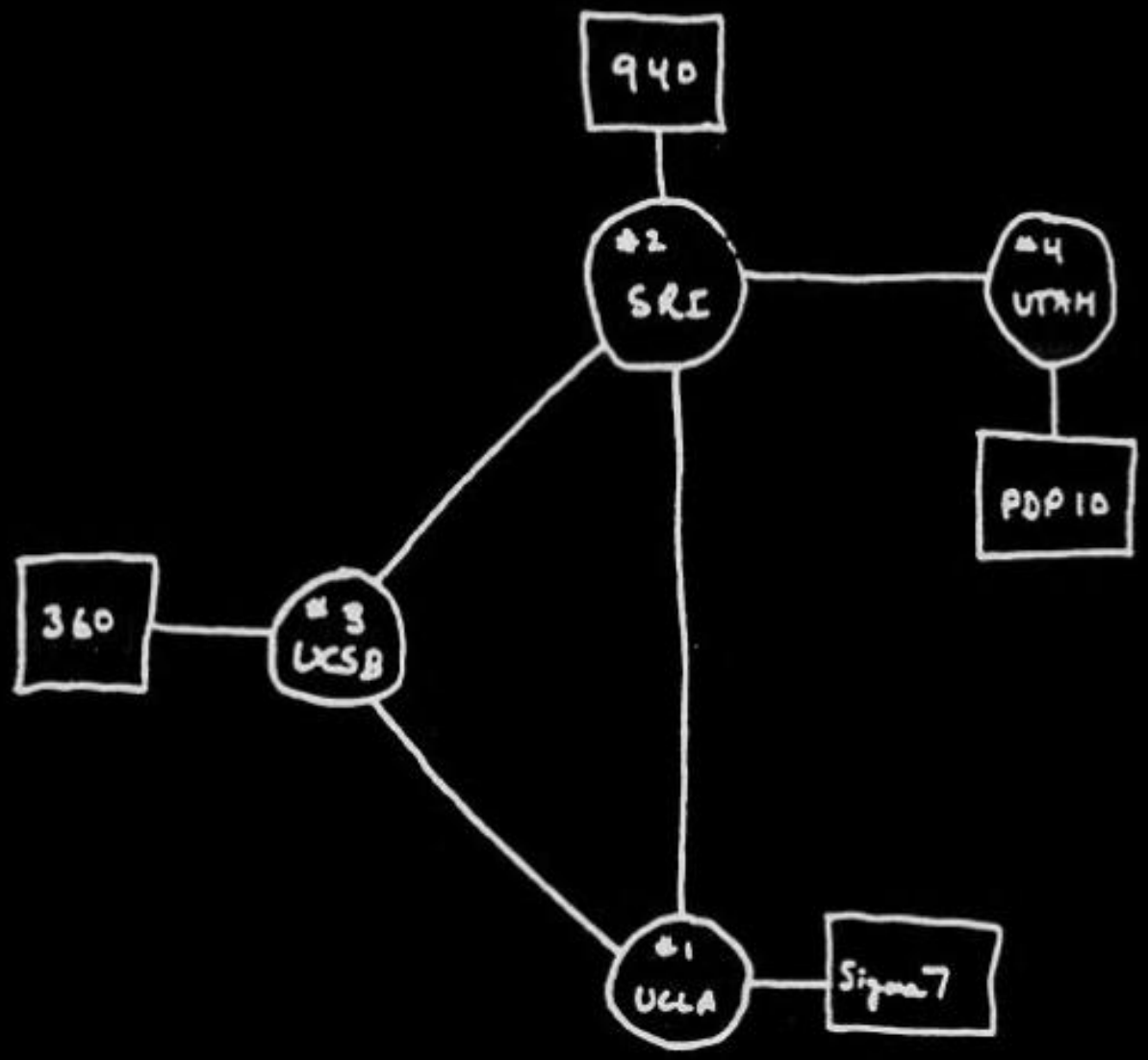
We Fight For The User's... Session

BSides Las Vegas 2025

Mark Hoopes


940
#2
SRI
#4
UTAH
PDP 10
360
#3
UCSB
#1
UCLA
Sigma7

HTTP


N
A
M
C
X Wind
intosh

Session Defenses… Why?

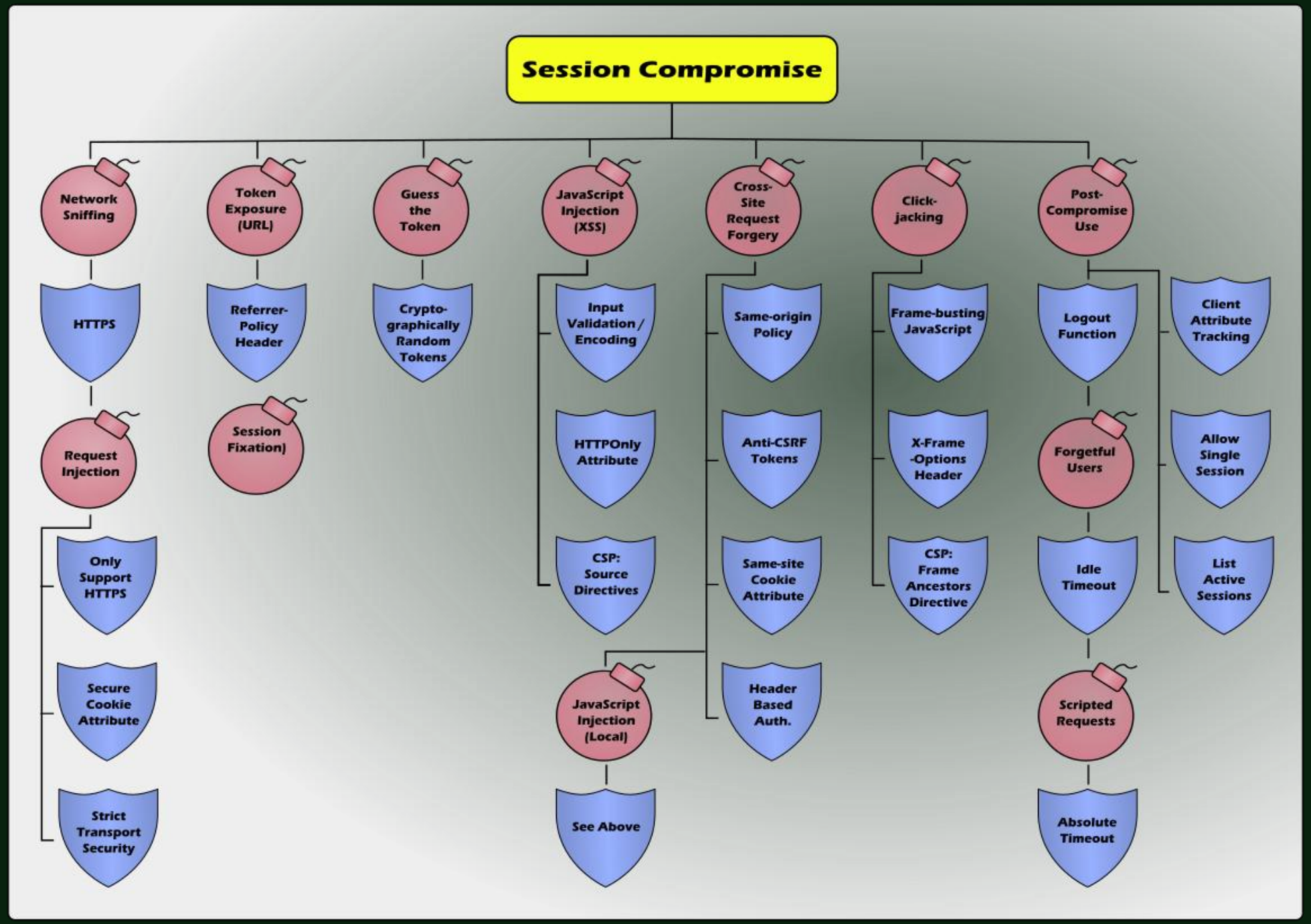
I'm tired attribute and s dings.

What do you want to sell?

Cookies and logins

[discussion about this not being the marketplace]

Sell away!

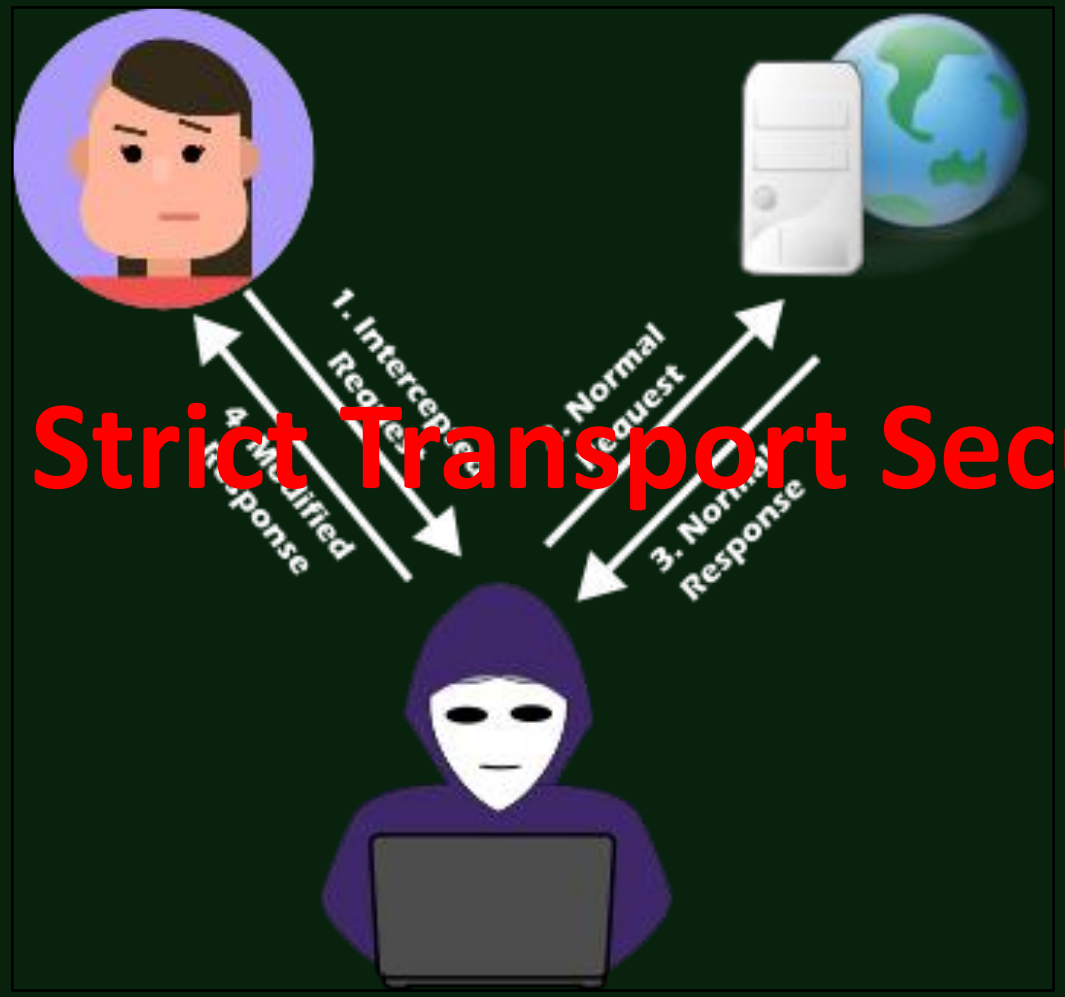

Session Compromise
Network Sniffing
HTTPS
Request Injection
Only Support HTTPS
Secure Cookie Attribute
Strict Transport Security
Token Exposure (URL)
Referrer-Policy Header
Session Fixation)
Guess the Token
Crypto-graphically Random Tokens
JavaScript Injection (XSS)
Input Validation / Encoding
HTTPOnly Attribute
CSP: Source Directives
JavaScript Injection (Local)
See Above
Cross-Site Request Forgery
Same-origin Policy
Anti-CSRF Tokens
Same-site Cookie Attribute
Header Based Auth.
Click-jacking
Frame-busting JavaScript
X-Frame -Options Header
CSP: Frame Ancestors Directive
Post-Compromise Use
Logout Function
Forgetful Users
Idle Timeout
Scripted Requests
Absolute Timeout
Client Attribute Tracking
Allow Single Session
List Active Sessions

Session Protection
“On-the-wire”

RFC 2109: Section 4.2.2

```
Secure
      Optional.  The Secure attribute (with no value)
directs the user agent to use only (unspecified) secure
means to contact the origin server whenever it sends back
this cookie.
```

HTTP Strict Transport Security



https://hstspreload.org/

But…Network Segmentation…

AFP

Home Services About us Crimes Jobs News Centre Police Checks

28 JUNE 2024, 7:30AM

Media Release

Man charged over creation of ‘evil twin’ free WiFi networks to access personal data

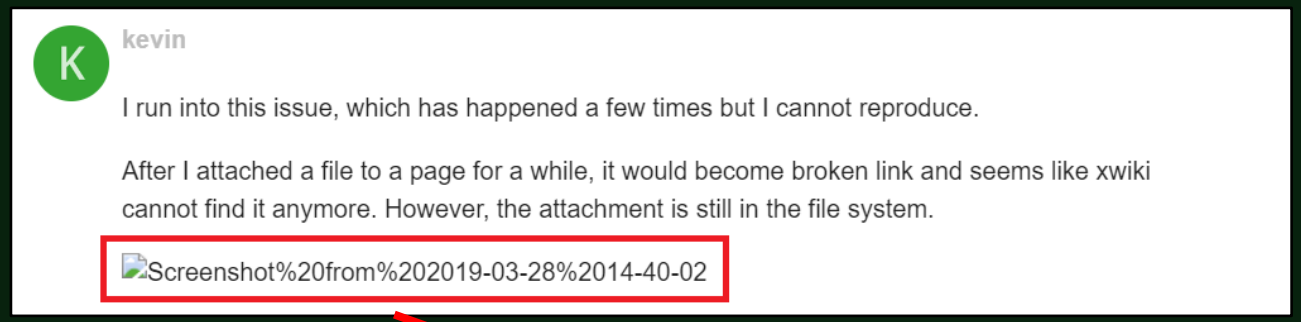
“On-the-wire” Defenses

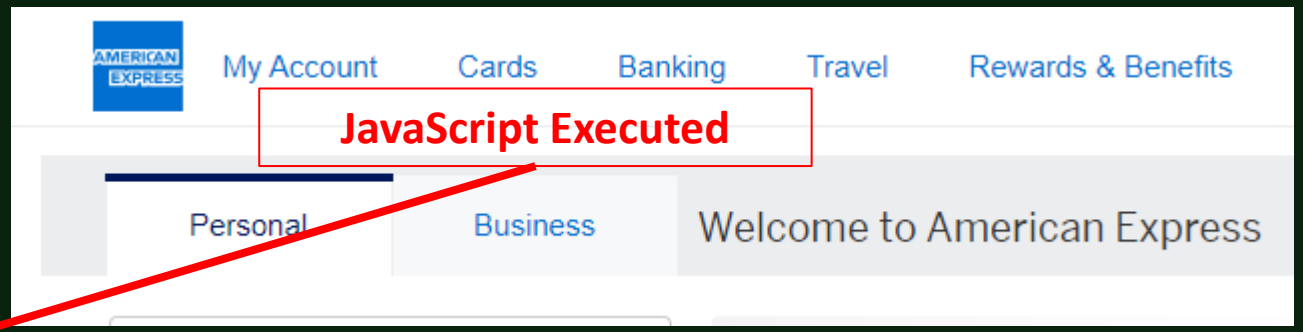
- ❑ Use HTTPS
- ❑ Set the “secure” attribute on all sensitive cookies
- ❑ Set the “Same-site: Strict” attribute on all sensitive cookies
- ❑ Return the “Strict-Transport-Security” response header
- ❑ List your site on the HSTS Preload List (https://hstspreload.org/)
- ❑ Disable HTTP completely

JavaScript Injection

(a.k.a. Cross-site Scripting)

Origins of “Cross-site Scripting”





JavaScript Injection (JSI) Defenses

```
4.1.2.6.  HttpOnly

   Syntax  Servers MUST NOT include a value.

   Semantics  The user agent SHOULD protect
confidentiality of cookies with the HttpOnly
attribute by not revealing their contents via
"non-HTTP" APIs.  (Note that this document does not
define which APIs are "non-HTTP".)
```

```
Set-Cookie:
session=03HMPAPjvLt8UISBQBnDrSSAAAAWVFR9gg41o;
domain=.example.com; secure; httponly; SameSite=None
```

```
x=new XMLHttpRequest();
u='https://example.com/api/cookietest’;
x.open('GET', u, true); x.send();
```

→ undefined

```
document.cookie
```

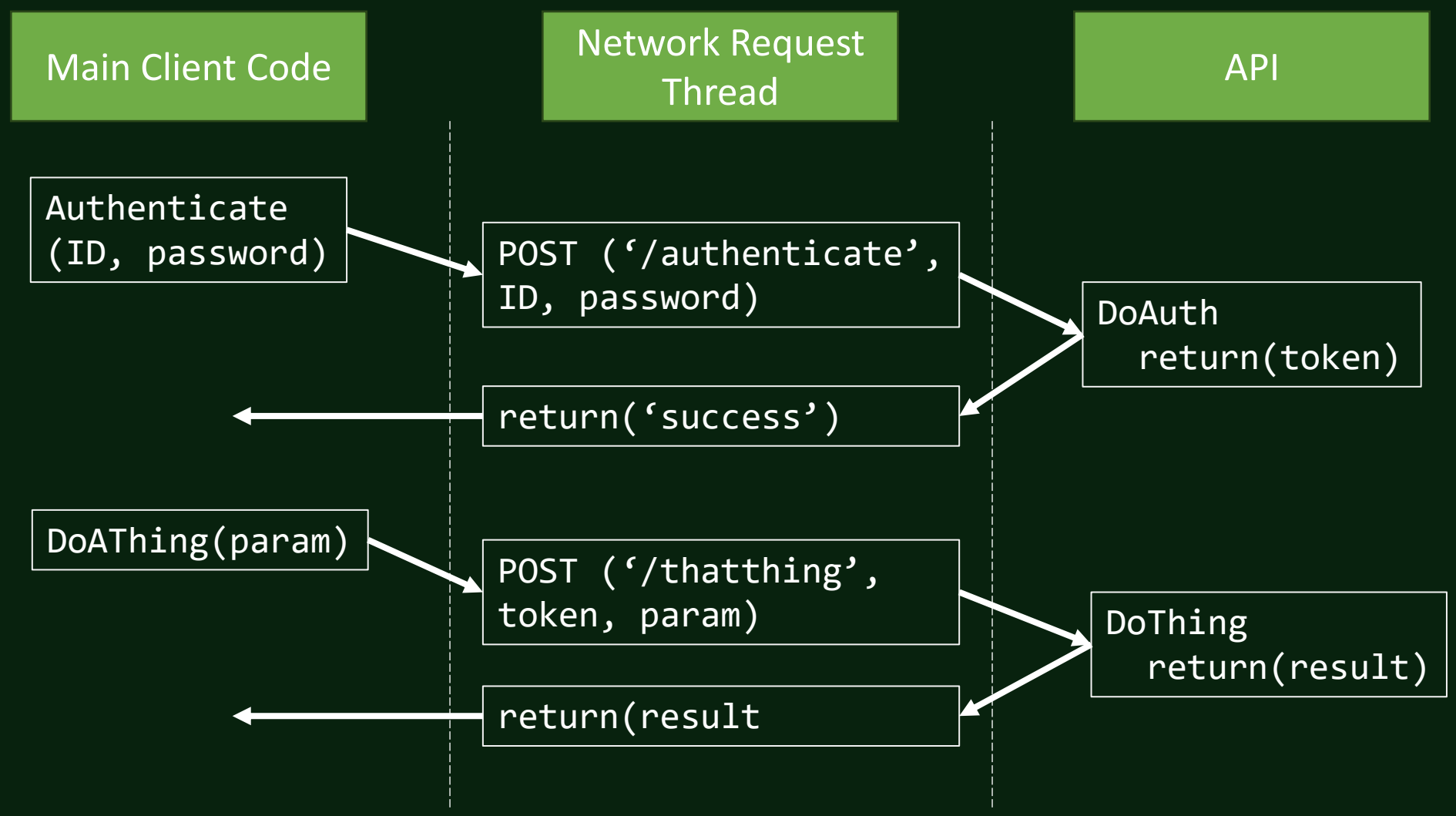
↓

```
GET /api/cookietest HTTP/2
Host: www.example.com
Cookie: session=03HMPAPjvLt8UISBQBnDrSSAAAAWVFR9gg41o
```

Not All Session Tokens are Cookies

```
Authorization: BEARER 03HMPAPjvLt8UISBQBnDrSSAAAAWVFR9gg
```

- Authorization headers are not handled automatically by the browser
  - Except for “basic” authentication (bad for different reasons)
- Client-side JavaScript MUST be able to manage the token
  - Best-practice: hide the token inside a worker thread

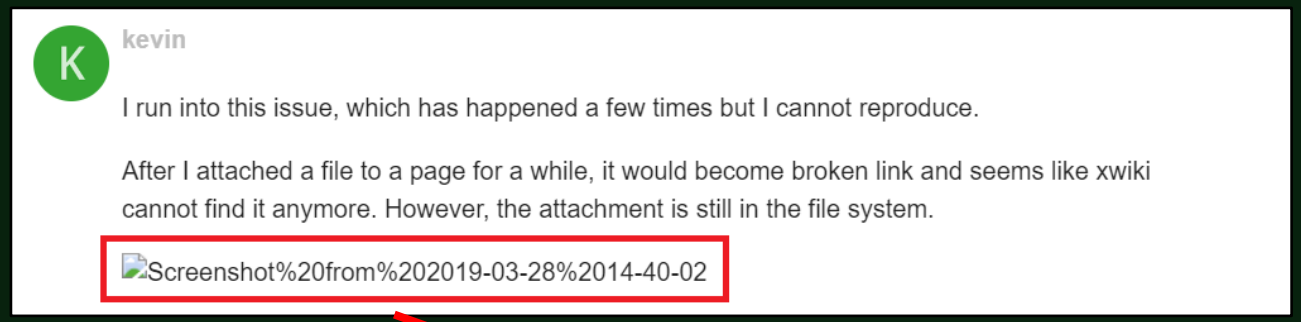

Main Client Code
Network Request Thread
API
Authenticate (ID, password)
POST ('/authenticate', ID, password)
DoAuth return(token)
return('success')
DoAThing(param)
POST ('/thatthing', token, param)
DoThing return(result)
return(result

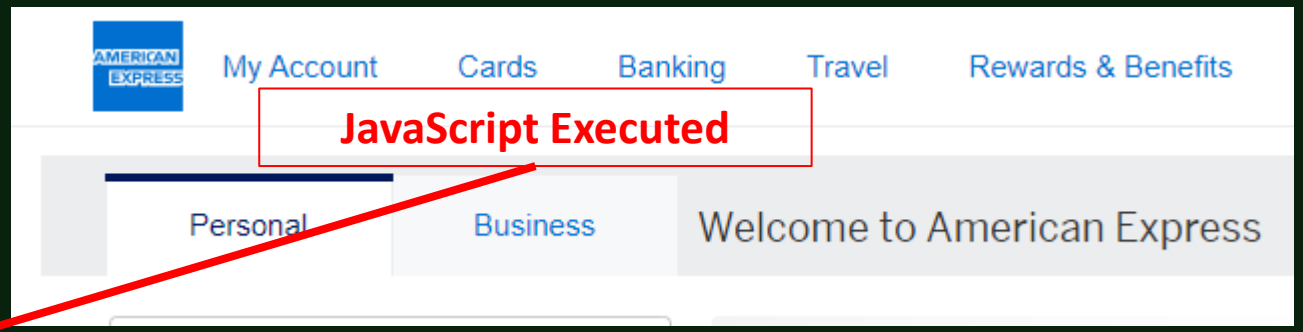
JavaScript Injection Defenses

- Set the “httponly” attribute on all sensitive cookies
- Isolate header token values in worker threads
- Don’t be vulnerable to JavaScript Injection
  - Classic Input Validation and Output Encoding
  - Use a tailored Content Security Policy

Blind Session Attacks

No token. No problem?





/admin/addEditor?docId=21&userId=42

CSRF Requirements - Then

Condition	Defense
The target site must use cookies for authentication	Authorization Header **Anti-CSRF Token in Header** **Same-Site Attribute on session cookie**
The endpoint must use the GET or POST methods	Using UPDATE or DELETE methods
The endpoint must use only predictable parameters	**Require an anti-CSRF token as a parameter**
The results must be state-changing	Above defenses can in theory be applied only to requests that change state.

CSRF Requirements - Now

Condition	Defense
The target site must set **Same-Site: None** attribute on the session cookie OR The attack must originate from the same "origin"	**DON'T DO THIS**

The JavaScript Injection Loophole

If you have a JavaScript Injection vulnerability on your site…

ALL IS LOST

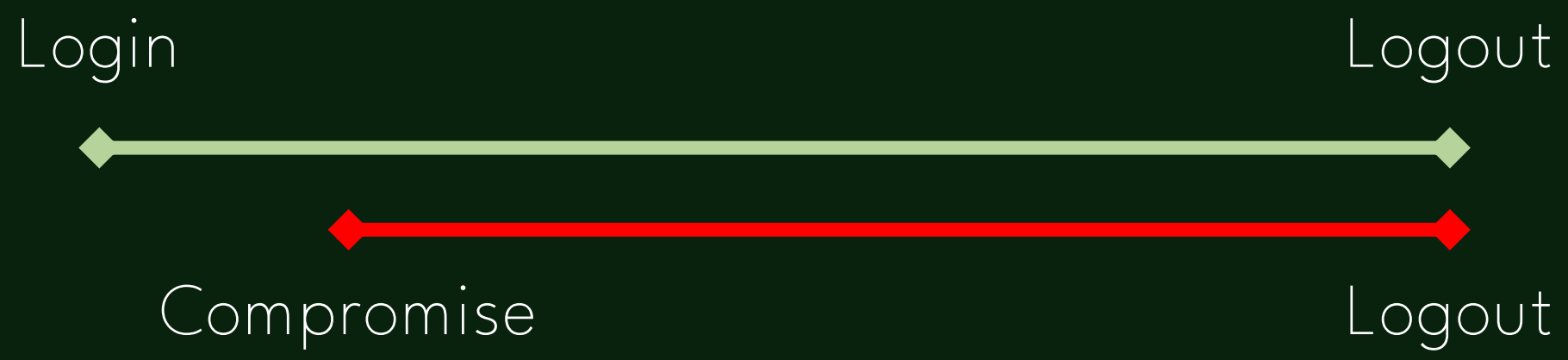
- Since the request comes from the same site
  - Cookies will always be sent
  - JavaScript is allowed to read page contents to retrieve CSRF tokens
  - JavaScript is allowed to add custom request headers (authentication and anti-CSRF)
  - JavaScript can call worker threads to submit protected tokens

Clickjacking / UI Redressing

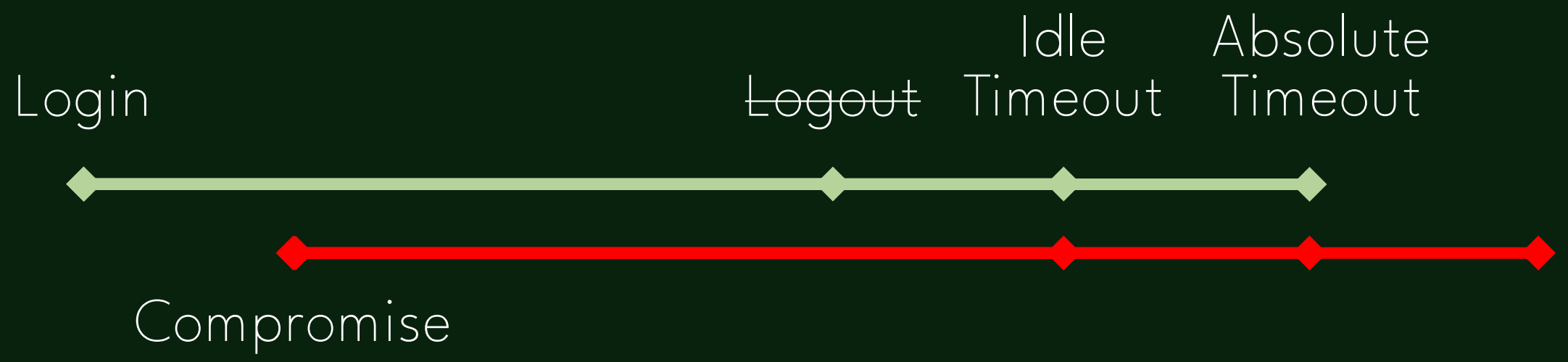
- Technically a risk
  - Facebook got hit with a number of attacks around 20xx
- Very rarely seen in web apps
- More of an issue in mobile apps, but sessions are different inside an application
- Simple to prevent
  - X-Frame Options Header
  - Frame-Ancestors Content Security Policy Directive

Post-Compromise Defenses

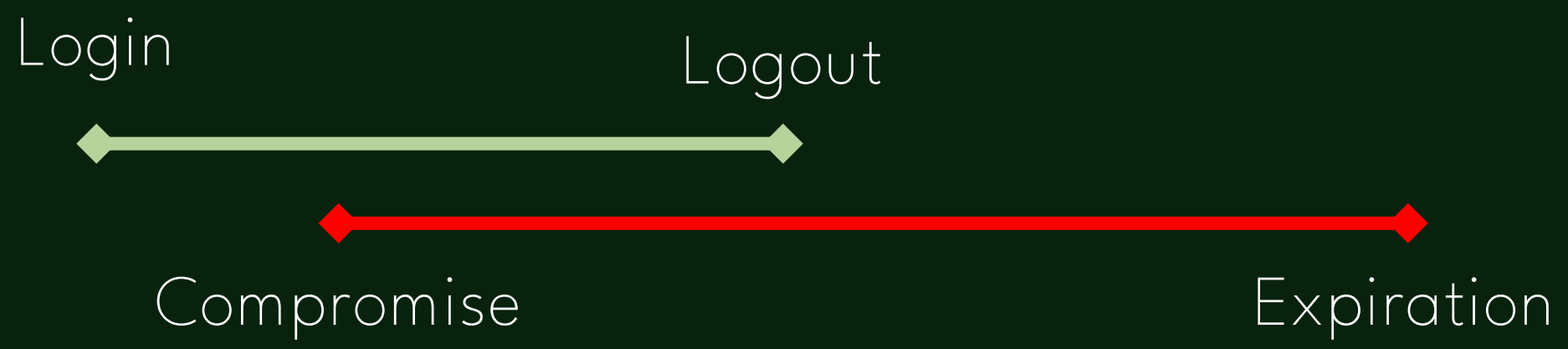
Lifetime of a Session Token



Session Lifetime: User doesn't log out



Why JWTs are ~~BAD BAD BAD~~ Slightly Worse

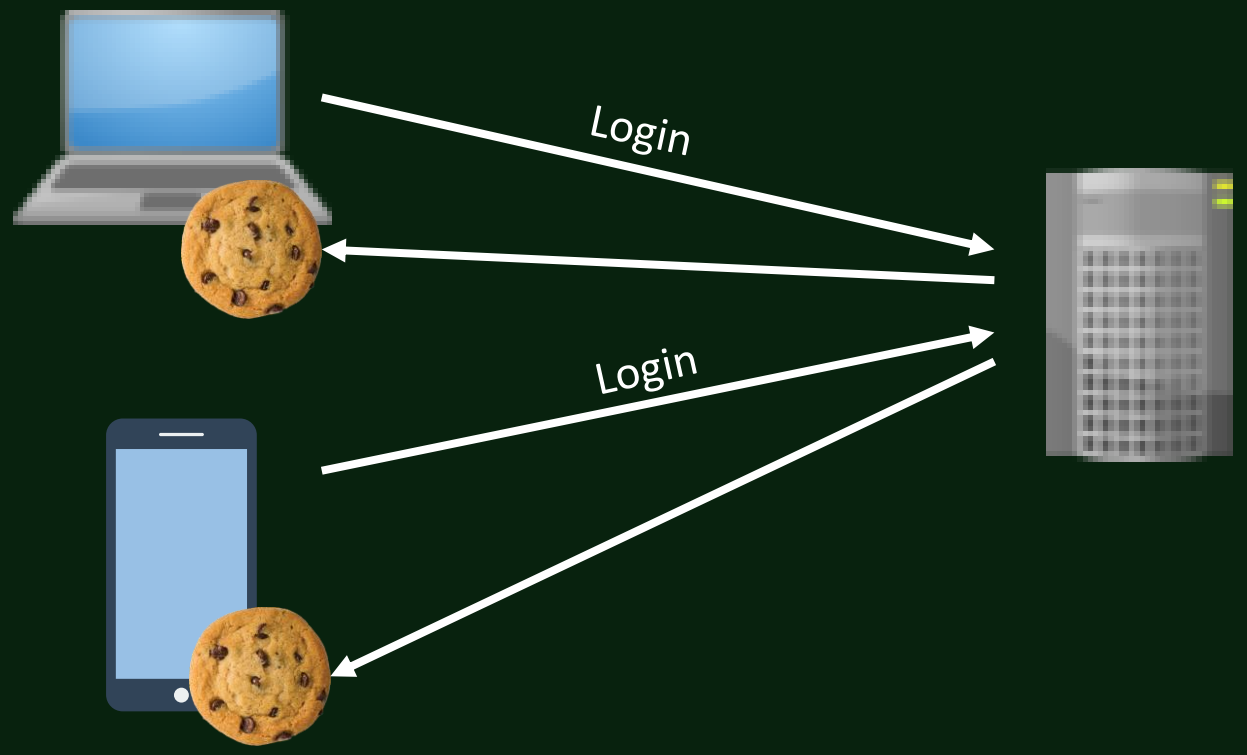


Session Length Summary

- Make your idle and absolute session timeouts as short as the business will allow
- Allow multiple sessions only if the use case requires it
- If multiple sessions are allowed, show active sessions to the user somewhere
- Implement a “Last Login” message
- Require reauthentication before doing the most sensitive tasks

State of the Art

Cross-device tokens aren't a thing

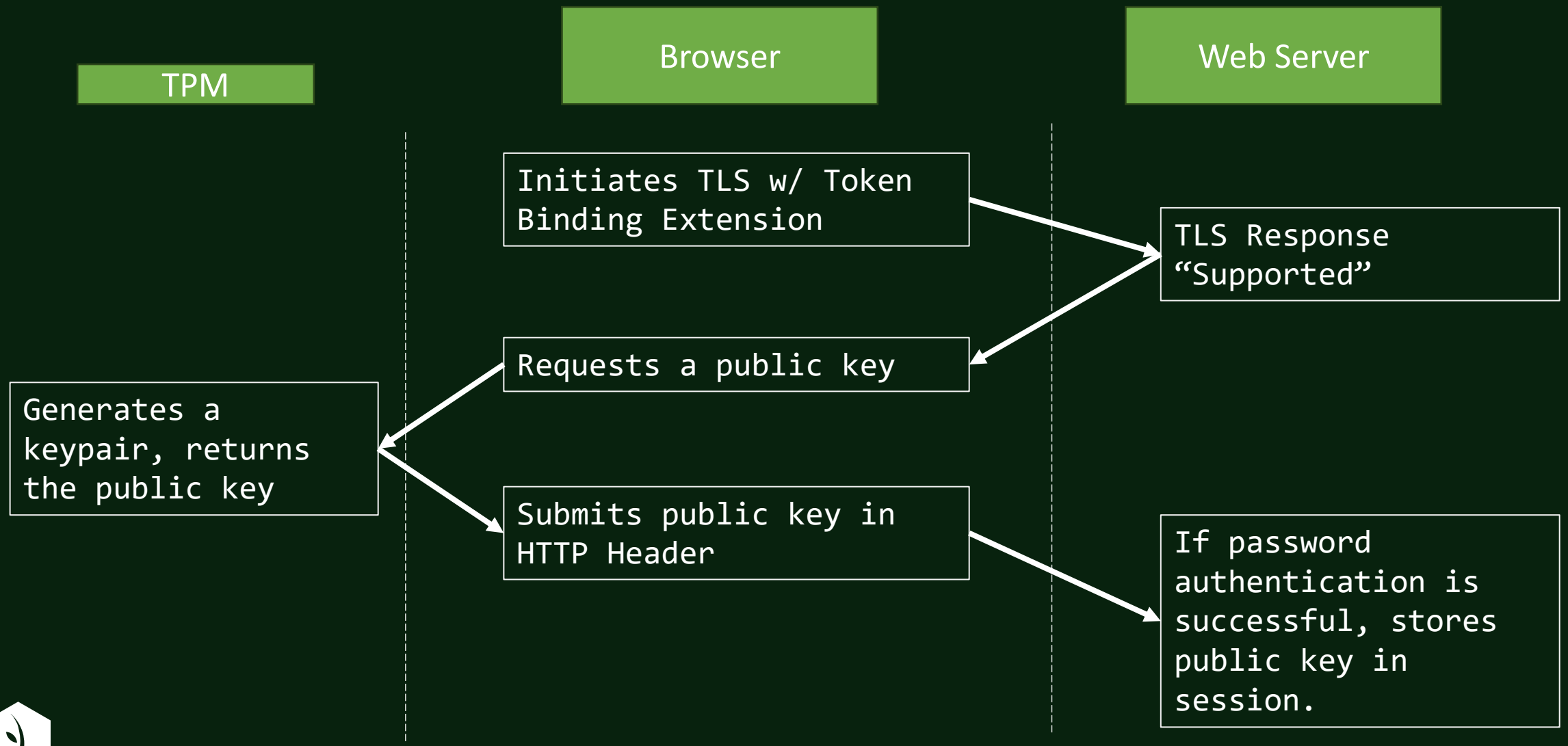


Token Binding (Microsoft 2016)

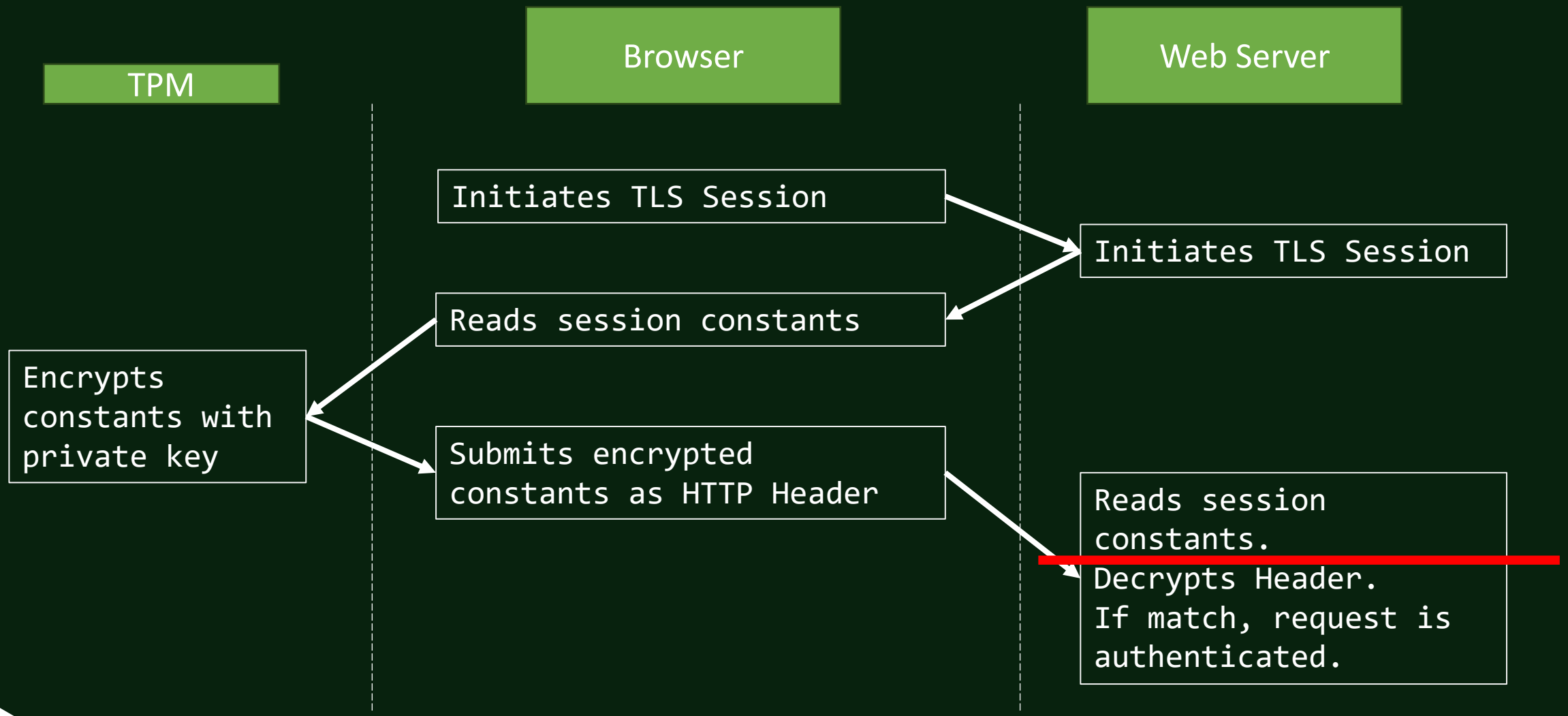
Pub/Priv Key Pair | HSM/TPM | TLS Extension | HTTP Header

1. Client initiates the TLS connection with the “token binding” extension set to active and the server responds that it is supported.
2. Client generates a public/private key pair for this specific server/session and stores the private key in the TPM.
3. Client sends the public key in an HTTP Request header during the authentication process.
4. If authentication is successful, the server stores the public key with the session.
5. On future requests, the client takes random constants exchanged during TLS initiation, encrypts them with the private key, and sends the encrypted values to the server.
6. The server uses the client’s public key to decrypt the values and compares them with the random constants it also knows to verify the session.

Token Binding – Session Initiation



Token Binding – Future Requests



Device Bound Session Credentials (Google 2024)

1. The user signs in to the website.
2. The server returns an HTTP header with a challenge, and token management endpoints.
3. The browser generates a public-private key pair with the private key stored in the TPM.
4. The browser POSTs the signed challenge and public key to a “startsession” endpoint.
5. The server returns a session token with a “short” lifetime.
6. When the token expires, the browser requests a new challenge from the “refresh” endpoint.
7. The challenge is signed and resubmitted to get a fresh token.

DBSC Specification Disclaimer

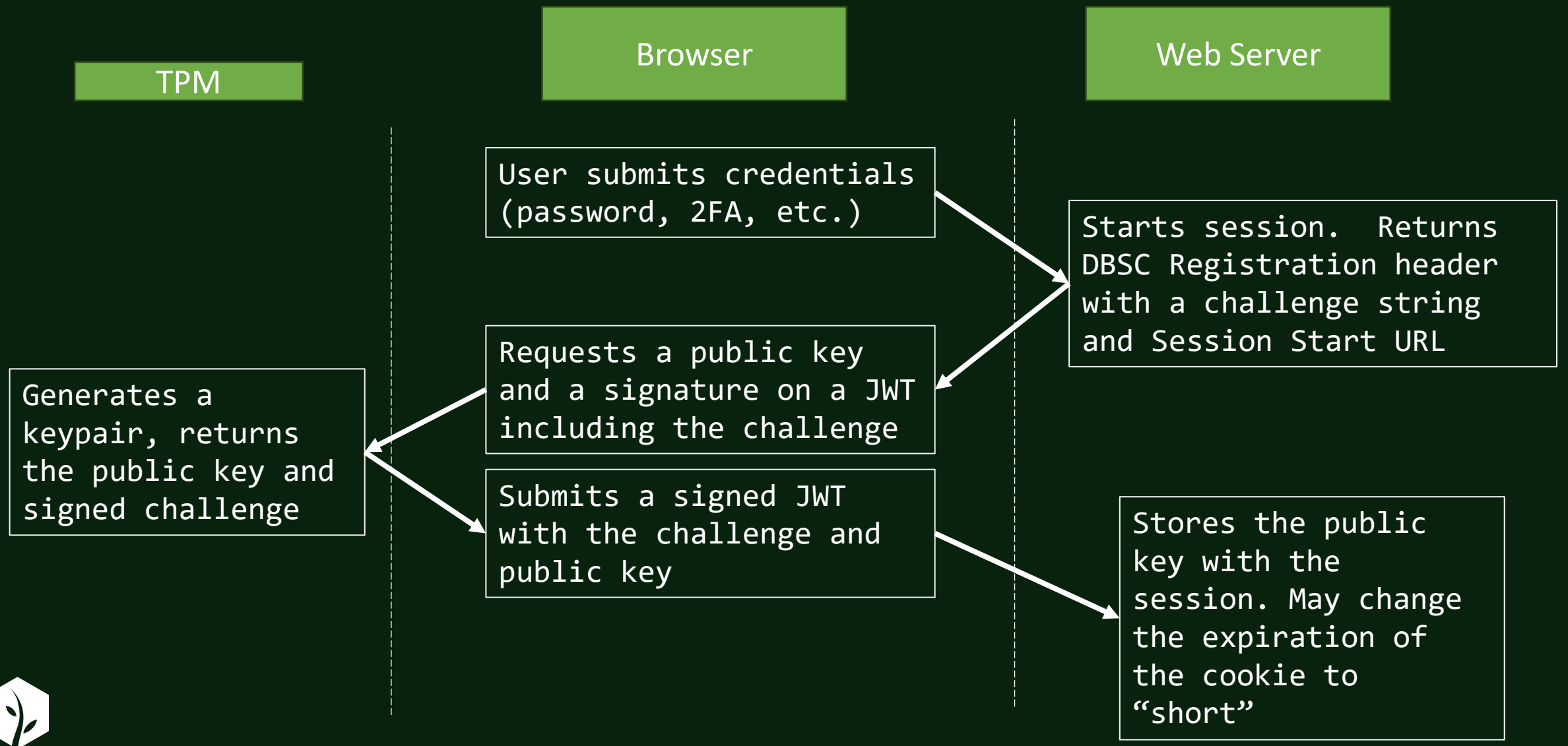
§ 2.1. Non-goals

DBSC will not prevent temporary access to the browser session while the attacker is resident on the user's device. The private key should be stored as safely as modern operating systems allow, preventing exfiltration of the session private key, but the signing capability will likely still be available for any program running as the user on the user's device.

DBSC will also not prevent an attack if the attacker is replacing or injecting into the user agent at the time of session registration, as the attacker can bind the session either to keys that are not TPM bound, or to a TPM that the attacker controls permanently.

DBSC is not designed to give hosts any sort of guarantee about the specific device a session is registered to, or the state of this device.

DBSC – Session Initiation



Sample Response Headers AFTER Credential Validation

Supported Encryption Modes

DBSC Registration Endpoint

Sec-Session-Registration: (ES256 RS256); path="/internal/StartSession"; challenge="ycwmtuwvmr"; authorization="auth-code-123“

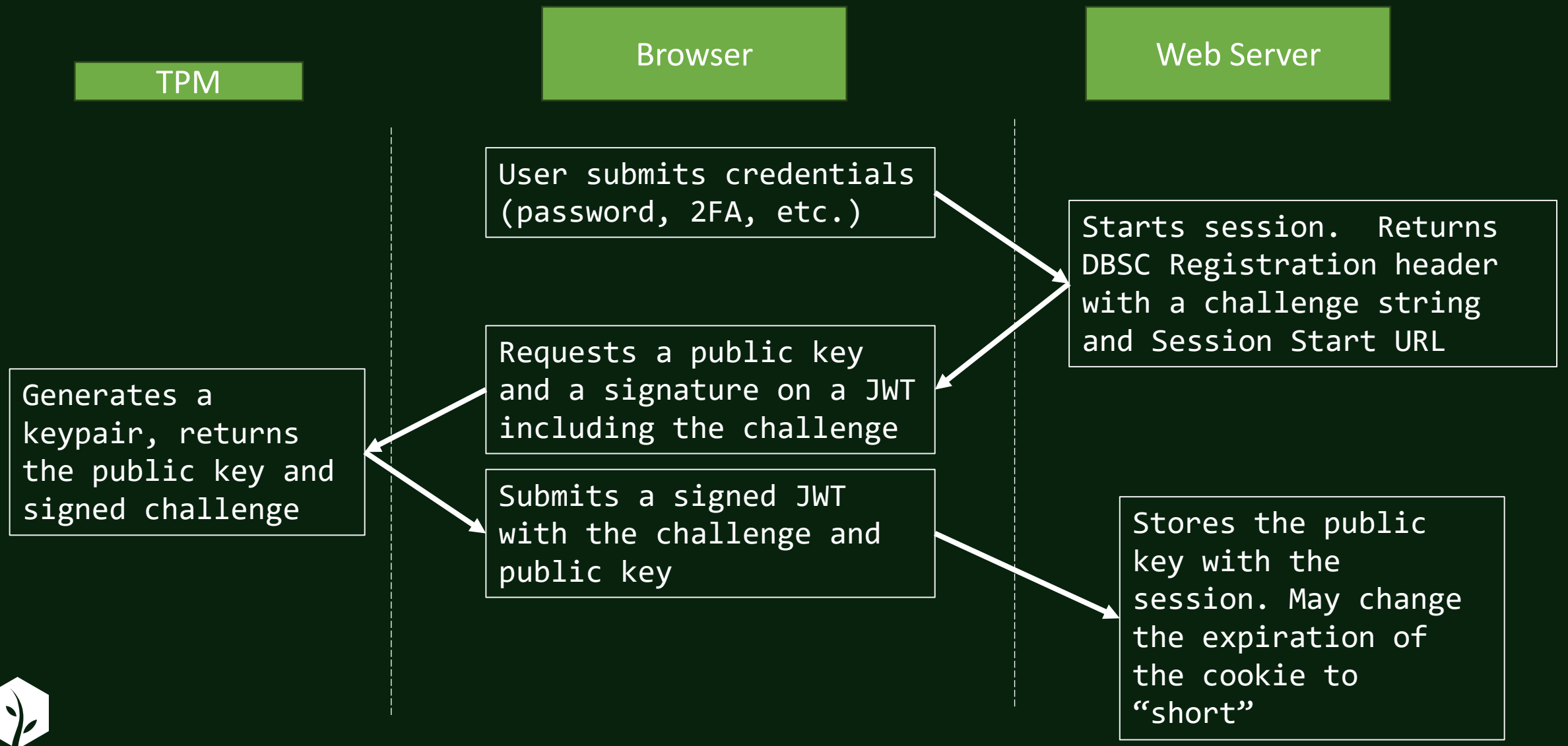
Challenge Text

Registration ID

Set-Cookie: sessionID=b145181c4cbd4321841bee21d5a876e6; Domain=drubery-dbsc-test-server.deno.dev; Path=/

Normal Session Cookie – Long Lived?

DBSC – Session Initiation



JWT Payload Submitted to DBSC Registration Endpoint


```
Payload = {
  "aud": "https://drubery-dbsc-test-server.deno.dev/internal/StartSession",
  "authorization": "auth-code-123",
  "iat": 1753291358,
  "jti": "ycwmtuwvmr",
  "key": {
    "crv": "P-256",
    "kty": "EC",
    "x": "2W83hQlY13w4fwyR8_W9D7cvx_HrnsSNVHYqc109TLA",
    "y": "J-2qGmNIio9CtN07Zsxab2Sy2kqivFAIeJH7LiNeyWY"
  }
}
```


Registration ID

Session ID / Challenge

Public Key

JSON Response Body Returned from DBSC Registration Endpoint

```
{ "session_identifier": "ycwmtuwvmr",
 "refresh_url": "/RefreshEndpoint",
 "scope": {
  "origin": "https://example.com",
  "include_site": true,
  "scope_specification": [
   { "type": "exclude", "domain": "*.example.com", "path": "/static" }
  ]},
 "credentials": [{
  "type": "cookie",
  "name": “sessionID",
  "attributes": "Domain=example.com; Secure; SameSite=Lax"
}}]
```

Session ID / Challenge

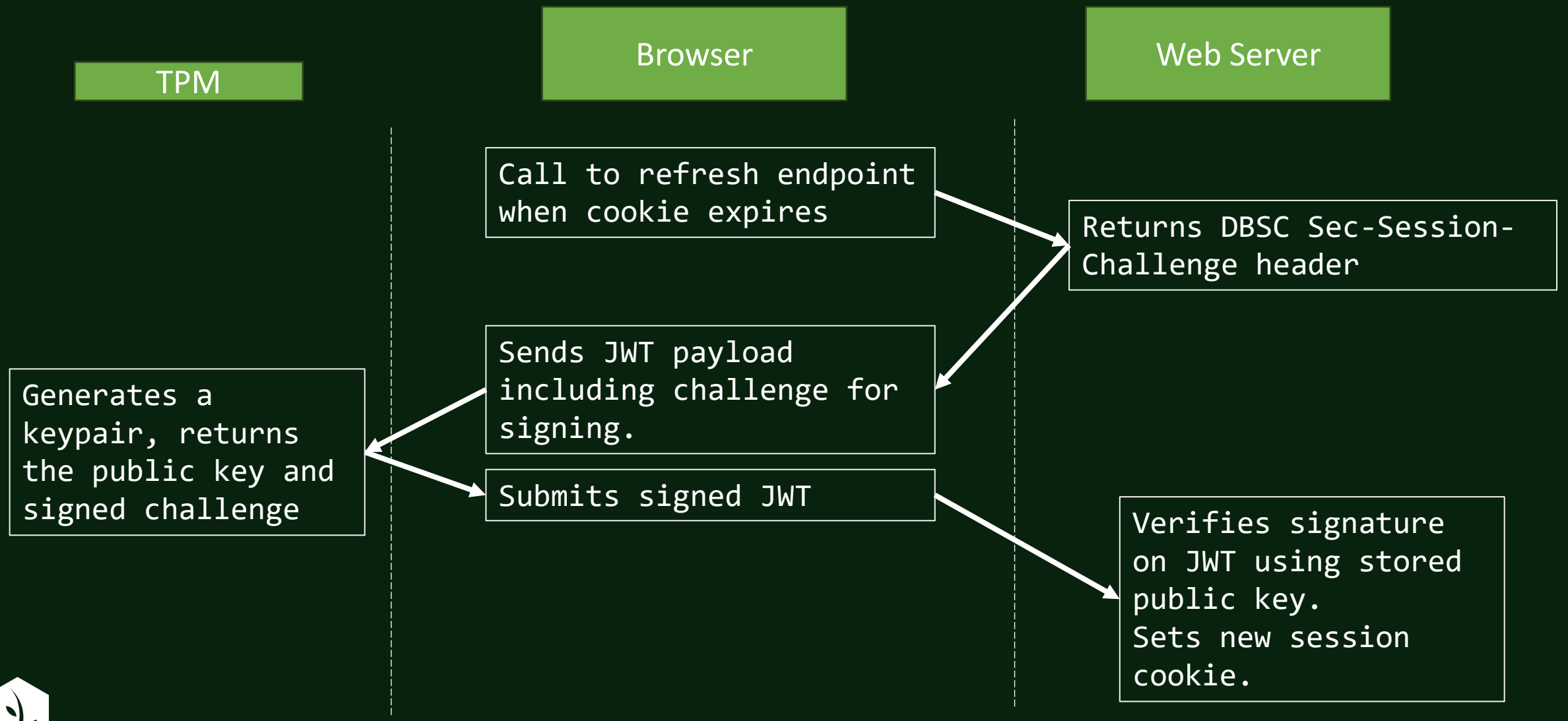
Refresh Endpoint

Paths to be protected or excluded

Cookies protected by this DBSC session

Also sets a short-lived session cookie named as indicated in the response

DBSC – Session Refresh



Tips if you are trying to implement a PoC

- A valid HTTPS certificate is required
- You must “enable” the following Chrome flags
  - enable-bound-session-credentials
  - enable-bound-session-credentials-software-keys-for-manual-testing
  - enable-standard-device-bound-session-credentials
  - enable-standard-device-bound-session-persistence
- DBSC network requests aren’t shown in developer tools
  - Can be captured using Burp, ZAP, etc. if proxy certificate is trusted
  - Can be viewed in
    - chrome://net-export/
    - https://netlog-viewer.appspot.com/

Challenges for DBSC Adoption

- Requires implementation by application developers of two custom endpoints
  - Could be built into frameworks
- Having the browser take over some portions of session refreshing can be confusing
- Debugging is difficult
  - Reference implementations are needed

Thank You!

Slides and
Articles Suitable for Sharing with Developers
https://www.merisec.com/blog

Mark Hoopes
mark@meristeminfosec.com
https://www.linkedin.com/in/markhoopes/