Can Schönhage multiplication speed up the RSA decryption or encryption?

(extended abstract)

Luis Carlos Coronado García
FB 20, Technische Universität Darmstadt
Hochschulstr. 10, D-64289, Darmstadt. Germany.
coronado@cdc.informatik.tu-darmstadt.de
http://www.cdc.informatik.tu-darmstadt.de/mitarbeiter/coronado.html

1 Introduction

The RSA cryptosystem makes use of modular exponentiation for encrypting and decrypting and, intrinsically, multiplication of integers. The lenght of which is the one of the RSA-modulus. In this paper, we describe the multiplication algorithms Karatsuba, Toom Cook and Schönhage, in order to calculate the number of multiplications of base words ($\mathcal{MOB}$). In our calculations the additions of base words are included. A multiplication of two base words is considered as a unit of computation. One addition of two base words is considered as q times one unit of computation. The value q ($0 < q \leq 1$) depends on the architecture or platform.

We use Zimmermann's version of the Schönhage algorithm [Zim92] and suggest an improvement. In Zimmermann's version a parameter κ must be fixed in advance, which has been experimentally chosen for some sizes of the integers to be multiplied, and therefore part of our contribution is to show which adequate value for such κ can be selected in order to make less $\mathcal{MOB}$ using the Schönhage algorithm in compare with the use of the other ones for big integers. Our result is that the Schönhage algorithm is the best amongst the examined ones for numbers whose length is greater than or equal to 2^{17} bits.

2 Notation

Let $\nu_0 \in \mathbb{Z}^+$ and $\mathcal{B} = 2^{2^{\nu_0}}$. A *base word* is an x such that $0 \leq x < \mathcal{B}$. Let $z \in \mathbb{Z}^+$, the *base-word length* (or simply the length) of z is the number $\ell_{\mathcal{B}}(z)$ of base words needed to represent such z. In a similar way we define the bit length $\ell_2(z)$ of z. We define $\ell_{\mathcal{B}}(0) = \ell_2(0) = 1$ and for $z \in \mathbb{Z}^-$ we define $\ell_{\mathcal{B}}(z) := \ell_{\mathcal{B}}(-z)$ and $\ell_2(z) := \ell_2(-z)$.

Let $s_a = a_{m-1} \cdots a_1 a_0$ be a not null bit string. We consider it as the integer $a = \sum_{j=0}^{m-1} a_j 2^j$ and we say that its bit length is $\ell_2(s_a) = m$ and its base-word length is $\ell_{\mathcal{B}}(s_a) = \lceil \frac{m}{2^{\nu_0}} \rceil 2^{\nu_0}$. So, if we want to add two integers $\alpha = \sum_{j=0}^{n_1-1} \alpha_j 2^j$ and $\beta = \sum_{j=0}^{n_2-1} \beta_j 2^j$, we can consider the bit strings $s_\alpha = \alpha_{m-1} \cdots \alpha_1 \alpha_0$ and $s_\beta = \beta_{m-1} \cdots \beta_1 \beta_0$, where $m = \max\{n_1, n_2\}$ and $\alpha_i = \beta_j = 0$ for $i \geq n_1$ and $j \geq n_2$. In this way we can define the number of additions of base words for a couple of integers of bit length m as $\mathrm{add}_{\mathcal{B}}(m) = \lceil \frac{m}{2^{\nu_0}} \rceil$.

We take a multiplication of two base words as a unit. We suppose that an addition of two base words is equivalent to q multiplications of two base words, where $0 < q \leq 1$ depends on the implementation. Therefore the addition of two m-bit integers is equivalent to $q\mathrm{add}_{\mathcal{B}}(m)$ multiplications of base words.

If $x, n, B \in \mathbb{Z}^+$, such that $B > 1$ and $x = \sum_{j=0}^{m} x_j B^j$, then $xB^n = \sum_{j=0}^{m} x_j B^{j+n}$ will be denoted by $x <<_B n$, an n *B-block right shift*. The symbol | represents the bitwise or.

3 Multiplication algorithms and $\mathcal{MOB}$

We describe some multiplication algorithms and calculate the number of multiplications of base words for each one of them, in order to compare those quantities with each other. In all subsections we assume $\mathcal{B} = 2^{2^{\nu_0}}$ for a previously set $\nu_0 \in \mathbb{Z}^+$, which defines our base words. We suppose that the two integers to be multiplied in each algorithm have the same length N.

3.1 Naïve algorithm

The complexity of this algorithm is $O(N^2)$. In Figure 1 a description of this algorithm can be found. Four multiplication must be made, one addition of integers of size ℓ' and one addition of size $2\ell'$ as well. We could write this as 4 mult and 1 $\text{add}_{\mathcal{B}}(\ell')$ and 1 $\text{add}_{\mathcal{B}}(2\ell')$. Note that one addition of integers of size ℓ' and one addition of size $2\ell'$ is equivalent to $\left\lceil \frac{\ell'}{2^{\nu_0}} \right\rceil + \left\lceil \frac{2\ell'}{2^{\nu_0}} \right\rceil = 3 \left\lceil \frac{\ell}{2^{\nu_0}} \right\rceil$ additions of base words. And then we have that $\mathcal{MOB}(\texttt{Na}, \ell)$ is $4^r \mathcal{MOB}(\texttt{Na}, \frac{\ell}{2^r} + 2^{\nu_0} \sum_{i=0}^{r-1} \frac{c_{r-i}}{2^i}) + 3q \left(\frac{\ell}{2^{\nu_0}}(2^r - 1) + \sum_{i=0}^{r-1} 4^i \sum_{j=0}^{i} \frac{c_{i-j+1}}{2^j} \right)$, where $0 \leq c_i < 1 \ \forall i$. For instance, if $\ell = 2^{\nu_0+\nu}$, where $\nu \geq 0$ is an integer, then $c_i = 0 \ \forall i$ and $\mathcal{MOB}(\texttt{Na}, \ell) = 4^\nu + 3q(4^\nu - 2^\nu)$.

3.2 Karatsuba algorithm

The complexity of this algorithm is $O(N^{\log_2 3})$. In Figure 1 a description of this algorithm can be found. Three multiplication must be made, two additions of integers of size $\frac{\ell'}{2}$, two additions of integers of size ℓ' and one addition of size $2\ell'$ as well. Note that two additions of integers of size $\frac{\ell'}{2}$, two additions of integers of size ℓ' and one addition of size $2\ell'$ are equivalent to $2 \left\lceil \frac{\frac{\ell'}{2}}{2^{\nu_0}} \right\rceil + 2 \left\lceil \frac{\ell'}{2^{\nu_0}} \right\rceil + \left\lceil \frac{2\ell'}{2^{\nu_0}} \right\rceil = 5 \left\lceil \frac{\ell}{2^{\nu_0}} \right\rceil + \left(\left\lceil \frac{\ell}{2^{\nu_0}} \right\rceil \mod 2 \right)$ additions of base words. Therefore, we have that $\mathcal{MOB}(\texttt{Ka}, \ell)$ is $3^r \mathcal{MOB}(\texttt{Ka}, \frac{\ell}{2^r} + 2^{\nu_0} \sum_{i=0}^{r-1} \frac{c_{r-i}}{2^i}) + 10q \left(\frac{\ell}{2^{\nu_0}} \cdot \frac{3^r - 2^r}{2^r} + \sum_{i=0}^{r-1} 3^i \sum_{j=0}^{i} \frac{c_{i-j+1}}{2^j} \right) + C_r(\ell)q$, where $0 \leq c_i < 1 \ \forall i$ and $0 \leq C_r(\ell) \leq \frac{3^r - 1}{2}$. For instance, if $\ell = 2^{\nu_0+\nu}$, where $\nu \geq 0$ is an integer, then $c_i = 0 \ \forall i$, $C_r(\ell) = 0$ and $\mathcal{MOB}(\texttt{Ka}, \ell) = 3^\nu + 10q(3^\nu - 2^\nu)$.

3.3 Toom-Cook algorithm

The complexity of this algorithm is $O(N^{\log_3 5})$. In Figure 1 a description of this algorithm can be found. This is a faster algorithm; it is better than the Karatsuba one for integers of bit length greater than or equal to 2^{14}, under the assumption that the size of a base word is 2^5 bits. This behaviour is owned to the linear system which must be solved and which increases the overhead of the algorithm. In the Toom-Cook algorithm represented in Figure 1 five multiplication must be made, 12 additions of integers of size $\frac{\ell'}{3}$, 6 additions of integers of size $\frac{2\ell'}{3}$ and one addition of integers of size $2\ell'$ as well. If we define $c_1 = t_{\frac{1}{2}} - \gamma_4 - 16\gamma_0$, $c_2 = t_1 - \gamma_4 - \gamma_0$ and $c_3 = t_2 - 16\gamma_4 - \gamma_0$, because of all elements in the equation that must be solved are integers, c_1 and c_3 must be even. Therefore, we can set $c_1' = \frac{c_1}{2}$ and $c_3' = \frac{c_3}{2}$ and then the solution for the linear system is

$$\begin{aligned} 3\gamma_1 &= c_1 - 6c_2 + c_3' \\ \gamma_2 &= -c_1' + 5c_2 - c_3' \\ 3\gamma_3 &= c_1' - 6c_2 + c_3 \end{aligned}$$

Note that $5x = (x <<_2 2) + x$ and $6x = ((x <<_2 1) + x) <<_2 1$ and then we have 9 additions of numbers of size $\frac{2m}{3}$.

Therefore, the solution of the linear system requires 9 additions more of integers of size $\frac{\ell'}{3}$.

Note that 12 additions of integers of size $\frac{\ell'}{3}$, 15 additions of integers of size $\frac{2\ell'}{3}$ and one addition of integers of size $2\ell'$ is equivalent to $12\left\lceil \frac{\frac{\ell'}{3}}{2^{\nu_0}} \right\rceil + 15\left\lceil \frac{\frac{2\ell'}{3}}{2^{\nu_0}} \right\rceil + \left\lceil \frac{2\ell'}{2^{\nu_0}} \right\rceil = 48\left\lceil \frac{\ell}{3\cdot 2^{\nu_0}} \right\rceil$ additions of base words. And then we have that $\mathcal{MOB}(\mathtt{T3}, \ell)$ is $5^r\mathcal{MOB}(\mathtt{T3}, \frac{\ell}{3^r} + 2^{\nu_0}\sum_{i=0}^{r-1}\frac{c_{r-i}}{3^i}) + 48q\left(\frac{\ell}{2^{\nu_0+1}}\cdot\frac{5^r-3^r}{3^r} + \sum_{i=0}^{r-1}5^i\sum_{j=0}^{i}\frac{c_{i-j+1}}{3^j}\right)$ where $0 \le c_i < 1\ \forall i$. For instance, if $\ell = 2^{\nu_0+\nu}$, where $\nu \ge 0$ is an integer, then $\ell = 3^{\nu\log_3 2}2^{\nu_0}$ and $\mathcal{MOB}(\mathtt{T3}, \ell) = 5^{\nu\log_3 2} + 24q(5^{\nu\log_3 2} - 2^\nu) + C(\ell)$, where $C(\ell) < 18q5^{\nu\log_3 2}$.

3.4 Schönhage algorithm

The complexity of this algorithm is $O(N\log N\log\log N)$. This algorithm takes advantage of the Fast Fourier Transform (FFT) whose complexity is $O(M\log M)$, where M is the number of elements to be transformed. The FFT is computed in the ring $R = \mathbb{Z}/(2^m+1)\mathbb{Z}$, where m is a power of two and $\zeta_p = 2^q$ is a p primitive root of unit in R if $pq = 2m$.

Proposition 3.1 *Consider the function* $n(\kappa) = \left\lceil 2^{\nu_0+\nu+1-2\kappa} + \frac{\kappa+3}{2^\kappa}\right\rceil 2^\kappa$ *for an integer* κ *and fixed real numbers* ν *and* ν_0. *If*

1. $\left\lceil \frac{\nu_0}{2}\right\rceil \ge 3$, *then for* $\left\lceil \frac{\nu_0}{2}\right\rceil \le \kappa \le \left\lfloor \frac{\nu_0+\nu}{2}\right\rfloor$ *we have that* $n(\kappa)$ *is a non increasing function and therefore* $n(\left\lfloor \frac{\nu_0+\nu}{2}\right\rfloor) \le n(\kappa)$.

2. $\kappa \ge 4$, *then for* $\left\lceil \frac{\nu_0+\nu}{2}\right\rceil + 1 \le \kappa < \nu_0+\nu$ *we have that* $n(\kappa)$ *is an increasing function and therefore* $n(\kappa) \ge n\left(\left\lceil \frac{\nu_0+\nu}{2}\right\rceil + 1\right)$.

For the proof of part 1 we have that if $3 \le \left\lceil \frac{\nu_0}{2}\right\rceil \le \kappa$, then $\frac{\kappa+3}{2^\kappa} < 1$. If $\kappa \le \frac{\nu_0+\nu}{2}$ then $2^{\kappa+1} \le 2^{\nu_0+\nu+1-\kappa}$ and therefore $3\cdot 2^\kappa \le 2^{\nu_0+\nu+1-\kappa} + 2^\kappa$.

Now, if $3 \le \left\lceil \frac{\nu_0}{2}\right\rceil \le \kappa, \kappa+1 \le \left\lfloor \frac{\nu_0+\nu_0}{2}\right\rfloor$, then $n(\kappa+1) = 2^{\kappa+1} + 2^{\nu_0+\nu+1-\kappa-1} = \frac{1}{2}(2^\kappa + 2^{\nu_0+\nu+1-\kappa}) + \frac{3}{2}\cdot 2^\kappa \le n(\kappa)$. And therefore $n\left(\left\lfloor \frac{\nu_0+\nu}{2}\right\rfloor\right) \le n(\kappa)$ for all integer $3 \le \left\lceil \frac{\nu_0}{2}\right\rceil \le \kappa \le \left\lfloor \frac{\nu_0+\nu}{2}\right\rfloor$. Note that in this case $2^{\kappa+log_2 3} \le n(k) < 2^{\kappa+1+\log_2 3}$ and if $\nu_0+\nu > 6$, then $\nu_0+\nu > \frac{3}{2}(k+1+\log_3)$ and therefore $n(k) < 2^{\frac{2}{3}(\nu_0+\nu)}$.

For the proof of part 2 we have that if $\max\{4, \frac{\nu_0+\nu}{2}+1\} \le \kappa$, then $\frac{\kappa+3}{2^\kappa} + 2^{\nu_0+\nu+1-2\kappa} \le 1$ and therefore $n(\kappa) = 2^\kappa$.

Part of the Schönhage algorithm represented in Figure 1 was described by Zimmermann in [Zim92] with a previously fixed parameter κ. Because of the proposition 3.1, we use $4 < \nu_0$ and $4 \le \kappa$, in order to add in that description the lines denoted by ***.

If $\ell_{\mathcal{B}}(\alpha) = \ell_{\mathcal{B}}(\beta) = 2^\nu$ and we want the product $\alpha\beta$ as integer (instead of $\bmod 2^{\nu_0+\nu}$), it must be used $m = 2^{\nu_0+\nu+1}$. If $8 \le \nu_0+\nu$, then in order to compute $\mathcal{MOB}(\mathtt{Sch}, 2^{\nu_0+\nu})$ 2^κ multiplications of integers of bit length $n(\kappa)$ are required, two FFT of 2^κ elements of bit length $2^{\nu+\nu_0-\kappa}$ and one FFT of 2^κ elements of bit length $2^{\nu+1+\nu_0-\kappa}$ as well, where $\kappa = \left\lfloor \frac{\nu_0+\nu}{2}\right\rfloor$ i.e. $\mathcal{MOB}(\mathtt{Sch}, 2^{\nu_0+\nu}) = 2^\kappa\mathcal{MOB}(\mathtt{Sch}, n(\kappa)) + q\kappa 2^{\nu+3}$.

4 Comparison amongst the multiplication algorithms

From the surveyed algorithms in Section 3 we have obtained the value of $\mathcal{MOB}(\mathrm{MA}, 2^{\nu+\nu_0})$ as follows. For the naïve algorithm (`Na`): $2^{2\nu} + 3q(2^{2\nu} - 2^\nu)$. For the Karatsuba algorithm (`Ka`) :

$2^{\nu \log_2 3}+10q\left(2^{\nu \log_2 3}-2^{\nu}\right)$. For the Toom-Cook algorithm (`T3`): $2^{\nu \log_3 5}+24q\left(2^{\nu \log_3 5}-2^{\nu}\right)+C$; $C < 18q2^{\nu \log_3 5}$. For the Schönhage algorithm (`Sch`): $\mathcal{MOB}(\texttt{Ma}, 2^{\nu+\nu_0})$ if $\nu_0+\nu<8$, where `MA` is an adequate multiplication algorithm, otherwise $2^{\kappa}\mathcal{MOB}(\texttt{Sch}, 2^{\kappa}(2^{\nu+\nu_0+1-2\kappa}+1))+q\kappa 2^{\nu+3}$.

For instance, if $\nu_0 = 5$ and $q = 0.95$ are set, we have that $\mathcal{MOB}(\texttt{Ka}, m) \leq \mathcal{MOB}(\texttt{Na}, m)$ for $2^{10} \leq m$ and $\mathcal{MOB}(\texttt{T3}, m) \leq \mathcal{MOB}(\texttt{Ka}, m)$ for $2^{15} \leq m$. If the Karatsuba multiplication algorithm is used in the Schönhage algorithm, then $\mathcal{MOB}(\texttt{Sch}, m) \leq \mathcal{MOB}(\texttt{T3}, m)$ for $2^{17} \leq m$.

In Figure 2 we show the time needed for multiplying integers of length $2^{\nu_0+\nu}$ with the naïve, Karatsuba, Toom-Cook and Schönhage algorithms. We have used the gmp library [GMP].

5 Conclusion

We take into consideration not only multiplication of base words, but also each addition of them as q times a multiplication. We have shown that `Sch` needs no more multiplications of base words than `Ka`, or `T3` for integers whose bit length is greater than or equal to 2^{17}. Therefore, for a modulus of such size, the modular multiplication [Mon85] and the modular exponentiation [Gor88] would be faster if Schönhage is used as multiplication algorithm. As a consequence the RSA encryption or decryption, too. We have shown also a recurrent formula for computing the number of multiplications of base words for `Sch`. In Figure 2 the graphic of $y = \log_2(\mathcal{MOB}(alg, 2^{log\_size}))$ is shown, where alg is one of the naïve, Karatsuba, Toom-Cook or Schönhage algorithms and $0 \leq log\_size \leq 25$.

6 Acknowledgments

We would like to thank Johannes Buchmann, Evangelos Karatsiolis, Christoph Ludwig, and Ulrich Vollmer for helpful comments and fruitful discussions.

References

[AK63] Yu. Ofman A. Karatsuba. Multiplication of multidigit numbers on automata. *Soviet Physics - Doklady*, 7(7):595–596, 1963.

[GMP] GMP. GNU multiple precision arithmetic library. http://www.swox.com/gmp/.

[Gor88] Daniel M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27:129–146, 1988. Article No. AL970913.

[Knu97] Donald E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, 1997.

[Mon85] Peter L. Montgomery. Modular multiplication without trial division. *Mathemetics of Computation*, 44(170):519–521, April 1985.

[Pol71] J. M. Pollard. The fast fourier transform in a finite field. *Mathematics on computation*, 25(114):365–374, 1971.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[SS71] A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7:281–292, 1971.

[SWV] A. Schönhage, A. F. W.Grotefeld, and E. Vetter. *Fast Algorithms.* Wissenschaftsverlag.

[Zim92] P. Zimmermann. An implementation of Schönhage's multiplication algorithm. Technical report, Loria, 1992.

Naïve Na(α, β)

Input: $\alpha, \beta \in \mathbb{Z}$, s.t. $\ell_{\mathcal{B}}(\alpha) = \ell_{\mathcal{B}}(\beta)$.
Output: $\gamma = \alpha\beta$.
Procedure:

$\ell \leftarrow \ell_2(\alpha)$
if ($\ell \leq 2^{\nu_0}$) return $\alpha * \beta$
$\ell' \leftarrow \left\lceil \frac{\ell}{2^{\nu_0}} \right\rceil 2^{\nu_0}$
$B \leftarrow \frac{\ell'}{2}$
write $\alpha = \alpha_1 B + \alpha_0$ and $\beta = \beta_1 B + \beta_0$
$c_3 \leftarrow$ Na(α_1, β_1)
$c_2 \leftarrow$ Na(α_0, β_1)
$c_1 \leftarrow$ Na(α_1, β_0)
$c_0 \leftarrow$ Na(α_0, β_0)
return $\gamma = ((c_3 <<_B 2)|c0) + ((c_2 + c_1) <<_B 1)$

Toom-Cook T3(α, β)

Input: $\alpha, \beta \in \mathbb{Z}$, s.t. $\ell_{\mathcal{B}}(\alpha) = \ell_{\mathcal{B}}(\beta)$.
Output: $\gamma = \alpha\beta$.
Procedure:

$\ell \leftarrow \ell_2(\alpha)$
if ($\ell \leq 2^{\nu_0}$) return $\alpha * \beta$
$\ell' \leftarrow \left\lceil \frac{\ell}{3 \cdot 2^{\nu_0}} \right\rceil 3 \cdot 2^{\nu_0}$
$B \leftarrow \frac{\ell'}{3}$
write $\alpha = \alpha_2 B^2 + \alpha_1 B + \alpha_0$ and $\beta = \beta_2 B^2 + \beta_1 B + \beta_0$
$\gamma_4 \leftarrow$ T3(α_2, β_2)
$t_{\frac{1}{2}} \leftarrow$ T3$(\alpha_2 + 2\alpha_1 + 4\alpha_0, \beta_2 + 2\beta_1 + 4\beta_0)$
$t_1 \leftarrow$ T3$(\alpha_2 + \alpha_1 + \alpha_0, \beta_2 + \beta_1 + \beta_0)$
$t_2 \leftarrow$ T3$(4\alpha_2 + 2\alpha_1 + \alpha_0, 4\beta_2 + 2\beta_1 + \beta_0)$
$\gamma_0 \leftarrow$ T3(α_0, β_0)
Solve a linear system.

$$\begin{aligned} 2\gamma_3 + 4\gamma_2 + 8\gamma_1 &= t_{\frac{1}{2}} - 16\gamma_0 - \gamma_4 \\ \gamma_3 + \gamma_2 + \gamma_1 &= t_1 - \gamma_0 - \gamma_4 \\ 8\gamma_3 + 4\gamma_2 + 2\gamma_1 &= t_2 - \gamma_0 - 16\gamma_4 \end{aligned}$$

return $\gamma = ((((\gamma_4 <<_B 2)|\gamma_2) <<_B 2)|\gamma_0) + (((\gamma_3 <<_B 2)|\gamma_1) <<_B 1)$

Karatsuba Ka(α, β)

Input: $\alpha, \beta \in \mathbb{Z}$, s.t. $\ell_{\mathcal{B}}(\alpha) = \ell_{\mathcal{B}}(\beta)$.
Output: $\gamma = \alpha\beta$.
Procedure:

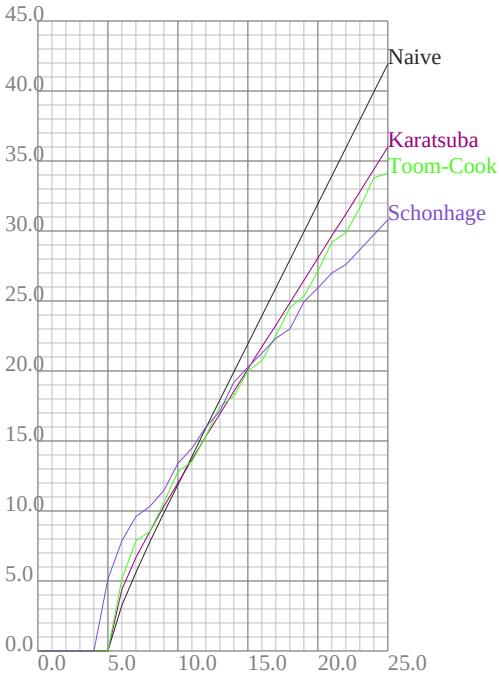
$\ell \leftarrow \ell_2(\alpha)$
if ($\ell \leq 2^{\nu_0}$) return $\alpha * \beta$
$\ell' \leftarrow \left\lceil \frac{\ell}{2^{\nu_0}} \right\rceil 2^{\nu_0}$
$B \leftarrow \frac{\ell'}{2}$
write $\alpha = \alpha_1 B + \alpha_0$ and $\beta = \beta_1 B + \beta_0$
$x \leftarrow$ Ka(α_1, β_1)
$y \leftarrow$ Ka$(\alpha_0 - \alpha_1, \beta_1 - \beta_0)$
$z \leftarrow$ Ka(α_0, β_0)
return $\gamma = ((x <<_B 2)|z) + ((x + y + z) <<_B 1)$

Schönhage Sch(α, β)

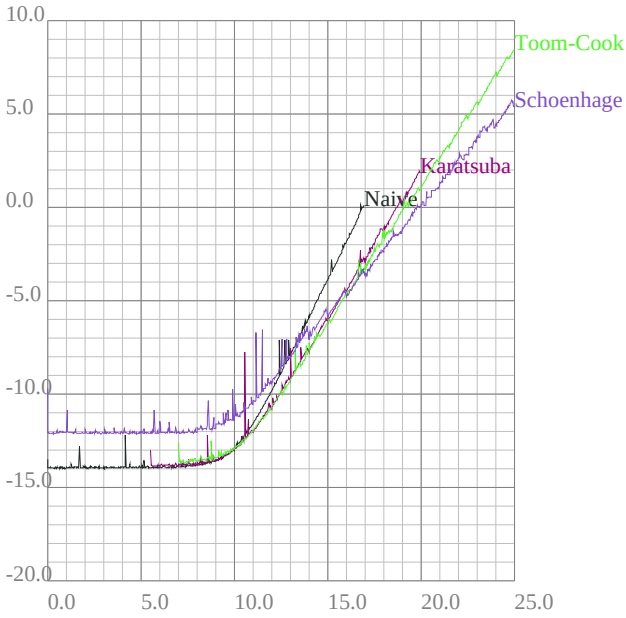
Input: $\alpha, \beta \in \mathbb{Z}/(2^m + 1)\mathbb{Z}$, where $m = 2^{\nu_0 + \nu}$.
Output: $\gamma = \alpha\beta \in \mathbb{Z}/(2^m + 1)\mathbb{Z}$
Procedure:

Choose.
*** If $\nu_0 + \nu < 8$ use an adequate multiplication algorithm.
Set:
*** $\kappa \leftarrow \left\lfloor \frac{\nu_0 + \nu}{2} \right\rfloor$
$M \leftarrow 2^{\kappa}$
$L \leftarrow \frac{m}{M}$
$n \leftarrow \left\lceil \frac{\kappa + 2L + 3}{M} \right\rceil M$
$d \leftarrow \frac{n}{M}$
Represent:
$\alpha = \sum_{j=0}^{M-1} \alpha_j \left(2^L\right)^j$, $\beta = \sum_{j=0}^{M-1} \beta_j \left(2^L\right)^j$.
Define:
$f_\alpha(j) \leftarrow \alpha_j \zeta_{2M}^j \mod (2^n + 1)$, $f_\beta(j) \leftarrow \beta_j \zeta_{2M}^j \mod (2^n + 1)$, where $\zeta_{2M} = 2^d$ and $0 \leq j < M$.
Compute direct FFT:
$FFT_M(f_\alpha, k) \mod (2^n + 1)$ and $FFT_M(f_\beta, k) \mod (2^n + 1)$ for $0 \leq k < M$, where 2^{2d} is a M-th primitive root of unit.
Multiply:
$H(k) \leftarrow$ Sch$(FFT_M(f_\alpha, k), FFT_M(f_\beta, k)) \mod (2^n + 1)$ for $0 \leq k < M$.
Compute inverse FFT:
$h(j) \leftarrow IFFT_M(H, j) \mod (2^n + 1)$ for $0 \leq j < M$, where 2^{2d} is a primitive root of unit.
Compute:
$\gamma_j \leftarrow h(j)(2^d)^{-j} \mod (2^n + 1)$. If $\gamma_j > (j + 1)2^{2L}$, subtract $2^n + 1$ to it (considered as integers).
Result:
$\gamma = \sum_{j=0}^{M-1} \gamma_j \left(2^L\right)^j \mod (2^m + 1)$.

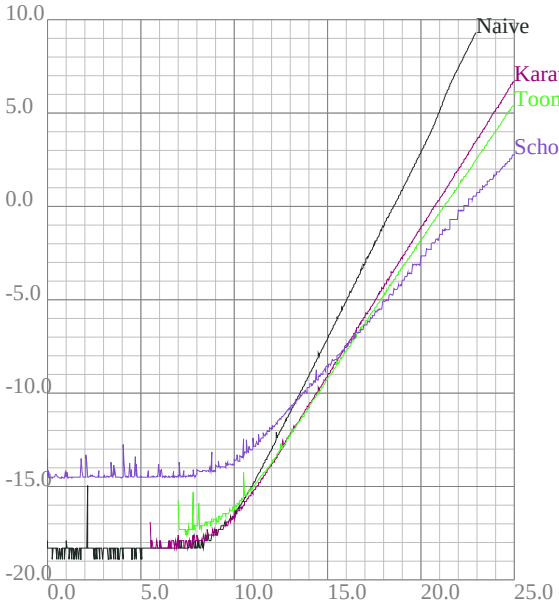
Figure 1: Multiplication algorithms: Naïve, Karatsuba, Toom-Cook and Schönhage



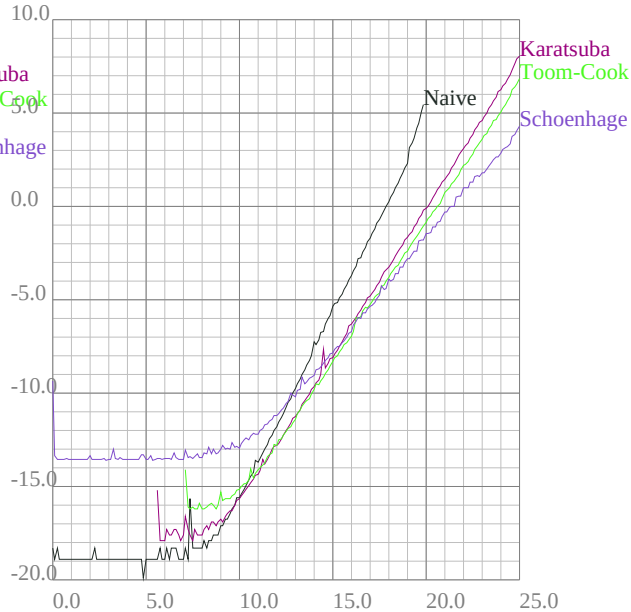
$\log_2(\mathcal{MOB}(alg, 2^{log\_size}))$ with $q = 0.95$



Implementation running on Solaris at 500MHz.



Implementation running on Linux on Intel(R) Pentium(R) 4 at 2.40GHz



Implementation running on Linux on Intel(R) Pentium(R) III at 1.10GHz.

Figure 2: Timing of the multiplication of two numbers of size 2^{log_size}. Graphic of $\log_2(\text{timing}(alg, 2^{log\_size}))$. The multiplication algorithms where used on different processors.