CONSTRUCTING THE DIGITAL SIGNATURE SCHEME BASED ON THE DISCRETE LOGARITHMIC PROBLEM

Le Van Tuan[1,*], Bui The Truyen[2], Leu Duc Tan[3]

Abstract: *In this report, the DSA digital signature scheme has been introduced. Based on these digital signature scheme, a new digital signature scheme on the ring of Z_n is developed. Its c*omputing complexity is similar *to the DSA and getting rid of some unwanted disadvantages of the DSA digital signature scheme and can be applied in practice.*

Keywords: Digital Signature Scheme, Discrete logarithmic problem, Hash Function.

1. INTRODUCTION

Nowadays, the digital signature plays an important role for authentication, integrity, information security. Therefore, it has been being applied in many organizations and countries over the world, such as the DSA digital signature scheme of US [7], the GOST digital signature scheme of Russia, or Okamoto's key distribute system based on the identification information [8]... However, these schemes' standards of parameters are often published and they are applied for commercial purpose, if they are used in defense security area, the information highly runs the risk of being unsafe. So, developing the digital signature schemes is a researching trend of many countries and the code scientists in the world. [1], [2], [8]. In this study, we have studied and developed the digital signature scheme based on the ring of Z_n safe based on the complication of discrete logarithmic problem on ring Z_n. This new digital signature scheme can be applied in practice.

2. THE DEFINITION OF FUNCTIONS

Definition 2.1. The Number function converts a binary string into an integer that does not exceed T bits, notated **Number**: $\mathbb{N}\times\{0, 1\}^H \to \mathbb{Z}$.

Definition 2.2. Random function: The random function is a function that randomly takes an integer in [a, b], a random number (a, b).

Definition 2.3. The bitlength (m) function returns the size of m as how many bits

Definition 2.4. A||B is a connection between the string A with the string B.

3. DSA DIGITAL SIGNATURE SCHEME

3.1. DSA parameters

p is a prime, its length is bit, bitlength(p)= L.

q is a divisor prime of p – 1, bitlength(q)= N.

g is a primitive element of subgroup q on Zp, 0< g < p

x is the private key that must be kept in secret; x is randomly chosen or pseudorandomly in [1, q–1].

y is the public key, where $y = g^x \bmod p$.

k is a secret number for each message (another name is session key); k is randomly chosen or pseudorandomly in [1, q–1].

Set of (p, q, g, x) is also called private key and set of (p, q, g, y) is a public key of signer.

3.2. Signature generation algorithm

Algorithm **3.1[7][10]:**

Input: (p, q, g, x), k, M.

Output: (r,s).

1. z ← **Number**(N, **Hash**(M)).
2. k ← **Random**(1, q).
3. r ← (g^k mod p) mod q.
4. w ← (z + x.r) mod q.
5. if (r = 0) or (w = 0), then go to 2.
6. s ← (k^{-1}.(z + x.r)) mod q.
7. return (r, s).

3.3. Signature verification algorithm

Algorithm 3.2 [7], [10]

Input: (p, q, g, y), (r,s), M.

Output: "*accept*" or "*reject*".

1. w ← s^{-1} mod q.
2. z ←**Number**(N, **Hash**(M)).
3. u1 ← (z.w) mod q.
4. u2 ← (r.w) mod q.
5. v ← ((g^{u1}.y^{u2}) mod p) mod q.
6. if (v = r) then return "accept" Else return "reject".

3.4. Computing complexity

The computing complexity of the algorithm 3.1 is focused on method g^k mod p. According to [6, p176], if bitlength (p) = L, bitlength (q) = N, the computing complexity of g^k mod p ≈ O(logk.L^2). If M_L is a computing complexity for a multiplication on the field Z_p with bitlength (p)= L and M_N the computing complexity for a multiplication on the field Z_q with bitlength (p)= N. If bitlength(k) ≈ N, the charge of the algorithm 3.1 is estimated as below:

$$C_G \approx N.(M_L + M_N) \quad (3.1)$$

The computing complexity of algorithm 3.2 is centered on the statement of step 5 with two powers ((g^{u1}.y^{u2}) mod p) mod q. If bitlength (u_1) ≈ N bitlength (u2) ≈ N, total charge for algorithm 3.2 is estimated as below:

$$C_v \approx 2N(M_L + M_N) \quad (3.2)$$

3.5. The security of DSA digital signature scheme

The security of DSA digital signature scheme is displayed on the difficulty that generates the valid signature not for the owner of parameter x or that called "forging signature". Indicatively, signature forging is generated when a discrete logarithm problem is solved on Zp or Hash function is collided, so FIPS_186-4 standard [7] proposed the size of bit of the L and N parameter that would allow the DSA to be safe until 2030 as follows: (L, N) = (2048, 224), (L, N) = (2048, 256) and (L, N) = (3072, 256). However, when the primitive element g is published, that makes the DSA digital signature scheme unsafe in some of the situations as following:

The first situation: While signing the message M, if the session key is revealed, the secret key x is calculated by the following formula:

$s = (k^{-1}(z + r.x)) \bmod q$

The secret key x is computed easily using the following formula:

$x = ((s.k - z).r^{-1}) \bmod q$

This situation is illustrated as follows: The message m is considered, which is transformed by the Hash function SHA-512 and the output is M as follows:

M=97975747105107409187956387180507750847513881963918549765479
88076566303385116000144790078176179684986903594710324051795779872 9
04918107245447649435608188242

The value of the prime number p is:

p=112424395959223300945000868448049663577593134318123853714945
11369847330693855087879539343469849551078435848382192561802575175
14245595104385114046841894 3323

The value of prime q is as follows:

q=153329186497049199093793510233616626933461813467801664 8433

The value of the primitive element g is:

g=100786640705444536784647092580848041314913651922049510249275
48083774068483673108796539309434316671809670409438483987936673806
83247545854020686206014390 1544

The secret key x is: **x**=74679656459306509739026621399

The public key is calculated by the formula $y = g^x \bmod p$, the value y is:

y=25356434363194005296219381386234511997842772125732011576745
96403950923612510145765332612020189996976776676924581369583555461 6
78797023907419768950517837742

Session key **k**=369149257163353279199024650 72.

The signature (r, s) is calculated as follows:

r=18250632337354015030699176560185181482393902611224487818 0

s=13641556575944489777898318098393741096265373757565434508 08

If the session key k is revealed, the secret key x is calculated as follows:

$x=((s.k- z).r^{-1}) \bmod q$. The value z = **Number**(N, **Hash**(M)).

While
z=171857999475270176118545303731951970954583939072760902587 4

s.k=503577047656176547481481010757784564124921357379285064971 7083189044334554835657017817 6

(s.k-z) mod q =

524050766533797319315236385558804362401516138504649902691

r^{-1} mod q=

997958988071514117853806215293691244299223701750102569501

x =74679656459306509739026621399.

The second situation: two different messages are signed with the same session key k (using the same key). If the session key of the message M and M is k and the two signatures corresponding to M and M' are respectively (r, s) and (r, s'). The secret key x will be found by the attacker as follows: The first z and z'values are calculated from the formula z = **Number** (N, **Hash** (M)) và z' = **Number** (N, **Hash** (M')), then s and s' are calculated as follows:

$$s= (k^{-1}(z + r.x)) \bmod q \leftrightarrow k= s^{-1}(z+r.x) \bmod q \quad (*)$$

$$s'= (k^{-1}(z' + r.x)) \bmod q \leftrightarrow k= s'^{-1}(z'+r.x) \bmod q \quad (**)$$

From (*) and (**) the equation is as following:

$s^{-1}(z+r.x)= s'^{-1}(z'+r.x) \bmod q \leftrightarrow s^{-1}z- s'^{-1}z'= (s'^{-1} – s^{-1}).r.x \bmod q$, from this equation it is easy to calculate the secret key x as follows: $x= r^{-1}(s^{-1}.z-s'^{-1}z')(s'^{-1} – s^{-1})^{-1} \bmod q$.

This situation is illustrated as follows: when the message is m, which is transformed by the Hash function SHA-512 and the output is M as follows:

M=979757471051074091879563871805077508475138819639185497654798807656630338511600014479007817617968498690359471032405179577987290491810724544764943560818824 2

z=171857999475270176118545303731951970954583939072760902587 4

p=112424395959223300945000868448049663577593134318123853714945113698473306938550878795393434698495510784358483821925618025751751424559510438511404684189433 23

q=153329186497049199093793510233616626933461813467801664843 3

g=100786640705444536784647092580848041314913651922049510249275480837740684836731087965393094343166718096704094384839879366738068324754585402068620601439015 44

x=74679656459306509739026621399

y=253564343631940052962193813862345119978427721257320115767459640395092361251014576533261202018999697677667692458136958355546167879702390741976895051783774 2

k= 369149257163353279199024650 72

r= 182506323373540150306991765601851814823939026112244878180

s= 136415565759444897778983180983937410962653737575654345080 8

When the message is m', which is transformed by the Hash function SHA-512, its output is M' as follows:

M'=155545797790378805270704727624617010783982696611943776590855190766119384544294776774814775812703228285592734031011663492273191806970727216848019621260955 1

Z' is calculated by the formula as follow: z' = **Number** (N, **Hash** (M'))

z'= 26867836264497953922610884459517899067113833946073530905 75

p=112424395959223300945000868448049663577593134318123853714945113698473306938550878795393434698495510784358483821925618025751751424559510438511404684189433 23

q=153329186497049199093793510233616626933461813467801664843 3

g=100786640705444536784647092580848041314913651922049510249275480837740684836731087965393094343166718096704094384839879366738068324754585402068620601439015 44

x=746796564593065097390266213 99

y=253564343631940052962193813862345119978427721257320115767459640395092361251014576533261202018999697677667692458136958355546167879702390741976895051783774 2

k=369149257163353279199024650 72

r=r'=182506323373540150306991765601851814823939026112244878180

s'= 419930701966257575864048550021807565165401949643752549481

In this situation, the secret key x is calculated by the formula as follow:

$x= r^{-1}(s^{-1}.z-s'^{-1}z')\ (s'^{-1}- s^{-1})^{-1} \bmod q$. Where:

$s'^{-1} \bmod q$=1104369623657571975932291813648449955114049489078722571641

$s'^{-1}z' \bmod q$=606082038342155792558174208526507093346465357060780120714

$s^{-1} \bmod q$ =145127829784000970277745960078595116569853127058090205917 4

$s^{-1}.z \bmod q$= 131371363426454724057098100452221082558549695476562682277 7

$(s^{-1}.z-s'^{-1}z') \bmod q$= 707631595922391448012806795995703732239031597704846702063

$r^{-1} \bmod q$= 997958988071514117853806215293691244299223701750102569501

$r^{-1}(s^{-1}.z-s'^{-1}z') \bmod q$,

where q=125361325611663443023622459317563888056980102847137437 0307

$(s'^{-1} - s^{-1}) \bmod q =$

1186383190788054264092767315198665058750136353175837160900

$(s'^{-1} - s^{-1})^{-1} \bmod q =$

1004917863501878505091781121120933399705847132586335489436

The secret x can be easily calculated as follows:

x=746796564593065097390266213 99.

It's noted that if a session key is used for both signatures, both signatures of the component r will be the equal value, but if the both signatures of value r are of the equal value, the session key for both signatures may not be of the equal value.

4. CONSTRUCTION THE NEW SIGNATURE SCHEME

4.1. Parameters domain

n = p.q with p, q are primes and n-factorization is a difficult problem; m is private, $m = p_1.q_1$ where p_1, q_1 are primes and $p_1 \mid (p-1)$, $q_1 \mid (q-1)$, $p_1 \nmid (q-1)$, $q_1 \nmid (p-1)$; **mbit**= bitlength (m).

g is a primitive element, $0 < g < n$ $ord_g = m$; $<g>$ is cyclic subgroup on Z_n.

x is chosen randomly in (1, m−1) and x is private.

y is public, where $y = g^x \bmod n$.

k is chosen randomly in (1, m−1), k is the unique private number for each message (also known as session key)

the set of five values (n, m, g, x, m) are the secret key and set of four values (n, g, y, mbit) are the public key.

4.2. Generation signature

Algorithm 4.1.

Input: (n, m, g, x, **mbit**), M.//mbit= bitlength (m)

Output: (r,s).

1. z←0; tg←0;

2. While (z=0) or ((tg,m) <> 1)

3. k← Random(1,m-1)

4. $r \leftarrow g^k \bmod n$.

5. z ← **number** (**mbit**,H(M||r))// The output is the value z, its size is not more than mbit

6. tg←(z + x) mod m

End while // The loop stops when z + x exists the inverse value

7. $s \leftarrow (k.(z+x)^{-1}) \bmod m$)

8. return (r, s).

4.3. Verifying signature

Algorithm 4.2.

Input: M, (r, s), (n, g, y, **mbit**)// **mbit=** bitlength (m).

Output: "*True*" hoặc "*False*".

1. z ← **number**(**mbit**,H(M||r))// value of z < a **mbit number**.

2. u ← ($g^{s.z}$.y^{s}) mod n.

3. if (u = r) return "*True*" else return "*False*".

4.4. Correctness of the algorithm

It's easy to see that:

$$(g^{s.z}.y^{s}) \bmod n = \left(g^{\left(k(z+x)^{-1}z\right)\bmod m} y^{\left(k(z+x)^{-1}\right)\bmod m}\right) \bmod n$$

$$= \left(g^{\left(k(z+x)^{-1}z\right)\bmod m} g^{\left(x.k(z+x)^{-1}\right)\bmod m}\right) \bmod n = g^{\left(k(z+x)(z+x)^{-1}\right)\bmod m} \bmod n$$

$= g^{k} \bmod n = r$ and the correctness of the digital signature scheme is proven.

4.5. Example

Algorithms 4.1 and 4.2 are illustrated by the following example:

Parameter generation:

If the set of parameters of the new digital signature scheme are as following:

The value of the prime number p is:

112424395959223300945000868448049663577593134318123853714945113698473306938550878795393434698495510784358483821925618025751751424559510438511404684189433 23

p_1 is a prime divisor of p-1 and its value is as follows:

153329186497049199093793510233616626933461813467801664843 3

The value of prime q is:

130000560972504071512788165804246947621261895980352286081606552852861010770838146376021398668218881682311290872002078816520341293468337587075000997537562 33

q_1 is a prime divisor of q-1 and its value is as follows:

693542050471501789153827056172377247128936626662022537753

n = p.q and its value is as follows:

146152345416939490956091520613847874423616916721349390780549140556793655977365602162236593399386567108441253988669808643399361466761237118155352320830000165967553946441662646723200854344777853261427609882390385466334904827355895951273494390019020781138833381706728100718302316964348020060060727207161984982259

The value of the primitive element g is as follows:

138424556970583530973076308549473018965961792410341925212468180290979944969726115786587692319213233324223907829217807013100248390060776582934703889199940317368973504004180108312052761173489006555000946973208786876770358348612739499110656338674320443306738235991077062669737065186351843811074484096101625162002

$m = p_1.q_1$ and is the order of g. The value of m is as follow:

106340238400290806253227211342005767285895153438697402519985413274571962314572862434670686722649492219439827079104 9

The value of the private key x is as follows:

389809198418777200281265970415938210483812622066833804737591569778957938885573951169495

$y= g^x$ mod n and is the public key. Its value is as follows:

180813141264636817072081056005313548032194449429373642267666558621364099363452983730981645753986853360303052027075209193295301427430665261264366451065792467571527987132744113824315624100985632223663355772785530265485039789417302784812021441666318480198674461633139529515311810094012976736930326238665998345 93

Note: The prime numbers are used in this example that will be generated by the algorithm in [3] [5]. With the set of the parameter above, the result of signature generation and signature verification is as follows:

Signature generation (algorithm 4.1)

Input: Message M

Output: (r, s) is signature of M

Step 1: If the size of the message T is 13.87 Mb. Hash function used in example is SHA-512, denoted H and M = H (T). M is represented by the Hex number as follows:

BB118C3045EB6214275894575B6CD7036504379792BAB744E3B6EB6FDF4213D0921E67819974A7654616CFF6214A056398C026848CD131C757907D280EE88552.

The decimal value of M is as follow:

979757471051074091879563871805077508475138819639185497654798807656630338511600014479007817617968498690359471032405179577987290491810724544764943560818824 2

Step 2: Number k is randomly selected in (1, m-1), the result is as follows:

k= 655590539111223343691176877 06

Step 3: The value of r is calculated by the formula below:

$r \leftarrow g^k$ mod n, the result is as follows:

r=700075376919223825896662238446486887192303123740882929410199049353139983819732156200176461364426777339235615578940280578869971077324767508786541362979546740656434723749239588803228906749181225476240293095750593205272535388486577819602972736032978332611470815351219911559084272292863529611445059498443577431 76

Step 4: z= number (N, M||r), the result is follows:

z=155478582251494173649984772551104614294175933861221891771125744023144819210398724028720631593181071391037019615737 14

Step 5: s is calculated by formula as follow: $s= (k.(z + x)^{-1}) \bmod m$. If s does not exist (because there is no inverse of (z + x)) then goes to step 2), value of s is:

s=577507141296403366601531541536948842155948696090904353917855626403078896511057634111000995527913016381540600291993

The result of signature verification (**Algorithm 4.2):**

Input:

Pair of (r, s) is computed in algorithm 4.1

M=979757471051074091879563871805077508475138819639185497654798807656630338511600014479007817617968498690359471032405179577987290491810724544764943560818824 2

Step 1: computed z= number (mbit, M||r), the result is follows:

Z=155478582251494173649984772551104614294175933861221891771125744023144819210398724028720631593181071391037019615737 14

Step 2: u is computed by formula as follow: $u \leftarrow (g^t.y^s) \bmod n$, where $t=(s.z) \bmod m$. Value of t is as follow:

t=52043826357699143581644993156302382560968906978719658295868556660326006851318852877557879289120311977248897224820. Then value of $g^t \bmod n$ is:

13844681836188787050490416867068509771590285902016616400783650957734311355486486435527647703334553405840029394487960028532497459814389197572124605811554459426661757447249592896271588855957968225033434748506994886694757244516753159045582211664278264328337982381941185838545122142439785511308786843843541983368 5.

With value of s is:

s=577507141296403366601531541536948842155948696090904353917855626403078896511057634111000995527913016381540600291993.

The value of $y^s \bmod n$ is as follow:

12816170749087009435477966847753686635354362533420059134229242260063485426448713694511362952689978278402176620915385550216450289169708686105622979392488244289452879294305003689544189051410160007820070589877996495733249044611900436776739808890379714645276294533604134813858546206551981195412169371142671541255 2

Then value of u with $u \leftarrow (g^t.y^s) \bmod n$ as follow:

u=70007537691922382589666223844648688719230312374088292941019904935313998381973215620017646136442677733923561557894028057886997107732476750878654136297954674065643472374923958880322890674918122547624029309575059320527253538848657781960297273603297833261147081535121991155908427229286352961144505949844357743176

Step 3: Check both values: u and r show that u = r, so the pair of (r, s) is the signature of the message M.

4.6. Computing complexity

There is at least one loop in the algorithm 4.1. The computing complexity for each of these loops is the total of the computing complexity of g^k mod n and the computing complexity of $(x + z)^{-1}$ mod m. According to [6], the computing complexity of g^k mod n is approximately O (logk.L^2), where L= bitlength (n). Because of $(x+z)^{-1}$ mod m = $(x + z)^{\varphi(m)-1}$mod m, where $\varphi(m) = (p_1-1)(q_1-1)$, so the computing complexity of $(x + z)^{-1}$ mod m is approximately O(log $\varphi(m).N^2$), in which N = bitlength (m). If M_L denotes for the computing complexity of a multiplication on ring Z_n, where bitlength (n) = L and M_N denote for the computing complexity of a multiplication on Z_m, where bitlength (m) = N, then the computing complexity for each of these loops is as follow:

$$N(M_L + M_N)$$

The loop will exit if gcd (z + x, m) = 1 or (z + x) must be have an inverse in Z_m. Probability (z + x) exists its inverse element is as follow:

$$\text{Prob}_{(\gcd(z+x,\,m)=1)} = \frac{\varphi(m)}{m} = \frac{(p_1-1)(q_1-1)}{m}$$

If the m is a large enough number, then $\text{Prob}_{(\gcd\,(z+x,\,m)\,=\,1)} \approx 1$, so the computing complexity of the signature generation algorithm denoted by C_G is as follows:

$$C_G \approx NM_L + NM_N \quad (4.1)$$

The complexity of algorithm 4.2 is mainly focused on the complexity of exponentiation $g^{s.z}.y^s$ mod n, so $g^{s.z}.y^s$ mod n= $g^{s.z}.g^{s.x}$ mod n. If M_L denotes for the computing complexity of a multiplication on ring Z_n, where bitlength (n) = L and M_N denotes for the computing complexity of a multiplication on Z_m, where bitlength (m) = N, then the computing complexity of algorithm 4.2 is as follow:

$$C_v \approx 2NM_L. \quad (4.2)$$

4.7. Some comparisons of the new digital signature scheme and the DSA

Comparison about the security:

Scheme the new digital signature scheme is similar to scheme DSA, so the analysis of abilities to forge signatures can be displayed similarly to DSA. However, there is a basic difference between the DSA digital signature scheme and the new digital signature scheme as follow: the order of g is published on the DSA digital signature scheme, but on the new digital signature scheme, the order of g is not published. From this difference, when the new digital signature scheme is displayed there's no revealing or coinciding session key as using DSA. Moreover, when key session of the new digital signature scheme is revealed, it's difficult to forge the signatures.

Charge of computing:

Basing on formula 3.1 and formula 4.1, the charge of computing of DSA signature generation algorithm is similar to the new digital signature scheme. Similarly, the computing charge of signature verification algorithm on the new digital signature scheme is similar to the DSA scheme (based on formula 3.2 and 4.2).

5. EXPERIMENT

In this experiment, the length of keys on the scheme the new digital signature scheme and the DSA scheme gradually is 1024, 1280, 1536, 1792, 2048 (bits). The message's size of the testing is 18.87 MB. The time of experiment for each set of parameter is 1000 times. The Hash function SHA 512 used in signing and confirms algorithm of DSA digital signing scheme and the new digital signature scheme digital signing scheme. The test program is written by C ++ programming language, compiled by QT Creater and installed on Laptop in which its processor is a Core2 Duo 2.2 GHz and its memory is 2GB. The experimental parameter of the DSA signature scheme and the new digital signature scheme are generated by the algorithm [5], results are listed in the following table:

Table 1. *Consulting table of signing generation and verification.*

Key size(bit)	**Generation time**		**Verification time**	
	DSA	**New**	**DSA**	New
1024	1.416	1.539	6.836	5.527
1280	1.814	1.953	10.765	8.931
1536	2.89	3.182	14.813	13.3
1792	3.5	4.916	19.18	17.13
2048	5.138	5.929	22.59	26.398

In order to compare the signing speed between the DSA scheme and the new signing scheme. The test results in table 1 is illustrated by the following graph.

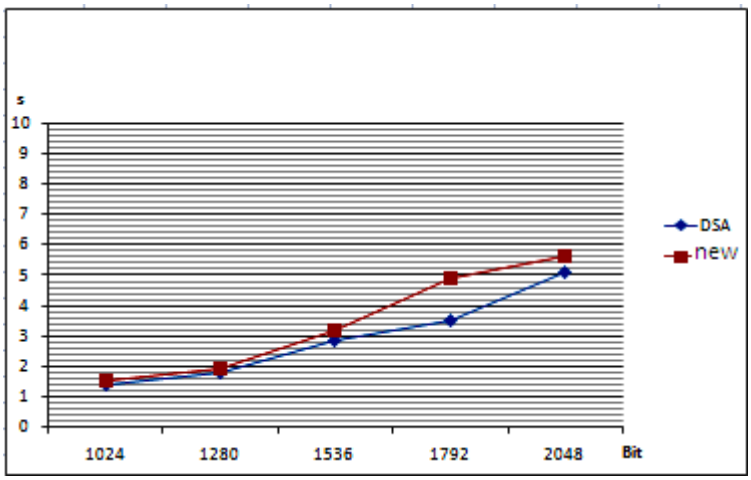

Figure 1. *The signature generation graph of the DSA and the new digital signature scheme.*

Similarly, based on Table 1, the relationship between the wasting time and key size to verify the digital signature scheme DSA and the new digital signature scheme is depicted by graph as follow:

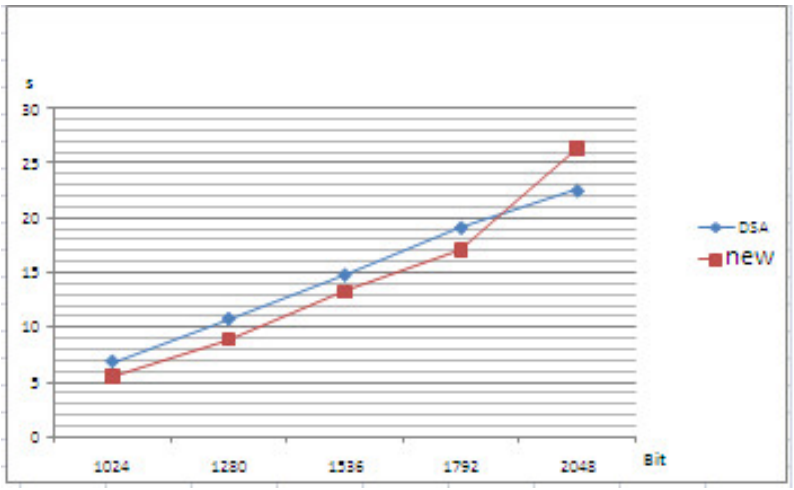

Figure 2. *The signature verification graph of the DSA scheme and the new digital signature scheme.*

Figure 1 and Figure 2 shows that the computing complexity of the signature generation algorithm and the signature verification algorithm of the the new digital signature scheme and the DSA digital signature scheme is equivalent. The graphs (Figure 1 and Figure 2) show that the results of the test and the results of the mathematical analysis shown in formula (3.1), (3.2), (4.1) and (4.2) is similar.

6. CONCLUSIONS

The authors' research results are a new digital signature scheme, in which its security is based on discrete logarithm problem in the ring of Z_n. This new digital signature scheme has overcome some of the defect of the DSA the digital signature scheme (shown in 3.5). Based on the mathematical theoretical analysis and the test results, the computing complexity of the new digital signature scheme and the computing complexity of the DSA digital signature scheme are equivalent. However, if this digital signature scheme can be applied in practice, it should be estimated the security and the computing complexity, these research problems will be introduced in the next article.

REFERENCES

[1]. Lưu Hồng Dũng *"Phát triển một dạng lược đồ chữ ký số mới"*, Hội thảo quốc gia lần thứ XVI, 13-14-11-2013.

[2]. Lưu Hồng Dũng *"Phát triển lược đồ chữ ký số tập thể"*, Luận án tiến sỹ, Hà nội – 2014.

[3]. Lều Đức Tân *"Số nguyên tố và đa thức nguyên thủy"*, Hà nội – 2002.

[4]. Lê Văn Tuấn, Lưu Hồng Dũng, Nguyễn Tiền Giang, "*Phát triển lược đồ chữ ký trên bài toán phân tích số,*" Tạp chí Nghiên cứu KH& CNQS - Đặc san CNTT, 04 – 2014, ISSN 1859 – 1043.

[5]. Lê Văn Tuấn, Bùi thế truyền *"Xây dựng thuật toán sinh số nguyên tố tất định"*, Tạp chí Nghiên cứu KH& CNQS, Số 42, 4- 2016.

[6]. D.R Stinson, *Cryptography: Theory and Practice*, CRC Press 2003.

[7]. FIPS PUB 186-4,"*Digital Signature Standard (DSS)*", 2013.

[8]. OKAMOTO E, *Key distribution systems based on identification information*. Proc. Of Crypto -1987.

[9]. Richard Crandall, Carl Pomerance. "*Prime Numbers, A Computational Perspective"*, Second Edition, Springer Science + Business Media, Inc, 2005.

[10]. https://en.wikipedia.org/wiki/Digital_Signature_ Algorithm.

TÓM TẮT

XÂY DỰNG LƯỢC ĐỒ CHỮ KÝ SỐ MỚI DỰA TRÊN BÀI TOÁN LOGARIT RỜI RẠC

Bài viết này giới thiệu lược đồ chữ ký số DSA. Dựa trên lược đồ chữ ký số DSA, chúng tôi phát triển một lược đồ chữ ký số mới trên vành Z_n có độ phức tạp tính toán tương đương với lược đồ chữ ký số DSA, khắc phục được một số nhược điểm của lược đồ chữ ký số DSA và có thể ứng dụng trên thực tế.

Từ khóa: Lược đồ chữ ký số, Hàm băm.

Received date, 13th Sep., 2017
Revised manuscript, 10th Oct., 2017
Published, 1st Nov., 2017

Author affiliations:

[1] Academy of Military Science ;

[2] Military Technical Academy;

[3] Academy of Cryptography Technique.

[*] Corresponding author: levantuan71@yahoo.com.