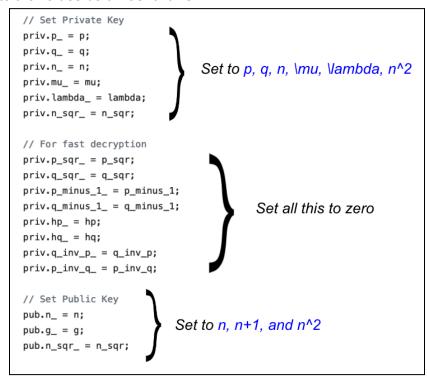
STEP 1: Inject Paillier modulus

Choose 16 small primes p_1 , p_2 , ..., p_{16} of size greater than 2^{16} Set n=p+q where $p=p_1\cdot p_2\cdots p_{16}$ and q is a big prime to match the expected size of nSet $\lambda=(p_1-1)\cdot(p_2-1)\cdots(p_{16}-1)\cdot(q-1)$ and $\mu=\lambda^{-1} \mod n$

Bypass the following in round0.cpp of the GG18 keygen

Populate the values below as follows



STEP 2: Choose k and cheat in the range proof

For i = 1 ... 16 do

(the following is iterated 16 times for separate signing sessions)

• Bypass the following in round 0. cpp of the GG20 sign (populate k=0)

```
// Sample gamma, k in Z_q

ctx->local_party_.gamma_ = safeheron::rand::RandomBNLt(curv->n);

ctx->local_party_.k_ = safeheron::rand::RandomBNLt(curv->n);

ctx->local_party_.Gamma_ = curv->g * ctx->local_party_.gamma_;
```

Bypass the following in round0.cpp of the GG20 sign

When running the Prove above: while e \mod p_i \neq 0, do
 Update \gamma = \gamma + 1 and w := w*h2 % N_tilde

```
// w = h1^alpha * h2^gamma mod N_tilde
w_ = ( h1.PowM(alpha, N_tilde) * h2.PowM(gamma, N_tilde) ) % N_tilde;
```

. . .

```
sha256.Finalize(sha256_digest);
BN e = BN::FromBytesBE(sha256_digest, sizeof(sha256_digest));
e = e % q;
```

STEP 3: Extract key material

• When obtaining \alpha below (before taking mod the curve) in round2.cpp, do: $x_i = (\alpha - (\alpha \setminus mod N/p_i))/(N/p_i)$

Once you have all the x_i 's, reconstruct the key using CRT.

STEP 4 (Optional): Make it sign

• When obtaining \alpha below (for both alpha's) in round2.cpp, do: α : = $\alpha \setminus mod N/p_i$