Mapping and Recreating Digital Signature Algorithms Using MATLAB

Elizabeth Wright

INTRODUCTION

A digital signature (DS) is a mathematical scheme for demonstrating the authenticity of a digital message or document. A valid digital signature gives a recipient reason to believe that the message was created by a known sender, such that the sender cannot deny having sent the message and that the message was not altered in transit. DS are commonly used for software distribution, financial transactions, and in other cases where it is important to detect forgery or tampering [1].

EXPLANATION

DS are often used to implement electronic signatures, a broader term that refers to any electronic data that carries the intent of a signature, but not all electronic signatures use digital signatures.

DS employ asymmetric cryptography. In many instances they provide a layer of validation and security to messages sent through a nonsecure channel: properly implemented, a digital signature gives the receiver reason to believe the message was sent by the claimed sender. Digital seals and signatures are equivalent to handwritten signatures and stamped seals. DS are equivalent to traditional handwritten signatures in many respects, but properly implemented digital signatures are more difficult to forge than the handwritten type. DS can also provide non repudiation, meaning that the signer cannot successfully claim they did not sign a message, while also claiming their private key remains secret; further, some non-repudiation schemes offer a time stamp for the digital signature, so that even if the private key is exposed, the signature is valid. Digitally signed messages may be anything representable as a bitstring: examples include electronic mail, contracts, or a message sent via some other cryptographic protocol.

Technology of DS applying assumes that we have a network of subscriber, sending signed electronic documents each other. The pair of keys is generated for every subscriber – an open key and a close key. Close key is kept secret by the abonent and is used for generating of DS. Open key is known for all users and is intended for the DS checking by the addressee of electronic document. In other words, open key is necessary tool for checking authority and authenticity of the document. Open key does not allow calculating a secret one.In particular: Alice wants to sign message ***m***. She computes the signature of ***m*** (let's call it ***S***) and sends the signed message (***m,S)*** to Bob. Bob gets ***(m,S)***, runs the verification algorithm on it. The algorithm returns "true" if ***S*** is Alice's signature of ***m***.

RSA SIGNATURE SCHEME

The RSA digital signature scheme applies the sender's private key to a message to generate a signature. The signature can then be verified by applying the corresponding public key to the message and the signature through the verification process, providing either a valid or invalid result. These two operations — sign and verify — comprise the RSA digital signature scheme [2]. Taking a closer look at the signature generation portion of the process in Figure 1, the first step in generating an RSA signature is applying a cryptographic hash function to the message.

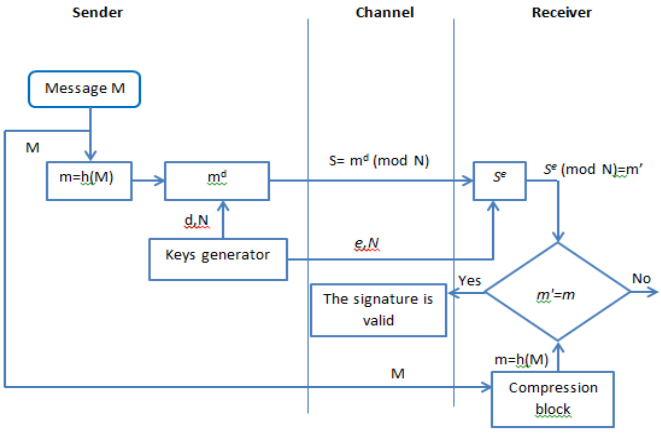

Figure 1. Generalized scheme of digital signature RSA

The hash function is specifically designed to reduce a message of any length to a short number, called the "hash value" (typically 160 bits long), and to do it in a way such that two conditions are satisfied:

- It is difficult to find a message with a specific hash value.
- It is difficult to find two messages with the same hash value (an easier problem to solve)

Let's show how this this scheme works using Alice and Bob actions description.

The algorithm is

1. Alice compresses the initial message M into integer number m using hash-function

   h:m=h(M).

2. Alice chooses secret big odd primes *p,q* and computes $N=p\cdot q$ and ϕ(N)=*(p-1)·(q-1)*.

3. Alice chooses e_A with condition gcd $(e_A,\phi(N))=1$

4. Alice computes $d_A = e_A^{-1} mod\ \varphi(N)$.

5. Alice's signature is $S = m^{d_A} mod\ N$. She sends signed message is (M,S) to Bob.

6. Bob recover hash value m' by calculating $m' = S^{e_A} \text{mod } N$.

7. Bob verify the signature by comparing *m* and *m'*. The signature is valid if *m=m'*.

Extended Euclid's algorithm for finding of multiplicative inverse

To compute mm_{AA} Alice should use a special technique - Extended Euclid's algorithm . Let's recall that number ***d*** is called the *multiplicative inverse* of *e* (modulo φ(n)) [3] if

$$e \cdot d \equiv 1 (\text{mod } \varphi(n)) \qquad (1)$$

For illustration of extended Euclid's algorithm acting we will use two-stage scheme (see below) [4].

Pseudocode (stage 1)

AT THE INPUT: two natural a and b, a>=b

AT THE OUTPUT: D =GCD(a,b) and integers x and y such that ax +by=D 1.

1. Let's x1:=1, x2:=0, y1:=0, y2:=1
2. While b>0
3. q:=[a/b], r:=a-qb, x2:=x1-q*x2, y2:=y1-q*y2
4. a:=b, b:=r, x1:=x2, x1:=x, y1:=y2, y1:=y
5. Put D:=a, x:=x1, y:=y1 and return (D,x,y)

Here [c] means integer part of c.

Example

Let us find D=GCD(500,440) and integers x and y such that 500x +440y=D. Initial data:

x1:=1, x2:=0, y1:=0, y2:=1

1-st step:

q=[500/440]=1, r=500-1*440=60;

x2=1-0=1,y2=0-1=-1, a=440,b=60

It is suitable to place the intermediate results of computation into the table:

№	a	b	q	r	x1	x2	y1	y2
1	500	440	1	60	1	0	0	1
2	400	60	7	20	0	1	1	-1
3	60	20	3	0	1	-7	-1	8
4	20	0	-	-	-7	22	8	-25

Summary : GCD(500,440)=20=500*(-7)+440*8.

This algorithm was realized in MATLAB (see e.g. [5]) as a function gcd ([g,u,v] = gcd(A,B)) is calculated using the extended Euclidian algorithm)

Part2

Algorithm for finding multiplicative inverse due to the formula (1)

Pseudocode (stage 2)

AT THE INPUT: two natural e and N.

AT THE OUTPUT: inverse of e in modulus of N.

1. Use extended Euclid's algorithm for finding of x and y such that ex + Ny = D, where D= GCD(e,N)
2. If D>1 then there is no inverse element else return x.

Exercise 1.

Generate a code realizing Extended Euclid›s algorithm in Matlab. SolutionSee code below.

```
function R = reverse (M, N)
%Calculate reverse for M modulo N.
[x, y, d] = egcd(M, N);
if d == 1
R = mod(x, N)
   return
end
end
function [x, y, d] = egcd(a, b)
%Extended Euclidean algorithm.
%Calculate greatest common divisor.
a = abs(a);
b = abs(b);
```

```
if b == 0
   x = 1;
   y = 0;
   d = a;
   return
end
x1 = 0;
x2 = 1;
y1 = 1;
y2 = 0;

while b > 0
   q = floor(a / b);
   r = a - q * b;
   xtmp = x2 - q * x1;
   ytmp = y2 - q * y1;
   a = b;
   b = r;
   x2 = x1;
   x1 = xtmp;
   y2 = y1;
   y1 = ytmp;
end
x = x2;
y = y2;
d = a;
end
```

Example of using

```
>> reverse(3, 5874292)
R =     3916195
```

Exercise 2.

Choose q=3083, p=1907 and check how the RSA signature algorithm works for m= 14615016373309029182036848327162830196559325429742 using Matlab.

Solution

See code below for Alice

```
function [m, s] = f2(P, Q, e, m)
n = P * Q
phi = (P - 1) * (Q - 1)
if e < 1 | e > phi | gcd(e, phi) ~= 1
 'Enter correct data'
end
d = reverse(e, phi)
s = modulopower(m, d, n)
end
function R = modulopower(X, N, M)
X = rem(X, M);
R = 1;
while N > 0
   if rem(N, 2) == 0
      X = rem(X * X, M);
      N = N / 2;
   else
      R = rem(R * X, M);
       N = N - 1;
   end
end
end
function R = reverse(M, N)
%Calculate reverse for M modulo N.

[x, y, d] = egcd(M, N);
 if d == 1
   R = mod(x, N);
   return
end
'There are no reverse for M modulo N.'
› end
function [x, y, d] = egcd(a, b)
%Extended Euclidean algorithm.
%Calculate greatest common divisor.
a = abs(a);
b = abs(b);

if b == 0
   x = 1;
   y = 0;
   d = a;
   return
end
x1 = 0;
x2 = 1;
```

```
y1 = 1;
y2 = 0;
while b > 0
   q = floor(a / b);
   r = a - q * b;
   xtmp = x2 - q * x1;
   ytmp = y2 - q * y1;
   a = b;
   b = r;
   x2 = x1;
   x1 = xtmp;
   y2 = y1;
   y1 = ytmp;
end
x = x2;
y = y2;
d = a;
end
```

Example of using

```
   >> f2(3083,1907,5777,14615016373309029182036848327162830196559325429 74)
   n =  5879281
   phi =       5874292
   d =  64061
   s =  4257374
```

The next function is for Bob

```
function f2_test(n, e, m, s)
mtest = modulopower(s, e, n)
m = rem(m, n)
if mtest == m
'Document is authentic'
end
end
function R = modulopower(X, N, M)
 X = rem(X, M);
R = 1;
while N > 0
   if rem(N, 2) == 0
      X = rem(X * X, M);
      N = N / 2;
   else
      R = rem(R * X, M);
      N = N - 1;
   end
end
end
function R = reverse(M, N)
%Calculate reverse for M modulo N.
… % See eponymous function above
end
function [x, y, d] = egcd(a, b)
%Extended Euclidean algorithm.
%Calculate greatest common divisor.
… % See eponymous function above
 end
```

Example of using

```
>>  f2_test(5879281,5777,14615016373309029182036848327162830196559325429 74,4257374)
mtest = 1748613
m =     1748613
ans = Document is authentic
```

RSA signature faults

RSA signature is vulnerable for so called multiplicative attack [6]. In other words, RSA signature algorithm allow malefactor to generate signatures on those documents which hashing results can be computing by the product of signed documents hashing results without of knowing secret key d.

Suppose, that attacker can construct 3 messages M1,M2 and M3 with hash-values

$$m_1=h(M_1),\ m_2=h(M_2),\ m_3=h(M_3),$$

moreover

$$m_3=m_1*m_2\ (mod\ N).$$

We also assume, that for two messages M1,M2 low signatures S1 and S2 were obtained:

$$S_1 = m_1^d \bmod N,\ S_2 = m_2^d \bmod N.$$

Then malefactor could easy calculate signature S3 for the document M3 without knowing secret key d:

$$S=S_1*S_2\ (\text{mod } N).$$

Indeed,

$S_1*S_2\ (\text{mod } N)= m_1^d \cdot m_2^d\ (\text{mod } N) = (m_1 \cdot m_2)^d (\text{mod } N) = m_3^d\ (\text{mod } N) = S_3.$

More reliable and suitable DS algorithm was designed by El Gamal [7].

El Gamal signature scheme

The idea of El Gamal Singnature Algorithm (EGSA) is based on the fact that to justify the practical impossibility of falsification of the digital signature can be used more complex computational problems than factoring a large integer - the discrete logarithm problem. In addition, El Gamal avoided overt weakness of RSA digital signature algorithm, coupled with the possibility of forgery of digital signatures under some messages without specifying a secret key.

Let us consider the digital signature algorithm El Gamal. In order to generate a key pair (public key - a secret key), a first chosen large prime integer P and large integer G, where G <R. The sender of the signed document (Alice) and receiver (Bob) use in the calculations similar large integers P (~ 10308 or ~ 21024) and G (~ 10154 or ~ 2512), which are not secret.

The algorithm is

1. Alice chooses random integer X, 1< X ≤ (P-1), and compute

   $$Y = G^X \bmod P.$$

   The number Y is the public key used to verify the signature of the sender. The number of Y is open to all potential recipients of transferred documents. The number X is the sender›s private key for signing documents and should be kept secret.

2. Alice hashes a message M using the hash function *h*:

   $$m=h(M),\ 1<m<(P-1),$$

   and generates random integer K,

   $1<K<(P-1)$ such that K and $(P-1)$ are coprime.

3. Alice computes an integer a by the formula

   $$a = G^K \bmod P.$$

4. Alice computes integer b, solving the equation (see e.g.[9]):

   $$m=X*a+K*b \pmod{(P-1)}$$

   with help of extended Euclid›s algorithm.

   The pair of numbers (a,b) form digital signature S

   $$S=(a,b)$$

   affixed to the document M.

   A triple (M, a, b) is sent to the recipient, while the pair (X, K) is kept secret.

5. After receiving the signed message (M, a, b) Bob should verify is the signature

   S = (a, b) corresponding to the message M.

   He first calculates the hash-value of the received message M:

   $$m = h(M).$$

6. Then he calculates the value

   $$A = Y^a a^b \pmod P$$

   and recognizes the message authentic if and only if

   $$A = G^m \pmod P.$$

   In other words, a receiver checks the equity of ratio

   $$Y^a a^b \pmod P = G^m \pmod P.$$

One can be strictly mathematically shown that the last equation is satisfied if and only if the signature is S = (a, b) under the document M is obtained using a secret key of X, from which the public key Y was obtained. Thus, one can reliably make sure that the sender of the message M was the holder of the private key is X, without disclosing the key itself, and that the sender signed namely this concrete document M.

It should be noted that the execution of each signature on the El Gamal method requires a new value of K, and this value should be chosen randomly. If an intruder ever discloses the value of K, reusable sender, then he will be able to reveal the secret key X sender.

Example 3.

Let's choose: the numbers R = 11, G = 2 and the secret key X = 8. Calculating the value of the public key:

$Y = G^X \bmod P = Y = 2^8 \bmod 11 = 3.$

Assume that the original message M is characterized by the hash value m = 5.

In order to compute the digital signature for the message M having a hash value m = 5, first select a random integer K = 9. Make sure that the number of K and (P-1) are coprime. Indeed, gcd (9,10) = 1. Next, calculate the elements a and b of the signature:

$a = G^K \bmod P = 2^9 \bmod 11 = 6$

Then determine the element b by using an extended Euclid›s algorithm:

$m = X * a + K * b \pmod{(P-1)}.$

When $m = 5, a = 6, X = 8, K = 9, P = 11$ we obtain $5 = (6*8 + 9 * b) \pmod{10}$

or

9 * b = -43 (mod 10).

Solution:

b = 3.

A digital signature is a pair: a = 6, b = 3. Then the sender sends the signed message. By adopting a signed message and the public key Y = 3, the receiver calculates the hash value for the message M:

m = 5

and then calculates two numbers:

1) Yaab (mod P) = 36 * 63 (mod 11) =10 (mod 11);

2) Gm (mod P) = 25 (mod 11) =10 (mod 11).

Since these two integers are equal, the message, taken by the recipient, deemed authentic.

Exercise 3.

Generate MatLab code to realize El Gamal signature scheme including

a. Fixation of P, G, X values and hash-value m

b. Computing open key Y and selection of K satisfying (2).

c. Computing a & b for signature, using a program module, generating in the Exercise1.

d. Computing Yaab (mod P) and Gm (mod P)

e. Output of the message like “message deemed authentic”

Solution

See code below

```
function [a, b] = f3(P, G, X, M)
m = hash(M);
Y = modulopower(G, X, P);
%Generate random K
while 1
   K = randi(P - 1);
   if gcd(K, P - 1) == 1
      break
   end
end
a = modulopower(G, K, P);
b = mod(reverse(K, P - 1) * (m - X * a), P - 1);
end
```

```
function m = hash(M)
m = length(M);
end

function R = modulopower(X, N, M)
X = rem(X, M);
R = 1;
while N > 0
   if rem(N, 2) == 0
      X = rem(X * X, M);
      N = N / 2;
   else
      R = rem(R * X, M);
      N = N - 1;
   end
end
end

function R = reverse(M, N)
%Calculate reverse for M modulo N.
… % See eponymous function above
end

function [x, y, d] = egcd(a, b)
%Extended Euclidean algorithm.
%Calculate greatest common divisor.
…% See eponymous function above
end
   Example of using
   >> f3(11,2,8,›Example of El Gamal digital signature›)
   a =  2
   b =  1
```

Let’s remark, that the numbers a and b in this fragment are different from the same in the

Example 3

Cause of this is in the random character of k.

For checking of authenticity we generate the next Matlab function

```
function f4(P, G , Y, a, b, M)
m = hash(M);
A1 = mod(modulopower(Y, a, P) * modulopower(a, b, P), P);
```

```
A2 = modulopower(G, m, P);
if A1 == A2
   'Document is authentic'
end
end

function m = hash(M)
m = length(M);
end

function R = modulopower(X, N, M)
... % See eponymous function above
end

function R = reverse(M, N)
%Calculate reverse for M modulo N.
... % See eponymous function above
end

function [x, y, d] = egcd(a, b)
%Extended Euclidean algorithm.
%Calculate greatest common divisor.
... % See eponymous function above
end
```

```
Example of using
>> f4(11,2,3,2,1,›Example of El Gamal digital signature›)
ans = Document is authentic
```

It should be noted that EGSA is a typical example of an approach that permits sending the message M in the open form together with the attached authenticator (a, b). In such cases, the procedure of establishing the authenticity of the received message consists on the verifying of compliance a message to the authenticator.

Digital signature scheme El Gamal has a number of advantages over digital signature scheme RSA:

1. For a given level of firmness of DS algorithm, the integers involved in the calculations have a quarter shorter representations, which halve the computational complexity and noticeably decreases the amount of memory used.
2. When you select a module P it is sufficient to verify that this number is simple and that the number (P-1) has a large prime factor (i.e., only two the conditions simply checking).
3. The procedure for signature generation due to El Gamal scheme does not allow calculate digital signatures for the new messages without the knowing of the secret key (as in the RSA).

References

1. Digital signature URL: http://en.wikipedia.org/wiki/Digital_signature
2. Kaliski B. S. RSA Digital Signatures // Dr. Dobb's electronic journal, May 01, 2001. URL: http://www.drdobbs.com/rsa-digital-signatures/184404605
3. Modular multiplicative inverse URL: http://en.wikipedia.org/wiki/Modular_multiplicative_inverse
4. Bogatov E. MATHEMATICA realization of Encryption algorithm based cryptosystem RSA.
5. MATLAB URL: http://en.wikipedia.org/wiki/MATLAB
6. Misarsky J-F. A Multiplicative Attack Using LLL Algorithm on RSA Signatures with Redundancy // Proceeding CRYPTO ‹97. P.221-234
7. ElGamal signature scheme URL: http://en.wikipedia.org/wiki/ElGamal_signature_scheme
8. Discrete logarithm URL: http://en.wikipedia.org/wiki/Discrete_logarithm
9. Bogatov E. MATHEMATICA realization of ELGAMAL Encryption algorithm