PingOne Signals Integration Best Practices and Troubleshooting Guide

This document is intended for Williams-Sonoma's web developers who are integrating the **PingOne Signals (Protect) SDK** on the "My Account" login page. It explains why device payloads sometimes go missing, outlines best practices for data collection, and provides specific guidance for improving the current implementation.

Why device payloads can be missing

Late or incomplete SDK initialization

The PingOne documentation highlights that the Signals SDK must be initialized as early as possible. Initialization triggers the collection of device attributes and behavioural data that are necessary for risk evaluations. The docs entail that "the earlier you can initialize the PingOne Signals SDK, the more data it can collect to make a risk evaluation"[1]. If the SDK is loaded late in the page lifecycle or if it waits for the `load` event (the default behaviour), users may submit the form before any data is gathered, resulting in an empty payload.

Short payload retrieval timeout

Your `getPingDevicePayload()` helper races the asynchronous `getData()` call against a hard-coded 2.5-second timeout. When the SDK is still gathering signals or the network is slow, `getData()` may not resolve before the timeout, so the function returns an empty string. Longer page loads, slow devices or blocked scripts can all contribute to this condition.

Incorrect or missing configuration parameters

Recent versions of the PingOne JavaScript SDK use a `start()` method that accepts an **environment ID** (`envId`) and optional flags, such as `waitForWindowLoad`, `disableTags`, `enableTrust`, and `deviceAttributesToIgnore`. Using the older `init()` method or failing to supply an `envId` may prevent the SDK from starting correctly. Some optional flags can also reduce the data collected—for example, setting `disableTags` stops the SDK from recording page tags, and setting `disableHub` stores data only in local storage[2].

Paused behavioural data collection

The Signals SDK collects behavioural data (typing rhythm, scrolling, etc.) in addition to device attributes. There are methods to pause and resume behavioural capture[3]. If behavioural capture is accidentally paused—either by your code or by a misinterpreted callback—the SDK will only return static device attributes, which may be considered insufficient and lead to a missing or low-quality payload.

Script blocking or duplicate loading

Ad-blockers and privacy extensions can block scripts that collect device information. If the script fails to load, the `_pingOneSignals` object will be undefined. In addition, your `isScriptAlreadyExists()` function compares script URLs using strict equality. If the script URL differs by protocol or query parameters (e.g., versioning), the helper might inject multiple copies of the SDK. Multiple initializations can lead to unpredictable results and missed payloads.

Best practices for integrating PingOne Signals

- **Initialize early:** Call the SDK's initialization method as soon as the HTML and JavaScript load. The documentation stresses that early initialization allows the SDK to collect more data[1].
- **Use the latest initialization method:** For the JavaScript SDK, use the `start()` method with your PingOne **environment ID**. Avoid using legacy `init()` calls.
- **Avoid waiting for the `load` event:** Set the `waitForWindowLoad` parameter to `false` so the SDK starts on the `DOMContentLoaded` event. The default value (`true`) delays initialization until all external resources load[4].
- **Handle the SDK's readiness callbacks:** Only call `getData()` after the SDK signals it is ready—either via your custom `PingOneCollectionReady` event or the built-in `PingOneProtectEvaluationCallback`[5]. This ensures that data collection has finished.
- **Increase payload-retrieval timeouts:** Allow more time for `getData()` to complete, especially on pages with heavy content or users on slow connections.
- **Pause and resume behavioural data correctly:** Only call `pauseBehavioralData()` when you deliberately want to stop capturing behavioural signals and resume it when you need behavioural data again[3].
- **Place the SDK script at the top of your template:** In DaVinci flows, Ping recommends placing the `skrisk` script at the very beginning of your HTML template so that device data collection begins immediately[6].
- **Keep the SDK script unblocked:** Ensure that ad-blockers or content security policies do not block `signals.min.js`. When testing, disable such tools or whitelist the PingOne domain.

For additional guidance, see Ping Identity's official **Best practices for PingOne Protect** and **Ping SDKs – Step 3. Develop the client app** documentation. These pages explain the available SDK flags, initialization timing and how to return collected data[1][7].

Tailored instructions for Williams-Sonoma's implementation

Your current helper functions perform a dynamic script injection and then call `window._pingOneSignals?.init()` with no parameters. They also race a `getData()` call against a 2.5 s timeout. To improve reliability and conform to PingOne's recommended patterns, adopt the following changes:

1. Load and initialize the SDK early

Instead of setting the `defer` attribute on the script tag, append the script to the `<head>` and listen for the `DOMContentLoaded` event. Once the script loads, call the `start()` method with your **environment ID** and set `waitForWindowLoad` to `false`:

```
<!-- Load PingOne Signals SDK at the beginning of the page -->
<script
src="https://cdn.pingone.com/pingone-protect/sdks/web/v2/pingone-protect.min.
js"></script>
<script>
  document.addEventListener('DOMContentLoaded', () => {
    if (window._pingOneSignals) {
      window._pingOneSignals.start({
        envId: 'YOUR_ENV_ID',
        waitForWindowLoad: false,  // initialize on DOMContentLoaded
        consoleLogEnabled: false
      }).then(() => {
        // Dispatch your custom readiness event once start() resolves
        window.dispatchEvent(new Event('PingOneCollectionReady'));
      }).catch(err => console.error('Signals init error', err));
    }
  });
</script>
```

This ensures that the SDK starts collecting data immediately after the DOM is parsed, rather than waiting for the full page load[4].

2. Adjust your readiness helper

Refactor your `onPingOneSignalsReady()` function to use promises so you only signal readiness after `start()` completes:

```
const onPingOneSignalsReady = (callback) => {
  function run() {
    const result = callback();
    if (result && result.then) {
      result.then(() => window.dispatchEvent(new
Event('PingOneCollectionReady')));
    } else {
      window.dispatchEvent(new Event('PingOneCollectionReady'));
    }
  }
  if (window._pingOneSignalsReady) {
    run();
  } else {
    document.addEventListener('PingOneSignalsReadyEvent', run);
  }
};
```

3. Retrieve the payload only after readiness

Listen for your `PingOneCollectionReady` event and then call `getData()`. Remove the race with a fixed timeout or increase the timeout to accommodate slower devices:

```
window.addEventListener('PingOneCollectionReady', async () => {
  try {
    // Optionally show a loader while waiting
    const data = await window._pingOneSignals.getData();
    // Now include `data` in your risk-evaluation request
  } catch (error) {
    console.error('Error fetching PingOne data', error);
  }
});
```

This pattern matches Ping's guidance, which instructs developers to call `getData()` in response to the evaluation callback[5].

4. Clean up duplicate script detection

Simplify `isScriptAlreadyExists()` by checking for the global `_pingOneSignals` object rather than comparing full URLs:

```
const isPingOneSignalsLoaded = () => {
  return typeof window._pingOneSignals !== 'undefined';
};

// Only inject the script if it hasn't been loaded yet
if (!isPingOneSignalsLoaded()) {
  const script = document.createElement('script');
  script.src =
'https://cdn.pingone.com/pingone-protect/sdks/web/v2/pingone-protect.min.js';
  document.head.appendChild(script);
}
```

Checking for `_pingOneSignals` avoids false negatives due to URL variations and prevents duplicate initializations.

5. Key changes to your existing code snippet

To help the development team adopt these best practices quickly, the following table maps the original helper functions to the recommended changes:

Current function	Issues	Recommended changes
`initPingOneSignals(sdkSource, defer = "true")`	Uses a `defer` attribute on the script tag and calls `_pingOneSignals.init`	Remove the `defer` attribute; load the SDK early (e.g., inside a script tag in the

Current function	Issues	Recommended changes
	`()` with no configuration. This delays initialization until the `load` event and omits required parameters such as `envId`, which reduces the collected data.	`<head>`). After the script loads, call `_pingOneSignals.start({ envId: 'YOUR_ENV_ID', waitForWindowLoad: false })`. Dispatch your custom readiness event only after the `start()` promise resolves.
`isScriptAlreadyExists()`	Compares script URLs using strict equality, which may miss existing scripts or result in duplicate loading. Dispatches the `PingOneCollectionReady` event even if the SDK is not fully initialized.	Replace URL comparison with a simple check for `window._pingOneSignals`. Only dispatch the readiness event after the `start()` promise resolves, not immediately upon finding the script.
`getPingDevicePayload(responseTimeOutInMs = 2500)`	Races the `getData()` call against a 2.5-s timeout. If the SDK takes longer to collect data, the function returns an empty string. It does not wait for the SDK's readiness event.	Listen for `PingOneCollectionReady`, then call `await window._pingOneSignals.getData()`. Increase the timeout if you continue to use a race condition, or remove the timeout altogether. Include proper error handling around `getData()` to surface SDK errors.
`onPingOneSignalsReady(callback)`	Immediately dispatches `PingOneCollectionReady` after invoking the callback, regardless of whether the SDK has finished initializing.	Refactor to run the callback (which calls `start()`) and then dispatch the readiness event only when the returned promise resolves. This

Current function	Issues	Recommended changes
		prevents `getData()` from being called before initialization completes.

These changes ensure that your initialization code meets Ping's guidelines: initialize as soon as the DOM loads, supply the environment ID, wait for the SDK to signal readiness, and avoid premature timeouts.

6. Review optional SDK flags

- **Behavioural capture:** Only pause behavioural data collection if your journey explicitly requires it; otherwise leave it running so the payload contains behavioural signals[3].
- **Disable tags / hub:** Avoid setting `disableTags` or `disableHub` to `true` unless you have a privacy or compliance reason to do so[2].
- **Enable trust:** If you need to bind the device payload to a non-extractable key for authenticity, set `enableTrust: true`[8].

7.Documentation

Refer y to the official PingOne documentation for further details:

- **Ping SDKs – Step 3. Develop the client app:** This guide explains how to initialize the SDK and return collected data for risk evaluations[1][5].
- **Best practices for PingOne Protect:** Describes why predictors (e.g., "New Device") rely on the Signals SDK and recommends placing the `skrisk` component at the start of your HTML template[7][6].

By following these instructions—initializing early, using `start()` with an environment ID, waiting for readiness before retrieving the payload, and adjusting your timeout—you should see consistent, non-empty device payloads during login. These practices align with Ping Identity's official recommendations and provide a robust integration path for your login experience.

---

[1] [2] [3] [4] [5] [8] Step 3. Develop the client app | Ping SDKs

https://docs.pingidentity.com/sdks/latest/sdks/integrations/pingone-protect/03-app.html

[6] [7] Best practices for PingOne Protect | PingOne

https://docs.pingidentity.com/pingone/threat_protection_using_pingone_protect/p1_protect_best_practices.html