Ethereum decentralized mixing service (codename Laundromat)

Laundromat is an Ethereum smart contract deployed by one of the mixing initiators.
Laundromat is using ring signatures to detach the sender from the recipient of the transfered funds.
Laundromat is using the withdraw address to create the ring signature. It prevents the forging and frontrunning of the redeem request.
Laundromat requires a predefined amount of participants and predefined amount of Ether from each participant.
Laundromat has a timeout to initiate a mix. After the timeout no mix is possible and all participants can take their funds back.

Laundromat API:

Laundromat has a constructor with mixing settings: amount of participants, funds denominator (how much Ether required from each participant), timeout.

deposit(uint pubkey1, uint pubkey2):
Accepts Ether deposit from an account and add it to participants list. pubkey1 and pubkey2 are the 2 halves of 64 byte public key of the participant.

withdraw(uint[] signature, uint x0, uint Ix, uint Iy):
Try to withdraw funds by the sender using the ring signature. Signature should be valid (checked against public keys of the participants), signature should be created using the withdraw account address (prevents frontrunning). x0 is the first element in the ring. Ix and Iy is a part of the private key - it does not disclose neither private nor public key but gurantees no double spend will be possible.

Problems:

1. Gas limit. Verifying a signature uses a large amount of computations and likely hit gas limit. To prevent this the witdhraw call can be break to witdhraw1, withdraw2,... subsequent calls. Sequence should be maintained by a finite state machine using local contract storage.
2. Stack limit. Verifying a signature will likely hit EVM stack limit of 16 32 bytes local variables. It is possible to avoid it using contract storage (requires additional for storing variables) and will likely be a smaller issue when using a finite state machine.
3. Debugging. Use pyethereum python project to verify against working solution: https://github.com/ethereum/pyethereum
4. Checking contract validity. The mixing should be organized by an app which will verify the contract validity.
5. Complexity of usage. The mixing should be organized by an app to hide the inner complexity and guide the user.

Use case:

1. Alice decides to mix 100 ether.
2. Alice looks for existing mixing contracts.
3. Alice finds an existing 10 ether mixing contract.
4. Alice joins this contract (and sends 10 ether here) and creates own for 90 ether.
5. Alice find that the 10 ether contract has full amount of participants. She can mix her 10 ether.
6. Alice creates new Ethereum account and tops it up with 1 ether using Shapeshift.
7. Alice uses the new account to withdraw 10 ether. Now her 10 ether cannot be traced back to originate account.
8. The deposit and withdraw procedure should be using the same private key so key safety should be ensured until withdrawing. The key is onetime so it can be dropped safely after the redeem.

References:

1. A PoC mixing contract: https://github.com/ethereum/pyethereum/blob/serenity/ethereum/ringsig.se.py
2. Elliptic curve ariphmetic: https://github.com/ethereum/serpent/blob/develop/examples/ecc/jacobian_arith.se
3. EIP 102 proposal for elliptic curve arithmetics opcodes: https://github.com/ethereum/EIPs/issues/102
4. EIP 101 proposal for bigint ariphmetics: https://github.com/ethereum/EIPs/issues/101
5. Deployed contract for EC arithmetic (source code attached): https://etherscan.io/address/0x600ad7b57f3e6aeee53acb8704a5ed50b60cacd6#code
6. Deployed mixing contract with 3 participant for 1000 wei each (source code attached): https://etherscan.io/address/0x401e28717a6a35a50938bc7f290f2678fc0a2816#code
7. Public repository for Laundromat project: https://github.com/blackyblack/laundromat