COMP4901W – Final Project

NAME: SON Hangyul
EMAIL:hson@ust.hk
SID: 20537267

Assumptions

Logical Assumptions

- All calculations for this protocol are in the space of mod N.
- Each bank (address) is associated with an unique public key.
- Each user (address) can submit one signed message. In other words, one address can only mix 1 ether.
- Since this protocol is decentralized any potential user of this protocol can act as a bank.
- Users are available to approach an anonymous forum to exchange information or to ask for help.
- All participants of the smart contract are rational. Malicious actors will not act maliciously if they risk losing their own money. In other words, if they do not risk losing their money, they will do it.
- Each public key has an associated deadline and **cannot** be reused after the deadline.

Technical Assumptions

- Size of N is 'uint128' in order to prevent overflow.
- Public and Private Keys have size of uint256.
- Size of signed message is uint128, in accordance to the size of N. In order for the calculations fast exponentiation algorithm to work properly, **signed message must be smaller than N and is not '0'.**
- Therefore the degree of security of this protocol is capped with 2^{128}.

Protocol Procedure

Deposit

1. Contract gets deployed. This contract is named 'BlindedMixer'.
2. Anyone can play the role of a bank. To become a bank, one must upload their public key to the contract. Once the public key has been uploaded, the bank cannot modify the deposit deadline and withdraw deadline associated to it. Each deadline is given 1 week in a sequential order.
3. User can choose one of many public keys (denoted as 'e') available, and make a deposit into the BlindedMixer.
    1) Create a new address which would retrieve the money back from the Mixer after the deposit deadline.
    2) Choose a nonce (uint64) and append it to the new address.
    3) Take the hash of it with keccak256(). The result hash will be 32 bytes. However, the message will be of uint128 of size. Cast the hash to uint128.
    4) Choose a random number 'r', which is gcd(r,N) = 1;
    5) Encode 'r' with 'e'. Hide the message by multiplying it with r^e. The result of the encoded message should not be 0. If it is, choose a different nonce.
    6) Make 1 ETH deposit and submit the encoded message declaring which public key has been made of use.

$$uint128\ encodedMessage = (message * r^e)\ mod\ N.$$

4. After submission, the user must wait until the public key owner submit the signed hash to the contract, $m^d * r$. (message will be denoted as 'm')
    - The owner of the public key must make 1 ETH deposit to the contract for each submission of signed message to prevent any malicious activity. The contract keeps track of how much deposit in total has been made by the public key.
    - The number submitted signatures will be tracked by public key basis.

Potential Deposit Security Issues

- If the bank fails to submit the signed message for the user, user can get a refund for the deposit after the deposit deadline.
- The bank and user can have same real-life identity. After user receives the refund, prevent the bank from signing the refunded message. Each message from a user can only be either signed or refunded. The deposit deadline enforces this security issue.
- The contract will verify whether or not the sign is valid. If the re-encrypting with the public key is equal to the value of original submitted hash, then we can assume the signature is valid. $(m^d * r)^e == (\ m\ * r^e) == encodedMessage$
- The bank cannot submit a signed message which has not been registered with 1 ether deposit.
- The contract ensures that the bank assign the signed message to the user whom actually submitted the encoded message.
- User should not deposit more than once with the same address. Ultimately, only one signed message will be assigned to her for this contract's lifetime.

Withdraw

1. Using the address that has which a part of the message, user can withdraw from contract. The withdrawal should be done within 1 week from the deposit deadline.
2. The user must submit the receiver address, nonce, public key 'e', and m^d. Given that the public key decrypts m^d and its value matches the uint128 casted version of the hash of address concatenated with a nonce, the contract allows the withdrawal to the submitted address. Also, the contract tracks the number of withdrawal related to each public key.

$$(m^d)^e = uint128(uint256\left(keccak256(abi.encodepacked(address, nonce))\right))$$

3. The public key owner (bank) receives 0.01 ether commission for his contribution to the contract.
4. Given that the new account does not have the gas to execute withdrawal function call, one can make a post to an anonymous forum and upload the address, nonce, public key, and m^d and ask for help. The smart contract checks whether the "msg.sender" of the function matches the submitted address. If there is a mismatch, 0.01 ether is given towards the function "msg.sender" as a commission. The rest of the amount is transferred to the address, to the original user.
5. After the withdraw deadline, the bank can get the refund for the deposit made for each and every submitted signed messages.

Potential Withdraw Security Issues

- If the user has a valid signed message, the user is ensured to withdraw regardless of the current state of the bank. The public key and the corresponding 'N' are saved at a contract level and thus can validate the signed message.
- If the number of the withdrawal for the public key exceeds the number of the number of sent signatures with the public key, this implies that the public key owner(bank) has forged a false deposit signed by oneself. In this case the bank cannot get the refund that has been made as the bank signs the message, in other words deposit is confiscated by the smart contract. As the deposit amount is at least as large as, or could be lot larger than the amount in which bank can steal by issuing more bank notes. Therefore, the bank would do no such malicious activity as it the risks are higher than the potential gain. The bank must have also made use of the gas fees to send the signed hash and to withdraw, which makes the bank even more irrational to do such malicious activity.
- Each unique value of m^d will be tracked by the contract. No user can withdraw the deposit twice and steal other user's money unless they have several unique value of m^d. Using this protocol, the same address can be assigned 1 ETH several times, if several deposits have been made to the address. At the same time, privacy is maintained because hashes are not revealed.
- The agent who helps the user receive money without gas cannot connect the relationship between the deposit address and the receiver address because the m^d and m has not been revealed in the deposit state. Only $encodedMessage = m^d * r$ has been revealed.
- The helper cannot receive more than the commission because the address is hardcoded in the process of creating the hash. If the receiver address changes, it means that the decrypted message and hash of address and nonce would no longer match.

- Any helper who are willing to receive the commission can execute the function which means that there is very small risk of ETH being locked up in the smart contract until the deadline.
- All transfers of ether is done through _address.call{value: _value, gas: 2300}(“”) in order to prevent any malicious exceptions inside the withdraw function. The gas limit has also been set in order to prevent out of gas errors that could occur. All methods are prevented for any reentrancy attacks.

Conclusion

- BlindedMixer help detach the connection of addresses and past transaction records.
- The contract is decentralized. No central power.
- Withdraw of ether without gas is possible but with a fee.
- The contract is safe from all security vulnerabilities assuming users are rational.