
EAP-id request
Client 2
Client 3
Access Point
FreeRADIUS



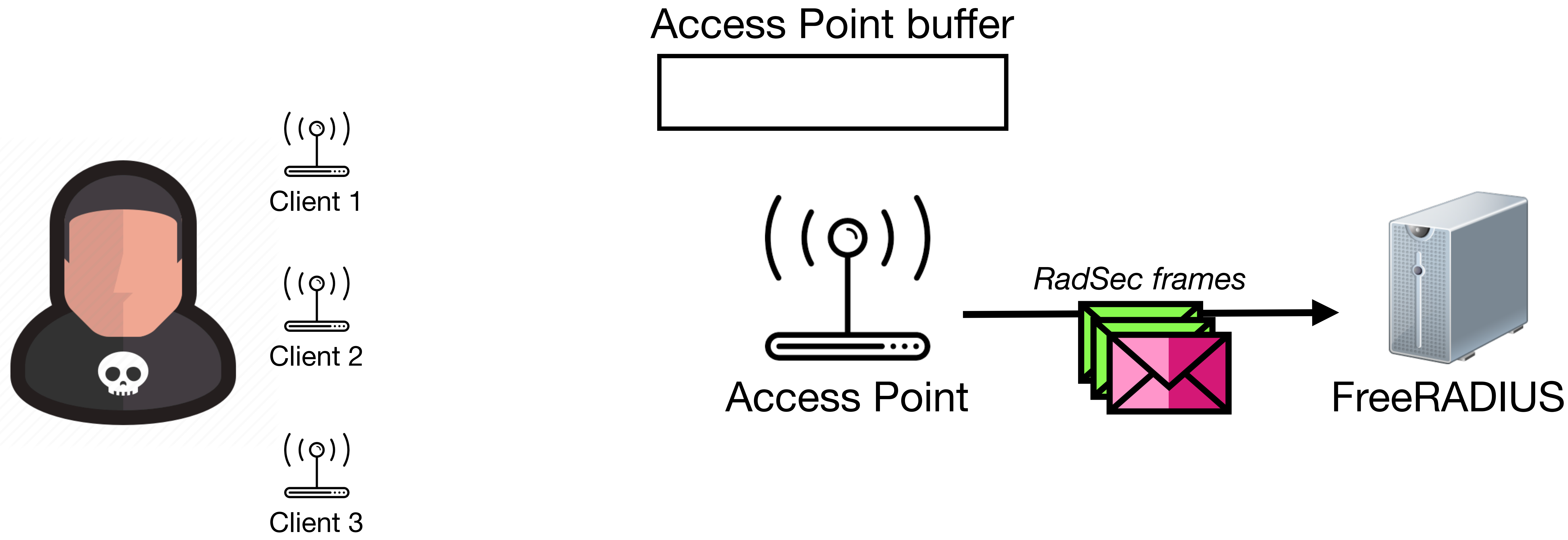
Client 1
EAP-id response
Client 2
EAP-id response
Client 3
Access Point
RadSec frames
FreeRADIUS

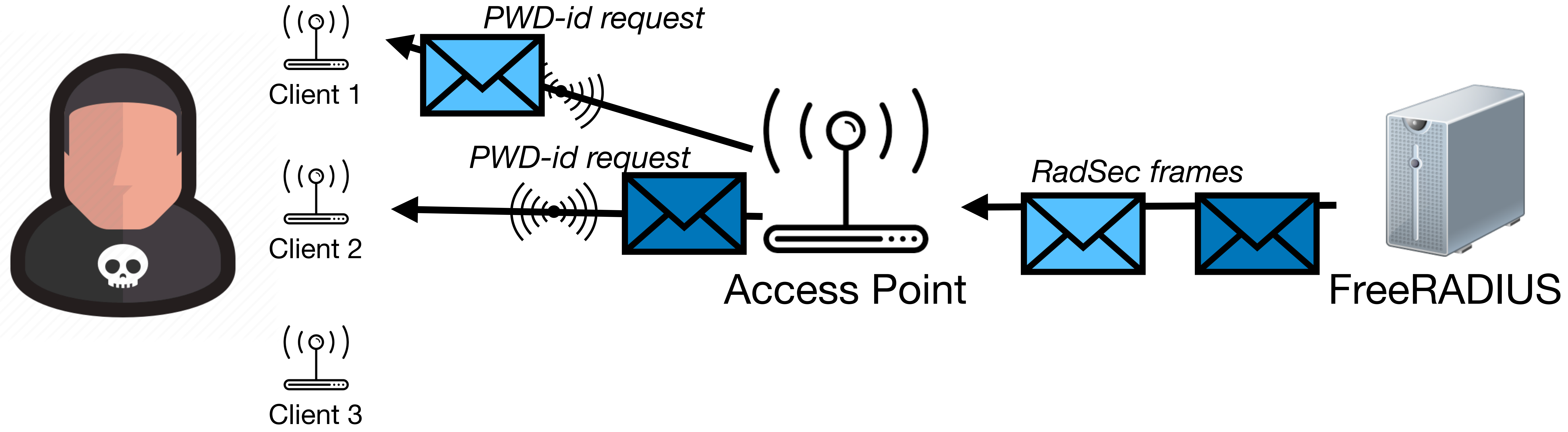

PWD-id request
Client 1
PWD-id request
Client 2
Client 3
Access Point
RadSec frames
FreeRADIUS


Access Point buffer
Client 1
Client 2
Client 3
ReAuth request
Access Point
RadSec frames
FreeRADIUS



Access Point buffer
PWD-id
Client 1
PWD-id
Client 2
Single A-MPDU frame
Client 3
Access Point
RadSec frames
FreeRADIUS

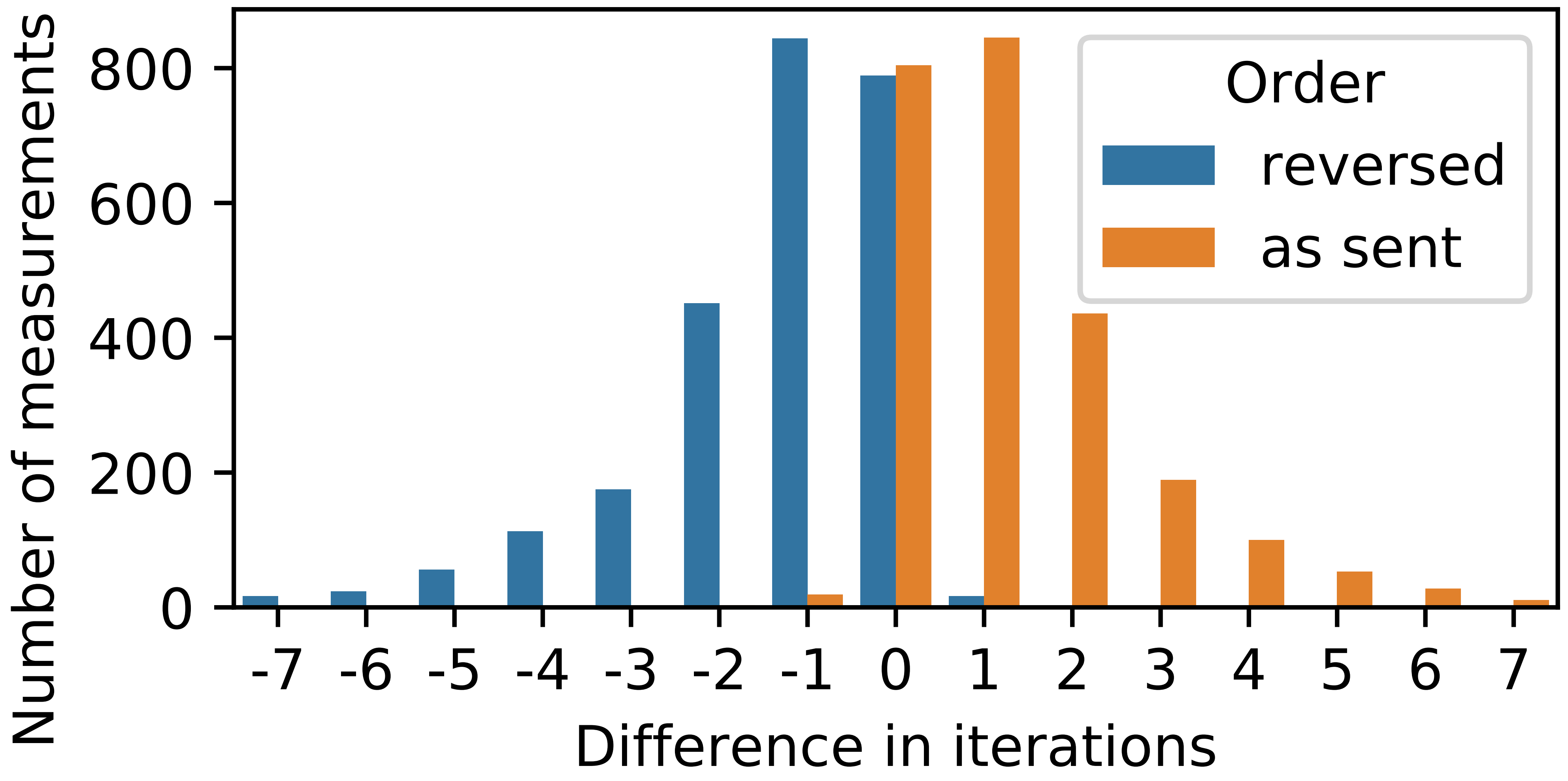

Access Point buffer
Client 1
Client 2
Client 3
Access Point
RadSec frames
FreeRADIUS



PWD-id request
Client 1
PWD-id request
Client 2
Client 3
Access Point
RadSec frames
FreeRADIUS

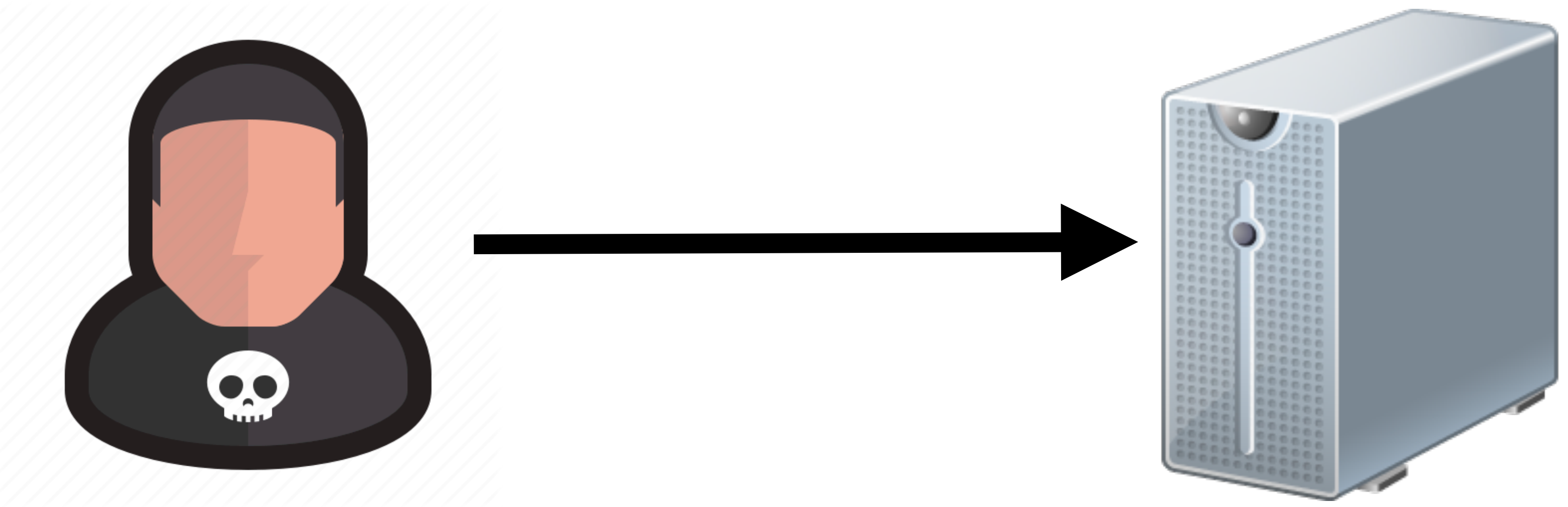
Bruteforcing Wi-Fi passwords

- Timing side-channel in hash-to-curve method is exploited
- Response order is enough information to perform bruteforce attack
- Probability of incorrect order only 0.38%
- Example RockYou password dump
  - 14M passwords
  - 40 measurements needed
  - ~86% success probability
- Costs less than $1 to bruteforce password on cloud

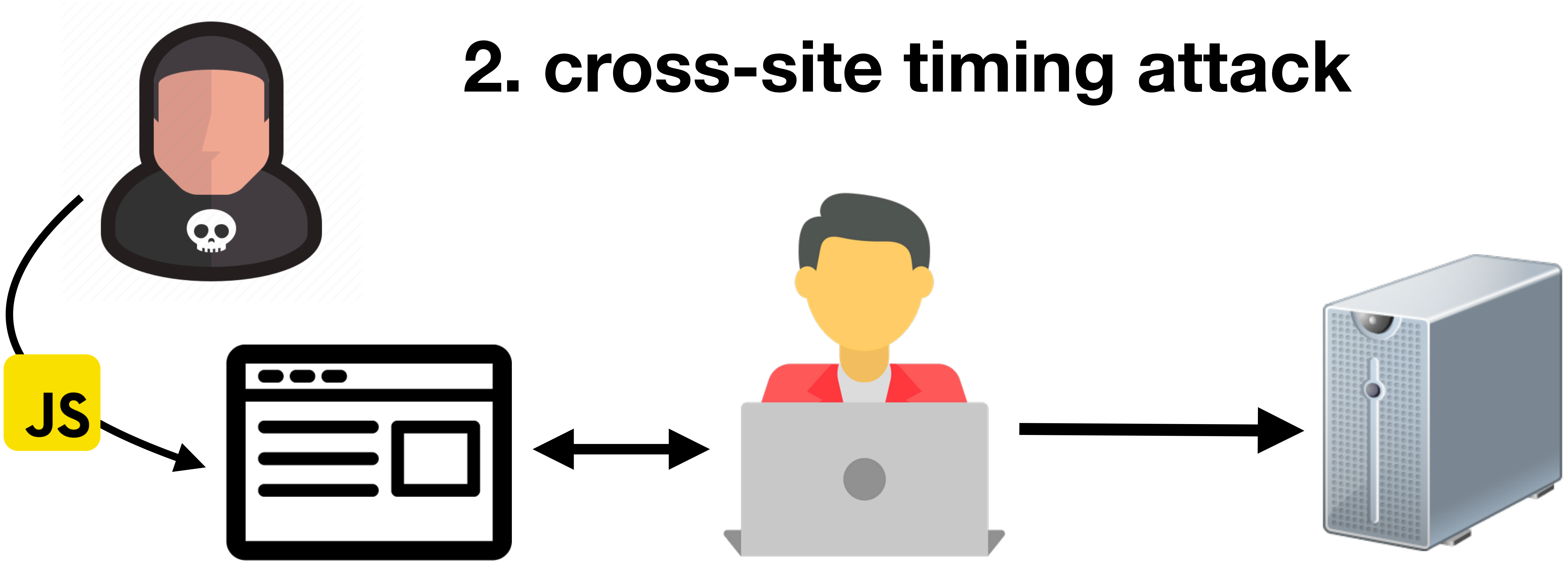


Overview

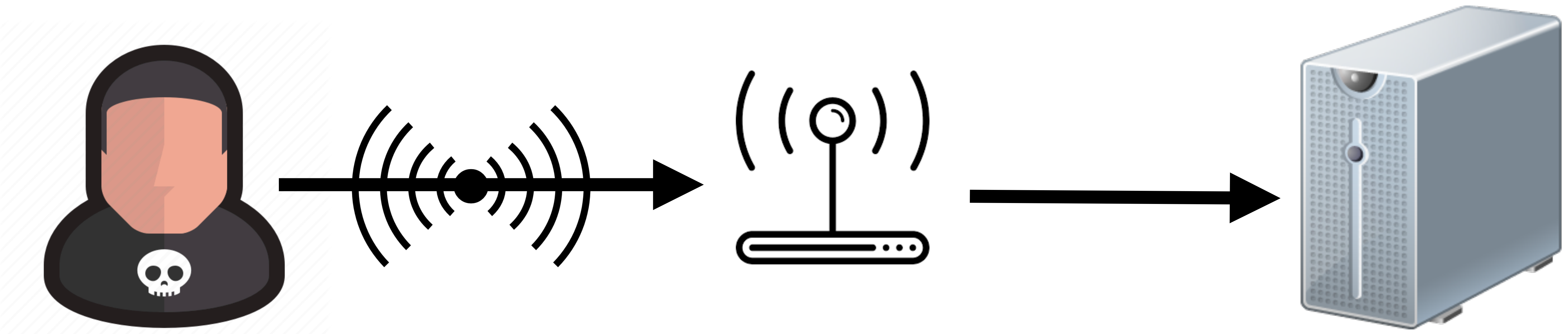
1. direct timing attack



2. cross-site timing attack



3. Wi-Fi authentication



DEMO

```php
$documents = textSearch($query);

if (count($documents) > 0) {
  $securityLevel = getSecurityLevel($user);

  // filter documents based on security level...
}
```

attack.py

```
url_prefix = 'https://vault.drud.us/search.php?q=BLACKHAT_PASSWORD='
r1 = H2Request('GET', url_prefix + char)
# @ is not part of the charset so serves as baseline
r2 = H2Request('GET', url_prefix + ‘@')

async with H2Time(r1, r2, num_request_pairs=15) as h2t:
    results = await h2t.run_attack()
    num_negative = len([x for x in results if x < 0])
    pct_reverse_order = num_negative / len(results)

if pct_reverse_order > threshold:
    print('Found next character: %s' % char)
```

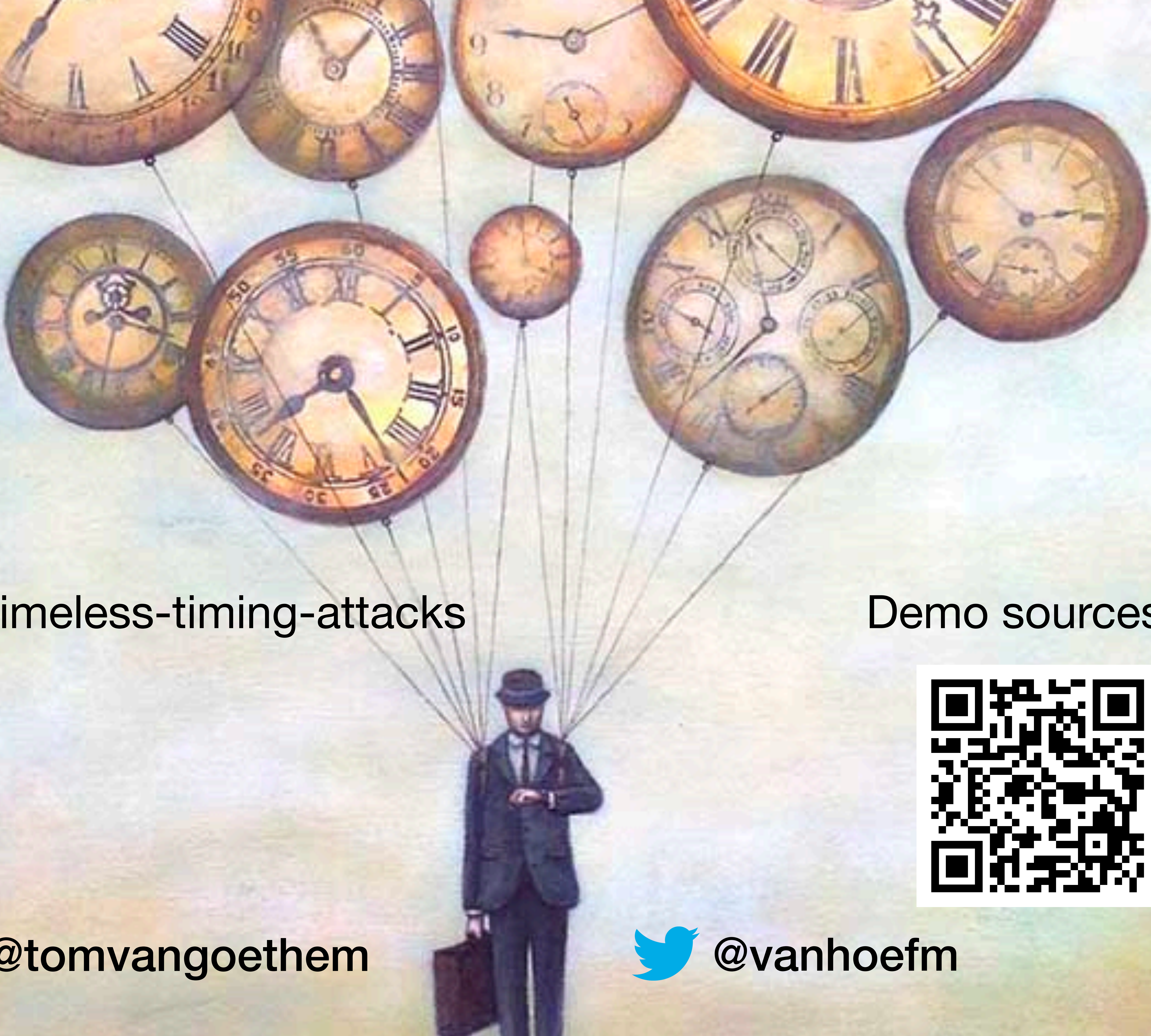
Conclusion

- Timeless timing attacks are **not affected by network jitter** at all
- Perform **remote** timing attacks with an **accuracy similar to** an attack against the **local system**
- Attacks can be launched against protocols that feature **multiplexing** or by leveraging a transport protocol that enables **encapsulation**
- All **protocols that meet the criteria** can be **susceptible to timeless timing attacks**: we created practical attacks against **HTTP/2** and **EAP-pwd** (Wi-Fi)

Thank you!

https://github.com/DistriNet/timeless-timing-attacks

Demo sources:



@tomvangoethem

@vanhoefm