Bring Mass Adoption For the Polkadot Ecosystem —— zkLogin

Eliminating reliance on passwords reduces the risks of phishing, brute force attacks, and replay attacks, providing a simpler and faster login process for both Web2 & Web3 Users.

TABLE OF CONTENTS

01. **Status Quo**
Elaborate on the background of the project

02. **What is zkLogin**
Breif explanation of zkLogin

03. **Main value**
How can zkLogin benefit Polkadot Ecosystem?

04. **Details Of zkLogin Solution**
In-depth explanation of zkLogin

STATUS QUO

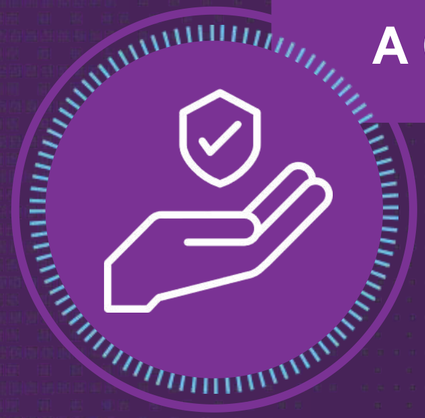
Polkadot's Account Abstraction (AA) is an innovative concept introduced to address the challenges associated with managing cryptographic keys on a blockchain.

For traditional EOA, The loss of private keys poses significant risks, leading to the inaccessibility of digital assets and fragmentation of digital identity.

Initially mentioned in Ethereum's EIP-4337, the concept of Account Abstraction aims to provide users with flexibility in programming enhanced security and improved user experiences into their accounts.

This separation of user experience from private keys enables code to decide account behavior, increasing the flexibility of initially rigid accounts and reducing the chances of key mismanagement.

STATUS QUO



A Common Goal: Achieving Mass Adoption from Web2

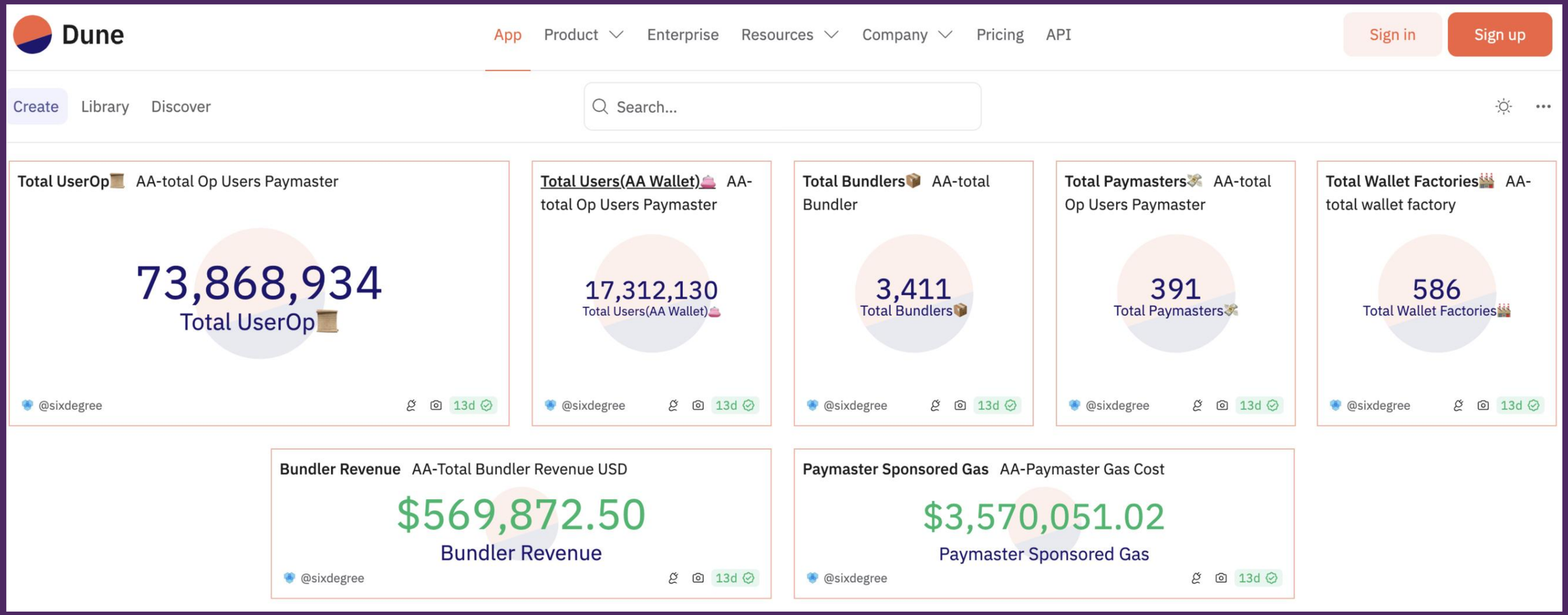
All current Web3 ecosystems **share a common goal**: achieving **mass adoption** from Web2.

AA aims to **simplify the user experience (UX)** in Web3 applications, allowing users to interact with the system without needing to worry about complexities like custody, wallets, or blockchain intricacies. Instead, they are presented with a familiar and user-friendly interface.

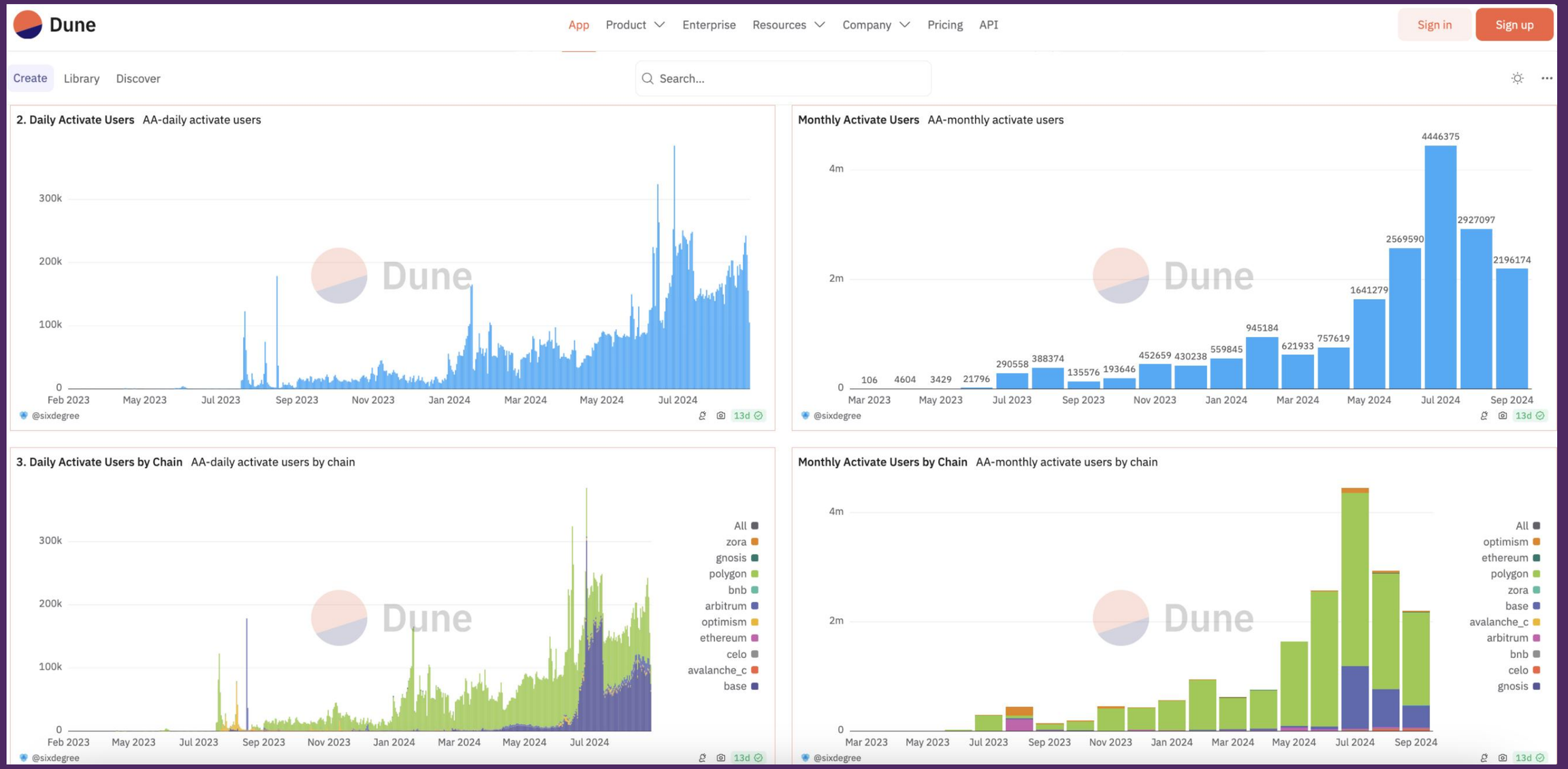
Other Blockchain Ecosystem Efforts

Chains like **TON** and **STARKNET** have already **integrated AA natively** at the architectural level, providing seamless support.

The **Ethereum** ecosystem has explored various approaches to improve account management, including **Social Logins, Passkeys, and Email Logins.**



Showing **exponential growth** in both **transaction volume** and **user base**



Showing **exponential growth** in both **transaction volume** and **user base**

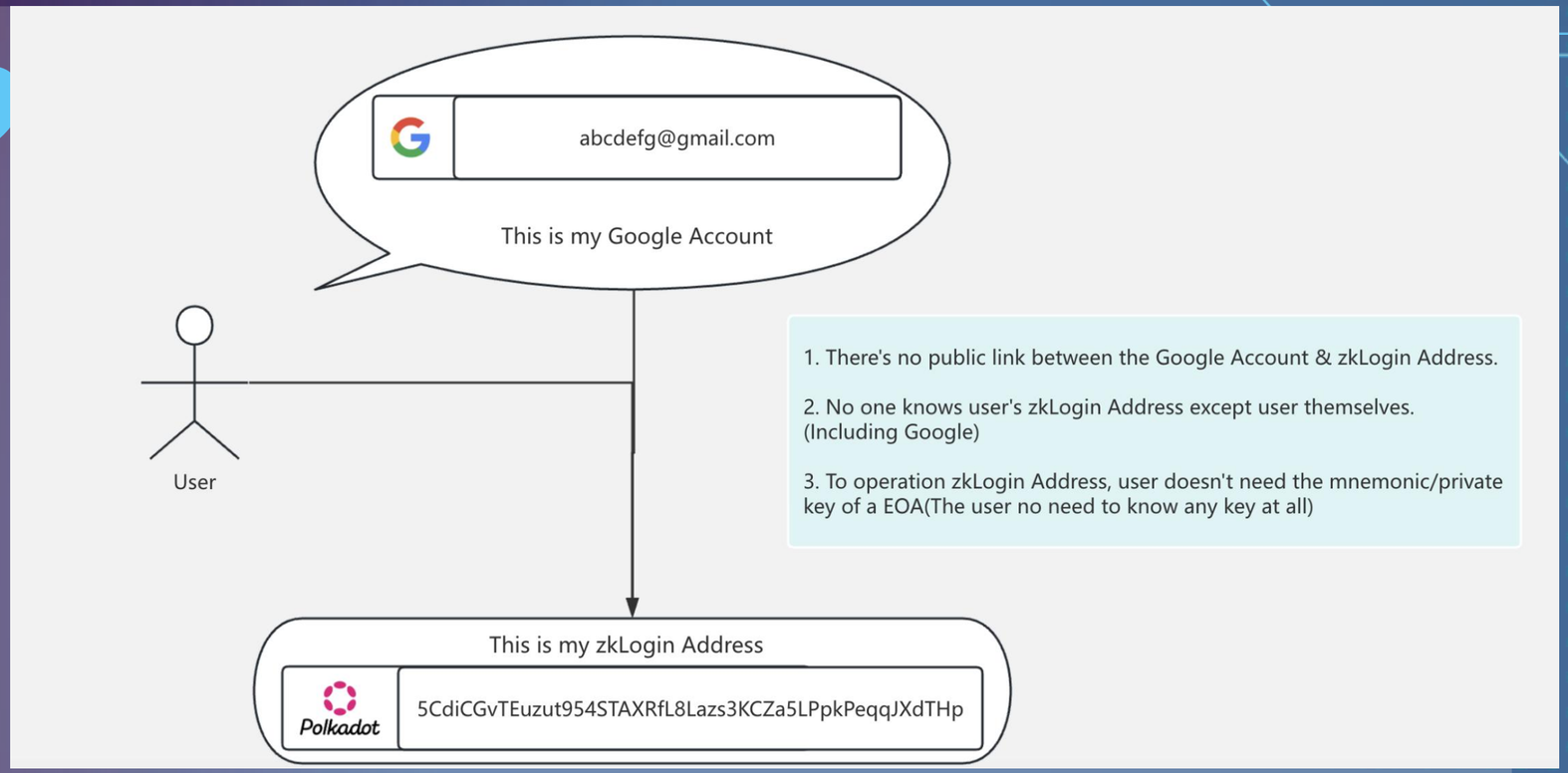
What Is zkLogin?

The zkLogin is built on **WebAuthn** technology and exists as a **Runtime Pallet**, **eliminating** the need for **any modifications** to Polkadot's existing architecture. Any project can integrate zkLogin by simply adding the Runtime Pallet.

WebAuthn is a modern authentication standard offering robust identity verification.

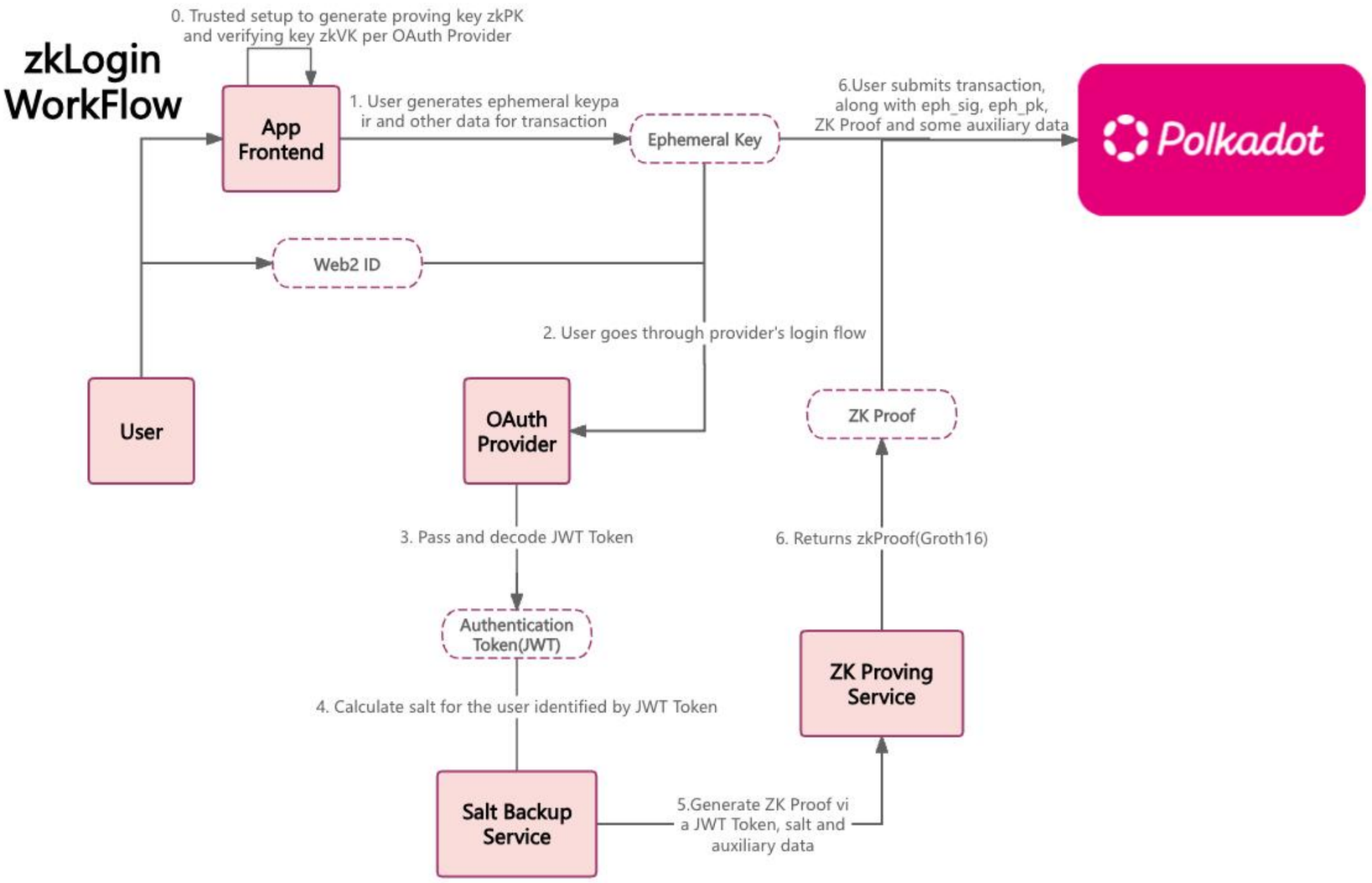
What Is zkLogin?

zkLogin utilizes **Google accounts** as the medium for **user authentication**, proving ownership of the Google account as a way to demonstrate **ownership** of the corresponding **on-chain address(zkLogin Address)**.



– **Traditional blockchain address:** address -> permanent public-private key pair, and the address is derived from the public key.

– **zkLogin:** address -> Google JWT and the user's unique user_salt, making the user unaware of any private key throughout the transaction process. The user's identity is verified via Google's OAuth service.



From privacy perspective:

- No public link between the Google Account and the zkLogin Address.
- No one knows the user's zkLogin Address except the user themselves.

From security perspective:

- Users no longer need to worry about losing a private key
- The only scenario in which a user could potentially lose access to their funds is if, **before the expiration time**, they **simultaneously** lose their temporary private key, JWT, and sal

Performance Comparison: zkLogin vs. Externally Owned Account (EOA)

zkLogin	vs	EOA
Uses zero-knowledge proofs to authenticate without revealing sensitive information	Privacy Protection	Relies on public/private key pairs
Resilient against replay attacks and social engineering attack	Security	Faces risks from private key exposure and phishing attacks
Passwordless login enhances user experience	User Experience	Users must manage private keys and recovery phrases
Authenticate with various services without repeatedly entering sensitive information	Scalability	Typically requires separate authentication for each service

How can zkLogin benefits Polkadot Ecosystem?

Seamless integration of Relaychains & Parachains	Fully compatible with polkadot{.js}
Works as a Runtime Pallet, can be simply added	No need to change Polkadot's on-chain transaction architecture.

Enhance Privacy and Security in the Polkadot Ecosystem	Bring Mass Adoption For Polkadot Ecosystem	Address the lack of 'zero-knowledge technology' and 'account abstraction'
Built upon zero-knowledge proofs, offers users a heightened level of privacy and security	Enhance accessibility, lowers the entry barriers; Reduces the development complexity for DApp developers	Protecting user privacy, Improving user experience

How can zkLogin benefits Polkadot Ecosystem?

Seamless integration of Relaychains & Parachains

zkLogin works as a **runtime pallet**, allowing any relaychain or parachain to directly interact with zkLogin **by simply adding this runtime pallet**.
Polkadot and all its parachains **share a unified codebase**, providing a distinctive advantage for the introduction of the zkLogin solution.

Fully compatible with polkadot{.js}

The construction of zkLogin transaction bodies **is identical to** that of **native Polkadot tx bodies**. Users can utilize it without disrupting Polkadot's on-chain transaction architecture.
This is very **friendly to existing tools**, as any current tools for constructing or initiating transactions in Polkadot (such as polkadot{.js}) do not require any modifications, making it extremely developer-friendly.

How can zkLogin benefits Polkadot Ecosystem?

Enhance Privacy and Security in the Polkadot Ecosystem

Built upon **zero-knowledge proofs**, offers users a heightened level of privacy and security. User authentication information is not required to be stored in plaintext on the blockchain, effectively **safeguarding user privacy**.

Bring Mass Adoption For Polkadot Ecosystem

– **Enhances the accessibility** and **lowers the entry barriers** for users engaging with Polkadot and its parachains. Users can effortlessly and securely manage their authentication information, enhancing the overall user experience.

– **Reduces the development complexity** for DApp developers, allowing any project to quickly integrate zkLogin and attract a large number of Web2 users.

How can zkLogin benefits Polkadot Ecosystem?

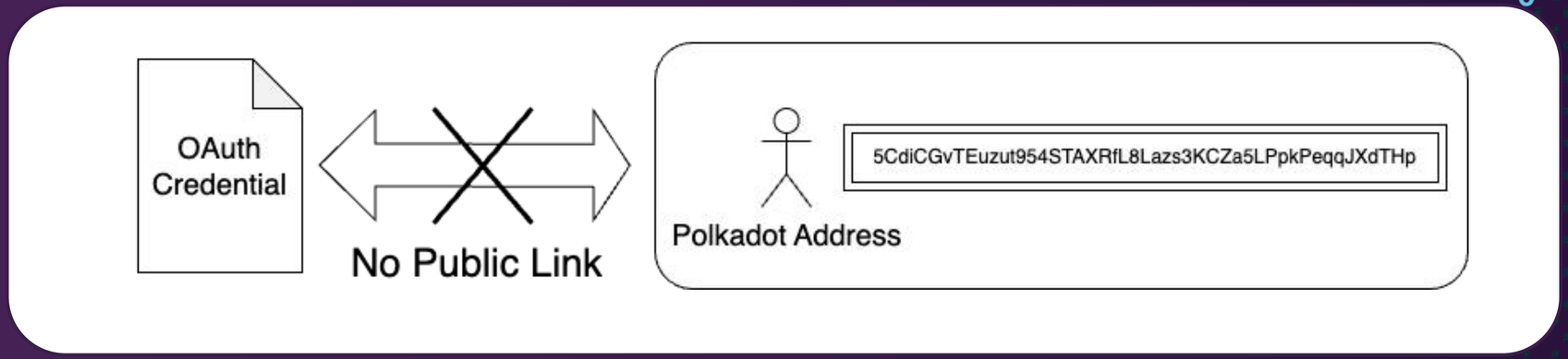
Address the lack of 'zero-knowledge technology' and 'account abstraction' applications

– **Zero-knowledge technology**, which is crucial for protecting user privacy and enhancing transaction efficiency, has **demonstrated significant potential in other blockchain ecosystems**, yet Polkadot has **not fully explored** its practical applications.

– **Account abstraction**, can significantly simplify the interaction process between users and the blockchain, **improving user experience**. This is especially important for onboarding new users and Web2 users.

zkLogin Solution

zkLogin provides the ability for Users to send transactions from a Polkadot address using an OAuth credential, **without publicly linking the two**.



This is one of the simplest ways to onboard Users onto the blockchain. zkLogin **allows users to log in to Web3 applications** using **existing Web2 Authentication Providers** like ***Google***, eliminating the need for users to remember or record private keys.

When using zkLogin, **the only data submitted to the blockchain is the zero-knowledge proof**, **a temporary signature and some auxiliary data**, eliminating the need to submit any user information to the blockchain. Additionally, Web2 Authentication Providers are unaware that users are using the blockchain, ensuring privacy.

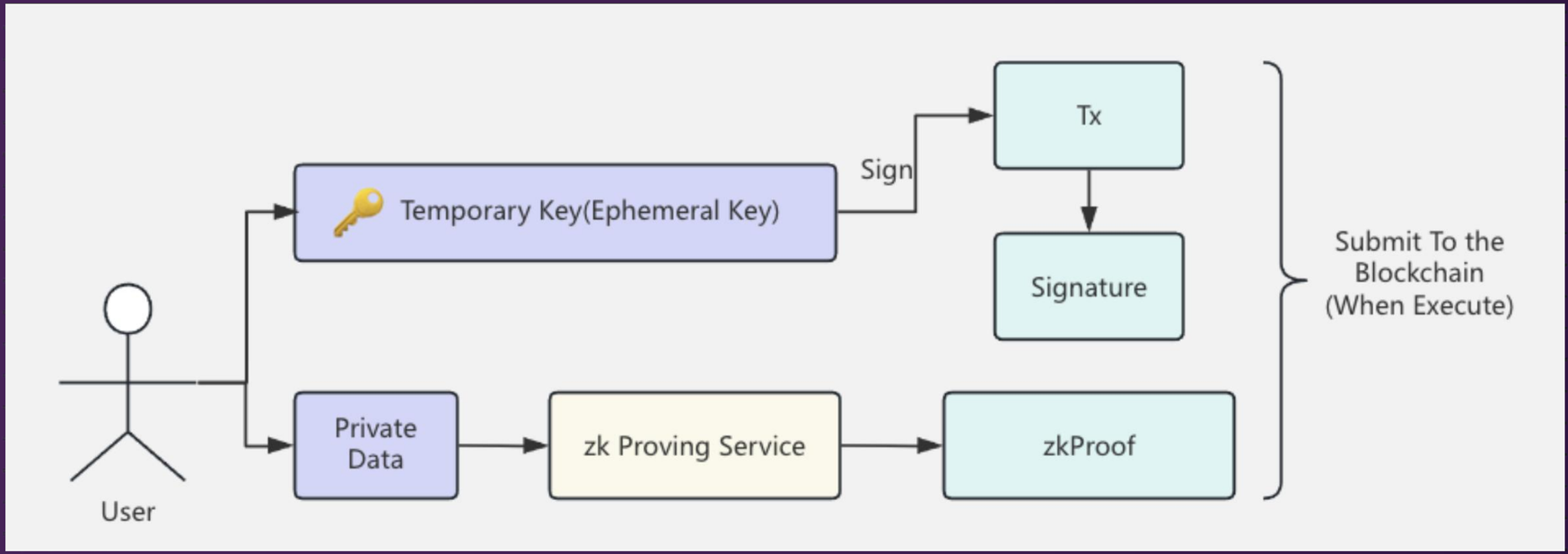
Why using zk?

Aimed at ensuring that users **do not expose any private data**. The following tasks are accomplished within the framework of zero-knowledge proofs:

1. Verifying that the nonce provided to Google for authentication complies with zkLogin rules.
2. Validating the user's JWT token: to prove that the JWT Token was indeed signed by Google.
3. Confirming that the user's zkLogin Address adheres to the generation rules.

– These calculations involve the user's private data and are **performed locally on the user's device**. Only the zero-knowledge proof is submitted to the blockchain for verification, effectively preventing malicious impersonation attacks.

Why using zk?



The zero–knowledge proof represents that **only a specific temporary key** has the right to **manipulate a given zkLogin address** within a designated timeframe(before the expirationTime)

Implementations

01. Frontend Code(Wallet)

02. Backend Code(RuntimePallet)

03. ZKP Circuit Code

Frontend Implementation

Secure Polkadot Account Generation

Implementing sophisticated cryptographic techniques to create and manage secure user accounts

Authentication Platform Integration

Seamlessly integrate with authentication platforms, facilitating a smooth user login experience

Zero-Knowledge Proof Generation

Ensures the privacy and integrity of user data during authentication processes

zkLogin Transaction Integration

Authenticate users and authorize transactions without exposing sensitive information

Frontend Implementation

Secure Polkadot Account Generation

Implementing sophisticated cryptographic techniques to create and manage secure user accounts

During the Account Generation Phase:

1. Generate and store an ephemeral KeyPair(`eph_sk`, `eph_pk`). (Follow the same process as you would generating a KeyPair in a traditional wallet.)

2. Set the **expiration time** for the ephemeral KeyPair. The wallet decides from which block number the ephemeral keypair will become invalid.

3. Generate the JWT randomness(`jwt_randomness`) for current session.

Frontend Implementation

During the Authentication Phase:

1. **Google Account Signing in**, Fetching and Decoding JWT Token.

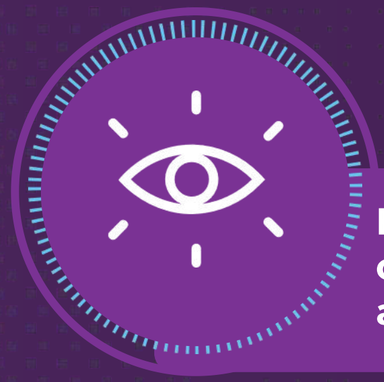
2. Generating or restoring `user_salt`: After obtaining the `JWT Token`, the Wallet Extension generates/restores a random number called `user_salt`.

3. Constructing User's **Polkadot Account Address**:
const zkLoginUserAddress = jwtToAddress(jwt, userSalt);

Authentication Platform Integration

Seamlessly integrate with authentication platforms, facilitating a smooth user login experience

Frontend Implementation



Zero-Knowledge Proof Generation

Ensures the privacy and integrity of user data during authentication processes

During the zkProof Generation Phase:

The Wallet Extension is embedded with a **ZK Proof Generation Service**, enabling the local generation of zk proofs, utilizing the **groth16 proof system**. This is an attestation (proof) over the ephemeral key pair that proves the ephemeral key pair is valid.

Frontend Implementation

During the Transaction Phase:

1. Signing txs via `Ephemeral Key`: **Sign the transaction bytes** with the **ephemeral private key** using the key pair generated previously. This is the same as traditional KeyPair signing.

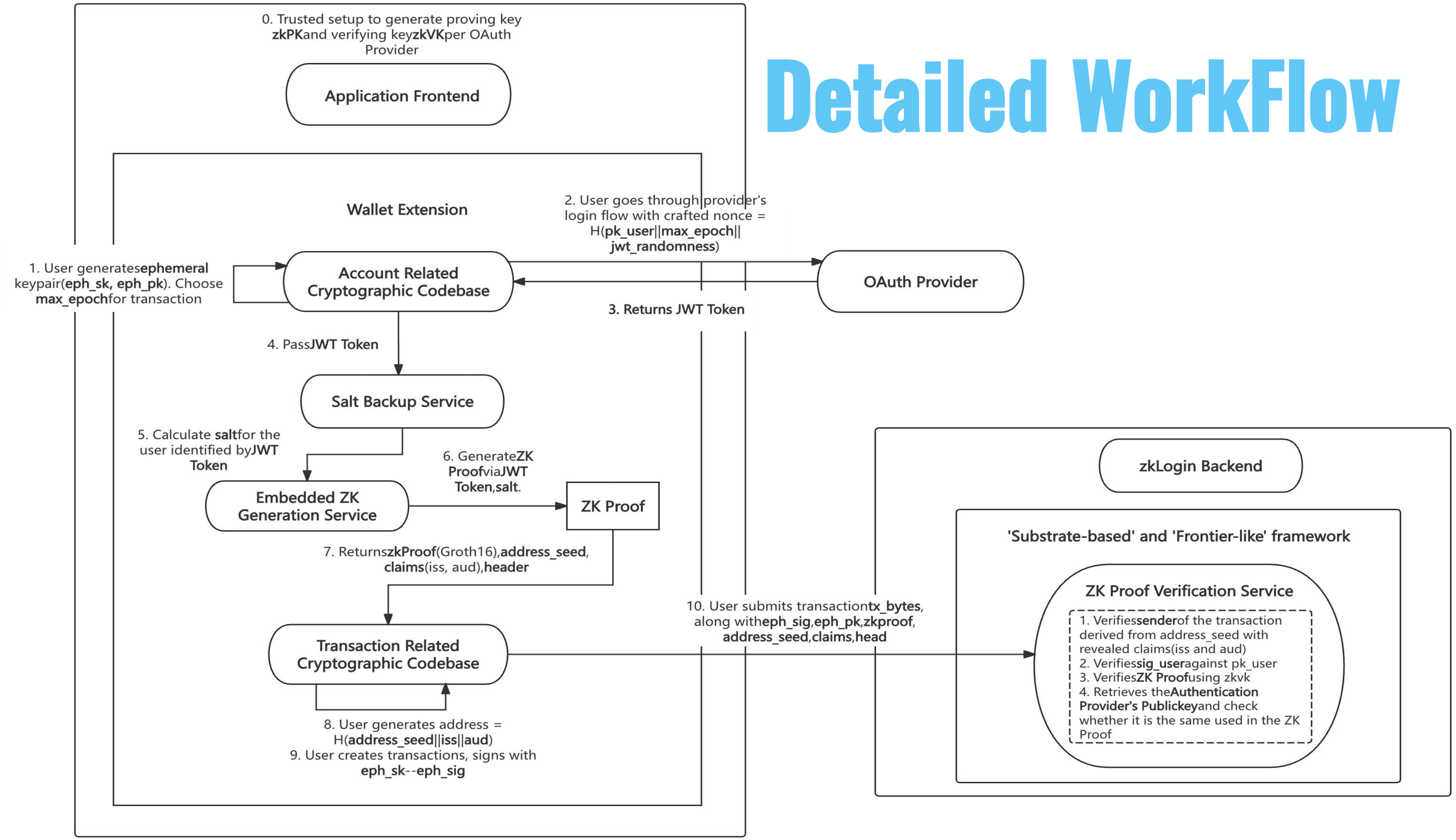
2. Combining ZK Proof With the previous Sig: **Combine** the previously obtained **zero-knowledge proof** with the **signature** of the transaction made by the user using the ephemeral keypair, resulting in the final signature(zkLoginSignature)

3. Submiting the tx

zkLogin Transaction Integration

Authenticate users and authorize transactions without exposing sensitive information

Detailed WorkFlow



Backend Implementation

Integrating with 'Substrate-based' Framework

- 'Substrate-based' Framework
- High Portability Code

Zero-Knowledge Proof Verification

The backend ensures a robust and streamlined verification process

Backend Implementation

Integrating with 'Substrate-based' Framework

- 'Substrate-based' Framework
- High Portability Code

'Substrate-based' & High Portability Code

– The backend **using the "substrate–based" framework, Runtime Pallet!**

– This project is designed for **universal use** within the entire Polkadot ecosystem. We prioritize the **robustness and portability of the backend**

Backend Implementation

Robust & Streamlined Verification Process

– **Efficient verification modes** which prioritizes efficiency and accuracy are needed for implementation in the backend.

– The backend ensures a **robust and streamlined verification process**, contributing to the overall efficiency and reliability of the zkLogin system.

Zero-Knowledge Proof Verification

The backend ensures a robust and streamlined verification process

ZKP Circuit Implementation

Stable and Efficient Circuit Code

We have **implemented the ZKP Circuit Code** for the zero-knowledge proof component within the process, ensuring seamless integration of this functionality into the overall workflow.

THANKS!