Client identifiers in AMP

November 11th, 2015
Authors: cramforce
Short link: https://goo.gl/Mwaacs

Published in GitHub intent to implement

- Context
- Design problem
- The client identifier
  - Privacy invariants
  - Usage of LocalStorage
    - Expiration
  - Base client identifier
    - 1st party on cdn.ampproject.org
    - Viewer based
    - 1st party on origin domain
  - External client identifier
  - Consent

Context

This document deals with the question how advanced analytics can be supported inside of AMP files and in particular how a notion of a user or a session can be established.

See Intent to Implement: Analytics APIs for AMP for further context.

Design problem

AMP documents will frequently be loaded in 3rd party context. When loading an AMP document in a viewer, actual documents are loading in an iframe from cdn.ampproject.org. This prevents cdn.ampproject.org to be able to set 1st party cookies/storage. Because we aim to provide analytics fidelity of AMP documents displayed inside of viewers to be comparable to them being loaded individually, we need to find an equivalent mechanism that provides similar or better privacy aspects than the cookie based solution used by publishers today.

A separate issue is that when AMP documents are not loaded inside of a viewer but are loaded from a cache domain like cdn.ampproject.org then the scope of a 1st party cookie/storage for the entire domain would allow tracking users across publishers using a 1st party cookie/storage which is not desirable. Cookies have a way to limit their scope to a path like /nytimes.com/ but that mechanism is not usable at the scale of AMP due to per-domain cookie limits.

The client identifier

We introduce a client identifier that has the following properties:

- it gets exposed to tracking and analytics integrations in AMP.
- it gets deleted when the user deletes their cookies.
- when 2 AMP documents were retrieved from 2 **different** source origins they are **not** the same.
- when 2 AMP documents were retrieved from the same origin they are the same.
- the client identifier does not imply any kind of authority over the viewer.
- when sending the same original client identifier to 2 different third party tracking services 2 different derivatives of the client id are sent.
- Publishers may provide a mechanism for users to express consent to tracking before the client identifier is exposed to the publisher.

Privacy invariants

- Everything proposed in this doc works fine with setting only 1st party cookies/storage.
- This document does not provide a way to facilitate cross source origin tracking. It keeps the scope of identifiers exposed to a particular document to what could be achieved with a 1p cookie in a non-proxy domain setup.
- If the user deletes their cookies/localStorage, the client identifier is deleted. A new one would be created on the next visit that has no relation to the previous one.

Usage of LocalStorage

This document proposes usage of LocalStorage for storing the `BaseCID` for the following reasons:

- The token is never sent to the proxy host or any other party, so it is externally verifiable that users can’t be tracked across multiple source origin domains.

- We do not need expiration of entries.
- In terms of deletion LocalStorage works the same as cookies in modern browsers (They are deleted together). All browsers that support LocalStorage have a way to delete it.

This document does not make a final decision as the whether any of the following technologies are used for LocalStorage

- Actual localStorage API (Currently most likely)
- IndexedDB
- WebSQL

All of them behave exactly the same in term of privacy and security with respect to this project (and at least in terms of how they are exposed to the web), but they do have varying performance properties.
Confirmed that the following browsers delete localStorage when the user asks to delete cookies:

- Chrome
- Safari
- Firefox
- IE
- Opera

Expiration

Once a day we store with the localStorage entry when it was last read. If we read a key that has not been read for more than N days (likely 365 days but TBD), we proceed to delete the key (expire it) and act like none was present.

Base client identifier

Assume CHASH is a cryptographic hash function. According to crypto experts the scheme proposed here is safe when using SHA-384[1] (Closure implementation). Also a version based on XXTEA is proposed[2] that would require significantly less sophisticated JS implementation.

There are three types of client identifiers:

1. `BaseCID`: This is an internal value that is never exposed to anyone except AMP core code. It is never sent to any server.
2. `SourceCID`: This is a per “source origin” (See below) derivative of `BaseCID`. It never exposed to any server nor to any non-core AMP code. Just an intermediate client-only value.
3. `ExternalCID`: This is a CID to be sent to an actual server for the purpose of a single source origin and tracking provider.

---

[1] SHA1 is subject to https://en.wikipedia.org/wiki/Length_extension_attack with the constructions suggested below.
[2] X=KeyMasterHMAC(Seed)
D=Hash(domainName)
CID=XXTEA(X, D)

1st party on cdn.ampproject.org

In this scenario the AMP file is loaded as the primary document into the browser from the proxy. A sample URL is https://cdn.ampproject.org/c/www.theguardian.com/us-news/2015/sep/26/obama-africa-hiv-aids-treatment-women/amp

Marked in pink (www.theguardian.com) is what we call the `SourceOrigin`. Documents from a different source origins must not have access to the same client identifier.

To create the client identifier we run through the following steps:

1. Check if localStorage key `amp_cid` exists on https://cdn.ampproject.org/
2. If yes:
    a. `BaseCID` is the value of the existing storage entry.
3. If no, create a new `BaseCID`
    a. Collect 128bits of entropy using `window.crypto.getRandomValues` and hash the result with CHASH.
    b. If `getRandomValues` is not available (it is available for the vast majority of browsers supported by AMP) we take CHASH of the user's screen dimensions, the current document location, ~~the current cookies~~[3], the current time, the time AMP first loaded and a random number.
    c. Set localStorage key `amp_cid` on https://cdn.ampproject.org/ to `"0" + Base64(a || b)`
        i. a or b are the values produced by the bullet points a or b above.
        ii. "0" is used as a version identifier of the encoding scheme.
    d. `BaseCID` is now the value of the new `amp_cid` entry.
4. We now calculate `SourceCID = CHASH(BaseCID + SourceOrigin)`

Viewer based

In this version the AMP document is embedded into a viewer. The viewer resides on its domain different from cdn.ampproject.org and loads an iframe on cdn.ampproject.org.

We produce a CID if none is available by delegating an identifier generated by the viewer to the AMP document. An equivalent operation is e.g. a script from an ad network that sets 1st party cookies on a site and sends the values down to its third party iframe.

Note, that the below is only a recommendation. Non-Web environments may use different strategies and also web based viewers may decide on a different strategy for CID generation.

The steps to create the CID are:

1. AMP documents asks viewer via `postMessage` to provide a `BaseCID`.
2. Viewer now has control.

---

[3] Not use because in AMP they contain no useful entropy.

1. Check if localStorage key `amp_cid` exists on viewer origin.
2. If yes:
    a. `BaseCID` is the value of the existing storage entry.
3. If no, create a new `BaseCID`
    a. Collect 128bits of entropy using `window.crypto.getRandomValues` and hash the result with CHASH.
    b. If `getRandomValues` is not available (it is available for the vast majority of browsers supported by AMP) we take CHASH of the user's screen dimensions, the current document location, ~~the current cookies~~[4], the current time, the time AMP first loaded and a random number.
    c. Set localStorage key `amp_cid` on https://cdn.ampproject.org/ to `"0" + Base64(a || b)`
        i. a or b are the values produced by the bullet points a or b above.
        ii. "0" is used as a version identifier of the encoding scheme.
    d. `BaseCID` is now the value of the new `amp_cid` entry.
4. The viewer send a postMessage with targetOrigin cdn.ampproject.org and the value of `BaseCID` to the AMP document.
5. `BaseCID` is the value produced by that postMessage.
6. We now calculate `SourceCID = CHASH(BaseCID + SourceOrigin)`

1st party on origin domain

This is the case where the AMP file is just another HTML document on the web. Publishers should be able to track them together with other documents on their site.

To create the client identifier we run through the following steps:

1. The party asking for the CID must pass in a cookie name: `1pCookieName`
2. Check if that cookie exists.
3. If yes:
    a. `ExternalCID` is the value of the existing cookie called `1pCookieName`.
4. If no
    a. A cookie is generated given `1pCookieName` and the value "amp-" followed by base64 encoded random string (using same algorithm as base CID generation).

External client identifier

The `ExternalCID` is the actual value that is sent to a 3rd party tracking provider. They are based on the same `BaseCID` (except in the "1st party on origin domain" case above) and `SourceCID`, but have the property that e.g. the `ExternalCID` for Google Analytics is different from the `ExternalCID` for Adobe Analytics.

The API to get a `ExternalCID` is the actual API exposed by AMP's core CID code to users such as tracking libraries.

---

[4] Not use because in AMP they contain no useful entropy.

The API is roughly this:

```
Promise<string> getCid(string fallback1pCookieName)
```

The API returns a Promise (because CID generation might be asynchronous) for the `ExternalCID`. The passed in variable `fallback1pCookieName` is the name of the cookie that should be used to read this value in the “1st party on origin domain” case. The value of this string is also used to transform the `SourceCID` into the `ExternalCID` in the other cases:

```
SourceCID = See above for the non-1st party on origin domain cases.
ExternalCID = CHASH(SourceCID + fallback1pCookieName)
```

Consent

It may sometimes be required for publishers to get user consent for the cookie/localStorage based tracking.

AMP has support for dialogs that can be used to acquire user consent through the `amp-user-notification` element. This element actually requires a CID itself, so that publishers have an id they can use to associate consent (and avoid asking again). The CID exposed to the element is an `ExternalCID` that can not be correlated with other CIDs exposed to the publisher. Publishers must not use this `ExternalCID` for tracking.

We add a second argument to the `getCid` API called `consentDialogId`. This id is the HTML id of the `amp-user-notification` element.

```
Promise<string> getCid(string fallback1pCookieName, string consentDialogId)
```

The Promise for the CID will not resolve (and it won’t actually be created) until the referenced dialog was either accepted by the user or the publisher’s `data-show-if-href` signaled that no consent is necessary.

If a `BaseCID` is created (it does not exist yet) to create the `ExternalCID` for the request to `data-show-if-href`, that BaseCID is not persisted until user consent is given.