Here’s the code for elliptic curve signature recovery, copied from pybitcointools, and cleaned of details that are irrelevant to our exposition

```
def ecdsa_raw_recover(msghash, vrs): v, r, s = vrs y = # (get y coordinate for EC point with x=r, with same parity as v) Gz =
jacobian_multiply((Gx, Gy, 1), (N - hash_to_int(msghash)) % N) XY = jacobian_multiply((r, y, 1), s) Qr = jacobian_add(Gz,
XY) Q = jacobian_multiply(Qr, inv(r, N)) return from_jacobian(Q)
```

Suppose that we feed in msghash=0, and s=r*k for some k. Then, we get:

- Gz = 0
- XY = (r,y) * r * k
- Qr = (r,y) * r * k
- Q = (r, y) * r * k * inv(r) = (r, y) * k

Hence, the elliptic curve signature feeds out k

times the point (r, y)

, where r

and k

are values you feed in, and you specify the parity of y

via v

.

Unfortunately, the secp256k1 precompile outputs the truncated hash of Q, and not Q itself. But you can get around that by requiring the function caller to submit Q as a witness. The extra cost of 64 bytes of data is ~4000 gas, so altogether you can get an ECMUL with <10k gas, under a quarter of what you need if you use the ECMUL precompile that uses the alt_bn128 curve.

This could be used as a ghetto hack to optimize ring signatures in the EVM today, as well as other applications.