# Passing Signals to ID5

07/02/2025 4:11 pm EDT

## What is Partner Data?

Partner Data includes all signals provided by partners that assist ID5 in accurately identifying users. By sharing additional information, you improve the precision, stability, and effectiveness of user identification. To ensure the highest-quality ID5 ID and unlock the full potential of your addressable audience, we recommend providing as many user signals as possible via the Partner Data (PD) string.

## Why Share More Signals?

With browsers increasingly obfuscating signals to protect user privacy, it is vital to proactively share robust data to maintain addressability and maximize user recognition.

**Recommended Next Steps:**

- **Share Core Signals**: Ensure IP address and user agent are always included in the PD string.
- **Include Additional Signals**: Contribute hashed emails, mobile ad IDs, or other identifiers for users who have provided consent. These signals help strengthen user identity resolution and deliver better outcomes across your properties.
- **Explore Custom Signals**: If you have specific signals that could enhance ID5's ability to reconcile users across your properties, please reach out to your ID5 representative. We will provide tailored advice on how to integrate them effectively.

## How is Partner Data Used?

Signals passed in the ID5 ID request are used to inform ID5 ID connections across domains. "Hard signals", such as hashed email addresses, take priority for cross-domain linking purposes, and help to train ID5's probabilistic algorithm. Publisher provided signals, such as IP address and user agent, may be used in ID5's probabilistic algorithm when ID5 determines that they may be more accurate than those that ID5 can source directly from the HTTP request. ID5 requires sha256 hashing & URL-safe base64 encoding to ensure that personally-identifiable information isn't transmitted in the ID5 ID call.

## Supported Partner Data Keys

| Key | Description | Required | Example |
|---|---|---|---|
| 0 | Other | Optional | |
| 1 | SHA256 Hashed Email[1] | Recommended | `f97ea86ed181d60b0ba62a30579f1e10ad71eaf21b548e173de75718065c5` |
| 2 | SHA256 Hashed Phone Number[2] | Recommended | `f687e4a1be889a45c13e417f77cb9bff9c67f46e35fd68d936e6b01a933ecbc` |
| 3 | Cross-Partner User ID Value | Optional | `e3206fbc-b2f1-11ed-afa1-0242ac120002` *(a user id that could be used across ID5 Partner Numbers / Accounts)* |
| 4 | Cross-Partner User ID Source *(hard-coded value will be provided by ID5)* | Optional | `super-pub-identifier` |
| 5 | Partner-Specific User ID Value | Optional | `e3206fbc-b2f1-11ed-afa1-0242ac120002` *(i.e. a user id that is specific to a single ID5 Partner Number / Account)* |
| 6 | Apple ID for Advertising (IDFA) *(lowercased)* | MAIDs should only be sent via the pd string for mobile web traffic where available. Please use our mobile app specific endpoint for in app traffic. Documentation here | `ea7583cd-a667-48bc-b806-42ecb2b48606` |

| Key | Description | Required | Example |
|---|---|---|---|
| 7 | Google Advertising ID (GAID) *(lowercased)* | MAIDs should only be sent via the pd string for mobile web traffic where available. Please use our mobile app specific endpoint for in app traffic. Documentation here | `cdda802e-fb9c-47ad-9866-0794d394c912` |
| 8 | Full URL | Recommended | `https://id5.io/solutions/?partner_type=publisher` |
| 9 | Domain | Recommended | `id5.io` |
| 10 | IPv4 Address of the end-user's device | Recommended | `77.99.190.227` |
| 11 | IPv6 Address of the end-user's device | Optional | `2001:0db8:85a3:000:000:8a2e:0370:7334` |
| 12 | User Agent String of the end-user's device | Recommended | `Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, lik Gecko) Chrome/94.0.4606.81 Safari/537.36` |
| 13 | Is Burner Email[3] | Optional | `false` |
| 14 | Apple ID for Vendor (IDFV) *(lowercased)* | Recommended (when available) | `f325g3gb-12fc-352f-c6c3-dz52f0f690d8` |
| 15 | **DEPRECATED** ~~CTV ID~~ | CTV IDs can be sent when integrating our ctv specific endpoint for ctv traffic. Documentation here | |
| 16 | **DEPRECATED** ~~CTV ID Type~~ | CTV IDs can be sent when integrating our ctv specific endpoint for ctv traffic. Documentation here | |
| 17 | An IAB TechLab Tokenization Framework token (e.g. uid2) | Optional | `AdvertisingTokenmZ4dZgeuXXl6DhoXqbRXQbHlHhA96l....` |



**ATTENTION**
If any of the signals is not available at the moment of the PD string creation, please leave key out of the PD string. Don't include keys for which you don't have the values or include default values you set.

- [1] See below for instructions on "Normalizing Emails Prior to Hashing"
- [2] See below for instructions on "Normalizing Phone Numbers Prior to Hashing"
- [3] A flag for whether the hashed email provided is a burner email (boolean). As emails are provided to ID5 in hashed form, ID5 is unable to determine whether provided hashed emails are "burner" emails. Publishers can use this flag to signal to ID5 whether the provided email should be treated as a "burner" email. If you're unsure how to determine if an email is a burner, we recommend you simply send `true` for all icloud email addresses

## Deriving the Partner Data (pd) Value

The general procedure to build a PD string is:

1. Normalize any values that need to be hashed, like emails, (see details below for how to) and then `sha256` hash them
2. URL-encode each value using UTF-8 charset (according to RFC 3986, or at the very least, in JS using a function like `encodeURIComponent`)

3. Create the raw pd string containing the keys and the URL-encoded value, using querystring formatting (order does not matter)
    - e.g. `<key1>` = `<value1>` & `<key2>` = `<value2>` ...
4. URL-safe base64 (RFC 4648) the entire raw PD string (using a function in JS like `btoa()`)
5. Once you have the encoded PD string, it can be passed into the `pd` field in any of our integrations (i.e.: PD parameter for our Prebid integration or JS API integration)

## Normalizing Hashed Inputs

Because ID5 doesn't see the original raw values for some of the signals we accept (e.g. hashed emails, hashed phone numbers), you will need to normalize them first. Normalizing the raw values before hashing them ensures that the signals sent by you and other partners will always be the same, ensuring the ID5 IDs can be properly linked.

### Normalizing Emails Prior to Hashing

Prior to hashing an email address, you must normalize the string by removing unnecessary characters:

1. Remove leading and trailing spaces
2. Convert all ASCII characters to lowercase
3. Please find below an example with email accounts ending in `@gmail.com`. You can apply this method to all the email accounts:
    a. Remove `.` (ASCII code 46) from the username of the email address
        - e.g. `jane.smith@gmail.com` normalizes to `janesmith@gmail.com`

    b. Remove `+` (ASCII code 43) and all subsequent characters from the username of the email address
        - e.g. `janesmith+test@gmail.com` normalizes to `janesmith@gmail.com`

### Normalizing Phone Numbers Prior to Hashing

Phone numbers should be normalized to the E.164 format, which is an international phone number format to ensure global consistency and uniqueness. When normalizing with the E.164 format, the result should be no more than 15 digits in length, prior to hashing.

1. Remove all spaces, hyphens, parentheses, or other special characters
2. Format the phone number as follows: `[+][country code][subscriber number including area code]`
    - e.g. `+111 22 333-44-555` normalizes to `+1112233344555`
    - e.g. `+1 (222) 333-4444` normalizes to `+12223334444`

## Example

Here is an example to show you how to generate a PD string, given you have the following raw signals to share:

- Email = `Jane.Smith+test@gmail.com`
- IPv4 = `77.99.190.227`
- IPv6 = `2001:0db8:85a3:000:000:8a2e:0370:7334`
- UA = `Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36`

### PD Creation Steps

#### Step 1: Normalize and Hash Inputs

In this example, we only need to normalize the email address. Following the instructions above ("Normalize Emails Prior to Hashing"), the result is `janesmith@gmail.com`, which when sha256 hashed, becomes `9a0f2978ccf8af196d24f627062a2d4054c9da92e9d998a514bda4a01a3cfec7`

#### Step 2: URL-encode the Values

- **Email (key 1)**

```
9a0f2978ccf8aaf196d24f627062a2d4054c9da92e9d998a514bda4a01a3cfec7
```

- **IPv4 (key 10)**

```
77.99.190.227
```

- **IPv6 (key 11)**

```
2001%3A0db8%3A85a3%3A000%3A000%3A8a2e%3A0370%3A7334
```

- **UA (key 12)**

```
Mozilla%2F5.0%20(Windows%20NT%2010.0%3B%20Win64%3B%20x64)%20AppleWebKit%2F537.36%20(KHTML%2C%20like%20
```

#### Step 3: Create Raw PD String

```
1=9a0f2978ccf8aaf196d24f627062a2d4054c9da92e9d998a514bda4a01a3cfec7&10=77.99.190.227&11=2001%3A0db8%3A85a3%3A00
```

#### Step 4: URL-safe base64 Raw PD String

```
MT05YTBmMjk3OGNjZjhhZjE5NmQyNGY2MjcwNjJmMmQ0MDU0YzlkYTkyZTlkOTk4YTUxNGJkYTRhMDFhM2NmZWM3JjEwPTc3Ljk5LjE5MC4y
```

## Sample Javascript Implementation

Here is just one approach to generate the PD string, but there are numerous other ways to accomplish the same result. Here we're using the same inputs as the example above.

```
// get these values from your webserver or the browser's apis
const ipv4  = '77.99.190.227',
      ipv6  = '2001:0db8:85a3:000:000:8a2e:0370:7334',
      ua    = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.0.0 Safari/537.36',
      email = 'jane.Smith+test@domain.com'; // alternatively, normalize and then sha256 server-side and return the hashed value

// normalize the email string with normalizeEmail() from  https://github.com/validatorjs/validator.js/blob/master/src/lib/normalizeEmail.js
const cleansedEmail = normalizeEmail(email);

// set the keys and URL-encode each value
const pdKeys = {
       1: CryptoJS.SHA256(cleansedEmail), // requires the crypto-js package https://www.npmjs.com/package/crypto-js
      10: encodeURIComponent(ipv4),
      11: encodeURIComponent(ipv6),
      12: encodeURIComponent(ua),
};

// convert the key/values into a querystring format
const pdRaw = Object.keys(pdKeys).map(key => key + '=' + pdKeys[key]).join('&');

// base64 encode the raw string; this is the final value you can pass into the pd field
const pdString = btoa(pdRaw);
```