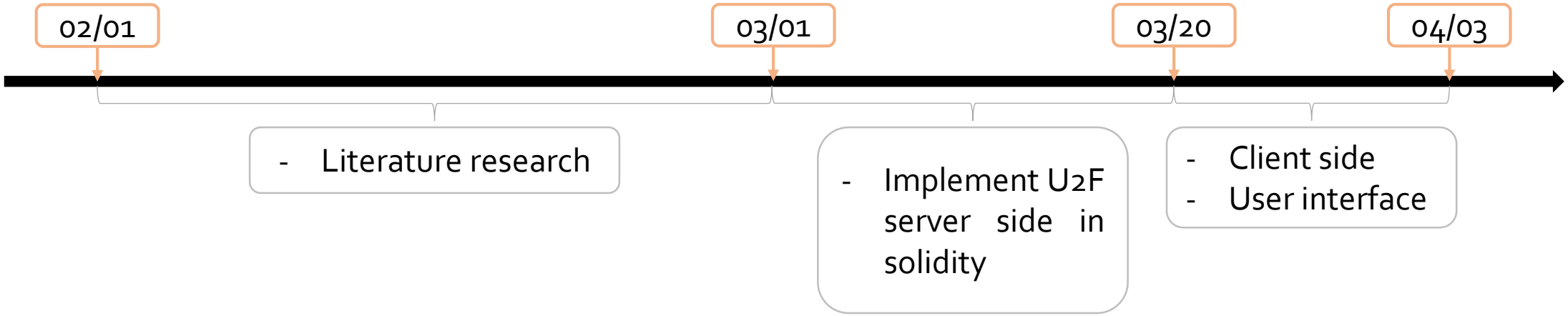
U2F Project Report

2019/02/01 – 2019/04/03

Victoria Chen

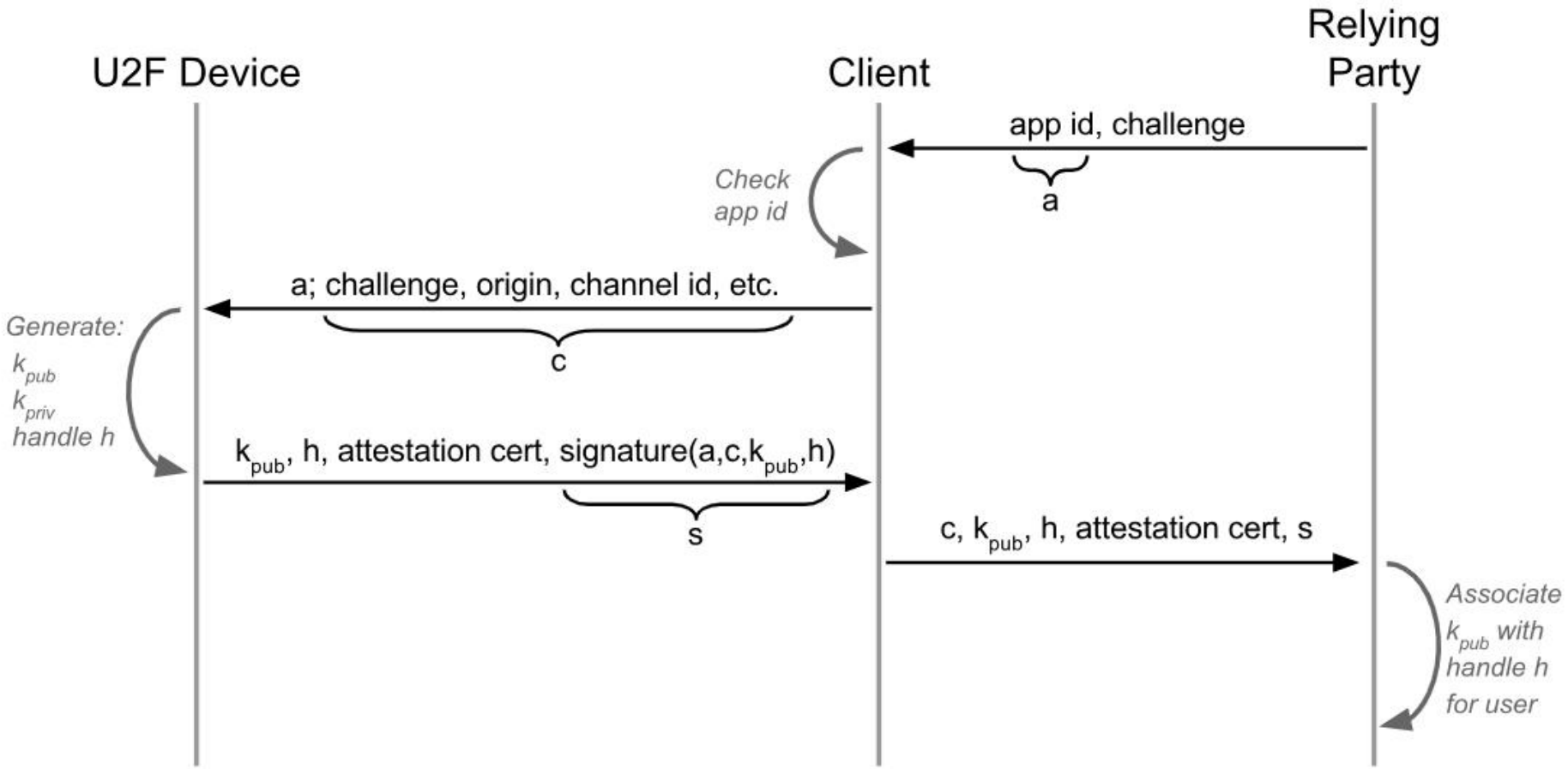
Overview



1. Implemented and debugged U2F server side in solidity.
2. Combined the server side and the client side (u2f-api in Chrome browser).
3. Developed a Bank DApp using smart contract.
4. Designed and implemented a user interface to interact with the DApp.

U2F Server Side

Challenge-Response



Constraints in Smart Contract

1. Web-safe encode and decode in smart contracts take a large amount of gas.
2. The size of a single smart contract cannot be too large.
3. Verification of signature via X.509 certificate is hard to implement in smart contracts.

↓

1. Remove a part of the parsing, encoding and decoding of the data out of smart contracts. These operations are instead realized in Javascript.
2. Utilize multiple libraries. Split the U2F library into two libraries, `Register` and `Authenticate`. (While all other libraries that realize basic functionalities contain only internal functions, `Register` and `Authenticate` contain two public functions each. This is because if all the functions are internal, the bytecode will be too large.)
3. Use Javascript to extract the public key from the certificate and verify the signature using the public key.

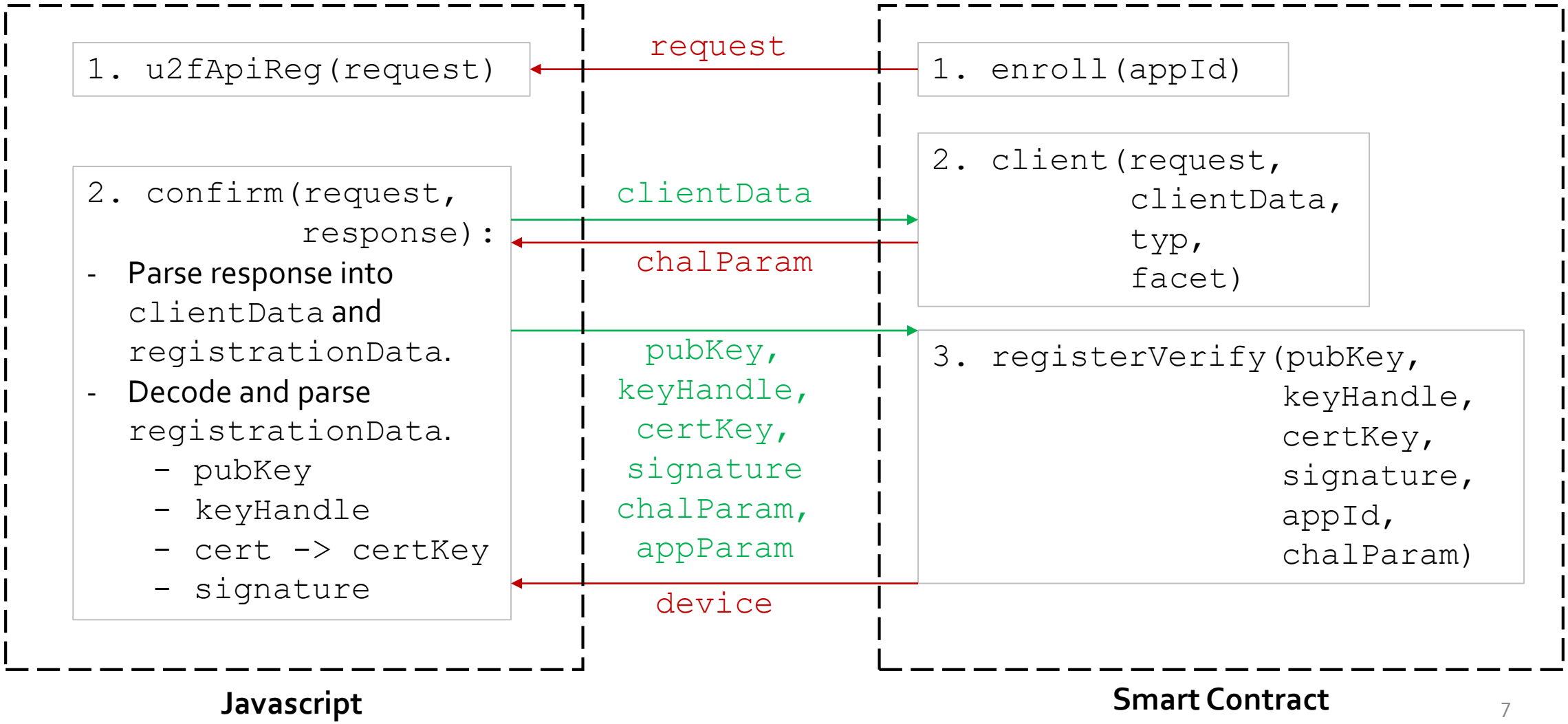
U2F Server Functions: Register (Python)

1. `enroll(appId)`:
    - Generate 32-byte challenge.
    - Pack the `challenge`, `appId`, `version` information as a JSON request string.
    - Store the request string for the user.
2. `bind(request, response, appId)`:
    - Fetch and delete the stored register request.
    - Parse the response into `version`, `challenge`, `clientData`, and `registrationData`.
    - Verify `clientData`:
        - Same `type` (register or authenticate).
        - Same `challenge`.
        - Same `appId`.
    - Verify `registrationData`:

```
+-------------------------------------------------------------+
| 1 |      65      | 1 |    L    |    implied    |     64    |
+-------------------------------------------------------------+
0x05 : pubKey : keyHandleLength : keyHandle : cert : signature
```

    - Store the device information.

U2F Server Functions: Register (Solidity)



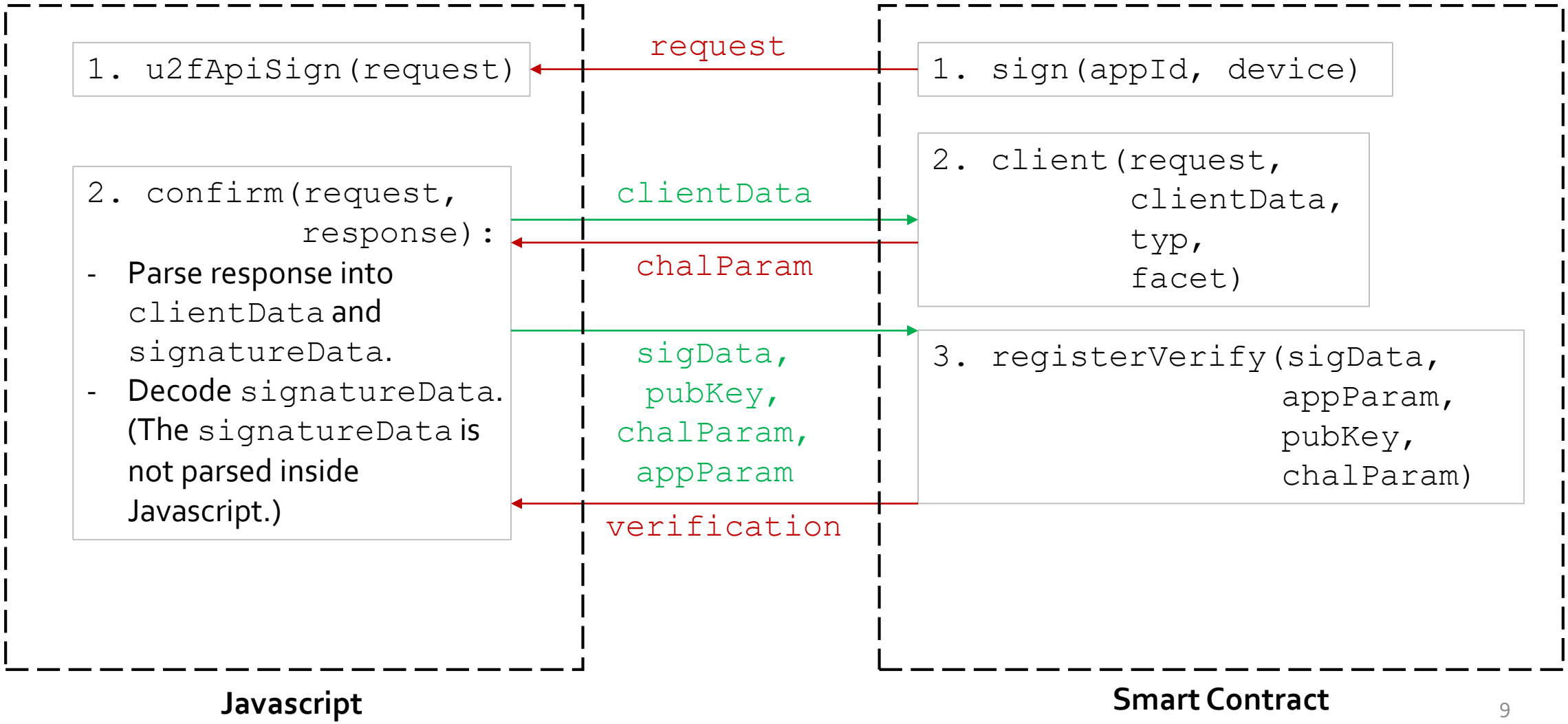
U2F Server Functions: Sign (Python)

1. `sign(appId, device)`:
    - Generate 32-byte challenge.
    - Pack the `challenge`, `appId`, `version`, and `device` information as a JSON request string.
    - Store the request string for the user.
2. `verify(request, response, appId)`:
    - Parse the response into `version`, `challenge`, `clientData`, and `signatureData`.
    - Verify `clientData`:
        - Same `type` (register or authenticate).
        - Same `challenge`.
        - Same `appId`.
    - Verify `signatureData`:

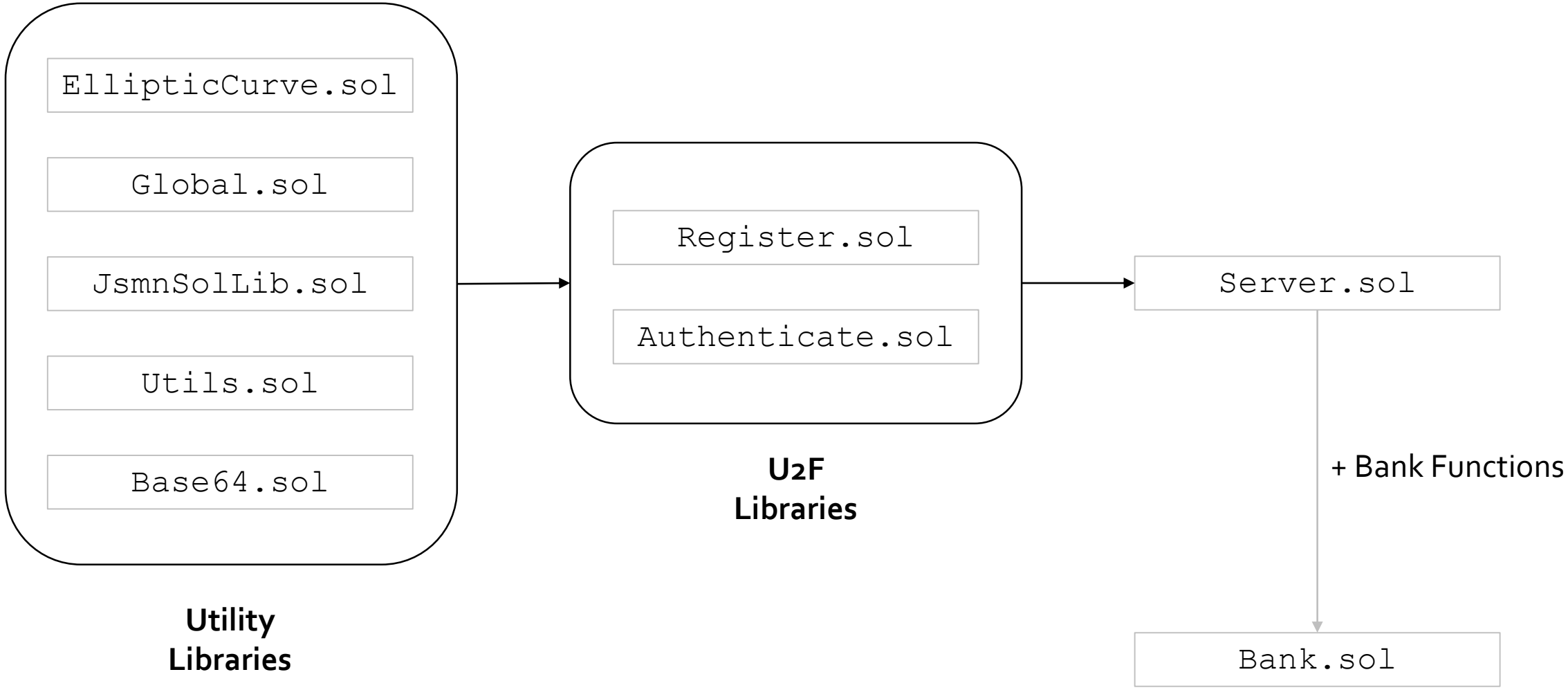
```
+-------------------------------------+
| 1 |      4      |       implied       |
+-------------------------------------+
userPresence : counter : signature
```

    - Store verification details.

U2F Server Functions: Sign (Solidity)



Dependencies of Smart Contracts



ETH Bank

ETH Bank: Overview

Register and Login

1. Register:
   - Deploy a Bank contract that is associated with the user.
   - Setup password.
2. Login:
   - Direct the user to the homepage.

ETH Bank Page

Register U2F token

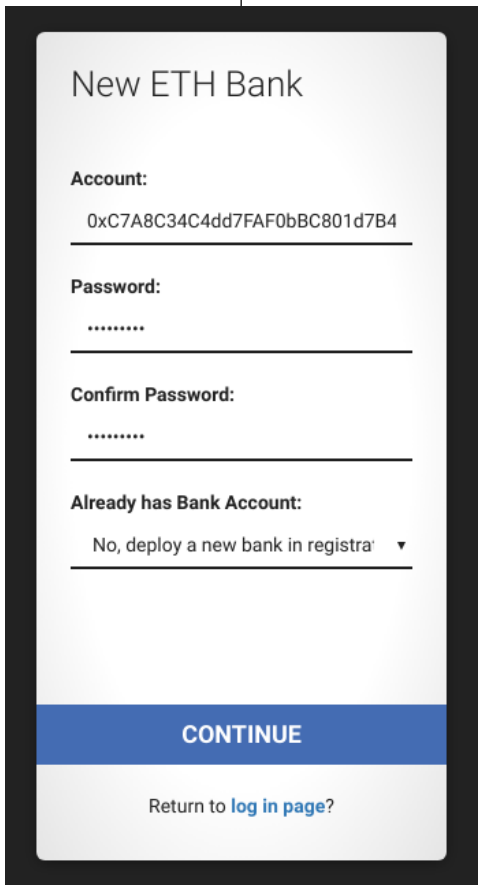
1. Deposit
2. Withdraw
3. Transfer via bank

4. Lock the bank

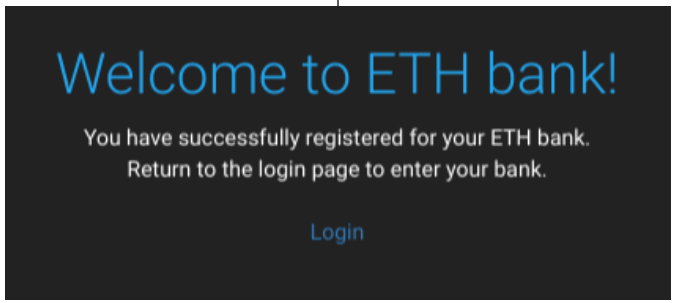
[When the value in the first three operations exceeds the transfer limit, U2F authentication is required.]

ETH Bank: Register and Login

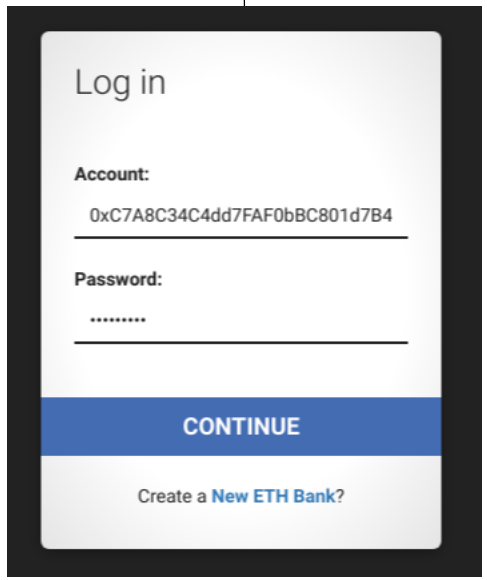
Register



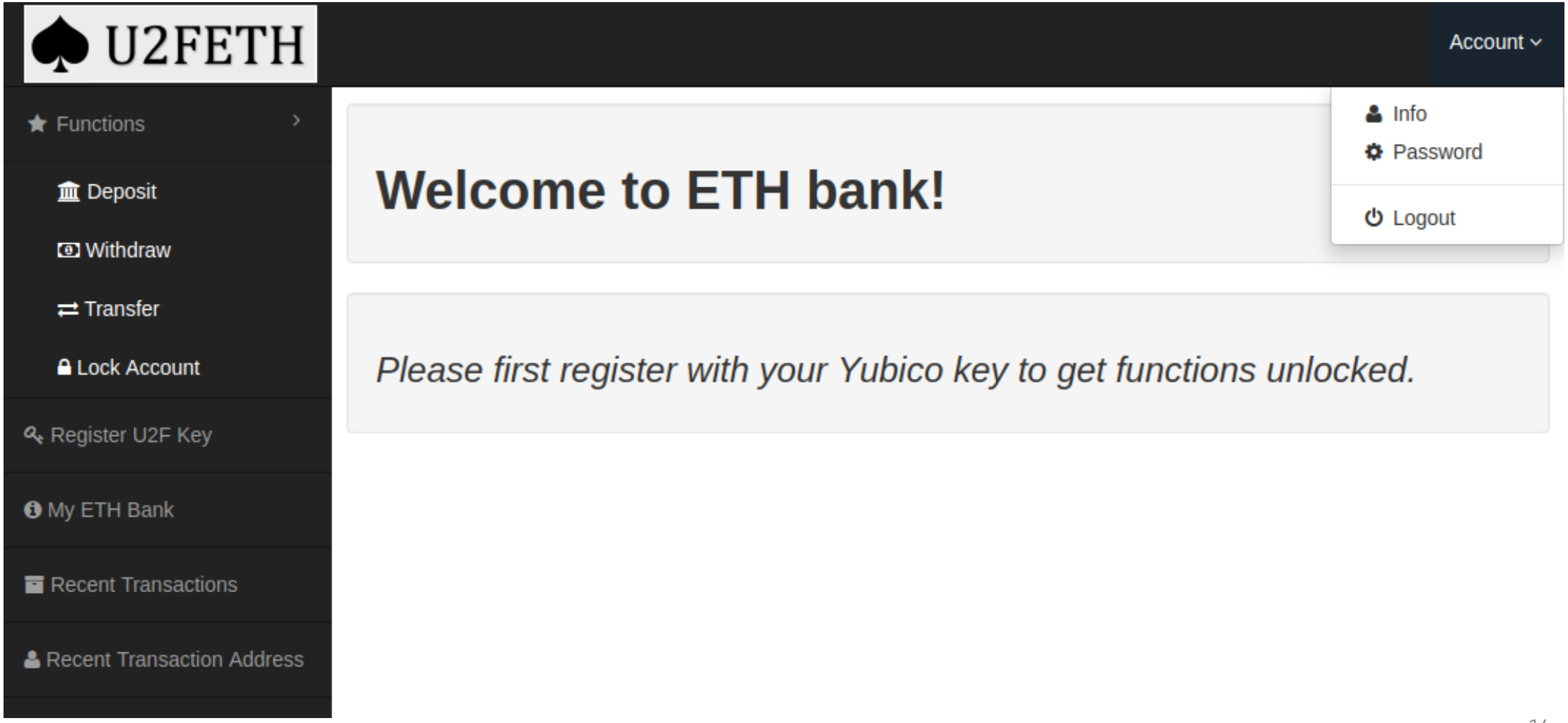
Register Success



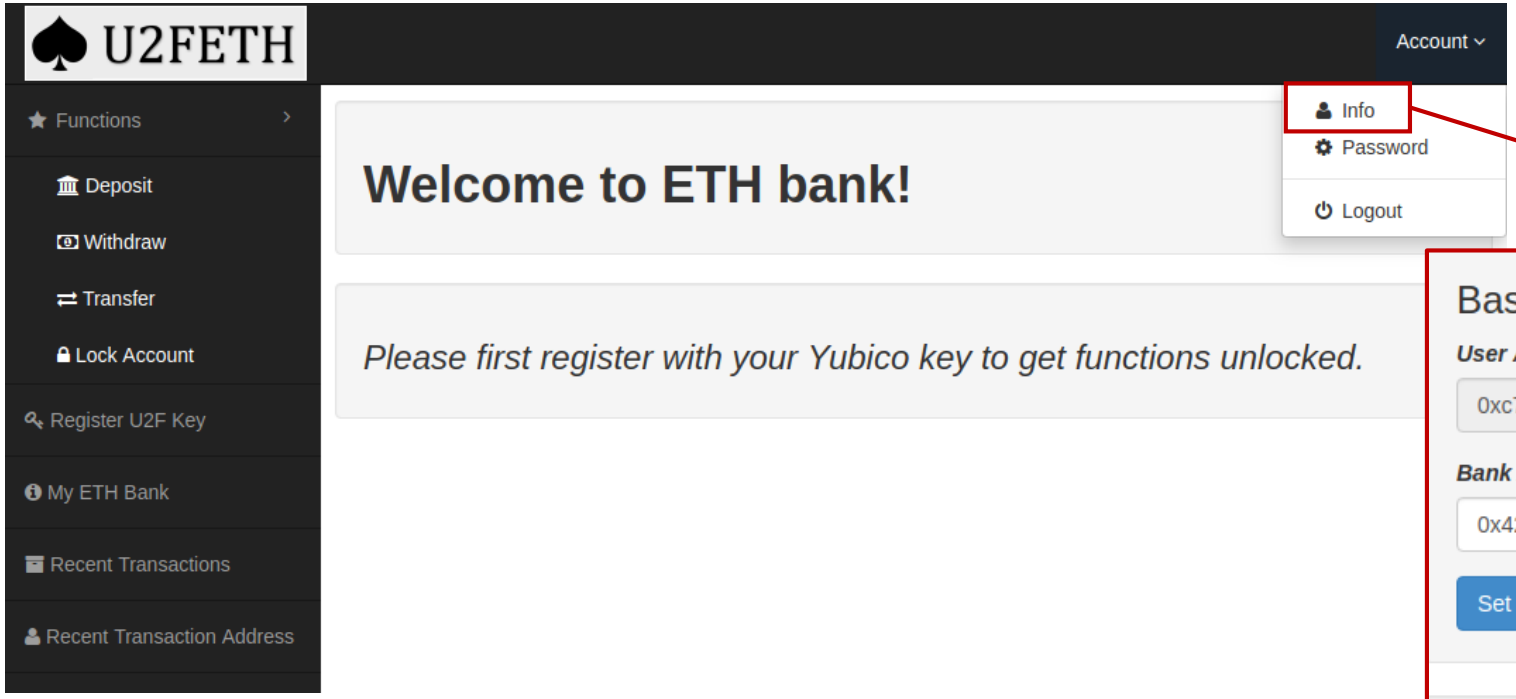
Login

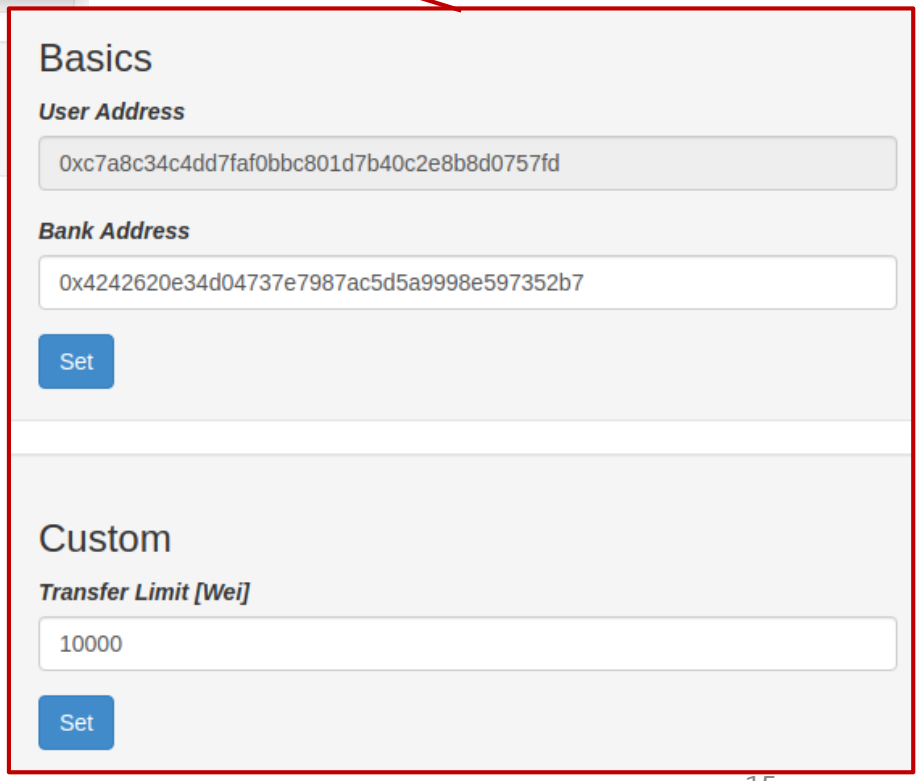


ETH Bank: Homepage



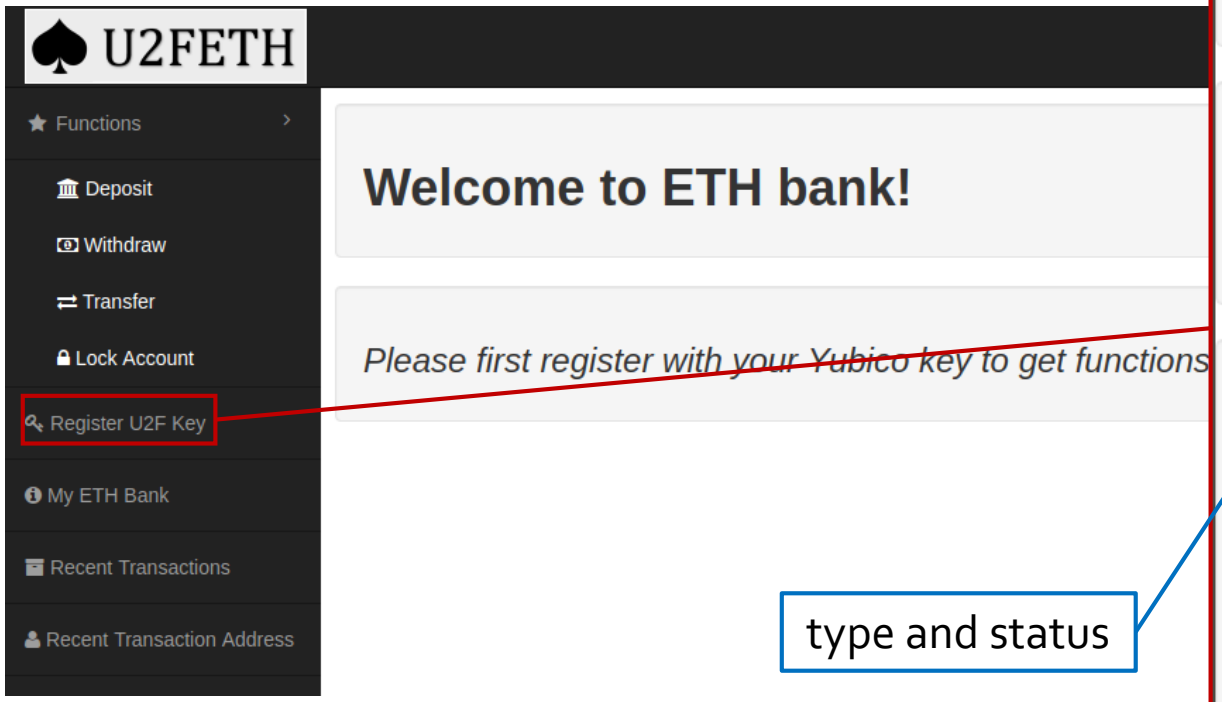
ETH Bank: Info Page (Unregistered)

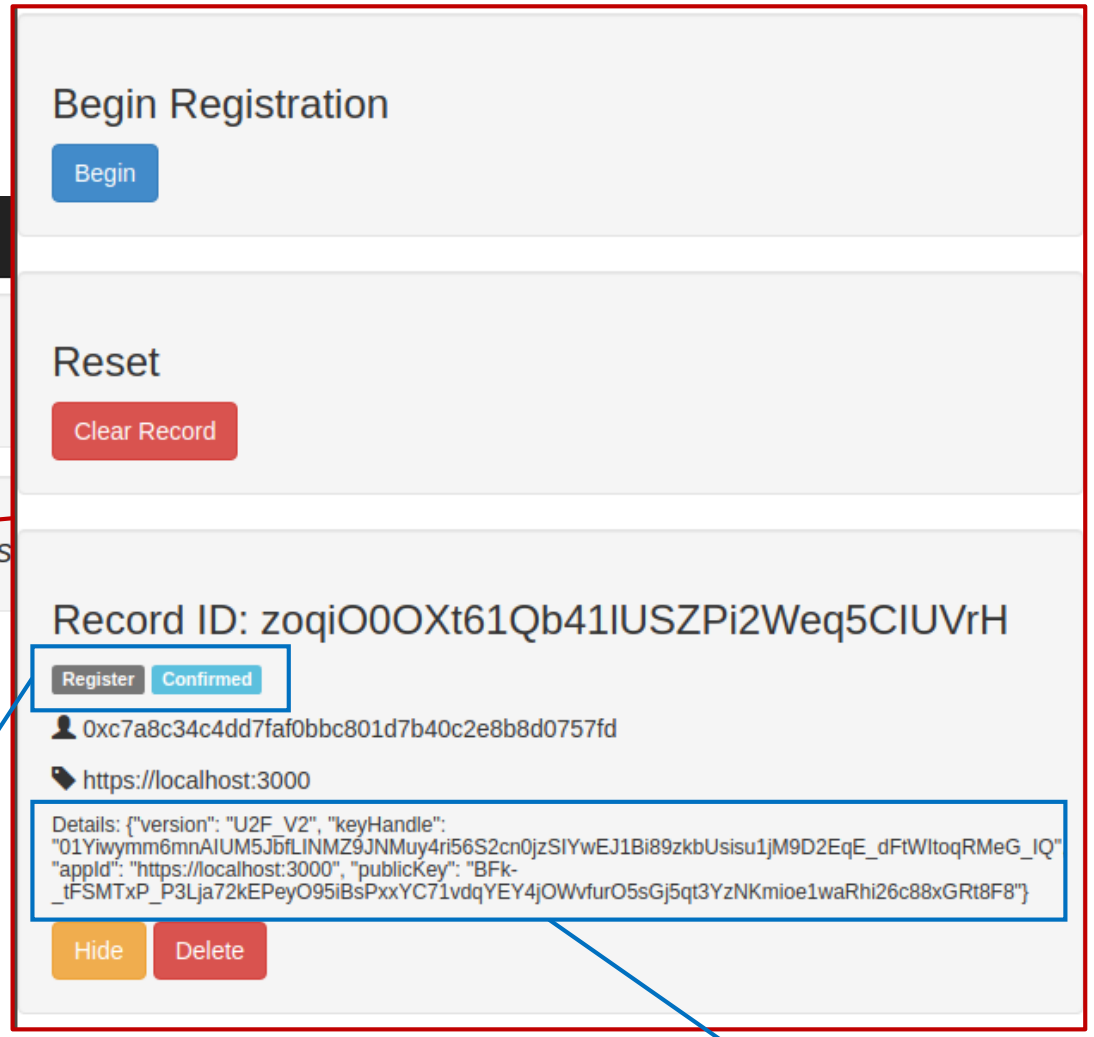




1. Set bank address:
   - Homepage will be reloaded.
   - A new U2F registration process is required.
2. Set transfer limit:
   - No U2F authentication required.

ETH Bank: U2F Register



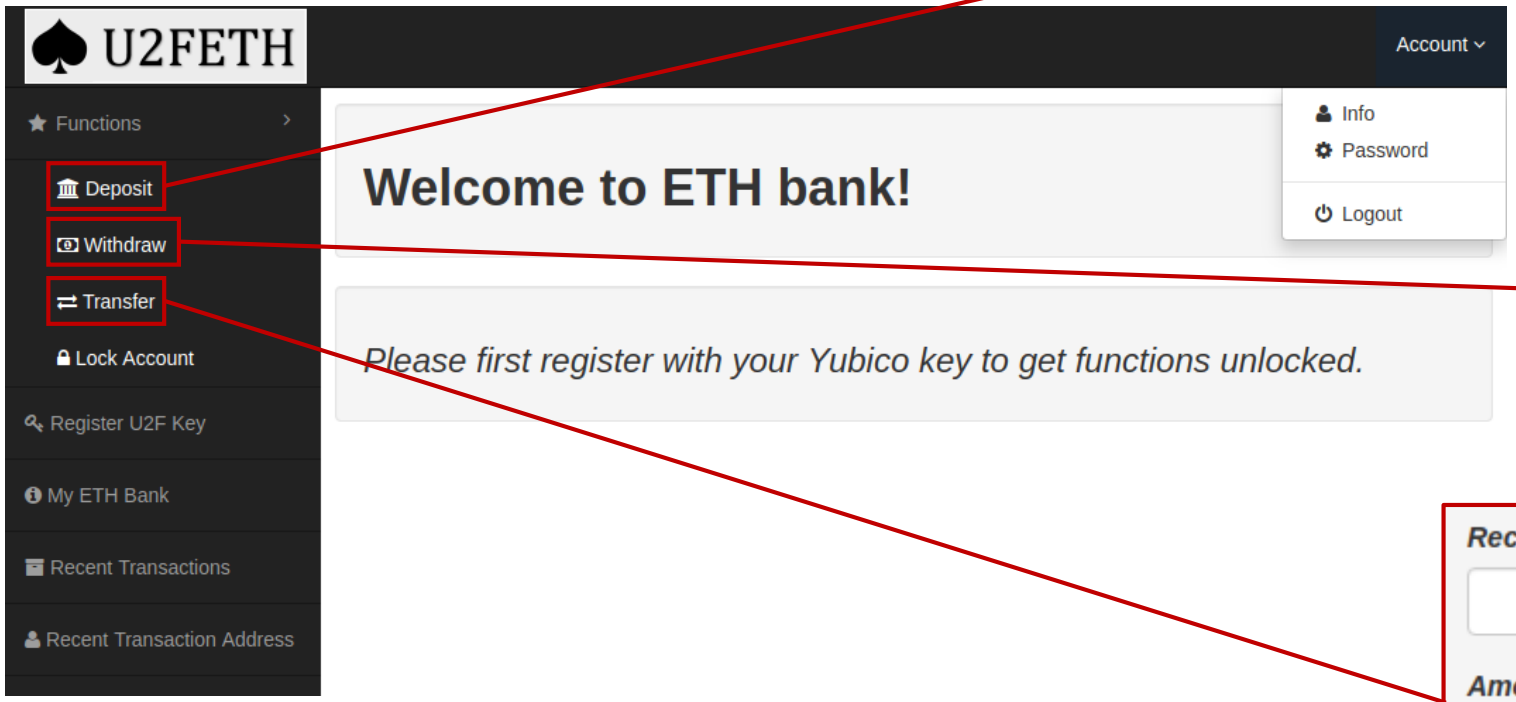


type and status

registered device

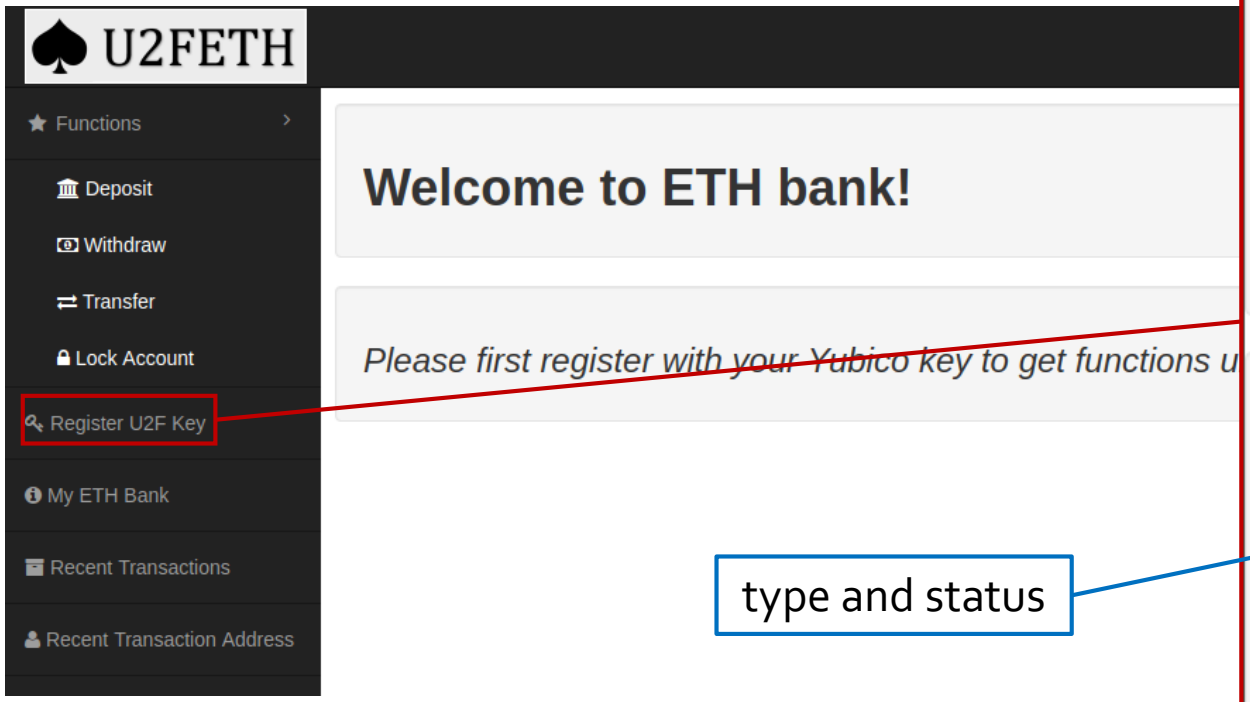
1. Begin registration.
2. Request appears in the record panel. Confirm registration with the specific record.

ETH Bank: Functions



1. Functions are available after U2F registration.
2. When values exceed the transfer limit, U2F authentication is required. (U2F authentication fires automatically, so users need to make sure that their U2F token has been properly inserted before sending the transaction.)

ETH Bank: U2F Sign

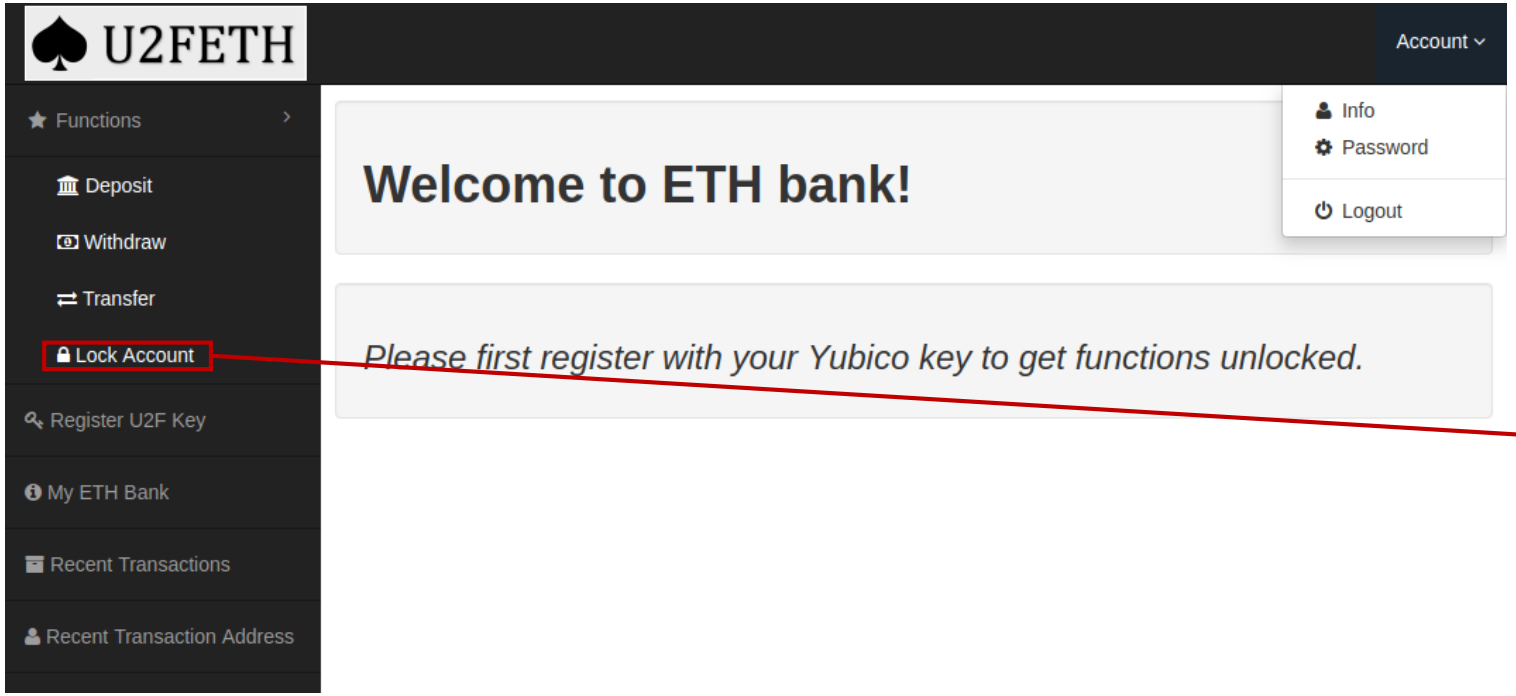


type and status

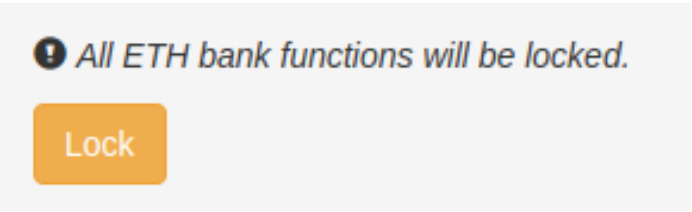
verification

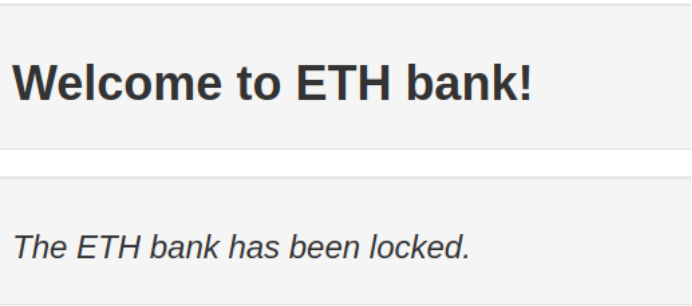
1. Authentication records can also be viewed in the U2F page.

ETH Bank: Lock Account

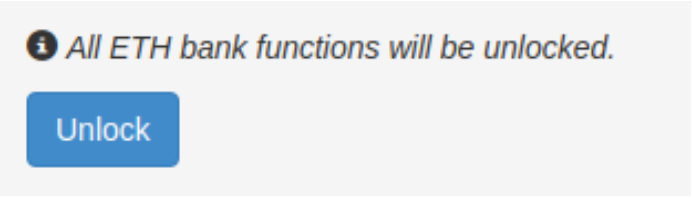


Lock:



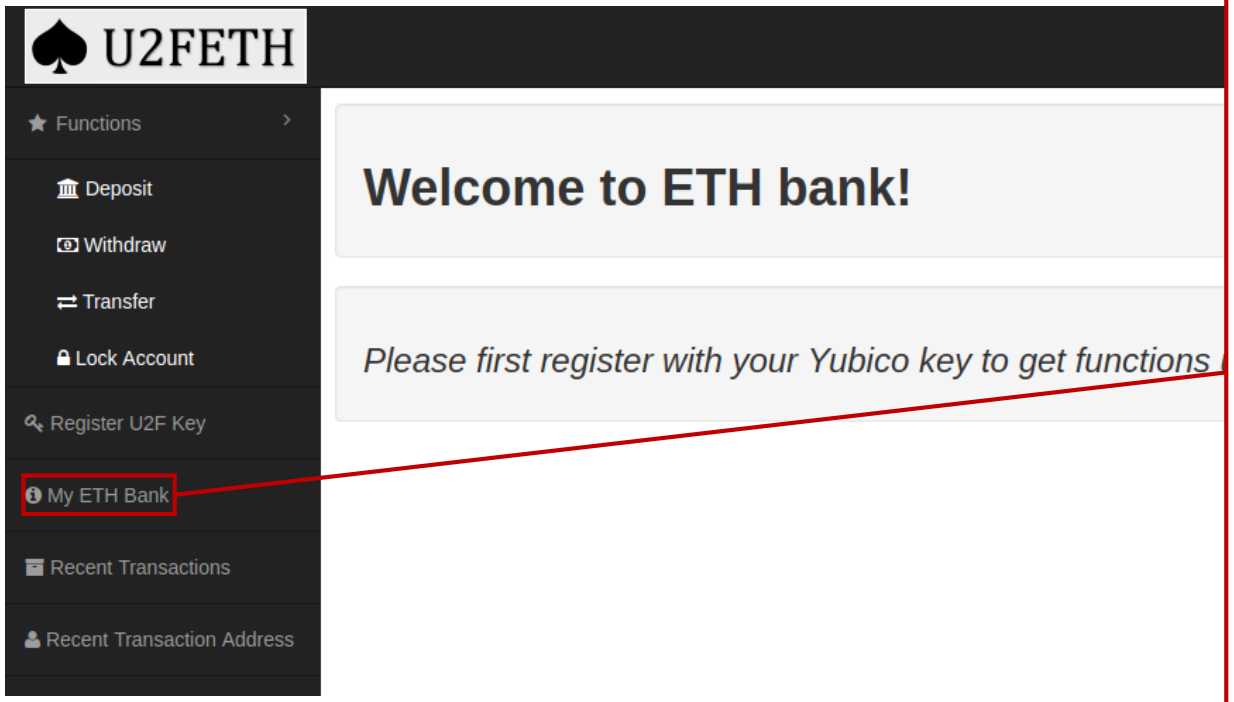


Unlock:



1. Functions are not available after locking the account.
2. Functions are available after unlocking the account on the same page.
3. Locking and unlocking account require U2F authentication.

ETH Bank: Information



1. Basic information about the bank account is displayed on this page.

Status

Owner	0xc7a8c34c4dd7faf0bbc801d7b40c2e8b8d0757fd
Bank	0x4242620e34d04737e7987ac5d5a9998e597352b7
Locked	false
Balance	20000 Wei

U2F

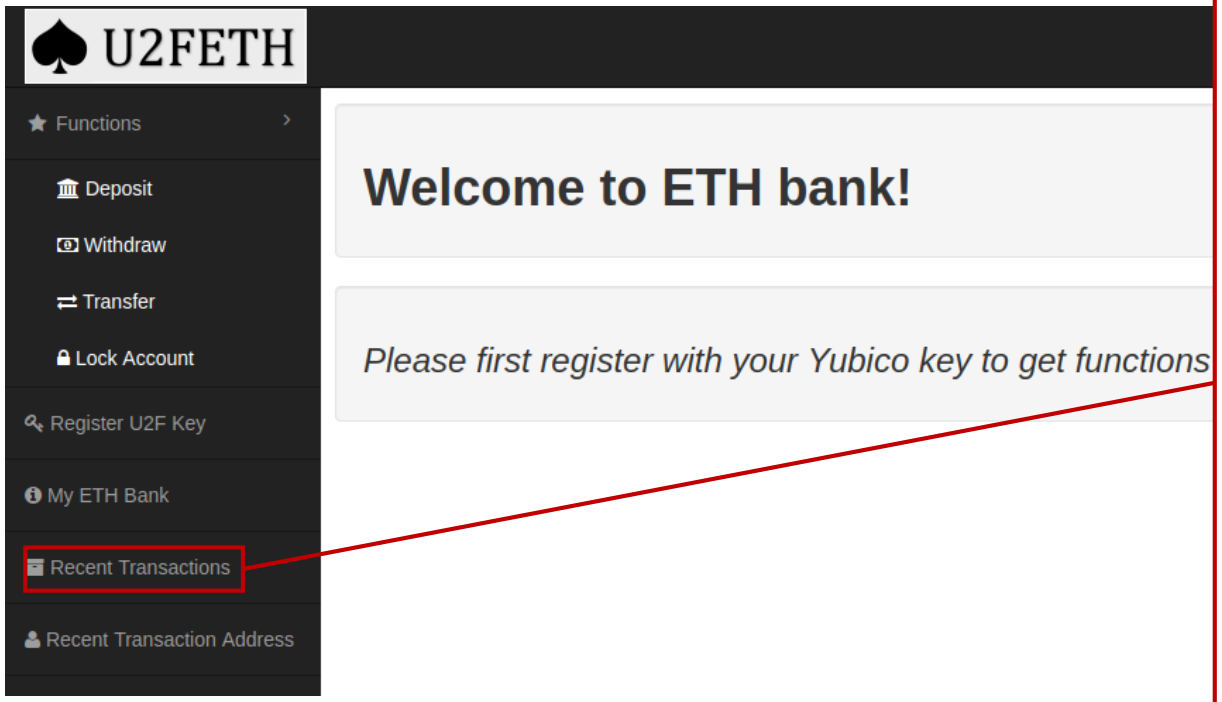
Registered	true
Transfer Limit	10000 Wei
Registered Key Info	U2F_V2 https://localhost:3000
Counter	112

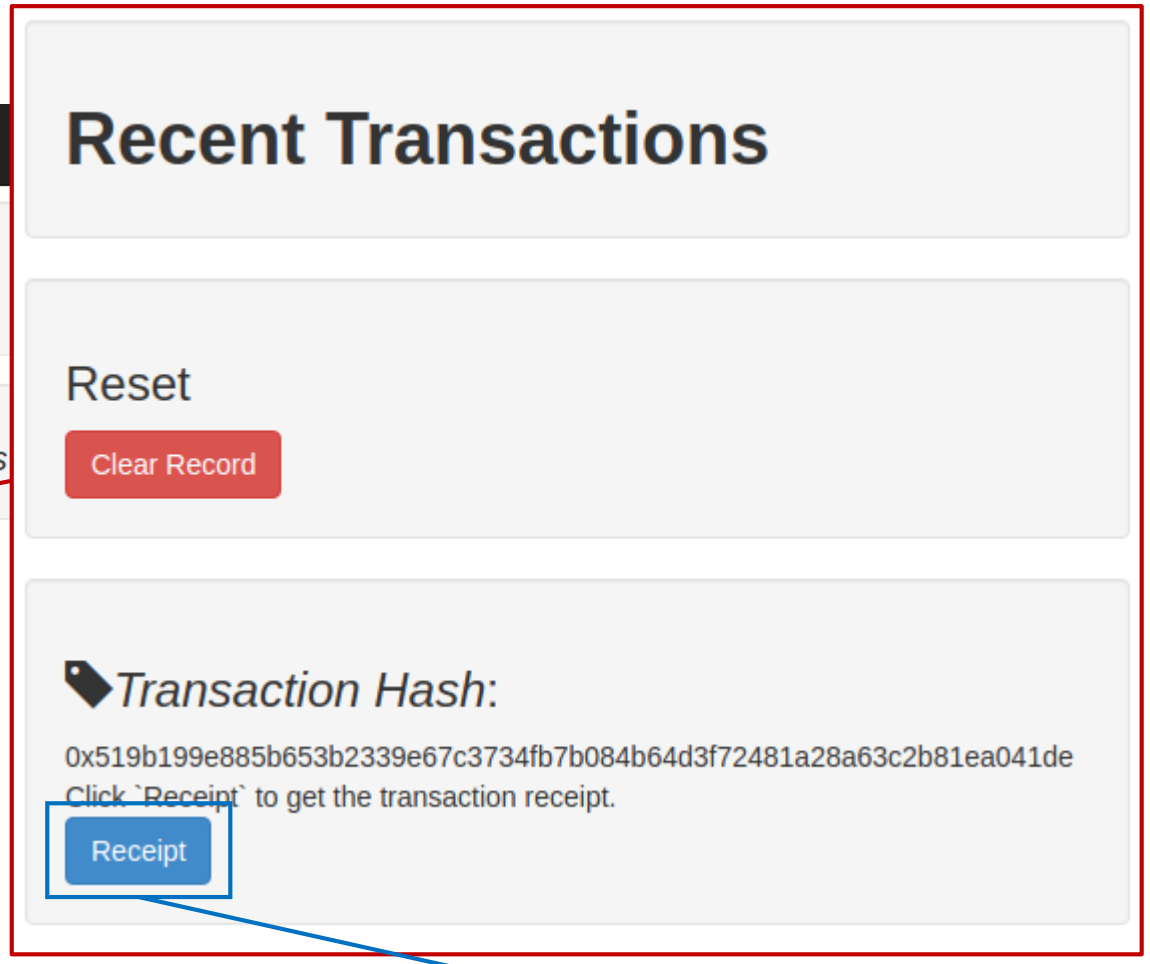
Transactions

Events	Count	Amount [Wei]
Deposit	1	20000
Withdraw	0	0
Transfer	0	0

accumulated amount

ETH Bank: Information

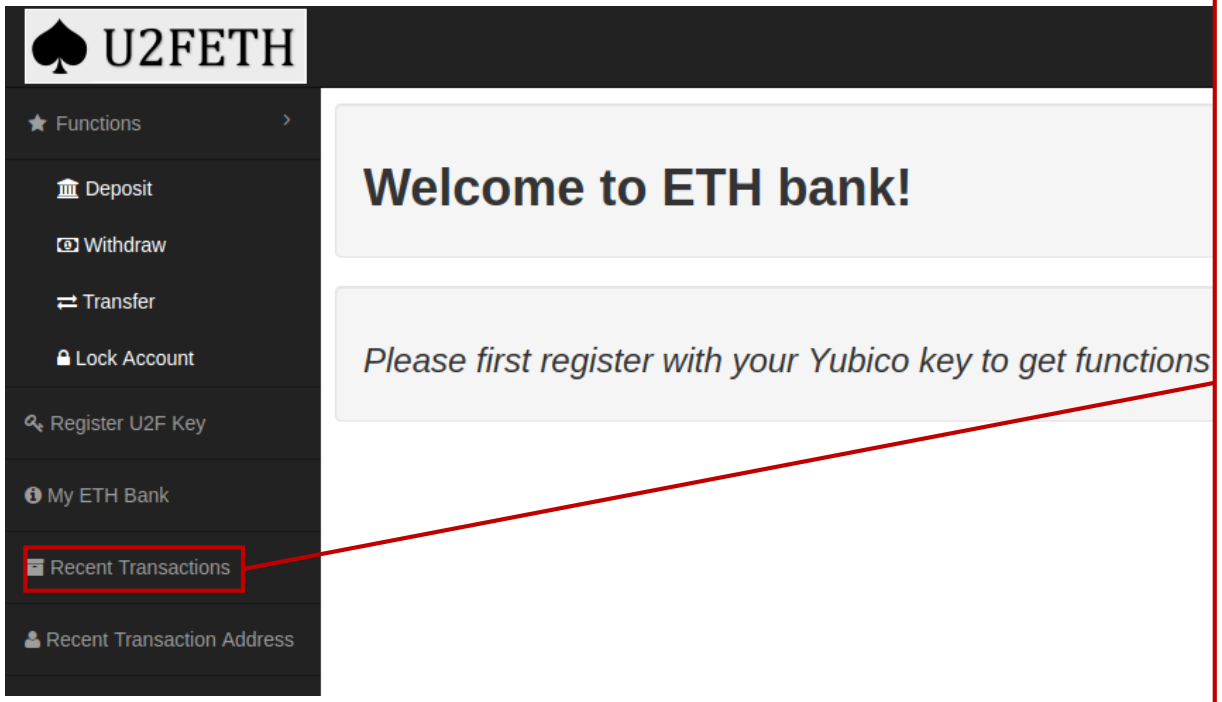


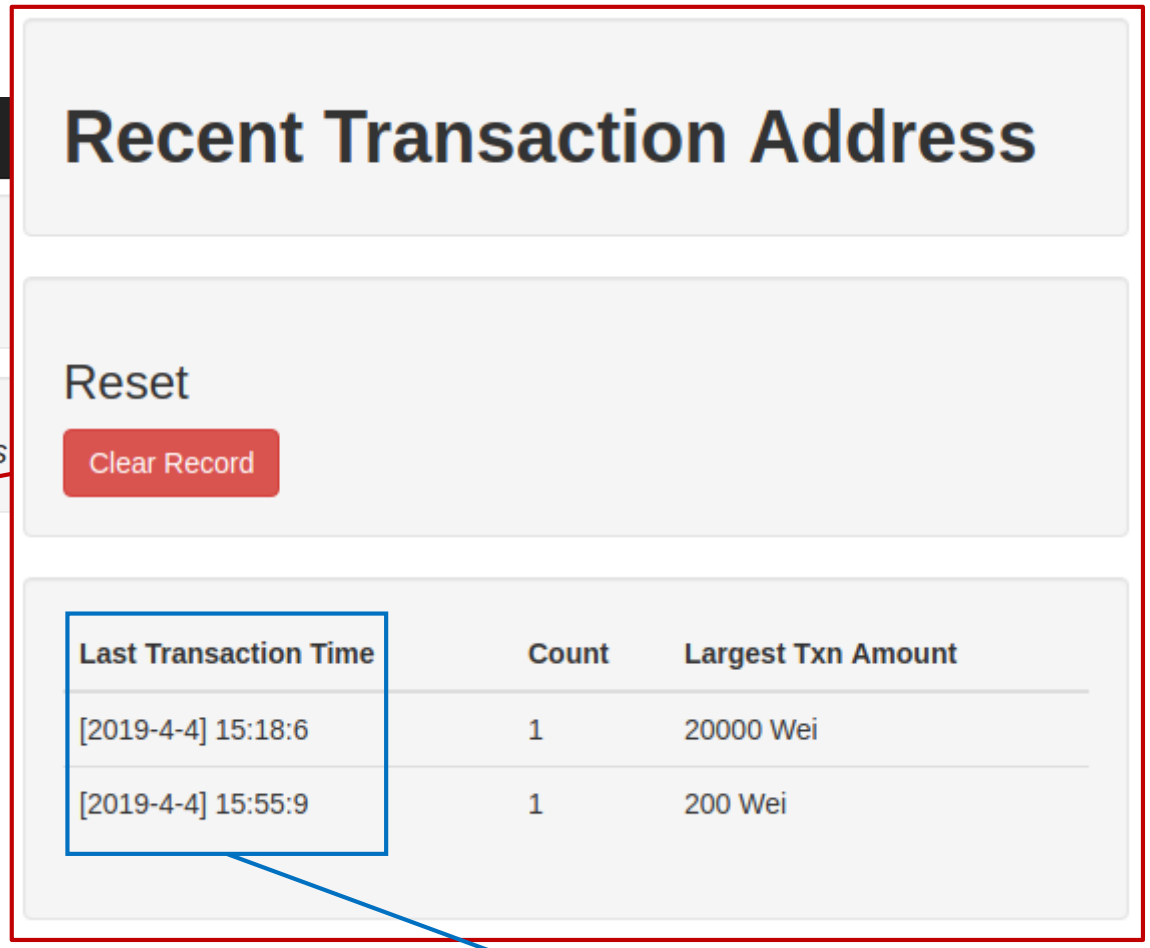


1. Transaction records are displayed in this page.

obtain receipt of transaction

ETH Bank: Information





1. Recent interacted addresses are displayed on this page.
2. The first row displays the interacted address with the largest transaction amount.

most recent txn timestamp

Discussion

Major Concerns

1. In verifying U2F registration, only public key is used instead of the X.509 certificate. There might be other information (e.g., valid from, valid by) that needs to be verified.
2. In verifying U2F authentication, parsing the signature data is not completed in Javascript but rather in smart contract. Moving this part out of the solidity contract (as what is done in U2F registration) might reduce gas consumption.

3. In the Bank login page, the password is currently stored in local storage. (Not yet researched about how to store users' passwords…)
4. Every time the user changes the address of the contract, a new U2F registration is requires. There are cases that the user has already registered in the contract level but is required to register again in the DApp level. (An argument against this fixation is that this is not what will occur in normal procedure, and might increase the security level of the user's account.

TODO

1. Move parsing data out of smart contract and test gas reduction.
2. Research password storage if necessary.
3. Test and improve user interface.

4. Explore other functions of bank.

Demonstration