**Integration Guide**

This page describes how to onboard zkMe services

**Onboard procedure**

|  |  |
| --- | --- |
| **Service Integration** | * Provide your business Email account, we will open a zkMe dashboard account for you. * Customise everything and get the API key on dashboard based on your requirements. * Begin technical integration with your dApp. |
| **Testing and Optimization** | * Execute functional, performance, and security testing. * Based on testing results, perform necessary optimizations and adjustments. |
| **Training and Support** | * Prior to going live, conduct training sessions for the client's team to ensure they understand how to use zkMe service. * Provide necessary support to assist them in resolving any issues. |
| **Go-Live** | * After all preparations, launch the service officially for the web3 dApps. |
| **Continuous Service and Monitoring** | * Continuously monitor the performance of the service to ensure stability and reliability. * Based on client feedback and requirements, continuously optimize and upgrade zkMe service. |

**Integrate with zkMe**

**Step 1: Register with zkMe**

The first step is to register your project with zkMe. This will involve providing some basic details about your project, such as its name, the nature of your project, and your contact information. You will also need to agree to zkMe's Terms of Service and Privacy Policy.

**Step 2: Configure Verification Levels**

Next, you'll need to configure the verification levels for your project. zkMe offers Basic, Standard, and Advanced levels of verification, and you can choose the one that best suits your project's needs. You can also configure any add-ons at this stage.

**Step 3: Integrate zkMe API**

Once you have registered and configured your verification levels, you'll need to integrate the zkMe API into your project. This will involve adding the appropriate code to your project to connect it with zkMe's servers. zkMe will provide documentation and technical support to assist with this process.

**Step 4: User Onboarding**

With the zkMe API integrated, you can now begin onboarding users. When a user signs up for your project, they will be directed to complete the zkMe verification process. The specifics of this process will depend on the verification level you have chosen:

* For the Basic level, users will need to provide proof of personhood, complete a liveness check, and a uniqueness check.
* For the Standard level, users will also need to provide proof of citizenship, and complete AML, KYT, ID document verification, Age of Majority checks, PEP/Sanction lists checks, and adverse media checks.
* For the Advanced level, users will also need to verify that they are accredited investors.

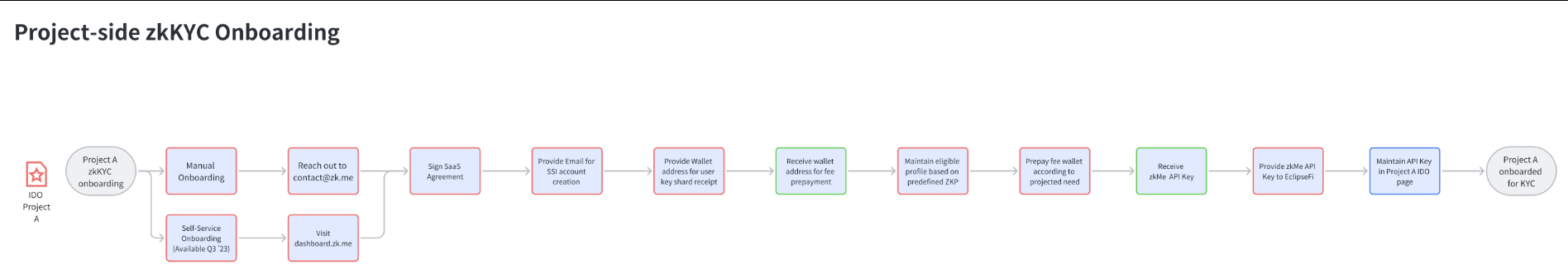
**Step 5: Verification Status**

Once a user has completed the verification process, zkMe will return a verification status to your project. This will inform you whether the user has successfully completed the verification process and at what level.

**Step 6: Ongoing User Management**

After a user has been onboarded, zkMe provides tools for ongoing user management. This includes tools for monitoring user activity, re-verifying user identities if necessary, and managing user data in compliance with privacy laws and regulations.

**Project-side zkKYC Onboarding**



​

# zkMe zkKYC SDK

zkKYC SDK - Simply execute KYC progress using the RedirectURL as the src-attribute value for the iFrame.

### Use Case

To reduce the development cost for the project side, the project can use zkKYC capability by simply accessing the link. Users can achieve full KYC capability on the web/H5, reducing the possibility of user churn by minimizing the need to navigate to another page.

### ****zkMe-Widget KYC Process****

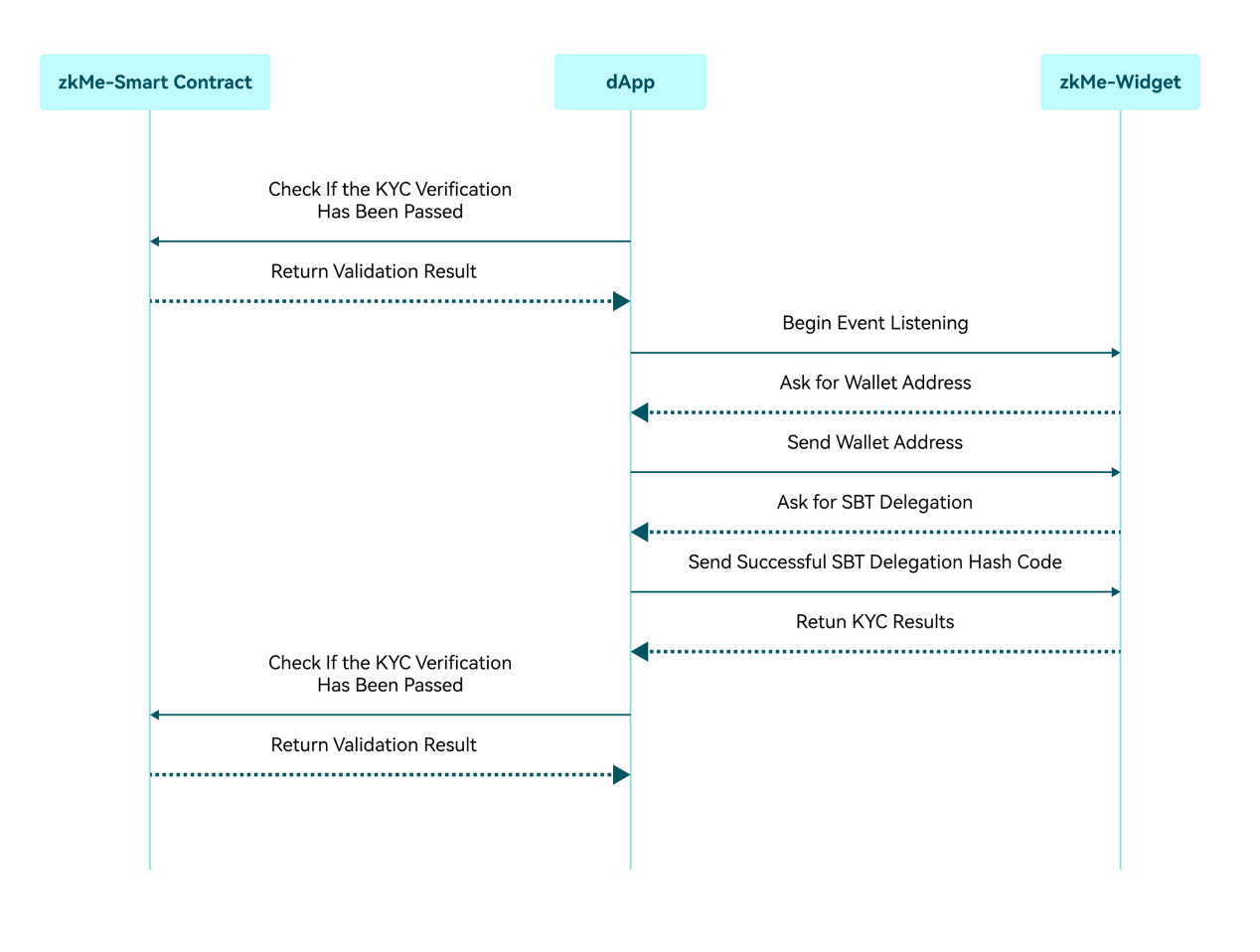
**Step 1:** Enter the service authorization Widget page, the user confirms and goes to the next step

**Step 2:** E-mail verification login

**Step 3:** Verify the SBT to confirm that it is authenticated

**Step 4:** Depending on the KYC configuration of the project, determine whether the user needs to undergo different verification processes.

### Interaction Instructions



### Integrate with iFrame

An iFrame is an HTML tag that allows you to embed another HTML document or page within the current one. It's often used to display content from another website or to embed media content such as videos. In this case, the iframe is being used to redirect the user to a URL that will allow them to perform liveness detection and document scanning. The iframe is a secure way to integrate the zkMe Widget into the user's application.

The iFrame’s URL: <https://widget.zk.me/>​

<iframe allow="camera" src="<%= REDIRECT URL %>"></iframe>

**NOTE:**

Adding the **allow="camera"** attribute allows the user to perform liveness detection and document scanning.

#### ****Security Considerations****

The iFrame integration is a secure way to integrate zkMe-widget into your application. However, you must take into account the following security considerations:

* Review your Content Security Policy (CSP).
* Employ defensive code in the UI to ensure that the current frame is the top-level window.
* Properly set authentication cookies with SameSite=Strict (or Lax) and Secure flags.

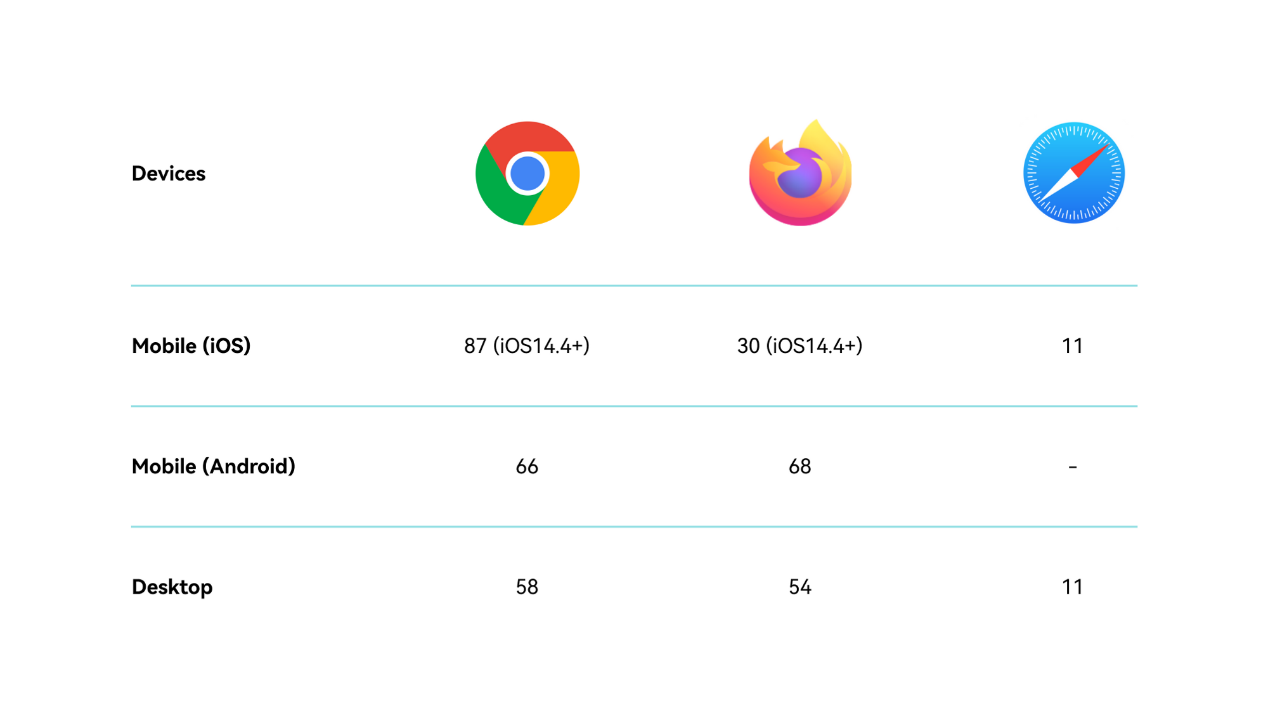
We strongly recommend you review the following resources to learn more about security considerations when using iFrames:

* ​[Clickjacking "UI redress attack"](https://owasp.org/www-community/attacks/Clickjacking)​
* ​[Cross Frame Scripting (XFS) attacks](https://owasp.org/www-community/attacks/Cross_Frame_Scripting)​
* ​[OWASP CheatSheet HTML5 Security](https://cheatsheetseries.owasp.org/cheatsheets/HTML5_Security_Cheat_Sheet.html)​

#### ****iFrames and the**** ****CORS-policy****

Cross-Origin Resource Sharing (CORS) is a security policy, that handles the sharing of external resources on a given website, for instance externally hosted Javascript libraries, but also iFrames. If this policy is not followed in a correct manner, it can happen, that the iFrame is blocked by the respective browser. When integrating zkMe-widget into an iFrame, please always consider this policy.

**Compatibility**



### Usage Example

**Exchanging API\_KEY for Access Token**

To use your API\_KEY to obtain an accessToken, you will need to make a specific HTTP request. Here's how you can do it:

a. **Endpoint**: Send a POST request to the token exchange endpoint.

POST https://nest-api.zk.me/api/token/get

b. **Request Body**:

| Parameter Name | Required | Type | Desc |
| --- | --- | --- | --- |
| apiKey | True | string | The API\_KEY provided by zkMe. |
| appId | True | string | A unique identifier (mchNo) to DApp provided by zkMe. |
| lv | True | number | 1-zkKYC 2-Anti-Sybil |

c. **Response**

**Success**

{

"code": 80000000,

"data": {

"accessToken": "8641259808779c53de65c3698e42b402b112cfe3856202189c37eae9f0b23babbcc1429ea9adcb52283dca4dab024a640651f855d8c78c7bde308f721a6e0cb80d51dab7c775ebfe0ae74eb9ab02f503094a9b2a2e2aeabf70e03a0cac9773a93dba743ca0dc3fa4af77375351bc48f76515d72dbee3a8bd5c034e6ffb94bd97"

},

"msg": "success",

"timestamp": 1691732474552

}

**Exception (AppId and API\_KEY not matched)**

{

"code": 81000014,

"data": null,

"msg": "AppID and API Key do not match. Access token generation failed",

"timestamp": 1691732568774

}

**Exception (Parameter Error)**

{

"code": 80000002,

"data": null,

"msg": "parameter error",

"timestamp": 1691732593484

}

**Exception (System Error)**

{

"code": 80000001,

"data": null,

"msg": "system error",

"timestamp": 1691732593484

}

### HTML

<style>

.app-wrap {

background: #fff;

display: flex;

align-items: center;

justify-content: center;

}

​

#zkme-widget{

width: 510px;

height: 700px;

border: 2px solid #005563;

border-radius: 20px;

}

</style>

<html>

<div class="app-wrap">

<iframe id="zkme-widget" title="zkme-widget" width="100%" height="100%"

src="https://widget.zk.me/?mchNo=YOUR\_MERCHANT\_NUMBER&name=YOUR\_PROJECT\_NAME&chainId=HEX\_CHAIN\_ID&accessToken=ACCESS\_TOKEN" />

</div>

</html>

**Query params for Iframe embedding:**

| Parameter Name | Required | Description |
| --- | --- | --- |
| mchNo (appId) | Yes | A unique identifier to DApp provided by zkMe. To get a mchNo, [https://zk.me](https://zk.me/)​ |
| name | Yes | Project name. |
| chainId | Yes | A hexadecimal number uniquely identifies the supported chain. |
| accessToken | Yes | The access token could be exchanged by using the provided API\_KEY. To get an API\_KEY, [https://zk.me](https://zk.me/)​ |
| origin | No | The origin name of your project e.g. encodeURIComponent(location.origin) |

**Supported chains:**

| Chain | Testnet | Mainnet |
| --- | --- | --- |
| Ethereum | 0x5 | coming soon |
| Polygon | 0x13881 | 0x89 |
| Scroll | 0x82751 | coming soon |

### TypeScript

When adding a listener event, please note that it is from the same origin as zkMe-widget

**Check if the KYC has been passed (DApp and zkMe-Smart Contract)**

import { Contract } from "ethers"

const contract = new Contract(

ZKME\_VERIFY\_CONTRACT\_ADDRESS,

ZKME\_VERIFY\_ABI,

RPC\_PROVIDER

)

const isApproved = await contract.hasApproved(DAPP\_ACCOUNT, walletAddress)

**ZKME\_VERIFY\_ABI**

zkmeverify.abi.json

16KB

Code

**Communication between DApp and zkMe Widget**

import { Contract } from "ethers"

​

interface ZkMeMessageBody {

id?: string;

method?: 'getUserAccounts' | 'delegateTransaction';

params?: TransactionRequest;

kycResults?: 'matching' | 'mismatch';

verifiedAddress?: string;

event?: 'close' | 'finished';

}

​

window.addEventListener('message', async(ev: MessageEvent<ZkMeMessageBody>) => {

const zkMeWidgetOrign = 'https://widget.zk.me'

​

// Check if it is the same origin

if (ev.origin !== zkMeWidgetOrign) {

return

}

​

if (kycResults === 'matching') {

//KYC Verification Completed

//zkMe Widget can be turned off here ...

//Determine if the address that passes verification is the same as the currently connected address ...

//Check the KYC verification has been passed

const contract = new Contract(

ZKME\_VERIFY\_CONTRACT\_ADDRESS,

ZKME\_VERIFY\_ABI,

RPC\_PROVIDER

)

​

const isApproved = await contract.hasApproved(DAPP\_ACCOUNT, walletAddress)

} else if (event === 'close') {

//zkMe Widget close event

}

​

if(method ==='getUserAccounts'){

//Send wallet address

}else if(method === 'delegateTransaction'){

//Send successful transaction hash code

}

})

id is randomly generated and needs to be returned as-is by the dApp when sending messages.

message field is used to indicate whether the call to the dApp interface was successful or not. 'matching' indicates success, while any other value indicates failure.

**Send wallet address**

//If the wallet is connected

if(method ==='getUserAccounts'){

try {

const accounts = await window.ethereum.request({

method: "eth\_requestAccounts"

});

​

ev.source.postMessage({

id,

message: "ok",

data: accounts,

}, ev.origin);

} catch (err: any) {

ev.source.postMessage({

id,

message: err.message,

}, ev.origin);

}

}

**Send successful SBT delegation hash code**

​

if (method === 'delegateTransaction') {

try {

const { params } = ev.data;

​

await window.ethereum.request({

method: 'wallet\_switchEthereumChain',

params: [{

chainId: params.chainId

}]

})

​

const tx = await signer.sendTransaction(params);

​

ev.source.postMessage({

id,

message: "ok",

data: tx.hash,

}, ev.origin);

} catch (err: any) {

ev.source.postMessage({

id,

message: err.message,

}, ev.origin);

}

} else {

ev.source.postMessage({

id,

message: "Unauthorized",

}, ev.origin);

}

**Event (accountsChange)**

//example

ethereum.on('accountsChanged', (accounts) => {

ev.source.postMessage({

event: 'accountsChanged',

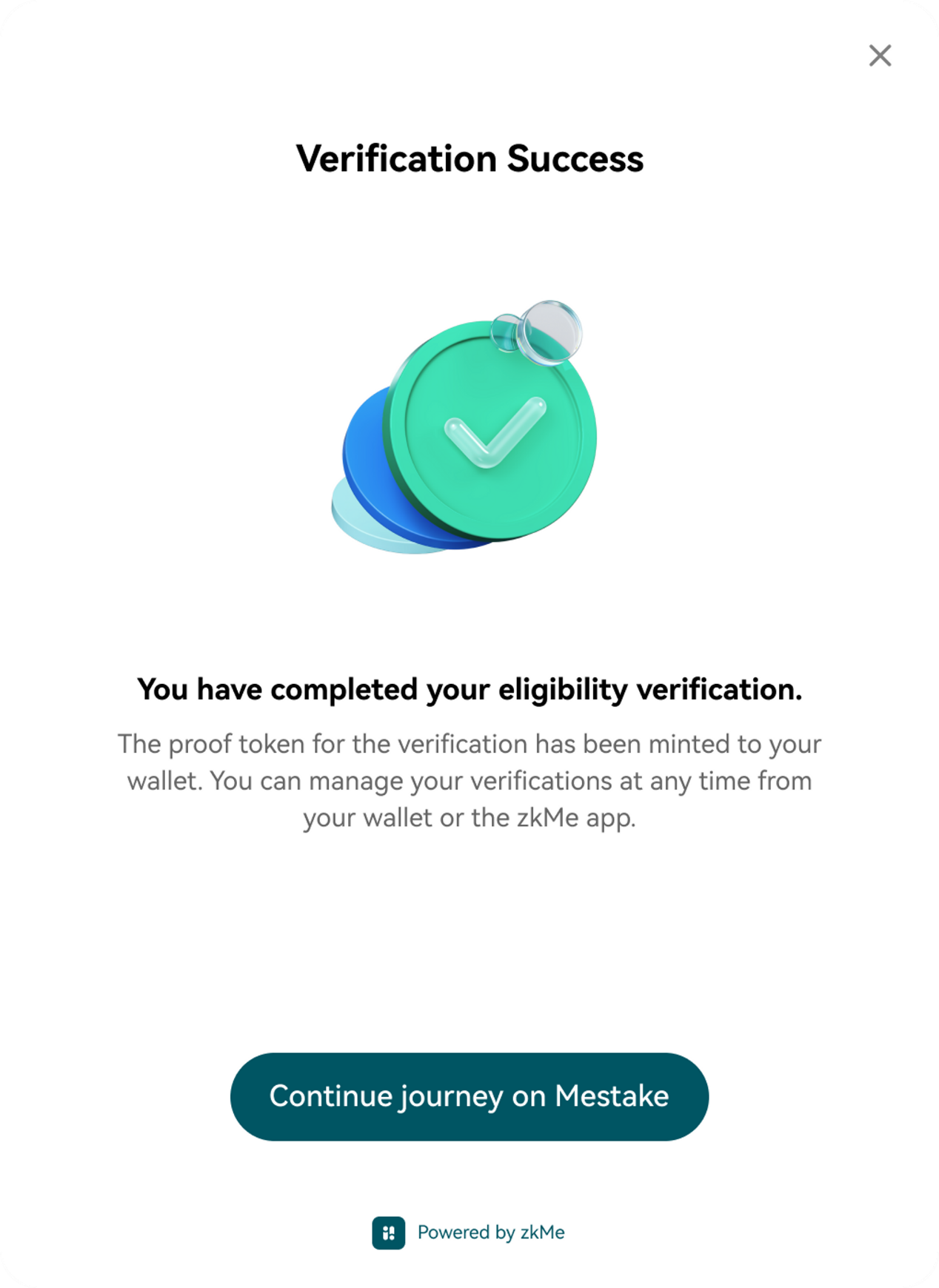
data: accounts

}, ev.origin)

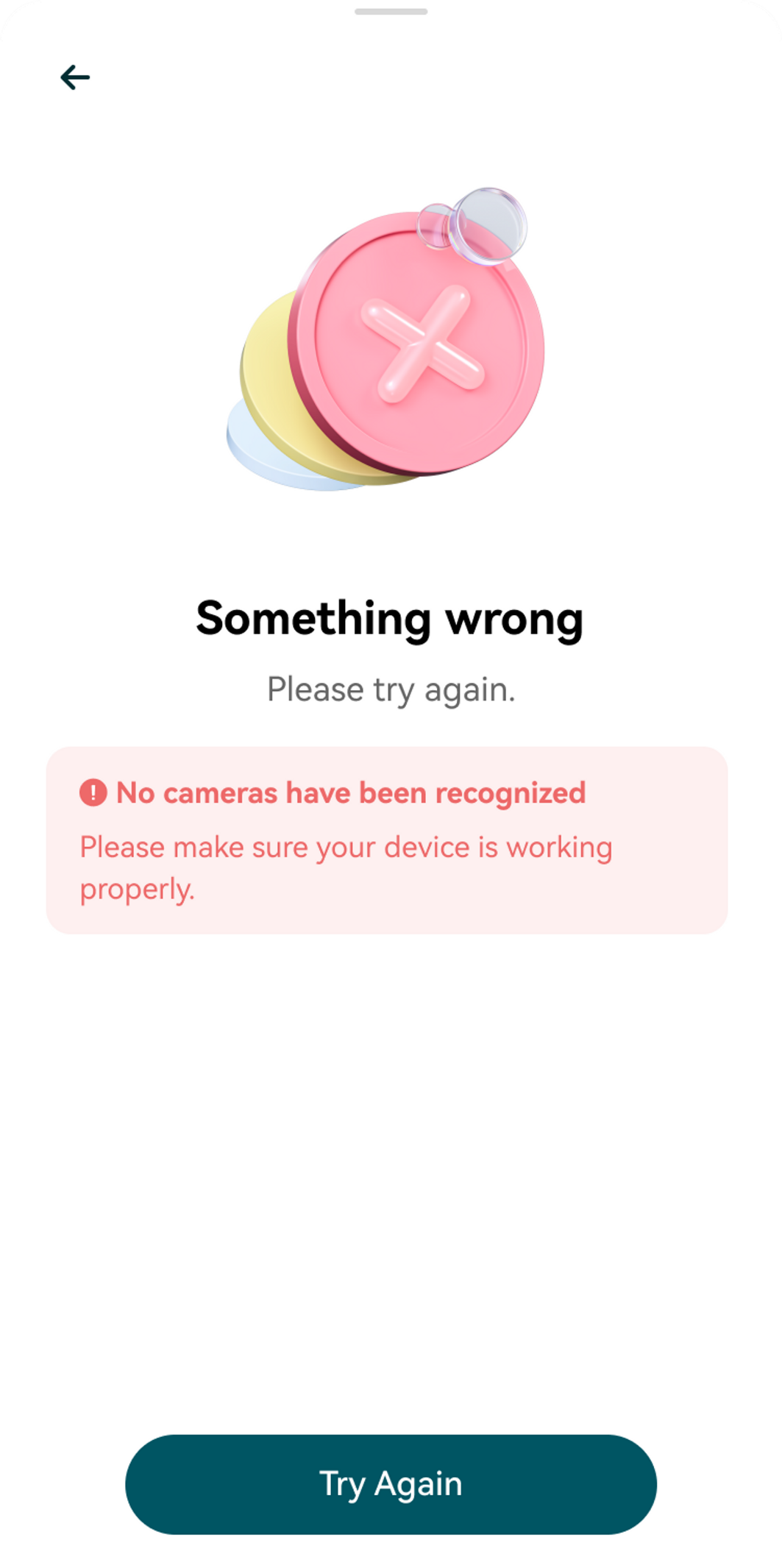
})

### Response & Exceptions

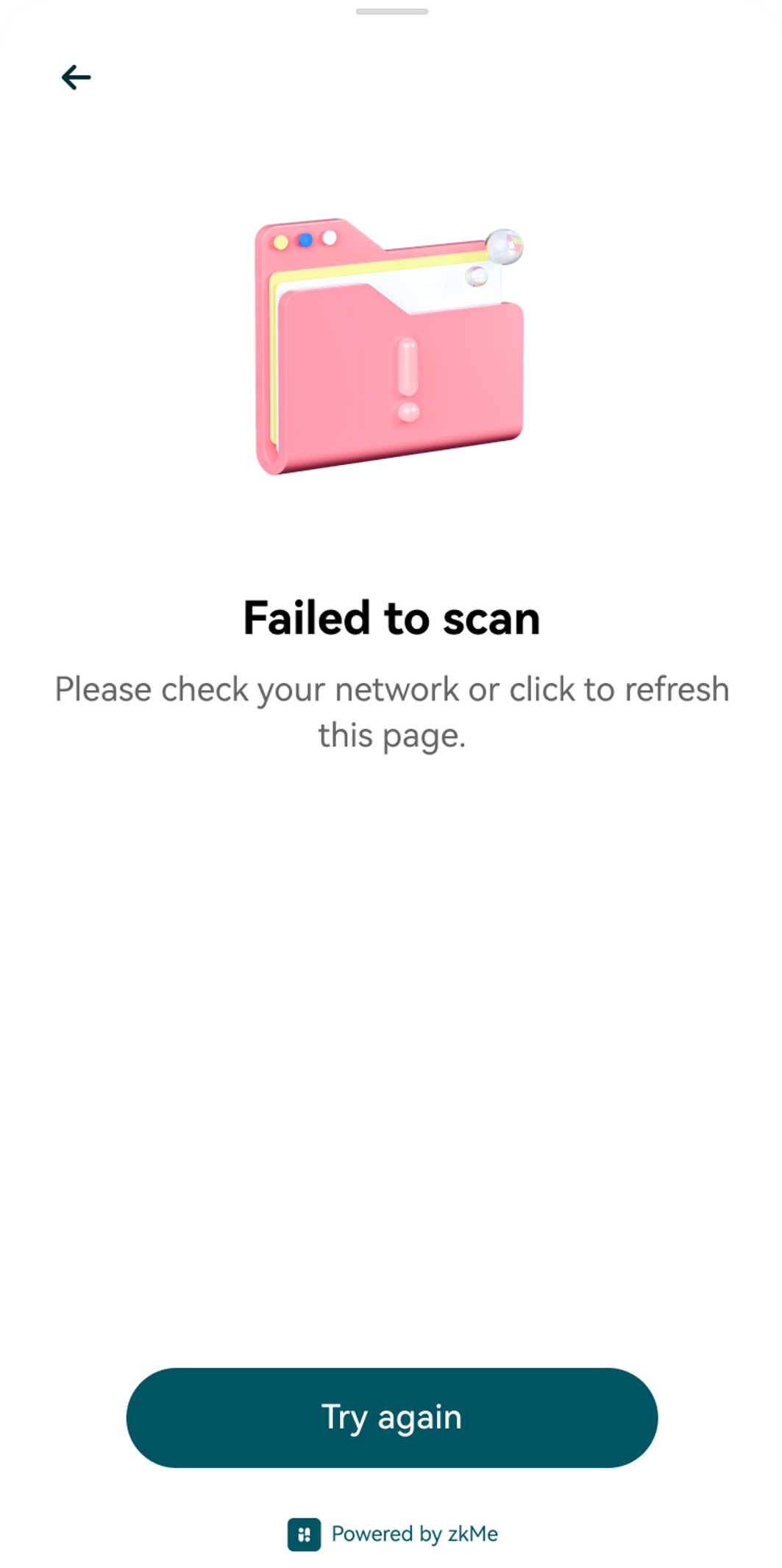
**Success**



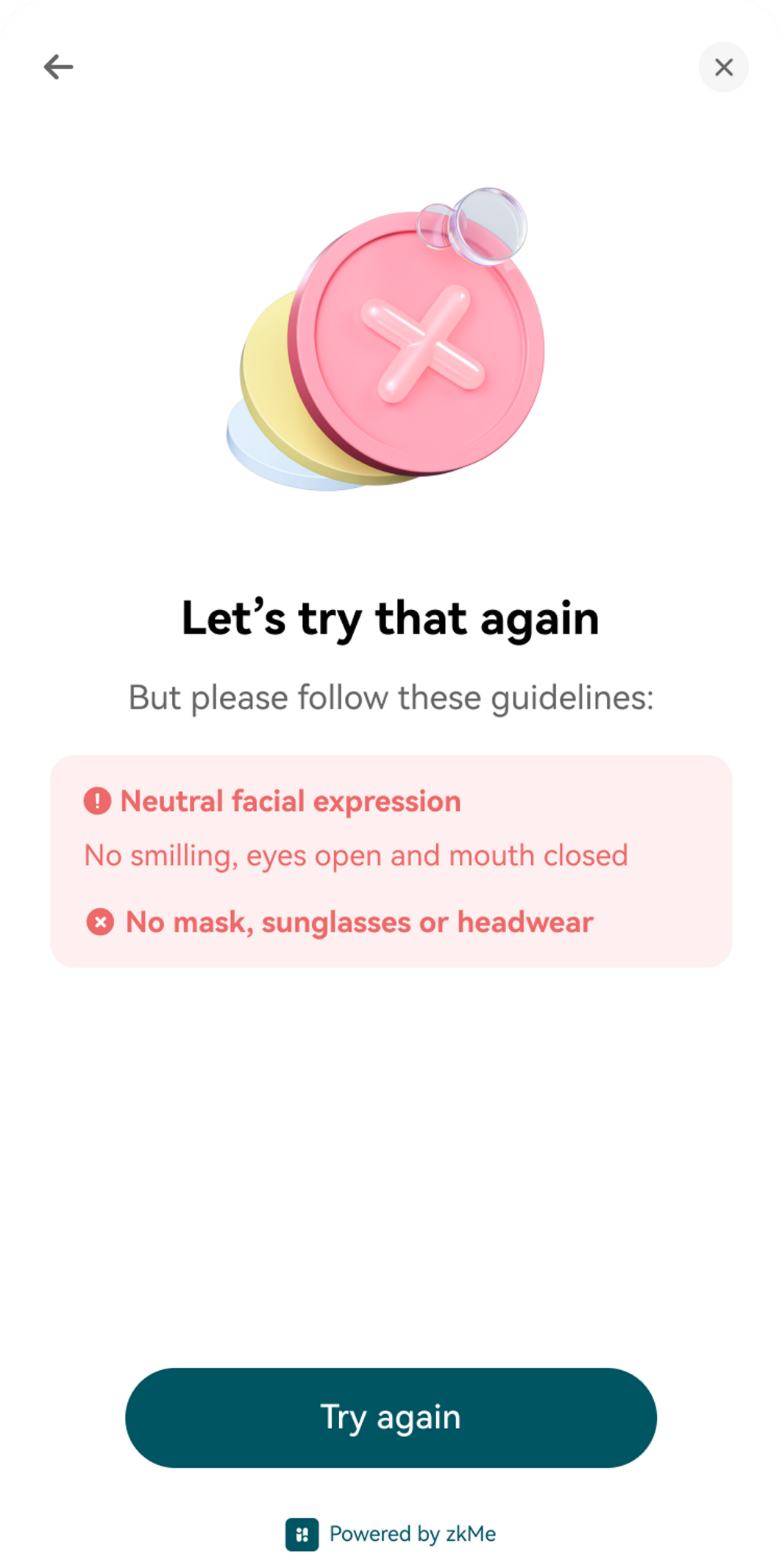
**Camera Permission Denied Error**



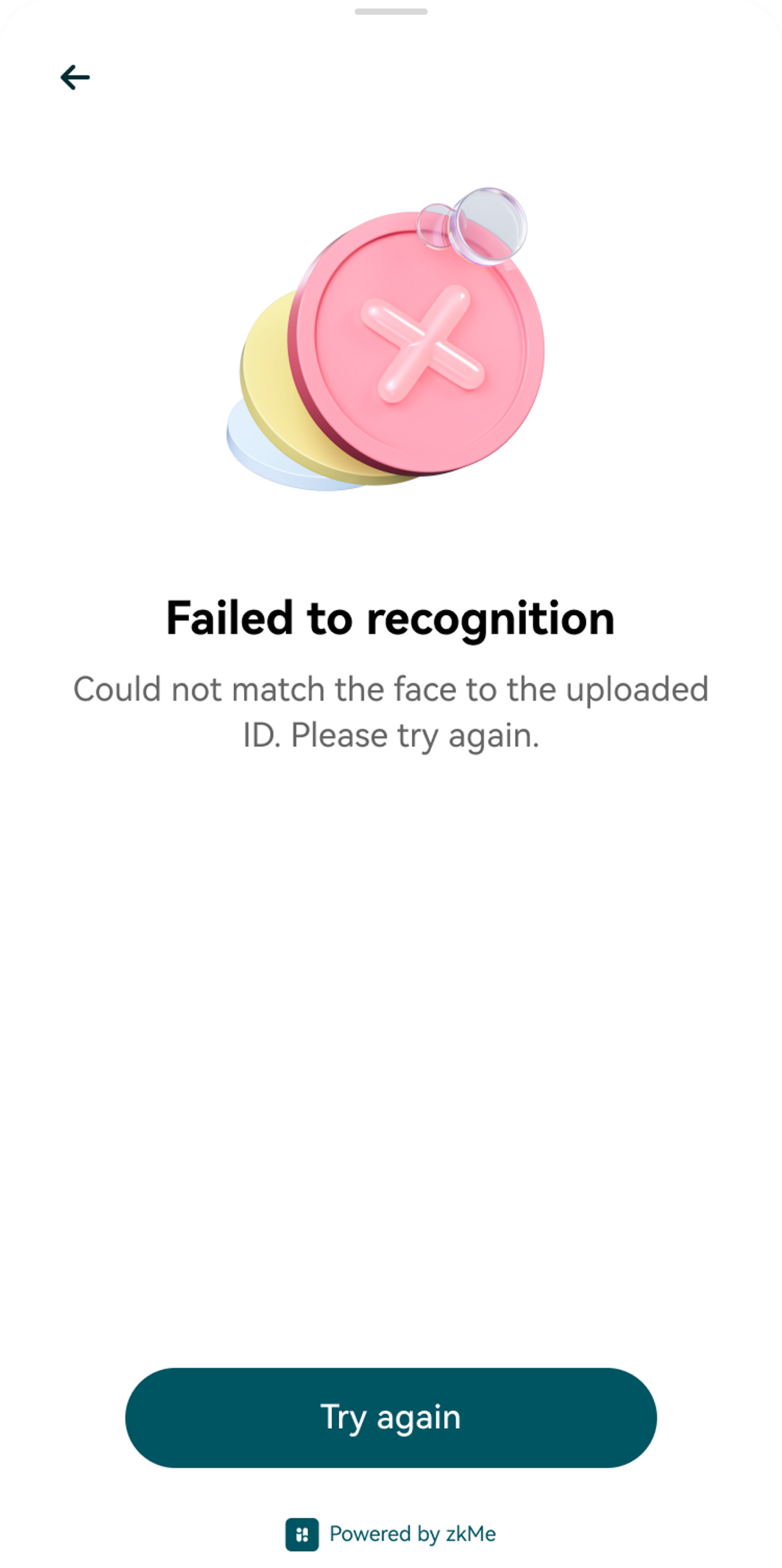
**OCR Scan Error**



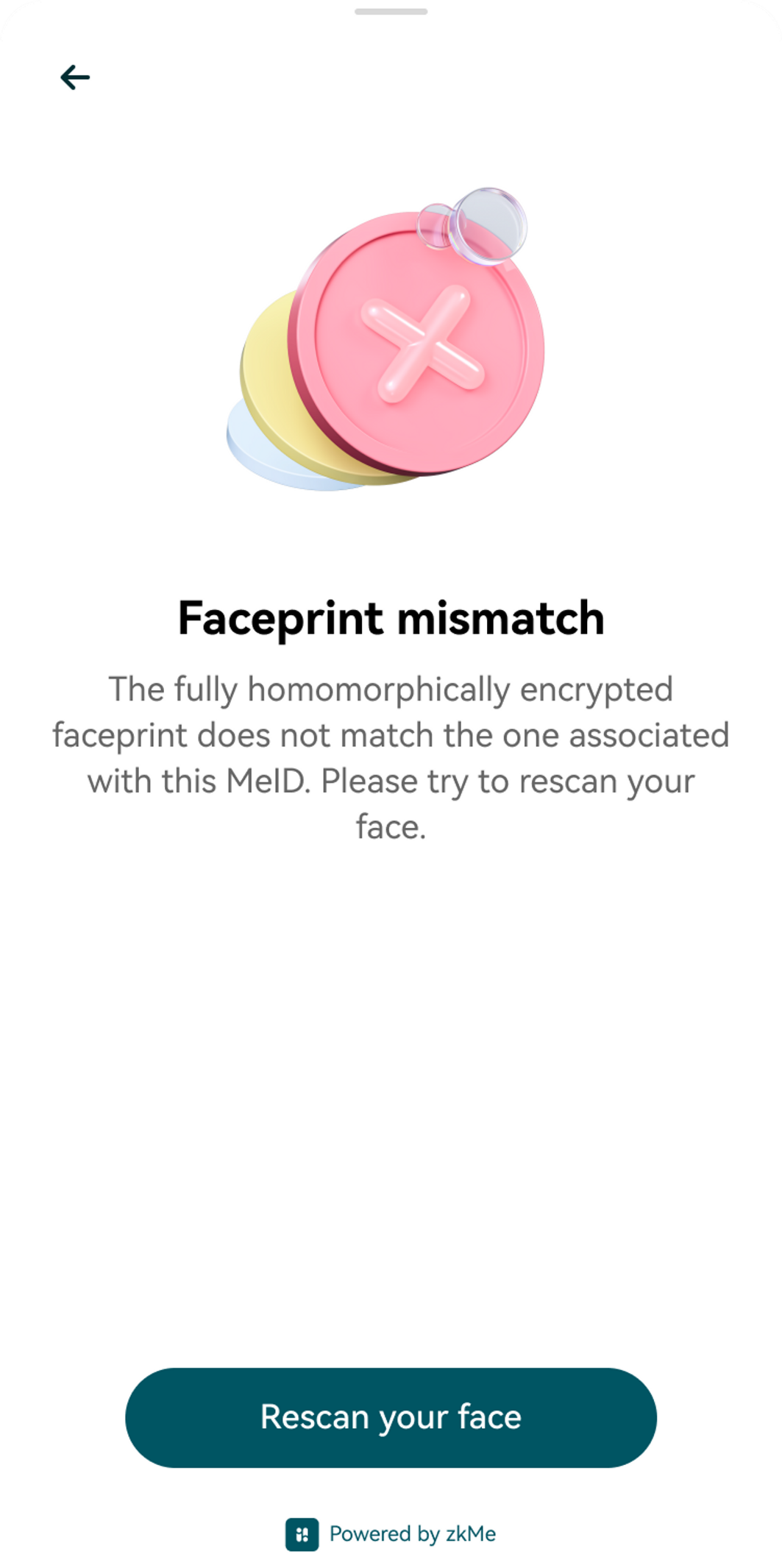
**Face Recognition Error**



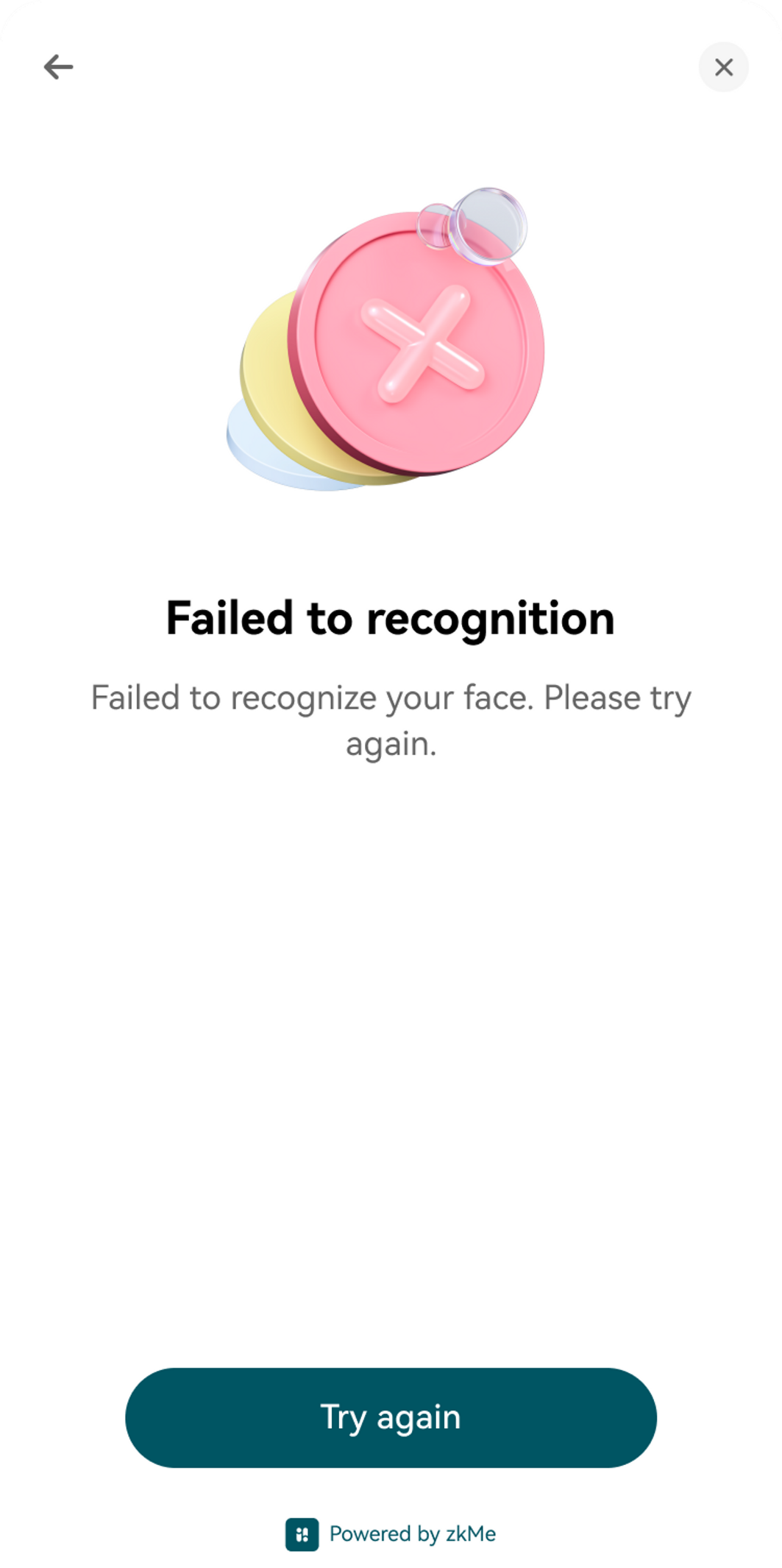
**Face Mismatch Error**



**Faceprint Mismatch Error**



**Faceprint Recognition Server Error**



**Unknown Error**

**Best Practices**

Best Practices for integrating zkMe SDKs

**Tips**

When integrating with a zkMe SDK, it is important to follow these tips:

* Ensure that you are using the **latest version** of the SDK to take advantage of new features and bug fixes.
* Follow the **integration instructions** provided in the SDK documentation carefully.
* **Test** your integration thoroughly before deploying it to production.
* Optimize performance by following the SDK's **recommendations and guidelines**.

**Github Example Code**

​<https://github.com/zkMeLabs/widget-demo>