Draft proposals for supporting local HTTPS without publicly trusted certificates

TOSHIBA

Ajitomi, Daisuke
Toshiba Corporate R&D Center

Index

- Solutions/Proposals based on genuine Web PKI certificates
  - PLEX
  - Mozilla IoT Gateway
  - Igarashi-san's proposal (Technically Constrained Subordinate CA Certificates, etc.)

- Proposals NOT based on genuine Web PKI certificates
  - "HTTPS for Local Domains" by Martin Thomson (Mozilla)
    - Extended origins for non-unique names
      https://github.com/httpslocal/proposals/issues/1

  - "Supporting Local HTTPS communication" By Ajitomi, Daisuke (Toshiba)
    - https://github.com/httpslocal/proposals/issues/2

Issues on Web PKI based solutions

1. Responsibility for device (local server) authenticity
   - Can Web PKI root CAs guarantee the authenticity of each device?
2. Readability/Memorability of domain names
   - Globally unique names for local devices might not be readily and memorable.
   - Local domain names (.local, .home.arpa) indicate that the device is located in local network.
3. Scalability/Feasibility
   - For example, are CT logs required for the large number of devices?
4. Privacy
   - Globally unique names and assigned IP addresses are registered to public DNS servers.
5. Offline availability
   - When there is no Internet access, name resolution will fail

Solutions described in this presentation are intended to solve some of the issues above.

“HTTPS for Local Domains” by Martin Thomson

- https://docs.google.com/document/d/170rFC91jqvpFrKIqG4K8Vox8AL4LeQXzfikBQXYPmzU/edit?usp=sharing

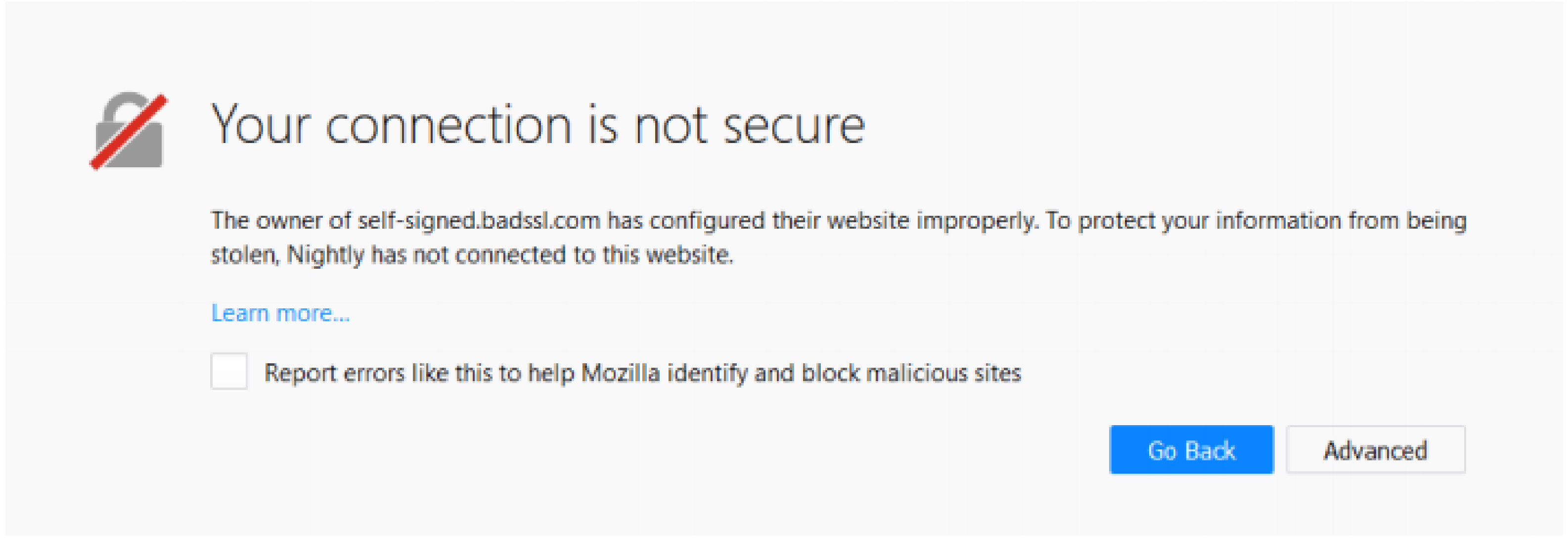
“HTTPS for Local Domains” by Martin Thomson

- Problem Statement
  - “The key challenge for local domains is that names are not unique. Any printer is entitled to claim the name printer.local and use it. Thus, any server has a legitimate claim to that name. If we are to make https://printer.local a viable origin, how do we ensure that it is unique?”

- Solution
  - “Extending the origin makes this possible. A web origin is defined as a tuple of scheme, host and port.” …
  - “By adding an additional property to that tuple that cannot be used by any other server, we ensure that the tuple is able to uniquely identify that server. For this, the public key of the server is ideal.”

“HTTPS for Local Domains” – User Interface

This is the current view when we access a device that offers untrusted certificate



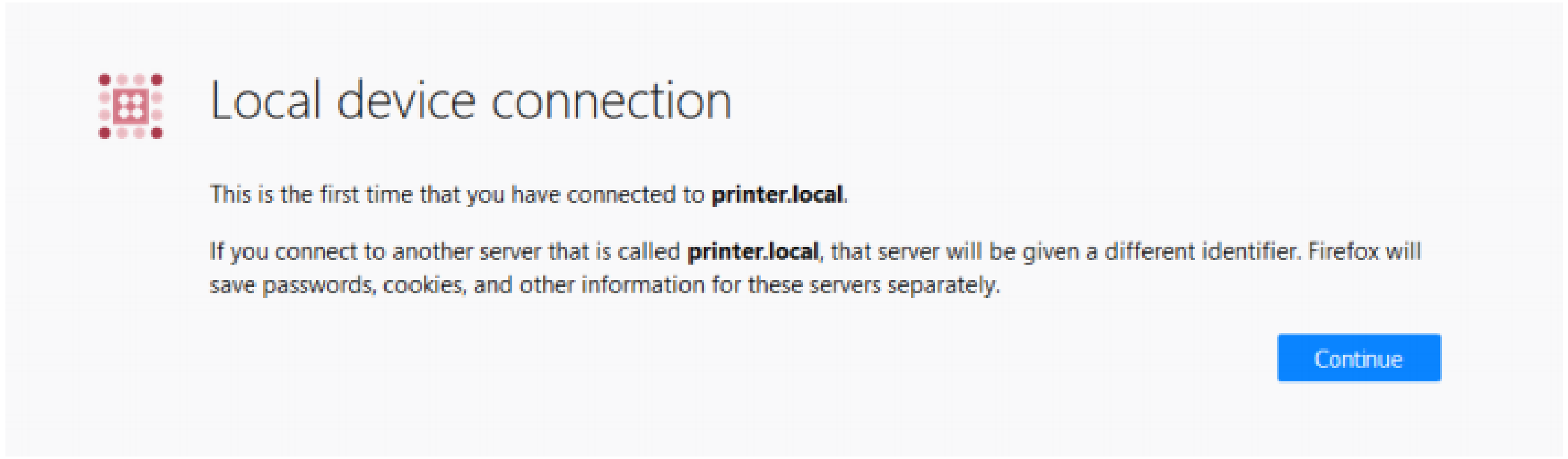
Your connection is not secure

The owner of self-signed.badssl.com has configured their website improperly. To protect your information from being stolen, Nightly has not connected to this website.

Learn more...

Report errors like this to help Mozilla identify and block malicious sites

When a local server is contacted the first time, the following is shown instead[2]:



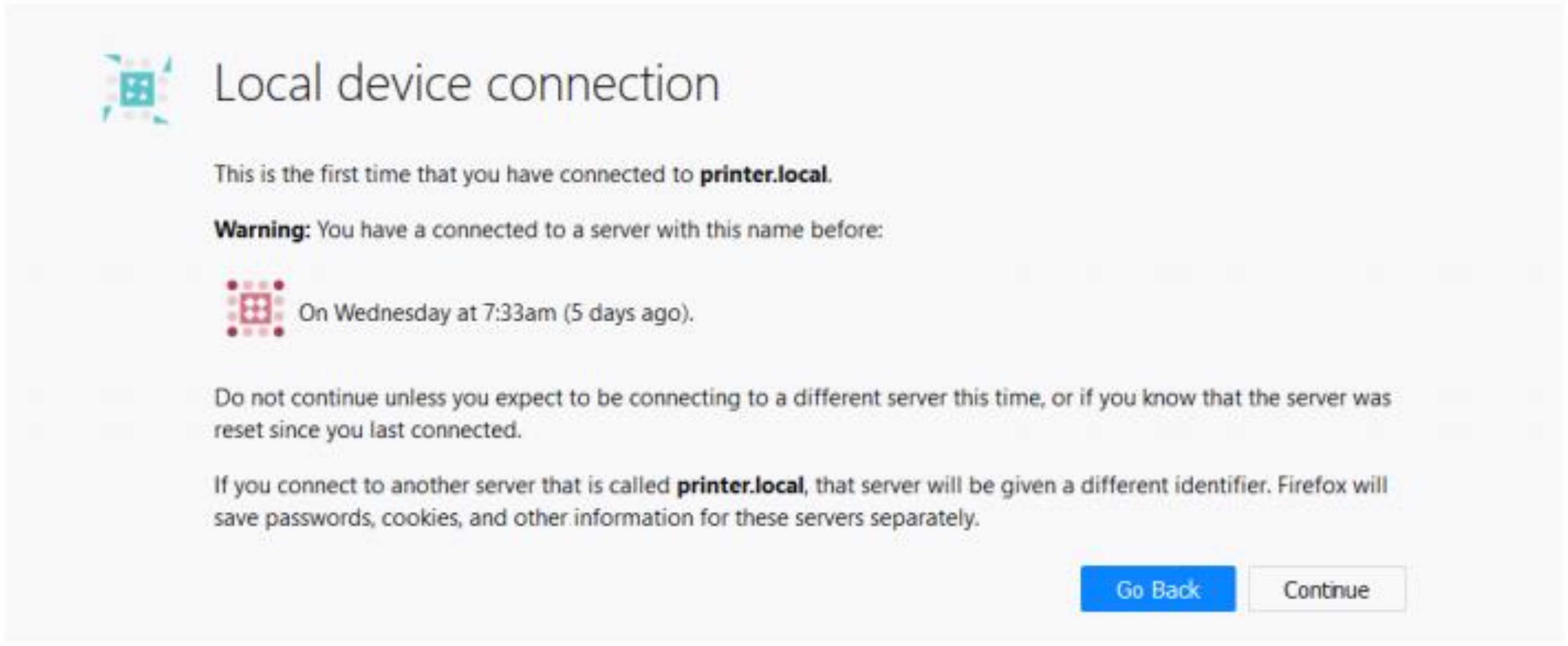
Local device connection

This is the first time that you have connected to **printer.local**.

If you connect to another server that is called **printer.local**, that server will be given a different identifier. Firefox will save passwords, cookies, and other information for these servers separately.

“HTTPS for Local Domains” – User Interface

To avoid the potential for confusion about identity, a different warning is shown if the same name was previously seen. This shows some information about previous connections to the same name:



Local device connection

This is the first time that you have connected to **printer.local**.

Warning: You have a connected to a server with this name before:

On Wednesday at 7:33am (5 days ago).

Do not continue unless you expect to be connecting to a different server this time, or if you know that the server was reset since you last connected.

If you connect to another server that is called **printer.local**, that server will be given a different identifier. Firefox will save passwords, cookies, and other information for these servers separately.

“HTTPS for Local Domains” – Identifying Servers as Local

- Followings will be considered local:
  - “Servers with a .local(RFC 6762) or .home.arpa(RFC 8375) suffix”
  - “Servers with IPv4 literals from the RFC 1918 address spaces, such as 10.0.0.3”
  - “hosts with link-local IPv6 literals (fe80::/64)
  - “Unique Local IPv6 Unicast Addresses (fc00::/7)”
  - “Servers on loopback interfaces are local. This includes”
    - the IPv4 literal (127.0.0.1)
    - the host-scope IPv6 literal (::1)
    - And any origin with the name “localhost”

“HTTPS for Local Domains” – Origin Serialization

- “The primary drawback of adding more attributes to the origin tuple is the effect it has on applications that use origin in their processing.”
- “This proposes a change to the serialization of origins for devices so that it includes a hash of the server’s public key information (SPKI). This is added to the ASCII and Unicode serializations of the origin.”
  - E.g., https://_NPNE4IG2GJ4VAL4DCHL64YSM5BII4A2X.printer.local
  - “TBD: Does this need a new scheme? Is it a new field, or can it be added to the domain name? What separator would this use? How should the SPKI hash be encoded (base64url, base32, hex)? How many bits are enough? Do we need to signal hash function? Is this a new field at the end, a change to the name, or something else? Should SPKI go in the middle to discourage prefix-matching?”

“HTTPS for Local Domains” – Advantages and Drawbacks

- Advantages
  - “The key advantage of this approach is that the extension of origin allows local services, including those running on the local machine, to use and benefit from HTTPS.”

- Drawbacks
  - User Involvement
    - “Part of the security of this system involves user awareness.”
  - Key Changes
    - “Devices that rotate keys will gain a new identity, and lose access to any existing state.”
  - Address Changes
    - It is intentional but “devices that are identified by their IP address will receive a new identity when their assigned address changes.”
  - Ergonomics of Origins
    - “the ergonomics of an origin that includes a SPKI hash is not always ideal.”

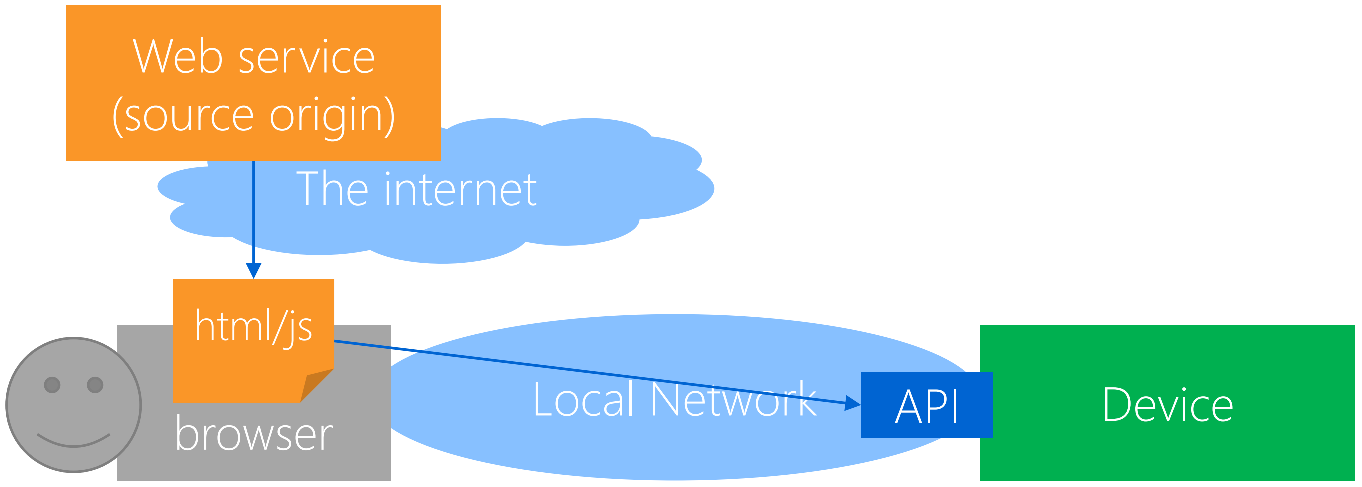
“A draft proposal for local HTTPS communication” by Ajitomi, Daisuke

- https://github.com/dajiaji/proposals/blob/abstract_proposal/draft_proposal_supporting_local_https_communication.md

Device Access Pattern

The use cases for local HTTPS can be categorized into two device access patterns

- Normal access pattern
  - the device has web contents and a user types the address of the device (e.g., ‘https://device.local’) on the browser directly and fetches the contents.
- Cross-origin access pattern
  - the device has API endpoints and a web frontend loaded on a browser from the internet accesses the APIs through a browser API (e.g., fetch API) and fetches the response.



Overview of Proposal

Focus on what the rationale for browsers to trust RPK, self-signed cert. or private CA cert.

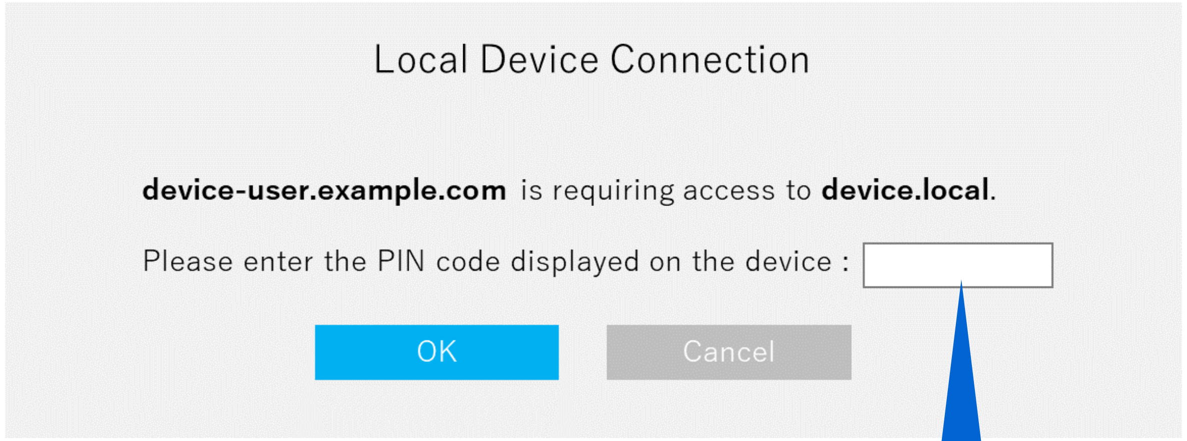
- Approach #1: PAKE-based approach
  - The rational: user's approval
- Approach #2: OAuth/ACE-based approach
  - The rational: webauthn-like trust model and user's approval
- Approach #3: Private CA Cert. (&ACME)-based approach
  - The rational: webauthn-like trust model and user's approval

Approach #1

- Use PAKE/TLS based on user's approval:
  1. A web service (https://device-user.example.com) calls fetch API with extensions below.
  2. When the device accepts the use of J-PAKE (or TLS1.3-compatible PAKE), the UI below is shown.
  3. The browser allows the access only if the user grants the access through the UI.

- An Example of the extension for browser API and UI

```
fetch("https://device.local/stuff", {
  tlsExtension: { // Should this be available only to local servers?
    type: "pake", // Should this be done on underlying TLS handshake?
    // Following is an optional argument for subsequent TLS sessions
    // to identify the local server and to omit user approval.
    pinnedIdentity: "<base64-encoded SPKI of the certificate or its fingerprint>"}});
```



Local Device Connection

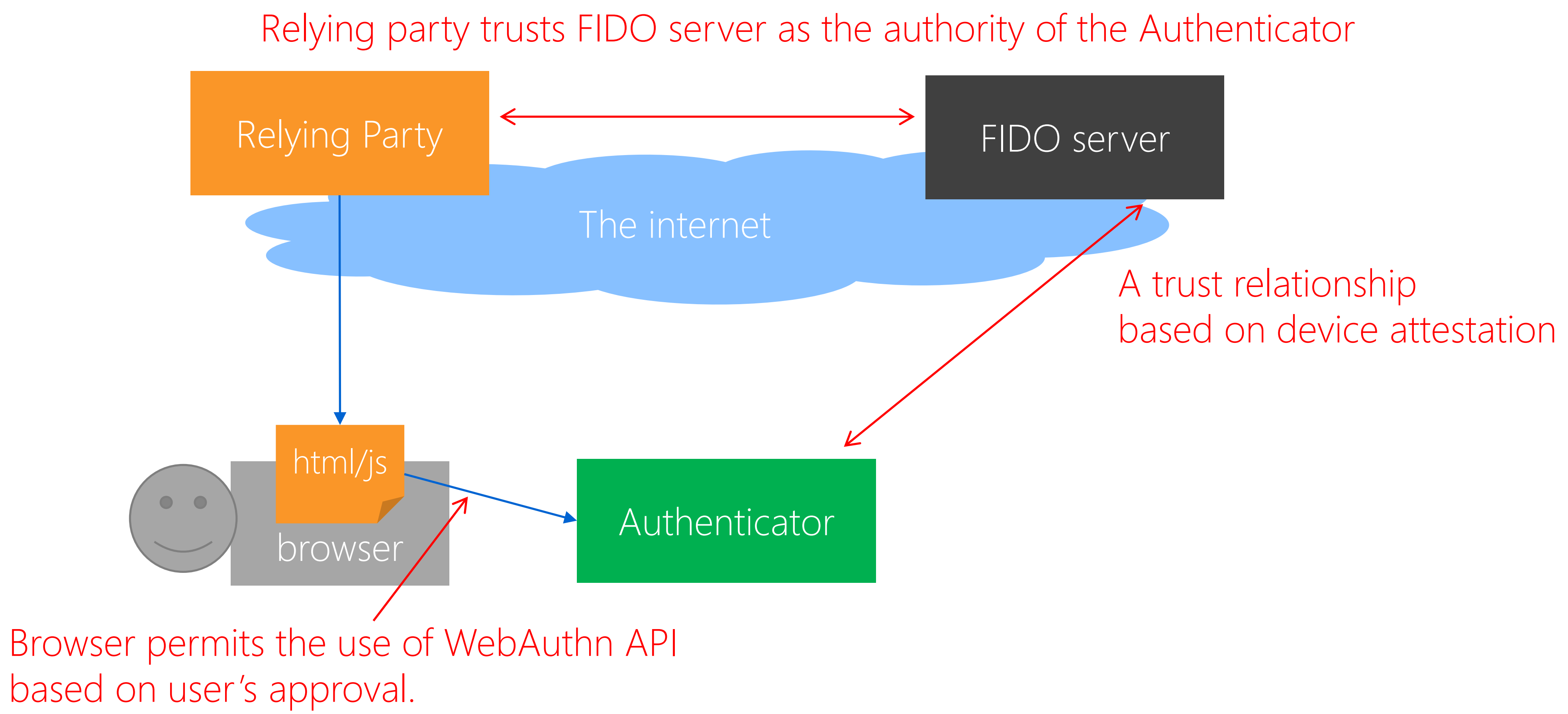
device-user.example.com is requiring access to **device.local**.

Please enter the PIN code displayed on the device :

To make sure that the 'device.local' displayed on the pop-up window is really the same as the domain name of the device, user inserts either a PIN or password through the window.

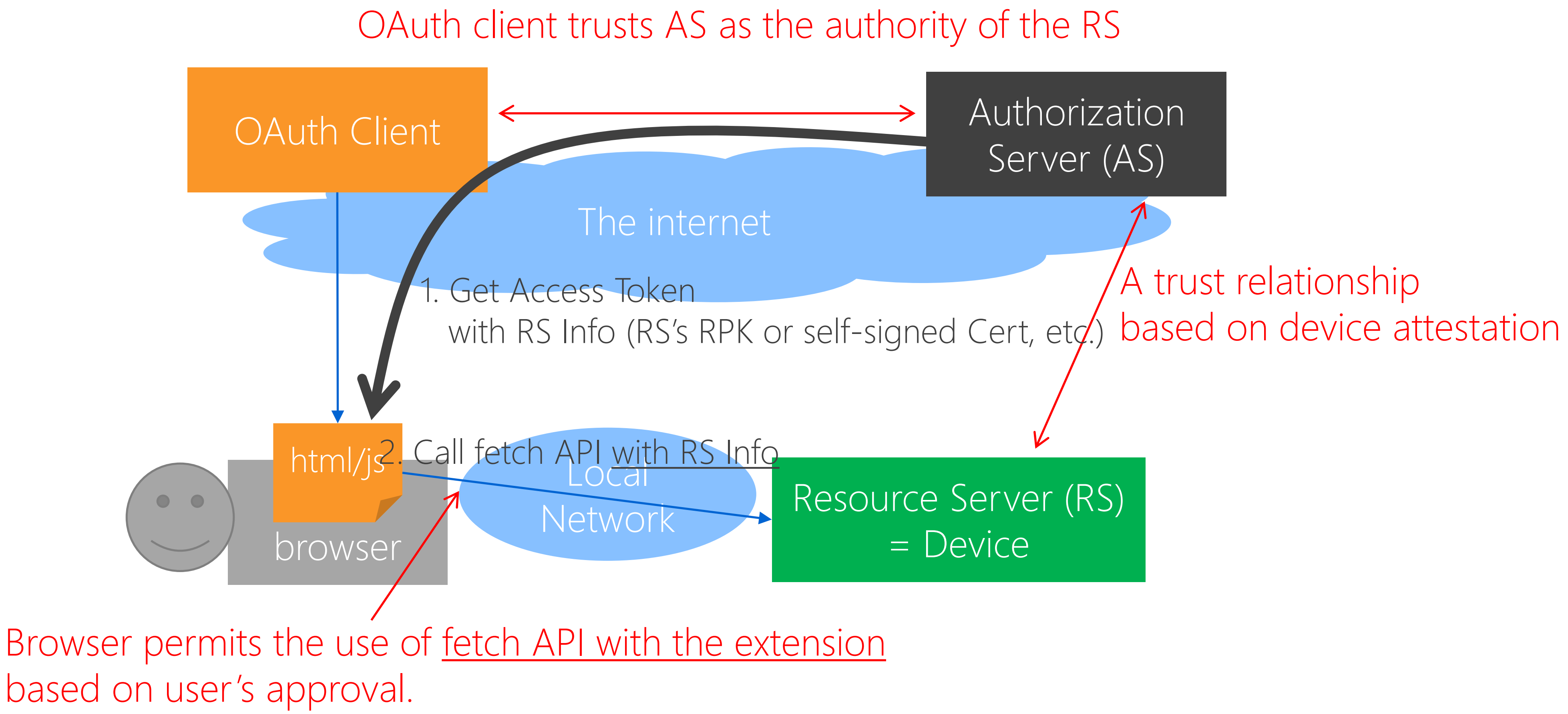
Basic Idea of Approach #2 and #3

- WebAuthn's trust model:


Relying party trusts FIDO server as the authority of the Authenticator
Relying Party
FIDO server
The internet
A trust relationship
based on device attestation
html/js
browser
Authenticator
Browser permits the use of WebAuthn API
based on user's approval.

Approach #2

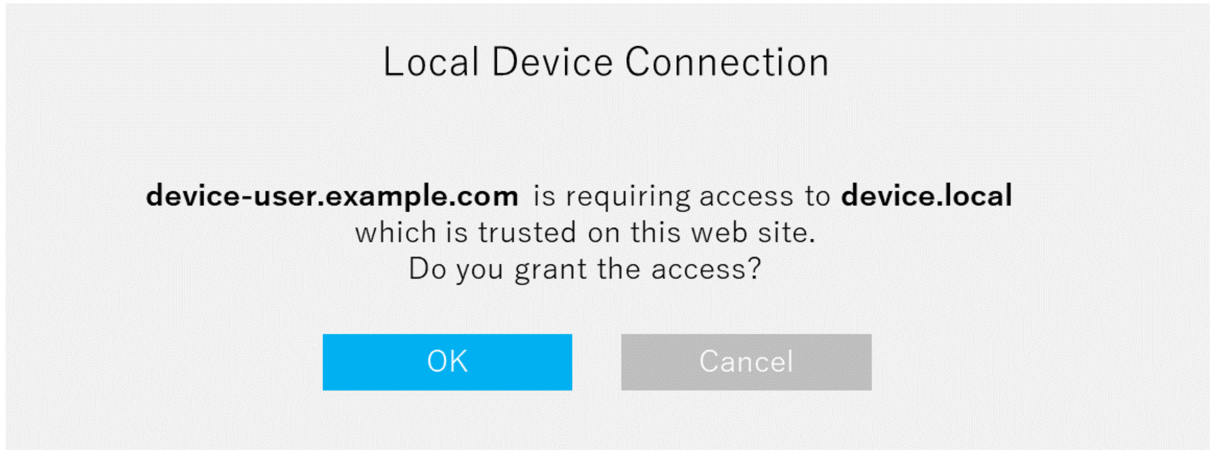
- Use OAuth/ACE framework with device attestation and user's approval:


OAuth client trusts AS as the authority of the RS
OAuth Client
Authorization Server (AS)
The internet
1. Get Access Token
with RS Info (RS's RPK or self-signed Cert, etc.)
A trust relationship
based on device attestation
html/js
2. Call fetch API with RS Info
Local Network
Resource Server (RS) = Device
browser
Browser permits the use of fetch API with the extension
based on user's approval.

Approach #2

- An Example of the extension for browser API and UI:

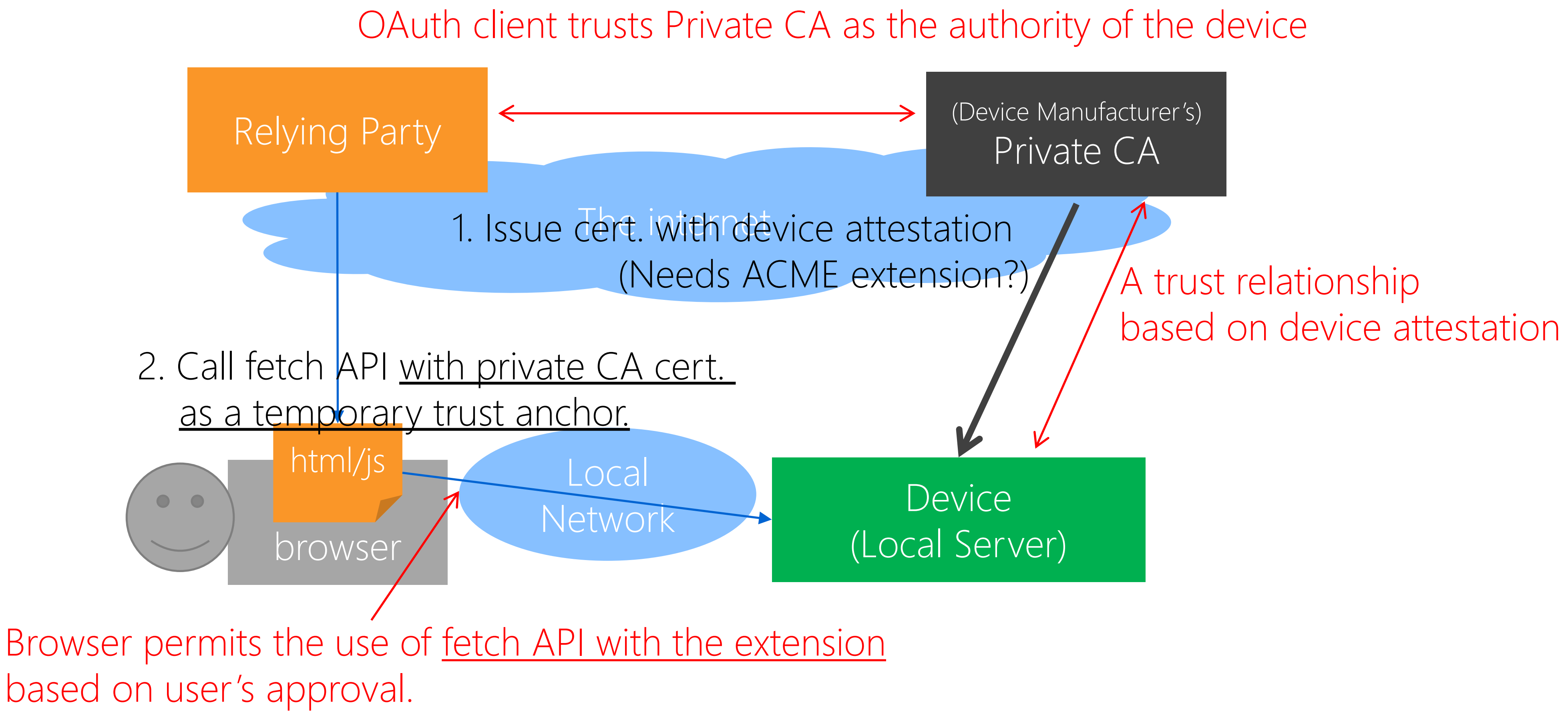
```
// When RS Information includes a RPK,
fetch("https://device.local/stuff", {
  tlsExtension: {
    type: "rpk",
    pinnedIdentity: "<base64-encoded raw public key or its fingerprint>"}});

// When RS Informatin includes a self-signed certificate,
fetch("https://device.local/stuff", {
  tlsExtension: {
    type: "pkix",
    pinnedIdentity: "<base64-encoded SPKI of the certificate or its fingerprint>"}});
```


Local Device Connection

device-user.example.com is requiring access to **device.local**
which is trusted on this web site.
Do you grant the access?

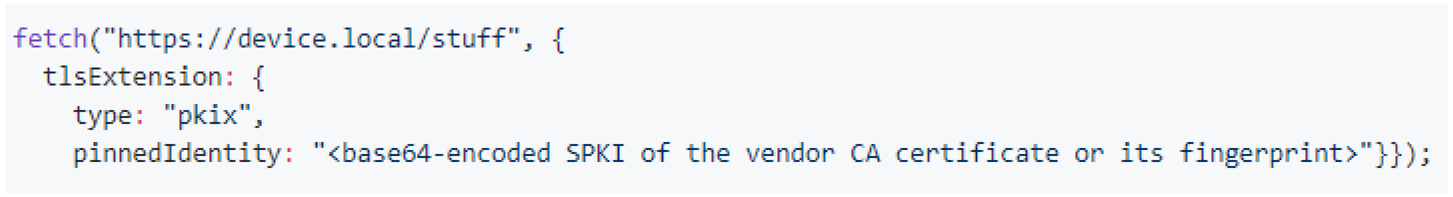
Approach #3

- Use private CA certs based on device attestation and user's approval:

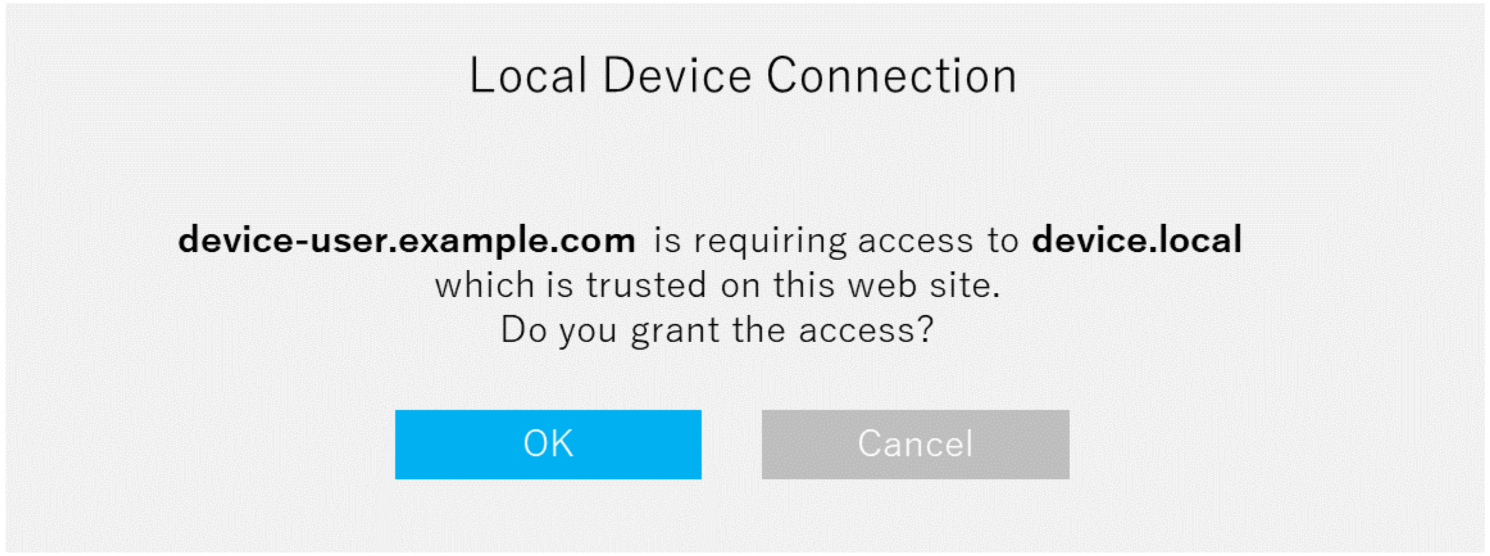

OAuth client trusts Private CA as the authority of the device
Relying Party
(Device Manufacturer's)
Private CA
The internet
1. Issue cert. with device attestation
(Needs ACME extension?)
A trust relationship
based on device attestation
2. Call fetch API with private CA cert.
as a temporary trust anchor.
html/js
browser
Local
Network
Device
(Local Server)
Browser permits the use of fetch API with the extension
based on user's approval.

Approach #3

- An Example of the extension for browser API and UI:



```
fetch("https://device.local/stuff", {
  tlsExtension: {
    type: "pkix",
    pinnedIdentity: "<base64-encoded SPKI of the vendor CA certificate or its fingerprint>"}});
```



Local Device Connection

device-user.example.com is requiring access to **device.local** which is trusted on this web site. Do you grant the access?

Pros and Cons of the Proposal

	Pros	Cons
Approach #1	• There is no need for manufacturers operate their own servers on the internet.	• There is no trust anchor for web services to trust the devices. • A user has to input a PIN/password, or a device has to support a secondary communication channel (e.g., BLE, NFC). • Web browsers need to support PAKE.
Approach #2	• Web services can trust devices as far as they can trust AS. • The authenticity of devices can be guaranteed when the AS authenticates devices based on remote device attestation.	• Manufacturers have to operate their own server (AS). • (Device domain names are not validated).
Approach #3	• Web services can trust devices as far as they can trust private CAs. • The authenticity of devices can be guaranteed when the private CA authenticates devices based on remote device attestation. • Existing PKI-based methods for managing the lifecycle of certificates can be used (e.g., CRL, OCSP). • (Device domain names can be validated if ACME can be extended for local domain names.)	• Manufacturers have to operate their own servers (AS and/or private CA).

Discussion

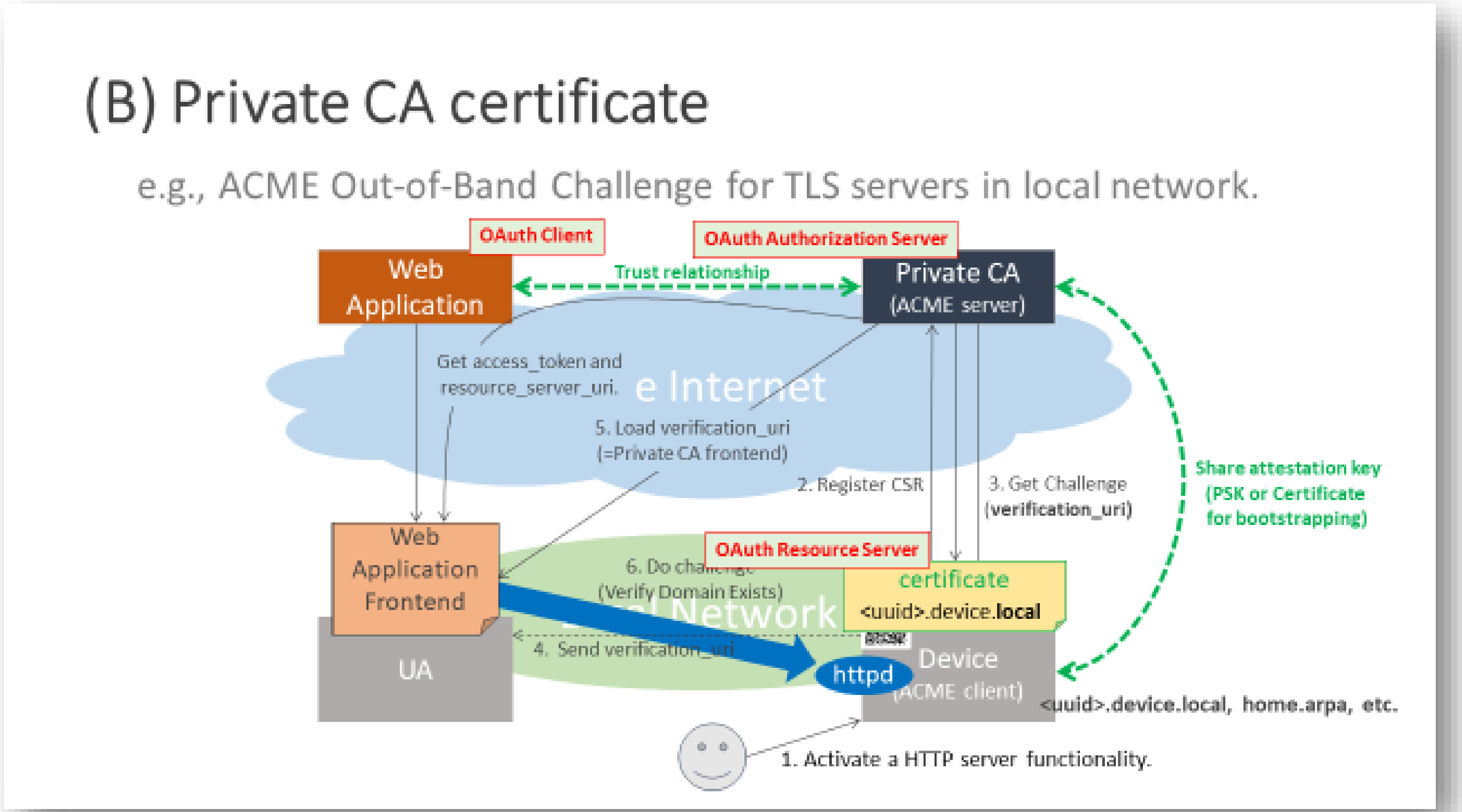
- Any comments on the proposals?
- Issues on the github repository:
  - Web PKI certificates and/or Vendor(s)-Issued certificates?
  - Initial step to trust a device.
  - Re-authenticating a device.
  - Extending Web PKI for local network devices.
  - Server-managed device authentication.
- What is the next step?
- Are there any browser vendor people who are interested in this activity and move forward with it?

TOSHIBA

Backups

Approach #3

- FYI: My proposal at the breakout session of TPAC 2017
  - https://www.w3.org/wiki/File:TPAC2017_httpslocal-2.pdf



(B) Private CA certificate

e.g., ACME Out-of-Band Challenge for TLS servers in local network.

OAuth Client
Web Application
OAuth Authorization Server
Private CA (ACME server)
Trust relationship
Get access_token and resource_server_uri.
e Internet
5. Load verification_uri (=Private CA frontend)
2. Register CSR
3. Get Challenge (verification_uri)
Share attestation key (PSK or Certificate for bootstrapping)
Web Application Frontend
OAuth Resource Server
6. Do challenge (Verify Domain Exists)
Network
certificate
<uuid>.device.local
UA
4. Send verification_uri
httpd
Device (ACME client)
<uuid>.device.local, home.arpa, etc.
1. Activate a HTTP server functionality.