Nos-Santos-Izquierdo Field

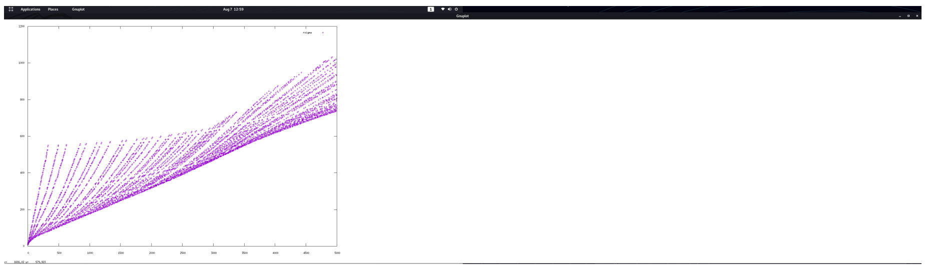

Figure 1: Sum of factors

The image shows the expansion of the sum of factors in semiprimes.

PRIME PERIOD GRIMOIRE VOL. 2

This grimoire is a draft under continuous development.

It explains how the Nos-Santos-Izquierdo Field (NSIF) works, focusing in the similarities between the RSA problem, factorization, and the calculation decimal expansions.

RSA is used to verify if a number is a valid divisor of the period in the decimal expansion.

The NSIF method is an efficient way to calculate the N decimal expansion, for semiprimes.

The NSIF method allows calculating the sum of factors in a efficient way, without knowing the prime factors, when the product of primes are in the field.

The NSIF method improves factorization methods for the majority of small numbers and a significant percentage of large numbers.

The NSIF method proofs factorization and decrypt a message with RSA are diferent problems, depending N is more fast factorize or decrypt a message.

The code of Nos-Santos-Izquierdo Field has been written in the Haskell programming language.

Grimoire basic spells

These spells are from famous math researchers, including Fermat. Euler. Carmichael.

- The RSA problem
  - Encrypt

m = powMod m e n = mc

- Decrypt

  mc = powMod

- Other known formulas

  p^2 mod 6 = 1

  N^2 = x6+1 * y6+1

  Totient = N - (sum factors) - 1

  totient mod carmichael = 0

  period T = powMod 10 T N = 1

  carmichael(n) mod T(n) = 0

  totient(n) mod T(n) = 0

  x^2 - y^2 mod N = 0

Spells logics

Any number can be represented by

N = time * period + (sum factors N) - 1

Nos semiprimes are the product of primes which follow the following formulas:

Deduction of a perfect prime square:

p^2 * p^2 = n^2

Deduction for different factors:

p^2 * q^2 = n^2

Then

p^2 * q^2 = n^2 and p^2 mod 6 = 1

Then

t = times of decimal expansion length

((sqrt x6+1)* (sqrt y6+1)) - ((sqrt 6x+1) + (sqrt y6+1)) +1 = t * T -> RSA Solution

p^2 - p mod T = 0

Then in squares or numbers with square proportion

N^{2+N}2 - N^2 mod T = 0

or

n^2 - x mod T = 0

Sum Factors q p = (mod N T) * t + 1

Conjure spells

```
-- COMPUTE CARMICHAEL DERIVATION
-- The first paremeter is n (the semiprime) and the second the Nos-Santos-Izquierdo Field.

nsf n s = (n^(2) - 1) -s


-- EXTRACT PRIVATE KEY WITH EXPONENT AND N IN NSS NUMBERS

nss_privatekey e n s= modular_inverse e (nsf n s)

-- EXTRACT FACTORS in NSIF numbers

nsf_factorise_ecm n = (sg2-qrest, sg2+qrest)
    where
    sigma = (n+1)-(totient n)
    sg2 = div sigma 2
    qrest = integerSquareRoot ((sg2^2)-n)


nsf_factorise n t= (sg2-qrest, sg2+qrest)
    where
    sigma = ((n+1))-(t)
    sg2 = div sigma 2
    qrest = integerSquareRoot ((sg2^2)-n)




-- MAP NSIF PRODUCT OF PRIMES

-- N bits mapping

--for strong nss

nsf_map s x r= map fst (filter (\(x,c)-> c==0) $ map (\x-> (x,tryperiod x (nsf x r))) ([2^s.


-- N bits mapping checking with ECM just products of two primers
```

```
nsf_find nbits range to = take to $ filter (\(v,c)-> length c==2) (map (\x-> (x,P.factorise

-- N bits mapping without perfect squares nor prime numbers it is very slow at checking prim

nsf_map_nsq s x r =  filter (\(d)-> snd (integerSquareRootRem d) /= 0 ) (nsf_map s x r)


-- CHECK PERIOD LENGTH FOR N Using RSA
tryperiod n period = (powMod (powMod (2) 1826379812379156297616109238798712634987623891298413

-- GET DIVISORS WITH ECM METHOD
divs n = read $ concat (tail (splitOn " " (show (divisors n)))):: [Integer]

-- GET SUM OF FACTORS WITH ECM

sum_factors n = n + 1 - (totient n)


-- DECIMAL EXPANSION, THE PERIOD



-- Efficient way to calculate decimal expansion in semiprime numbers

-- With P Q
tpq p q = out
    where
    tp = div_until_mod_1 (p-1) (p-1)
    tq = div_until_mod_1 (q-1) (q-1)
    out = (lcm tp tq)


-- With N and ECM
tn n = tp
    where
    c = carmichael n
    tp = div_until_mod_1 (c) (c)


div_until_mod_1 p last
    | period == 1 = div_until_mod_1 dp dp
    | mp /= 0 = last
    | otherwise = last
    where
    (dp,mp) = divMod (p) 2
    period = powMod 10 dp (p+1)



-- Decimal expansion in a traditional slow way
period n = (length (takeWhile (/=1) $ map (\x -> powMod 10 x n ) ( tail [0,1..n])) ) +1

-- All numbers which decode msg, decode a number in a different kind of field

alldecnss n = filter (\(c)-> tryperiod n (n^2) == 0 || tryperiod n (n^2 - c-1)==0 || tryperi


alldec2 n = take 1000 $ filter (\(z,y) -> y == 0 ) (map (\x-> (x , tryperiod n ((x^2) + (x*6


alldec n = filter (\(z,y) -> y == 0) (map (\x->(x,tryperiod n x)) [1..n])
```

Casting spells, nsiZ = 0

```
-- We search for NSS numbers among 512 bits and 512 bit + 1000

*Nss> nsf_map_nsq 512 1000 (0)

[134078079299425970995740249982058461274793658205923933777235614437217640300735469768018742

(1.14 secs, 1,208,564,264 bytes)

-- We search for NSS numbers among 2048 bits and 2048 bit + 1000

*Nss> nsf_map_nsq 2048 1000 (0)

[32317006071311007300714876688669951960444102669715484032130345427524655138867890893197201

(16.11 secs, 8,801,785,040 bytes)

-- We search for NSS numbers among 4096 bits and 4096 bits + 1000

*Nss> nsf_map_nsq 4096 1000 (0)

[10443888814131525066917527107166243825799642490473837803842334832839539079715574568488268

(85.54 secs, 29,012,371,256 bytes)
```

```
*Ncs> P.factorise

32317006071311007300714876688669951960444102669715484032130345427524655138867890893197201411

[(Prime 974849,1),(Prime 319489,1),(Prime 3560841906445833920513,1),(Prime 16798855634176047

(255.44 secs, 139,784,410,360 bytes)

-- The same number with nsif, the result can be used just to decrypt messages

*Ncs> nsf_derivate

32317006071311007300714876688669951960444102669715484032130345427524655138867890893197201411

10907481356194159294629842447337828624482641619962326924318327861897213318491192952162642345

(0.01 secs, 1,948,808 bytes)
```

The spell show how easy is to solve faster many numbers than the factorization for message decryption.

Conclusion

For all products of primes who have the same proportion of n - sum factors in n^2 it is significantly easier to calculate a private key to decrypt a message.

When a multiple of carmichael is near n^2 is more easy decrypt than factorize.

The spells proves that find and decrypt with a public key and NSIF is faster than factorizing just one of the numbers, if you just want to decrypt a message you can use the NSIF if the number is "vulnerable", just in 0.01 seconds vs 255 seconds to factorize the number with the parallel ECM method.

Clearly in trillions of cases you can decrypt or solve rsa significantly faster than the current fastest factorization methods like GNFS or ECM when (p * q) are

n^2 - NSIF(e,x) mod T = 0

Many keys are vulnerable to find the message deciphered in just one operation, and some operations more to factorize the number if you can factorize the NSIF derivation.

Just some seconds to get keys of 512, 1024, 2048 and 4096 bits vulnerable to the Nos-Santos-Izquierdo Field

A method exists to calculate that for all numbers directly grouped by this kind of field. You need to factorize after you know the field, the result of the computation of NSS. This means the factorization process is a bit more expensive than decryption process.

A number smaller than Carmichael needed to decrypt can also be calculated, maybe in some cases such number is better for computation because of its size, and when the period is smaller than Carmichael the modular inverse of exponent and Carmichael can be smaller. This means in some cases decryption can be performed faster.

T(n) = lcm T1 T2

The RSA algorithm allows us to check divisors of the period. We can know if something is a divisor of the period, the Carmichael number or the Euler Totient. When RSA is used to decrypt messages.

```
tryperiod n period = (powMod (powMod (2) 18263798123791562976161092387987126349876238912984
```

tryperiod computes the RSA algorithm to encrypt and decrypt a test number.

The maps of the demonstration result in the numbers who have the same period n^2 - NSIF(e,x), which decrypt the message "2".

This means the same function with the same parameters can decrypt infinitely different public key groups.

PrivateKey = PublicKeyN^2 - NSIF(PublicKeyE, x)

The exponent changes the field but always contains Totient, Carmichael and period numbers and derivations of them.

This seems obvious, but it is awesome how the decimal expansion of a number is really close to the square, allowing us to group the numbers by fields. Of course you can find new fields each time you increase the numbers, from just a few fields for small numbers, more fields appear as you define larger numbers. But the same field which works with small numbers also works in 256, 512, 1024 and 2048 bit ones.

The numbers can stay in didferent Period Fields at the same time. Because different numbers have the same decimal expansion and they cross with others in his expansion.

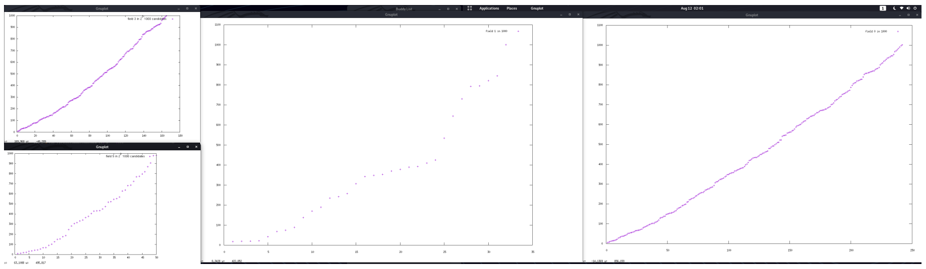

Figure 2: Sum of factors

General Decimal Expansion Longitude Field Conjecture

All numbers in a nos santos field can be factorized just by one operation. Each image represents the numbers who can by decripted in each field, with the same field you can decrypt all different numbers, the image is in a small scale to understad how it works.

All numbers can be explained as :

N = t * T + (SUM FACTORS) - 1

Decimal Expansion Longitude = (T) or Period

All numbers can be grouped by his period (T) field . The distance among n^2 and T multiple in any number

nsif n e x = (powMod (powMod (2) e n) (modular_inverse e n^2-x) n) - (2) == 0

nsiZ = n^2 - NSIF(n,e,x)

To calculate decimal expansion distances of squares. The nos santos field. The X is the field

*e allways a coprime of period , totient or carmichael. using a big prime number is enougth.

T(n^2) = lcm T(n) n

factors N T

```
P = ((N - T) +1 / 2 +1) +  sqrt ( ((N - T) +1 / 2 )^2 - N )

Q = ((N - T) +1 / 2 ) -  sqrt ( ((N - T) +1 / 2 )^2 - N )
```

Period of decimal expansion

```
-- With P Q
tpq p q = out
    where
    tp = div_until_mod_1 (p-1) (p-1)
    tq = div_until_mod_1 (q-1) (q-1)
    out = (lcm tp tq)


-- With N and ECM
tn n = tp
    where
    c = carmichael n
    tp = div_until_mod_1 (c) (c)


div_until_mod_1 p last
    | period == 1 = div_until_mod_1 dp dp
    | mp /= 0 = last
    | otherwise = last
    where
    (dp,mp) = divMod (p) 2
    period = powMod 10 dp (p+1)
```

The fields are crossing the lines of carmichael expansion (private keys)

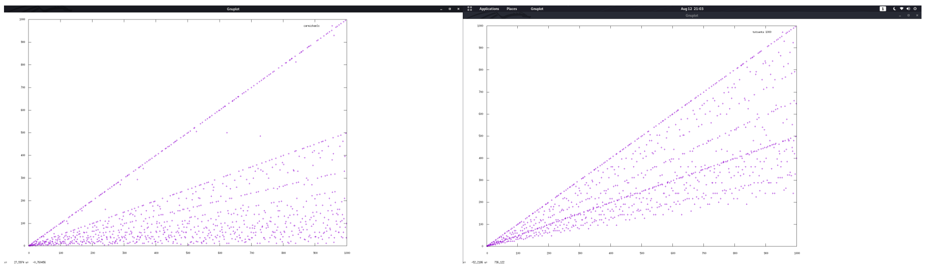

Figure 3: Sum of factors

nsZ = n^2 - NSIF(n,e,x)

Example and proof of a field englobes a group of period , carmichaels or totient (private keys), and same numbers can be found in diferent fields.

Just one operation to decrypt messages, one quadratic more to factorize, one more to calculate the decimal expansion of N, with the same "KEY" for all group.

```
map 1000 numbers from 2^2 to 2^2+1000 who are in the field 5

*Nss> e=nsf_map_nsq 2 1000 (5)

(0.01 secs, 0 bytes)

*Nss> e

[6,10,11,18,21,31,35,42,45,54,66,69,90,101,126,150,154,174,186,246,281,306,315,331,341,366,3

(0.09 secs, 89,292,144 bytes)

get period from  e mapping

*Nss> map period it

[7,11,2,19,6,15,36,43,46,55,67,22,91,4,127,151,155,175,187,247,28,307,316,110,30,367,379,407

(0.07 secs, 51,419,840 bytes)

get carmichael from e mapping

*Nss> map carmichael e

[2,4,10,6,6,30,12,6,12,18,10,22,12,100,6,20,30,28,30,40,280,48,12,330,30,60,18,84,84,430,30,
(0.01 secs, 328,512 bytes)


get totient from e mapping

*Nss> map totient e

[2,4,10,6,12,30,24,12,24,18,20,44,24,100,36,40,60,56,60,80,280,96,144,330,300,120,108,168,16
```

As you can see diferrent private keys can be solved from all collection of field 5

Proof of more multiples of x can be found than multiples of factors from 0 to N, using NSIF fields.

```
-- function to try all fields until N, tunned with 2*3 because all carmichael numbers can be
is to reduce number of loops , because we are just interested in multiples of 6

-- improbes rsa method for decryption.


alldec2 n = take 1000 $ filter (\(z,y) -> y == 0 ) (map (\x-> (x , tryperiod n ((x^2) + (x*6


*Nss> alldec2 1189

[(1184,0),(1170,0),(1155,0),(1134,0),(1130,0),(1120,0),(1114,0),(1100,0),(1094,0),(1092,0),(

*Nss> length it


157


*Nss> P.factorise 1189


[(Prime 29,1),(Prime 41,1)]

*Nss> 41+29

70
```

This demostrates how de multiples of divisors of carmichael can decrypt message "2" in 157 from 0 to 1189, and how just 70 multiples of 29 and 41 are to be found from 0 to 1189

In probability terms allways is more probable find a number who decrypts than a factor of N if the longitudes of the periods of decimal expansion of the semiprime are small.

This probability changes when the prime factors have a long or large period.

Increase the probability of succes for x value when makes the product * 6

This means the probability to decrypt is linked to the period longitude behavior not to factors factorization.

The probability to decrypt rsa are de multiples of carmichael divisors when is applied as a Nos-Santos-Izquierdo Field

Proof more numbers from 0 to N who decrypts a message than factorize a number

Calculate numbers who decrypts a message from 0 to N

```
*Nss> length $ alldec2 427

94

*Nss> length $ alldec2 203

30

*Nss> length $ alldec2 323

78

*Nss> length $ alldec2 377

58

*Nss> length $ alldec2 2049

409
```

As wikipedia say .... Rivest, Shamir, and Adleman noted[2] that Miller has shown that – assuming the truth of the Extended Riemann Hypothesis – finding d from n and e is as hard as factoring n into p and q (up to a polynomial time difference).However, Rivest, Shamir, and Adleman noted, in section IX/D of their paper, that they had not found a proof that inverting RSA is equally as hard as factoring. ...

Calculate numbers from 0 to N , who have common factors p or q, (P + Q)

```
*Libs.Events> rsigma 377

42

*Libs.Events> rsigma 427

68

*Libs.Events> rsigma 1189

70

*Libs.Events> rsigma 1121

78
```

```
*Libs.Events> rsigma 767
72
*Libs.Events> rsigma 323
36
*Libs.Events> rsigma 2049
686
```

When the primes are weak (with small period) you have a lot more probabilities than in factorization to decrypt the message.

Seems clear in some cases you have mor multiples who gcd n multiple = factor is more probable, and others is more probable decrypt befor find a multiple with common factor from 0 to N.

```
*Libs.Events> rsigma 2049
686
```

This is a contradiction , assume than Rivest, Shamir, and Adleman demostration is true. Because decrypt and factorize are linked to different things, the first to the length of decimal expansion period , the second to the value of factors of N.

PROBABILITIES TO DECRYPT > PROBABILITIES TO FACTORIZE

Like more semiprimes are composed by by weak primes in period terms, than strong periods, we can deduce more semiprimes going to have more probabilities to be decrypted than to be factorized.

In the nss.hs you can review and test with the library.

License

Contributions and collaborations

All contributions are welcome and collaborations are welcome.

If you want to collaborate as a coder in the project to develop libraies and research experiments you are invited. please contact in pedro@blackhole.consulting

Donations of hardware to build a super computer are welcome, we need graphic cards, micro processors , ram and all kind of computer components to build a cluster for make more fast the investigation. Old gpu rig miners are welcome.

If you have a place with some KW of energy we are glad to build the computer in your place.

If you have a super computer we are glad to do experiemnts toguether.

We are a small group of geeks without resources

Serious paper will be realease soon

Authors

Main idea, investigation and dirty haskell functions.

Author - Vicent Nos Ripolles, Consultant, Dev, Cybersecurity Auditor, Bussinesman, Hacker (srdelabismo) Spain

Author - Pedro el Banquero - Banker SysAdmin - Panamá

Functions and Performance Haskell and Maths

Co-Author - My master, Enrique S., Physhicist, Maths, Computer Science (MathMax) Spain

Enrique help me to develop all functions and explain me how rsa works and all that i know about maths and cryptografy. He was investigating factorization more than 8 years.

Co-Author - Francisco Blas Izquierdo, Hacker, Ph.D. student, Chalmers University of Technology" (KLONDIKE) Sweeden

Francisco, checking carmichael divisors using RSA, is his most relevant contribution to this paper.