Code

- Montgomery modular arithmetic
- Requires r, n'
- r: while loop or
- n': Extended Euclidian Algortithm
- Tested with Python
- Compared/verified with regular algorithm
- Significant speed increase

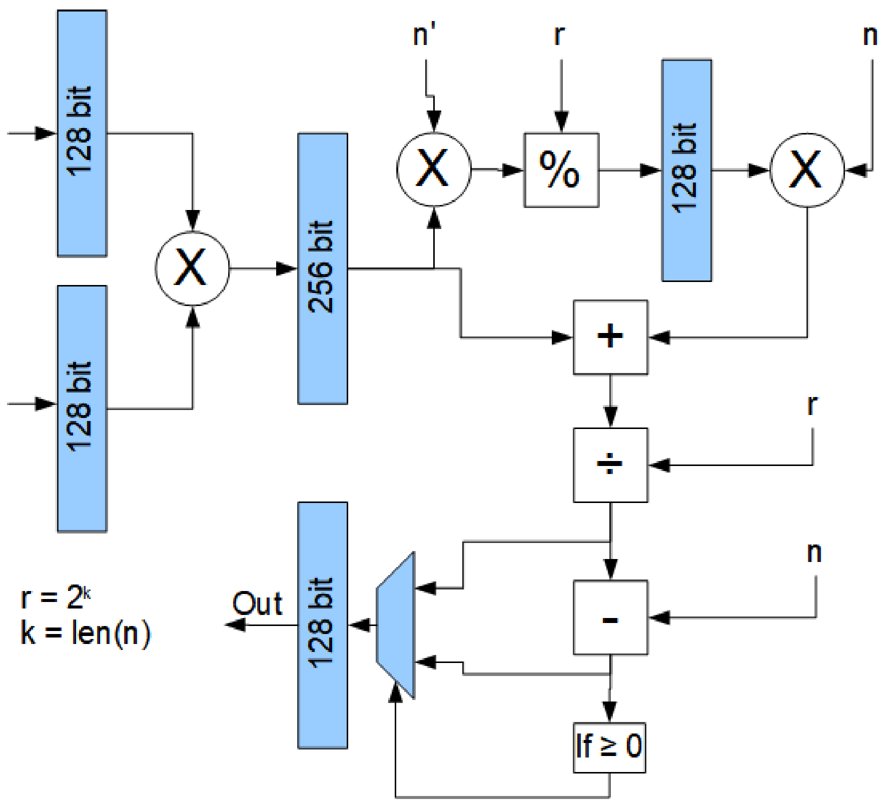
function ModExp(M, e, n) { n is an odd number }
Step 1. Compute n' using the extended Euclidean algorithm.
Step 2. $\bar{M} := M \cdot r \bmod n$
Step 3. $\bar{x} := 1 \cdot r \bmod n$
Step 4. **for** $i = k-1$ **down to** 0 **do**
Step 5. $\bar{x} := \text{MonPro}(\bar{x}, \bar{x})$
Step 6. **if** $e_i = 1$ **then** $\bar{x} := \text{MonPro}(\bar{M}, \bar{x})$
Step 7. $x := \text{MonPro}(\bar{x}, 1)$
Step 8. **return** x

function MonPro$(\bar{a}, \bar{b})$
Step 1. $t := \bar{a} \cdot \bar{b}$
Step 2. $m := t \cdot n' \bmod r$
Step 3. $u := (t + m \cdot n)/r$
Step 4. **if** $u \geq n$ **then return** $u - n$
else return u

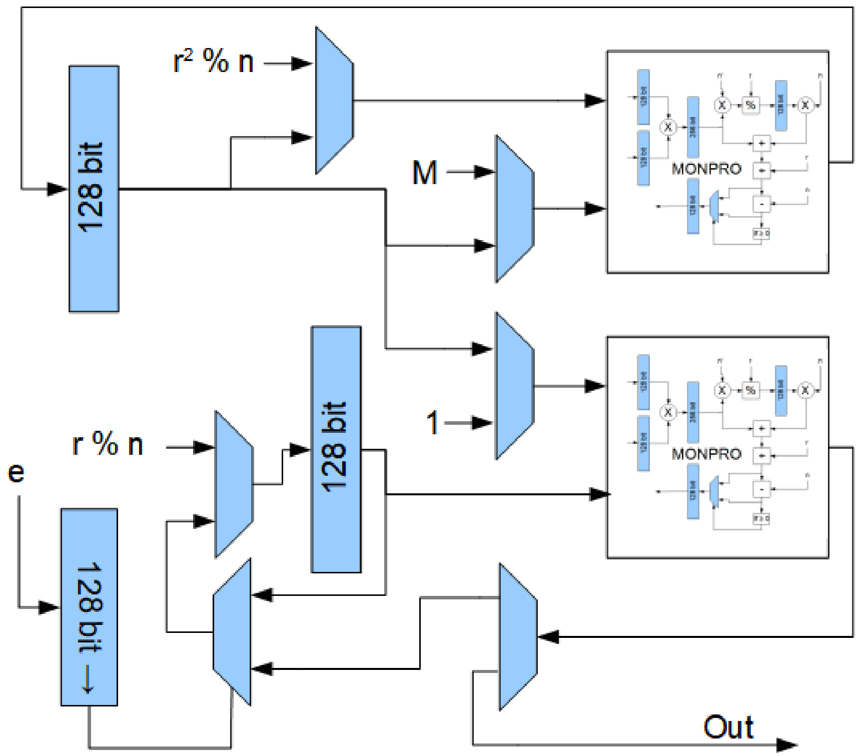
```python
def MonPro(a_strek, b_strek, n, n_merket, r):

    t = a_strek * b_strek
    m = (t * n_merket) % r
    u = (t+(m*n))/r

    if u >= n:
        return u-n
    return u


def ModExp(message, e, n):

    r = 2 ** len(bin(n)[2:]) # r = (r mod n) + n
    n_merket = -extended_gcd(r,n)

    #NOTE! "r mod n" and "r*r mod n" is given in the exercise
    M_strek = MonPro(message,(r*r) % n,n,n_merket,r)
    x_strek = r % n

    for i in bin(e)[2:][::-1]:
        if i == '1':
            x_strek = MonPro(M_strek,x_strek,n,n_merket,r)
        M_strek = MonPro(M_strek,M_strek,n,n_merket,r)

    return MonPro(x_strek,1,n,n_merket,r)


def extended_gcd(a,b):
    t = 1; oldt = 0
    r = b; oldr = a
    while r != 0:
        quotient = oldr / r
        (oldr, r) = (r, oldr - quotient*r)
        (oldt, t) = (t, oldt - quotient*t)
    return oldt
```

Blocks

MONPRO



MODEXP



The case of even modulo

- Special case
- N = even
- Use Chinese Remainder Theorem
- Requires extra functions

```python
if n % 2 == 0:
    binret=BinSplit(n)
    j = binret[0]
    q = binret[1]
    x1 = ModExp(message, e, q)
    x2val = 2**j
    x2_1 = message % x2val
    x2_2 = e % 2**(j-1)
    x2 = BinExp(x2_1, x2_2, x2val)
    q_inv = ModInverse(q,j)
    y = (x2 - x1)*q_inv % x2val
    x = x1 + q*y
    return x
```

ModInverse