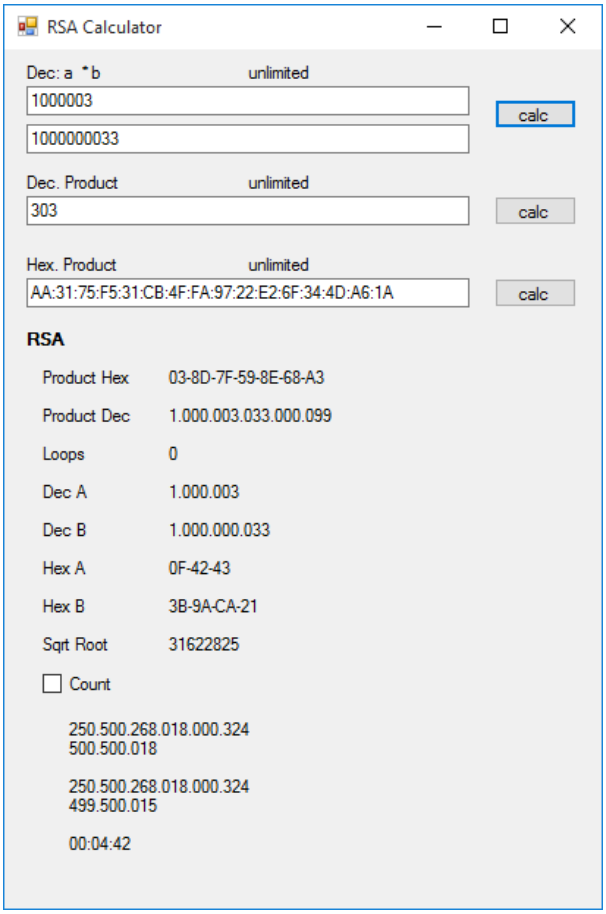
RSA Cryptosystem Optimisation

Mario GRÄF

My goal was to found a solution for the asymmetrical cryptographically algorithms of the RSA System. This type of encrypting applies since 1977 as standard of extreme safety. Until now nobody was able to determine the RSA. The reason is the multiplication of prime numbers, so natural numbers of which product exclusive one option does exist. The longer the code and the further the side lengths, the more difficult is to calculate the coefficients.



The product is unencrypted. Only until of knowing one of the factors a message of modulated information is able to decrypt. The USB-Debug modus of Android Smartphone is encrypted by a value of 16Byte, which corresponds to a value of $3.4 * 10^{38}$. These key seems despite present technology incomputable.

I started a C#.NET project for convert number systems of arbitrary size. The data type *ulong,* as UINT64 is able to process a value with maximum 8 Byte. Any function of the fundamental operations is limited to these value. Therefore it was necessary to generate an expandable data type of a wrapper class.

The function of these class calculates the maximum of the memory consumption and saves the value from a *string* to Byte array. The C# runtime allows data type values up to 1GB. To manage the fundamental operations methods based on shift registers were created. So the problem was solved but to extract a root seems not to be accomplish. I decided to recreate the logarithm of 2 method to calculate how many bit were in use. Divide it by 2 and increment in a small loop to wipe out fuzziness. This functions works perfect.

At the project start I notice that the RSA is a cuboid its area is known but not the side length. Each cuboid is in relation to its area distant at a square factor. Out of it two square sequences are able to point at their intersection at unknown peak. For this it is necessary to increment 1 to the square value per loop and compare. If both figures are ident the function square was found.

An example

The public key 115 as product of the unknown figures 5 and 23 result of the square root 10.72 and have to be round up, because x and y cannot be smaller together.

$$5 * 23 = 115$$

$$\sqrt{115} = 10.72$$

$$10.72 \rightarrow 11$$

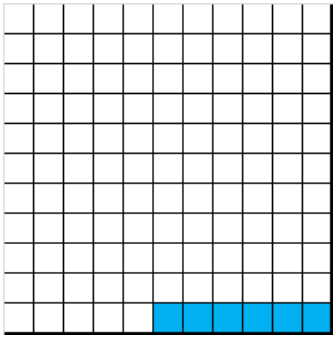
Q as origin square with a side length of 11 result an area of 121. The difference to the cuboid is 6 and its characteristic is not square.

Loop 0

$$a = 11$$

$$11^2 = 121$$

$$121 - 115 = 6$$



n loops with incrimination of the side length until the difference create a square

Loop 1

$$a = 12$$
$$12^2 - 115 = 29$$

Loop 2

$$a = 13$$
$$13^2 - 115 = 54$$

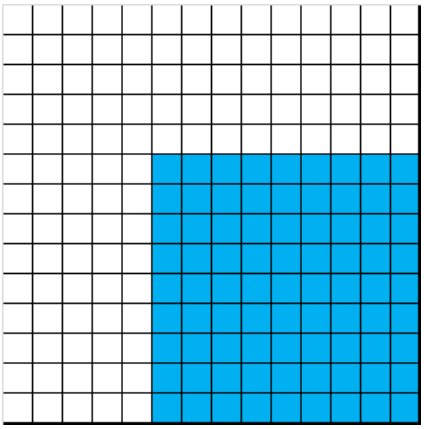
Loop 3

$$a = 14$$

$$14^2 = 196$$

$$196 - 115 = 81$$

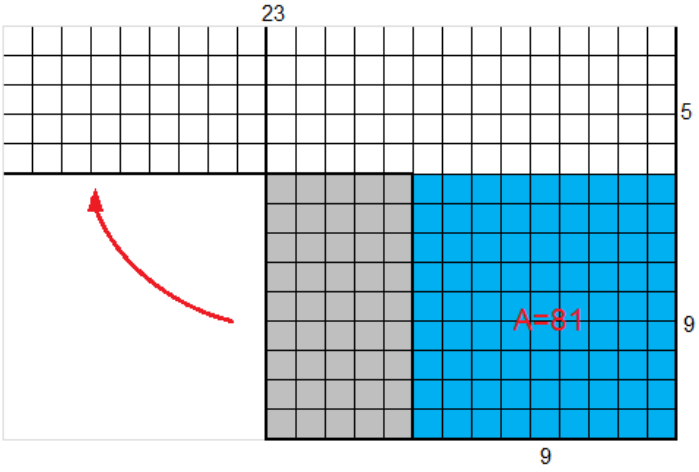
$$\sqrt{81} = 9$$



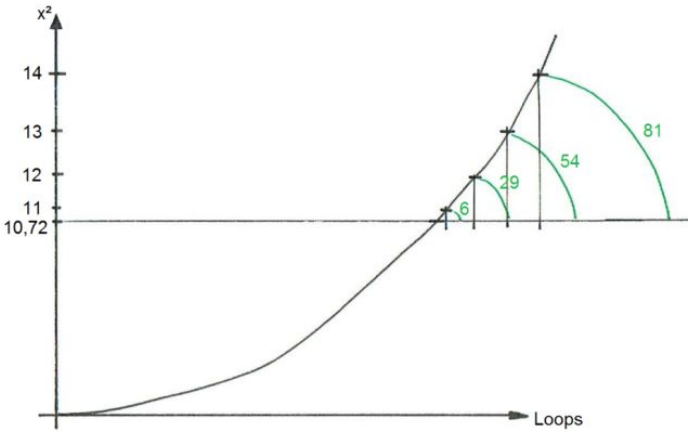
Q with a side length of 14 is in proportion to desired cuboid because the difference is square and so it's able to move the fraction.

$$14 + 9 = \mathbf{23}$$

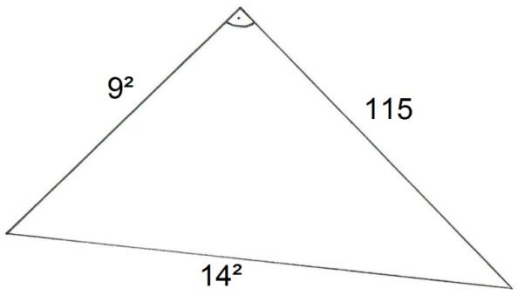
$$14 - 9 = \mathbf{5}$$



In a chart the operation is demonstrated.



The rules of Pythagoras were inserted. The increase and slowdown function have to be square.



Since the insight of square conduct in the RSA Code I had programmed the core process in only two days and optimise it every day. On the strength of integration to multithreads 30 digits in length public keys are able to compute to its private keys in less than one minute.