Writing Secure APIs

Armin Ronacher

for PyCon.ru 2014

Armin Ronacher

Independent Contractor for Splash Damage / Fireteam
Doing Online Infrastructure for Computer Games

lucumr.pocoo.org/talks

… but does it support SAML?

Why Secure APIs?

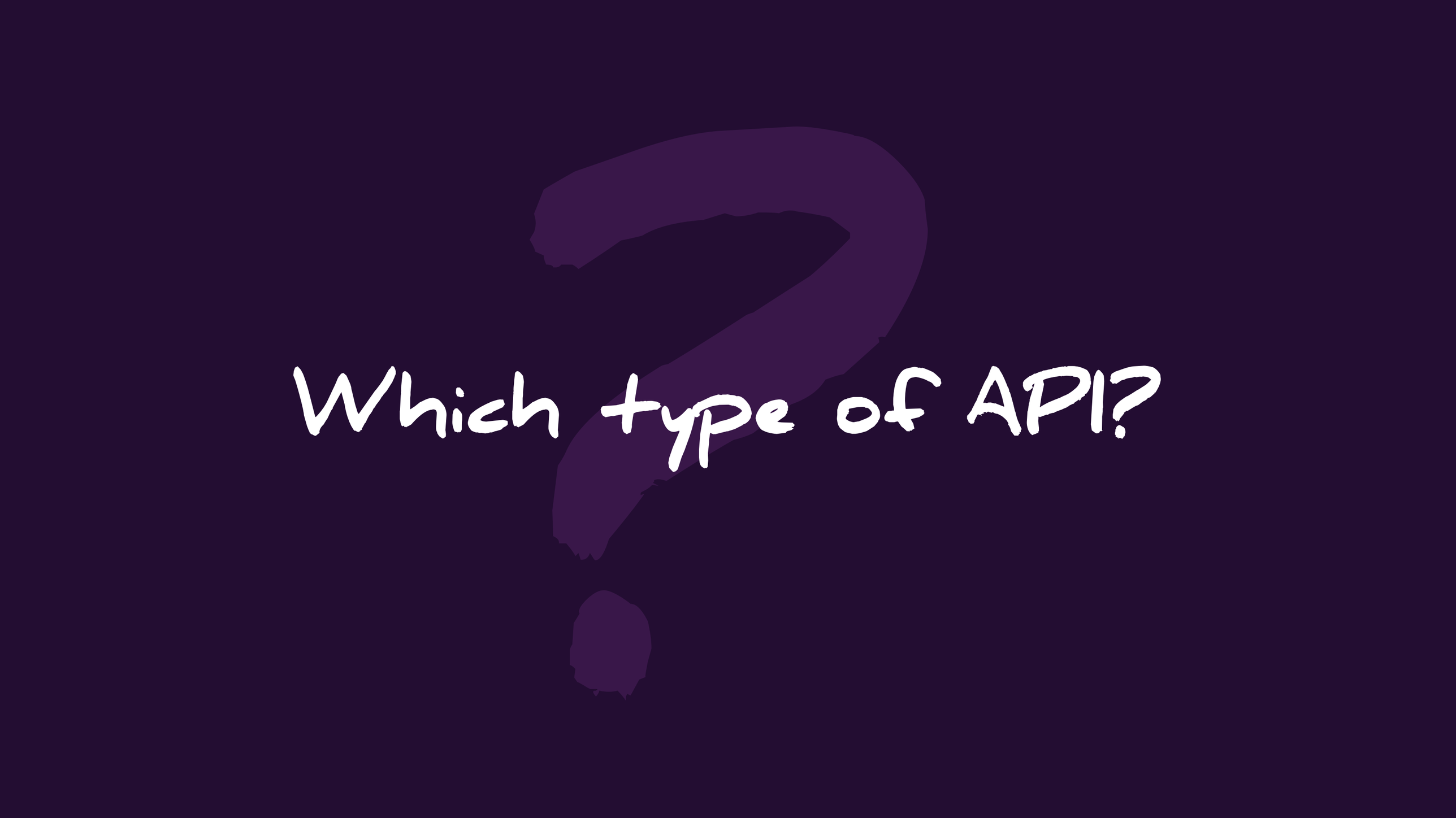
Starbucks killed the enencrypted connection

your enemy
surfs on the
same Wifi
as you

Things to secure:

Session Cookies
Access Tokens
Credit Card Numbers
…

don’t be afraid of government,
your enemy is sitting on the same Wifi


Which type of API?

Web vs Browser Web

It's all about the CAs

Browsers trust Public CAs

Services should trust Certificates

Browser Web

You use HTTP
And there is a good old browser

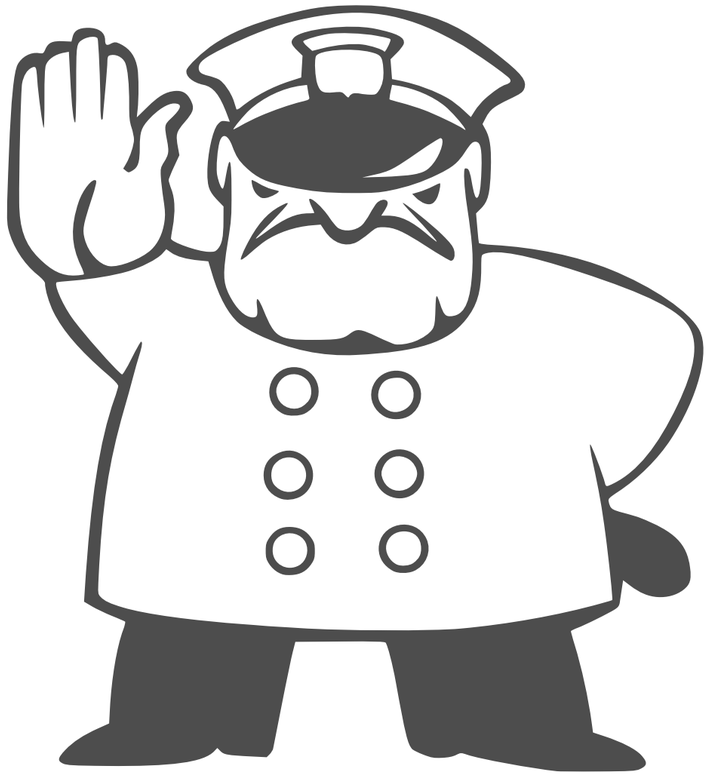
Websites, JavaScript APIs, etc.

Web

You use HTTP
There is no browser
Or it's a browser under your control

Service to Service communication, Custom APIs, etc.

If there is no browser
I do not need a public CA



If there is a browser there is
not much you can do

What does a CA do?

let's look at something else first

Understanding Trust

Authenticity vs Secrecy

Authenticity:

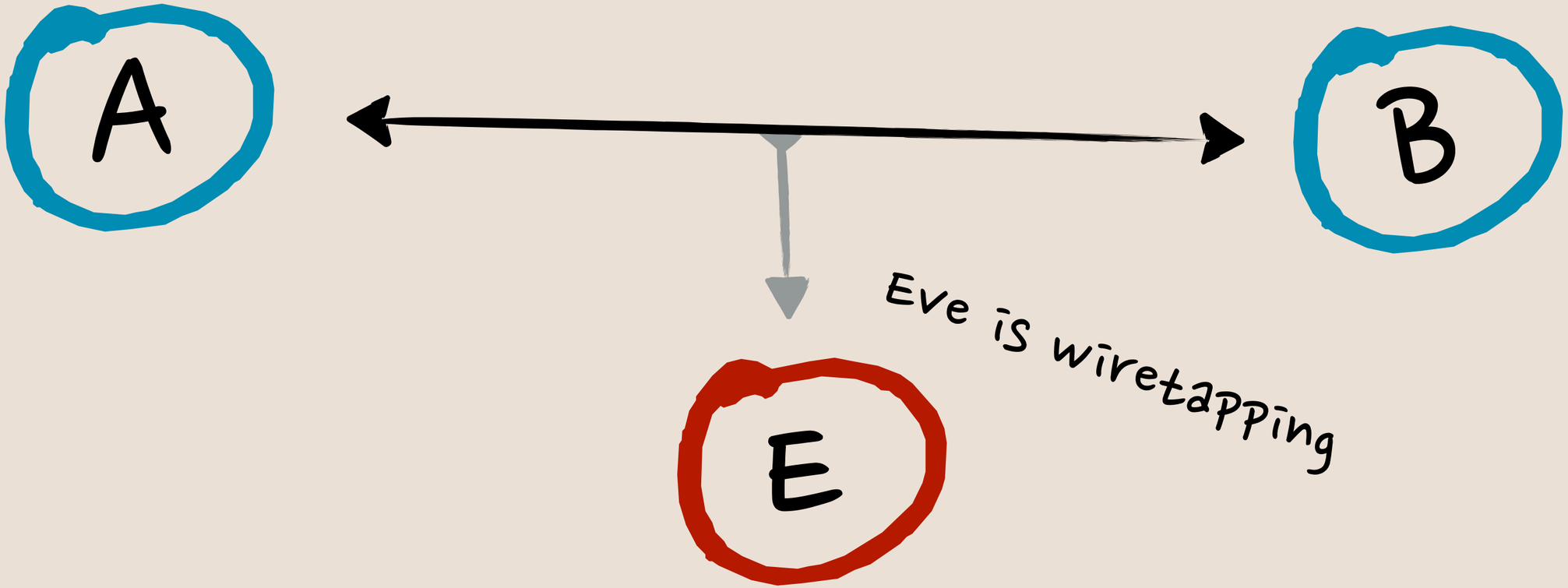
the author of the message is the author the receiver knows and trusts.

Secrecy:

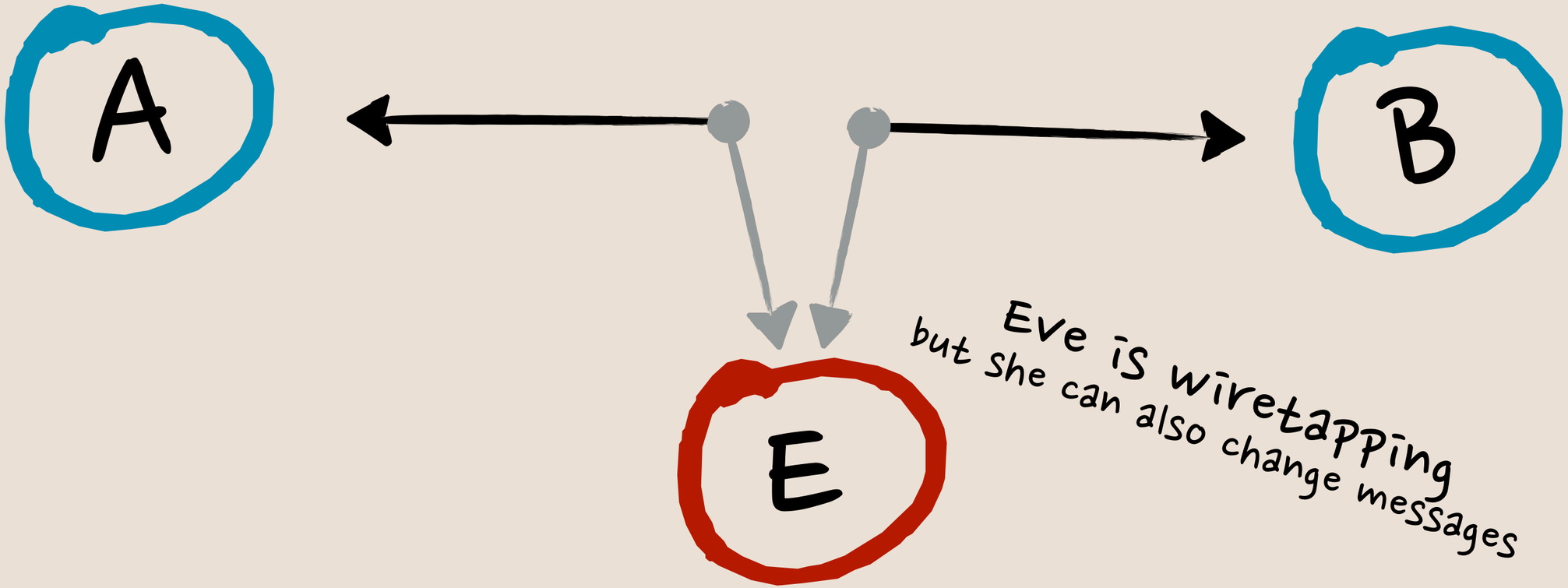
nobody besides author and intended receiver read the message.

Authenticity > Secrecy

Authenticity:



Secrecy without Authenticity:




Signatures

Signatures add:

authentication, *integrity*
and *non-repudication*

non-repudication is pointless for APIs!

MAC

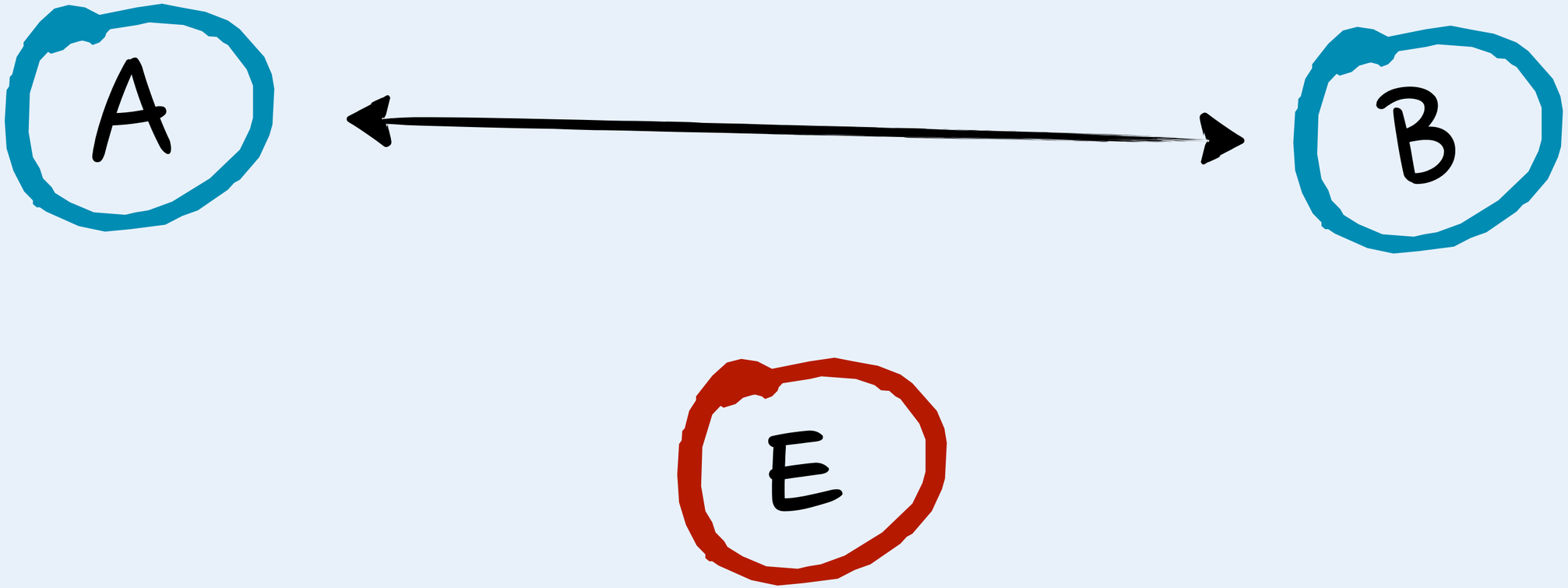
“Signature without non-repudication’

1. Consumer gets shared key
2. Signs request with shared key
3. Sends request to server
4. Server verifies request
5. Server creates and signs response
6. Client verifies response

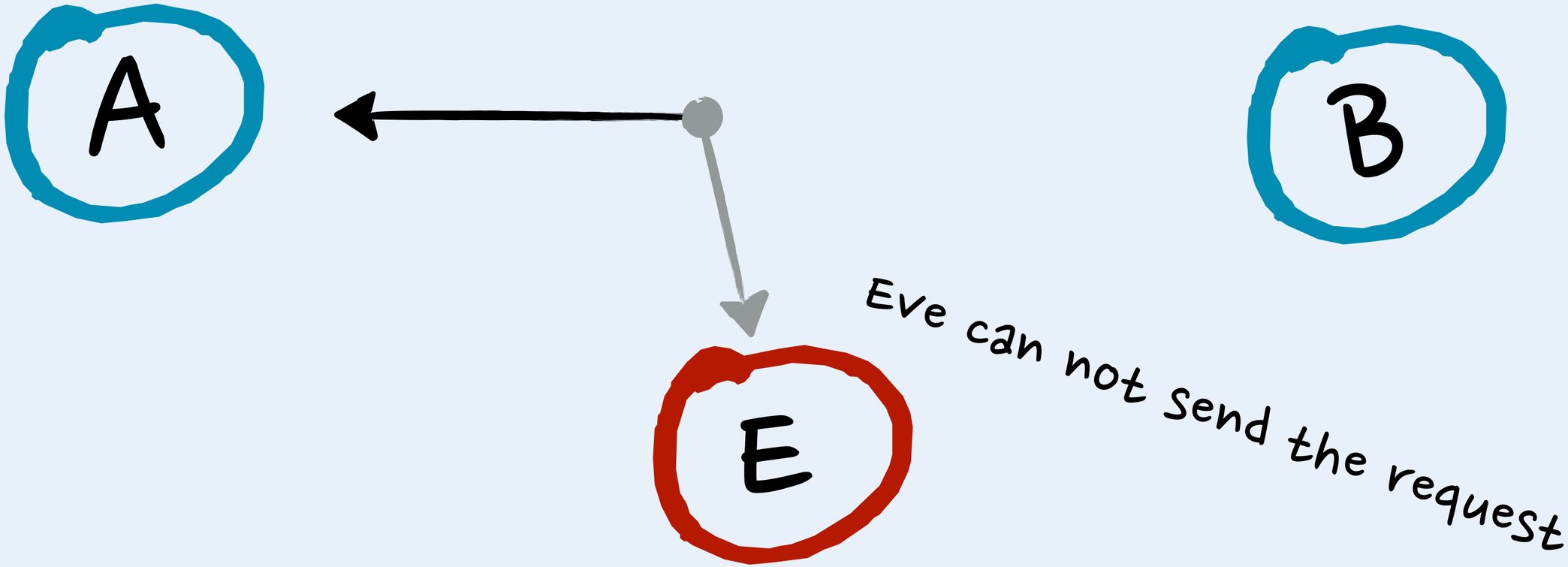
the key is never on the wire!

(Eve is a sad attacker)

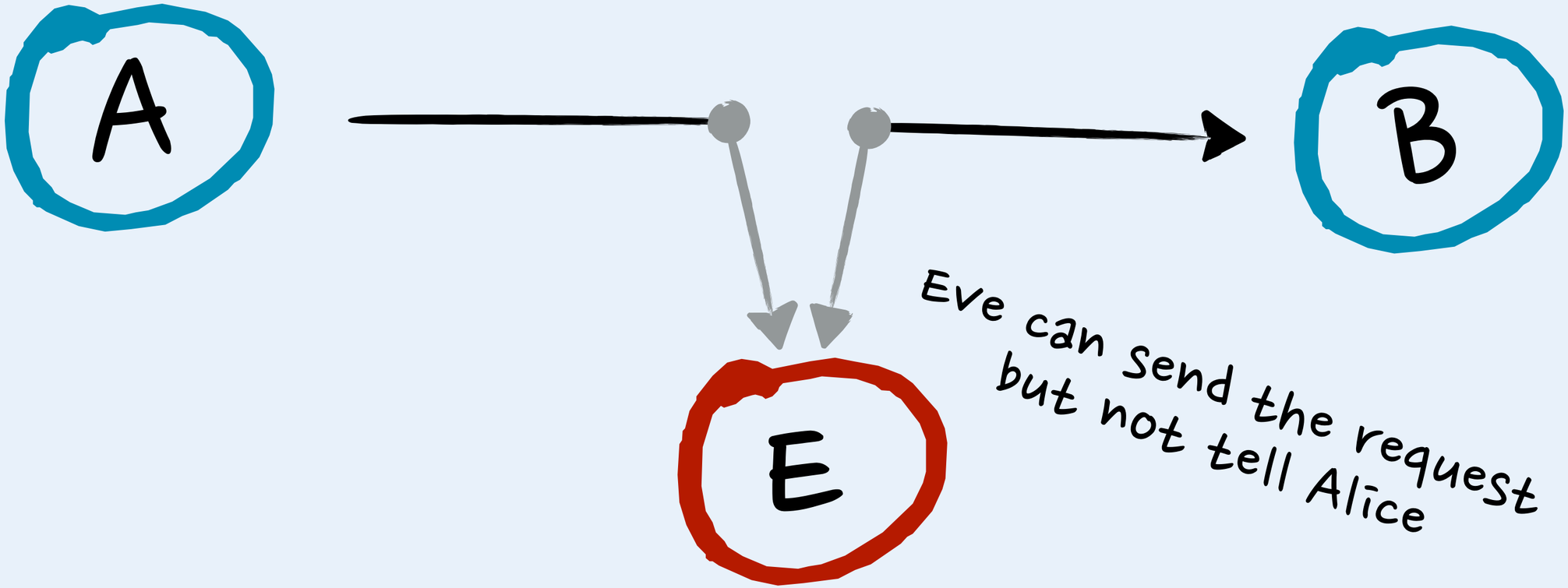
what's the worst Eve can do?



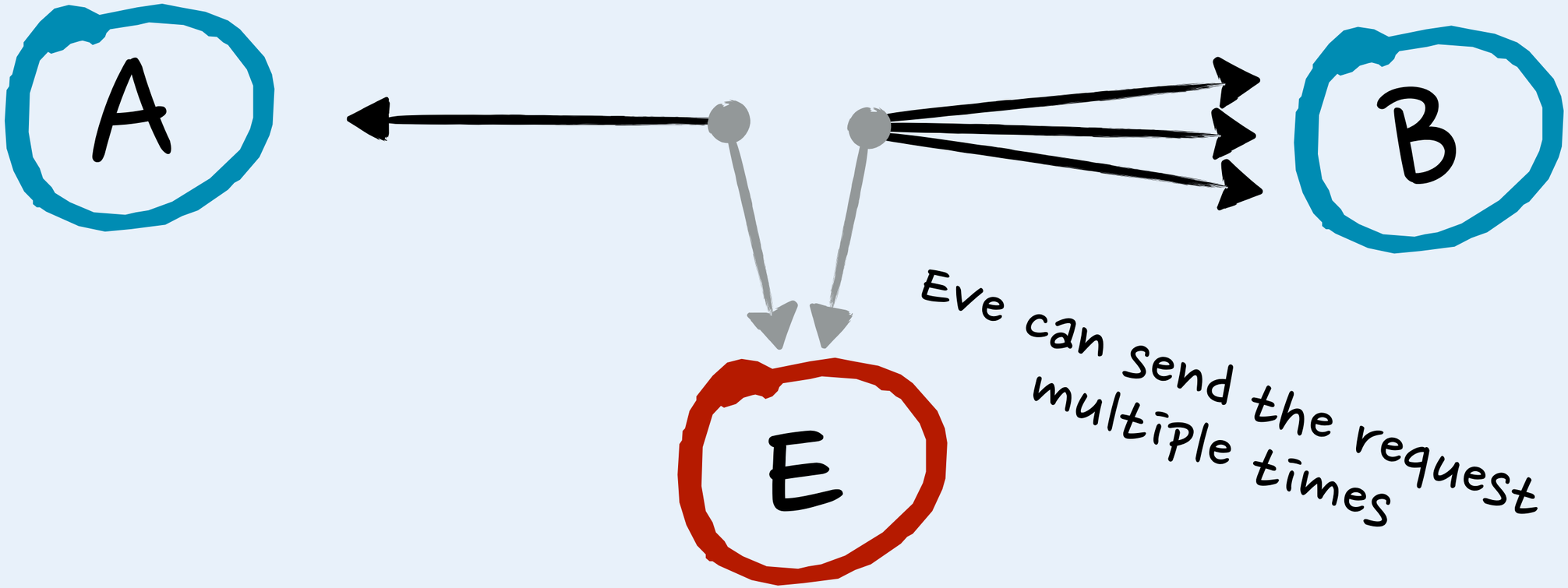
what's the worst Eve can do?



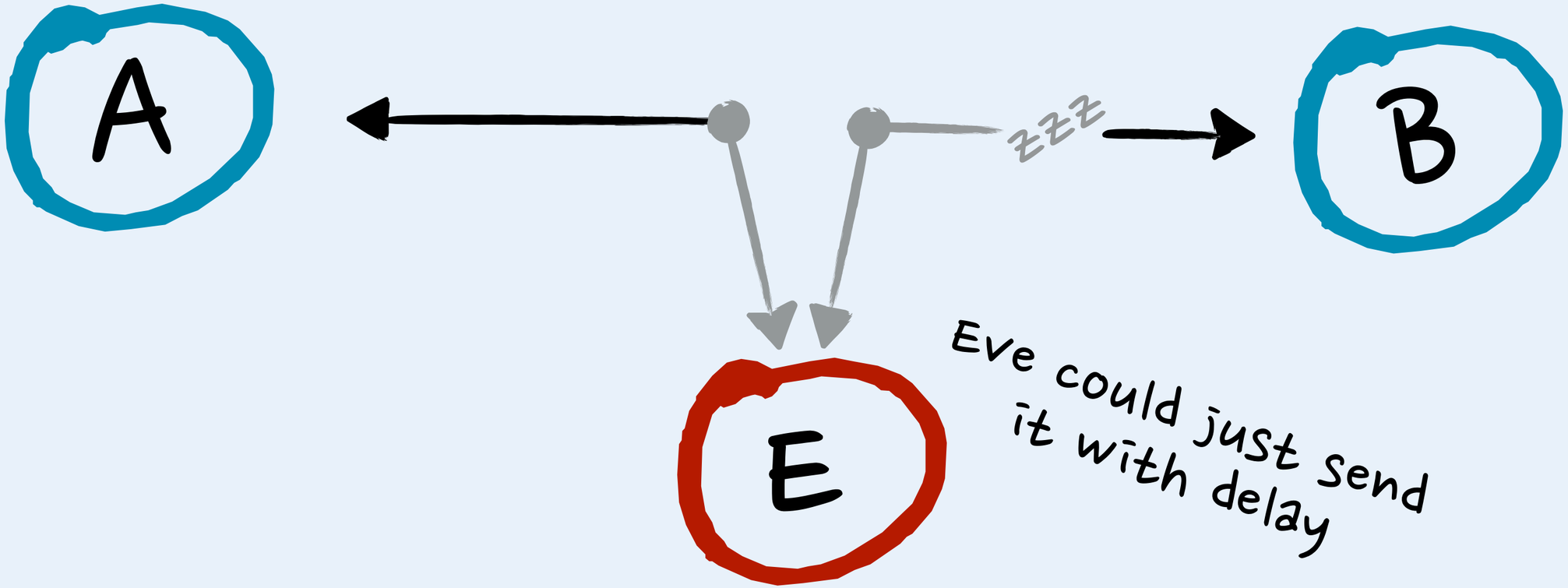
what's the worst Eve can do?



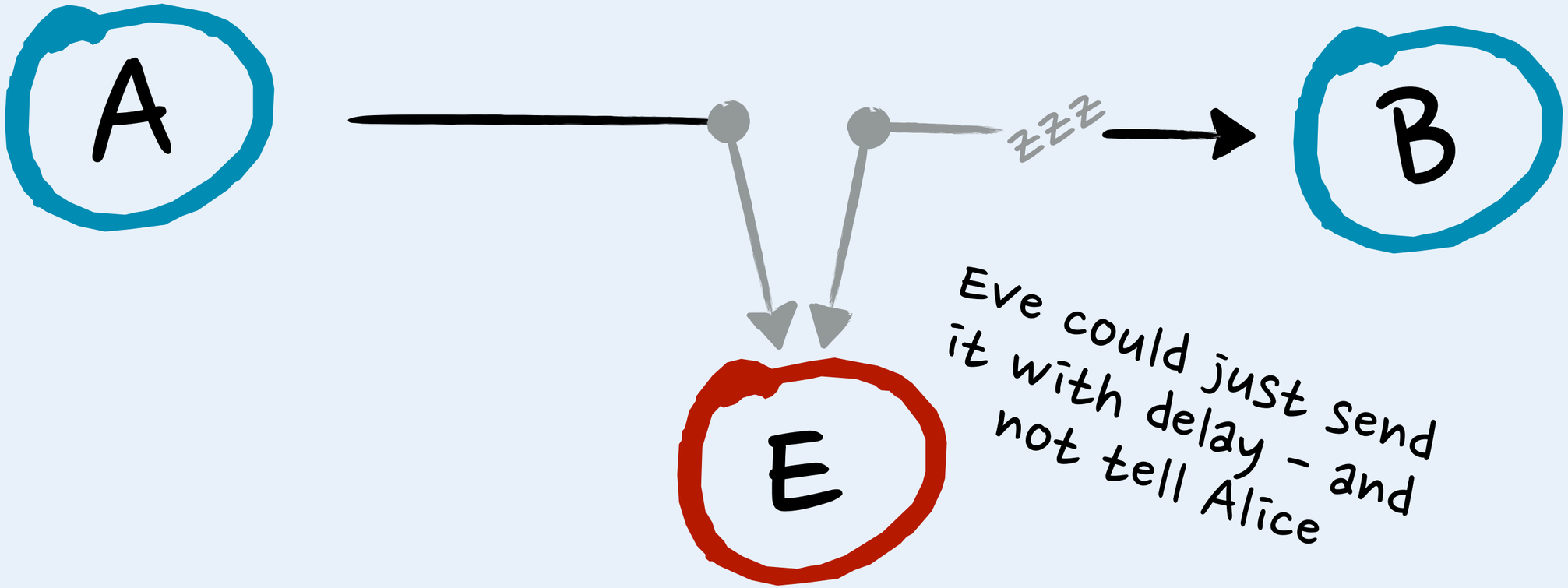
what's the worst Eve can do?



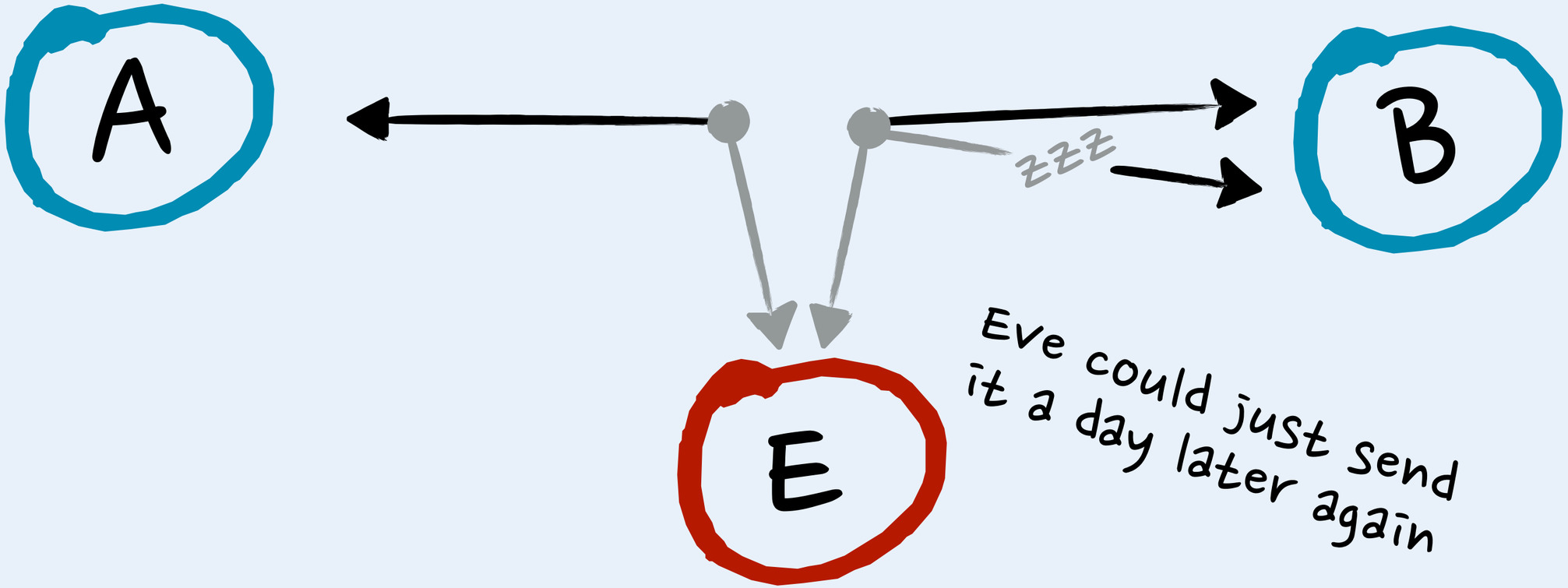
what's the worst Eve can do?



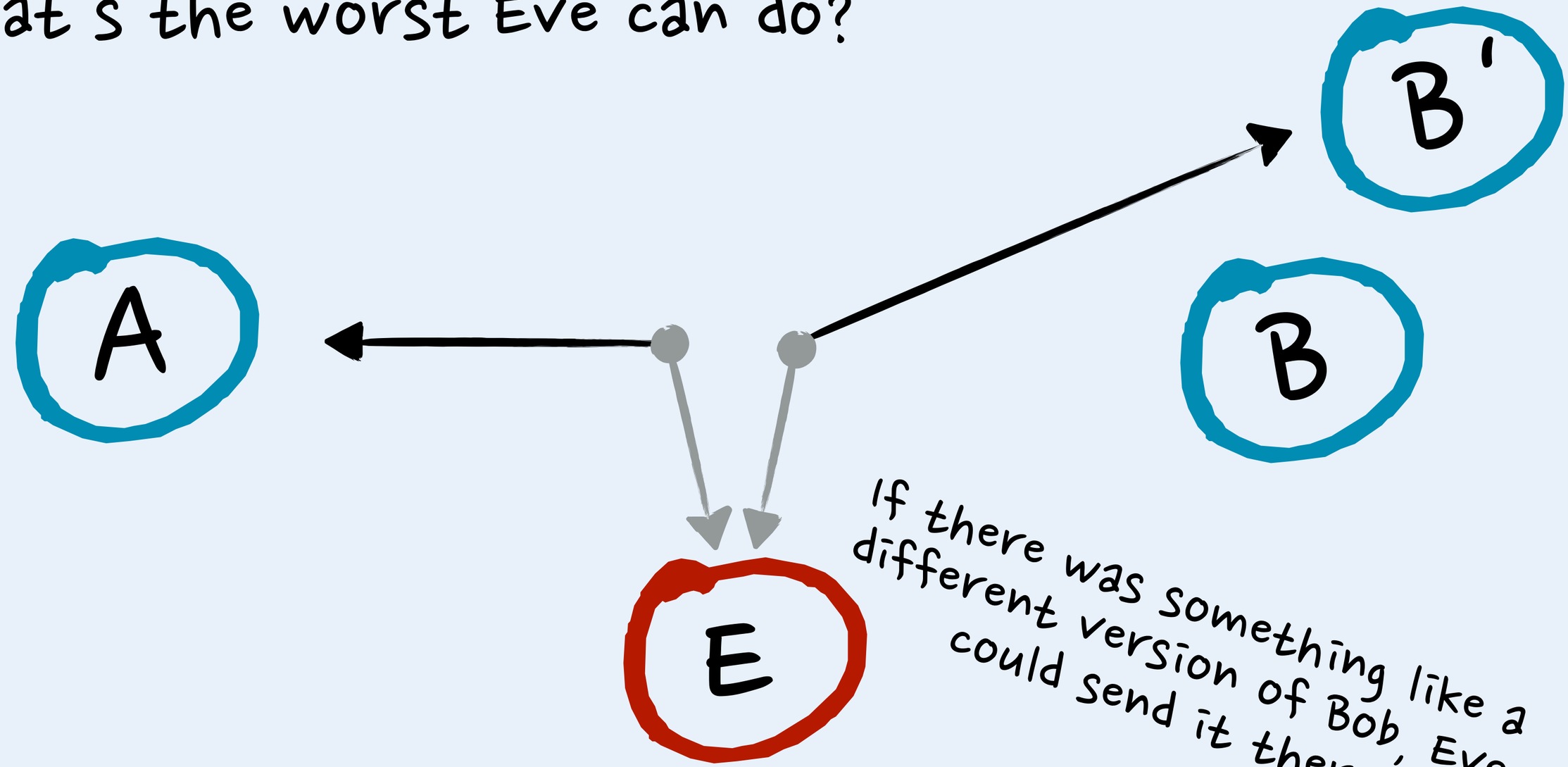
what's the worst Eve can do?



what's the worst Eve can do?



what's the worst Eve can do?

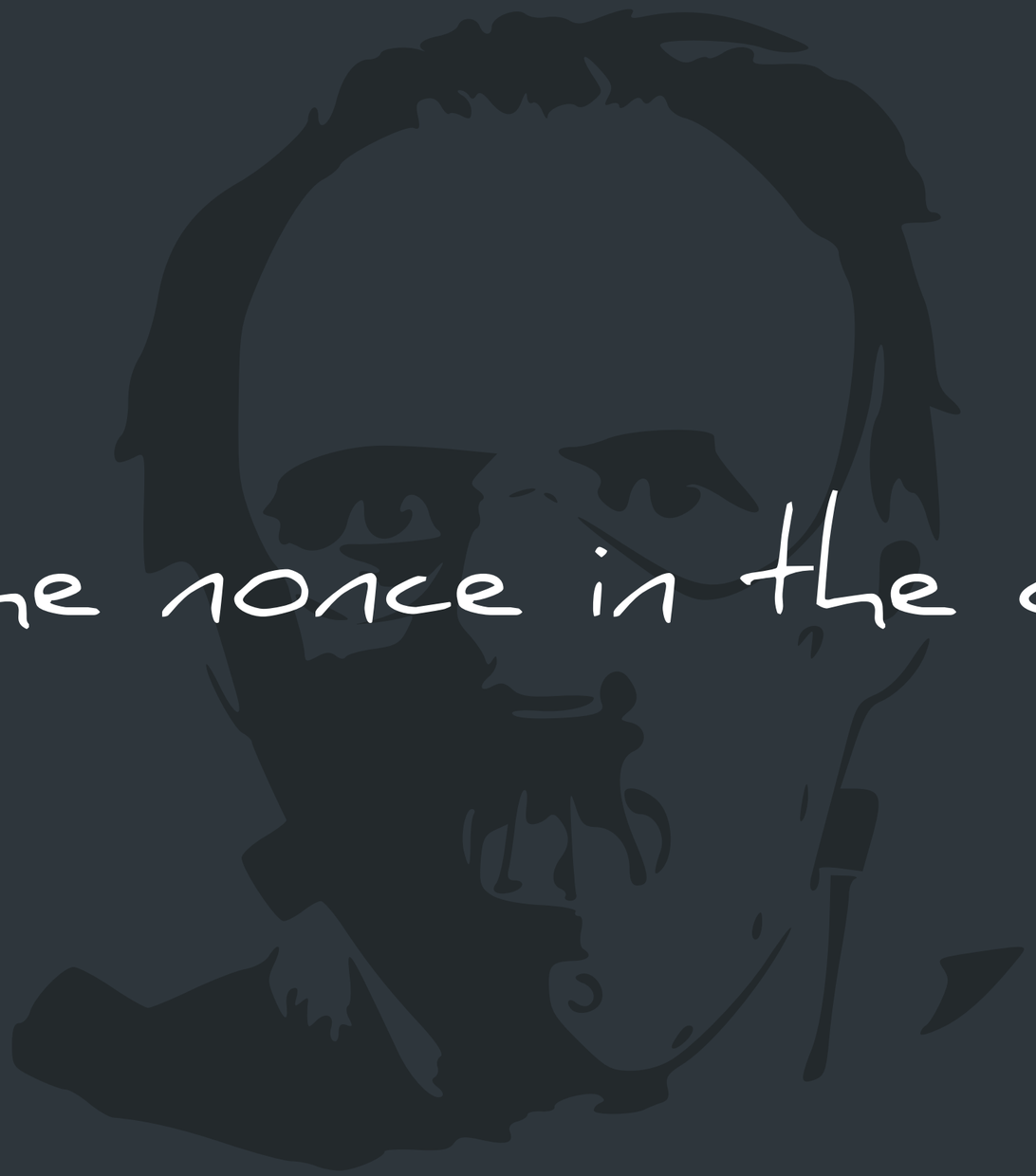


Why most of those things don't matter

The core problem is that Eve can wiretap

Even without Eve a client can never know if a message was sent!

Idempotency needs to be implemented anyways



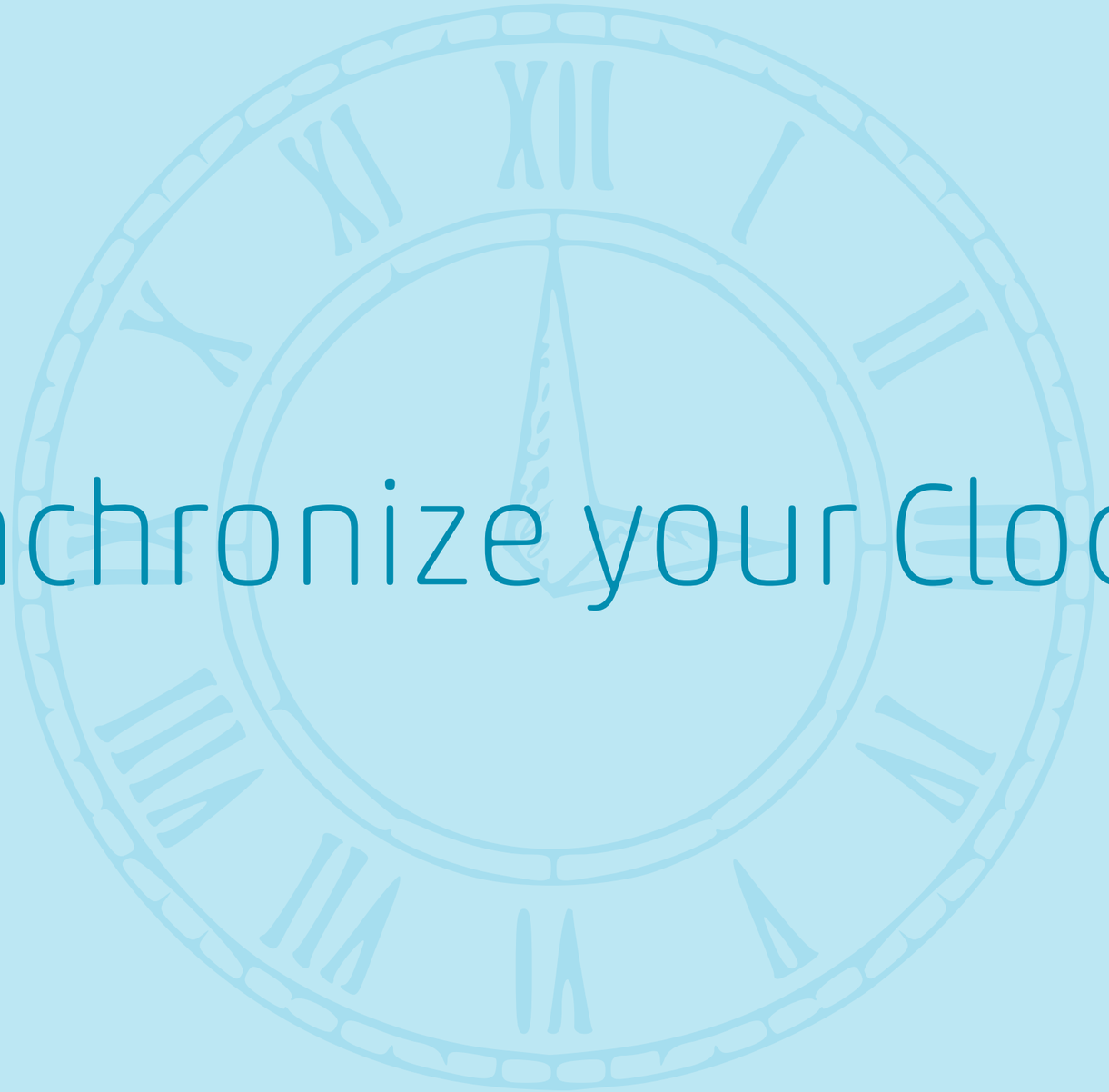
put the nonce in the cache!

```
def handle_request(request):
    signature = create_signature(request)
    if signature != request.supplied_signature:
        abort_with_error()
    if not nonce_has_been_used(request.nonce):
        perform_modification()
        remember_nonce(request.nonce)
    result = generate_result()
    return generate_response_with_signature(result)
```

Signature Expiration

Make Signatures Expire

or you store years and years of nonces



Synchronize your Clocks!

```
def verify_message(data):
    sig, message = split_signature(data)
    reference_sig = calculate_signature(message)
    if reference_sig != sig:
        raise InvalidSignature()
    header, payload = split_message(message)
    expires_at = get_expiration_time(header)
    if expires_at < current_time():
        raise SignatureExpired()
    return header, payload
```

Message can only be used once
only need to remember nonce for signature lifetime

itsdangerous

pypi.python.org/pypi/itsdangerous

```python
import time
from itsdangerous import URLSafeSerializer, BadSignature


def get_serializer():
    return URLSafeSerializer(secret_key=get_secret_key())

def make_auth_ticket(user_id, expires_in=60):
    return get_serializer().dumps({
        'user_id': user_id,
        'expires_at': time.time() + expires_in,
    })

def verify_auth_ticket(ticket):
    data = get_serializer().loads(ticket)
    if data['expires_at'] < time.time():
        raise BadSignature('Ticket expired.')
    return data['user_id']
```

Signing killed oAuth 1.0a

People did not want to sign

... then only sign on the Server

Enter: Token Based Authentication

Token Based Authentication

requires SSL or another secure transport
uses short lived tokens
used for exchanging authentication information

This is what OAuth 2.0 is

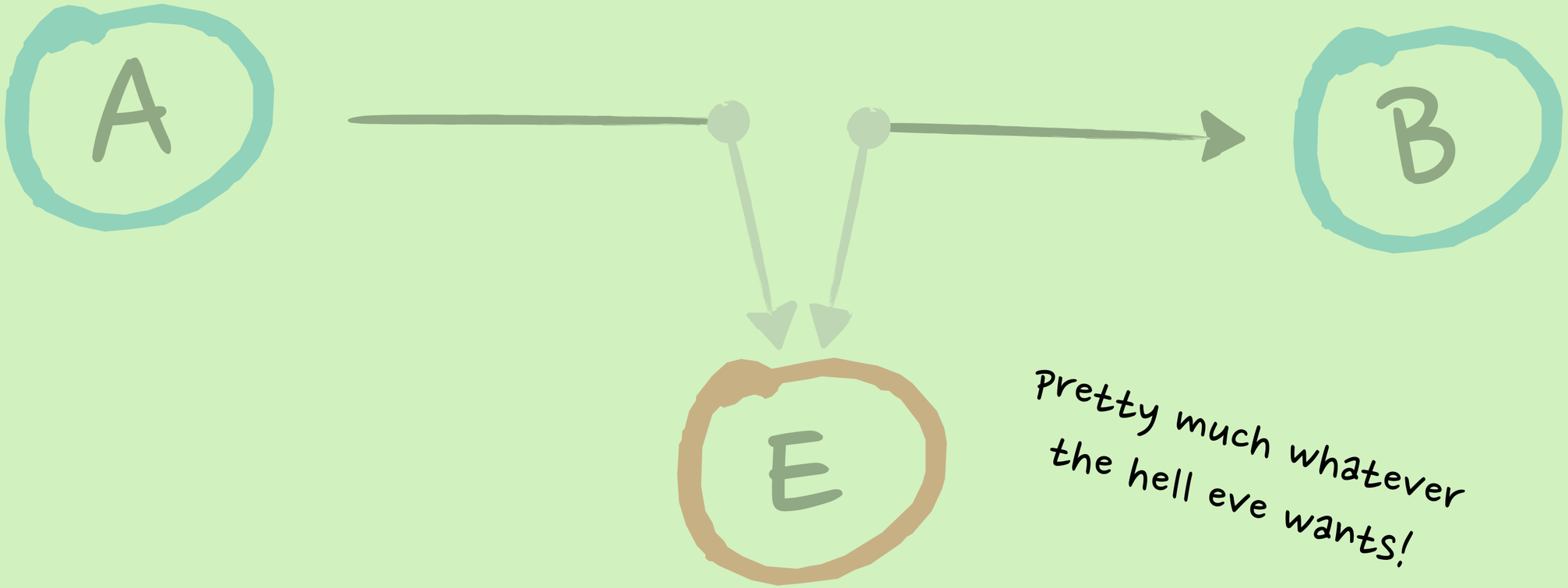
(also called bearer token)

Access & Refresh Token

short and lives for one day

signed, complex and lives for a really long time (eternal)

what's the worst Eve can do?



Token Based Authentication is a Tradeoff
It ‘limits’ what an attacker can do

Stolen Access Token: ~24 hours of damage
Refresh Tokens are only exchanged on Token Refresh

only ever used in combination with SSL!

```python
import uuid
from itsdangerous import URLSafeSerializer


def get_serializer():
    return URLSafeSerializer(secret_key=get_secret_key())


def make_token(user):
    token_data = {
        'user_id': user.id,
        'generation': user.token_generation,
    }
    refresh_token = get_serializer().dumps(token_data)
    access_token = str(uuid.uuid4())
    store_token(access_token, token_data)
    return access_token, refresh_token
```


SSL without Public CAs

Certificate Revocations do not work!

Certificate says valid until 2020

Private Key Leaked

Certificate says valid until 2020

Private Key Leaked

Certificate says valid until 2020

Private Key Leaked

what now?

treat it like token based authentication

self signed certificates are good

(just not if you deal with normal users)

Become your Own Private CA

MAKE CERTIFICATES EXPIRE

every 24 hours

They are not the same
But you treat them the same

refresh token » root certificate

access token » connection certificate

Certificate Travels over Wire

Private Key Does Not

how it works:

1. create root certificate
2. trust root certificate always
3. have a cron job issue certificates
4. signed by that root every 12 hours
5. distribute them to the web servers
6. cycle certs every 12 hours

what makes it good:

you can now looks your private key
maximum damage is ~1 day
your root's private key is on a box not connected
to the internet with all ports closed.

Survives Heartbleed :-)

```
from requests import get

resp = get('https://api.yourserver.com/',
           verify='your/certificate.bundle')
```

Apple's crappy OpenSSL always trusts Keychain :-(

why not with public CAs?

1. why trust the whole world?
2. you need to sign on their shitty web application
3. on most CAs you pay for each signature
4. and they are most of the time valid for a year


Structure for Security

Request comes in
Response goes out

(you can explain that)

Example

Annotate Views for Security Rules

```
@requires_role('system_management')
def manage_system(request):
    return Response(...)
```

Example

Encapsulate Security Systems

```
class UpdateUser(IdempotentEndpoint):
    id = Parameter(UUID)
    changes = Parameter(dict)

    def load_state(self):
        self.user = fetch_user(self.id)

    def update_state(self):
        self.user.update(self.changes)

    def get_response(self, state):
        return JSONResponse(self.user.to_json())
```

Example

Add Security Contexts

```
class Context(object):

    def __init__(self):
        self.account = None
        self.role_subset = set()

    def load_token(self, token, role_subset):
        self.account = find_account(token['account_id'])
        self.role_subset = set(role_subset)

    def has_role(self, x):
        if self.account is None or x not in self.role_subset:
            return False
        return x in self.account['roles']
```

Feel Free To Ask Questions

Talk slides will be online on lucumr.pocoo.org/talks
You can find me on Twitter: @mitsuhiko
And gittip: gittip.com/mitsuhiko
Or hire me: armin.ronacher@active-4.com