# SoloSafe : Offline permissionless payment protocol

---

Presented by El (el@solosafe.xyz)

Date: 2025-04-15

Version: 0.1

## Table of Contents


- Abstract
- Introduction
- Problem Statement
- Proposed Solution
  - Bridge
  - PCD
  - Publisher
- System architecture
- Implementation
- Conclusion
- References
- Appendix
- Acknowledgements
- License
- Contributing
- Contact
- About
- Changelog

## Abstract

# Introduction

Payment systems are a critical part of modern commerce, enabling the transfer of value between parties. However, many existing payment systems are now fully decentralized and rely on a consensus mechanism to validate transactions. These consensus mechanisms relay on a state machine that is replicated across all nodes in the network. While this is truly revolutionary, many people that do not have reliable internet access are unable to use these systems. Most of developed countries have good centralized financial infrastructure and internet access, but many developing countries do not. This paper proposes a new payment system that is offline and tamper-resistant, allowing users to make transactions without the need for a centralized authority or internet access. The system is based on a combination of cryptographic techniques and hardware security modules to ensure the integrity and authenticity of transactions. The proposed system is designed to be easy to use and accessible to anyone, regardless of their technical expertise or access to technology.

# Problem Statement

Currently, digital payments rely uppon access to the Internet. Consensus mechanisms allow permissionless transactions to be verified and recorded on a distributed ledger. However, it is currently difficult to let users to make transactions using edge devices that are connected through ad hoc networks like Bluetooth or Wi-Fi Direct. This is due to the fact that each device having its own state machine, could be out of sync with the rest of the network and some malicious user could modify their state to steal funds.

This paper proposes a new payment system that is offline and tamper-resistant, allowing users to make transactions without the need for a centralized authority or internet access. The system is based on a combination of cryptographic techniques and hardware security modules to ensure the integrity and authenticity of transactions. The proposed system is designed to be easy to use and accessible to anyone, regardless of their technical expertise or access to technology.

# Proposed Solution

| Header 1 | Header 2 | Header 3 |
|---|---|---|
| Cell A1 | Cell A2 | Cell A3 |
| Cell B1 | Cell B2 | Cell B3 |

| Number | Name | Description |
|---|---|---|
| 1 | Jean | A person |
| 2 | John | A person |

---

### Preliminaries

This section describes the main components of the proposed system to allow the reader to understand the system architecture. The system is composed of the following components :

- `Zero Knowledge Bridge` : This bridge is used to connect the offline payment system with the existing blockchain networks. The zero knowledge aspect of the bridge is not used to hide the transaction details, but to allow the bridge to be used in a trustless manner. The bridge is used to sign transactions and publish them to the blockchain network.
- `Device ID` : This is a unique identifier for each device in the world. The device ID is used to identify the device and to sign transactions. It is made up of a public and a private component.
- `Trusted Execution Environment (TEE)`: This is a secure area of the device that is used to store the private key and to sign transactions. The TEE is used to ensure that the private key is not exposed to the outside world and that it is only used to sign transactions. We also use a TEE to allow us preserve the key to the state of the device.
- `Offchain transaction` : This is a transaction that is created and signed offline between two devices. Tipically, this is a bridge transaction from a device to another device.
- `Download an asset` : This is a bridge transaction that burns the asset onchain and mints it on the device.
- `Upload an asset` : This is a bridge transaction that mints the asset onchain and burns it on the device.
- `Defungibilize` : By this operation, a fungible asset is converted into a non fungible asset to allow the user to use it in a trustless manner and make it trackable.
- `Fungibilize` : By this operation, a non fungible asset is converted into a fungible asset to allow the user to use it in a trustless manner and make it trackable.

- `Proof Carrying Data(PCD)[^2]` : This is a proof that is used to verify the transaction. The PCD is used to ensure that the transaction is valid and that it has not been tampered with. The PCD is also used to ensure that the transaction is signed by the device ID and that it is valid.

**1. Bridge**

**2. PCD**

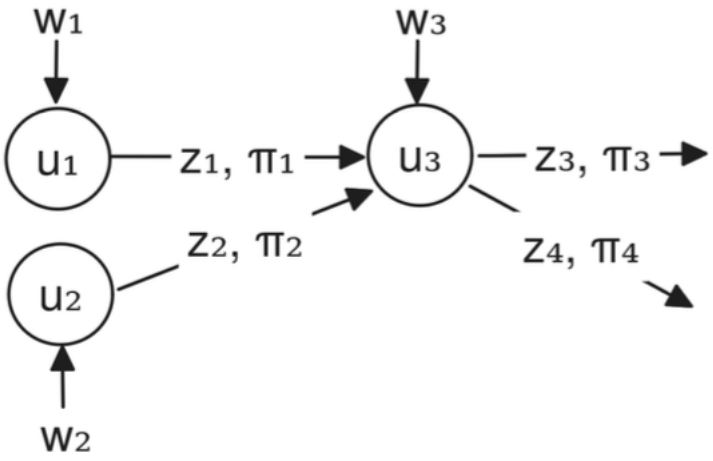


Figure 1: PCD

**3. Publisher**

**4.**

**5. Virtual Trusted Execution Environment (vTEE)**

A Trusted Execution Environment is a hardware component allowing to execute some program in a seprate environment for security reason. Instead of running some programs on the main operating system environment, the code is executed on an isolated and safe environement.

Unfortunately, the TEE feature is not available on all devices. For the device having it, it provides limited features that are not fit to our usecase. The Licel project introduces the use of virtual TEE(vTEE)[1] to allow the use of TEE on all devices.

[1]Why virtual trusted execution environments are set to boost mobile payment security, June 28th, 2024

In order to optimize execution of our code, we present our vTEE architecture that is adapted to offline payments.

**Inputs :** The SoloSafe vTEE is designed to receive an array of plain or encrypted bytes.

—> [ byte[], output flag, output public key ] or —> [ encrypted bytes, output flag, output public key ]

**Outputs :** The SoloSafe vTEE is designed to return an array of bytes. It can output unencrypted or encrypted data depending on a flag passed to the vTEE.

–> [ encrypted bytes ] –> or –> [ byte[] ] –> [ byte[] ] –> [ byte[] ]

**Execution :**

**Instruction set :** The SoloSafe vTEE possesses a set of programs that are executed in a secure environment. Here is a list of the predefined programs that are available in the vTEE: - `Receive` : This program is used to receive data from the main operating system. The data is passed to the vTEE in a secure manner and is not exposed to the outside world. - `Send` : This program is used to send data to the main operating system. The data is passed to the vTEE in a secure manner and is not exposed to the outside world.

**Arbitrary code execution :** The SoloSafe vTEE can also execute arbitrary code in the secure environment using an Assembly Language inspired by RISC architecture or brainfuck architecture. The code is executed in a secure environment and the result is returned to the main operating system. Here is a list of the instructions set Supported by the SoloSafe VTEE:

| Instruction | Hexadecimal | Description |
|---|---|---|
| `>` | 0x01 | Move the pointer to the right |
| `<` | 0x02 | Move the pointer to the left |
| `+` | 0x03 | Increment the value at the pointer |
| `-` | 0x04 | Decrement the value at the pointer |
| `.` | 0x05 | Output the value at the pointer |
| `,` | 0x06 | Input a value and store it at the pointer |

| Instruction | Hexadecimal | Description |
|---|---|---|
| `[` | 0x07 | Jump to the matching `]` if the value at the pointer is zero |
| `]` | 0x08 | Jump to the matching `[` if the value at the pointer is non-zero |
| `@` | 0x09 | Execute the code in the secure environment |
| `#` | 0x0A | End the program |
| `!` | 0x0B | Execute the code in the secure environment and return the result |
| `&` | 0x0C | Encrypt the data on the pointer |
| `*` | 0x0D | Decrypt the data on the pointer |
| `^` | 0x0E | Sign the data on the pointer |
| `~` | 0x0F | Verify the signature on the data on the pointer |
| `\|` | 0x10 | Hash the data on the pointer |
| `:` | 0x11 | Compare the data on the pointer |
| `;` | 0x12 | Jump to the matching `:` if the data on the pointer is equal |

## System architecture

Here is a diagram of the system architecture:

```
graph TD;
    A[User] -->|Create transaction| B[PCD]
    B -->|Sign transaction| C[Bridge]
    C -->|Publish transaction| D[Publisher]
    D -->|Verify transaction| E[User]
```

> [!NOTE] Useful information that users should know, even when skimming content.
>
> [!TIP] Helpful advice for doing things better or more easily.
>
> [!IMPORTANT] Key information users need to know to achieve their goal[^2].

[!WARNING] Urgent info that needs immediate user attention to avoid problems.

[!CAUTION] Advises about risks or negative outcomes of certain actions.

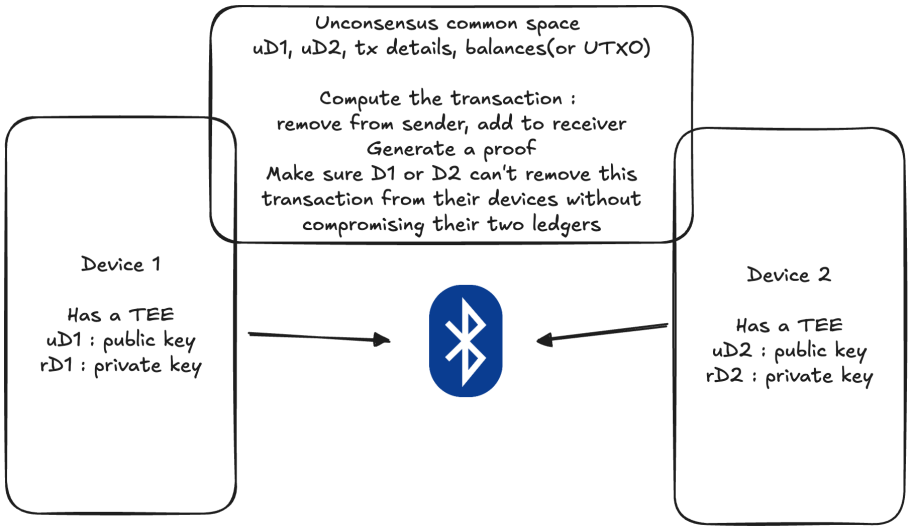


Figure 2: System Architecture

## Implementation

## Conclusion

## References


-