# Fast Truncated Multiplication for Cryptographic Applications

Laszlo Hars

Seagate Research, 1251 Waterfront Place
Pittsburgh, PA 15222, USA
Laszlo@Hars.US

**Abstract**. The *Truncated Multiplication* computes a *truncated product,* a contiguous subsequence of the digits of the product of 2 integers. A few truncated polynomial multiplication algorithms are presented and adapted to integers. They are based on the most often used $n$-digit full multiplication algorithms of time complexity $O(n^\alpha)$, with $1< \alpha \le 2$, but a constant times faster. For example, the least significant half products with Karatsuba multiplication need only 80% of the full multiplication time. The faster the multiplication, the less relative time saving can be achieved.

**Keywords***: Computer Arithmetic, Short product, Truncated product, Cryptography, RSA cryptosystem, Modular multiplication, Montgomery multiplication, Karatsuba multiplication, Barrett multiplication, Optimization*

## 1 Notations

- Long integers are denoted by $A=\{a_{n-1}\dots a_1,\ a_0\}=a_{n-1\dots0}=\sum d^i a_i$ in a $d$-ary number system, where $a_i$, $0\le a_i\le d-1$ are *digits* (usually 16 or 32 bits: $d=2^{16}=65{,}536$; $d=2^{32}=4{,}294{,}967{,}296$)
- $|A|$ or $|A|_d$ denotes the number of digits, the length of a $d$-ary number. $|\{a_{n-1}\dots a_1a_0\}|=n$
- $A\,||\,B$ the number of the concatenated digit-sequence $\{a_{n-1}...a_0,b_{m-1}...b_0\}$; $|A|=n$, $|B|=m$.
- $\lg n = \log_2 n = \log n / \log 2$
- LS stands for **L**east **S**ignificant, the low order bit/s or digit/s of a number
- MS stands for **M**ost **S**ignificant, the high order bit/s or digit/s of a number
- (*Grammar*) *School multiplication, division*: the digit-by-digit multiplication and division algorithms, as taught in elementary schools
- $A\ltimes B$, $A\rtimes B$ denote the MS or LS half of the digit-sequence of $A\times B$ (or $A\cdot B$), respectively
- $A\otimes B$ denotes the middle third of the digit-sequence of $A\times B$
- $M_\alpha(n)$ the time complexity of the Toom-Cook type full multiplication, $O(n^\alpha)$, with $1<\alpha\le 2$
- $\gamma_\alpha$ = the speedup factor of the half multiplication, relative to $M_\alpha(n)$
- $\delta_\alpha$ = the speedup factor of the middle-third product, relative to $M_\alpha(n)$

## 2 Introduction

Many cryptographic algorithms are based on modular arithmetic operations. The most time critical one is multiplication. For example, exponentiation, the fundamental building block of RSA-, ElGamal- or Elliptic Curve - cryptosystems or the Diffie-Hellman key exchange protocol [17], is performed by a chain of modular multiplications. For modular reduction division is used, which can be performed via multiplication with the reciprocal of the divisor, so fast reciprocal calculation is also important. Modular multiplications can be performed with reciprocals and regular multiplications, and in some of these calculations truncated products are sufficient.

We present new speedup techniques for these and other basic arithmetic operations.

For operand sizes of cryptographic applications school multiplication is used the most often, requiring simple control structure. Speed improvements can be achieved with Karatsuba's method and the Toom-Cook 3- or 4-way multiplication, but the asymptotically faster algorithms are slower for these operand lengths [9], [14]. In this paper we consider digit-serial multiplication algorithms of time complexity $O(n^{\alpha})$, $1 < \alpha \leq 2$, similar to microprocessor software, that is, no massive-parallel- or discrete Fourier transform based multiplications, which require different optimization methods [3].

## 3 Truncated Products

The *Truncated Multiplication* computes a *Truncated Product*, a contiguous subsequence of the digits of the product of 2 integers. If they consist of the LS or MS half of the digits, they are sometimes called short products or half products. These are the most often used truncated products together with the computation of the middle third of the product-digits, also called middle product.

No exact speedup factor is known for truncated multiplications, which are based on full multiplications faster than the school multiplication. For half products computed by Fourier or Nussbaumer transform based multiplications no constant time speedup is known.

One way to calculate truncated products is implied by covering the corresponding area in the multiplication square (Figure 1) with polygons (like smaller squares or triangles), which correspond to partial products of known complexity. Figure 3 shows a covering of a triangle, corresponding to the MS half product, proposed by Mulders [19] for polynomials. Covered areas, which don't belong to the truncated product to be computed, get ignored when the final digit-sequence is generated. They don't cause extra costs beyond the work needed to calculate them (except handling carries), but the resulting shapes might correspond to products, which can be faster calculated. (A full square, for example, has many dependencies among the digit-products contained, so full multiplications could be faster than truncated multiplications of similar size, corresponding to narrow and long regions, without that many dependencies.)

The covering shapes may extend out from the full product square, like in Figure 10. The products in the excess area are calculated with 0-padding the multiplicands, not affecting the result. If the covering polygons overlap, the involved area has to be processed again and the corresponding digits subtracted from the result digit-sequence one fewer times than they were covered.

## 4 Time complexity

Multiplication is more expensive (slower and/or more hardware consuming) even on single digits, than addition or store/load operations. Many computing platforms perform additive- and data movement operations parallel to multiplications, so they don't take extra time. In order to obtain general results and to avoid complications from architecture dependent constants we measure the time complexity of the algorithms with the ***number of digit-multiplications*** performed.

For the commonly used multiplication algorithms, even for moderate operand lengths the number of digit-multiplications is well approximated by $n^{\alpha}$, where $\alpha$ is listed in the table below.

| School | Karatsuba | Toom-Cook-3 | Toom-Cook-4 |
|---|---|---|---|
| 2 | $\log 3 / \log 2 = 1.5850$ | $\log 5 / \log 3 = 1.4650$ | $\log 7 / \log 4 = 1.4037$ |

On most computational platforms the actual running time is $\approx c \cdot n^{\alpha}$ multiplication time, with a small constant $c = 1 \ldots 3$, dependent on the architecture, because these multiplication algorithms don't perform much more other operations than multiplications.

On shorter operands asymptotically slower algorithms could be faster, until at longer operands architecture-dependent minor terms become negligible. (We cannot compare different multiplication algorithms, running in different computing environments, without knowing all these factors.) For example, when multiplying linear combinations of partial results or operands, a significant number of non-multiplicative digit operations are executed, that might not be possible to perform in parallel to the digit-multiplications. They affect some minor terms in the complexity expressions and could affect the speed relations for shorter operands. To avoid this problem, when we look for speedups for certain multiplication algorithms, when not all of their product digits are needed, we only consider algorithms performing ***no more auxiliary digit operations than what the corresponding full multiplication performs***. When each member of a family of algorithms under this assumption uses internally one kind of black-box multiplication method (School, Karatsuba, Toom-Cook-*k*), the speed ratios among them are about the same as that of the black-box multiplications. Consequently, if on a given computational platform and operand length one particular multiplication algorithm is found to be the best, say it is Karatsuba, then, within a small margin, the fastest algorithm discussed in this paper is also the one, which uses Karatsuba multiplication.

### 4.1 Complexity Paradox

We don't consider algorithms with excessive operand mixing, like many linear combinations of partial results. They might speedup calculations asymptotically, but at certain operand lengths the speed relations among the algorithms could be changed, due to the hidden minor terms in the time complexities. For example, consider the complexity of truncated multiplications using Karatsuba's method. Let us partition the multiplicands into $k > 2$ digit-blocks and perform one $k$-way Toom-Cook multiplication step. The digit-blocks are multiplied with Karatsuba's algorithm

of time complexity $n^{\lg 3}$. The product (an extreme truncated product) takes asymptotically $(2k-1)(n/k)^{\lg 3} < n^{\lg 3}$ digit-products to calculate. Since the Toom-Cook multiplication algorithm performs more non-multiplicative operations than Karatsuba (long additions and multiplications / divisions with small constants), some minor terms become significant (left out from our asymptotical complexity expression), and so we might paradoxically conclude that Karatsuba's multiplication is faster than itself.

## 5 School Multiplication

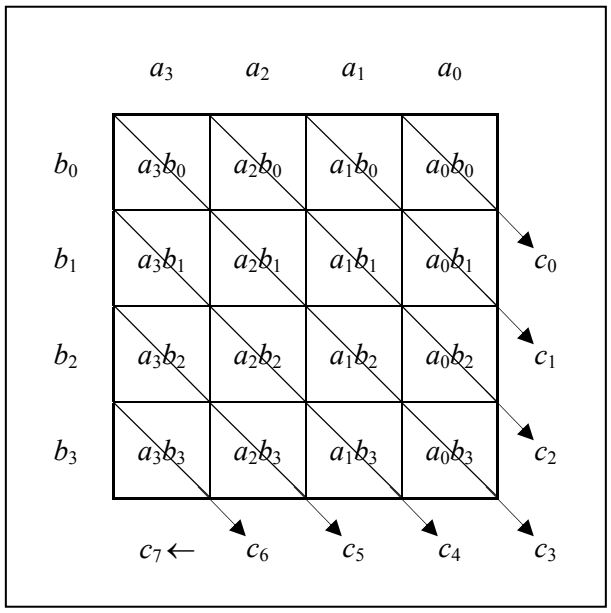


**Figure 1**. School multiplication

In multiplications C = A·B, the product digits are calculated as $c_k = \Sigma_{i+j=k}\, a_i b_j$ for indices when $a_i$ and $b_j$ exist. It is convenient to write the digits of the multiplicands on the top and left side of a rectangle, and the digit-products inside, where the corresponding rows and columns meet. The digits of the product are then calculated by summing up the digit-products inside the rectangle along the diagonals, starting with the single entry diagonal at the top right corner, moving left and downward. The MS digit is the result only of the last carry: Figure 1. (We can think this square as a "straightened up" product parallelogram, taught in the school for arranging the partial products.)

## 6 Carry

The digit-products can be 1 or 2 digits long. When calculating the digits of the (truncated) product these digit-products are added, so there is usually a carry, propagating to more significant digits. The carry is the largest when all digits of both multiplicands are $d-1$. The diagonal $i$, $i = 0\ldots n-1$, in the triangle I in Figure 2 contributes to the total $(i+1)(d-1)^2 d^i$. Summing them up from 0 to $k$ (e.g. with the help of the differential of the generator function $G(x) = (d-1)^2 \sum_i x^{i+1}$, a geometric series) gives the total of the first $k+1$ diagonals (see also [16]):

$$s_k = kd^{k+2} + (d-k-2)d^{k+1} + 1.$$

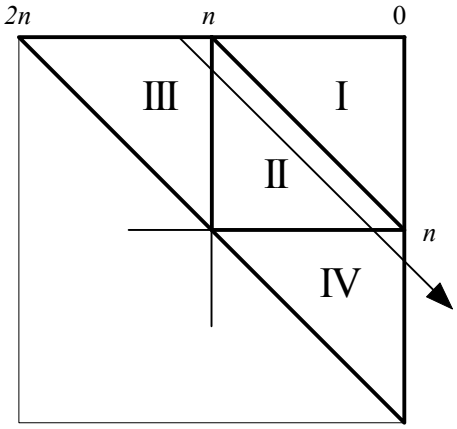


**Figure 2**. Carry in MS half

If $k = n\ldots 2n-2$, (triangle II in Figure 2) we embed the multiplication square in a double size one, and sum the diagonals there. The products belonging to parts of the diagonals, which lie in the triangles III and IV are then subtracted:

$$\begin{aligned} \boldsymbol{s_k - 2d^n s_{k-n}} &= (2n-k-1)d^{k+2} + (k-2n+2)d^{k+1} - 2d^n + 1 \\ &= (2n-k-2)d^{k+2} + (d-2n+k+1)d^{k+1} + (d^{k+1} - 2d^n) + 1 \end{aligned}$$

**Lemma C.** The carry is maximal at the main diagonal: $s_{n-1}$.

**Proof.** The above equations show, that the carry $s_k/d^k$ in the upper right triangle is increasing, in the lower right triangle, decreasing with $k$. The larger of the two middle ones is the maximum. Compare $d \cdot s_{n-1} = (n-1)d^{n+2} + (d-n-1)d^{n+1} + d$ and $s_n - 2d^n s_0 = (n-2)d^{n+2} + (d-n+1)d^{n+1} + (d^{n+1} - 2d^n) + 1$. The coefficient of the dominant $d^{n+2}$ term is larger at the former one, proving the proposition for $d > 2$. For $d = 2$ direct substitution proves it. []

### 6.1 Guard digits

The most often used truncated products of $n$-digit multiplicands contain either the MS half of the product digits $(c_{2n-1}, c_{2n-2}, \ldots, c_n)$, or the LS half $(c_{n-1}, \ldots, c_1, c_0)$. There is no problem with the carry at the LS half, but the MS half product is affected by the carry from the LS half. This is the same problem as proper rounding of floating point multiplications [16].

Other truncated products without the LS digits have also problems with the carry. From Lemma C it follows that the largest carry is caused by the LS half products, and it can be as large as $s_{n-1} = (n-1)d^{n+1} + (d-n-1)d^n + 1$. That is, the rightmost 2 digits of the MS half product can be affected. Therefore, we calculate 2 more digits than requested, called "guard" digits, with the carry they generate. There is, however, still a chance of carry propagation from further to the right. The LS digits all, up to $c_{n-3}$ can contribute $s_{n-3} = (n-3)d^{n-1} + (d-n+1)d^{n-2} + 1$. If the first guard digit $c_{n-1}$ was $d-n+3$ or larger, the left out LS product could cause a carry propagating to the digit $c_n$ and even further.

In the middle of cryptographic calculations partial results look uniformly random. In this sense, having ignored the LS $n-2$ product digits, the chance of a possible carry to $c_n$ is $(n-3)/d$. It is small at usual settings (0.1% at $d = 2^{16}$, $n = 64$; and $6.7\times10^{-9}$ at $d = 2^{32}$, $n = 32$ − corresponding to RSA-1024). We can see the danger of ineffective guard digits. If they happen to be too large we calculate a third guard digit, too. A carry could only be propagating, if $c_{n-1} = d-1$ and $c_{n-2} \geq d-n+4$. The chance of this is very small: $1.4\times10^{-8}$ at $d = 2^{16}$, $n = 64$; and $1.5\times10^{-18}$ at $d = 2^{32}$, $n = 32$. In Newton iterations or in the Barrett multiplication an occasional error of 1 does not matter, but in other situations we ***cannot allow wrong results*** even with this low probability.

Keeping calculating more guard digits when they turn out to be large, the expected amount of work to get the ***exact carry*** for the MS half product is barely larger than $2n$, but the worst case complexity is $O(n^2)$. Stopping at the 2nd or 3rd guard digit and calculate the full product when there is a possible carry propagation, gives slightly more expected work, but in the worst case only one extra multiplication is performed. It is better at sub-quadratic multiplication algorithms.

Let us denote by $T_1$ and $T_0$ the average time complexity of truncated multiplications, providing the product digits $\{c_k, c_{k+1} \ldots c_m\}$, $k \geq m$, with or without proper rounding, respectively. Since the main diagonal is the longest, the reasoning above covers the worst case carry, proving

**Lemma G.** $T_1 < T_0 + c \cdot m$, with $c \approx 2$. []

Consequently, the LS and MS half products are of equal complexity (within an additive linear term). In the sequel we don't distinguish between truncated products with or without proper rounding, unless the linear term of the time complexity is in question.

## 7 Half products

With school multiplication half products can be calculated in half of the time of the full multiplication, simply by not calculating the digit-products, which are not needed.

Let $M_\alpha(n) = n^\alpha$ denote the time complexity of the full $n$-digit multiplication, $1 < \alpha \le 2$; and $H_\alpha(n)$ denote the time needed for a half product. The construction of Figure 3 leads to the inequality:

$$H_\alpha(n) \le M_\alpha(n-k) + 2H_\alpha(k).$$

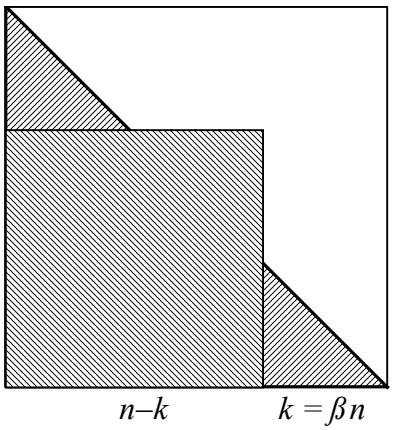


**Figure 3**. Half product computation

**Note.** If we chose $k = n/2$, such that there is no excess area (the small square does not cross the diagonal), we get $H_\alpha(n) \le M_\alpha(n/2) + 2H_\alpha(n/2)$. Denoting the complexity of the resulting algorithm by $S'_\alpha(n)$, we get the recursive equation $S'_\alpha(n) = M_\alpha(n/2) + 2\,S'_\alpha(n/2)$. Unfortunately, the solution shows $S'_\alpha(n)$ larger than the full multiplication time $M_\alpha(n)$ if $\alpha < \lg 3$, and when $\alpha = \lg 3$ (Karatsuba), any initial speed advantage (for small $n$ values) diminishes very fast with increasing $n$, giving $S'_{\lg 3}(n) \approx M_{\lg 3}(n)$. See [16].

Let us denote the time complexity of the half multiplication algorithm by $S_\alpha(n)$, derived from the covering shown in Figure 3, with the underlying multiplication complexity $M_\alpha(n) = n^\alpha$.

Looking for a constant speedup ratio, let's chose appropriate $\beta$ and $\gamma$ factors: $S_\alpha(n) = \gamma n^\alpha$, and $k = \beta n$. The recurrence relation $S_\alpha(n) = M_\alpha(n-k) + 2S_\alpha(k)$ becomes

$$\gamma n^\alpha = n^\alpha(1-\beta)^\alpha + 2\gamma n^\alpha \beta^\alpha \qquad (1)$$

**Theorem H.** The time complexity of the half multiplication, based on multiplications of time complexity $M_\alpha(n) = n^\alpha$, $1 < \alpha \le 2$, is at most $\gamma_\alpha M_\alpha(n)$, with

$$\gamma = (1-\beta)^\alpha/(1-2\beta^\alpha) \text{ and } \beta = 2^{-1/(\alpha-1)}.$$

**Proof.** The value of $\beta$, which minimizes $\gamma$ is found by differentiation of (1). []

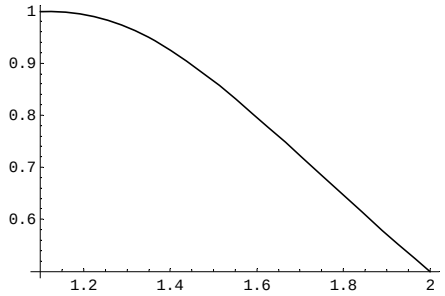


**Figure 4**. Half product speedup $\gamma$ vs. $\alpha$

For shorter operands numerical calculations or simulations find the exact speedup factors [28]. They were found close to the asymptotic values graphed in Figure 4. The numerical values of these asymptotical speedup factors and the corresponding splitting ratios are tabulated below.

| | Exponent $\alpha$ | Speed $\gamma_\alpha$ | Split at $\beta_\alpha$ |
|---|---|---|---|
| School | 2 | 0.5 | 0.5 |
| Karatsuba | 1.585 | 0.8078 | 0.3058 |
| Toom-Cook-3 | 1.465 | 0.8881 | 0.2252 |
| Toom-Cook-4 | 1.404 | 0.9232 | 0.1796 |

There are many more ways to cover a product triangle. For example:

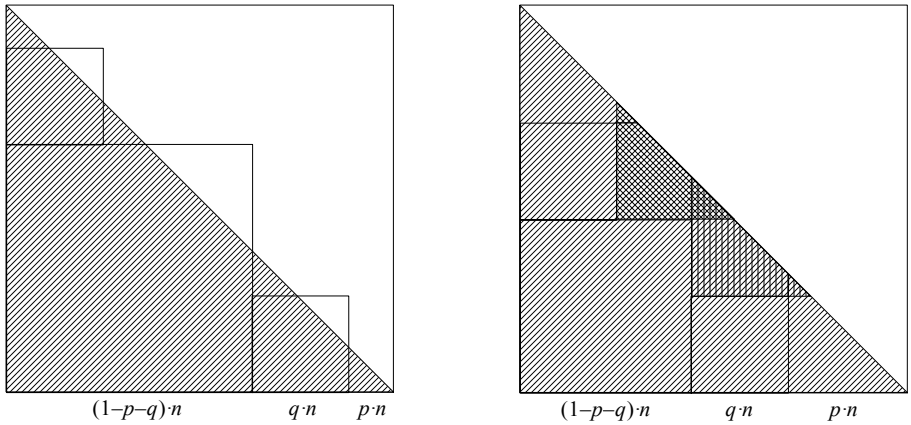


**Figure 5**. Alternative coverings for half products

The left hand side configuration on Figure 5 is optimal if $q+p = \beta_\alpha$, when it becomes the same as Figure 3, recursively applied in the small triangles. The right hand side configuration is actually worse, because of the overlapping triangles.

## 8 LS and MS products

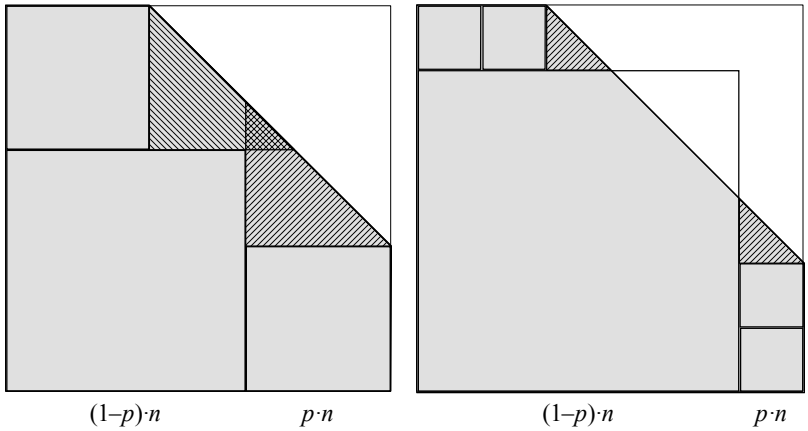


**Figure 6**. Cut corner product constructions

In the geometric constructions of the half product calculations in the previous section there were large squares, with not needed product digits corresponding to their upper right corner. These represent MS truncated products. A natural question is: How much must we cut off, such that the truncated product corresponding to the remaining part is faster to compute than the whole product?

The constructions of Figure 6 give lower bounds on the size of the largest MS truncated products faster computed than the full products. With varying number of the small squares on the sides numerical calculations give the following optimum $p$ values:

| | | Karatsuba | Toom-Cook-3 | Toom-Cook-4 |
|---|---|---|---|---|
| □ | Fastest product | 0.9402 | 0.9744 | 0.9860 |
| | Max $p$ | 0.2174 | **0.1225** | **0.0791** |
| □<br>□ | Fastest product | 0.9706 | 0.9895 | 0.9950 |
| | Max $2p$ | **0.2176** | 0.1020 | 0.0575 |
| □<br>□<br>□ | Fastest product | 0.9822 | 0.9944 | 0.9976 |
| | Max $3p$ | 0.1982 | 0.0817 | 0.0419 |

The "Fastest product" entries indicate the maximum speedup for MS truncated products over full products at the best $p$ value with the corresponding number of little side-squares (Figure 6). With this kind of constructions the gain is not large, at most 6% (Karatsuba with 1 small square at the side and a triangle stacked up).

**Proposition S.** The MS and LS truncated multiplication of two $n$-digit numbers is faster than the full multiplication if $k$ of the $2n$ product-digits are computed, $k < 61\%$ of $2n$ with Karatsuba multiplication: $(1+kp)/2 = 0.6088$, $k < 56\%$ with Toom-Cook-3 and $k < 54\%$ with Toom-Cook-4. []

These MS products, which can be faster calculated than the full product, are not long enough to improve half multiplications, needing speed improvement at $½/(1-\beta)$.

## 9 Middle-third products

Some crypto algorithms could use products containing the middle $n$ digits $a \otimes b$ of the $3n$ digits of the product $a \times b$, when one of the multiplicands is twice longer ($2n$-digit) than the other. The corresponding digit-products are contained in the shaded parallelogram in Figure 7. The simplest way to achieve faster multiplications is to split it in half, and combine the 2 half products corresponding to the left and right triangle, with time complexity $2\gamma_\alpha \mathrm{M}_\alpha(n)$.

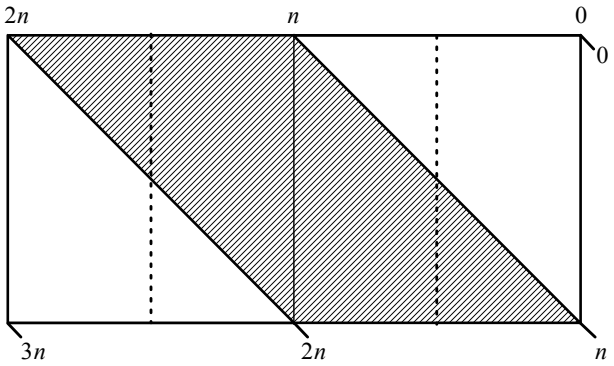


**Figure 7**. Middle-third product

Alternatively, an $n \times n$ square covers the center of the parallelogram (dotted lines in Figure 7). When we process separately the two small triangles left out: the corresponding complexity is better (for the Karatsuba case it is 1.5385):

$$2\gamma_\alpha \mathrm{M}_\alpha(n) > \mathrm{M}_\alpha(n) + 2\gamma_\alpha \mathrm{M}_\alpha(n/2) = (1+2^{1-\alpha}\gamma_\alpha)\mathrm{M}_\alpha(n).$$

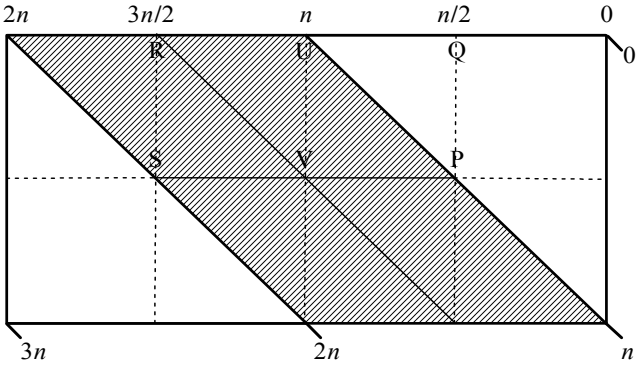


**Figure 8**. Karatsuba Middle-third product

In [11] a clever speedup was presented for the Karatsuba case. It dissects the operands and calculates products of sums, but uses only as many extra addition/load operations as the multiplication, so it does not change speed relations. The resulting complexity is the same as that of the Karatsuba multiplication. Unfortunately, this idea does not improve the asymptotically faster multiplications, so only the second entry is affected in the table below for the middle product speedup factors, but the Karatsuba case is the most important one in practice.

The idea is to cut the middle product parallelogram into 4 congruent pieces, as shown in Figure 8. Each smaller parallelogram represents a middle third product, as seen, e.g. PURV in the rectangle PQRS. With the notation $A_{ij} = \{a_{jn/2-1} \ldots a_{in/2}\}$ the 4 middle products are $MP(A_{13},B_0)$, $MP(A_{24},B_0)$, $MP(A_{13},B_1)$, $MP(A_{02},B_1)$. Calculate α, β and γ:

$$\alpha \leftarrow MP(A_{02}+A_{13},B_1)$$
$$\beta \leftarrow MP(A_{13}, B_0-B_1)$$
$$\gamma \leftarrow MP(A_{13}+A_{24},B_0)$$

The desired result is $(\alpha+\beta) \parallel (\gamma-\beta)$, with carry propagation. That is, the middle third product is calculated with 3 half size middle third products and 5 additions, giving the same recursion (for power of 2 operand lengths) as at the Karatsuba multiplication. (Other lengths are slightly slower.) Let $\delta_\alpha$ denote the speedup factor relative to $M_\alpha(n)$. It is summarized in the following table. (The Toom-Cook entries represent new results.)

| School | Karatsuba | Toom-Cook-3 | Toom-Cook-4 |
|---|---|---|---|
| 1 | 1 | ***1.6434*** | ***1.6979*** |

## 10 Third-quarter products

The third quarter of the product digits ($c_{3n/2-1}\ldots c_n$) contains $^3/_8$ of the digit-products. Accordingly, the school multiplier takes $^3/_8\,n^2$ time. It is of the worst shape discussed here. Narrow, and long, so no much room is left to find dependencies among digit-products, which would allow faster multiplication algorithms.

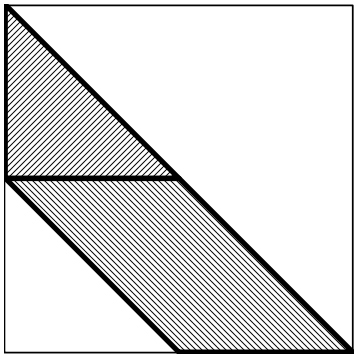

**Figure 9.** Third-quarter product computation

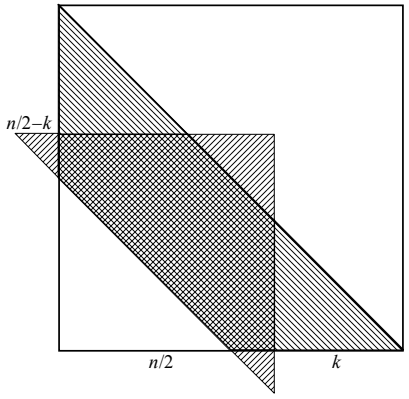


**Figure 10**. Alternative computation for third-quarter products

The shaded trapezoid can be cut into a parallelogram and a triangle, as shown in Figure 9, and the corresponding time complexity is

$$(\delta_\alpha+\gamma_\alpha)\mathrm{M}_\alpha(n/2) = (\delta_\alpha+\gamma_\alpha)2^{-\alpha}\mathrm{M}_\alpha(n).$$

It gives the following coefficients: 0.375, 0.6026, 0.9170 and 0.9907. The school and the Karatsuba multiplication case are faster than calculating the full triangle, but at the Toom-Cook multiplications we are better off taking the containing whole half product. (One could try to cover the trapezoid with triangles, which overhang the multiplication square, as in Figure 10. Simple calculations show, that it does not help. The complexity is minimal at $k = n/2$, that is, where the cover is exact.)

## 11 Squaring

All the known multiplication algorithms with time complexity $\mathrm{O}(n^\alpha)$ speed up almost twofold for squaring, which is the case for small operands and the recursion these algorithms follow keeps this ratio. Additions and processing overhead in the algorithms reduce the realizable speedup ratio to 0.55…0.65. See [28]. (A speedup below 0.5 cannot be achieved, since it would yield faster general multiplications with the identity $4ab = (a+b)^2-(a-b)^2$ and the corresponding algorithm gives 0.5 multiplication time for squaring.)

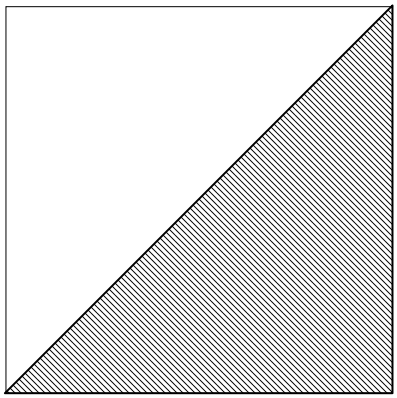

**Figure 11**. Squaring as partial product

Squaring is not a truncated product, being represented by a triangle with its hypotenuse perpendicular to the main diagonal of the multiplication square. At general products the upper-left and lower-right triangles contain different digit-products, but the digit-sequences corresponding to these triangles could be calculated faster than full products, broadening the choice of building blocks for truncated products. In this paper we don't use this tool.

In the **school multiplication** the terms $a_ib_j$ and $a_jb_i$ become the same, $a_ia_j$. All products are repeated, except the squares $a_ia_i$, which reduces the number of the necessary digit-multiplications (and so the time complexity) from $n^2$ to $n(n+1)/2$, almost twofold, as seen on Figure 11.

The general **Karatsuba** multiplication cuts the operands into halves, and performs 5 additions and 3 multiplications on them to get the product. It has a recurrence relation $M(2n) = 3M(n) + 5cn$ (see [9], [14])[1], where the term $5cn$ comes from 5 additions of $n$-digit numbers. Squaring has a similar recurrence equation[2] $S(2n) = 3S(n) + 4cn$.

The solution of the recurrence equation $f(2n) = 3f(n) + k \cdot n$, with $f(m) = C$ is

$$f(n) = n^{\lg 3}\frac{C + 2k \cdot m}{m^{\lg 3}} - 2k \cdot n$$

School multiplication is faster below a certain operand length $m$, like 8 digits, where the recursion is stopped. There the speed ratio of squaring over multiplication $C_S/C_M$ is close to ½. The ratio of the number of the respective auxiliary operations $k_S/k_M$ everywhere is close to ½, too (at squaring only one operand is fetched and stacked, and 4 additions performed, not 5). This leads to $S(n)/M(n) \approx ½$. (Practical values are $S(32)/M(32) \approx 0.6$ and $S(64)/M(64) \approx 0.62$, with huge $n$ values the ratio remaining below 0.65. [28])

---

[1] $(x_1d^k+x_2)\cdot(y_1d^k+y_2) = x_1\cdot y_1d^{2k} + x_2\cdot y_2 + [(x_1+x_2)\cdot(y_1+y_2)-x_1\cdot y_1-x_2\cdot y_2]d^k$

[2] $(x_1d^k+x_2)^2 = x_1^2d^{2k} + x_2^2 + [x_1^2+x_2^2-(x_1-x_2)^2]d^k$, the first + is concatenation.

**Toom-Cook** multiplications ([9], [14]) are similar. They, too, cut the operands into smaller pieces and build the product from them. At the bottom of the recursion asymptotically slower multiplication methods get faster, so we turn to them. Their relative speedup for squaring is approximately maintained to large operands.

## 12 Summary

The presented fast truncated multiplication algorithms were developed to improve performance of several cryptographic algorithms (see [29]). The most important results in this paper:

- Carry estimation and exact rounding algorithms for truncated products
- Proof of equivalent complexity of LS and MS half products, within a linear term
- Fast truncated long integer multiplication algorithms (half products, middle third products, third quarter products) for Toom-Cook type multiplications.
- Finding the lengths of MS and LS truncated products, which can be faster computed than the full product
- Close to double speed squaring algorithms

## References


[1] J.-C. Bajard, L.-S. Didier, and P. Kornerup. *An RNS Montgomery multiplication algorithm*. In 13th IEEE Symposium on Computer Arithmetic (ARITH 13), pp. 234–239, IEEE Press, 1997.

[2] P. D. Barrett, *Implementing the Rivest Shamir Adleman public key encryption algorithm on standard digital signal processor*, In Advances in Cryptology-Crypto'86, Springer-Verlag, 1987, pp.311-323.

[3] D. J. Bernstein, *Fast Multiplication and its Applications*, http://cr.yp.to/papers.html#multapps

[4] Bosselaers, R. Govaerts and J. Vandewalle, *Comparison of three modular reduction functions*, In Advances in Cryptology-Crypto'93, LNCS 773, Springer-Verlag, 1994, pp.175-186.

[5] E.F. Brickell. *A Survey of Hardware Implementations of RSA*. Proceedings of Crypto'89, Lecture Notes in Computer Science, Springer-Verlag, 1990.

[6] C. Burnikel, J. Ziegler, *Fast recursive division*, MPI research report I-98-1-022

[7] B. Chevallier-Mames, M. Joye, and P. Paillier. *Faster Double-Size Modular Multiplication from Euclidean Multipliers*, Cryptographic Hardware and Embedded Systems – CHES 2003, vol. 2779 of Lecture Notes in Computer Science,, pp. 214−227, Springer-Verlag, 2003.

[8] J.-F. Dhem, J.-J. Quisquater, *Recent results on modular multiplications for smart cards*, Proceedings of Cardis 1998, Volume 1820 of Lecture Notes in Computer Security, pp 350-366, Springer-Verlag, 2000

[9] GNU Multiple Precision Arithmetic Library manual http://www.swox.com/gmp/gmp-man-4.1.2.pdf

[10] W. Fischer and J.-P. Seifert. *Increasing the bitlength of crypto-coprocessors via smart hardware/software co-design*. Cryptographic Hardware and Embedded Systems – CHES 2002, vol. 2523 of Lecture Notes in Computer Science, pp. 71–81, Springer-Verlag, 2002.

[11] G. Hanrot, M. Quercia, P. Zimmermann, *The Middle Product Algorithm, I*. Rapport de recherche No. 4664, Dec 2, 2002 http://www.inria.fr/rrrt/rr-4664.html

[12] K. Hensel, *Theorie der algebraische Zahlen*. Leipzig, 1908

[13] J. Jedwab and C. J. Mitchell. *Minimum weight modified signed-digit representations and fast exponentiation*. Electronics Letters, 25(17):1171-1172, 17. August 1989.

[14] A. H. Karp, P. Markstein. *High precision division and square root*. ACM Transaction on Mathematical Software, Vol. 23, n. 4, 1997, pp 561-589.

[15] D. E. Knuth. *The Art of Computer Programming*. Volume 2. Seminumerical Algorithms. Addison-Wesley, 1981. Algorithm 4.3.3R

[16] W. Krandick, J.R. Johnson, *Efficient Multiprecision Floating Point Multiplication with Exact Rounding*, Tech. Rep. 93-76, RISC-Linz, Johannes Kepler University, Linz, Austria, 1993.

[17] A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1996.

[18] P.L. Montgomery, "*Modular Multiplication without Trial Division*," Mathematics of Computation, Vol. 44, No. 170, 1985, pp. 519-521.

[19] T. Mulders. *On computing short products*. Tech Report No. 276, Dept of CS, ETH Zurich, Nov. 1997 http://www.inf.ethz.ch/research/publications/data/tech-reports/2xx/276.pdf

[20] P. Paillier. *Low-cost double-size modular exponentiation or how to stretch your cryptoprocessor.* Public-Key Cryptography, vol. 1560 of Lecture Notes in Computer Science, pp. 223–234, Springer-Verlag, 1999.

[21] K. C. Posh and R. Posh. *Modulo reduction in Residue Number Systems*. IEEE Transactions on Parallel and Distributed Systems, vol. 6, no. 5, pp. 449–454, 1995.

[22] J.-J. Quisquater, *Fast modular exponentiation without division*. Rump session of Eurocrypt'90, Arhus, Denmark, 1990.

[23] R. L. Rivest; A. Shamir, and L. Adleman. 1978. *A method for obtaining digital signatures and public key cryptosystems*. Communications of the ACM 21(2):120--126

[24] J. Schwemmlein, K.C. Posh and R. Posh. *RNS modulo reduction upon a restricted base value set and its applicability to RSA cryptography*. Computer & Security, vol. 17, no. 7, pp. 637–650, 1998.

[25] SNIA OSD Technical Work Group http://www.snia.org/tech_activities/workgroups/osd/

[26] C.D. Walter, "*Faster modular multiplication by operand scaling*," Advances in Cryptology, Proc. Crypto'91, LNCS 576, J. Feigenbaum, Ed., Springer-Verlag, 1992, pp. 313—323

[27] L. Hars, "*Long Modular Multiplication for Cryptographic Applications*", CHES 2004, (Corrected original misprint in LNCS 3156, pp 44-61. Springer, 2004.) http://eprint.iacr.org/2004/198/

[28] L. Hars, "*Finding the Fastest Multiplication for Cryptographic Operand Lengths: Analytic and Experimental Comparisons*" manuscript.

[29] L. Hars, "*Applications of Fast Truncated Multiplication in Cryptography*", presented at CHES 2005, Edinburgh (not printed in the proceedings because of page number limitations). EURASIP Journal on Embedded Systems, vol. 2007, Article ID 61721, 9 pages, 2007