# Assignment 10: Digital Signature

Geetansh Juneja

November 10, 2021

## 1 RSA Algorithm

1. Select two prime numbers p and q.
2. Calculate n = p
3. Find Euler phi function $\phi(n)$ = (p-1)×(q-1).
4. Find e such that gcd(e,$\phi(n)$) = 1.
5. Find d such that d×e mod$\phi(n)$ = 1. The public key is the pair (n,e) and the private key is the pair (n,d).
6. To encrypt a message m first its hash h(m) is calculated and then the h(m) is broken into 32 bit integers on which RSA encryption function is applied i.e $M^d mod n$ where (n,d) is the private key. The encryption value of all these 32 bit integers are concatenated in such a way that after every encryption value of a integer there is '.'. This concatenated string is called the digital signature of the message m.
7. To decrypt the signature the value between two '.' is applied with RSA decryption function i.e $C^e mod n$ where (n,e) is the public key. All the characters corresponding to these values are concatenated . If this concatenated string is equal to the initial hash string then the message is valid.

# 2 Code

```
#include<iostream>
#include<cstring>
#include<string>
#include <bits/stdc++.h>
using namespace std;
#define uchar unsigned char
#define uint unsigned int

#define DBL_INT_ADD(a,b,c) if (a > 0xffffffff - (c)) ++b; a += c;
#define ROTLEFT(a,b) (((a) << (b)) | ((a) >> (32-(b))))
#define ROTRIGHT(a,b) (((a) >> (b)) | ((a) << (32-(b))))

#define CH(x,y,z) (((x) & (y)) ^ (~(x) & (z)))
#define MAJ(x,y,z) (((x) & (y)) ^ ((x) & (z)) ^ ((y) & (z)))
#define EP0(x) (ROTRIGHT(x,2) ^ ROTRIGHT(x,13) ^ ROTRIGHT(x,22))
#define EP1(x) (ROTRIGHT(x,6) ^ ROTRIGHT(x,11) ^ ROTRIGHT(x,25))
#define SIG0(x) (ROTRIGHT(x,7) ^ ROTRIGHT(x,18) ^ ((x) >> 3))
#define SIG1(x) (ROTRIGHT(x,17) ^ ROTRIGHT(x,19) ^ ((x) >> 10))

typedef struct {
    uchar data[64];
    uint datalen;
    uint bitlen[2];
    uint state[8];
} SHA256_CTX;

uint k[64] = {
    0x428a2f98,0x71374491,0xb5c0fbcf,0xe9b5dba5,0x3956c25b,0x59f111f1,0x923f82a4,0xab1c5ed5,
    0xd807aa98,0x12835b01,0x243185be,0x550c7dc3,0x72be5d74,0x80deb1fe,0x9bdc06a7,0xc19bf174,
    0xe49b69c1,0xefbe4786,0x0fc19dc6,0x240ca1cc,0x2de92c6f,0x4a7484aa,0x5cb0a9dc,0x76f988da,
    0x983e5152,0xa831c66d,0xb00327c8,0xbf597fc7,0xc6e00bf3,0xd5a79147,0x06ca6351,0x14292967,
    0x27b70a85,0x2e1b2138,0x4d2c6dfc,0x53380d13,0x650a7354,0x766a0abb,0x81c2c92e,0x92722c85,
    0xa2bfe8a1,0xa81a664b,0xc24b8b70,0xc76c51a3,0xd192e819,0xd6990624,0xf40e3585,0x106aa070,
    0x19a4c116,0x1e376c08,0x2748774c,0x34b0bcb5,0x391c0cb3,0x4ed8aa4a,0x5b9cca4f,0x682e6ff3,
    0x748f82ee,0x78a5636f,0x84c87814,0x8cc70208,0x90befffa,0xa4506ceb,0xbef9a3f7,0xc67178f2
};
```

```
void SHA256Transform(SHA256_CTX *ctx, uchar data[])
{
    uint a, b, c, d, e, f, g, h, i, j, t1, t2, m[64];

    for (i = 0, j = 0; i < 16; ++i, j += 4)
        m[i] = (data[j] << 24) | (data[j + 1] << 16) | (data[j + 2] << 8) | (data[j + 3]);
    for (; i < 64; ++i)
        m[i] = SIG1(m[i - 2]) + m[i - 7] + SIG0(m[i - 15]) + m[i - 16];

    a = ctx->state[0];
    b = ctx->state[1];
    c = ctx->state[2];
    d = ctx->state[3];
    e = ctx->state[4];
    f = ctx->state[5];
    g = ctx->state[6];
    h = ctx->state[7];

    for (i = 0; i < 64; ++i) {
        t1 = h + EP1(e) + CH(e, f, g) + k[i] + m[i];
        t2 = EP0(a) + MAJ(a, b, c);
        h = g;
        g = f;
        f = e;
        e = d + t1;
        d = c;
        c = b;
        b = a;
        a = t1 + t2;
    }

    ctx->state[0] += a;
    ctx->state[1] += b;
    ctx->state[2] += c;
    ctx->state[3] += d;
    ctx->state[4] += e;
    ctx->state[5] += f;
    ctx->state[6] += g;
    ctx->state[7] += h;
}
```

```
void SHA256Init(SHA256_CTX *ctx)
{
    ctx->datalen = 0;
    ctx->bitlen[0] = 0;
    ctx->bitlen[1] = 0;
    ctx->state[0] = 0x6a09e667;
    ctx->state[1] = 0xbb67ae85;
    ctx->state[2] = 0x3c6ef372;
    ctx->state[3] = 0xa54ff53a;
    ctx->state[4] = 0x510e527f;
    ctx->state[5] = 0x9b05688c;
    ctx->state[6] = 0x1f83d9ab;
    ctx->state[7] = 0x5be0cd19;
}

void SHA256Update(SHA256_CTX *ctx, uchar data[], uint len)
{
    for (uint i = 0; i < len; ++i) {
        ctx->data[ctx->datalen] = data[i];
        ctx->datalen++;
        if (ctx->datalen == 64) {
            SHA256Transform(ctx, ctx->data);
            DBL_INT_ADD(ctx->bitlen[0], ctx->bitlen[1], 512);
            ctx->datalen = 0;
        }
    }
}
```

```
void SHA256Final(SHA256_CTX *ctx, uchar hash[])
{
    uint i = ctx->datalen;

    if (ctx->datalen < 56) {
        ctx->data[i++] = 0x80;

        while (i < 56)
            ctx->data[i++] = 0x00;
    }
    else {
        ctx->data[i++] = 0x80;

        while (i < 64)
            ctx->data[i++] = 0x00;

        SHA256Transform(ctx, ctx->data);
        memset(ctx->data, 0, 56);
    }

    DBL_INT_ADD(ctx->bitlen[0], ctx->bitlen[1], ctx->datalen * 8);
    ctx->data[63] = ctx->bitlen[0];
    ctx->data[62] = ctx->bitlen[0] >> 8;
    ctx->data[61] = ctx->bitlen[0] >> 16;
    ctx->data[60] = ctx->bitlen[0] >> 24;
    ctx->data[59] = ctx->bitlen[1];
    ctx->data[58] = ctx->bitlen[1] >> 8;
    ctx->data[57] = ctx->bitlen[1] >> 16;
    ctx->data[56] = ctx->bitlen[1] >> 24;
    SHA256Transform(ctx, ctx->data);

    for (i = 0; i < 4; ++i) {
        hash[i] = (ctx->state[0] >> (24 - i * 8)) & 0x000000ff;
        hash[i + 4] = (ctx->state[1] >> (24 - i * 8)) & 0x000000ff;
        hash[i + 8] = (ctx->state[2] >> (24 - i * 8)) & 0x000000ff;
        hash[i + 12] = (ctx->state[3] >> (24 - i * 8)) & 0x000000ff;
        hash[i + 16] = (ctx->state[4] >> (24 - i * 8)) & 0x000000ff;
        hash[i + 20] = (ctx->state[5] >> (24 - i * 8)) & 0x000000ff;
        hash[i + 24] = (ctx->state[6] >> (24 - i * 8)) & 0x000000ff;
        hash[i + 28] = (ctx->state[7] >> (24 - i * 8)) & 0x000000ff;
    }
}
```

```
string SHA256(char* data) {
    int strLen = strlen(data);
    SHA256_CTX ctx;
    unsigned char hash[32];
    string hashStr = "";

    SHA256Init(&ctx);
    SHA256Update(&ctx, (unsigned char*)data, strLen);
    SHA256Final(&ctx, hash);

    char s[3];
    for (int i = 0; i < 32; i++) {
        sprintf(s, "%02x", hash[i]);
        hashStr += s;
    }

    return hashStr;
}
int power(long long x, unsigned int y, int p)
{
    int res = 1;

    x = x % p;

    if (x == 0) return 0;

    while (y > 0)
    {
        if (y & 1)
            res = (res*x) % p;

        y = y>>1;
        x = (x*x) % p;
    }
    return res;
}
```

```
int main(){
    char data[] = "2101-CON101 INTRODUCTION TO COMP.SC. amp; ENG";
    string sha256 = SHA256(data);
    int p,q,e;
    scanf("%d%d%d",&p,&q,&e);
    int n = p*q;
    int phi = (p-1)*(q-1);
    int d;
    for(int i=2;;i++){
        if((i*phi+1)%e == 0) {
            d= (i*phi+1)/e;
            break;
        }
    }
    string cipher = "";
    for(int i=0;i<=sha256.length()-1;i++){
        char s1 = sha256.at(i);
        int m = (int)s1;
        int temp = power(m,d,n);
        cipher = cipher + to_string((int)temp)+".";

    }
    string decipher = "",t="";
    for(int i=0;i<=cipher.length()-1;i++){
        if(cipher.at(i)=='.'){
            int r = stoi(t);
            decipher = decipher + (char)power(r,e,n);
            t = "";
        }
        else t = t + cipher.at(i);
    }
    cout<<"Euler's phi function = "<<phi<<endl;
    cout<<"Public key("<<n<<","<<e<<")\n";
    cout<<"Public key("<<n<<","<<d<<")\n";
    cout<<"Digest of message "<<sha256<<endl;
    cout<<"Signature "<<cipher<<endl;
    cout<<"encryption generated by bob "<<decipher<<endl;
    return 0;
}
```

## 3 Output of Example in Assignment Sheet

```
geetansh@Asusvivobook:~/Documents/C++$ g++ main.cpp
geetansh@Asusvivobook:~/Documents/C++$ ./a.out
43 47 155
Euler's phi function = 1932
Public key(2021,155)
Public key(2021,1583)
Digest of message ad764bfc84abe5d490979675de9c318cfeaafa2e46ca1f9
Signature 441.289.1061.1516.668.244.403.339.2016.668.441.244.153.
96.403.547.1061.1513.244.403.847.1135.403.2016.153.1061.547.1135.
encryption generated by bob ad764bfc84abe5d490979675de9c318cfeaaf
```