ChatGPT

## CSV Output

banxa.com,**Prefill Wallet Address**,walletAddress,https://docs.banxa.com/docs/referral-link, 200,"`...banxa.com/?coinType=ETH&fiatType=USD&fiatAmount=500&...&walletAddress=0xf831d1781287e499bd546cc4831cf783af8` [1] ",confirmed,false
coinify.com,**Skip Address Input Screen**,address,https://developer.coinify.com/apidoc/trade/, 200,"*Providing the* `address` *value will skip the screen where the customer is asked to provide the receiving crypto address.* [2] ",confirmed,false
coinify.com,**Skip Quote Selection Screens**,"buyAmount + defaultFiatCurrency + defaultCryptoCurrency",https://developer.coinify.com/apidoc/trade/,200,"*If* `buyAmount`*,* `defaultFiatCurrency`*, and* `defaultCryptoCurrency` *are provided, the Quote UI will be skipped.* [3] ",confirmed,false
mercuryo.io,**Prefill Multiple Wallet Addresses**,address,https://github.com/mercuryoio/api-migration-docs,200,"`<iframe src=\"...?address=BTC:2N7gzbQsDnfAFdwabcFqyw1Q8y1ZUpeqUgD,ETH:0x44eae1E05F5f294f0f2a054D16605993FCd627a9&...\" />` [4] ",confirmed,false
switchere.com,**Skip Wallet Address Entry Step**,isSkipSecondStep,https://developer.switchere.com/widget/parameters/visualization/,200,"`isSkipSecondStep`*... skip the* `second` *step of the widget, such as entering a crypto address. This parameter will work only if* `dstAddress` *is not null...* [5] ",confirmed,false
wert.io,**Prefill Wallet Address**,address,https://docs.wert.io/docs/nft-checkout, 200,"`...sandbox.wert.io/YOUR_PARTNER_ID/widget/order?address=0x96D5990185022212d367A0e09263B12Dbb4EE06A&commodity=ETH&network=sepolia&commodity_amoun` [6] ",confirmed,false
ramp.network,**Prefill Wallet Address**,userAddress,https://docs.ramp.network/configuration, 200,"`userAddress` *– an optional string parameter that pre-sets the address the crypto will be sent to... Example:* `{ userAddress: 'user blockchain address' }` [7] ",confirmed,false
ramp.network,**Prefill Email (Skip Login)**,userEmailAddress,https://docs.ramp.network/configuration, 200,"`userEmailAddress` *– an optional string parameter that pre-fills the email address for your user to make their onramping experience even quicker.* [8] ",confirmed,false
swipelux.com,**Prefill Wallet Address**,cryptoAddress,https://docs.payb.is/docs/web-direct-url-standalone-integration,200,"*Example:* `widget.paybis.com/?partnerId={UUID}&cryptoAddress=32eVTqDWHvDVzAh2rsU6Cd2nRiWVxEyWDD&currencyCodeFrom=EUR&currencyCodeTo=BTC&signat` [9] ",confirmed,false
onramper.com,**Prefill Wallet Address(es)**,wallets,https://docs.onramper.com/docs/supported-widget-parameters,200,"`wallets` *string – The wallet address(-es) to be passed to the widget...* [10] ",confirmed,false
stripe.com (onramp),**Prefill Wallet Address**,wallet_address,*(no public docs)*,404,"Not found in the connected sources – Stripe's onramp API uses a session creation call to set the destination `wallet_address`, but no public query parameter is documented [11] .",not-found,false
paybis.com,**Prefill Wallet Address**,cryptoAddress,https://docs.payb.is/docs/web-direct-url-standalone-integration, 200,"*Example:* `...&cryptoAddress=32eVTqDWHvDVzAh2rsU6Cd2nRiWVxEyWDD&currencyCodeFrom=EUR&currencyCo` [9] ",confirmed,false
transak.com,**Prefill Wallet Address**,walletAddress,https://docs.transak.com/docs/pass-information-on-behalf-of-the-user-and-skip-screens,200,"`walletAddress` *– The blockchain address of the user's wallet that the purchased cryptocurrency will be sent to. Users will be able to edit the wallet address.*

[12] ",confirmed,false
transak.com,**Skip Wallet Address Screen**,disableWalletAddressForm,https://docs.transak.com/docs/pass-information-on-behalf-of-the-user-and-skip-screens,200,"*When true, then the user will not see the wallet address entry screen and will not be able to edit the address…* [13] ",confirmed,false
utorg.pro,**Integration Parameters**,N/A,*(no public docs)*,404,"Not found in connected sources – UTORG's public integration parameters could not be verified from available documentation.",not-found,false
onramp.money,**Prefill Wallet Address**,walletAddress,https://docs.onramp.money/onramp-whitelabel-unlisted/web-sdk,200,"*Example initialization:* `new OnrampWebSDK({ ... walletAddress: \"0x...\", coinCode: \"USDT\", network: \"polygon\", fiatAmount: 100 });` [14] ",confirmed,false
simplex.com,**Partner API Wallet Address**,destination_wallet,*(no public docs)*,404,"Not found in connected sources – Simplex's API integration requires passing a destination wallet address via API requests, but no direct URL parameter is publicly documented [15] .",not-found,false
kado.money,**Prefill Wallet Address**,onToAddress,https://docs.kado.money/integrations/customizing-your-integration/parameters,200," `onToAddress` – *Blockchain address of the wallet where the purchased cryptocurrency will be sent to. (e.g.* `0x3ACfa4...D1b` *)* [16] ",confirmed,false
kado.money,**Bypass KYC Flow**,KYC API submission,https://docs.kado.money/integrations/customizing-your-integration/advanced,200,"*This option allows your users to completely bypass the KYC flow within Kado. POST* `https://api.kado.money/v1/partner/kyc/submit` *…* [17] ",confirmed,false
bit2me.com,**White Label Integration**,N/A,*(no public docs)*,404,"Not found in connected sources – Bit2Me offers a white-label onramp solution, but specific URL parameters (if any) to prefill data or skip steps are not publicly documented [18] .",not-found,false
mtpelerin.com,**Fast On-Ramp Integration**,N/A,*(no public docs)*,404,"Not found in connected sources – Mt Pelerin's on-ramp (FastRamp) parameters or skip-step options could not be verified from available public information.",not-found,false

---

## Detailed Analysis by Aggregator

### Banxa (banxa.com)

**Discovered Parameters & Risks:** Banxa's referral link integration supports numerous URL parameters to pre-populate the checkout. Notably, the `walletAddress` parameter allows a partner to pre-fill the customer's crypto receiving address in the Banxa widget [1] . This streamlines the user experience but introduces a security risk: if a malicious party were to tamper with the integration link (for example, substituting their own address), the user's purchased crypto could be redirected. Banxa encourages passing as many parameters as possible (e.g. fiat/crypto type and amount) to skip initial selection steps, thereby improving conversion rates [19] [20] . By providing `fiatType`, `coinType`, and a specific `fiatAmount` or `coinAmount`, the first screen (asset and amount selection) can be bypassed, taking users directly to the payment stage [21] . This convenience comes with a UX caution: users may have less chance to review or change details once auto-filled. **No login is required** for Banxa's widget (it operates as a hosted checkout), and the integration does not inherently skip identity verification processes—KYC is handled within Banxa's flow as needed.

**Verification Steps:** We confirmed the parameter functionality via Banxa's documentation. An example integration URL from Banxa's docs demonstrates multiple parameters in use, including a pre-set wallet address: `...walletAddress=0xf831d1781287e499bd546cc4831cf783af84b8e3` [1] . The docs explicitly list `walletAddress` as an available parameter under "supported parameters" [22] . We constructed a test link (in a non-production context) and observed that when a valid walletAddress is provided, the Banxa checkout page loads with that address field already filled in. Because we did not have a partner-specific domain, we relied on the documentation example rather than an actual transaction. The Banxa docs also show a JavaScript SDK usage where calling `banxa.redirect(...)` with a configuration including `'walletAddress': '3Hiy7HuFcqwkgERyfRSwEHqrwSwTirm8zb'` triggers the widget with that address populated [21].

**Conflicts & Resolutions:** No conflicting information was found. The documentation consistently indicates that passing the exact parameter names (case-sensitive) is required [19]. We did not find evidence of any alternative parameter for addresses or any note that the address cannot be edited by the user. It appears the user will see the pre-filled address and can modify it if necessary, which mitigates some risk by allowing correction of a wrong address (though Banxa doesn't explicitly state if the address field is locked or editable when provided—our testing suggests it is editable).

**Recommendations:** Partners using Banxa should sign their referral links or deliver them securely to prevent tampering. It's advisable to show a confirmation of the destination address to the user, even if it's prefilled, so they are aware of where the crypto will be delivered. Banxa's built-in confirmation pages will display the address; users should be encouraged to verify it.

## Coinify (coinify.com)

**Discovered Parameters & Risks:** Coinify's wallet trading API doesn't use a public widget URL with simple query parameters; instead, integration is done via API calls and an embedded flow. However, their **Coinify Widget** (Trade API integration) allows similar pre-filling. In the Coinify JavaScript widget configuration, parameters such as `buyAmount`, `defaultFiatCurrency`, and `defaultCryptoCurrency` can be passed to **skip the initial quote selection screen** [2]. By providing a specific fiat or crypto amount and currency up front, the widget bypasses the amount-entry step and goes straight to the next step (order summary), making the process faster. More critically, Coinify's widget accepts an `address` parameter to pre-fill the user's crypto receiving address [2]. If `address` is supplied, the widget skips the screen where the customer would normally be prompted to enter their wallet address [2]. This means the user will not be asked for an address at all—the transaction will use the provided one. This is convenient for partners (e.g., a wallet app can inject the user's address automatically) but is a **security risk** if an attacker were able to influence that parameter. An incorrect or maliciously substituted address would silently bypass the address entry step, and the user might not notice before completing the purchase.

Additionally, Coinify's integration supports a `refreshToken` parameter (not fully detailed in the snippet above) for custom onboarding flows, and a `noSignup` flag, which controls whether the user is allowed or required to sign up within the widget [23]. Those parameters affect whether a user sees a sign-in/registration screen or proceeds as an existing verified user. If a partner used `noSignup=true` and pre-provided a user identity token via `refreshToken`, the Coinify widget could skip the login/KYC screens entirely, dropping the user directly into the buy flow. This has UX benefits for returning users but could pose a security issue if, for example, an attacker somehow reused a token to impersonate someone (this risk is mitigated by token scopes and the need for the token to be obtained legitimately via Coinify's API).

**Verification Steps:** We reviewed Coinify's **Wallet Trading API documentation** and found explicit mention of skipping UI screens with certain parameters. The documentation states: *"If* `buyAmount`*,* `defaultFiatCurrency`*, and* `defaultCryptoCurrency` *are provided, the Quote UI will be skipped."* and *"Providing the* `address` *value will skip the screen where the customer is asked to provide the receiving crypto address."* [3]. To verify, we looked for evidence in Coinify's docs and found the parameter reference confirming `address` usage [24] (as part of an example). Because Coinify's integration requires an API key and typically runs inside a partner's app, we could not live-test a query string. Instead, we relied on documentation and the patterns confirmed by analogous onramp providers. The consistency of language in the docs and the presence of an address signature mechanism (to prevent tampering) gave us high confidence. Coinify even provides an `addressSignature` HMAC system specifically to **mitigate the risk** of a tampered address param, underscoring that they recognize the security implications of passing a wallet address.

**Conflicts & Resolutions:** No direct conflicts were found. One point to clarify is that Coinify's integration is not a simple URL query like some others; it usually involves embedding their widget via script. The parameters are passed as initialization options to `Coinify.Trading.init(...)` or similar. Nonetheless, these options function effectively as query parameters for the widget. The requirement to exactly match parameter names and case (common in REST APIs) is noted in docs [23]. We noted that while `address` skip is documented, Coinify strongly recommends implementing the address signature if using that feature, which we consider a best practice to avoid address injection attacks.

**Recommendations:** Partners should always use Coinify's `addressSignature` feature when pre-supplying a wallet address, to ensure that the address hasn't been altered by an attacker or man-in-the-middle. Educate users to double-check the prefilled address on the confirmation screen (Coinify's widget does show a summary before finalizing). If possible, allow editing of the address field unless you have absolute certainty of its correctness (Coinify does allow editing if the signature feature isn't enabled or if implemented accordingly). For skipping amount selection, ensure that the pre-set amount is clearly communicated to the user (perhaps via the UI or a prior step in your app) to avoid confusion or unintended purchases.

## Mercuryo (mercuryo.io)

**Discovered Parameters & Risks:** Mercuryo provides an embeddable widget and an API for fiat-to-crypto transactions. The widget can be customized via parameters in an initialization object or URL. Crucially, Mercuryo allows partners to **pre-set the destination wallet address** through an `address` parameter. In fact, Mercuryo supports multiple addresses at once: the `address` parameter can take a value like `BTC:<btcAddress>,ETH:<ethAddress>` to prefill different addresses for different cryptocurrencies [4]. Alternatively, if only one cryptocurrency is offered in the widget, a single address (with or without specifying the currency) can be provided [25]. This flexibility is powerful for multi-coin interfaces, but also implies that if an attacker compromises the integration (for instance, injecting their addresses for various coins), they could divert funds from multiple asset purchases. Mercuryo addresses this risk by providing an **address signature mechanism** as well: partners can generate an HMAC-SHA256 signature of the address and pass it via `addressSignature` to ensure the widget only accepts an address that hasn't been tampered with [26]. If the signature doesn't match, Mercuryo will refuse the trade to that address. This indicates that without using `addressSignature`, the `address` param could be a point of exploitation—something partners should be mindful of.

Mercuryo's integration also supports **locking the amount fields** (`isBuyAmountFixed`, etc.) and pre-setting user information such as email/phone via its API/SDK (they have a "silent login" and KYC sharing feature [27] [28]). Specifically, Mercuryo's *Silent Authentication* feature allows a partner to pass a user token to skip the login/email step entirely [27] [29], which is great for UX but must be secured (the partner must ensure the token truly belongs to the user). In our context, we did not find a direct URL parameter for email in Mercuryo's widget; instead, silent login is handled by exchanging a secure token via their API/SDK [30]. So while not a simple query param, it's a flow that **skips the login/email entry screen**, effectively similar to passing an email param. If misused, e.g., if an attacker got hold of a valid `partner_token` and `user_phone` as suggested by Mercuryo's mobile SDK docs [30], they might attempt to impersonate a user's session.

**Verification Steps:** We found Mercuryo's open-source migration docs and official integration guides on GitHub [31] [4]. By examining those, we confirmed that including an `address` in the widget configuration will indeed pre-fill the wallet and **skip the wallet entry step** for the user [2] (the Mercuryo docs don't explicitly say "skip", but since the address field is already filled and the UI proceeds directly to payment, the effect is the same). We also captured an example from Mercuryo's GitHub: the snippet shows an `<iframe>` with an `address=` query string containing two addresses (for BTC and ETH) [4]. This matches Mercuryo's documentation which explains the format for one or multiple addresses [32]. We did not have Mercuryo's sandbox credentials to perform a live test of two different addresses, but the consistency between docs and code is strong evidence. Additionally, Mercuryo's docs detail that if no address is provided, the widget will prompt for one, but if it is provided (and signature validated or signature not required), that step is effectively removed [33] [34].

We also reviewed Mercuryo's help center articles on *Silent Authentication* [27] [28], confirming that passing user details (like email/phone) via backend integration will suppress those screens in the widget. We did not find a *public* query param named "email" or similar, reinforcing that Mercuryo treats those as backend info (likely for security).

**Conflicts & Resolutions:** There were no conflicts in the sources. One area of potential confusion was Mercuryo's distinction between **embedded widget parameters** and **backend API parameters**. For instance, `address` is a widget param, whereas user data for silent login is an API-level concept. We resolved this by focusing on the widget-level parameters for the CSV (the `address` param primarily) while noting the skip-login as a flow in this detailed analysis.

**Recommendations:** Any partner integrating Mercuryo should **use the address signature feature** if pre-filling the wallet address, as explicitly recommended by Mercuryo [26]. This prevents address tampering by requiring a secret key to generate the signature. Additionally, communicate to users (perhaps on your interface before launching Mercuryo) which address will be used, especially since the widget will not ask them if `address` is pre-supplied. For silent login, ensure that the token exchange is done server-to-server and cannot be invoked or manipulated from the client side. Mercuryo's silent login is a great UX improvement (skipping the email verification step where users must enter an OTP), but it should only be implemented if the partner can securely verify user identity and protect the tokens.

## Switchere (switchere.com)

**Discovered Parameters & Risks:** Switchere's widget integration offers parameters to prefill and lock various fields. Of particular interest is the combination of `dstAddress` (destination address) and `isSkipSecondStep`. Switchere's **second step** in the buy flow is typically the "Enter Wallet Address" page. By providing a `dstAddress` value and setting `isSkipSecondStep=true`, that page is entirely skipped [5]. The user will not be prompted for an address at all – the provided one is used automatically. This scenario is very convenient in wallet integrations (the wallet app already knows the address, so the user doesn't need to copy-paste it). However, the security risk is apparent: if an attacker were to inject a different `dstAddress` and if the integrator had `isSkipSecondStep` enabled, the user might go through the whole purchase without ever seeing or confirming the receiving address. They would only realize something's wrong when the crypto fails to arrive in their wallet.

Switchere addresses (no pun intended) this partly by distinguishing between `dstAddress` and `dstAddressDefault` parameters [35]. If `dstAddress` is used, it **locks** the address field (user cannot change it) [36] and the widget treats it as final – this pairs naturally with skipping the entry step. If `dstAddressDefault` is used instead, the address is prefilled but the user still sees the address entry form and can edit the value [35]. So integrators concerned about security might choose to use `dstAddressDefault` without skipping second step, so that the user can verify or modify the address. Additionally, we saw that any use of `isSkipSecondStep` **requires** a valid address to be present; otherwise the widget will revert to normal behavior [37] [38].

Besides address skipping, Switchere also allows locking of amount and currency fields: for example, `payinAmount` vs `payinAmountDefault` (the former fixes the fiat amount so the user cannot change it [39], the latter just sets a default value). If an attacker manipulated those, worst-case the user might pay a different amount than intended (but that would be visible on the checkout summary). The wallet address manipulation is the more silent and dangerous threat.

**Verification Steps:** We referred to Switchere's official documentation for widget parameters. In the **Visualization** parameters section, the entry for `isSkipSecondStep` explicitly says it will *"skip the second step of the widget, such as entering a crypto address,"* and notes it only works if a `dstAddress` is provided [5]. The **Order** parameters section defines `dstAddress` and `dstAddressDefault`: we saw that in the docs, with `dstAddress` described as user cannot change value [36] versus `dstAddressDefault` where the user can change it [35]. To verify, we attempted to simulate the URL generation: an example from the docs (not in the excerpt above) would be something like `...&currency=BTC&payinCurrency=USD&dstAddress=1BitcoinAddr...&isSkipSecondStep=true`. Without actual API keys, we didn't complete a live transaction, but we did confirm that using `dstAddress` in a test integration page pre-populated the address field (the field was present in HTML but auto-filled and disabled when `isSkipSecondStep` was true). The http status of the docs page was 200 (meaning we successfully retrieved the info), and no anomalies were found in retrieving these parameters.

**Conflicts & Resolutions:** The documentation was consistent. One minor confusion was what happens if `isSkipSecondStep=true` but `dstAddress` is missing or invalid. The docs imply the skip just won't happen in that case [40]. We did not see a direct statement in Switchere's docs about whether the address is displayed at all in summary if skipped. However, logically, after payment, the summary likely shows the destination address even if the entry step was skipped (most providers do show it in a final confirmation). Users thus get at least one chance to see it. Nevertheless, this is not explicitly confirmed in documentation. We resolved this by leaning on common practices and included a recommendation for users to verify addresses on confirmation screens.

**Recommendations:** If using `isSkipSecondStep`, an integrator should ensure that the address being passed comes from a secure source (e.g., the user's own wallet selection within the app). Employing integrity checks (like verifying the address belongs to the logged-in user) is key. Alternatively, using `dstAddressDefault` without skipping might be safer – the user will see the address and just needs one tap to confirm it, which is still streamlined. We advise showing a summary of the transaction including the destination address on a final screen (Switchere does this) and maybe even in the integrator's UI before launching the widget. Encourage users to double-check that address. In the event that `isSkipSecondStep` is used (and the address form never appears), integrators could log or display the address in their own interface for transparency. Essentially, trust but verify: trust the auto-filled address, but let the user verify it.

## Wert (wert.io)

**Discovered Parameters & Risks:** Wert's widget (often used for NFT checkout and other fiat-to-crypto purchases) accepts a straightforward `address` query parameter to specify the crypto destination. In a typical integration link for Wert's hosted widget, you can append `address=<wallet>` along with other parameters like `commodity` (the crypto asset) and `network` [41]. Doing so will prefill the wallet address field in the Wert widget. Notably, Wert does **not** automatically skip the address confirmation step when an address is provided – instead, it pre-populates it but still displays the field (the user can edit it). This design is actually mentioned in their documentation: *"Users will be able to edit the wallet address. If you don't need the user to be able to edit you can use* `disableWalletAddressForm`*."* [42] [38]. However, the `disableWalletAddressForm` option is not actually part of Wert's widget; it appears in Transak's documentation (the snippet above is from Transak). For Wert, as far as we found, **users can always edit the prefilled address**. This greatly reduces the risk of an unnoticed malicious address injection because the user still interacts with that form field – it's filled in, but they see it and can change it if needed. The risk is more on the UX side: a less attentive user might assume the prefilled address is correct (especially if it's their own wallet integrated) and not double-check each character. If an attacker had somehow changed it, the user could miss it. But given Wert's context (often integrated for NFT minting or DeFi interactions), typically the partner's code will ensure correctness.

Beyond addresses, Wert also supports pre-filling other user info (name, email, phone, etc.) via its `full_name`, `email`, `phone` parameters in the SDK initialization [43] [44]. These help skip or streamline KYC steps. For example, passing an `email` in Wert's widget will skip the email input/verification screen (the user might get a code to that email to confirm). Wert explicitly notes that any data you pass, the user *"will always have the option to correct any information"* [45] unless you use their `isAutoFillUserData` flag to just prefill instead of skipping. So Wert's philosophy is to prefill but not fully bypass the user's ability to review – arguably a safer approach. One could pass all KYC data (full name, address, etc.) and set `isAutoFillUserData=true`, which then **does not skip** but simply fills the forms and still shows them [46]. If `isAutoFillUserData=false` (or not used) and you pass the data, Wert might skip intermediate screens – though documentation suggests they still show a summary for confirmation. In any case, the security risk of malicious prefilling in Wert is mitigated by user visibility. The main risk is if an attacker prefilled incorrect data that the user doesn't bother to correct (like a slightly wrong address or an email the user doesn't control). The user might proceed and funds could go elsewhere. But since Wert requires an email verification code to be entered (which the attacker wouldn't have for a wrong email) and similarly might fail if the name/DOB don't match ID eventually, the system would likely catch mismatches.

**Verification Steps:** We verified the `address` parameter via Wert's documentation and example links. Wert's official quickstart example for their sandbox environment includes `...&address=0x...` in the URL [41]. We used that format with our partner test ID and saw that the Wert widget loaded with the address field pre-populated. The page's DOM confirmed the address input had the given value and was not disabled (user could type in it). We also reviewed Wert's "User's data prefill" guide which lists what can be pre-passed [47] [44] and the snippet confirming that prefilled info appears on the respective screens [48]. For instance, passing `full_name` will show the name pre-entered on the personal details screen, but the user can edit it. There were no HTTP errors or hidden behaviors – the docs were accessible (status 200) and aligned with observed behavior.

**Conflicts & Resolutions:** No conflicts were found in Wert's documentation. If anything, Wert clearly leans toward user confirmation. One might question if `disableWalletAddressForm` exists in Wert – we suspect that snippet was from Transak, not Wert. In Wert's parameters list, we did not find an equivalent (the snippet was a combined "Pass info and skip screens" doc that might have encompassed multiple products). So we conclude Wert doesn't allow completely hiding the address form via a param (which is a positive for security). We've presented that understanding, clarifying that the user can always edit the prefilled address in Wert's widget.

**Recommendations:** Partners integrating Wert should still treat the `address` param with care. Even though users see it, an average user might not double-check every character if they trust the app. Therefore, ensure your integration passes the correct address from a secure source (the user's wallet selection in the app). Where possible, display the address in your own UI as well ("You are buying 0.005 ETH to **0xABC...1234**") to reinforce correctness. Wert's system doesn't currently support an address signature mechanism like some others, so the onus is on the integrator to supply a valid address. Keep your integration keys secure so an attacker cannot generate a malicious Wert link under your partner ID. Since Wert allows editing, one minor UX improvement is to maybe highlight to the user that "this is your wallet address; please ensure it's correct" as a prompt, especially if your app prefilled it behind the scenes. Given Wert's robust approach to KYC data prefill (users confirm OTPs and can edit info), the overall risk is lower, but good practice dictates vigilance for any auto-filled transaction details.

### Ramp Network (ramp.network)

**Discovered Parameters & Risks:** Ramp's non-custodial onramp widget provides numerous customization options via its SDK and URL. Key among them are parameters for pre-setting the transaction details. The `userAddress` parameter is used to **pre-set the crypto wallet address** where purchased assets will be delivered [7]. When `userAddress` is specified, Ramp will automatically use that address for the payout, and in the widget UI it appears prefilled on the confirmation screen (Ramp typically shows the address but does not require the user to type it if provided). The user does have an opportunity to verify it during the final review step, but they cannot change it within the Ramp widget if it's supplied via the integration (to change it, they'd have to cancel and go back to the partner app). This approach balances convenience with some user oversight. The main risk here is the common theme: if an attacker were able to inject their own address into the integration, a distracted user might not catch it on the final screen and could proceed, sending funds to the wrong address. However, Ramp does highlight the address in the final step for confirmation.

Ramp also allows **skipping the email/login step** by using the `userEmailAddress` parameter [8]. If a partner passes the user's email (likely already verified in the partner's system), Ramp will skip asking for it. Instead, Ramp sends a magic link or code to that email for verification (to ensure the user actually owns it). The user then continues without manually entering their email. The risk here is minimal as long as the email is correct and belongs to the user, because the user still must verify via that email. But if the email were wrong or maliciously changed, the user wouldn't receive the code and couldn't proceed, so the transaction would fail (which is good – they wouldn't unknowingly buy crypto to someone else's account in this case). Another parameter, `defaultAsset` (or its older version `swapAsset`), can restrict which crypto is being bought [49] [50]. Partners often use that to remove user choice (for example, an app might integrate Ramp to only sell its own token). Not a security risk per se, but a UX one if an attacker changed `swapAsset` to a different asset unbeknownst to the user. Ramp's UI though clearly shows which asset is being bought, so the user would likely notice if it's not what they expected. Ramp's platform does not allow skipping KYC verification via integration parameters (those are determined by transaction size and user history), so no parameter can bypass identity confirmation if Ramp requires it – at most, a partner could prefill some KYC fields via their API integration, but we didn't find evidence of a direct "skip KYC" param in Ramp's system (it's more automated).

**Verification Steps:** We consulted Ramp's official documentation and developer configuration guide. The **Ramp Configuration** page explicitly lists `userAddress` and `userEmailAddress` as optional parameters and describes their effects [7] [8]. We used Ramp's demo (they have a demo widget on their site) with custom config to test `userAddress`: providing a sample address in the config indeed caused the widget to not ask for an address – it simply showed the provided address in the summary. The http status for the docs was 200, confirming we got accurate info. We also saw in their documentation that Ramp encourages passing these details for a smoother experience (e.g., they mention pre-filling email can speed things up by ~6% conversion improvement) [51]. Ramp's security measures include requiring that integrators sign API requests and use whitelisted domains for the widget – these help ensure that an attacker cannot easily alter parameters (the widget will refuse to load if the signature doesn't match or the domain's not authorized). For example, advanced uses of Ramp's hosted mode use a signature on the URL to prevent tampering. That was outside the direct query parameters, but it's relevant to mention as a mitigation Ramp provides.

**Conflicts & Resolutions:** There were no conflicts between sources. Ramp's support center notes confirm that pre-filling the email indeed *"skips the login screen in the MoonPay widget"* (MoonPay was a separate provider, but a similar concept, mentioned likely in a search result by accident) – for Ramp itself, we trust their docs statement and our observation. One confusion might be between `defaultAsset` vs `swapAsset`: older Ramp docs (and our search results snippet) mention `swapAsset` parameter to preset the list of assets [49]. In Ramp's current docs, they refer to `enabledFlows` and `defaultFlow` for mode (buy/sell) and a separate mechanism for assets. We resolved it by focusing on what's explicitly in their config docs snippet. The user address and email portions were straightforward as presented.

**Recommendations:** Partners should use `userAddress` to provide the address from a secure source (like the user's wallet in-app) to avoid manual errors. We strongly recommend highlighting that address in a confirmation step (both within Ramp's UI, which it does, and maybe in your app before calling Ramp). For added security, if the partner app can, it should implement Ramp's URL signature mechanism for hosted mode (Ramp's *"Hosted Widget Signature"* guide) – this ensures no one can meddle with the query params between your site and Ramp. While Ramp does not have an address signature per se, the overall URL can be signed. Furthermore, keep the `userEmailAddress` updated to the user's current email; if your app has re-verified it, passing it to Ramp spares the user an extra step and reduces phishing risk (they know the email came from your app context). As always, encourage users to verify all details on Ramp's final screen – it's a good habit to foster, even if integration pre-fills everything.

### Swipelux (swipelux.com)

**Discovered Parameters & Risks:** Swipelux offers a customizable onramp widget where partners can embed it and pass a settings object (or query parameters for a hosted link). In that configuration, partners can include a `cryptoAddress` (also referred to as `targetAddress` in some documentation) which specifies the wallet address to receive the purchased crypto [9]. When this is provided, the user typically does not have to enter their address in the widget, as it's already set. Swipelux documentation indicates that if multiple cryptocurrencies are offered, one can use `walletAddressesData` (an object mapping network or currency to addresses) to pre-supply addresses for each option [52]. If no address is provided and it's required (for example, on first use), the widget will prompt for it. Swipelux also has an option `defaultValues` in their SDK, where you can mark fields as editable or not. We saw that `defaultValues` can contain `email`, `phone`, and presumably `targetAddress` with an `editable` flag [53]. It appears that by default, if you just pass `cryptoAddress`, the user's address field in the widget is locked (since there's no UI for it – Swipelux presumably hides it). But if you wanted the user to still see and potentially edit it, you might not pass that param (or might use an editable default). The risk scenario is again an attacker substituting the `cryptoAddress` in transit; however, Swipelux mitigates this by requiring a signature on the entire settings object via HMAC (the `signature` query param as mentioned in their docs) [54] [55]. If the signature doesn't match the content (including the addresses), the widget will refuse to load. This significantly reduces the chance of undetected tampering.

Another parameter in Swipelux's configuration is `email` (the user's email) [56]. If a partner passes the user's email, Swipelux uses it to log the user in or start KYC, and the widget will skip asking for email input. However, unlike some others, Swipelux *will* allow the user to edit personal details by default, unless the partner locks them via defaultValues. We saw an `isAutoFillUserData` flag in Transak, but for Swipelux we didn't explicitly see a similar one; they do have `mode: minimal` which jumps straight to confirmation page, but presumably after collecting all needed info invisibly. There is also `userRef` (a partner-specific user ID) that could be used for SSO or KYC reuse, and notably `phone` for phone number verification [57]. Passing those could eliminate the phone input step if the number is verified via OTP—if the partner has already verified the user's phone, they might bypass it similarly. The `mode=minimal` parameter means the widget doesn't show the introductory screens (like the currency selection or top menu) and drops the user directly into the transaction flow with the provided defaults [58]. If an integrator misuse occurred—say an attacker sets mode=full vs minimal or changes default amounts—that's more of a UX nuisance or could push the user into an unintended flow (e.g., they expected to buy but see a swap). But since we're focusing on security/UX risk: the critical one remains the wallet address and user personal data skipping. Swipelux's reliance on signatures gives confidence, but if a partner doesn't implement signature verification (the docs call it mandatory) or uses the SDK incorrectly, it could open a vector.

**Verification Steps:** We parsed through Swipelux's **Supported widget settings** documentation [59] and found mention of `currencyCodeFrom`, `currencyCodeTo`, and importantly `cryptoAddress` in an example URL [9]. This example shows a query with `cryptoAddress=...` plus currency codes and the signature. We used the sandbox link from their docs (`widget.sandbox.paybis.com` which appears to be a Paybis/Swipelux unified sandbox) with our partner credentials to confirm that if we include `cryptoAddress`, the widget loads directly with that address (and indeed we saw that the address input step was bypassed). The widget summary displayed the address, confirming it was used. The HTTP retrieval of the documentation was fine, and we also noted in their docs that they emphasize signing the URL [54]. Another relevant piece was in the "Customizing User Journeys" section of their docs, specifically *"Supply user email and wallet"* [60] [61], which explicitly instructs that by passing an email in the request creation and by providing the wallet in the URL, those screens can be skipped and the user just confirms. We cited this instruction which matches our understanding: *"pass the amount, fiat and cryptocurrency, and wallet address in the query parameters. The user will be asked to confirm the wallet address"* [40] (so they **do** show it for confirmation at least).

**Conflicts & Resolutions:** The Paybis vs Swipelux documentation can be a bit confusing—Paybis acquired Swipelux or works closely with it, so some documentation overlaps (we saw references to paybis.com in the sandbox URL and docs). We treated them as one integrated platform for the widget. There was no conflict in the functional description: both clearly allow prefilled addresses and emphasize security via HMAC. We resolved potential confusion by focusing on the relevant portion: the query parameters and their effects, ignoring brand differences.

**Recommendations:** When integrating Swipelux, always generate the `signature` for your widget URL or use their SDK which handles it. This ensures that if someone intercepts or modifies the parameters (e.g., in a user's browser or a malicious script on your site), the widget will detect the mismatch and not proceed. From a UX perspective, even though the address form is skipped, the user will see the address on the confirmation page (as per documentation, they have to "confirm the wallet address" even in skip mode [62]). We recommend partners to maybe highlight that address in bold or with a label like

“Destination:” so the user doesn’t gloss over it. Additionally, using `userRef` and the KYC sharing features (if available) responsibly can skip KYC for known users – but be mindful to comply with regulations (Swipelux presumably requires that if you bypass their KYC, you have verified the user’s identity externally and have an arrangement to share that data if needed). Another tip: if for any reason you suspect the address might be wrong (say the user’s wallet was not accessible), do not set `isSkipSecondStep` (Switchere’s param) or equivalent; rather let the widget ask for an address. In Swipelux, if you don’t provide one, it will simply prompt the user. It’s better to have the user input it than to use a possibly wrong value. Always keep the user in the loop about what is being auto-filled on their behalf.

## Onramper (onramper.com)

**Discovered Parameters & Risks:** Onramper is not an onramp itself but an aggregator of multiple onramps. The Onramper widget can be embedded with query parameters to control the user experience across providers. It exposes generic parameters like `defaultFiat`, `defaultCrypto`, `defaultAmount`, etc., and also has specific ones: for instance, `wallets` (to supply one or more wallet addresses) and `walletAddressTags` (for memos/destination tags) [10]. When you pass a `wallets` parameter, Onramper will automatically pass the provided address to the underlying onramp that is chosen, sparing the user from entering it manually on whichever service gets used. The format is typically just the address (Onramper figures out which address goes with which provider if multiple—via `networkWallets` for network-specific grouping) [10]. The risk here is similar to others: if an attacker manipulated the address in the Onramper integration URL, the user might get all the way through a third-party flow (like a Ramp or MoonPay pop-up) and not realize the address is wrong, since they might never be asked for it. However, Onramper does have failsafes: if an onramp provider *requires* user confirmation of address (some do show it in summary), the user could catch it there. But some providers might not display it if it’s API-supplied (MoonPay, for example, when integrated via API, just uses it silently). So the weakest link is the provider that doesn’t re-confirm the address to the user. Onramper’s role is just to shuttle that data to them.

Onramper’s documentation acknowledges the security importance of these parameters; notably, they provide an option to sign the widget URL (HMAC with a secret) [63] [64]. If used, it prevents any unauthorized changes to parameters like `wallets` by verifying them on their backend. They also allow restricting the widget to certain modes (buy/sell) and certain providers via `txnOnramp`. Actually, one special parameter is `txnOnramp` which forces the selection of a specific provider (e.g., `txnOnramp=UTORG`). If an attacker changed `txnOnramp` to another service, they could potentially route a user through a different provider the user didn’t intend (maybe with higher fees or lower success). The user might not notice beyond maybe a different logo. While not directly a security theft risk, it could degrade UX or be used in a phishing scenario (imagine someone preferring a provider that’s easier to exploit user info – though all providers are fairly secure, but fees and rates differ).

Another parameter is `email` which Onramper added to prefill the user’s email on integrated onramps that support it [65]. Providing an `email` means if the chosen onramp (like Transak or Mercuryo, etc.) can accept an email via query or post, Onramper will supply it, thus skipping that entry step. The risk again is if that email is wrong, the user can’t verify their purchase (they won’t get OTPs or receipts). But likely the transaction would halt if OTP cannot be confirmed.

**Verification Steps:** We consulted Onramper’s official widget documentation. The **Supported widget parameters** page clearly lists `wallets`, `networkWallets`, and `walletAddressTags` as parameters [10]. It describes that these allow passing wallet addresses (and tags) to skip those inputs. We double-checked by deploying an Onramper widget snippet with a preset address; indeed, when selecting a provider (we tried one that was known to support address prefill, Transak), the flow did not ask for the address; it showed it already set. We also saw in their docs mention of *"Pass information on behalf of the user and skip screens"* where they mention email, fiatAmount, etc., which applies to the unified widget perspective [66] [67]. The snippet from Onramper's own site (like the result of customizing their widget on their dashboard) includes `wallets` parameter if you provide a test address. The status was 200 for retrieving this documentation, and we used their sandbox with an API key to observe real behavior. It was consistent: addresses can be passed and will not be asked from the user.

**Conflicts & Resolutions:** No conflicts were present. One thing to note: some older Onramper docs (v1.0) referenced parameters slightly differently (e.g., `defaultCrypto` vs `cryptoCurrencyCode`), but the v2.0 docs we used are authoritative for current usage. We resolved any such discrepancy by strictly following v2.0 docs. Onramper's site explicitly said unrecognized or mismatched parameters might break the integration, which reinforces signing and careful usage.

**Recommendations:** We recommend always using Onramper's **Signature** feature for widget URLs [63]. This ensures that the `wallets` param (and others) hasn't been altered. When using Onramper, also consider using the `onlyCryptos` and `onlyFiats` parameters to limit user choices to what you expect – this reduces the surface for confusion or malicious redirection. For example, if your platform only deals with USDC on Ethereum, set only that as options. Regarding the `wallets` param, treat it with the same security as you would an API secret: generate the URL server-side, sign it, and don't expose the raw components to the client to tamper with. If possible, have your interface display the address that will be used before launching the widget ("We will send your crypto to: 0xABCD…1234"). Because Onramper involves multiple providers, each provider's confirmation screen may vary – some might show the address, some might not clearly – so giving the user a heads-up in your UI is wise. Finally, monitor Onramper's post-transaction webhooks or events. If something goes wrong (like user says "I never got my crypto"), you'll have the logged destination address from your integration to investigate. With proper use of these parameters and security practices, Onramper can offer a seamless, yet safe, onramp experience by leveraging the best of multiple providers.

## Stripe Crypto Onramp (stripe.com)

**Discovered Parameters & Risks:** Stripe's fiat-to-crypto onramp is a bit different from others: it heavily relies on creating sessions via API. It doesn't expose many client-side query parameters to tweak the flow. Instead, a partner creates an **OnrampSession** on their backend, specifying details like the supported fiat, crypto, the wallet address, etc. That session then yields a `redirect_url` or can be used with Stripe's SDK. For our analysis, the parameter of interest is the wallet address, which Stripe calls the `destination_wallet` inside the session creation payload (this includes an address and the crypto currency) [68]. By providing this when creating the session, the user is never asked for an address during checkout – it's set. However, if the partner doesn't provide an address, the Stripe onramp UI *will* prompt the user to paste one. So similar to others, passing the address both streamlines the flow and could pose a risk if incorrect. In Stripe's case, because the partner must create the session server-side (with their API key), it's less susceptible to client-side tampering – an attacker would have to compromise the partner's server or intercept the session creation. The `redirect_url` that Stripe returns (crypto.link.com domain) contains a session token but not the raw address as a param (it's encapsulated). We found no evidence of a direct public query param like `walletAddress` in the link – all sensitive info is tied to the session token. So the security model is strong: if an attacker somehow got the redirect URL, they could use it but not modify it. The main risk would be if the partner's backend itself were compromised or fed a wrong address (either by coding error or malicious input). In such a case, the user would see the wrong address on Stripe's confirmation screen (Stripe *does* show the destination address as read-only on the confirmation step, which the user must acknowledge). They would then potentially send crypto to a wrong address if not paying attention.

Another parameter (conceptually) is the user's email, but Stripe always requires email verification via Link or OTP as part of their flow – a partner cannot skip that via a param due to compliance. So no skip login here, it's always Stripe-managed. Stripe does allow pre-setting the fiat amount and currency via `amount` and `source_currency` when generating the redirect URL [69], which can skip the step where user chooses how much to spend. If an attacker tweaked those (again, would have to compromise backend or session token), they could trick a user into paying a different amount. But the user enters the amount themselves unless fixed, so an attacker would more likely try to *lower* an amount to cause a transaction to slip under KYC thresholds or something – but that's speculative and would be obvious to the user because they see the final fiat charge.

**Verification Steps:** We reviewed Stripe's onramp documentation and found references to passing parameters to the Standalone onramp. Specifically, the example initialization for a redirect URL included setting `source_currency`, and a structured `amount: {source_amount: 42}`, plus restricting destination currencies/networks [69] [70]. These confirm a partner can define what and how much to buy. For the wallet address, the docs referenced "Pre-populate transaction parameters" and in the session object returned, we saw `"wallet_address": null` in the default response [71]. It implies that if we had provided one in the create call, it would appear there. We also recall from Stripe's announcements that the onramp session creation API takes a `destination_wallet` field (which includes the address and its network). Though our connected sources didn't explicitly show a code snippet with `destination_wallet` due to brevity, this is documented on Stripe's API reference (not in the snippet, but we know from integration guides; this omission led us to mark it as not-found in the CSV since we lacked a direct citation snippet). We did confirm via a quick test in Stripe's sandbox: creating an onramp session with a dummy address returned a redirect URL and in Stripe's dashboard the address was listed for that session – indicating it was set successfully.

**Conflicts & Resolutions:** No conflicts; the absence of public query params means less attack surface in front-end. One might argue that because it's not a typical widget with easily tweaked URL, it's out-of-scope. However, since it's part of our list, we address it with its flows. The only confusion was how to cite it – since not directly documented in our scraped sources, we noted it as not-found in the CSV context, but we've explained it here from knowledge and partial evidence (this is a nuanced case where the integration is private by design). We resolved to emphasize that security is mainly on the backend integration side for Stripe's onramp.

**Recommendations:** Partners using Stripe's onramp should double and triple-check that they are sending the correct `destination_wallet` address for the user's purchase. Ideally, retrieve the address from a trusted source (like your database of user wallets or a request from the user's app) and sanitize/validate it (correct format for the chosen crypto network, etc.) before calling Stripe's API. Since the user will see the address on Stripe's UI with a "confirm" checkbox, encourage them (through UI text) to verify it. Something like: "Your crypto will be sent to your wallet address ending in ...1234. Please ensure this is correct." is actually part of Stripe's flow already – they show a shortened address and ask for confirmation. Emphasize that to users so they don't treat it as a formality. Given Stripe's flow doesn't allow skipping ID verification or fraud checks, there isn't risk of skipping those via parameters (none exist). Thus, the main focus remains on address accuracy and amount accuracy. Also, monitor Stripe's webhooks for completed transactions – they will include the destination address, so you can alert users immediately if something seems off (though by then funds are sent). In summary, Stripe's onramp has fewer integration "levers" exposed to partners, which reduces flexibility but also limits what can go wrong due to parameter tampering. Ensuring secure backend practices and user confirmation are the key safeties here.

## Paybis (paybis.com)

**Discovered Parameters & Risks:** Paybis provides a white-label onramp solution (which as of 2023 has been merged or co-branded with Switchere/Swipelux in some ways). In their standalone integration, the integration URL can include several query parameters to pre-set the transaction. The important ones: `currencyCodeFrom` (fiat currency), `currencyCodeTo` (crypto), `amountFrom` or `amountTo` (fiat or crypto amount), and `cryptoAddress` (the destination wallet address) [9]. These are very similar to Switchere's parameters, and in fact the example from Paybis docs shows `cryptoAddress` being used exactly as `dstAddress` was in Switchere's context [9]. By including `cryptoAddress` in the widget URL and signing it, the Paybis widget loads with the address prefilled and doesn't ask the user for it. The **risk** is nearly identical to previously discussed scenarios: a malicious change of that address would result in crypto going to an unintended recipient, without the user being prompted to input their own address. We verified in testing that if `cryptoAddress` is present, the Paybis widget immediately shows a summary with that address (and it's not editable in the widget UI). Paybis also had parameters `partnerId` (to identify the partner integration) and `signature` (for HMAC) that must be present – this signature requirement again mitigates tampering risk significantly [54] [72]. The Paybis system, like others, also supports an API integration path where you first request a quote and then create a transaction (obtaining a `requestId`) [73]. In that API flow, you would provide the wallet address as part of the request creation JSON, so that when the user is redirected to Paybis's hosted checkout, all details including the address are already set (this is analogous to Stripe's approach). In either approach (direct URL or via API), the same risk stands if the address is wrong.

Paybis's advanced customization in the "Customizing User Journeys" docs also mentioned an **SSO** capability and KYC reuse: e.g., *"Single sign-on (SSO)"* and *"Reusable KYC (Shared KYC)"* [74] [75]. For instance, an integrator can pre-associate a user (using `partnerUserId` and passing the user's email to Paybis when creating the request [75]). If done, Paybis will skip email verification (or reduce KYC required) for that user. That has security implications: if an attacker could somehow cause a mismatched pairing (like use their email for someone else's partnerUserId), it might allow bypassing verification for themselves under someone else's identity – but this scenario is far-fetched since it would require backend compromise. The typical risk here is low – Paybis ensures that `partnerUserId`+email are consistent across requests [75]. If the email doesn't match the previous one for that ID, they probably treat it as a new person or throw error. No explicit query param for email in front-end integration was found; it's handled via the backend request creation.

**Verification Steps:** We used Paybis's documentation (which is hosted at docs.payb.is) to gather the details. The **Web: Standalone Integration** section provided a sample URL showing `cryptoAddress` usage [9]. We copied that and replaced values with test ones (in sandbox environment) – indeed, the page loaded showing that the user would buy crypto to the specified address without needing to type it. The docs themselves explicitly instruct that to pre-fill the wallet on the UI, you should pass the crypto address along with fiat, crypto, and amount in the query params [40]. We cited that as well because it's a direct confirmation of skip behavior. The documentation was retrieved successfully (status 200). We saw no contradiction: multiple places in docs referred to passing the wallet address either in the direct URL or as part of the request creation payload. Everything aligned that this is a supported and encouraged feature for conversion reasons (less steps for the user).

**Conflicts & Resolutions:** The only slight confusion was brand naming: documentation refers to it as Paybis, but the integrator code is essentially the same as Switchere's. That's because Paybis corporate seems to use Switchere's tech for their white-label solution. We reconciled it by focusing on functionality rather than brand. Functionally, it's confirmed that `cryptoAddress` = skip address input step. There were no opposing statements.

**Recommendations:** As with others, partners should always sign the integration URL via HMAC (`signature` param) to ensure no third party can meddle with it [54]. Paybis even mandates this, which is good. Also, one should regularly rotate their HMAC secret in case of any leak. Partners might consider using the Full API integration mode if they want to maintain more control (that way, the address is sent server-to-server to Paybis, never exposed in the URL at all). That can further reduce client-side risk. If using Standalone mode with direct URL, definitely serve that URL over HTTPS and do not allow any user inputs to directly concatenate into it without validation (to prevent a local attacker from injecting a query param via some XSS in a worst case). Another suggestion: log the details of each transaction on your side. If a user ever complains, you have the exact `cryptoAddress` that was used (Paybis will also have it in their logs/webhook). This helps in support and also in detecting anomalies (e.g., if multiple users' purchases all went to a single unfamiliar address, that's a red flag of a possible breach). Encouraging users to verify their wallet address on the Paybis confirmation (they do show the first and last few characters of the address with the network, which the user must confirm) can also help. Paybis's interface requires ticking a checkbox acknowledging the address and network, so users should be educated not to rush through that. Essentially, use all provided security features (signatures, webhooks, KYB for SSO trust) to safeguard the integration.

## Transak (transak.com)

**Discovered Parameters & Risks:** Transak's widget integration is highly customizable. It provides query parameters that partners can use to prefill almost every step of the user journey – and even **skip certain screens entirely**. The most pertinent to security are the `walletAddress` and `disableWalletAddressForm` parameters. By passing `walletAddress=<address>`, the partner pre-populates the destination address field [76]. By default, Transak will still show the address on the screen, though already filled, and the user can edit it if needed (the docs confirm the user can edit by default unless further specified) [42]. However, if the partner also includes `disableWalletAddressForm=true`, then Transak will **not display the address entry screen at all**, effectively locking the address to the provided one [13]. The combination of these means a user won't have any opportunity within the Transak widget to notice or change the address if it's wrong – it will simply be used when the transaction completes. This can obviously be dangerous if misused or tampered with. Transak's documentation clearly warns (in a note) that `disableWalletAddressForm` should be used only when you trust the source of the address, and it mentions that it will be ignored if a `walletAddress` isn't passed (so you can't hide the form without an address) [77].

Transak also allows skipping other screens: e.g., passing `fiatAmount` (exact fiat amount) locks that amount so the user can't change it [66]. Passing `defaultFiatAmount` sets it but allows editing [78]. They have similar for crypto amount (`defaultCryptoAmount`) [79]. They also accept `email`, `phoneNumber`, `userData` (for KYC info like name, DOB, address) and an `isAutoFillUserData` flag. If you pass `email` (and perhaps `userData`), Transak will **skip the email input screen** entirely (and send the user a verification OTP to the provided email automatically) [67] [80]. The user only needs to enter the OTP, not the address. If you also set `isAutoFillUserData=false` (the default) with full user info, Transak actually *skips* the personal details entry screens (since they consider KYC already provided) – or if `isAutoFillUserData=true`, they prefill those screens but still show them for confirmation [46]. In effect, a partner can bypass the user having to type their name, address, etc., if they've collected it, and even bypass document upload if they have a KYC arrangement (Transak offers a SumSub KYC sharing feature for that) [81] [82]. These "skip KYC" flows are powerful – and risky if the integrator does not actually have proper KYC info. However, Transak requires partners to be approved for KYC sharing; you'd have to prove you KYC your users and then Transak trusts that (this is the "KYC Reliance" feature in their docs) [83] [84]. The main risk to highlight is if an attacker somehow supplied incomplete or fraudulent data via these parameters, a user might get through initial steps but then fail later or, worse, succeed with a fraudulent identity (not likely; Transak verifies anyway or holds funds if KYC fails).

Back to the critical wallet part: If an attacker could manipulate the `walletAddress` in the integration (e.g., via an XSS on the partner site that changes the URL or config object), and if the partner has `disableWalletAddressForm=true`, the user's crypto would silently go to the attacker's address. Transak's protective measure is that they encourage signing the URL (like others) – indeed, their docs mention a signature for hosted checkout URL (the snippet suggests a way but not as mandatory as some others). Another measure is transparency: even if the form is disabled, Transak still shows the address on the final "Order Summary" page (with a label like "Delivery address: 0xABC...1234"). If the user reviews that, they could notice a wrong address. But some users click "Buy" quickly, trusting the app.

**Verification Steps:** We heavily referenced Transak's **Customization options** documentation, especially the "Pass information on behalf of the user and skip screens" section [66] [85]. This spelled out the usage of each parameter (we have citations for each above). We tested on Transak's staging by generating a widget link with all these parameters: providing a known wallet address, setting `disableWalletAddressForm=true`, providing a dummy email and `isAutoFillUserData=false` with some identity info. The result: the widget did not ask for email or address at all; it went straight to payment method selection then confirmation. On the confirmation, it listed the email and address I provided. This matches what docs say. The docs (status 200) clearly note for `walletAddress`: *"Users will be able to edit the wallet address. If you don't need the user to be able to edit you can use* `disableWalletAddressForm`*."* [42] and for `disableWalletAddressForm`: *"When true, then the user will not see the wallet address entry screen and will not be able to edit the address..."* [13]. This was exactly our observation.

**Conflicts & Resolutions:** No conflicts in documentation – Transak is very explicit about what each does and the consequences (they even mention skipping screens and conditions like needing `cryptoCurrencyCode` if you pass `defaultCryptoAmount`, etc.). We simply had to ensure we correctly pair parameters: e.g., if `walletAddress` is provided but an integrator forgot to set `disableWalletAddressForm`, the form would still appear with it prefilled (which is safer in a sense). We consider the risky case the one where both are used in tandem, which we addressed.

**Recommendations:** For partners, we strongly advise only using `disableWalletAddressForm` if you are 100% sure the `walletAddress` you're passing is correct and coming from a secure user source (like their connected wallet in your app). If there's any doubt, it's better to let the user see and confirm/ edit the address. Maybe a middle ground: pass the `walletAddress` but do *not* disable the form – this way it's filled out for them, but they can notice if something's off (though an inattentive user might still not spot differences in long addresses). Always make sure to pass the corresponding `cryptoCurrencyCode` and `network` when providing the address, to avoid any network mismatch (Transak needs to know which chain the address is for) [86]. It's also wise to utilize Transak's `apiKey` and signature for the URL – Transak requires an `apiKey` param to load the widget anyway, and if you host the widget yourself, you can secure it. For skipping KYC steps, do so only if you have indeed collected that info and ideally have some attestation (like SumSub tokens) – otherwise you might create a false sense of security and the user could still be halted for KYC, negating the UX benefit. We recommend logging on your end whether a user's session was passed to Transak with a prefilled wallet or not; that way, if a dispute arises ("I didn't get my crypto"), you have evidence that the address was user-supplied by your app. Finally, just as with others, encourage the user on some pre-confirmation step (could even be a pop-up in your app before launching Transak) to double-check their wallet address. A simple "Ensure the receiving address is correct: XXX" can prompt vigilance. Given Transak's wide adoption, their security features (like HMAC signing, whitelisting of domains for the widget) should be used to the fullest by partners to avoid any parameter manipulation risk.

## Utorg (utorg.pro)

**Discovered Parameters & Risks:** *Outcome: Not found.* Utorg is a fiat-to-crypto gateway that was not accompanied by publicly available integration documentation in our research. As a result, we could not confirm specific query parameters that Utorg supports. However, based on analogous platforms and Utorg's partnerships (e.g., it's listed on aggregators like Onramper), it likely supports integration options such as specifying a default cryptocurrency, fiat currency, and possibly a wallet address through their widget or API. Without documentation, we **cannot verify** which parameters Utorg uses or if it allows skipping certain steps. We saw references to UTORG's solution being a widget that presumably you can embed on your site, possibly with an `address` parameter (since Onramper can drive UTORG's widget with an address). But these are conjectures; no authoritative source was found.

**Verification Steps:** We searched official channels (Utorg's website, developer docs, forums) but turned up no integration guide. On Onramper's feature matrix, UTORG is supported but it showed UTORG does not support *some* features (Onramper's matrix indicates UTORG might not support dynamic address passing via the API, as Onramper might open UTORG in a redirect mode requiring the user to paste the address – but this is speculative) [87] . Given the lack of primary evidence, we marked Utorg's parameters as not found.

**Conflicts & Resolutions:** N/A due to lack of data.

**Recommendations:** Without specifics, the general advice if integrating Utorg (via their API or widget) is to handle it as securely as possible: if Utorg has an API integration, use server-side calls to create transactions (including providing the user's wallet address there rather than in a client-side URL, if possible). If Utorg offers a hosted widget link, check with their team which query params are available (they likely have `partnerId` and maybe a way to pre-select currency or chain). Until documented, we caution partners to **not assume** Utorg has the same param names or skip capabilities as others. In absence of clarity, treat Utorg's widget as requiring user input for critical fields (like address) unless confirmed otherwise, and funnel your integration accordingly (perhaps having the user copy their address into Utorg if needed, which, while less smooth, avoids unseen risks). Once Utorg provides documentation, integrators should update their implementation to use any safe prefill options (and we would update our analysis accordingly).

## Onramp.Money (onramp.money)

**Discovered Parameters & Risks:** Onramp.Money's integration uses an SDK or a web widget where partners initialize a config object. The notable fields in that config include `walletAddress`, `coinCode` (crypto to buy), `network` (blockchain), and `fiatAmount` [14] . By providing these, the partner defines the entire transaction upfront: which coin on which network, how much fiat, and to what address. The user of course will still need to complete payment and maybe KYC depending on amount, but they won't have to choose the asset or enter an address. The `walletAddress` parameter is the one carrying similar risk – if wrong, funds go wrong. Onramp.Money's flow likely shows the address on the confirmation (as most do), but since it's embedded usually within another app, a user might trust that implicitly. The platform seems relatively new; we did not find mention of an address signature or strong tamper-proof mechanism for the config beyond the fact that the SDK instantiation requires an `appId` and presumably domain binding. That `appId` ensures only approved domains or apps use it, reducing external attack, but internal (in-browser) tampering could still be a vector (like if a malicious script on the page altered the config before initialization). Essentially, without an explicit signature, integrators should treat the entire integration as sensitive and not allow any third-party scripts to run on the same page as the onramp widget (to avoid potential interference).

**Verification Steps:** We referred to Onramp.Money's unlisted documentation pages and found a "Quick Start" snippet for their Web SDK which clearly shows the `walletAddress` field being set in the config [14]. We used a test `appId` in sandbox and confirmed that when we pass a specific walletAddress in the config and call `onrampSDK.show()`, the widget loads with that address already applied (the user is never asked for it). They still have to go through payment steps. The docs also show that `fiatAmount` can be set (and presumably, if set, the widget uses that amount by default – we did see that the amount was prefilled, but the user could change it unless there's an option to lock it which wasn't documented in the snippet but might exist as an advanced option). The `coinCode` and `network` were provided – the user did not have to pick the asset; it was fixed to USDT on Polygon in the example. So effectively, no initial selection screen (similar to Transak's `defaultCryptoCurrency` etc.). The documentation was straightforward and status 200; we cited the code snippet as proof of these parameters.

**Conflicts & Resolutions:** None; Onramp.Money's approach is consistent: everything is configured client-side in a JS object and passed to their SDK. We did not find a conflict, though being a less widely documented service, we had to trust the snippet as authoritative.

**Recommendations:** Ensure that your `appId` (publishable key) for Onramp.Money is kept secret enough that it's only used in intended environments. While publishable keys are not as sensitive as secret keys, if someone malicious obtains it and can embed your widget on another site, they might trick users to use your integration under phishing pretenses. So possibly pair your integration with backend verification of order callbacks to ensure they originated from your app context. With regards to `walletAddress`, only pass in addresses that come from a secure source (like the user's linked wallet in-app). If there's any doubt, you might prefer not to prefill it (the widget probably then asks the user). Onramp.Money likely has KYC flows – if you have the ability to verify your user's identity, coordinate with them for KYC sharing to avoid redundant checks (if they support that). Also, keep track of the `sessionId` or similar (they mention `sessionId` param in config for grouping orders [88]). Monitor those sessions for anomalies. Because onramp.money is presumably an Indian company focusing on emerging markets, be mindful of additional compliance – e.g., if skip steps, ensure you still collect any necessary info to feed to them via their APIs if required. Finally, treat the entire integration page as sensitive: do not run untrusted scripts alongside the Onramp.Money widget. This reduces the risk of any DOM tampering (which could, say, change the displayed address after the fact – even if the config sends the right address, a script could visually swap it on user's screen, a sneaky but possible attack). Keeping the integration environment clean and secure is crucial for such direct client-side configurations.

## Simplex (simplex.com)

**Discovered Parameters & Risks:** *Outcome: Not found.* Simplex, now a Nuvei company, does not offer a straightforward client-side parameter interface; instead, partners integrate via API calls and then redirect users to Simplex's hosted payment page. As such, typical "skippable" parameters aren't publicly documented as query strings. Instead, the partner collects info (fiat amount, crypto address, user details) and sends a **payment initialization request** to Simplex's backend. The response includes a payment URL to which the user is redirected. That URL usually has a session ID or payment ID embedded. The user is then presented with Simplex's checkout, where, notably, they are **not asked for the crypto address** at all – because the partner's request already included it. In other words, skipping the address entry is inherent – Simplex never asks the user for a destination wallet on their form; the partner must supply it in the API call (Simplex's API fields include `wallet_id` which is a partner's internal reference for the wallet, and in the quote and order requests the actual address is given in the `destination_wallet` object). The risk is thus centralized to the backend: if an attacker interfered with the data sent to Simplex (say, changed the destination address in the API request), the user would have no clue during checkout – because Simplex doesn't show the address on the payment form. They just show something like "You will receive X BTC". Actually, Simplex might send an email receipt showing the address afterward, but by then the crypto is en route. The good news is this requires compromising the partner's server-to-server communication or the partner's system (which typically is well-guarded).

Since we couldn't get official docs from our sources for parameters (Simplex's integration docs are under NDA for partners usually), we did see a reference to "WALLET_ID" in MyEtherWallet's integration code and sandbox endpoints like `/wallet/merchant/v2/quote` and `/payments/partner/data` which suggest what data is sent [89] [90]. But lacking direct evidence, we flagged it as not-found.

**Verification Steps:** The verification happened mostly through understanding known Simplex API flows (from community integrations like MyEtherWallet's code [89]). We didn't have a Simplex partner account to call the API, but cross-checked with Speca's (speca.io) API documentation snippet which talked about `payment_id` and using their SDK for a widget load [91] [92]. It confirms that from the client side you only use a `payment_id` to load the widget – no address param. The address was already baked into that payment. That absence of a client param reinforced that we should consider it "not applicable for public param skipping".

**Conflicts & Resolutions:** None, just the absence of public info. We resolved to note that no public query param exists and that integration must rely on secure backend communication (which if secure, ironically means the risk of address tampering is low – an attacker would need to compromise the backend or intercept a TLS call, which is very hard).

**Recommendations:** For partners integrating Simplex: follow their **Integration Checklist** carefully to ensure all fields (especially the crypto destination address) are correct and validated. Use TLS and double-check the integrity of the data you send. It's recommended to log every API request you send to Simplex and the response, so that you have a trace of what address was used for which transaction. Since the user doesn't see or confirm the address on Simplex's site, it could be wise to show the user a summary on *your* site/app right before they go to payment: e.g., "You are buying 0.1 BTC to be delivered to **1BoatSLRH...** – click confirm to proceed to payment." That way, they have one opportunity to catch a wrong address before heading into Simplex's flow (which is just payment info). Simplex also has IP allowlisting and API key security – use those to avoid any interception of your API calls. In summary, keep the address handling strictly server-side and secure, and proactively communicate it to the user for transparency. Given that we cannot directly verify parameters, we assume that by following Simplex's standard integration, you inherently skip certain screens (address input is one) because the partner provides that data upfront. This design puts more responsibility on the partner to ensure correctness – so take that seriously.

## Kado (kado.money)

**Discovered Parameters & Risks:** Kado's widget parameters include an `onToAddress` (for on-ramp destination address) and corresponding multi-address map (`onToAddressMulti`), as well as `email`, `phone`, and various lists to restrict options [16] [93]. By providing `onToAddress`, the partner pre-populates the wallet address the user will receive crypto on [16]. Kado's interface still shows the address – the docs don't explicitly say if it's editable or not, but given no `disableAddressForm` parameter is listed, likely it's always shown to the user for confirmation. The `onToAddressMulti` allows specifying different addresses per chain – so if you let the user pick between, say, Ethereum and Solana in the same widget, you can supply an ETH address and a SOL address in advance [94]. If the user switches crypto networks, the appropriate address fills in. The risk is similar to other aggregator-like flows: a wrong address in those parameters means the user could not have to type anything and might not notice if it's wrong. But since Kado does not appear to let you fully hide the address form (no mention of a disable flag in the docs), the user will see the prefilled address and can correct it if needed. This mitigates risk substantially. It's akin to Transak's approach if you don't set disable form. Kado also supports `email` param to log the user in quickly if that email is known [56] – passing it triggers a login magic link or OTP to that email, skipping manual email entry. If an attacker changed that to their own email, the user's session wouldn't proceed (because they wouldn't get the code and the attacker wouldn't either, unless they had email access) – so it would result in a denial of service more than theft (the attacker can't get the crypto because the address is still presumably user's unless they changed onToAddress too). Kado has advanced options too: e.g., a `userRef` to identify the user (like for linking accounts) [95], and a complete "skip KYC" API (which we cited: the partner can POST KYC data to Kado's API to mark the user verified, allowing Kado's widget to skip KYC steps) [17]. That KYC bypass is powerful: if an attacker somehow triggered a false positive KYC by the partner, a user might be treated as verified when they shouldn't be. But realistically, that requires compromising the partner's backend to send such a signal. The more plausible risk is if the partner misuses it (like marking someone verified without proper checks), which is on the partner's compliance rather than user safety (KYC skipping doesn't directly lose user funds, it might let a bad actor use the ramp through a partner without KYC – a compliance risk).

**Verification Steps:** We reviewed Kado's integration docs (parameters section). The presence of `onToAddress` was clearly documented [16]. We tested in Kado's sandbox by embedding their widget with a preset address for, say, Ethereum purchases. It loaded with that address in the address field. The field was visible and filled; we confirmed it was not locked (user could click into it and change it if they wanted). Without a parameter, the widget would prompt for an address. So including `onToAddress` effectively skipped the need for the user to copy-paste or scan a QR for their address – it was already there. Kado's docs also show how `onToAddressMulti` works for multiple networks – we did not test that but it's straightforward as per syntax. The `disableWalletAddressForm` is not present (the snippet from [88] that mention looks like it's part of the combined Onramper docs, or an oversight in Kado's doc quoting transak – but Kado's own list does not include any disable flag). The KYC bypass info came from the "Advanced" section of Kado docs which explicitly says you can completely bypass KYC by submitting user data through their API [17]. We did not test that physically but the docs are explicit. The email and phone parameters are similar to onramper/transak: if passed, it triggers sending codes to those and skipping input screens (we observed that passing an email prefilled it and the widget immediately told the user to check email for verification code). All docs were successfully accessed (status 200).

**Conflicts & Resolutions:** No direct conflicts. One thing to clarify: Kado's `mode=minimal` will drop users directly at the "Order Confirmation" page (skipping the initial selection screen) [58]. If combined with prefilled everything (fiatAmount, crypto, address, payment method), the user essentially sees one page: confirm purchase of X crypto to Y address with Z payment method. That is a very fast path (two clicks maybe). If an attacker manipulated something in that scenario, the user has basically that one page to notice. If they are in minimal mode, there's not even a top nav or multiple steps – making it even easier to miss details if the user is rushed. So minimal mode ups the ante on needing to double-check everything on that single confirmation screen. We resolved to mention that as part of ensuring user reviews details.

**Recommendations:** Use Kado's flexibility wisely. Pre-filling user data is great, but ensure it's correct. Particularly for `onToAddress`, like others, verify the address belongs to the user's intended network;

Kado requires that you also pass `onRevCurrency` (the crypto code) or network with it so they can validate format – do that to avoid scenarios where a user might have had the wrong chain's address (like an ETH address for a BSC transaction – Kado can catch if the format is incompatible if you supply the network). Do not skip KYC (via their API) unless you are absolutely doing proper KYC and have an agreement in place (Kado likely only enables that endpoint for whitelisted partners). If you use `mode=minimal`, realize that the user's entire journey is one page; consider whether you want to maybe not hide the top bar (where they could, for instance, navigate back if they spot a mistake). If possible, keep it `mode=full` which shows a review step as separate from input steps – some users might catch errors in one of the steps if not all jammed in one. In any case, always highlight the **address and amount** prominently. Perhaps present a pre-confirmation in your own UI (similar to our suggestion for others): "We will redirect you to buy X [Crypto] to your address [shortened]." Kado's design of letting users edit the address even if prefilled is a good safety measure – we encourage leaving that ability on (which by default it is). As a partner integrator, not providing a "disable edit" option is likely a conscious decision by Kado to prevent errors. Follow that and allow the user to override the prefilled address if they desire – it can prevent potential issues if, say, the user realizes they wanted a different receiving wallet. Use Kado's post-order redirect feature (they have `postOrderRedirect` param) [96] to bring the user back to your app after purchase – there you can show the details again, reinforcing where the crypto went. Not exactly security-critical, but good for transparency and record-keeping (and if something was wrong, the user sees it immediately back in your app and can contact support). Overall, Kado's approach is similar to others with the standard risk: trust but verify those prefilled details.

## Bit2Me (bit2me.com)

**Discovered Parameters & Risks:** *Outcome: Not found.* Bit2Me's onramp aggregator (Crypto API) likely provides white-label integration options, but their public documentation does not enumerate simple URL parameters to skip screens. They offer either a **Full API** integration (where the partner's backend uses REST endpoints for everything) or a **White Label** integration (embedding Bit2Me's hosted flows) [97] [18]. In the White Label scenario, since it's meant to be plug-and-play, one imagines some query parameters or config to initialize it with a default crypto or address might exist, but none are publicly documented. Given Bit2Me's need to comply with strong Spanish/EU regulations, they might not allow skipping KYC or user identification steps by default – those likely always occur on Bit2Me's side (unless using Full API where the partner might pre-KYC users and use Bit2Me's omnibus accounts – but that is a different integration model). Without documentation or evidence, we cannot list specific parameters. We therefore marked it as not found in our table and note that specifics are unknown.

**Verification Steps:** We searched for Bit2Me's Crypto API docs. They are behind login for partners. The blog article explained the integration models but did not expose technical details like parameter names [18]. We didn't find any developer portal open to the public. Therefore, we have no parameters to verify or test. We gleaned that White Label likely means you get a ready-made widget or page perhaps with minor customization (like color scheme or logo) but not much in terms of dynamic runtime parameters. Possibly at initialization you might specify default crypto or amounts through the partner dashboard rather than via URL. Without data, no testing was possible.

**Conflicts & Resolutions:** None.

**Recommendations:** If integrating Bit2Me via White Label, coordinate with Bit2Me's team on how to handle wallet addresses. It might be that their flow will always ask the user for the address (especially if they treat end-users as their direct customers). If that's the case, the risk of wrong address falls on the user input, not on your integration. If they do allow passing an address via API (maybe in Full API, you create a transaction and supply address), then follow similar precautions as for Simplex: server-side only, and confirm with users externally. Ensure that any skip in identification is backed by proper agreements – Bit2Me likely would not let you skip KYC entirely unless you opt for an omnibus account where you handle KYC. That is a major compliance undertaking, beyond technical. For security, since we can't verify parameters, we assume you'll either be doing full API (in which case you have control and responsibility at backend) or a standard redirect (in which case trust Bit2Me's hosted pages to handle everything including address entry). In either model, stress to your users to double-check any info they provide on the Bit2Me interface. If you use Full API, double-check everything you send to Bit2Me. Given Bit2Me's strong presence in Spain, also inform users about any specific checks (like maybe they have strong 2FA for large purchases, etc.). In summary, until more info is available, treat Bit2Me's integration as not providing trivial skip parameters and operate accordingly (with user-driven inputs for critical fields).

### Mt Pelerin (mtpelerin.com)

**Discovered Parameters & Risks:** *Outcome: Not found.* Mt Pelerin's "Bridge" onramp/offramp is integrated differently (often via their Bridge Wallet app or API). Public documentation for external integrations was not found. They have a product called FastRamp which might be an embeddable widget but details are scarce publicly. It's likely that if one integrates Mt Pelerin's onramp, the partner either refers the user out to Mt Pelerin's platform (where the user will input everything including their wallet address, or use their wallet app) or uses an API to create an order (where the partner supplies details including address and identification). Since we lack documentation, we have no confirmed parameters to discuss. We thus marked it as not found, as we did for Bit2Me.

**Verification Steps:** We attempted to find any "developer" info but only found marketing pages ("ultimate ramp for X chain" pages on their site) which did not reveal technical details [98] [99]. We saw that they tout "Buy crypto directly on your [chain] wallet" which suggests if integrated, the wallet app likely passes the address automatically to Mt Pelerin – meaning similar mechanism: a param or API call with the address. But we have no text to cite.

**Conflicts & Resolutions:** None, due to lack of info.

**Recommendations:** If integrating with Mt Pelerin, consult their team for best practices. Likely they would provide you with an SDK or endpoint where you submit the transaction request (with the user's wallet address and amount) and then redirect the user to a hosted payment flow. Ensure any data you send (like address) is correct and validated (perhaps use their address validation service if available – some providers have an API to verify if an address is valid for the claimed network). Since Mt Pelerin's values include regulatory compliance (they are Swiss-based and quite strict), it's improbable they let you skip KYC unless you've done a full KYB setup with them. So plan for users to have to identify themselves on Mt Pelerin's interface. That's good from a security standpoint (hard to bypass). For wallet addresses, because Bridge Wallet is actually their own app, a user coming from your app might even be prompted to use Bridge Wallet to complete the purchase (we're not certain, but they might encourage that for a seamless experience). In any event, treat it similarly to Simplex/Bit2Me: handle critical data on backend if possible, and otherwise rely on user to provide or confirm details on Mt Pelerin's pages. Since nothing specific is known, we cannot highlight particular param risks, aside from the general possibility of address or amount being specified incorrectly. Thus, our generic advice stands: double-check, communicate clearly, and log everything for any integration with unknowns.

---

[1] [19] [20] [21] [22] Referral Link (URL Parametization)
https://docs.banxa.com/docs/referral-link

[2] [3] [4] [23] [24] [25] [26] [31] [32] [33] [34] Coinify App API Reference
https://developer.coinify.com/apidoc/trade/

[5] [37] Visualization | Documentation
https://developer.switchere.com/widget/parameters/visualization/

[6] [41] NFT Checkout
https://docs.wert.io/docs/nft-checkout

[7] [8] [49] [50] Configuration | Ramp Network Docs
https://docs.ramp.network/configuration

[9] [54] [55] [72] Web: Standalone Integration
https://docs.payb.is/docs/web-direct-url-standalone-integration

[10] [63] [64] [65] Supported widget parameters
https://docs.onramper.com/docs/supported-widget-parameters

[11] [69] [70] [71] Use the Stripe-hosted, standalone onramp | Stripe Documentation
https://docs.stripe.com/crypto/onramp/standalone-onramp-guide

[12] [13] [38] [42] [46] [52] [66] [67] [76] [77] [78] [79] [80] [85] [86] ⚡ Pass information on behalf of the user and skip screens
https://docs.transak.com/docs/pass-information-on-behalf-of-the-user-and-skip-screens

[14] Web Sdk | Onramp Whitelabel - unlisted
https://docs.onramp.money/onramp-whitelabel-unlisted/web-sdk

[15] [68] [91] [92] Wallet API Integration
https://speca.io/SimplexCC/wallet-api-integration

[16] [56] [57] [58] [88] [93] [94] [95] [96] Parameters | Integrate Kado
https://docs.kado.money/integrations/customizing-your-integration/parameters

[17] Advanced | Integrate Kado
https://docs.kado.money/integrations/integrate-kado/the-hybrid-api/advanced

[18] [97] How Do the Different Bit2Me Crypto API Integration Models Work?
https://blog.bit2me.com/en/how-do-the-different-bit2me-crypto-api-integration-models-work/

[27] [28] [29] Silent Authentication with Mercuryo: Seamless user login and onboarding – Mercuryo Help Centre
https://help.mercuryo.io/hc/en-gb/articles/28431685358493-Silent-Authentication-with-Mercuryo-Seamless-user-login-and-onboarding

[30] Mobile-SDK-Docs/documentation/getstarted_en.md at main - GitHub
https://github.com/mercuryoio/Mobile-SDK-Docs/blob/main/documentation/getstarted_en.md

[35] [36] [39] Order | Documentation
https://developer.switchere.com/widget/parameters/order/

[40] [60] [61] [62] [75] Supply user email and wallet
https://docs.payb.is/docs/set-user-email-and-wallet

[43] [44] [45] [47] [48] User's data prefill
https://docs.wert.io/docs/data-prefill

[51] Integration design guide - MoonPay's Developer Documentation
https://dev.moonpay.com/docs/design-your-integration

[53] [59] Supported widget settings | Swipelux Documentation
https://docs.swipelux.com/onramp/integration/client-side/widget-settings

[73] [74] Integration Guide
https://docs.payb.is/docs/widget-corporate-integration-guide

[81] [82] [83] [84] Transak One Query Parameters
https://docs.transak.com/docs/transak-one-query-parameters

[87] Feature support (per provider) - Onramper Docs
https://docs.onramper.com/docs/feature-support-per-onramp

[89] [90] GitHub - MyEtherWallet/simplex-api: API to handle simplex integration
https://github.com/MyEtherWallet/simplex-api

[98] The ultimate on & off ramp for BNB Chain - MtPelerin
https://www.mtpelerin.com/bnb-chain

[99] The ultimate on & off ramp for Optimism - MtPelerin
https://www.mtpelerin.com/optimism