Privacy-preserving enumeration of Payments

Mathieu Hofman (Invited Expert)

Goals for Payments on the Web

- **Low Friction**
  fewer clicks, swipes, taps, no typing
- **High Security**
  cryptographic certainty, risk-based policies, two-factor authN
- **Strong Privacy**
  only share data as required, always with consent

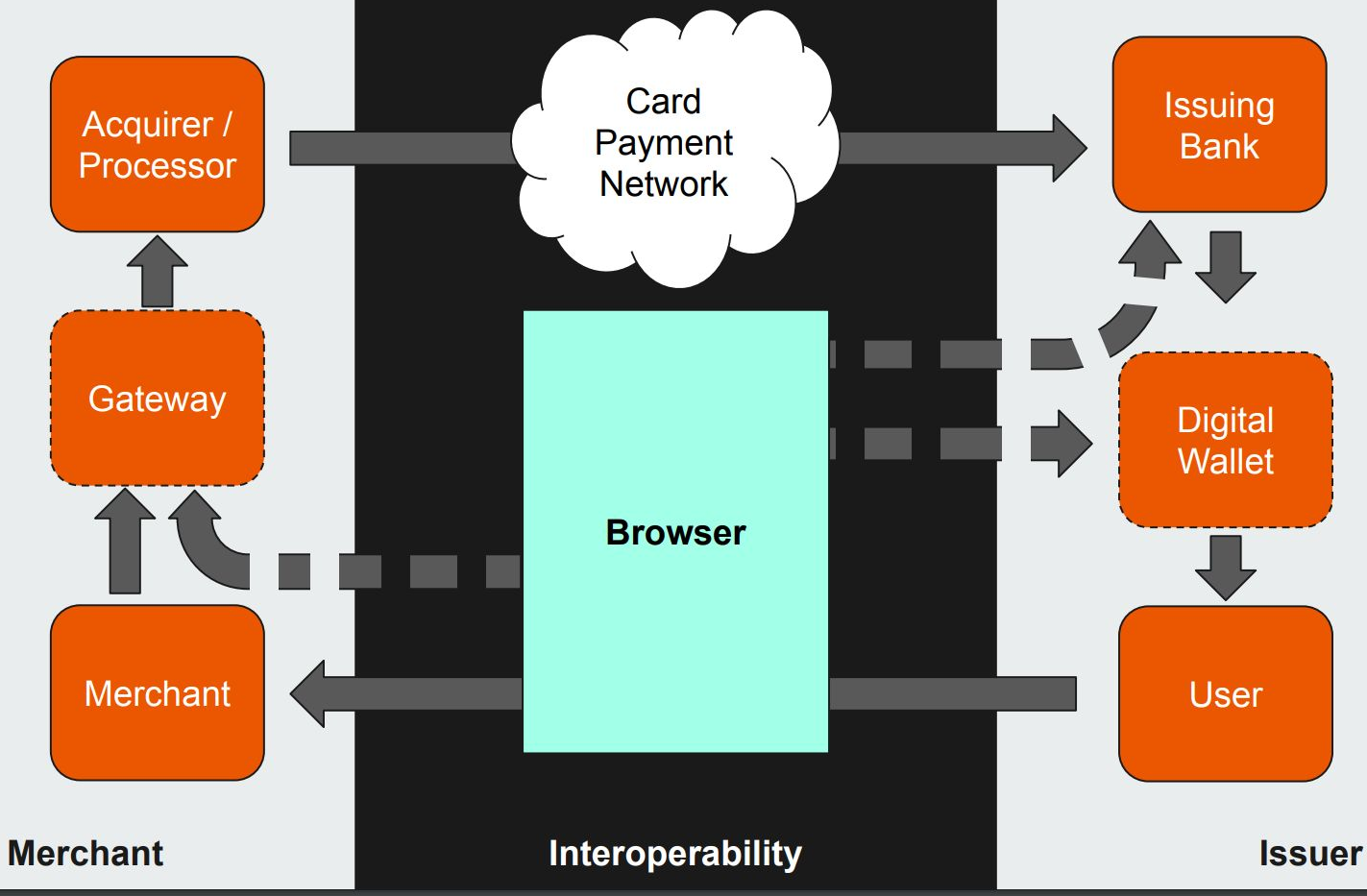
Invariants and Assumptions

- **Origin Security Policy** (browsers use origins to segment security domains)
- **Risk** assessment is facilitated by data collection
- Browsers **limiting 3rd-party access** to data/cookies to prevent tracking

Related tensions in the ecosystem:

- Same Origin Policy (SOP) <-> Payment Initiation often done by 3rd-party from 3rd-party context (iframe)
- Excess data collection is bad for **privacy** <-> Risk assessment consumes as much data as it can get

3 domains


Acquirer / Processor
Card Payment Network
Issuing Bank
Gateway
Browser
Digital Wallet
Merchant
User
Merchant
Interoperability
Issuer

Requirements from the browser

Maintain security and privacy

- Sensitive information not disclosed without user consent
- Merchant or its third parties shouldn’t be able to learn anything about who the customer is unless the customer is performing a payment action.
- Payment handlers shouldn’t be able to learn anything about which merchants the customer visits, unless the customer performs a payment action with an instrument managed by the handler

Specific threats: fingerprinting and 3rd party tracking

Assumptions for customer

May have different types of payment instruments:

- Saved with the merchant
- Saved in the browser, e.g. through a configured Payment Handler
- Not saved anywhere

Customer may not want to save their instrument in the browser (e.g. guest checkout on a friend’s computer)

Assumptions for merchant

May not support some of the instruments the customer possesses.

Wants to increase the likelihood the customer will complete the transaction (conversion):

- Offer as many supported options as possible
- Reduce friction where possible (increase cohesiveness of experience, remove unnecessary steps)
- Keep customer in experience they control and may optimize for

Merchant experience

Show **UI for supported payment selection**, **without user interaction** (drive-by), in a way the customer can understand what selection they are making

- Non WebPayments methods offered alongside, to support instruments the customer hasn’t saved, or for methods that do not support WebPayments API
- Avoid onboarding an instrument into WebPayments
  the flow can be onerous and leads to high drop offs (e.g. phone call verification to onboard some cards in a mobile wallet)

Handler implementation constraints

Needs to decide which or if any instruments are available when requested

- Proprietary business logic, which probably can’t be formalized
- Based on its notion of the current user (data scoped to the handler’s origin)

Example merchant integration

Show a list of branded card wallet buttons

Only display a button for which the user has a card ready to pay

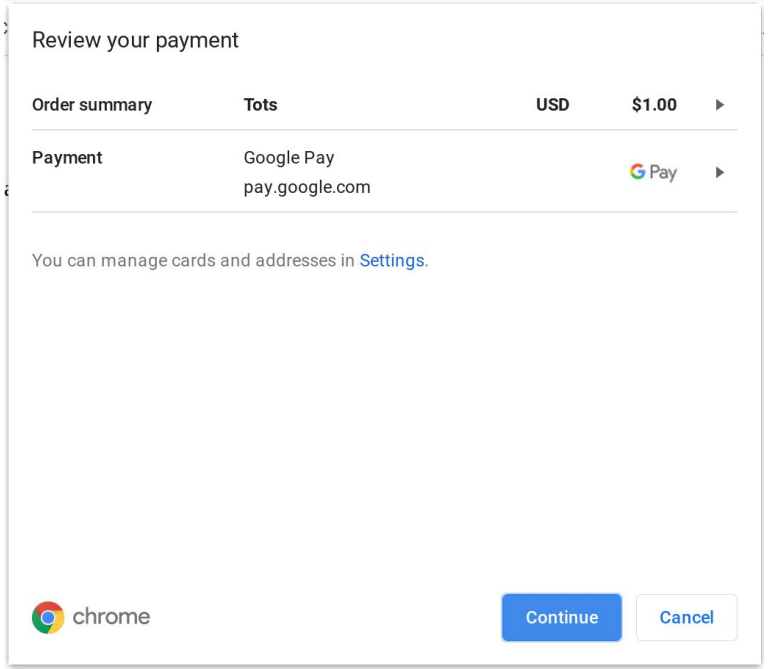
Merchant’s processor in the merchant’s country integrates with 3 wallets and only supports 2 card networks

Current Situation

`hasEnrolledInstrument()` has a quota to prevent fingerprinting

Requires merchants to choose between unbranded buttons, or allowing instrument onboarding in all wallets.

With multiple methods behind single action, browser has to implement a payment method selection sheet.



Proposal part 1: Enumeration

```
enumeratePayments(
  methodData: PaymentMethodData[],
  options: PaymentEnumerateOptions
): Promise<PaymentOpaqueIdentifers[]>
```

Can still pass arbitrary payment method data as in `PaymentRequestConstructor`, e.g. to indicate which card networks are supported

Shared options to control things like requiring enrolled instruments

Returns a list of opaque identifiers representing the available payment methods

Proposal part 1: Enumeration

Browser does mediation between merchant and payment handlers

Browser invokes handlers, which runs custom code to check if logged in user has any compatible enrolled instrument and return true or false

Browser generates temporary unique ids (valid only for this page load) to represent each payment method that returned true.

Enumeration with different data / options is rate limited

Proposal part 2: Presentation

```
<payment-request-method-button
  method-id={enumerateResult[0]}
  type="donate"
  theme="light"
  height="40px"
></payment-request-method-button>
```

Merchant can customize theme (e.g. dark) and “Pay with” wording type

Merchant creates `PaymentRequest` with single `supportedMethod` containing opaque identifier chosen by customer

Proposal part 2: Presentation

WebComponent provides DOM encapsulation

No intrinsic size, replaced elements in CSS model to avoid the website inferring information from the component’s layout

Open question: What data should the handler expose to allow the browser to render this component

Review of Isolation so far

✔️ drive by website has limited information about the user

Only how many compatible wallets with a set of given constraints the user has

Cannot vary constraints to gain more bits of information

Native Web Components encapsulation allows displaying user actionable data without leaking fingerprints to the website

❌ handler still learns about website the user visits, can establish side channels

Proposal part 3: Worklets

Payment Handlers would be based on Worklets, not Service Workers

Granted read only access to storage

No network access or other communication channels

Can only answer browser’s invocation

After customer makes selection, aka merchant invokes PaymentRequest with opaque id, Payment Handler has access to normal APIs (since it’s showing in a first party modal)

Open questions

What interface does the Payment Handler modal use to communicate with browser?
Keep Service Worker for this only?

How do we standardize Payment Method presentation details?
Manifest, Worklet APIs?

Are Payment Details needed at enumeration time?
Do Payment Handler Worklets need any of the information to determine availability

How to allow Handlers to perform merchant verification in privacy preserving way?
Independent step where handler has network access but no local data access?

Payment Instrument Enumeration

Can this concept be expanded to instrument enumeration?

How to visually merge instruments between multiple methods or with the merchant’s own saved payment instruments for the customer

Spectre

Does not protect against Spectre type attacks.

Maybe make `enumeratePayments` mutually exclusive with `SharedArrayBuffer`? Aka make Handlers equivalent of `CORP: same-origin` and consider handlers as having their own nested document

Or require User Agents to run Handlers in a different process than renderer? Attacker may still be able to gather extra bits from Web Components but that might be acceptable