Towards practical post quantum stealth addresses

Stealth addresses

are a type of privacy-enhancing technology used in cryptocurrency transactions. They allow users to send and receive cryptocurrency without revealing their public addresses to the public ledger.

In a typical cryptocurrency transaction, a sender must reveal their public address to the receiver, as well as to anyone who may be monitoring the blockchain. This can compromise the user's privacy and security, as it allows others to link their transactions and potentially track their funds.

With stealth addresses, however, the sender generates a unique, one-time public address for each transaction, which is not linked to their permanent public address on the blockchain. The receiver can still receive the funds to their permanent public address, but only they can link the stealth address to their own address and access the funds.

Stealth addresses provide an additional layer of privacy and security to cryptocurrency transactions, making it more difficult for third parties to track and monitor user activity on the blockchain.

You can read about it in a recentVitalik's post.

In this post we are going to analyze a possible Post Quantum version of stealth addresses based on Commutative Supersingular isogenies (CSIDH).

N.B

. If you wonder if this solution is affected by the new devastating attacks on SIDH the answer is NO

. They crucially relies on torsion point information that are not present in CSIDH based solutions.

Stealth addresses with elliptic curve cryptography

Recapping from Vitalik's post

- Bob generates a key m

, and computes M = mG

, where G

is a the generator point for the elliptic curve. The stealth meta-address is an encoding of M

.

- Alice generates an ephemeral key r

, and publishes the ephemeral public key R = rG

.

- Alice can compute a shared secret S = rM

, and Bob can compute the same shared secret S = mR

.

- To compute the public key, Alice or Bob can compute P = M + hash(S)G
- To compute the private key for that address, Bob (and Bob alone) can compute p = m + hash(S)

This is translated in sage's code:

Bob

private

m = ZZ.random_element(n)

public

M = m*G

Alice

private

r = ZZ.random_element(n)

publish

R = r*G Sa = r * M s = '' s+=str(R[0]) s+=str(R[1]) s+=str(Sa[0]) s+=str(Sa[1]) h.update(s.encode())

hashS = (int(h.hexdigest(), 16)) % n Pa = M + hashS*G

Bob

Sb = m*R Pb = M + hashS*G* p = m+hashS

assert Sa == Sb assert Pa == Pb == p*G

Commutative Supersingular isogenies (CSIDH).

This section (and the remainder of the post) will require some knowledge about elliptic curves and isogeny based cryptography. The general reference on elliptic curves is Silverman for a thorough explanation of isogenies we refer toDe Feo.

CSIDH is an isogeny based post quantum key exchange presented atAsiacrypt 2018 based on an efficient commutative group action. The idea of using group actions based on isogenies finds its origins in the now well known 1997 paper by Couveignes. Almost 10 years later Rostovtsev and Stolbunovrediscovered Couveignes's ideas .

Couveignes in his seminal work introduced the concept of Very Hard Homogeneous Spaces

(VHHS). A VHHS is a generalization of cyclic groups for which the computational and decisional Diffie-Hellman problem are hard. The exponentiation in the group (or the scalar multiplication if we use additive notation) is replaced by a group action on a set. The main hardness assumption underlying group actions based on isogenies, is that it is hard to invert the group action:

Group Action Inverse Problem (GAIP).

Given a curve E

, with End(E) = O

, find an ideal a $\subset$ O such that E = [a]E_0

.

The GAIP (also known as vectorization

) might resemble a bit the discrete logarithm problem and in this post we exploit this analogy to translate the stealth addresses to the CSIDH setting.

Stealth addresses with CSIDH

In this section we will show an (almost) 1:1 stealth addresses translation from the DLOG setting to the VHHS setting:

- Bob generates a key m

, and computes E_m = [m]E_0

, where E_0

is a the starting elliptic curve. The stealth meta-address is an encoding of E_m

.

- Alice generates an ephemeral key r

, and publishes the ephemeral public key E_r = [r]E_0

.

- Alice can compute a shared secret E_S = [r]E_m

, and Bob can compute the same shared secret E_S = [m]E_r

.

- To compute the public key, Alice or Bob can compute P = [hash(E_S)]E_m
- To compute the private key for that address, Bob (and Bob alone) can compute p = [m + hash(S)]

Here is the relevant sage snippet (here the full code)

Bob

private

m = private()

public

M = action(base, m)

private

r =private()

publish

R = action(base, r) Sa = action(M, r)

s = '' s += str(R) s += str(Sa) h.update(s.encode()) hashS = (int(h.hexdigest(), 16)) % class_number hashS_reduced = reduce(hashS,A,B)

P = action(M,hashS_reduced)

Bob

Sb = action(R, m) pv = [] for i,_ in enumerate(m): pv.append(m[i]+hashS_reduced[i])

assert Sa == Sb assert P == action(base,pv)

Acknowledgement

Thanks to Vitalik Buterin, Luciano Maino, Michele Orrù and Mark Simkin for fruitful discussions and comments.