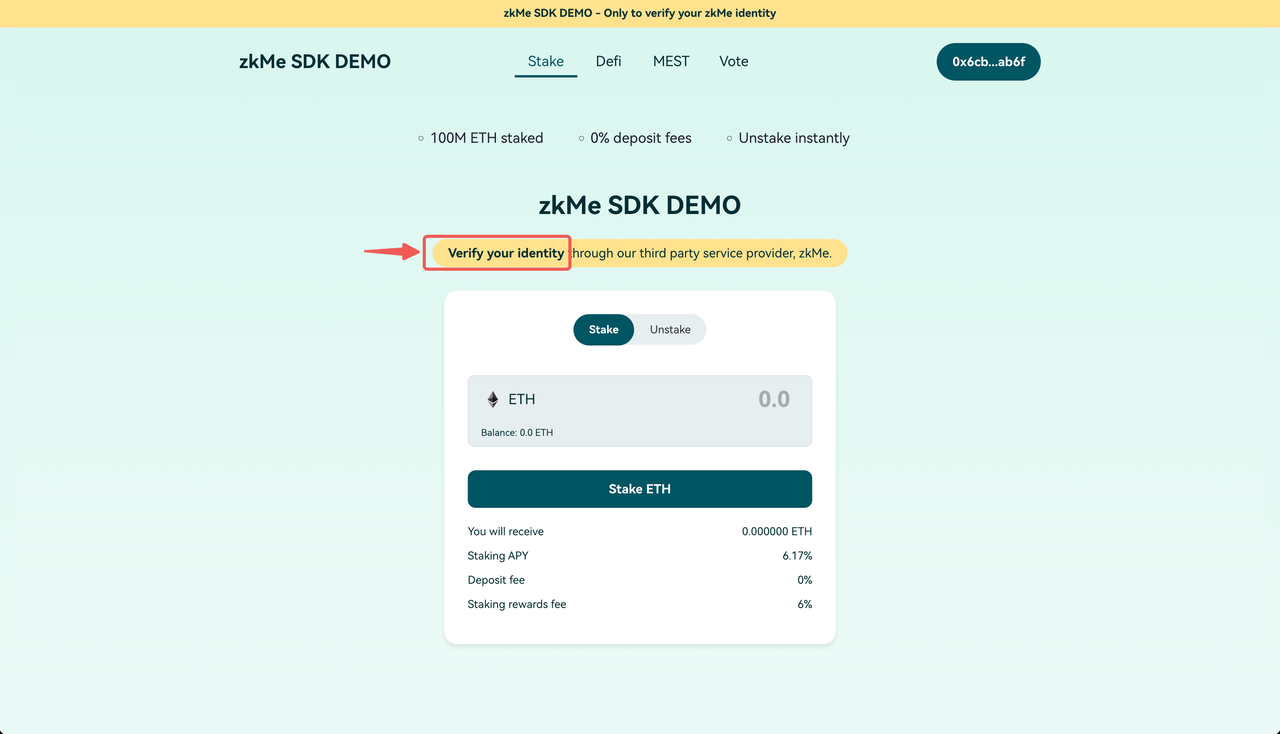
**zkMe Widget**

**Mint your zkMe SBT and Authorize a dapp**

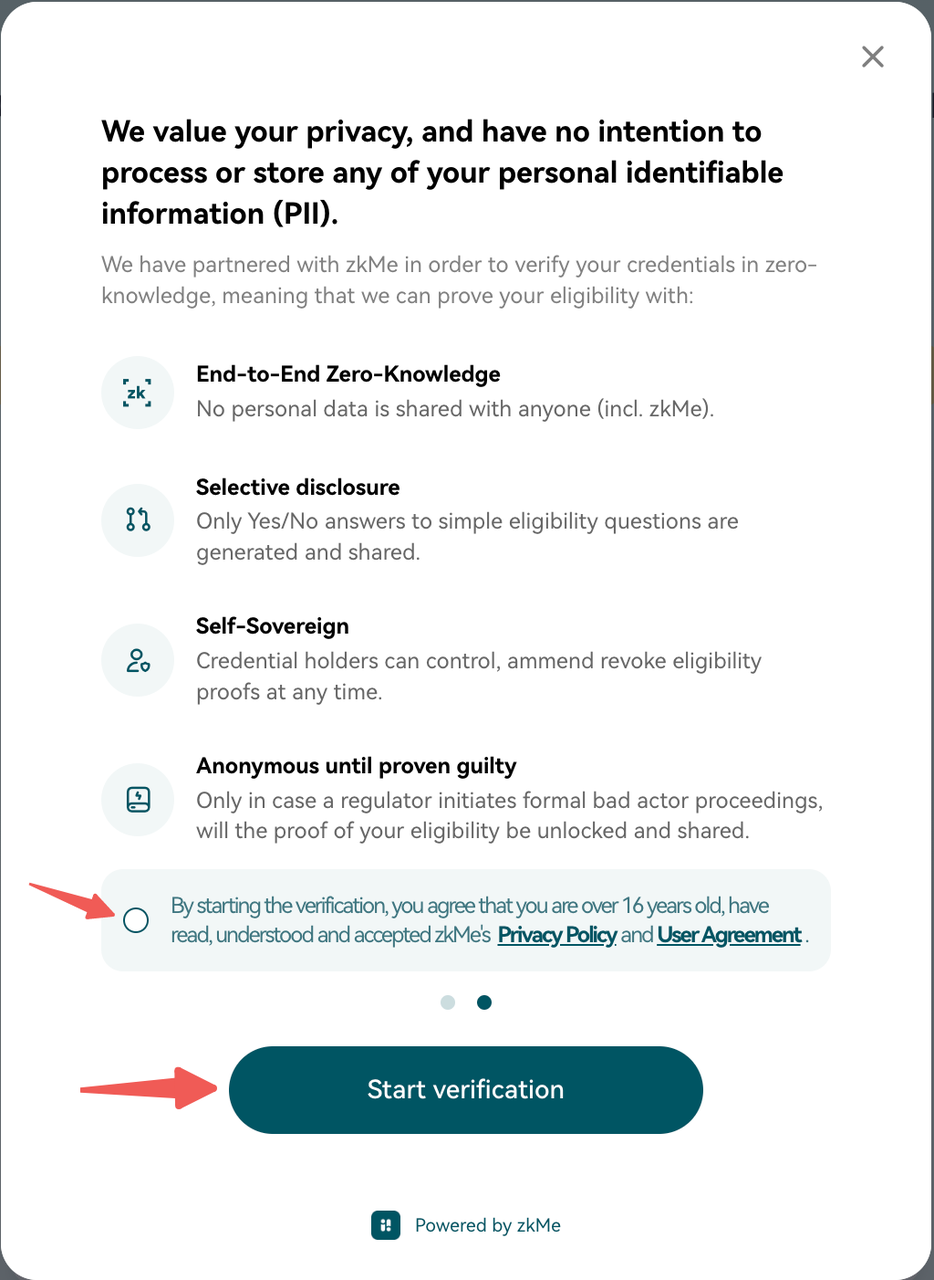
When you are participating in campaigns initiated by the dapps that require verification of your zkMe identity, you need to authorize with your zkMe SBT through zkMe widget to complete the identity verification. In order to make the authorization with zkMe SBT, you have to pass the proof-of-citizenship (PoC) verification to mint one. The verification process includes scanning your identity document and your face. To ensure personal privacy and security, your identification information will be encrypted, and we will not store your facial information.

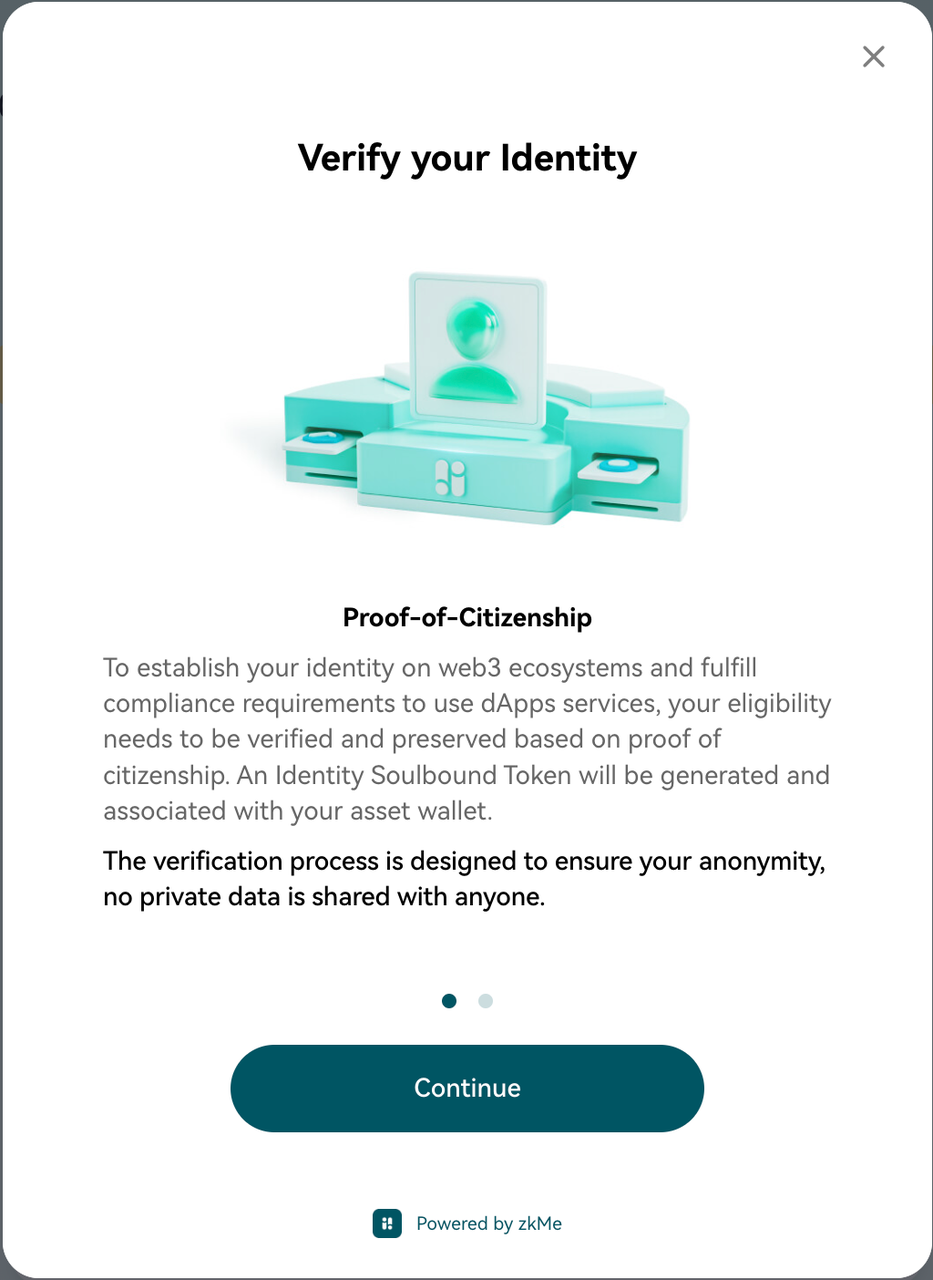
Connect your wallet to the dapp's website, then start the verification process and follow the steps below to finish the authorization.



**Step 1:**

Read the introduction and check the box at the bottom to make sure you agree with the privacy policy and user agreement. Then click the "**Start verification**" button to continue.

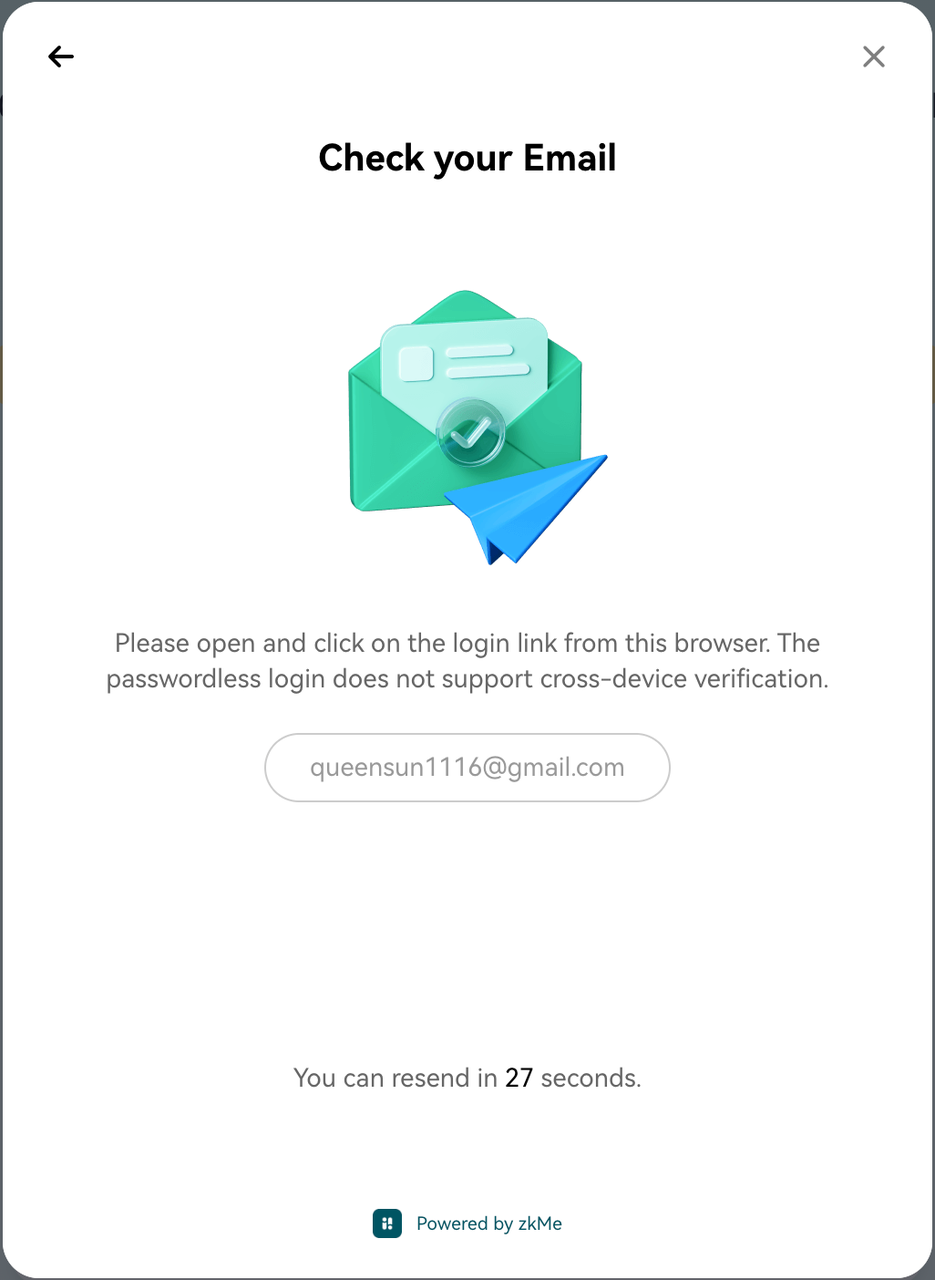




**Step 2:**

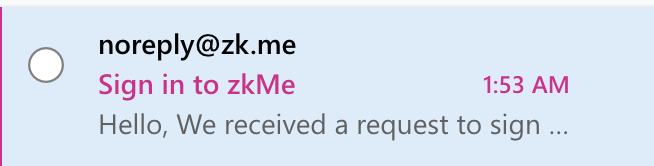
On this page, fill in your email address and click the "**Continue**" button. Make sure to provide a valid email address. Or you can also register through your **Apple ID** or **Google account** at the bottom of the page.





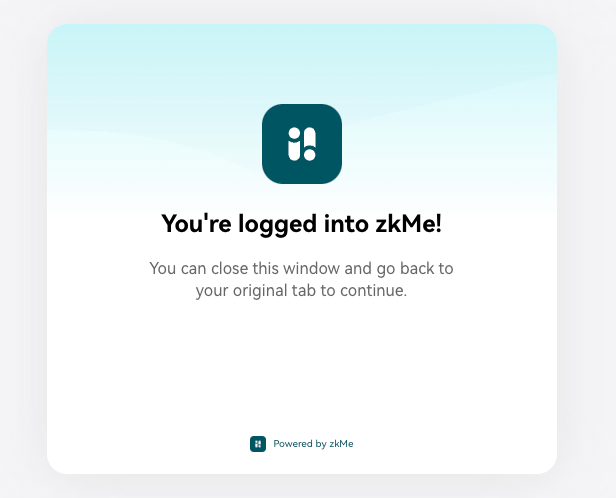
**Step 3:**

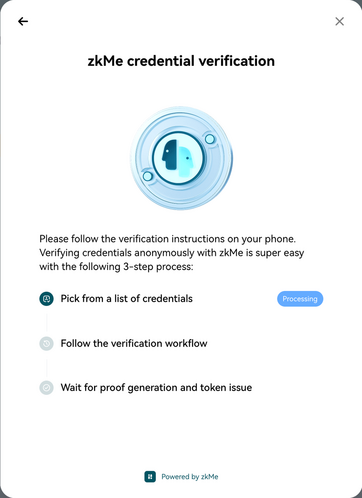
Check your email box and find an email named "**Sign in to zkMe**". Click the verification link in it and finish the process.

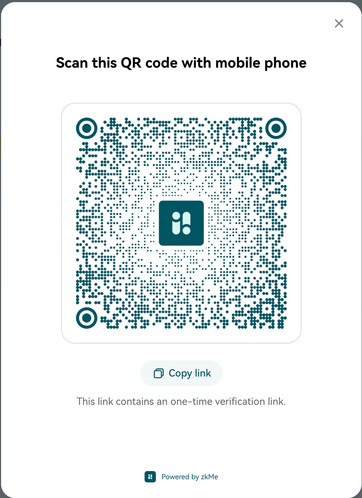


**Step 4:**

After your email has been verified successfully, go back to the zkMe widget and it will show a QR code. Scan the QR code with a mobile phone and continue the process on your phone.

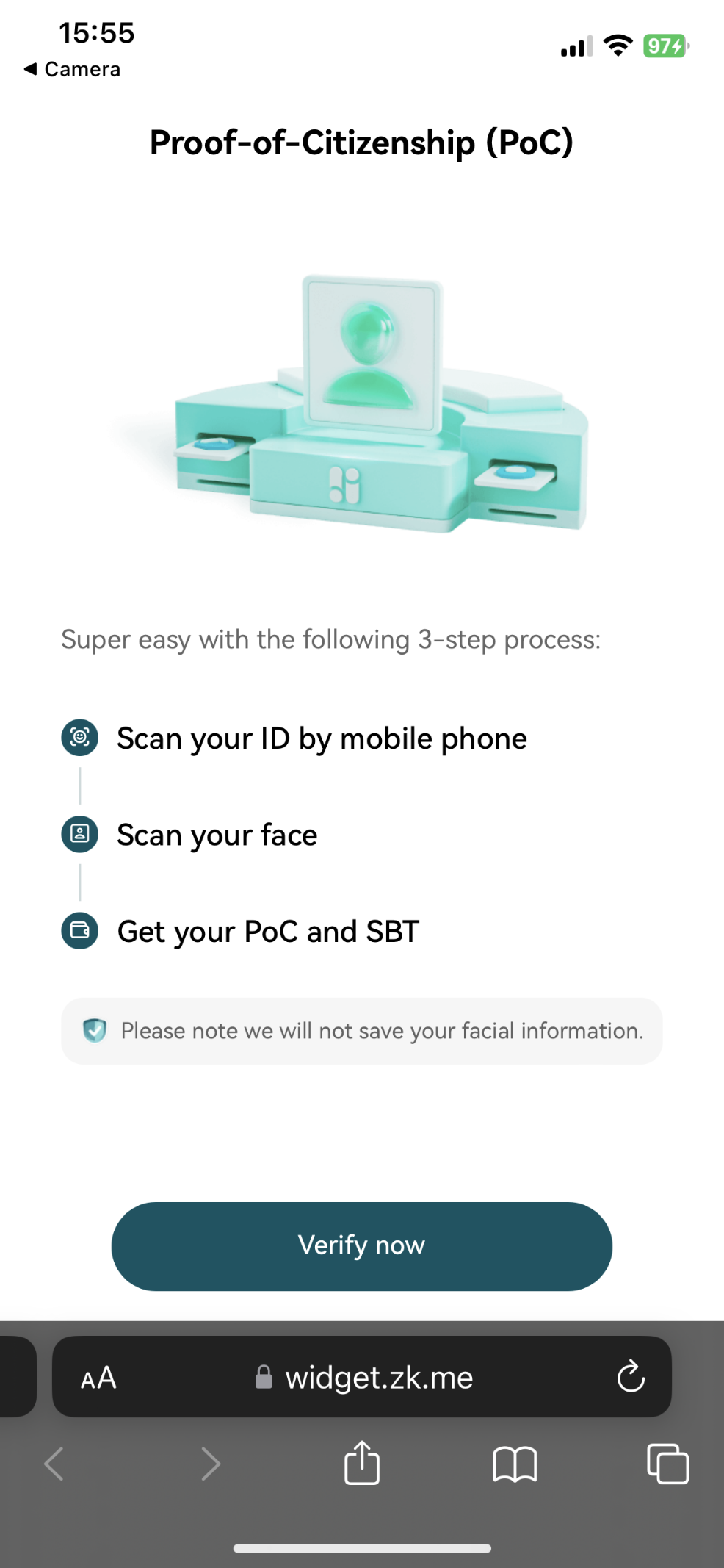






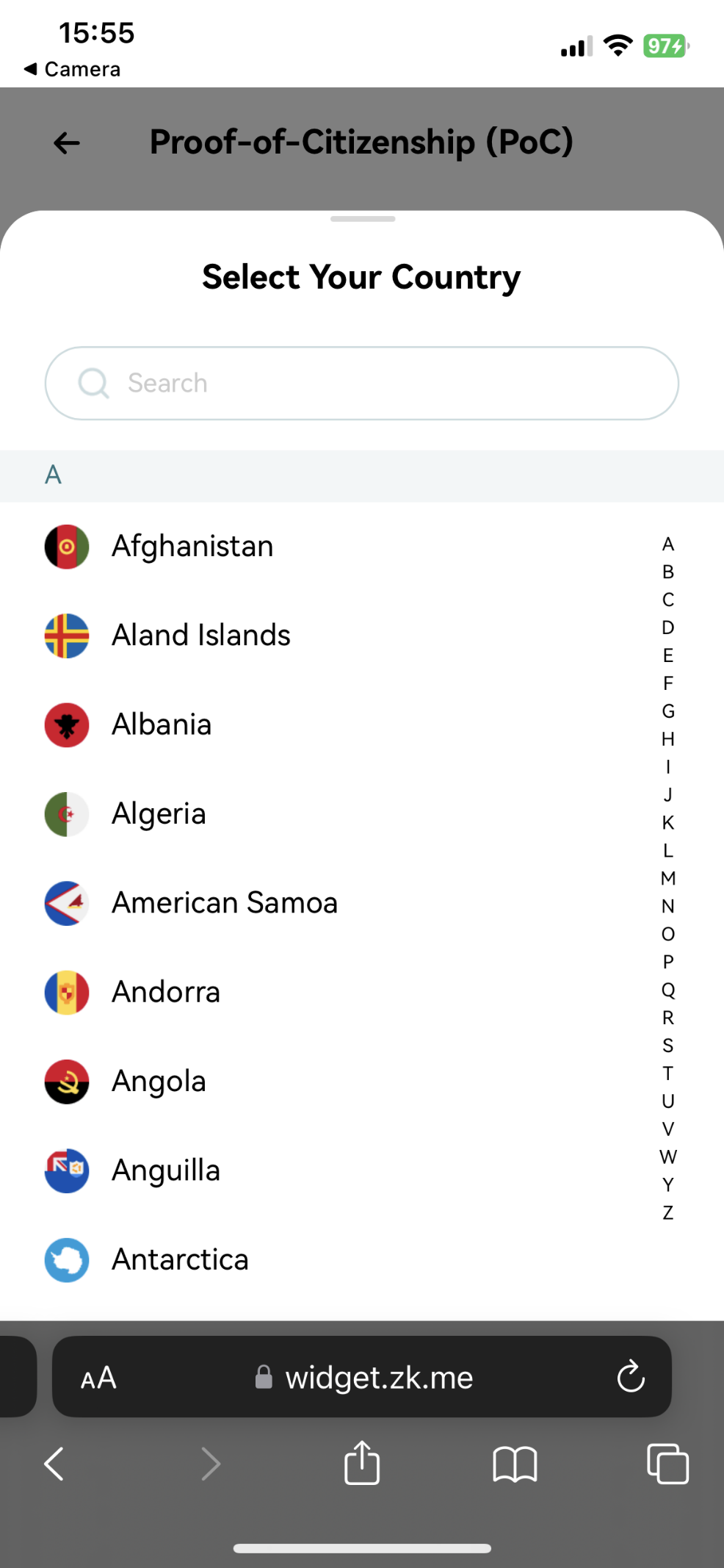
**Step 5:**

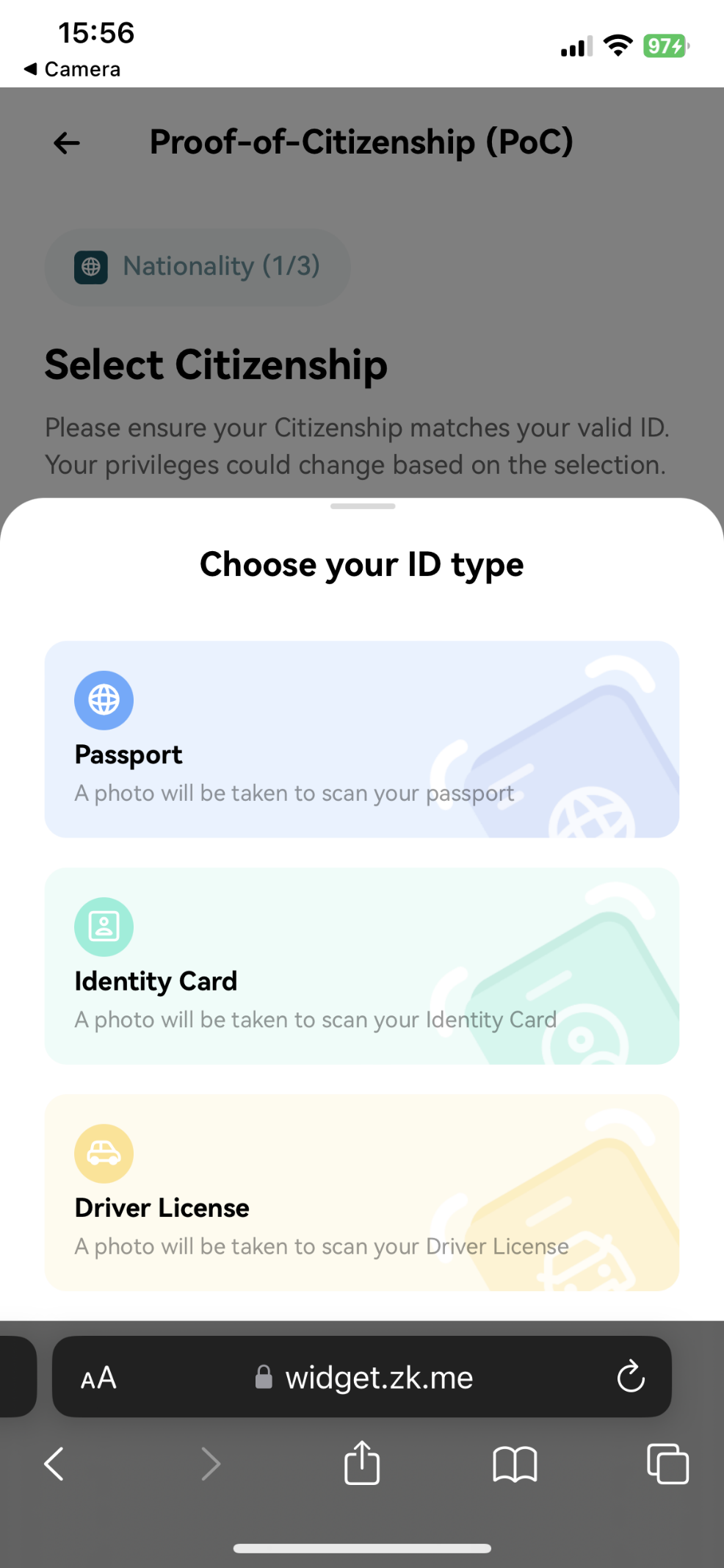
On your mobile phone, click the "**Verify now**" button on this page to start the verification process.



**Step 6:**

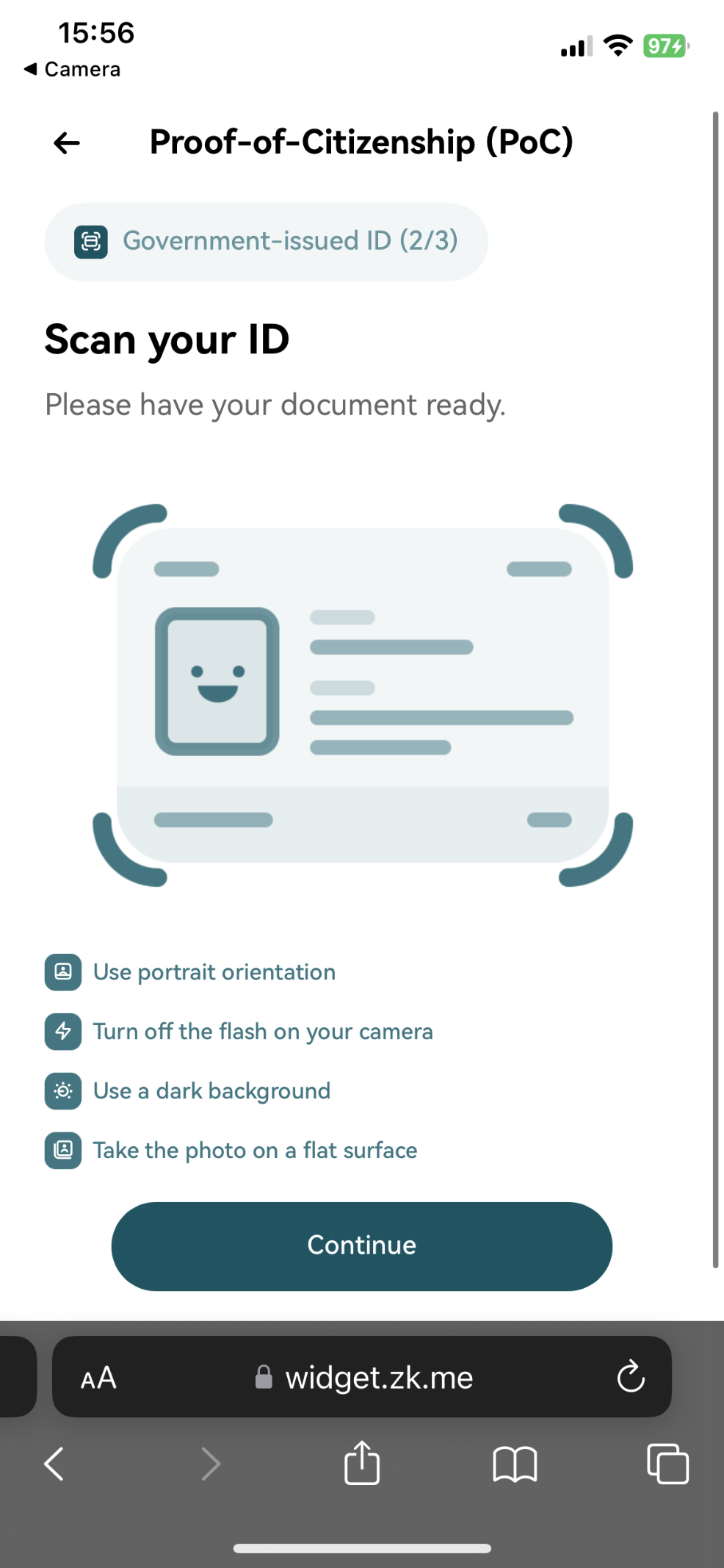
Select your country on this page, and choose an identity document type to continue.

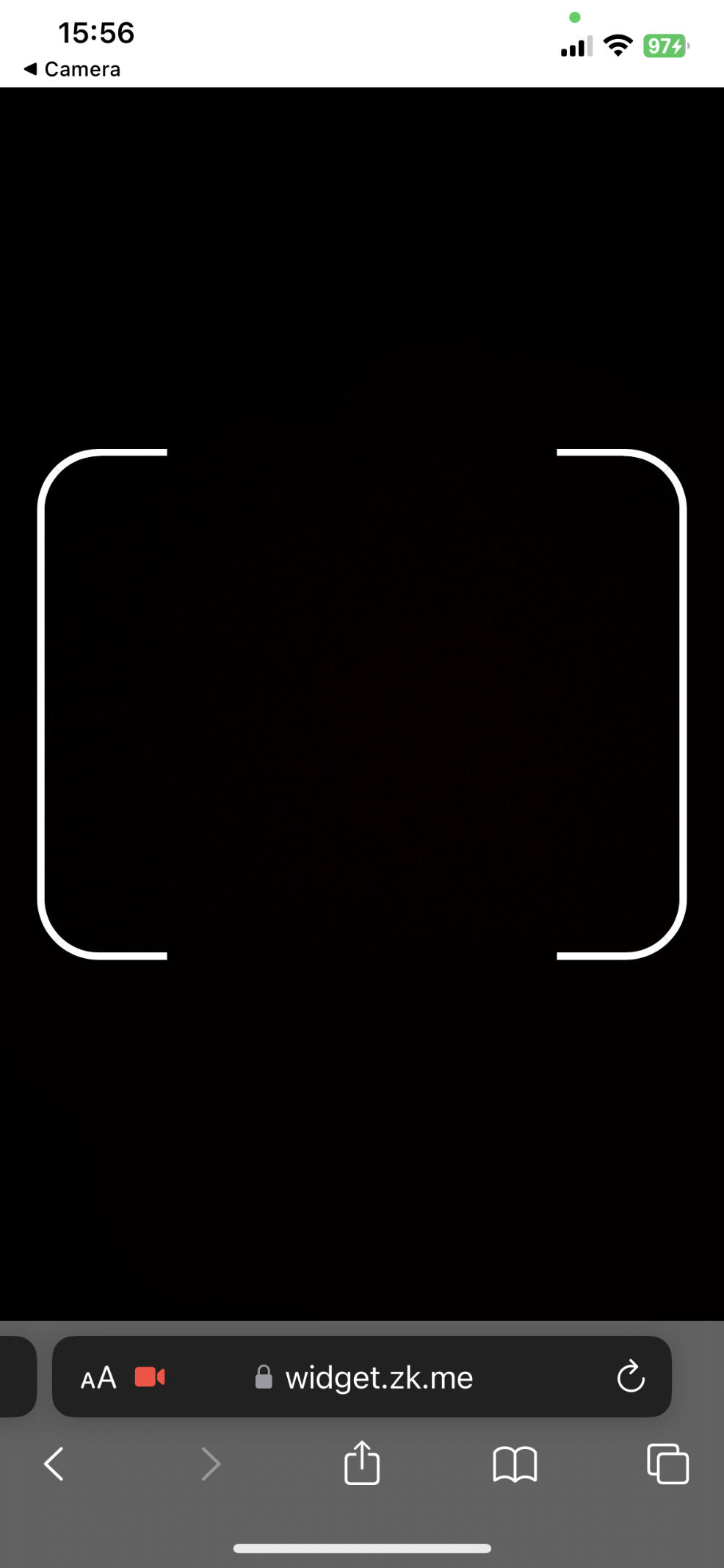




**Step 7:**

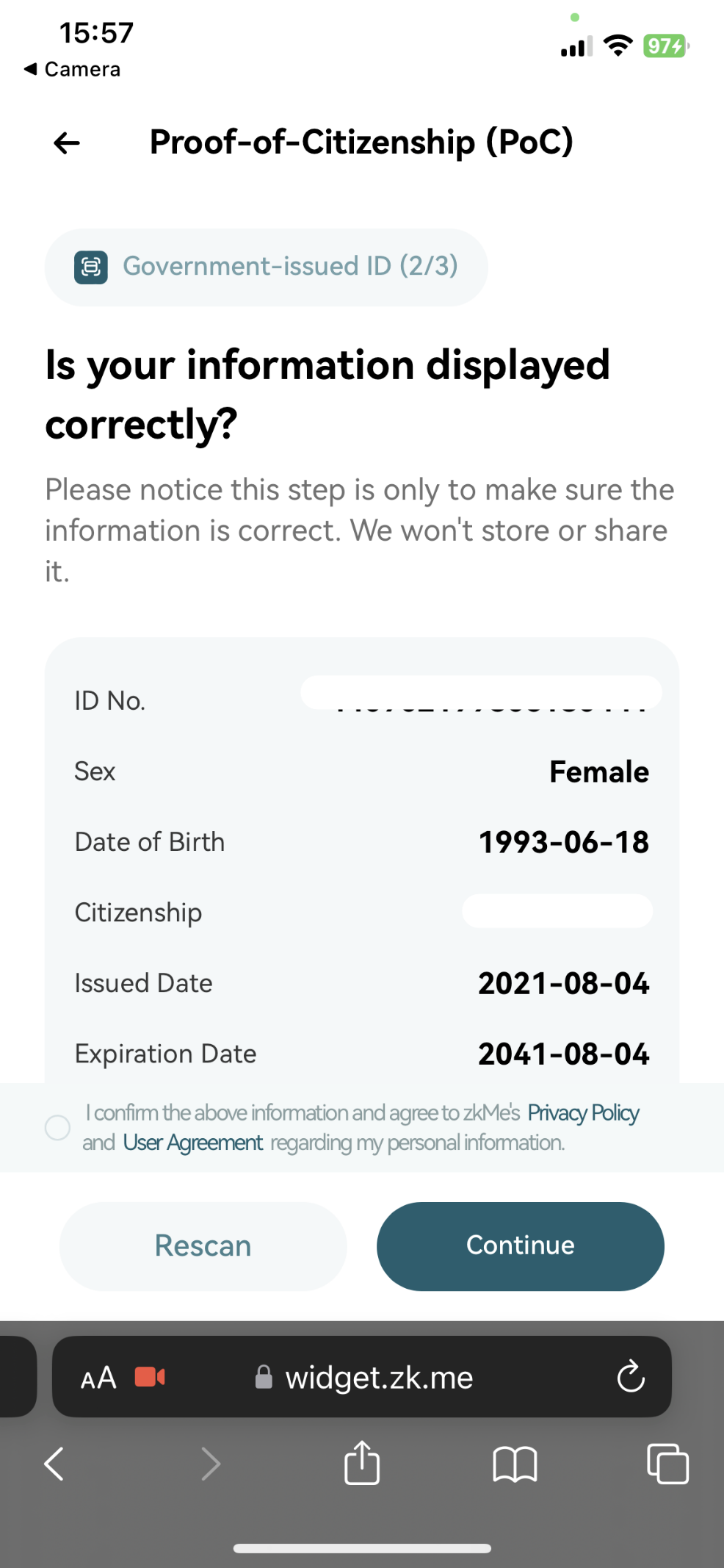
Click "**Continue**" on this page to start scanning your identity document.





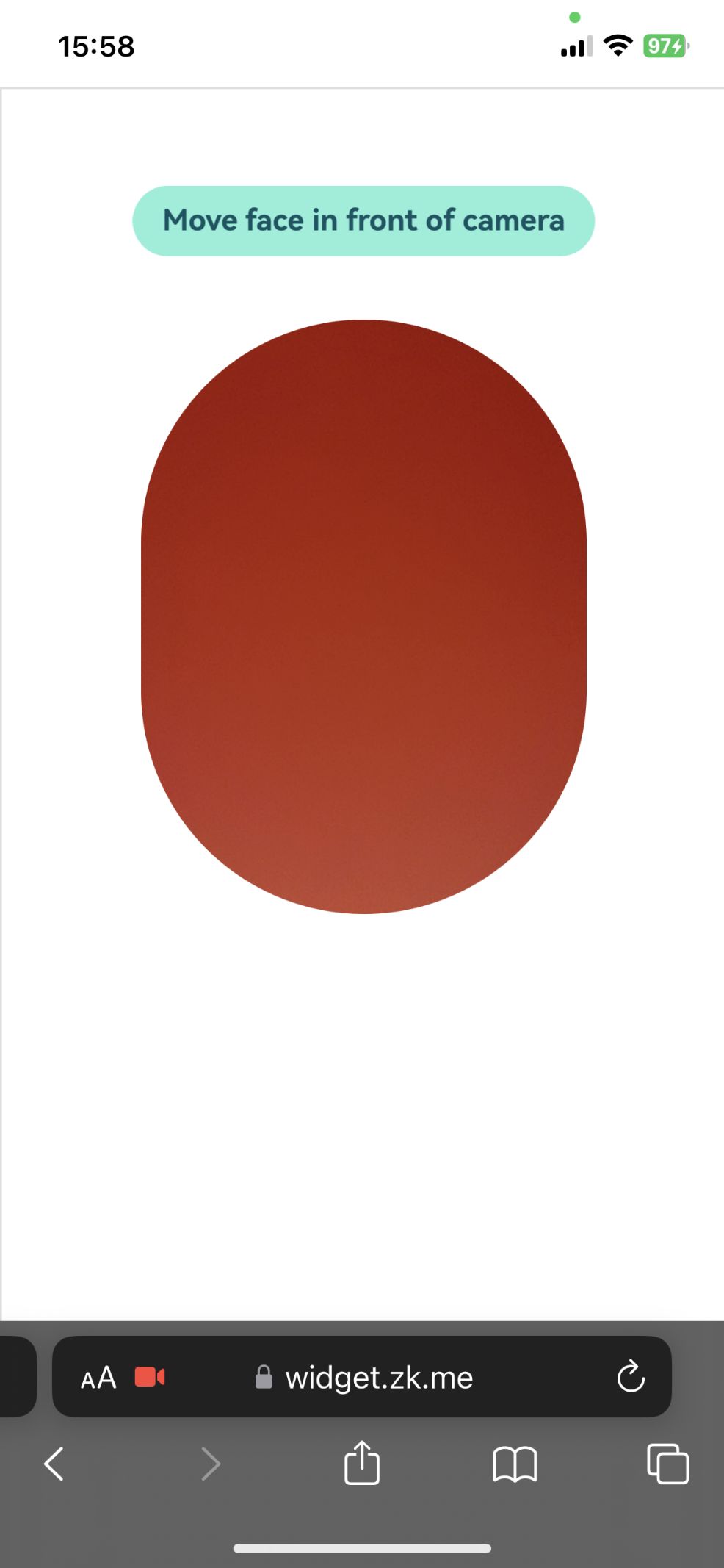
**Step 8:**

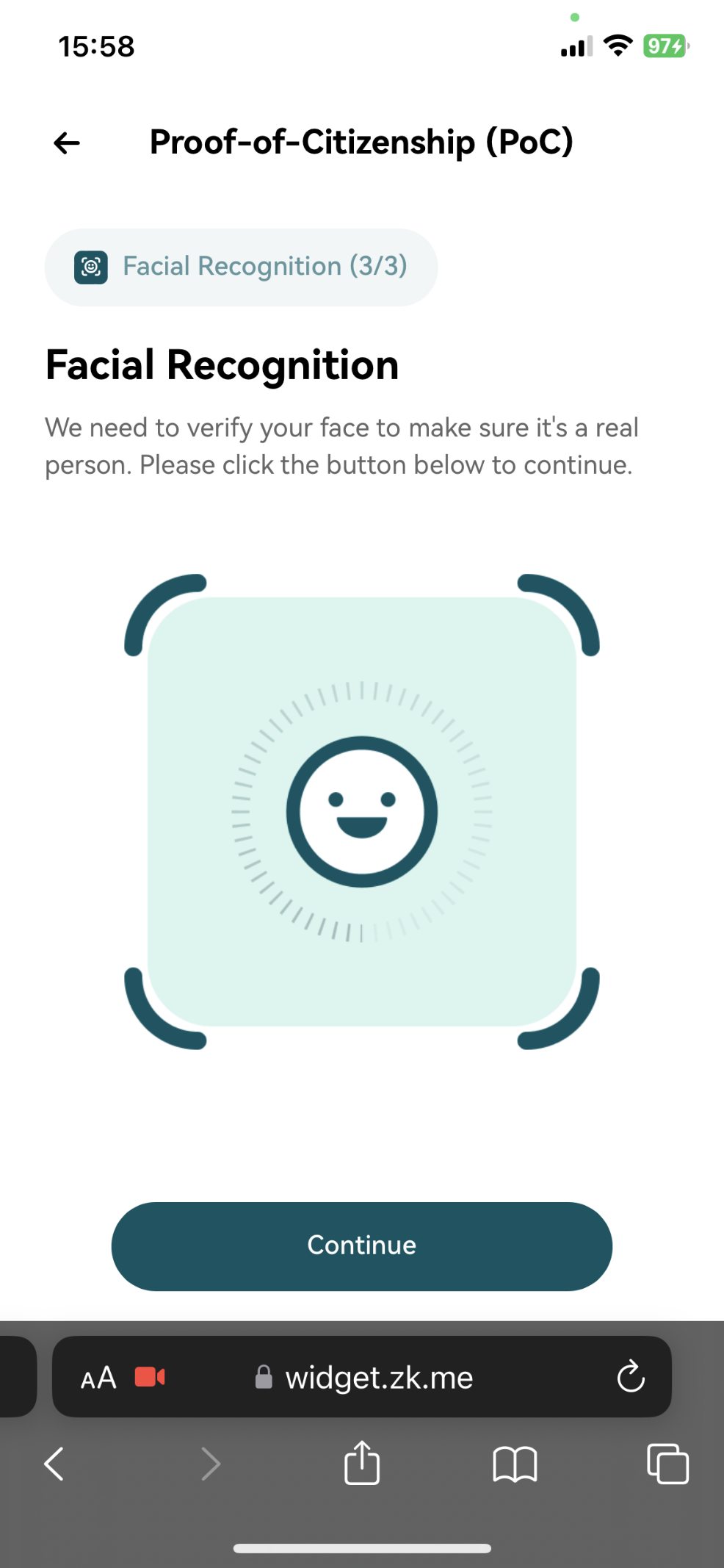
After successful document scanning, confirm that the identified information is correct on this page. Then check the box at the bottom to make sure you agree with the privacy policy and user agreement and click the "**Continue**" button to proceed with facial recognition.



**Step 9:**

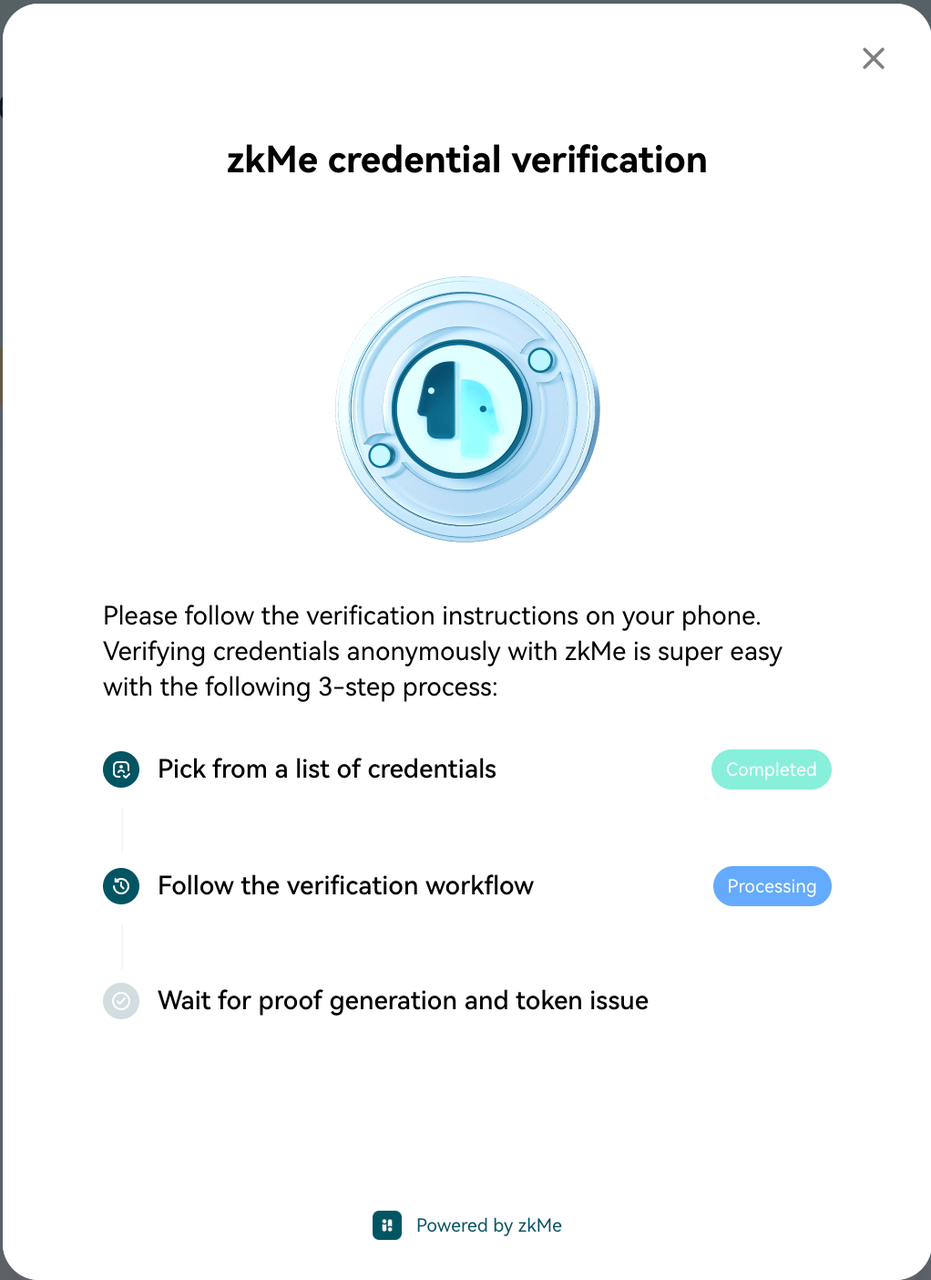
On this page, click the "**Continue**" button to scan your face.

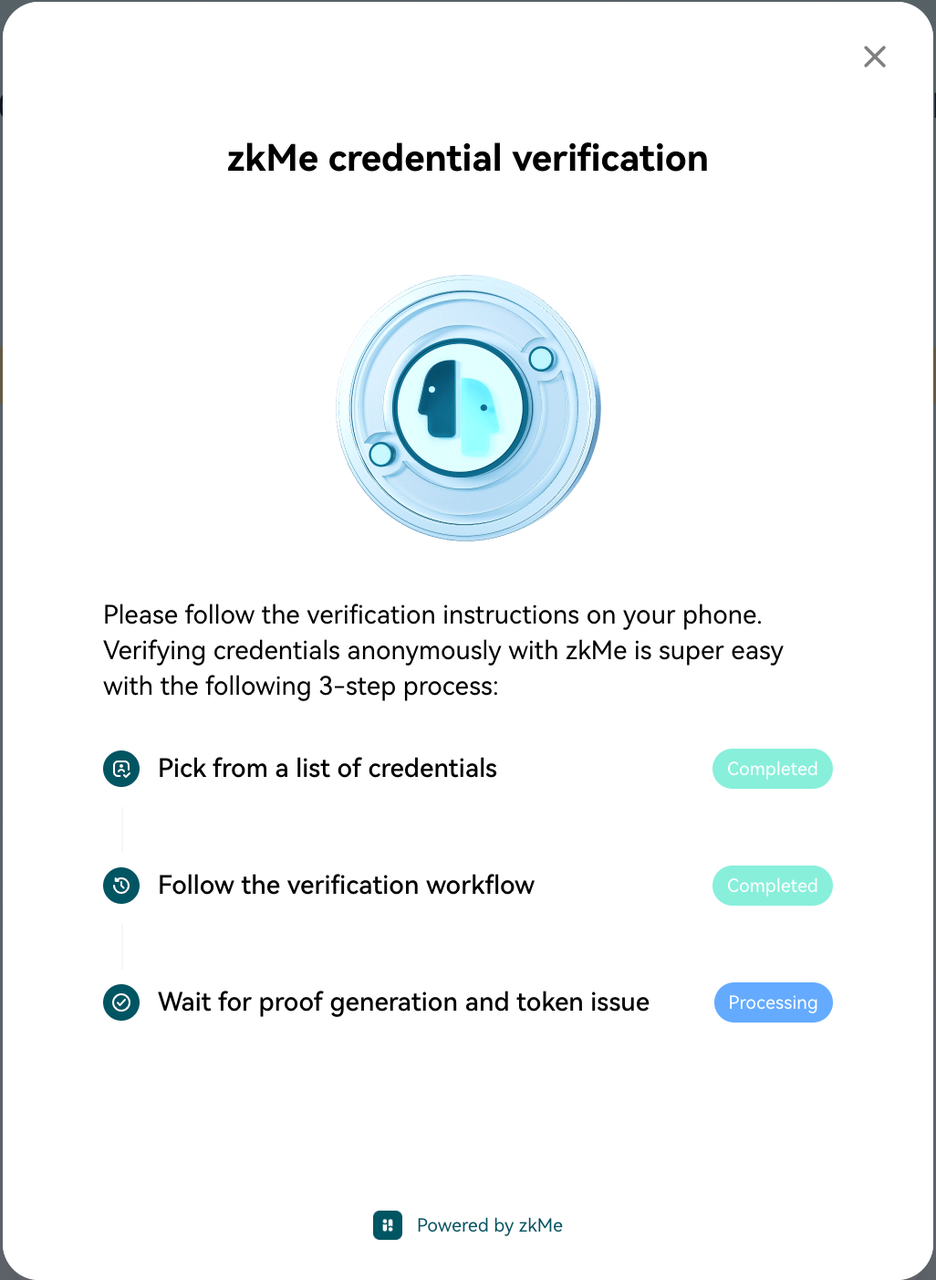


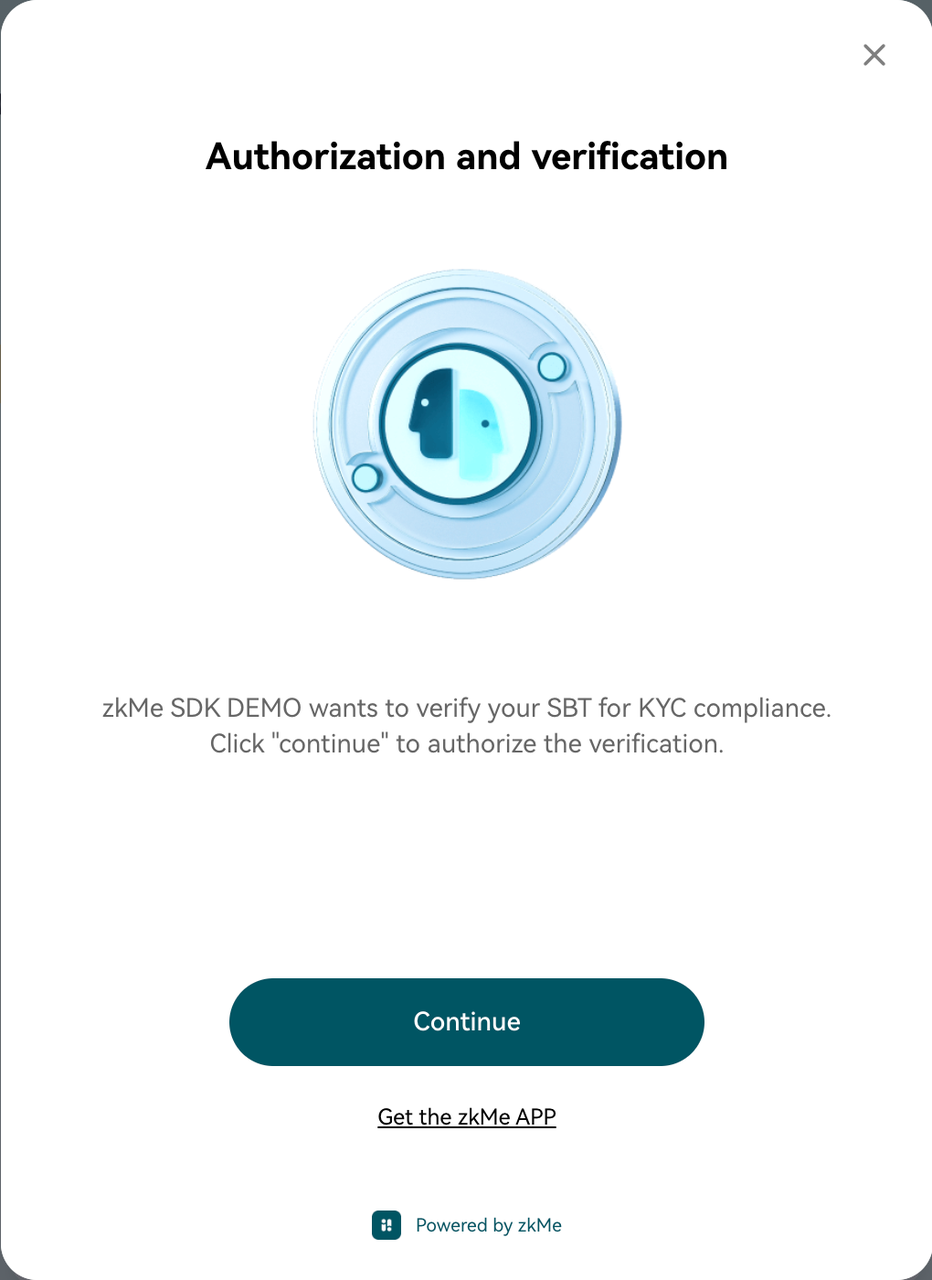


**Step 10:**

After successful facial recognition, it will automatically start the ZKP generation process. You can check the progress on the zkMe widget.After the ZKP is generated, it will automatically start minting your SBT. Once your SBT is minted, click the "**Coninue**" button on **Authorization and verification** page to authorize your zkMe SBT to the dapp.

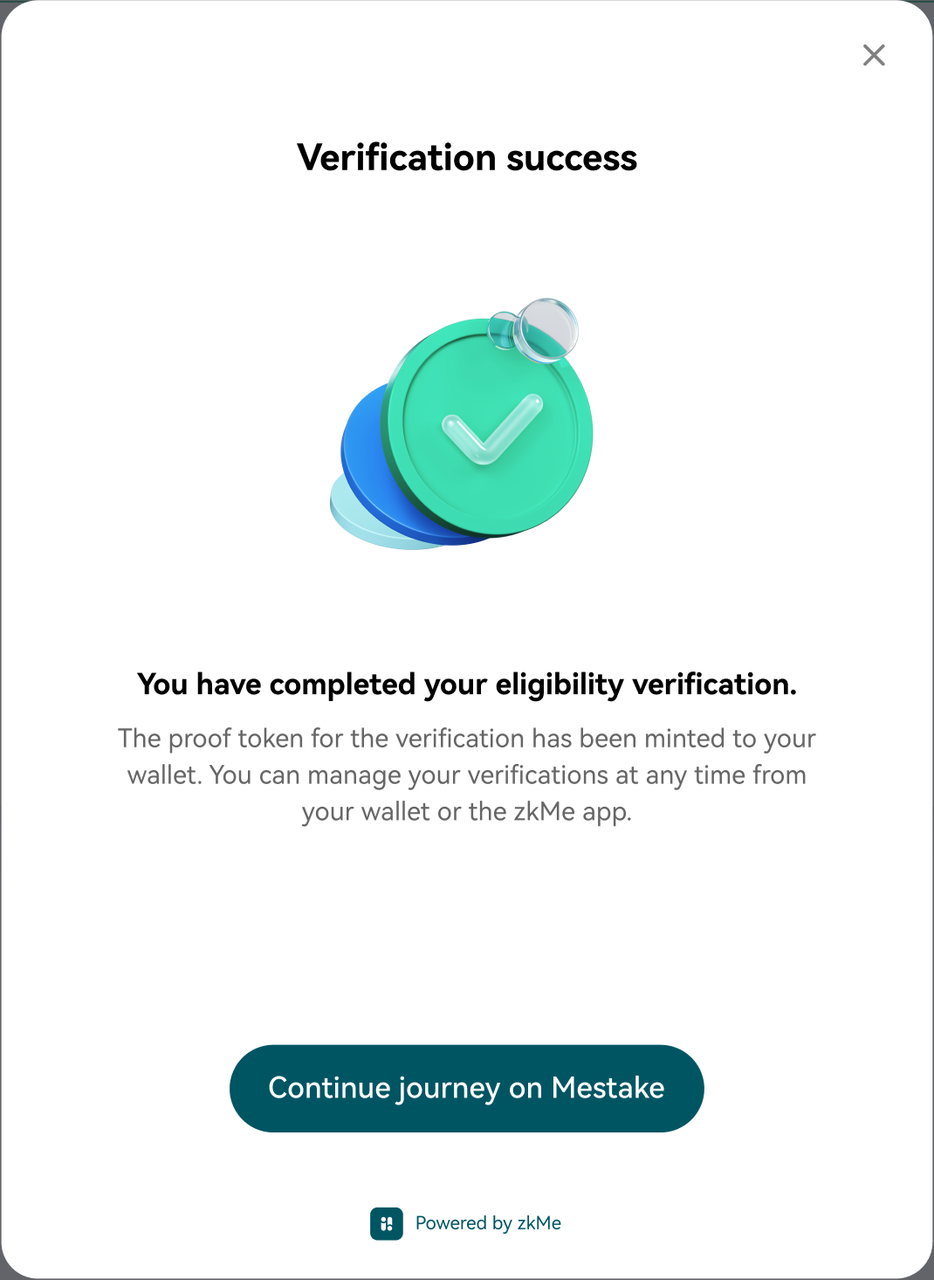


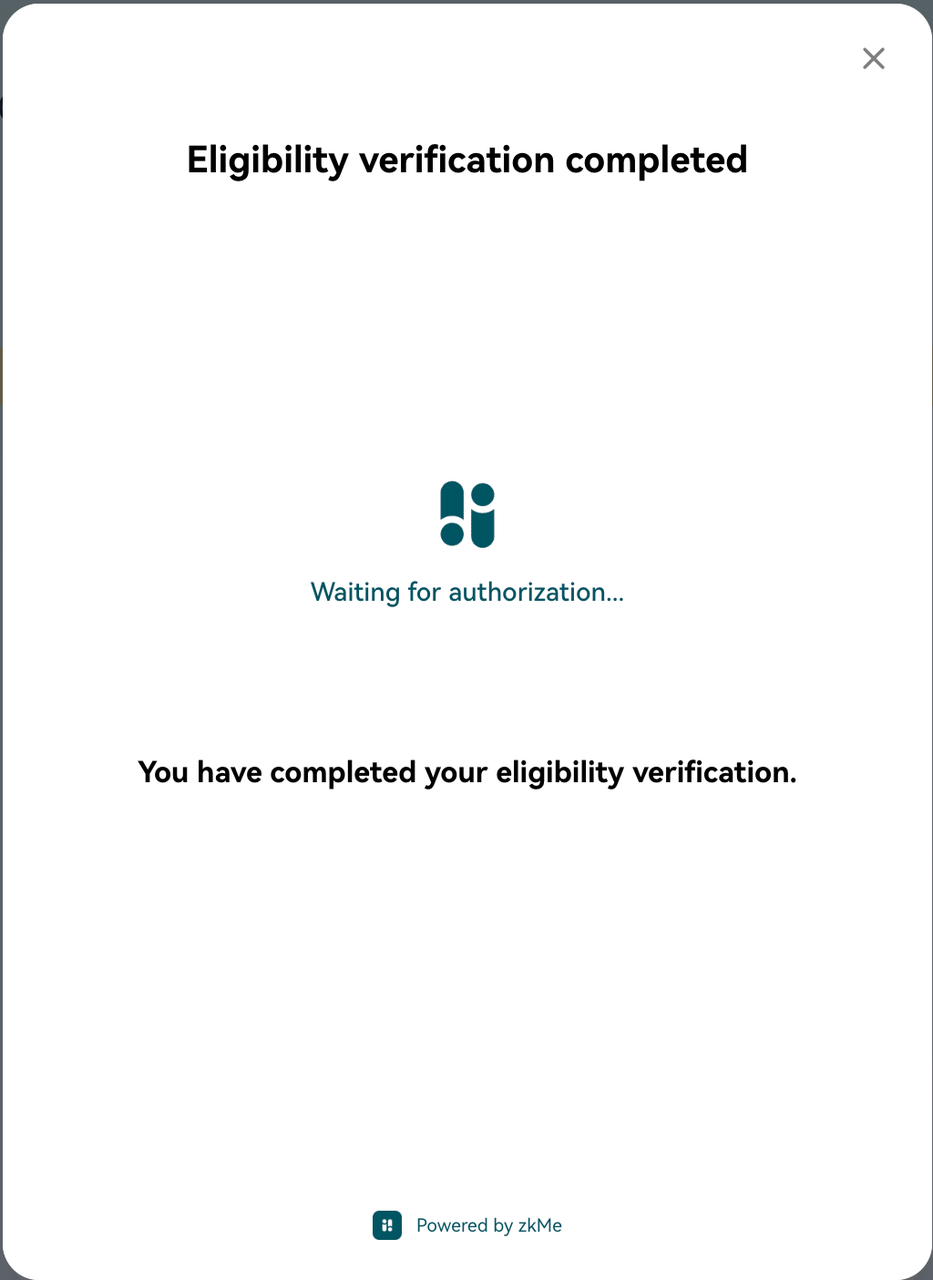




**Step 11:**

When this page is displayed, sign within the asset wallet plugin to complete the authorization operation.





**zkMe ZKP Infra**

**zk-SNARK**

zkMe uses Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK) to generate ZKPs that represent the Holder's identity information without disclosing any sensitive data as they run directly on the Holder's end device and only expose the anonymized ZKP results (yes/no statements). These ZKPs are then presented as VPs to the Verifier, who can verify the identity without accessing the underlying data. This approach enhances privacy and security by ensuring that only the necessary information is shared between the Holder and Verifier, and the sensitive data remains protected.

SNARK JS and Circom are the core tools for implementing zkMe technology. Snarkjs is a JavaScript library for zk-SNARKs that is used to generate and verify ZKPs, supporting multiple zero-knowledge proof systems such as Groth16 and PLONK. Circom, on the other hand, is a zk-SNARKs circuit DSL (domain-specific language) used to write circuit constraints. Developers can write circuit constraints in a more readable code using circom and then use snarkjs to generate and verify ZKPs.

In zkMe, the Groth16 algorithm is used, which requires a trusted setup to generate a parameter set known as a common reference string (CRS) for each statement (also known as circuit). Once this CRS is generated, it can be used for proving different statement instances throughout the system's entire lifetime. The Groth16 setup is performed through a multi-party computation setup ceremony that ensures the security of the ZKPs system as long as at least one party is honest.

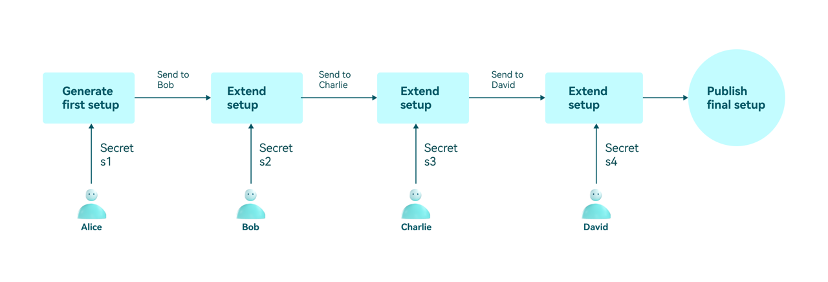
**zkMe's trusted setup**

In zk-SNARK systems, trust is established through the use of cryptographic techniques that ensure the integrity and security of the process. There are two phases for zkMe's trusted setup.

Phase 1 is the Perpetual Powers of Tau Ceremony that can be forever on-going and used by all circuits. zkMe uses the publicly available Ptau parameters from Snarkjs as part of its zk-SNARK implementation. These parameters will form the base for Phase 2 and are generated through a process known as the "toxic waste" process, which involves using a random number generator to generate some initial parameters, and then performing a series of computations to generate the parameters for zk-SNARK.

Phase 2 is the Circuit/Statement-specific Setup Ceremony that needs to be done for each circuit in zkMe. A circuit is a representation of the computation to be performed in zk-SNARK. Before zkMe goes into production, zkMe will customize and publish all circuits in protocol to ensure that circuits correctly represent computation and do not contain any vulnerabilities that could be exploited by an attacker. Then, based on circuit files and the publicly available Ptau parameters in Phase 1, zkMe generates the initial keys to start Phase 2 and contributes it with some randomness. Phase 2 can continue for as long as there are participants willing to contribute with their resource of entropy. And we will open the entry to welcome every potential participant to contribute to the latest keys. Once enough participants have contributed, we will terminate this relay process and publish the protocol keys so that the protocol can be used in production after that. To learn about the overall progress, please visit our website.

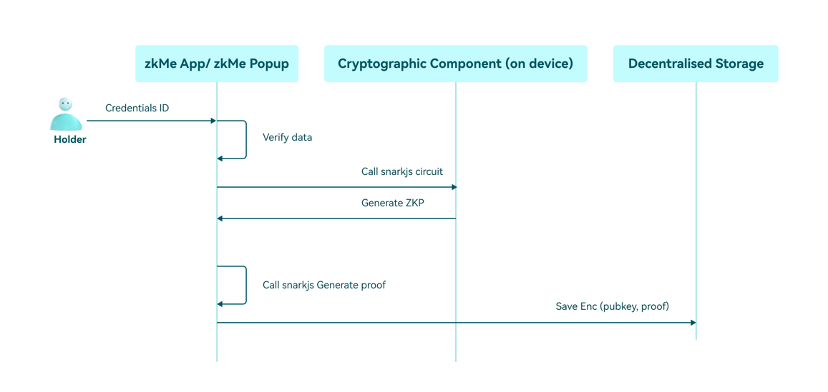
In summary, each computation verification problem in the zkMe is converted into an R1CS/circuit file, corresponding to a pair of proving/verification keys.



Multiple participants contribute with secrets in relay

**Proving and verification**

zkMe uses the Groth16 algorithm as its underlying zk-SNARK construction. The Groth16 algorithm is a variant of the pairing-based SNARK construction, which allows for efficient verification of complex computations in zero-knowledge. It uses a structured reference string to generate zk-SNARK proofs. The structured reference string consists of a set of public parameters that are used to represent the computation being performed, and it has been generated in 5.4.1 Trusted Setup Ceremony.



zkMe ZKP generation sequence diagram

The proving and verification process of the Groth16 algorithm through the following steps:

**Proving process:**

* The prover queries the circuit file and proving key corresponding to statement.
* The prover calculates the intermediate and output signals of this circuit as witness based on its own inputs. Note that the witness is as important as private input and should not be disclosed to anyone.
* The prover calculates the proof based on witness and proving key.
* Submit public inputs and proof to the Verifier.

**Verification process:**

* The Verifier receives the proof and verification key of CRS.
* With the proof and verification key known, the checking equation can be verified to hold by the pairing function.
* If the checking equation holds, the proof is valid; otherwise, it is invalid.

**zk-SNARK example**

Let's assume a scenario where Alice wants to go to a bar for a drink and the bar staff needs to verify that Alice is over 18 years old. In the interests of privacy, Alice does not want to reveal her real age. This is a good scenario for a real application of ZKPs, where Alice can prove her age meets the established requirements of zk-SNARKs.First, before that, Alice already had a verifiable credential (VC) issued by the federal government regarding the time of birth. It is stored in blockchain and is directly linked to the federal government's public key. Essentially, the VC is a hash value and is calculated according to the following rules.



where

pkA is Alice's public key in her digital wallet, birthdayGMTtsA is the GMT timestamp of Alice's birth, and r is a random salt value. The latter two are stored in Alice's digital wallet.First, before Alice generates the proof, the staff will calculate the interval between today's timestamp and the timestamp of the day 18 years ago, e.g., today is 2022-10-29 00:00:00, the timestamp is 1666972800, and the timestamp of 2004-10-29 00:00:00 eighteen years ago is 1098979200. The interval between the two timestamps is 567993600, which is denoted as

intervalThreshold. Alice then generates a proof in her wallet that she is indeed older than 18 years old in the following steps: Alice defines (skA,birthdayGMTtsA,r) as the private input and defines (pkA,VCA,age,todayGMTtsA,intervalThreshold) as the public input to generate the proof as follows:

​

The proof has the following steps：

* check that

(skA,pkA)

is a correct key pair

* check that

​

* check that

todayGMTtsA−birthdayGMTtsA>intervalThreshold

This proof will be used to call the validation contract specified by the staff, which will return the validated boolean value.

Lastly, since both the and its signature signed by the federal government are stored in Smart Contact, it could be checked that this verifiable credential is indeed issued by a specific authority. Note that the contract will generate a nullifier based on the proof that will be verified to prevent this proof from being replayed.