Tonelli-Shanks Algorithm

Alex V. Hagdahl

January 2023

Quadratic Residue of a Prime

Let p be an odd prime and $\mathbb{F}_p = \{0,\ \ldots,\ p-1\}$ be the field of p elements. Operations and comparisons on elements of the multiplicative group of integers modulo $\mathbb{Z}/p\mathbb{Z}$ are implicitly mod p. An integer $n \in \mathbb{F}_p$ is called a quadratic residue of p if there exist $x \in \mathbb{F}_p$ such that $x^2 \equiv n$. In other words, n has a square-root in $\mathbb{F}_p$.

- If $n \equiv 0$ the equation $x^2 = 0$ is trivial.
- Half of the nonzero elements of $\mathbb{F}_p$ are quadratic residues, the other half are not. Hence there are $(p+1)/2$ of each.
- If we include 0 there are $(p+1)/2$ quadratic residues and $(p-1)/2$ non quadratic residues in $\mathbb{F}_p$.
- Suppose n is a non-zero quadratic residue. Then the equation $x^2 \equiv n$ has exactly two solutions. Say $x_1 = a \in \mathbb{F}_p$ is a solution. Then $x_2 = p - a$ is the other solution. Therefore if we know one square root we can easily calculate the other.

Euler's Criterion

There is a test to see if n is a quadratic residue or not. Define the *Legendre symbol.*

$$(n|p) \equiv n^{\frac{p-1}{2}}$$

The only values the Legendre symbol takes are $\{-1,\ 0,\ 1\}$ Note that -1 means $p-1$.

$$(n|p) = \begin{cases} 0 & \text{if } n \equiv 0 \\ 1 & \text{if } n \text{ is a non-zero quadratic residue} \\ -1 & \text{if } n \text{ is not quadratic residue} \end{cases}$$

The following python code is simple function to check if n is a quadratic residue mod p.

```python
def is_qudratic_residue(n, p):
    # The trivial case
    if(n % p == 0):
        return True
    # The Legendre Symbol
    return pow(n, (p - 1) // 2, p) == 1
```

Finding the modular square root

Given the two inputs p and n we want to try and find an integer $R \in \mathbb{F}_p$ such that $R^2 \equiv n$.

Case of $p \equiv 3 \pmod 4$

Given a non-zero quadratic residue n of p then a solution to $x^2 \equiv n$ is given by

$$x \equiv n^{\frac{p+1}{4}}$$

Proof: Since n is a non-zero quadratic residue we know by Euler's criterion that $n^{\frac{p-1}{2}} \equiv 1$. So if $x \equiv n^{\frac{p+1}{4}}$ then

$$x^2 \equiv n^{\frac{p+1}{2}} \equiv nn^{\frac{p-1}{2}} = n(1) = n$$

Given what we know we can construct the beginning of our function as follow

```
def tonelli_shanks(n, p):
    if not is_qudratic_residue(n, p):
        print("n is not a quadratic residue.")
        return None

    # Trivial case
    if(n % p == 0):
        print("n % p == 0 is a trivial quadratic residue 0.")
        return 0

    # Case p = 3 mod 4
    if(p % 4 == 3):
        print("n is a quadratic residue.")
        return pow(n, (p + 1) // 4, p)
```

The Tonelli-Shanks Algorithm, when $p \equiv 1 \pmod 4$

We need to find Q and S where Q is odd such that $p - 1 \equiv Q2^S$. This can be done by factoring out powers of 2 as follow

```
Q = p - 1
S = 0
while(Q % 2 == 0):
    S += 1
    Q //= 2
```

Then we need to search for a non quadratic residue z

```
z = 2
while is_qudratic_residue(z, p):
    z += 1
```

Initialize

$$M \leftarrow S$$
$$c \leftarrow z^Q$$
$$t \leftarrow n^Q$$
$$R \leftarrow n^{\frac{Q+1}{2}}$$

```
M = S
c = pow(z, Q, p)
t = pow(n, Q, p)
R = pow(n, (Q + 1) // 2, p)
```

Now we loop while $t \neq 1$ and use repeated squaring to find the least i such that $t^{2^i} = 1$ and initialize

$$b \leftarrow c^{2^{M-i-1}}$$

and let

$$M \leftarrow i$$

$$c \leftarrow b^2$$

$$t \leftarrow tb^2$$

$$R \leftarrow Rb$$

Once we've found the congruence $t \leftarrow tb^2 \equiv 1 \bmod p$ we return R.

```
while(t != 1):
    # Calculate the least i, 0 < i < M such that t^2^i = 1
    i = 0
    temp = t
    while temp != 1:
        i += 1
        temp = (temp * temp) % p

    # Calculate b, M, c, t, R
    b = pow(c, pow(2, M - i - 1), p)
    M = i
    c = (b * b) % p
    t = (t * b * b) % p
    R = (R * b) % p

# We have found the square root
return R
```

Elliptic Curves

The Tonelli-Shanks Algorithm is useful because it helps us find points on an elliptic curve given only the x coordinate. Consider an elliptic curve

$$y^2 = x^3 + ax + b$$

we set

$$n = x^3 + ax + b$$

and solve for

$$y^2 \equiv n$$

```
def get_y_value_of_elliptic_curve(a, b, p, x):
    n = (x * x * x + a * x + b) % p
    return tonelli_shanks(n, p)
```

Applying this to cryptography can save a lot of data during transmition of points over a network. Consider the curve NIST P-224 where the x and y coordinates are 224 bits each meaning it would require 448 bits to exchange a point. Instead we can send only the x value and use the Tonelli-Shanks Algorithm to calculate the y value locally. However the x coordinate is associated to two y coordinates so one extra bit is required to determine the sign

$$\text{Extra bit} = \begin{cases} 0 & \text{if } 0 \leq y < \frac{1}{2}p \\ 1 & \text{if } \frac{1}{2}p < y < p \end{cases}$$

Now only 225 bits is required to send a point. This is referred to as *point compression*.

Full code

```
def is_qudratic_residue(n, p):
    # The trivial case
    if(n % p == 0):
        return True
    # The Legendre Symbol
    return pow(n, (p - 1) // 2, p) == 1

def tonelli_shanks(n, p):
    if not is_qudratic_residue(n, p):
        print("n is not a quadratic residue.")
        return None

    # Trivial case
    if(n % p == 0):
        print("n % p == 0 is a trivial quadratic residue 0.")
        return 0

    # Case p = 3 mod 4
    if(p % 4 == 3):
        print("n is a quadratic residue.")
        return pow(n, (p + 1) // 4, p)

    # Now p = 1 mod 4
    # Step one: find Q and S such that p - 1 = Q(2^S)
    Q = p - 1
    S = 0
    while(Q % 2 == 0):
        S += 1
        Q //= 2

    # Step two: find a non quadratic residue z
    z = 2
    while is_qudratic_residue(z, p):
        z += 1

    # Step three: Initialize M, c, t, R
    M = S
    c = pow(z, Q, p)
    t = pow(n, Q, p)
    R = pow(n, (Q + 1) // 2, p)

    while(t != 1):
        # Calculate the least i, 0 < i < M such that t^2^i = 1
        i = 0
        temp = t
        while temp != 1:
            i += 1
            temp = (temp * temp) % p

        # Calculate b, M, c, t, R
        b = pow(c, pow(2, M - i - 1), p)
        M = i
        c = (b * b) % p
        t = (t * b * b) % p
        R = (R * b) % p
```

```
    # We have found the square root
    return R

def get_y_value_of_elliptic_curve(a, b, p, x):
    n = (x * x * x + a * x + b) % p
    return tonelli_shanks(n, p)
```