Cross-device WebOTP

THIS DOCUMENT IS PUBLIC

Status: Final
Authors: yigu@chromium.org
Last Updated: 2021-06-29

tl;dr

We plan to support WebOTP API on desktop when both Chrome Desktop and Chrome Android are logged into the same Google account. A typical flow looks like this:

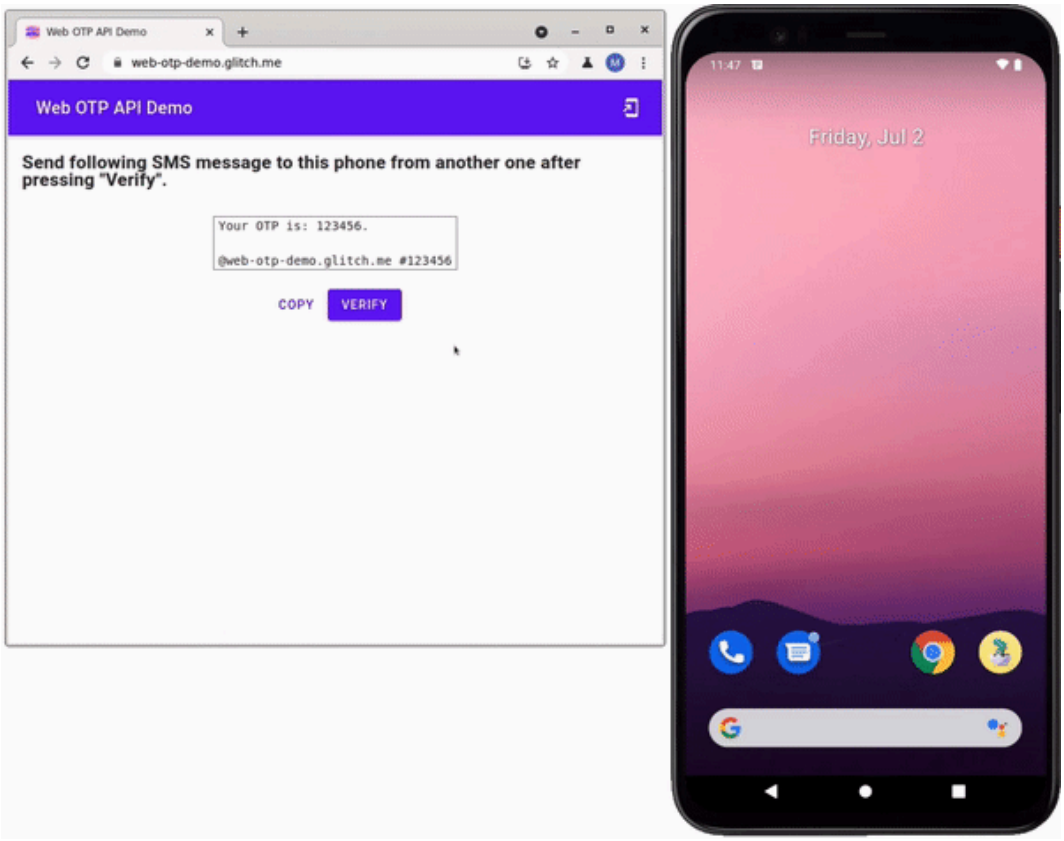


Figure 1. Example of using WebOTP across devices. The API is initially called on the desktop. When the SMS arrives on the phone with the same Google account, a prompt pops up on the phone to ask for user permission. Once the permission is granted, the code is forwarded to desktop and sent to the site.

Background

WebOTP API

The WebOTP API gives developers the ability to programmatically read one time code from specially-formatted SMSes addressed to their origin to reduce user friction. Currently it is only available on mobile devices where SMS is supported. This document extended the early exploration and proposed a solution to enable the API on desktop / laptop through the Sharing infrastructure.

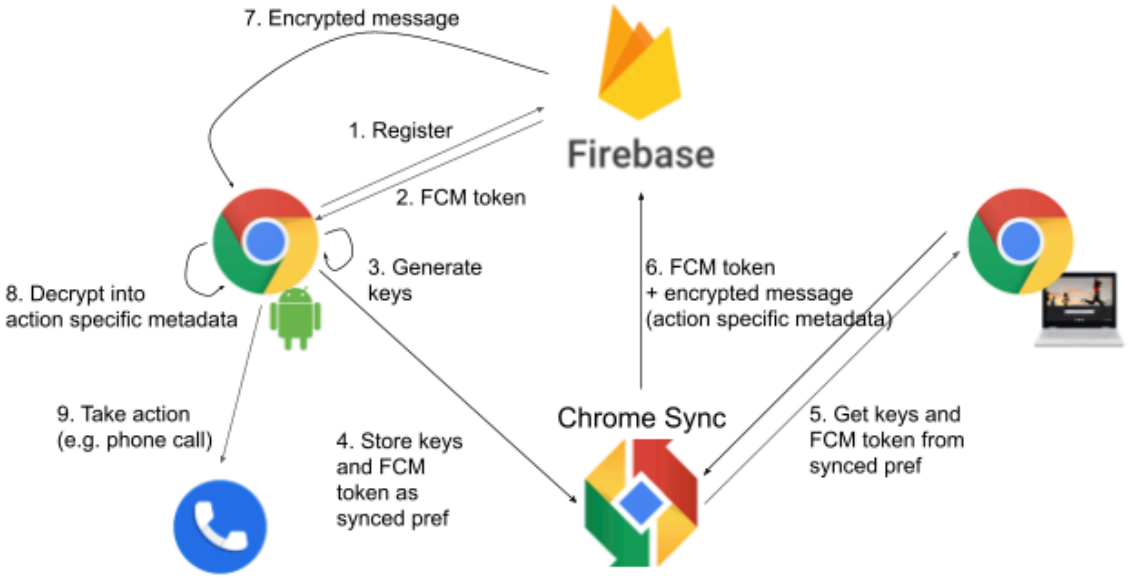


Figure 2. Sharing architecture

Precedents

There are three related precedents that sync information between desktop and mobile.

Google Prompts

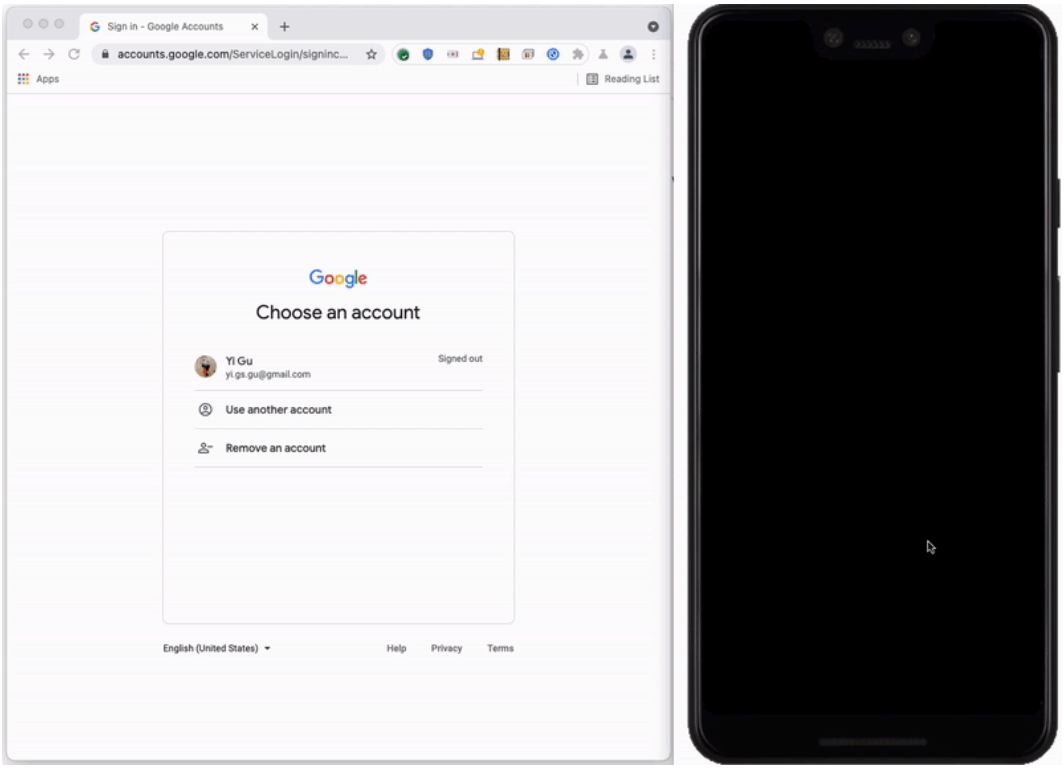


Figure 3. Gmail login via Google Prompts

- Verifies user's identity on mobile devices via a full screen prompt.
- A typical flow goes like this (user actions in bold):
    a. **User triggers verification on desktop**
    b. Google triggers prompt on linked phone
    c. **User unlocks phone and grants permission**
    d. User is logged in and navigated away automatically on desktop

Google Messages

- Allows a user to forward their SMS from Android to web (Chrome, Firefox, Safari Edge etc.)
- 1 phone to 1 desktop tab. No login needed. Permission is revoked when tab is closed / manual sign out / inactive for weeks (see here for more details)
    - Users can then go to the messages **tab** to copy the OTP

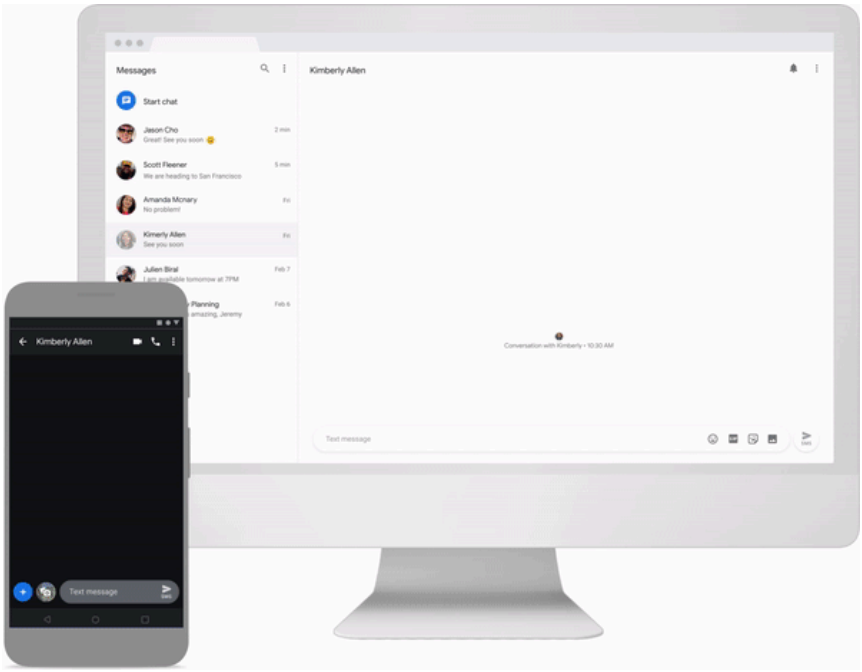


Figure 4. Messages received on Android get auto-synced to a tab via Google Messages

iMessage

- Allows a user to forward their iMessage from iPhone to Mac
- Bound to Apple id
- With this feature, Safari on Mac can **autofill one time code** that is received on iPhone.

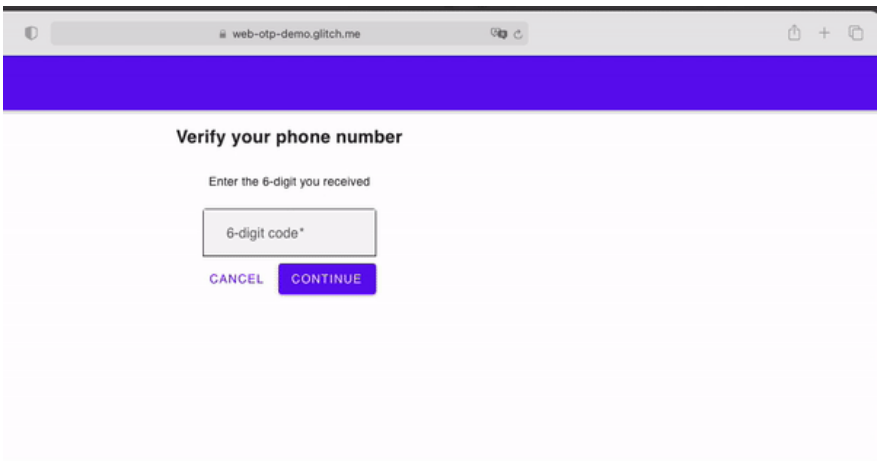


Figure 5. Autofill SMS OTP on Mac with iMessages auto-sync enabled

- A typical flow goes like this (user actions in bold):
    a. **User turns on SMS auto-sync from iPhone**
    b. User verifies identity on Mac
    c. SMS arrives on iPhone

d. SMS is forwarded to Mac
e. User focuses on a specific input field `autocomplete="one-time-code"`
f. A chip appears with expected OTP
g. User taps the OTP to fill it into the input field
h. User submits the form on the website

When a user needs to enter the SMS verification code on desktop they could either memorize the code and enter it manually or use either the Google Messages or iMessages formulation. Note that both techniques require pre-setup by users. i.e. **it's "opt-in" and users are aware**. With proper setup, for iPhone users autofill is just one tap while Android users need to go to the message tab and copy the code from there.

In this document we propose a verification process with minimal friction for Chrome users.

Overview

The Sharing service allows different devices connected via the same Google account to sync data in a safe and fast manner. Data is sent using Firebase Cloud Messaging (FCM) with end-to-end encryption. Based on this infrastructure, we introduced a sender on desktop and a receiver on mobile for the request. Then a sender on mobile and receiver on desktop for the response.

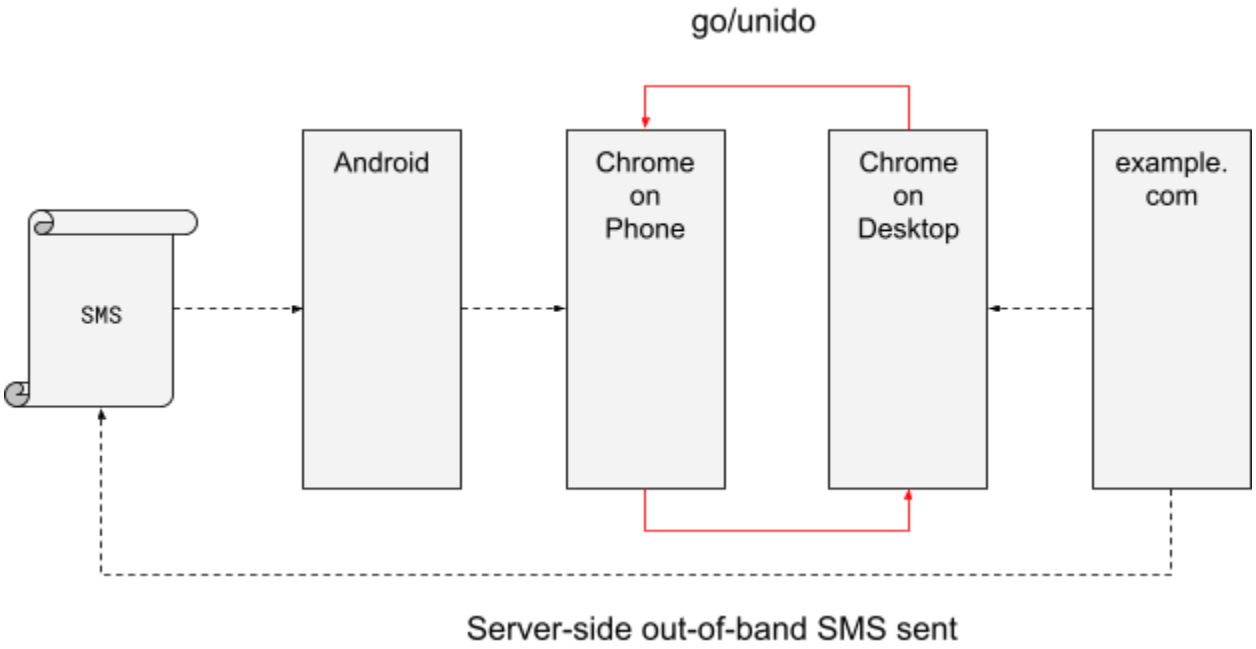


Figure 6. Integration of Sharing with WebOTP at a glance

Proposed verification flow

There are several decision points during the process, from how many accounts we support to how users grant permissions. Figure 7 shows how we made the decisions.

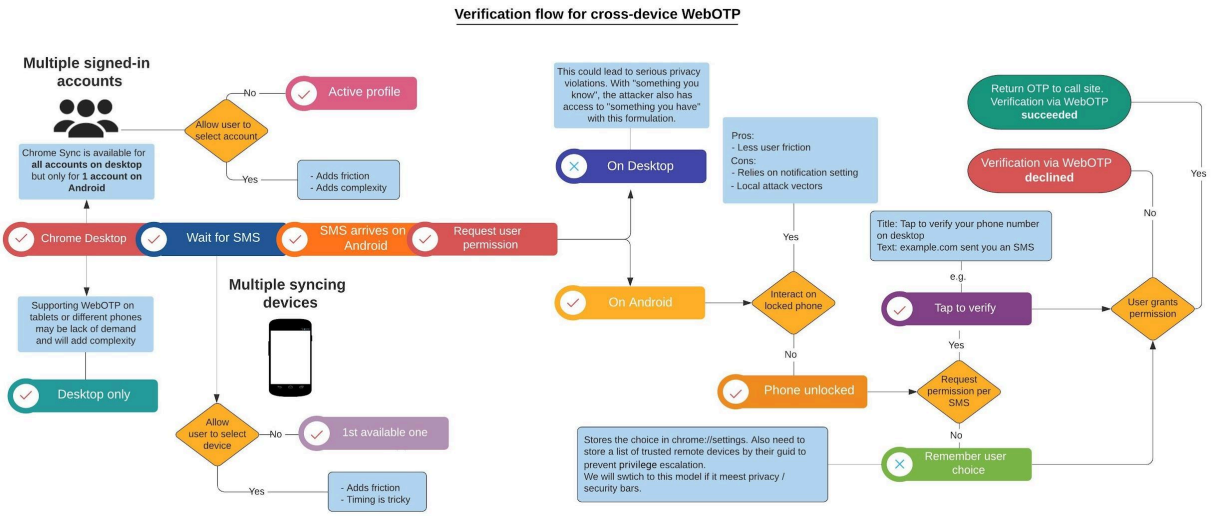


Figure 7. Decision making for cross-device WebOTP verification flow (click here to see details)

Based on the proposed design, the user journey is described as follows:

1. User opens Chrome on desktop and they have a signed-in account as the active profile.
2. User visits a site that calls the WebOTP API for verification
3. If the user has a mobile Chrome on a sms-capable phone and has signed in the same account, desktop Chrome automatically asks mobile Chrome to listen for SMS on the phone under the same account
4. Mobile Chrome receives the expected SMS (using existing WebOTP backend in Chrome)
5. Mobile Chrome checks the origin after parsing the SMS. If it matches the expected one then shows a prompt to ask for user permission.
6. User taps the “ok” button on notification to continue which forwards the origin and one time code to desktop
7. Desktop checks the origin again and if it matches the expected one then shares the code with the site.
8. The user is successfully verified

For code complexity reasons, in this proposal we assume:

1. Non-desktop devices such as tablet are not supported
2. User cannot select which account to use if multiple accounts are signed in on desktop.
3. User cannot select which phone to use for cross-device WebOTP if they have multiple mobile devices under the same account

Note that supporting WebOTP on desktop is a pure improvement compared to manually

entering a code. Our strategy is to enable it for common cases and expand over time based on demand/cost trade-off (e.g., mobile only -> desktop to mobile -> multiple devices...).

To reduce user friction from unlocking their phone each time they receive an SMS, we could memorize user's choice with the following caveats:

- Permission is revocable (via chrome://settings manually or auto expire)
- API calls on new desktop require user's interaction on Android

However we leave this as future improvement beyond dev-trials.

Detailed Design

Frontend

We show a notification when expected SMS arrives and users can grant permission to forward the code to the desktop by tapping it after unlocking the phone. See UX considerations for more details on the front end.

Backend

The desktop and android instances communicate via the sharing service. The data is transferred as shown below.

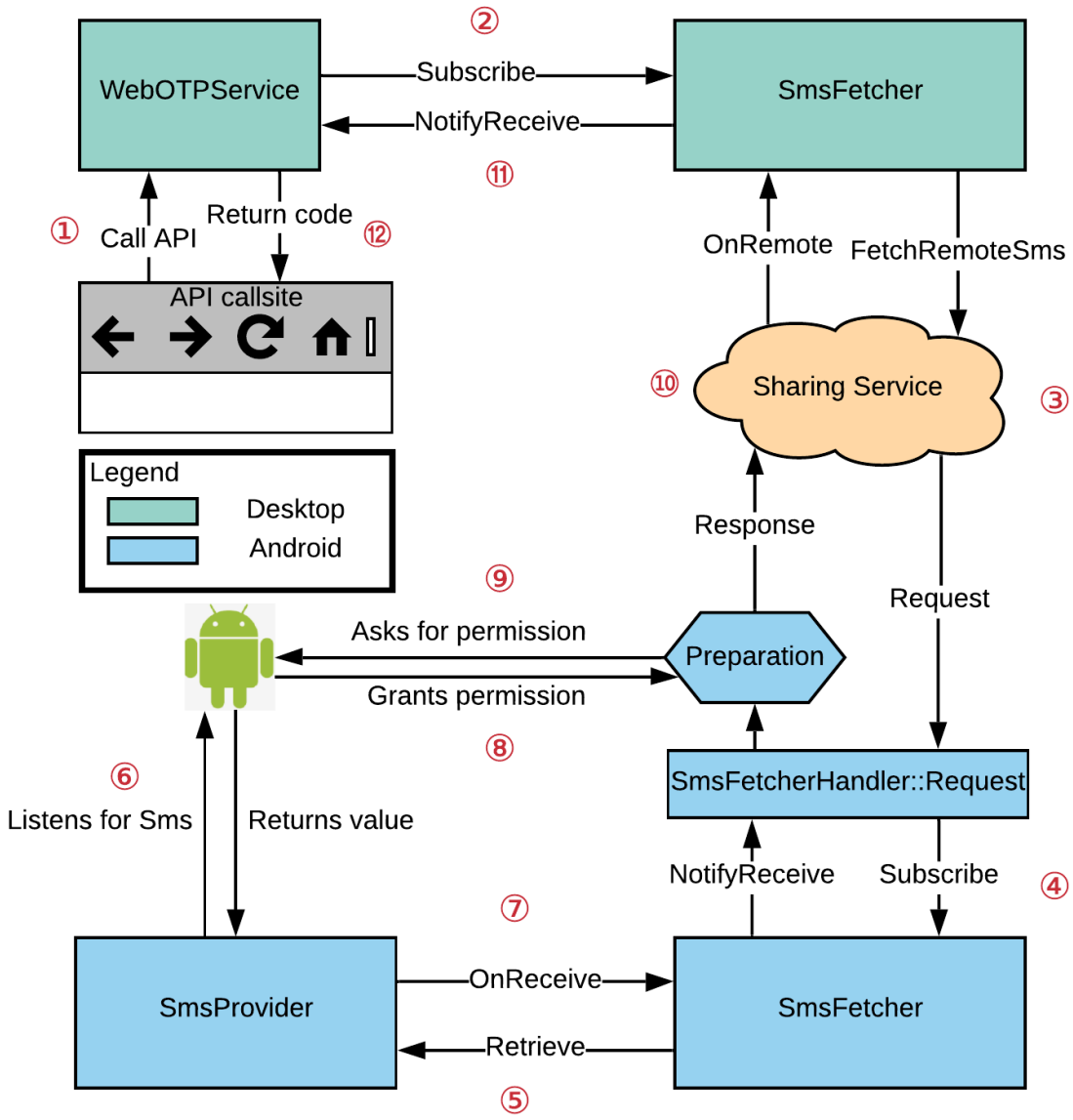


Figure 8. Cross-device WebOTP architecture

1. A user visits a website that calls the WebOTP API.
2. A `WebOTPService` is created on desktop and is subscribed by `SmsFetcher`
3. The `SmsFetcher` on desktop sends a request to the linked phone via `Sharing Service`
4. The `SmsFetcher` on Android subscribes to the incoming `Request`
5. It then asks `SmsProvider` to listen for incoming SMS
6. Android native returns the origin and OTP to `SmsProvider`
7. `SmsProvider` forwards the results back to `SmsFetcher`

8. Before sending the information back to desktop, we show a native prompt to ask for user permission on Android
9. User grants permission on the native
10. The origin and OTP are sent back to `SmsFetcher` on desktop via `Sharing Service`
11. `SmsFetcher` on desktop then notify the `WebOTPService` with OTP is the returned origin matches the call site
12. `WebOTPService` returns the code back to the website to finish the verification process

Desktop

When the API is called, `WebOTPService` as an `SmsFetcher::Subscriber` will try to fetch an SMS from a remote device. We create a `chrome_browser_sharing::SharingMessage` with the expected `origin` and send it to the first available syncing mobile device via `SendMessageToDevice`. It takes a response callback that receives the data from remote. Meanwhile, a notification pops up on omnibox to remind users to interact on mobile.

Once the `SmsFetcher` acquires origin and one time code, it first checks whether the received origin is expected. If so, it passes the code to `WebOTPService` which sends the code to the call site.

Android

API

There are currently two types of backends on Android that the WebOTP API is running on. The default backend prior to M90 is based on UserConsentAPI. An Android native permission prompt is required to grant Chrome access to the SMS. We are migrating to a new backend based on CodeBrowserAPI via Finch and it's expected to be shipped in M90 (see here for improvements). Note that even after M90 there will still be Chrome running with the old backend because we use it as a fallback solution when the new backend is unavailable for some users. e.g Chrome is not the default browser, play service is out of date etc.

The User Consent API requires an Android naive permission prompt which introduces extra complexity and user friction. Given that this will be a fallback backend with little usage, we only support mobile Chrome that runs with the new backendI.

Permission

When mobile chrome receives the expected SMS, `SmsProvider` parses it and sends the `origin` and `one_time_code` to `SmsFetcherImpl`. If the `origin` is in the subscriber list, the corresponding subscriber `SmsFetcherHandler::Request` will take the pair and initiate a permission prompt on Android via `SmsFetcherMessageHandler`. Upon user granting permission, `SmsFetcherHandler::Request` packs the origin and code into the response message and sends it back to desktop Chrome via Sharing service.

UX Considerations

We have been running a dev-trial with the proposed UX. There are some other UX flows that we have been thinking about. At the moment we believe that users have to grant permission on the phone for privacy reasons. This aligns with the “something you have” concept in the 2-factor authentication. See below for the considered options.

Android

Option 1: Notification (proposed)

As shown from the proposed UX, when the expected SMS arrives at the phone, we show a notification on the top and users can interact with it to grant permission to forward the OTP to the desktop.

- Note that it may overlap with the Android SMS notification which also appears on the top of the screen when the phone is unlocked.
    - We could delay showing the notification by 1-2 seconds to make sure it appears on top of the SMS notification.
- We propose to show [CODE] + [ORIGIN] + [REMOTE DEVICE] in the notification. See example below.

Buttons

To grant permission, **users have to explicitly click the `OK` button** on the notification. In previous iterations we only asked users to “tap” the notification to grant permission but it was too frictionless such that users may accidentally approve it which negates the value of 2FA.

There are two options in general:

- Two buttons (Yes/No or Allow/Deny based on how we describe to the users)
- Single button (Yes or Allow) and keep “swiping away or ignore” as the negative signal

Strings

To show the remote device information to the user, we could choose from one of the following: based on the Sharing infrastructure: manufacture name, model name, client name or OS info.

- Manufacturer name. e.g. `Apple Inc.`
    - It’s hard to integrate this name with a proper string
    - Not applicable to certain customized computers
- Client name
    - Pros:
        - Consistent with other Sharing features (shared clipboard, click to call etc.)
        - Recognizable if customized. e.g. `Jane Doe’s Macbook`
    - Cons:
        - This identifier is not recognizable by users which could be confusing

            - `Hewlett-Packard Computer HP Z840 Workstation`
            - `MacBookPro14,3`
            - `LENOVO Computer 20216`
        - Unlike other features, WebOTP needs to show an origin and a one time code in addition to the client name. This may be too much for a notification element.
- Model name. Similar to the client name, this may not be recognizable:
    - `HP Z840 Workstation`
    - `MacBookPro14,3`
    - `20216`
- OS info. e.g. `mac, linux`
    - This is vague but acceptable given the fact that the user has enough context on the operation.

Examples

*Figure 9. Notification when phone is **locked** (string TBD)*

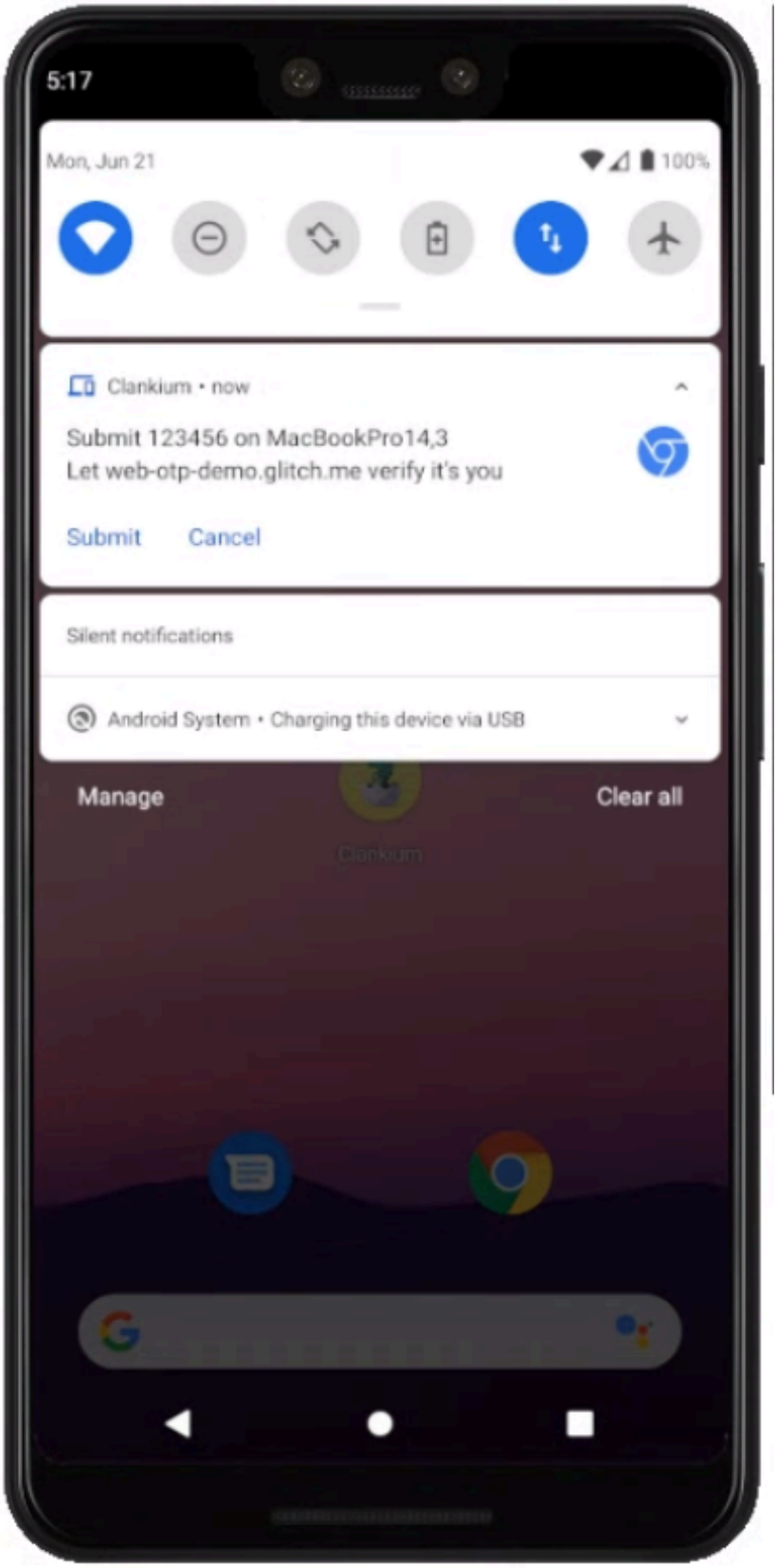


*Figure 10. Notification when phone is **unlocked***

Option 2: Prompt (future extensibility)

Similar to Google Prompts, instead of the notification (small room for necessary information), we could show a full screen prompt to ask for user permission like the following (new string required)

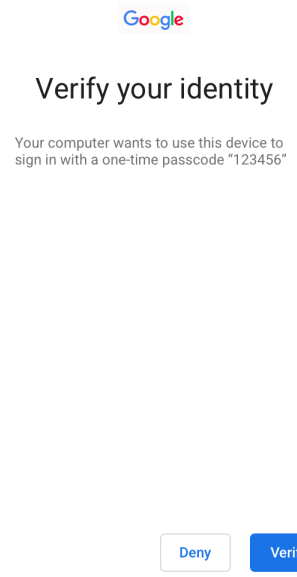


Figure 11. Full screen notification on Android to ask for user permission

However, this prompt is from GMSCore and Chrome cannot trigger it directly. Similar to PaaSK, we could trigger it via a native notification. i.e. user clicks the notification first to see the full screen prompt and they grant permission to forward the OTP to the desktop from there.

- This requires two taps on the phone which leads to more friction than the previous solution.
- Mitigation
    - We are trying to bypass the tapping on the notification. This requires Chrome to be granted some permissions such as `START_ACTIVITIES_FROM_BACKGROUND` or `SYSTEM_ALERT_WINDOW`. It benefits PaaSK and will be consistent with Google Prompts which brings unified verification / authentication experience for users. **However, given that Android is restricting access to sensitive permissions for Apps, this work may take an unknown amount of time.**

Desktop

Option 1: No UI (proposed)

On the one hand, with the proposed flow, given that users need to interact on the phone to grant permission, they are fairly aware (with proper strings) of the fact that Chrome is assisting them with identity verification. Therefore, **showing UI before user action on Android is not necessary**.

On the other hand, when OTP is received on the desktop, the user is likely to be navigated away to a different page. If we decided to show a confirmation UI, it must persist across navigation in order for them to read it. Showing a confirmation UI on the different site, may cause the false perception that Chrome has shared some of their data to it.

Option 2: Reminder on desktop (considered alternative)

Technically speaking we could remind users on desktop that Chrome is assisting them to automate the verification process.

- Upon API is called
    - Let users know that Chrome on Android is able to automatically send OTP to desktop upon their permission
        - Other features such as “click to call”, “shared clipboard” show a notification on the omnibox. See examples here.
        - An alternative would be showing notification inline with the input box
            - However, at the moment the WebOTP API doesn’t know about which field is for OTP therefore we don’t know where to show the reminder
        - Another alternative would be showing a modal on the page
            - According to the partner feedback from Origin Trial, this may interrupt users from doing other activities on the page, e.g. they may fill up other forms while waiting for the SMS

Constraints:
There are some limitations of the WebOTP API at the moment:

- It doesn’t know about which field on the page is the OTP field. As a matter of fact, the API was designed in a way that it works without an input field.
    - This constraint makes it really difficult to show any inlined UI.
    - Mitigation: We could update the API surface to accept an OTP field.
- The Sharing service cannot tell whether the user provided phone number matches the linked mobile device.
    - This leads to misleading information. e.g. the phone number of the account’s linked device is 1234-5678 and the user enters 8765-4321 on the page for verification. If we remind them that they could interact with a Chrome prompt on their phone to automate the process, we are misleading the user. Because they

can only see the prompt when the SMS arrives at the linked phone with number 1234-5678.
    - Mitigation: We could only show the “done” reminder when desktop chrome receives the OTP from mobile.

Option 3: Dropdown on desktop (future extensibility)

As shown above, Safari allows users to see the OTP when they focus on the OTP field on Mac. Instead of sending the OTP back to the website automatically, they ask users to click a dropdown pill and then submit the code manually.

Constraints:

- This solution is based on the fact that users have already turned on iMessage auto-sync from their iPhone to desktop. i.e. users do not need to grant permission on their phone. However, our current solution requires users to interact with the phone for privacy reasons. i.e. to implement the dropdown solution we have to ask users to interact on desktop again and “two permissions” makes less sense than “one permission” (either on mobile or desktop).
    - Mitigation: We are exploring the possibility of bypassing the interaction on the phone for Android users. Similar to Google Messages, we could allow users to remember remote desktop on mobile after the first interaction. Requests from the same trusted desktop in the near future can bypass the permission on mobile. There are privacy implications with this solution. See here for details.
- This solution is gated by the user focusing on the input box which normally is the last step after the user has obtained the OTP manually.
    - Mitigation: We could remind users to focus on the OTP field instead of obtaining it by themselves. Note that we should only remind users when desktop Chrome has received the OTP from mobile.

Security Considerations

The Sharing infrastructure uses end to end encryption, so the one-time-code cannot be read anywhere along the network, not even by FCM. For further details see here.

- There are two types of requests during the cross-device verification process.
    1. Chrome requests Sharing service to communicate with other devices
    2. Chrome requests Android to listen for SMS

  Both requests are initiated from the browser process.

- Consider the two scenarios:
    - Desktop -> Mobile (ask mobile to listen for incoming SMS with expected |origin|)
        - Compromised desktop instance
    - Mobile -> Desktop (parse SMS, if the parsed origin matches then send code to desktop)

        - Compromised mobile instance
    - In these two scenarios, the added risk is marginal and below our threshold of concern. E.g, if desktop Chrome is compromised, the attacker is able to steal the OTP and use it to log into the victim's account. However, the attacker can already do such damages or even worse ones without OTP and without the new functionality.

Note that offline devices will not receive requests from other devices.

Privacy Considerations

In general, the sharing infrastructure does not disclose the origin and one-time-code without explicit user consent. For further details on Sharing see here.

The general WebOTP API binds a user with their phone number when the user completes the verification process. i.e. the site learns about such combinations when the one time code is returned by Chrome. Note that this only happens when the user explicitly allows the verification flow provided by Chrome.

With cross-device support, the user's Google account is also involved in the process. However, this is not new information for the sites regardless of federated login.

User permission

The major question to answer is at which point in the OTP verification process the user permission grant is requested. There are two possible options:

- Ask for user permission on Android
    - Pros
        - Better privacy
    - Cons
        - Add another layer of friction since the user needs to unlock the phone
            - Mitigation 1: Ask for permission once and remember user selection with two caveats:
                - Permission is revocable (via chrome://settings manually or auto expire)
                - API calls on new desktop require user's interaction on Android
            - Mitigation 2: Allow users to grant permission on locked phone
                - This somehow breaks the "something you have" in 2SV. Ideally the site is expecting "something you have access to" but this formulation changes it to "something you have

with you”.

- Ask for user permission on desktop
  - Pros
    - Fast and instant
  - Cons
    - Anyone who can access the computer can complete phone number verification without possessing the phone

As mentioned in the background section, **both Android and iOS support syncing messages** to desktop without recurring permission request on the phone. i.e. with proper setup, **the SMS can be forwarded to the desktop even when the phone is locked or not nearby**. It highly reduces the potential user friction because unlocking the phone and granting permission every time negates the user convenience of using the feature.

However, bypassing the interactions on mobile allows local attackers to finish the verification without the phone owner’s awareness. e.g. a kid can use the parent’s desktop to buy things if the payment is guarded by phone number verification. Similarly, anyone who has access to the desktop can sign up on random websites using the linked phone number.

In addition, this may lead to privilege escalation where a remote attacker compromises the user’s google credentials. Previously they only possessed “something you know” and 2SV is able to protect the user. But bypassing user interaction on Android means they will also “possess” the “something you have” implicitly.

Therefore we propose to ask for user permission on Android.

API exposure

Unlike Android users, desktop users may not be able to use WebOTP if they do not sign in or have syncing phones. In this case, exposing `window.OTPCredentials` to the websites on desktop may lead to privacy issues without proper strategy.

Unsuccessful flow

If the verification flow is not completed, the website won’t learn anything from the user. e.g. if the user never sees the Android notification (because they are not signed in or they don't have linked phones etc.), or they dismiss / ignore the notification, we do not reject the promise immediately. Rather, we wait for 4 mins and send a unified "Timeout" message back to the website.

Successful flow

If the user grants permission and completes the verification process via WebOTP API. The website can learn extra information about the user (basically the requirements of using this API):

- The user is using Chrome
    - This info is available by other means
- The user is signed in to **Chrome** with **a** Google account
    - This info is available by other means
    - The website shouldn't know about which Google account it is
- The provided phone number is linked to that Google account
    - This info shouldn't be useful to the website

Testing

We have automated unit tests, browser tests and WPT tests to protect us from regressions. Here are the requirements for manual tests.

Desktop requirements

- Google Chrome (version >= 93.0.4555.0)
    - Other Chromium based browsers are not supported at the moment
- User is signed in with their Google credentials via https://myaccount.google.com/
    - No need to turn on "Sync"
    - It could be a typical gmail account or other Google managed accounts such as some university email accounts
    - If multiple accounts are signed in, only the primary account (as shown in chrome://settings/) will be used. To use other account, users have to add a new profile for that account to make it the primary account under that profile
- Validation
    - Go to chrome://sync-internals/
    - The "Transport State" under "Summary" (top left) should be "Active"

Android requirements

- Google Chrome (version >= 93.0.4555.0)
    - Other Chromium based browsers are not supported at the moment
- **Chrome is set to the default browser**
- User is signed into **Chrome** with the same Google credentials as the one on desktop
    - No need to turn on "Sync"
- User is signed into **Android** via "Settings->Google"
    - Multiple accounts are OK. Similar to desktop, only the primary account in Chrome (as shown in 3 dots -> settings) will be used
- Go to the Settings > Apps & notifications to check the Google Play services version, it should be >= 20.30.12
- Validation
    - Go to chrome://sync-internals/

- The “Transport State” under “Summary” (top left) should be “Active”

Basic flow

1. On desktop Chrome, visits https://web-otp-demo.glitch.me in a **normal** window and tap “Verify”
   - The feature does not work in incognito mode.
2. On Android, makes sure Chrome is running (foreground or background) as the default browser
3. Sends the following SMS to the testing phone from step 2:
   - @web-otp-demo.glitch.me #123456
   - The sender cannot be on the receiver’s contact list
   - https://voice.google.com/ supports sending SMSes
4. When the SMS arrives, an extra notification will show up. Example here.
5. Taps “Submit” (unlock the phone if needed)
6. The notification disappears and the verification page on desktop Chrome is auto-progressed to the “Success” page.

Caveats

This feature works the best when users have only one signed-in mobile device and there is only one Chrome client on that phone. This is because

a. Users cannot choose which end client that they want to use
b. Client selection by Sharing service is transparent to users. It auto chooses the one that updates the entry in the Sharing service the last.
c. Chrome on Android has to be the default browser.

Examples:

- A user has both Chrome Stable (default browser) and Chrome Beta on the same mobile device. If they only launch Stable, everything works fine. If they launch Beta **after** that, it may not be working because now Chrome on desktop may send the request to Chrome Beta on Android which is a deadend.
- A user has two phones. They first launch Chrome (default browser) on device A and launch Chrome (default browser) on device B. Now the request from desktop goes to device B. It’s not obvious to users how to reuse device A for this functionality. They could close Chrome on both devices and relaunch Chrome on device A but that seems like overkill.

This is considered acceptable in the initial launch and we will reevaluate the situation with metrics post launch. **For testing purposes, one can go to chrome://sync-internals/ on the desired phone or Chrome client and tap “Stop Sync (Keep Data)” then tap “Request Start” to make the Sharing service use this phone or Chrome client.**

Considered Alternatives

This section captures brainstorming on a variety of topics related to this project. In particular, we have considered whether to support multiple platforms / accounts / devices and how users should get involved in the process.

User Interaction

Where to ask for permission

Interactions on Android (this proposal)

On desktop (see below)

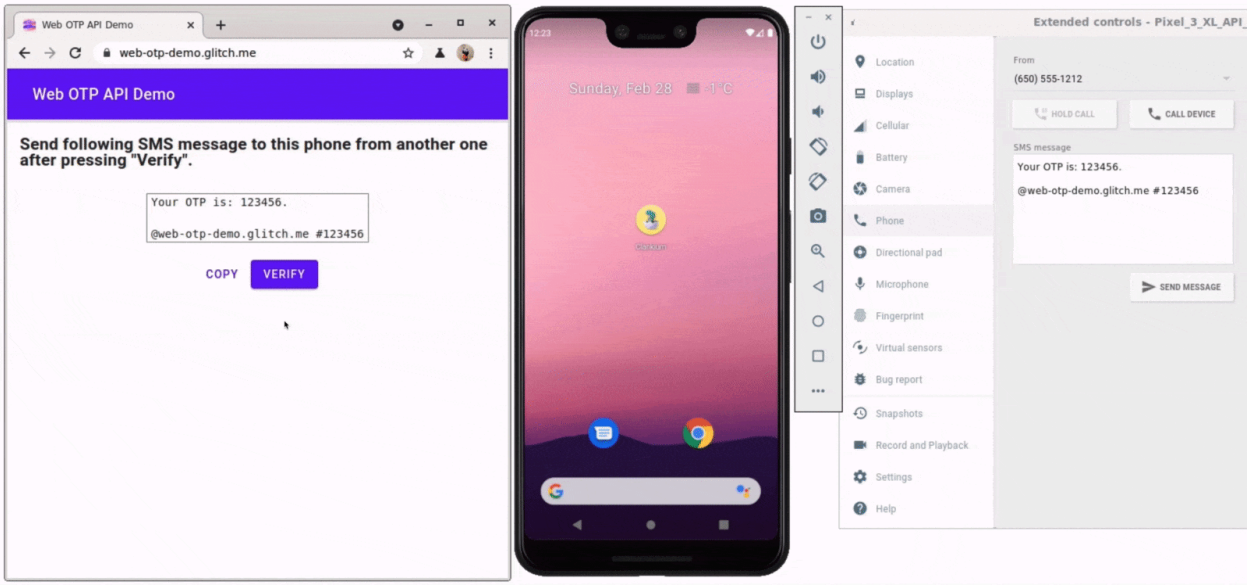


Figure 13. Bypassing the interaction on Android and asking for user permission on desktop

1. User opens Chrome on desktop and they have a signed-in account as the active profile.
2. User visits a site and it calls WebOTP API
3. If the user has a mobile Chrome on a sms-capable phone and has signed in the same account, desktop Chrome automatically asks mobile Chrome to listen for SMS on the phone under the same account
4. Mobile Chrome receives the expected SMS (using existing WebOTP backend in Chrome)
5. Mobile Chrome checks the origin after parsing the SMS. If it matches the expected one then forwards the SMS to desktop
6. Desktop checks the origin again and if it matches the expected one then shows a prompt to ask for user permission to share the code with the site.
7. Desktop sends the code to the web site upon user granting permission

Mitigations:

- Chrome on Android only forwards the one time code when the origin in the SMS matches the requested origin. e.g. while the phone is listening per desktop's request, the user receives a personal SMS *"Hello…."* and the expected verification SMS *"@example.com #1234"*. Only the code *"1234"* is sent to desktop if the user is trying to verify on example.com.
- After Chrome on desktop receives the code, it explicitly asks the user for permission to send the code to the site to finish the verification flow. This step is the same as WebOTP on Android.
- Notify the user on Android with drop-down notification (see example below)

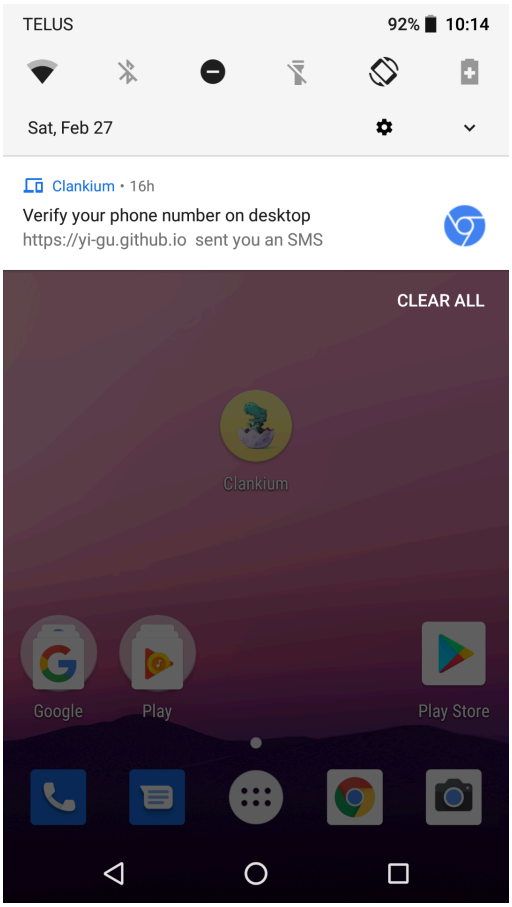


Figure 14. Notification that keeps users informed (exact string TBD)

Some notes:

- Chrome on Android learns about the **expected** SMS immediately after the SMS arrives because it's allowlisted on Android[1].

[1] This only applies to SMS with expected format "@exmaple.com #1234". i.e. Chrome cannot see any other SMSes received on the phone. More details here.

When to show notification on desktop

- When API is called
- When SMS is received

How to show notification on desktop

- Ommibox icon
- Centered modal
    - Interaction is blocking
    - Interfere with existing form filling

What to show as notification on desktop

- Ask whether user wants Chrome to verify phone number for example.com
    - Note that Chrome on Android has already obtained the SMS code by now
        - This is not concerning because the potential threat is exposing the interaction to the site instead of Chrome's awareness of the code.
        - ~~If we want Chrome to be able to fetch the SMS code after user granting the permission, we need another round of messages syncing between desktop and mobile:~~
            - ~~Desktop notifies mobile to listen for SMS -> SMS arrives -> mobile notifies desktop to ask for user permission~~
            - ~~Desktop notifies mobile that user has granted permission -> mobile sends the code back to desktop~~

- ~~Default to allow. Just notify the user that Chrome has helped with the verification. e.g. "Your Google Pixel 4 has received the expected SMS. Chrome is helping you with the verification process."~~

    - ~~We may need to ask users at least once for such permission. Other similar products require manual setup for syncing to work.~~
    - ~~Note that if the web site skips showing the code being filled, users may get navigated to the after-verification page immediately after the SMS arrives on their phone. This could be TOO smooth than expected.~~

One-to-many

Supported platforms

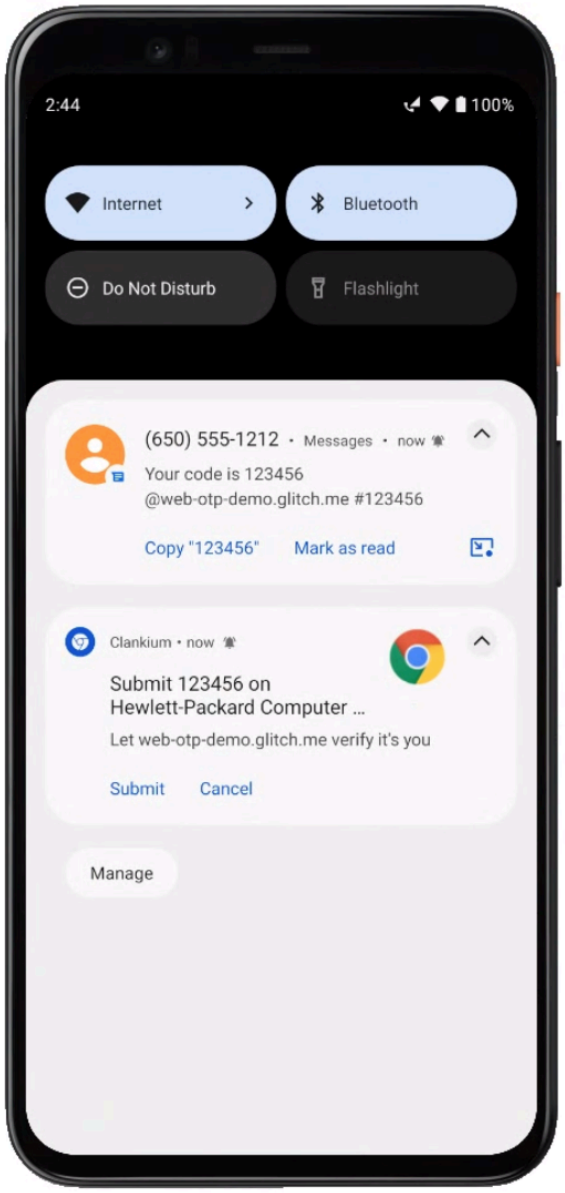
Currently the use case is focused on desktop users. Should we also support other Android devices such as tablets or different phones?

- Pros
    - Better ecosystem with the truly “cross-device” support
- Cons
    - Lack of demand?
    - Increased implementation complexity

Multiple syncing devices

- Should we allow users to select which device (phone) they want to use?
    - Pros
        - User has more control
    - Cons
        - Adds frictions
        - Timing is tricky. Only works when selecting a device before asking the site to send out SMS

UI


2:44
100%
Internet
Bluetooth
Do Not Disturb
Flashlight
(650) 555-1212 • Messages • now
Your code is 123456
@web-otp-demo.glitch.me #123456
Copy "123456"
Mark as read
Clankium • now
Submit 123456 on
Hewlett-Packard Computer ...
Let web-otp-demo.glitch.me verify it's you
Submit
Cancel
Manage