Update Cookie Matching

If the cookie matching table is currently hosted by Google:

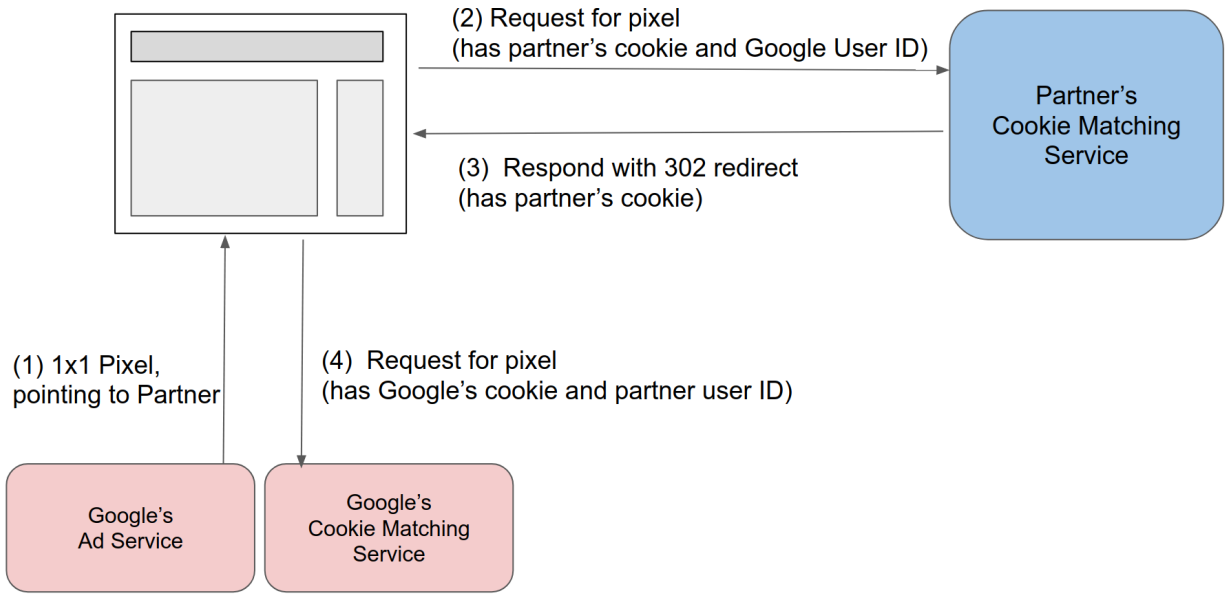
- Google will no longer send the `google_gid` parameter in the cookie matching request for users in affected US States.
- Review your service to make appropriate changes.

If the cookie matching table is hosted on the partner's side:

- Cookie matching must be updated to a Google hosted match table. Please follow the cookie matching migration guidance below to update the cookie matching pixel.
- If you use Bulk Upload, provide user id in your namespace following the Bulk Upload migration guidance below.

Changes to Google Initiated Cookie Matching

Current (Older) Workflow



1. In Google initiated cookie matching, also called pixel matching, Google serves a Cookie Matching pixel, pointing to partner's Cookie Matching Service. Note that the current request will contain Google's user ID. For example:
    a. `https://ad.network.com/pixel?google_gid=dGhpcyBpcyBhbiBleGFtGxl&google_cver=1`
2. User's browser requests a pixel from Partner's Cookie Matching Service.
3. Partner redirects the request to Google. Note that this redirect contains the Partner's user ID in the query parameter. For example:
    a. `https://cm.g.doubleclick.net/pixel?google_nid=<my_nid>&google_hm=<partner_user_id_base64_encoded>`
4. Browser requests pixel from Google. Note that this request contains the Partner's user ID in the query parameter. It also contains Google's cookie in the HTTP header. Google will store the mapping.

5. Google serves 1x1 pixel to the browser.

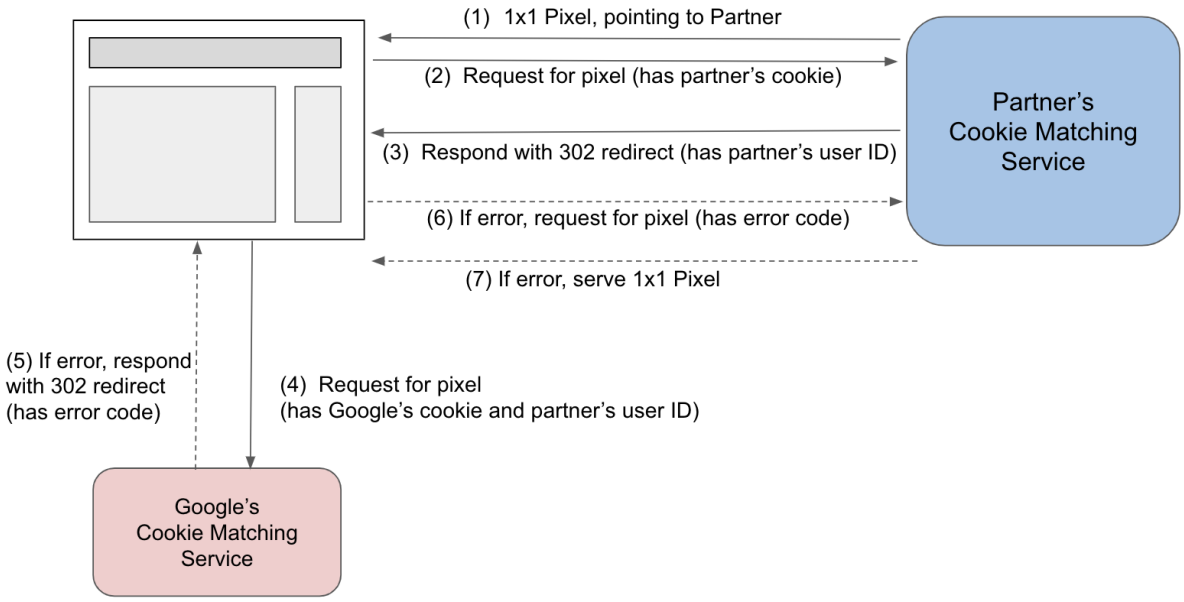
New Workflow
The new workflow is largely the same. The difference is that Step 1 and Step 2 will no longer contain Google User ID.

Changes required
Partner may need to update partner-owned Cookie Matcher to return a 302 redirect even when there is no Google User ID.

Changes to Partner Initiated Cookie Matching

The cookie matching pixel must be updated so Google can map the user id in the Google namespace to the user id in the partner's namespace.

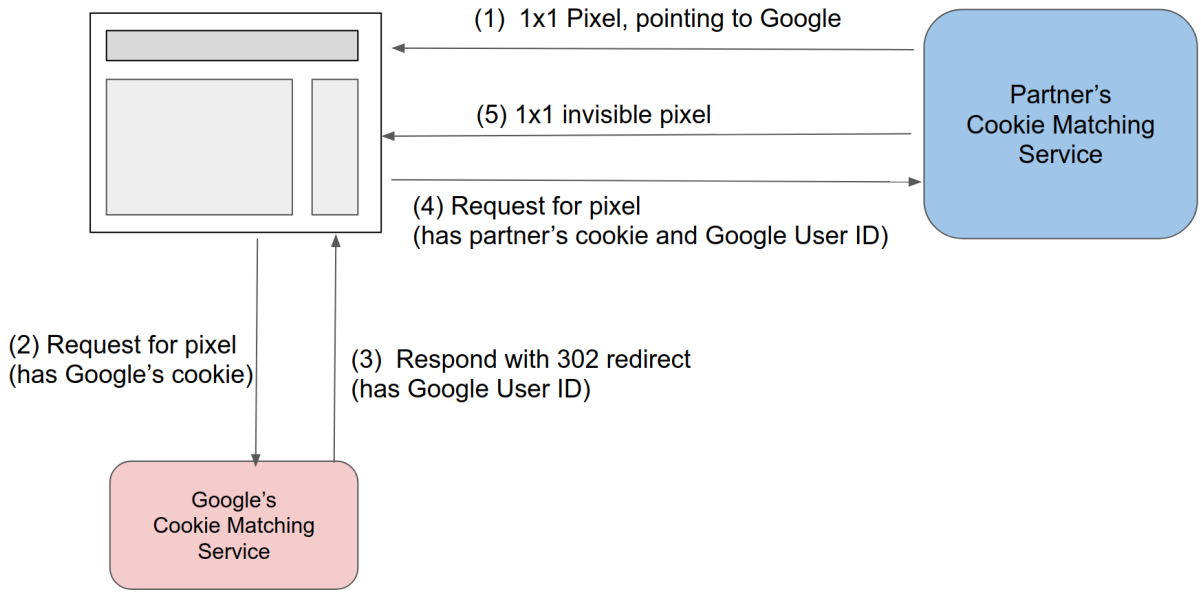
New Workflow



0. *(Optional)* Partner provides `cookie_matching_report_url` to Google.
1. Partner serves a 1x1 pixel pointing to Partner. For example:
   a. `<img src="https://ad.network.com/pixel" />`
2. Browser requests pixel from Partner. Note that this request contains the Partner's cookie in the HTTP header.
3. Partner redirects the request to Google. Note that this redirect contains the Partner's user ID in the query parameter. For example:
   a. `https://cm.g.doubleclick.net/pixel?google_nid=<my_nid>&google_hm=<partner_user_id_base64_encoded>`
   b. Note! Google expects URL safe base64 encoding (RFC 4648). This means we expect 62 and 63 to be '-' and '_', respectively.

4. Browser requests a pixel from Google. Note that this request contains the Partner's user ID in the query parameter. It also contains Google's cookie in the HTTP header. Google will store the mapping.
5. Google will respond in one of the following ways:
    a. If the partner did not provide a `cookie_matching_report_url`, Google will serve a pixel.
    b. If the partner provided `cookie_matching_report_url`, Google will redirect the request to `cookie_matching_report_url` only when there is an error. If the partner would like Google to redirect when there is no error, we can enable it per account. The error is indicated in `google_error` query parameter. See Cookie Matching documentation on the explanation of each `google_error` value.
6. If step 5b is performed, the browser requests a pixel from the Partner. This request contains status.
7. If step 5b is performed, the partner serves a pixel.
8. The pair of Google's user ID and Partner's user ID is stored at Google's cookie matching table.

Current (Older) Workflow



This flow is explained in the Cookie Matching Documentation. To reiterate:

0. Partner provides `cookie_matching_url` to Google. `cookie_matching_url` points to partner's Cookie Matching Service. For example:
    a. `https://ad.network.com/pixel`
1. Partner serves a 1x1 pixel pointing to Google. For example:
    a. `<img src="https://cm.g.doubleclick.net/pixel?google_nid=1234&google_cm" />`
2. Browser requests a pixel from Google. Note that this request contains Google's cookie in HTTP header.

3. Google redirects the request back to the partner at `cookie_matching_url`. In the redirect, `google_gid` will contain Google user ID for this user. For example:
    a. https://ad.network.com/pixel?google_gid=dGhpcyBpcyBhbiBleGFtGxl&google_cver=1
    b. Note! We will no longer attach `google_gid` for the requests in the affected region. Instead, we will attach `google_error=15`. See Cookie Matching documentation
4. Browser requests a pixel from the partner. Note that this request contains the partner's cookie in the HTTP header and Google user ID on the query parameter.
5. Partner serves 1x1 pixel to the browser.
6. The pair of Google's user ID and Partner's user ID is stored at Partner's cookie matching table

See additional details (e.g. valid parameters, error codes) in the Cookie Matching Documentation.

Supported parameters

For the redirect request described in step 3 above, we support the following query parameters, see API spec. for more detailed information.

- (Required) `google_cm`
    - Perform cookie matching. The value of the parameter is ignored and may be omitted.
- (Required) `google_nid`
    - This ID can be retrieved through the Buyer REST API Accounts resource's cookieMatchingNid field.
- (Required) `google_hm`
    - URL-safe base64 string. This is often a user ID.
- (Optional) `google_redir`
    - The encoded URL of where the partner wants Google to send a `302` redirect.
- (Optional) `google_ula`
    - Adds to the user list. The value is in the format userlistid[,timestamp]
- (Optional, Deprecated) `google_sc`
    - Sets the cookie if one is not present.
- (Optional, Deprecated) `google_no_sc`
    - Do not set the cookie if one is not present.

Changes required

To transition to the new flow, we recommend two methods:

Method 1: Use two pixels

Partners will leave the current pixel as-is and add an additional new pixel that will trigger the New Workflow. Google will continue to return Google user ID on the older flow until the deadline. Google will expect the Partner user ID on the new flow. During the transition period, both the Google Hosted Match Table and Partner Hosted Table can be used. After the partner has verified the new flow is working, the partner should remove the current pixel.

The current pixel is expected to contain only `google_cm` (if the current pixel contains both `google_cm` and `google_hm` see Method 2). The parameter indicates to Google that the partner is requesting a Google User ID, and Google will redirect to the `cookie_matching_url`. After the deadline, `google_cm` will no longer return Google user ID for the requests coming from the affected region.

Example current pixel:
`<img src="https://cm.g.doubleclick.net/pixel?google_nid=<my_nid>&google_cm" />`

The new pixel is expected to contain only `google_hm` (its value is the partner user ID). Google will redirect to **`cookie_matching_report_url`** if one is provided. Note that **`cookie_matching_report_url`** redirects only when there is error unless the partner stated otherwise.

Example new pixel:
`<img src="https://cm.g.doubleclick.net/pixel?google_nid=<my_nid>`
`&google_hm=<partner_user_id_base64_encoded>"` `/>`

The transition timeline would be:

1. Add the new pixel with new flow. Keep the current pixel as is.
2. Wait for the Google Hosted Match Table to be populated.
3. Remove the old pixel. Keep the new pixel.

The old pixel effectively be no-op after the deadline. Instead of returning Google user ID, it will contain a `google_error`. It will be draining resources for the user, the partner, and Google. We recommend removing it after the transition.

Method 2: Use one pixel
Method 2 requires the partner to change the current pixel so that Google can receive the Partner user ID and send the Google user ID using the same pixel. Partners using Google hosted match tables with partner-initiated cookie matching may already be using pixels with this structure. For example,

`<img src="https://cm.g.doubleclick.net/pixel?google_nid=<my_nid>`
`&google_cm&google_hm=<partner_user_id_base64_encoded>"` `/>`

When the pixel contains both `google_cm` and `google_hm`, Google will treat the request as part of the old Workflow. It means that Google will use **`cookie_matching_url`** as the base redirect URL. If a request is from the affected region, Google User ID is not attached. If it is not, then Google User ID is still attached. Google will still record the Partner User ID (i.e. `google_hm`).

The transition timeline would be:

1. Update the current pixel to include `google_hm`.
2. Wait for the Google Hosted Match Table to be populated.
3. *(Optional)* Remove `google_cm`.

Single Table for Multiple NIDs
Some partners may currently use multiple network IDs (i.e. `google_nid`) to do Cookie Matching and host a single table. We also support sharing a single Google Hosted Match Table for multiple network IDs.

If you have multiple network IDs (google_nid), and would like them to share a single Hosted Match table, please share the set of `google_nid` with your account team or use the support contact form at https://support.google.com/authorizedbuyers/gethelp.

Bulk Upload Changes

Note! Please let Google know whether you would like to use the User List upload functionality. This functionality needs to be enabled per account.

Currently, partners follow the guide here to upload user lists in Google's namespace (i.e. UserIdType = GOOGLE_USER_ID). After this change, the partner can upload in the partner's namespace. (i.e. UserIdType = PARTNER_PROVIDED_ID). Note that Google will continue to accept uploading user ids in Google's namespace (i.e. UserIdType = GOOGLE_USER_ID). The user list can be a mix of PARTNER_PROVIDED_ID and GOOGLE_USER_ID.

Note that the PARTNER_PROVIDED_IDs during Bulk Upload are not base64 encoded.

```
enum UserIdType {
  ...
  MSAI = 7;
  PARTNER_PROVIDED_ID = 4;
}

message UserDataOperation {
  optional string user_id = 1 [default = ""];
  optional UserIdType user_id_type = 14 [default = GOOGLE_USER_ID];
  ...
}

message UpdateUsersDataRequest {
  repeated UserDataOperation ops = 1;
  ...
}
```

The response will have a new error code to indicate when Google doesn't have a mapping:

```
enum ErrorCode {
  ...
  // Cannot decode provided cookie.
  BAD_COOKIE = 4;

  UNKNOWN_ID
  ...
}
```