Device Bound Session Credentials

Boulder Ruby Meetup
April 2026

Mark Hoopes

whoami

- Worked for 15 years in Enterprise IT delivery - mainly web-hosting/web development
- 10+ years ago pivoted into Information Security – mainly web application pentesting
- 3 years ago started as an independent consultant.

I love puzzles, travel, long walks in the forest, and home automation

What do you want to sell?

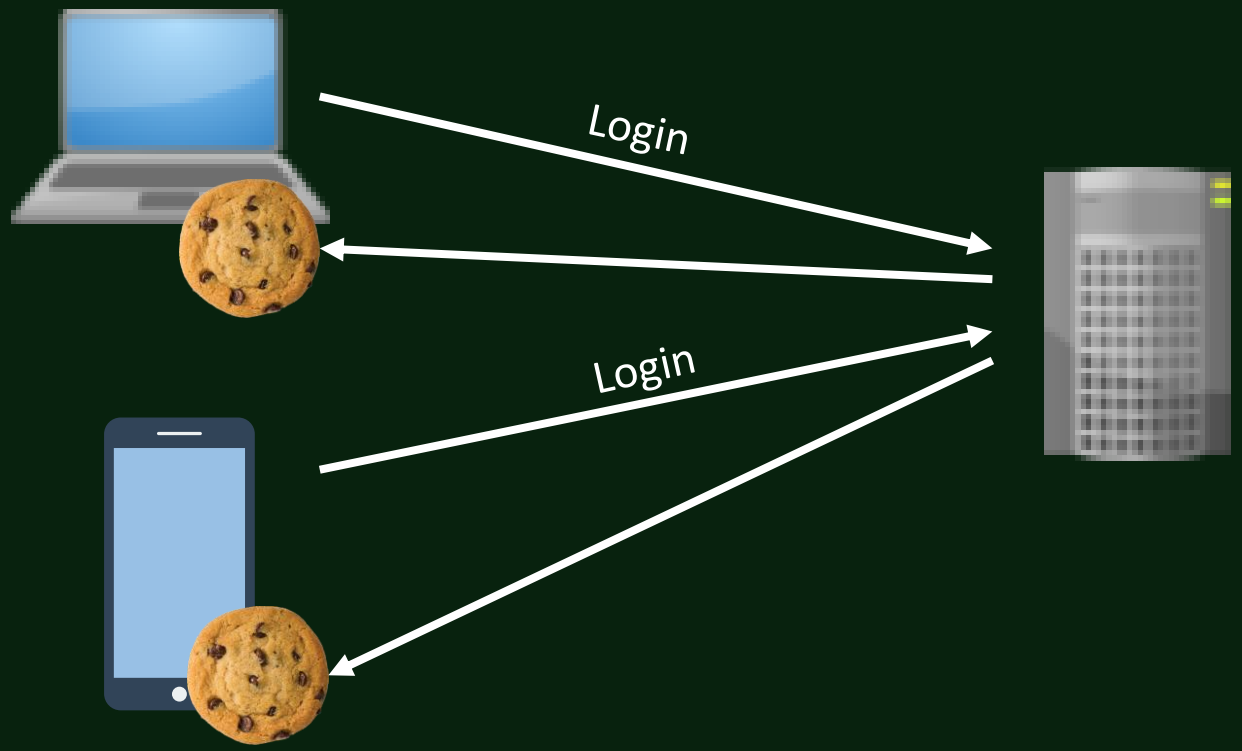
Cookies and logins

[discussion about this not being the marketplace]

Sell away!

How do we keep session tokens from being sold on the “Dark Web”?

Cross-device tokens aren't a thing



Token to Device Mapping - Simple Approaches

IP Address	What about corporate networks with multiple exit points?
User-Agent Header / Any Header	Spoofable by the attacker, if they capture it ahead of time
Deep Device Profiling	Makes users sad because it is abusable for cross-site tracking

Enter the TPM

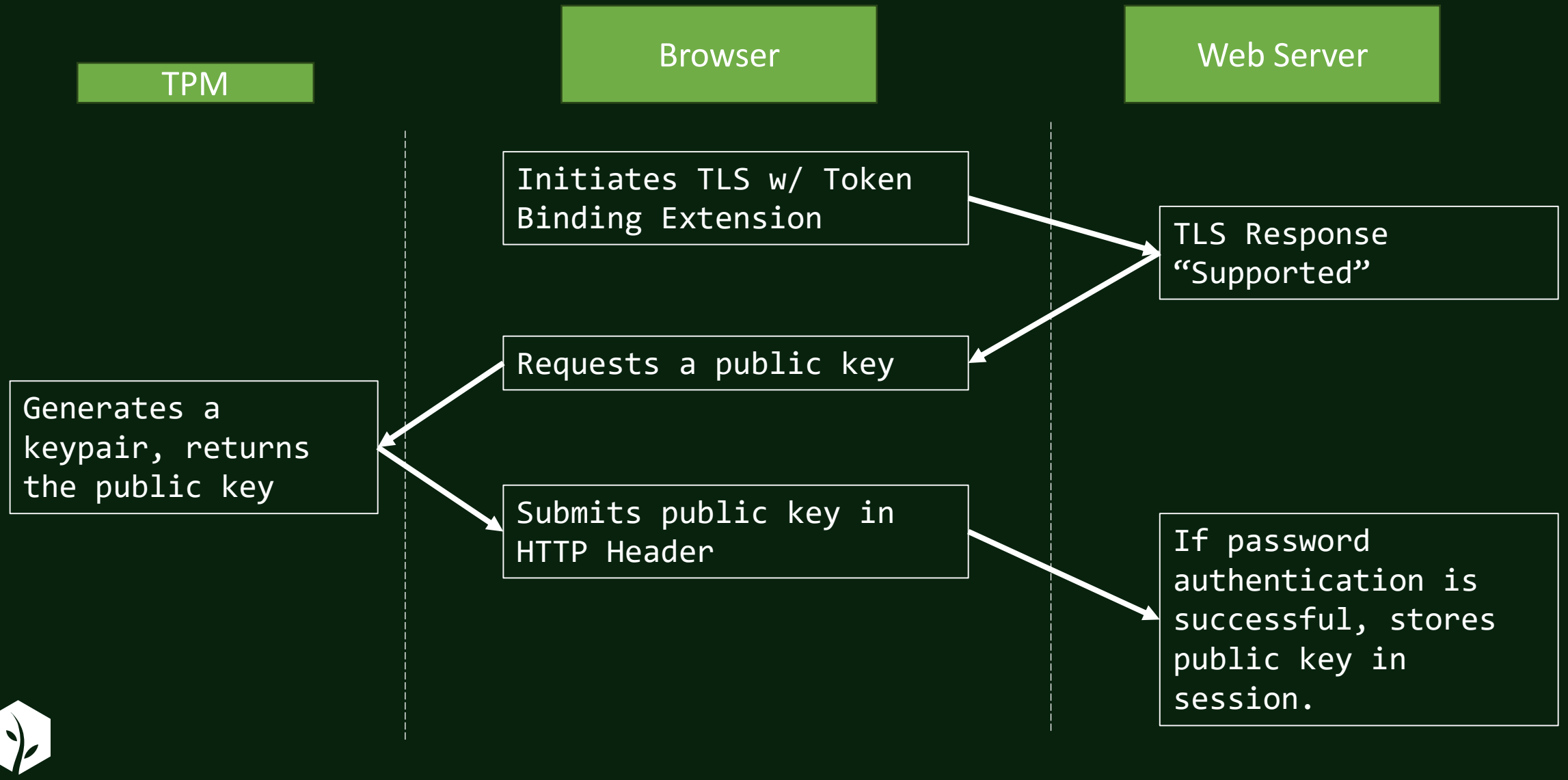
- Trusted Platform Module
  - WILL generate a public / private key pair
  - WILL return the public key to be shared
  - WILL sign data using the private key
  - WILL NOT return the private key

Token Binding (Microsoft 2016)

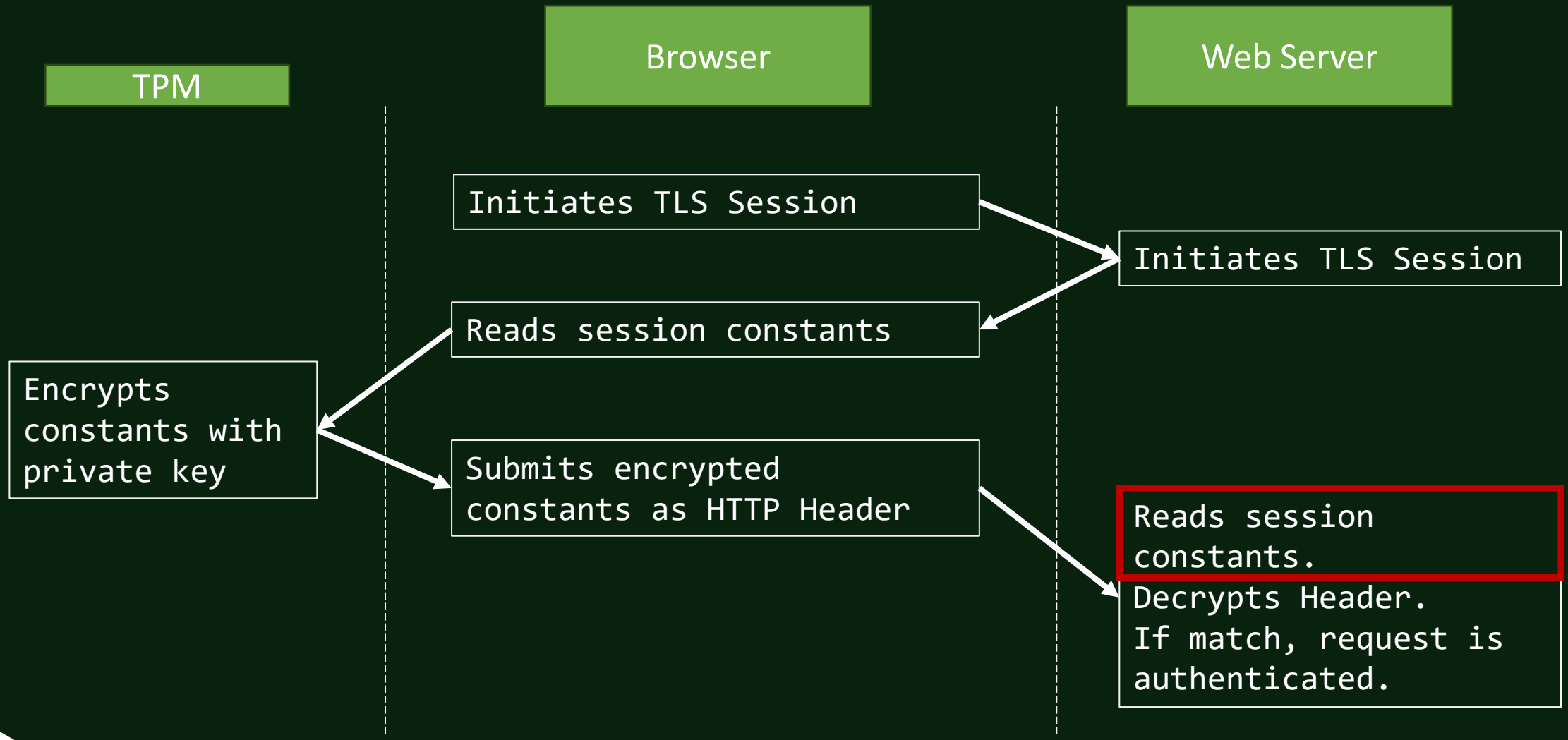
Pub/Priv Key Pair | HSM/TPM | TLS Extension | HTTP Header

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



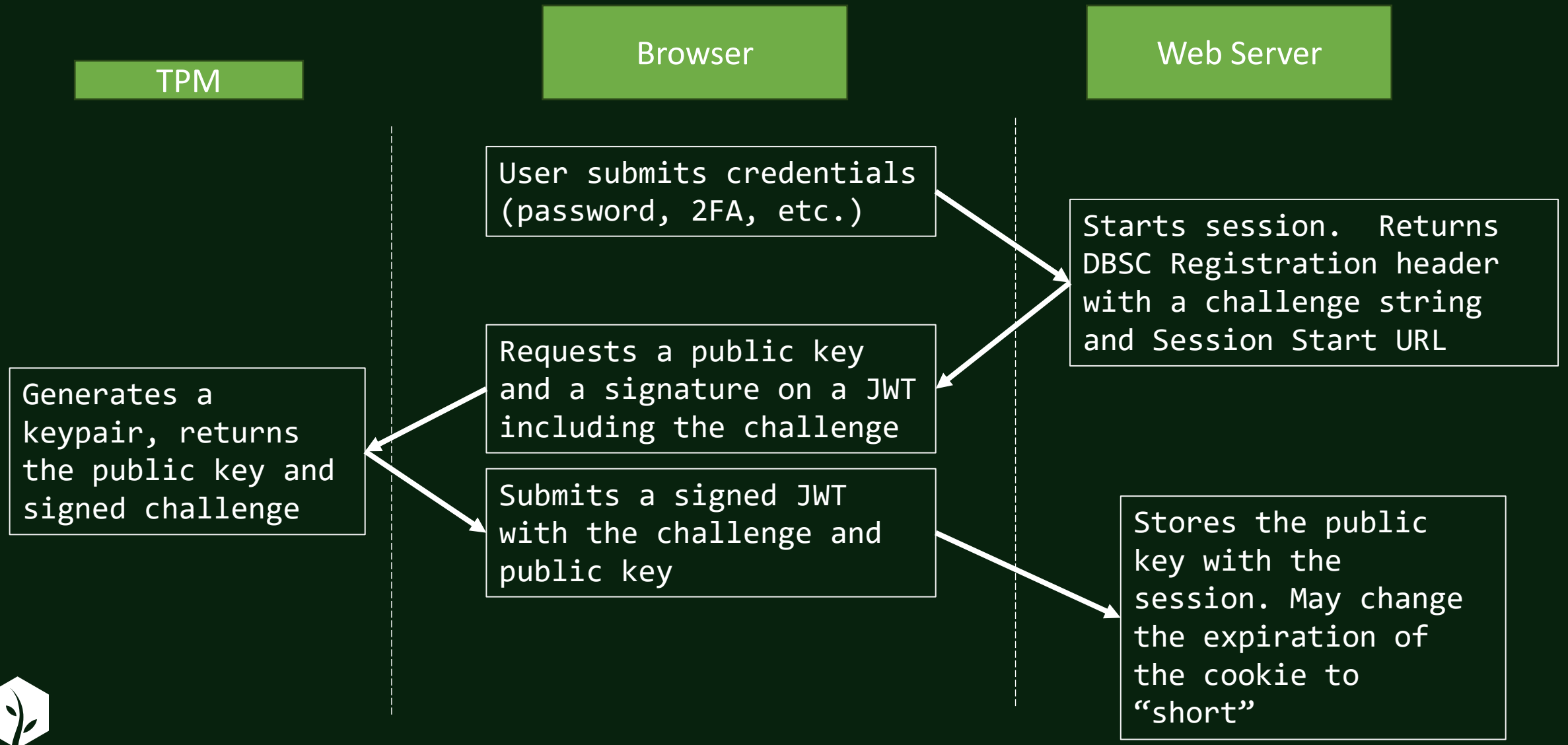
Token Binding – Future Requests



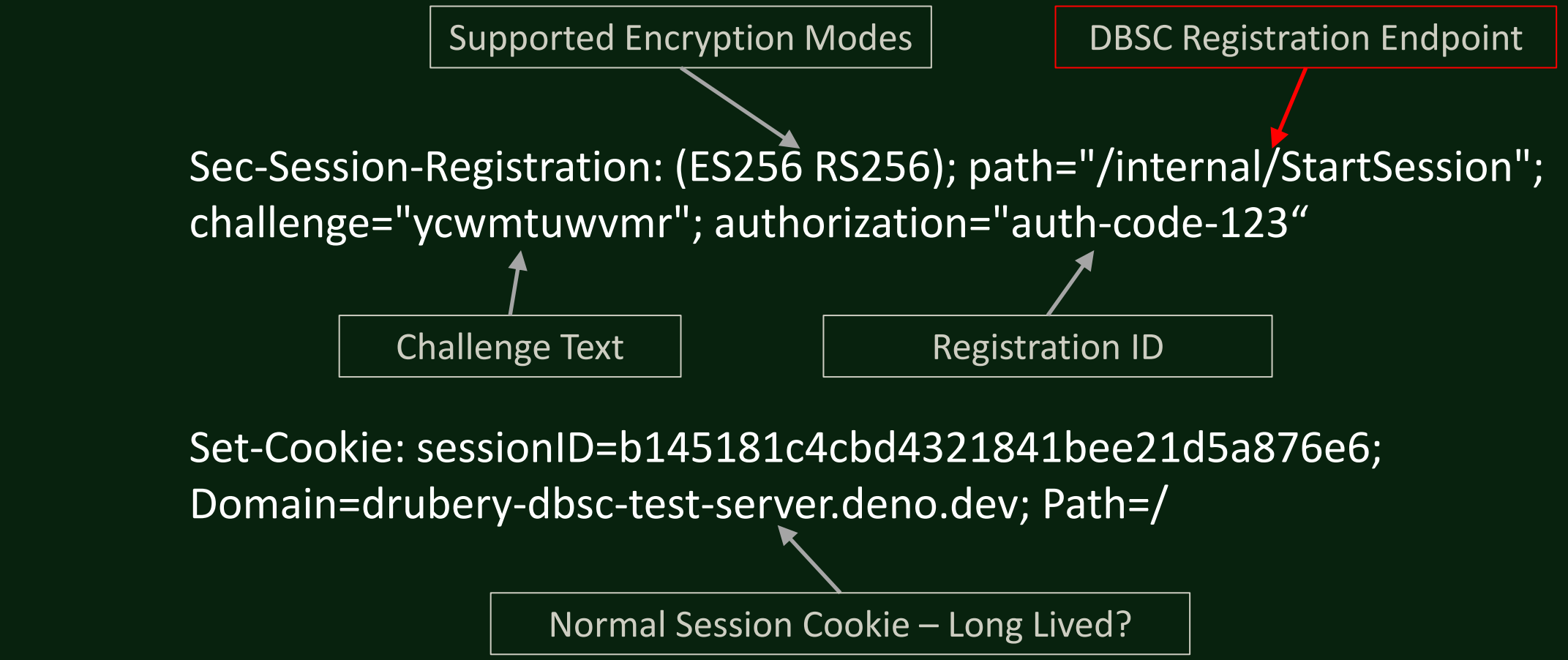
Device Bound Session Credentials (Google 2024)

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 – Session Initiation



Sample Response Headers AFTER Credential Validation



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

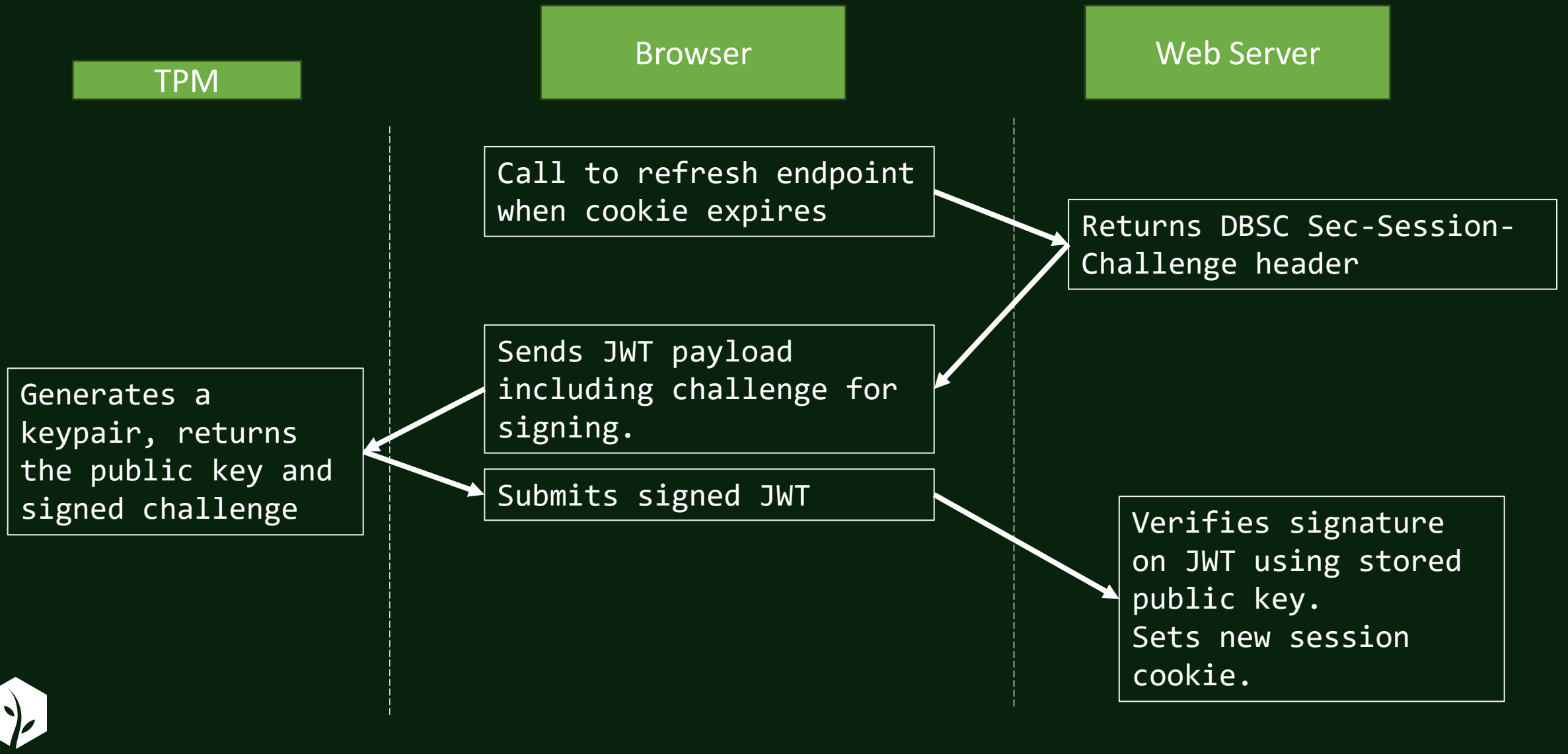
Refresh Endpoint

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



Summary of DBSC Responsibilities

- The BROWSER automatically recognizes the server DBSC response header
- The BROWSER handles all interactions with the TPM
- The SERVER must implement the “registration” and “token refresh” endpoints
- The BROWSER handles refreshing the token once it is expired

DBSC Current Status

- Chrome contains an implementation, but it must be enabled as a development feature
  - Additional server criteria must also be met
- Mozilla and Safari are “evaluating” the technology through their W3C participation
- Demonstration websites have been published by Google
- No reference implementation code has been published

Mostly Crickets

Thank You!

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

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

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.

Lifespan	
Credential	Months – Years
Token	Minutes - Weeks

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-2qGmNlio9CtN07Zsxab2Sy2kqivFAIeJH7LiNeyWY"
  }
}
```

Registration ID

Session ID / Challenge

Public Key

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/