# THE RSA CRYPTOGRAPHY PROCESSOR

Holger Sedlak

Institut für Theoretische Informatik
Technische Universität Braunschweig
D-3300 Braunschweig
Federal Republic of Germany
(531) 391-2384

## 1. Introduction

In commercial applications, a minimum ciphering rate of 64 $Kbit/sec$ is required which will be the transmission rate of public digital networks. In contrast, the RSA method has a very slow ciphering rate particulary when using software implementations of the algorithm. The solution of this problem is a hardware implementation of the RSA algorithm. A cryptography processor, however, consisting of standard chips like bit slice processors again does not achieve the speed necessary. Moreover, in a multi-chip processor, the security of the key management system cannot be guaranteed. Therefore, a single-chip implementation of the RSA algorithm seems to be the only solution. Such a solution is presented as an RSA *Cryptography Processor* (CP).

## 2. The Implementation of the RSA Method

The bottleneck of an implementation of the RSA method is the handling of very long numbers. If one wishes to cipher a 200 digit decimal number, then the length of each key is 660 $bit$ (the CP supports numbers up to a length of 780 $bit$). With this number the CP has to execute the function

$$C = M^E \bmod N,$$

where $C$ is the code generated, $M$ the data (message), and $(E, N)$ is the public encoding key. The same function is used for decoding (only another exponent is used), so the fast execution of this function is the heart of the problem. In this form, the function is not implementable, but after some transformations the function is reduced to a sequence of additions and subtractions. This is demonstrated in Fig. 1, at first sight in the well known way.

First, the powering is reduced to a sequence of multiplications (Fig. 1a), and it is executed in the ring of residual classes of N. Next, the multiplication is reduced to a sequence of additions. Also the modulo operation is reduced to a sequence of subtractions as in the classical algorithm of division (Fig. 1b). Last, for reducing the maximum size of the registers of the CP, the multiplication is computed in the ring of residual classes of N.

Up to this point it is the known way. But now the implementation leaves this way with the look-ahead algorithms which decrease the maximum number of additions for the multiplication and subtractions for the modulo operation. It is important to recognize the necessity of simultaneous acceleration of multiplication and modulo operation. If, for

D. Chaum and W.L. Price (Eds.): Advances in Cryptology - EUROCRYPT '87, LNCS 304, pp. 95-105, 1988.

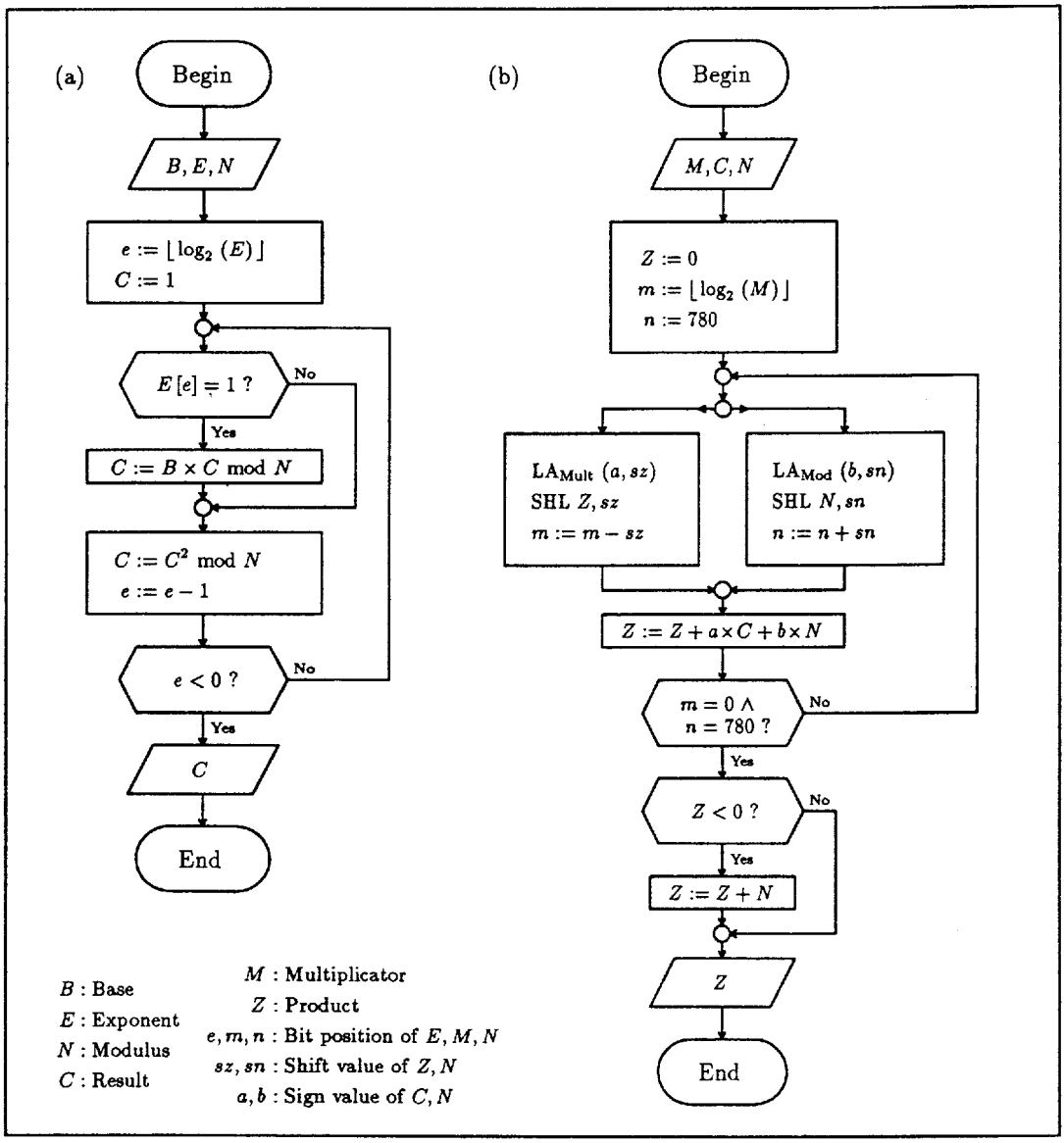


*Figure* 1: Flowchart of exponentiation (a) and multiplication mod $N$ (b). The two $LA$-Functions calculate in parallel with the adding circuits the next look-ahead values for shifting and addition. This main loop of multiplication takes only one processor cycle, e.g. 32 $ns$, except for some quite infrequent cases.

example, only the multiplication was accelerated, this would not result in an improvement of the algorithm because the number of subtractions for the modulo operation would remain unchanged. Since the multiplication and the modulo operation are executed in turn, the number of subtractions becomes the time determining part.

While the look-ahead algorithm of the multiplication is well known, a new look-ahead procedure for the division and hence the modulo operation had to be developed in [8]. The major condition was that the average reduction of the look-ahead of division is approximately the same as the one of the look-ahead of multiplication. In this case, the two look-ahead algorithms would cooperate optimally.

In the serial multiplication with the classical look-ahead, chains of maximal length of 0s and 1s are processed

alternatingly. When multiplying, for example, a binary number with

$$\begin{array}{ll} \mathrm{M} = \ldots 0 & 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad \ldots \\ \text{bit} & \ldots m_{11}\, m_{10}\, m_9\, m_8\, m_7\, m_6\, m_5\, m_4\, m_3\, m_2\, m_1\, m_0 \ldots, \end{array}$$

the algorithm starts in a state *0* processing the 0-chain at bits $m_0$ to $m_3$ as a whole. Then it changes to state *1* processing the 1-chain at bits $m_4$ to $m_6$ and bit $m_7$ analogously. At bit $m_7$ it remains in state *1* processing the 1-chain at bits $m_8$ to $m_9$ and bit $m_{10}$ where it finally changes back to state *0*. For the computation of the next change of state, it is sufficient to recognize the end of the present chain. For an optimal efficiency it suffices to consider the two bits after the end of a chain. In this case, the computation time as compared with the multiplication without look-ahead is reduced to an average of 1/3.

In the CP the multiplication is executed in the descending direction, and not as explained in the ascending order. Nevertheless, the look-ahead algorithm works the same way. The amount of computation reduction is identical, only the decision about a change of state is a little bit different but is of no interest for this concern.

The division of the modulo operation with look-ahead operates in quite a different manner. In the following, comparisons of $Z$ with $1/3, 1/6 \ldots$ of $N$ play an important role in the computation of $Z$ mod $N$. Let us assume that during the division of $Z$ by $N$, $N$ was already shifted and *subtracted* several times from $Z$ and let the remainder be stored in $Z$. The special look-ahead algorithm ensures that

$$|Z| \leq \frac{1}{3} N.$$

This will also be true for the following steps as will be shown now. By assumption, $Z$ falls into one of the following ranges:

$$\begin{array}{lrcl} \text{range} \quad si = 2: & \frac{1}{6} * N & < |Z| & \leq \frac{1}{3} * N, \\ \text{range} \quad si = 3: & \frac{1}{12} * N & < |Z| & \leq \frac{1}{6} * N, \\ & \vdots & < & \leq \quad \vdots \\ \text{i.e.,} & \frac{2}{3} * 2^{-si} * N & < |Z| & \leq \frac{4}{3} * 2^{-si} * N, \end{array}$$

for some $si > 1$. Now $N$ is shifted $si$ bits towards the lower bits. For the result $N' = 2^{-si} * N$, we have

$$\begin{array}{rcl} \frac{2}{3} * 2^{-si} * (2^{si} * N') < |Z| & \leq \frac{4}{3} * 2^{-si} * (2^{si} * N), \\ \frac{2}{3} * N' < |Z| & \leq \frac{4}{3} * N' \text{ or} \\ -\frac{1}{3} * N' < |Z| - N' & \leq \frac{1}{3} * N'. \end{array}$$

For the resulting $Z' = Z \pm N'$, $|Z'| \leq 1/3 * N'$ holds again. Since this condition is always met, $N$ is shifted in every step by at least $si = 2\,bit$. However, the expected value of $si$ is 3 thus reducing the computation time to an average of 1/3.

But what is the advantage of computing the modulo operation with the look-ahead algorithm compared with the normal computing? It seems that only subtractions are substituted with comparisons which take approximately the same time as the removed subtractions. Generally, this statement is correct but not with this look-ahead algorithm since it is only necessary to compare for example the highest $10\,bit$ of $Z$ and $N$. Occasionally (in this example every $2^{10}$ steps in average), the algorithm makes a wrong decision, namely if the highest $10\,bit$ of $Z$ and $N$ are identical. However, if this happens, the computation of the multiplication does not fail but only slows down the next step of the modulo operation to a shift value $si = 1\,bit$. Thus it is possible to reach nearly the complete theoretical advantage in a practical implementation of the look-ahead algorithm. The few $10\,bit$ comparisons can be done in parallel with the same number of comparators while the addition circuits compute the next intermediate result $Z$.

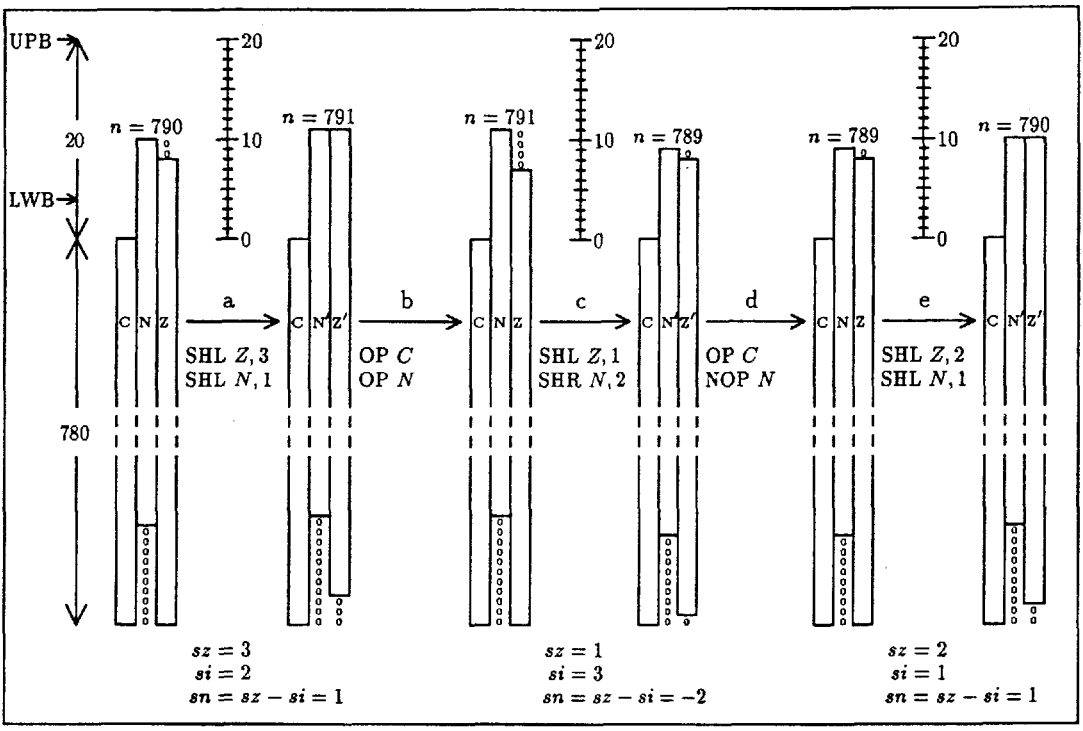


*Figure* 2: On principle three steps of the *flow buffer* are shown. One shift step (SHL or SHR) and one operation step (ADD, SUB or NOP) together last one processor cycle, i.e. $32ns$. The towers represent the registers. SHL $Z, 2$ means, shift left register $Z$ by $2\,bit$, i.e. multiply $Z$ by 4. This is demonstrated in increasing the tower representing $Z$ by 2 units. The abbreviations are: *UPB* → upper bound of flow buffer, *LWB* → lower bound of flow buffer, *SHL* → shift left, *OP* → addition *or* subtraction and *NOP* → no operation.

In reality, however, the improvement of the multiplication and modulo operation by an average of 1/3 is not reached because a realistic algorithm cannot look ahead an unbounded number of bits. In an implementation of this algorithm, a look-ahead bound $k$ is given. If a change of state cannot be decided by considering the next $k$ bits, the algorithm shifts $k$ bits in this step without taking a further action and tries again in the next step. With a bound $k$, the average reduction of the number of operations is developed in [8] as

$$\frac{1}{3} * \frac{2^k + 1}{2^k - 1}.$$

Since only the expected shift values of the two look-ahead algorithms coincide, it is necessary to uncouple them in the CP. In both cases, shift values $sz$ and $sn$ determine the number of bits by which the intermediate result of multiplication $Z$ and the modulus $N$ are to be shifted in each cycle. $Z$ is shifted absolutely, and $N$ is shifted relatively to $Z$. The two shifts are uncoupled by $N$ *floating* on $Z$. This property which is formally deduced in [8], is explained in Fig. 2.

The state of the CP is shown for three consecutive steps involving transitions (a) to (e) of Fig. 2. Transitions (a), (c), and (e) demonstrate the shift operations while (b) and (d) are additions or subtractions not discussed further at this point. The towers represent the registers. Their size is $780 + 20\,bit$. 780 is the maximal word length, and 20 is the size of the buffer. This buffer decouples the look-ahead procedures. If, for example, the shift value of the multiplication is greater than the one of the modulo operation, then $N$ is shifted by the difference partially into the buffer (transitions (a) and (e)). In the opposite case, $N$ is shifted towards the low end (transition (c)). The observation of the buffer bounds is taken care of but not discussed here.

Before transition (a), $N$ was shifted 10 *bit* into the buffer. Shift value $sz$ subsequently takes the values 3, 1, and 2. The shift value $si$ mentioned above which determines the shift of $N$ relative to $Z$, takes the values 2, 3, and 1 meaning that $N$ is absolutely shifted by $sn = sz - si$ hence 1, -2, and 1. After transitions (a), (c), and (e), $N$ is shifted into the buffer by 11, 9, and 10 *bit*, respectively.

In transition (c), the influence of the look-ahead bound $k = 3$ is demonstrated. Although the algorithm might have asked for an $si = 4$ shift, $N$ is moved by 3 *bit* relative to $Z$, and in the transition (e) by 1 *bit*. In transition (d), $N$ is neither added nor subtracted from $Z$.

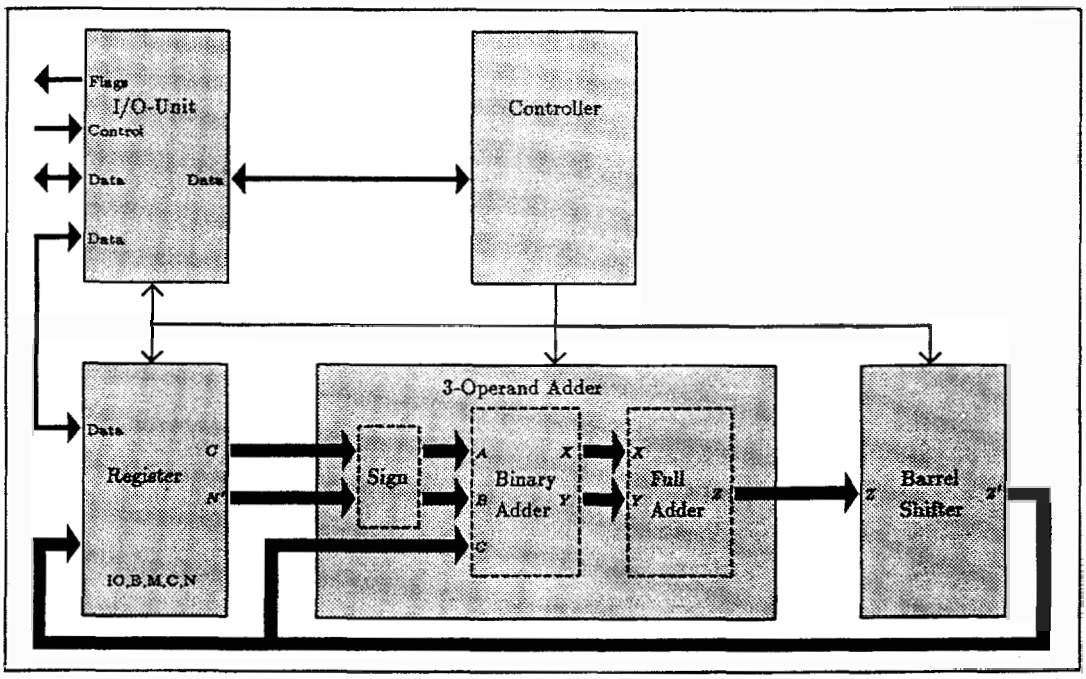


*Figure* 3: The block structure of the CP. In addition to the components discussed so far, the CP consists of an *I/O-Unit*, a *Controller*, a *Register Field* and a *Barrel Shifter*. One task of the Controller is to compute the look-ahead values of exponentiation (cp. [4]), multiplication and modulo operation. The Barrel Shifter is necessary to perform the shifting by several bits in one step.

Two additional features are worth mentioning. First, there are two different adders in the data path (Fig. 3). The first of them is a *one bit slice* adder with three data inputs but without a carry input. It generates two sum bits. In the CP these two bits of every bit slice are fit together to form two new numbers, called $X$ and $Y$. These numbers are directly passed to the second adder in the data path which is a *full adder*. Thus it is possible to perform one multiplication step requiring an addition and one modulo operation step requiring a subtraction in one processor cycle.

Second, the critical phase of each addition is the delay of the carry bit as long as the lowest bit position may influence the highest one. So for the full adder a new *carry look ahead* logic (CLA) was designed reducing the time for carry bit calculation to a minimum. In this CLA, the addition is normally calculated in a single processor cycle. For long carry bit calculations, the CLA increases the calculation time for this addition to more than one cycle. The major advantage of this proceeding lies in the fact that the rate of addition is not determined by the longest lasting addition, but by the average adding time.

The resulting block structure with the major components discussed so far is shown in Fig. 3.

## 3. The Ciphering Rate

The ciphering rate of the CP is determined by the structure of the inplementation of exponentiation. Thus the ciphering is influenced by

- the number of cycles needed for one addition in the full adder,
- the number of steps needed for one multiplication which is determined by the classical look-ahead algorithm of multiplication,
- the number of steps needed for one modulo operation which is determined by the new look-ahead algorithm of modulo operation, and
- the number of modular multiplications needed to perform one exponentiation.

All these points should not be classified by constant values but by probabilistic functions. Before describing these functions we make two assumptions. First, the probability of having a binary *0* or *1* is equal at every binary position of the modulus $N$, the exponent $E$ and the message $M$, the code $C$, respectively, e.g. $\mathcal{P}(X_i = 0) = \mathcal{P}(X_i = 1) = 1/2$. Second, it is meaningful to assume that one step of addition is not depending on the result of the previous addition.

The first assumption is obvious but it is harder to understand why the second one is rich in meaning. What happens if the opposite is true? Of course it is much more difficult to construct the probabilistic functions but this is not important. Important is that now an attacker has the possibility of building a probabilistic algorithm using the dependence between consecutive steps. Thus he had a good chance of breaking all cryptographic methods using modular multiplications without solving the basic problem of this method. Of course it is wrong making the conclusion that the second assumption is true because this attack must be permitted. But it is meaningful to assume it when using the modular multiplication to construct cryptographic methods. Thus five probabilistic functions can be defined which characterize the behaviour of the CP and therefore the ciphering rate completely:

$f_t\,(x) : \mathcal{P}(\#(\text{cycles per addition}) = x)$,

$f_a\,(x) : \mathcal{P}(\#(\text{additions per multiplication}) = x)$,

$f_s\,(x) : \mathcal{P}(\#(\text{cycles per modular multiplication}) = x)$,

$f_m(x) : \mathcal{P}(\#(\text{multiplications per exponentiation}) = x)$, and

$f_e\,(x) : \mathcal{P}(\#(\text{cycles per exponentiation}) = x)$.

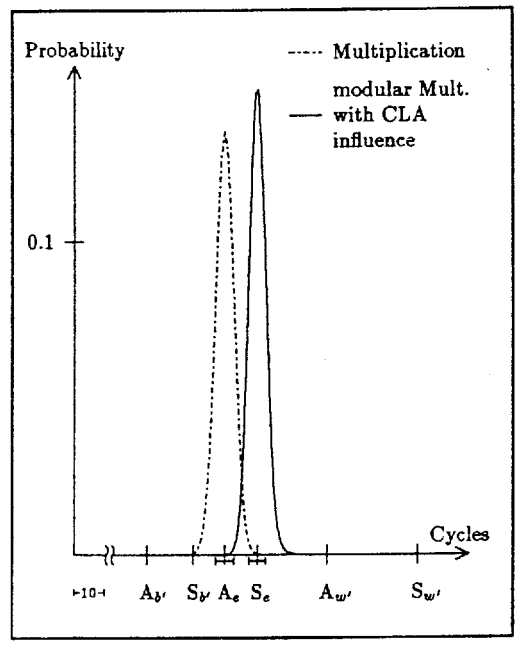


*Figure* 4: The probability distribution of $f_a$ : *Cycles needed for Multiplication* and $f_s$ : *Cycles needed for modular Multiplication on CP.* Rendered prominent are the best case $A_{b'}$,$S_{b'}$, the expexted case (with the standard deviation) $A_e$,$S_e$ and the worst case $A_{w'}$,$S_{w'}$ of execution. The exact values of these cases are listed in Tab. 2.

The function $f_s$ depends on $f_t$ and $f_a$ while $f_e$ depends on $f_s$ and $f_m$. Here is not enough space to describe these dependencies and the mathematical formula of all these functions. Thus the two functions $f_t$ and $f_e$ are described exactly but only an overview is given of building up the three functions $f_a$, $f_s$ and $f_m$. To construct the latter ones, the basic idea is to use the *probabilistic finite state machines* (PFSM) which describe the three look-ahead algorithms for accelerating the multiplication, modulo operation and exponentiation (cp. [8], [9]). Although these machines only

| Function $f_x$ | Best Case $X_b$ | Expected Case $X_e$ | Worst Case $X_w$ |
|---|---|---|---|
| $f_t$ | 1 | $1 + \mathcal{P}(\text{Panic}) * \left(\left\lceil \frac{1}{4 \times 20} \log_2(N) \right\rceil - 1\right)$ | $\left\lceil \frac{1}{4 \times 20} \log_2(N) \right\rceil$ |
| $f_a$ | $\frac{1}{k}\lceil \log_2(N) \rceil$ | $\frac{1}{3}\lceil \log_2(N) \rceil * \frac{2^k+1}{2^k-1}$ | $\frac{1}{2}\lceil \log_2(N) \rceil$ |
| $f_s$ | $A_b$ | $(A_e + c(k) * B_L) * T_e$ | $A_w * T_w$ |
| $f_m$ | $\lceil \log_2(N) \rceil$ | $\frac{4}{3}\lceil \log_2(N) \rceil$ | $\frac{3}{2}\lceil \log_2(N) \rceil$ |
| $f_e$ | $M_b * S_b =$ | $M_e * S_e \approx$ | $M_w * S_w \approx$ |
| | $\frac{1}{k}\lceil \log_2(N) \rceil^2$ | $\frac{4}{9}\lceil \log_2(N) \rceil^2 * \frac{2^k+1}{2^k-1} + \frac{4}{3}\lceil \log_2(N) \rceil * c(k) * B_L$ | $\frac{3}{320}\lceil \log_2(N) \rceil^3$ |

*Table* 1: The exact values of the best, the expected and the worst case ($X_b$, $X_e$ and $X_w$) of the probability distributions $f_t$, $f_a$, $f_s$, $f_m$ and $f_e$. The values are computed by complexity theory. However, it is not a good choice to characterize the abilities of the CP by $E_b$ and $E_w$. (The function $c(k)$ returns a result in the range $[0,1]$, $B_L$ is the length of the flow buffer.)

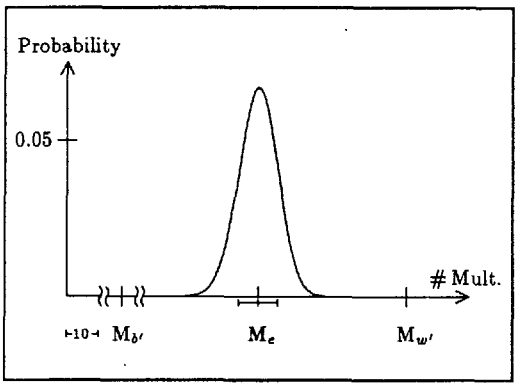


*Figure* 5: The probability distribution of $f_m$ : *Multiplications needed for Exponentiation.* Rendered prominent are the best case $M_{b'}$, the expexted case (with the standard deviation) $M_e$ and the worst case $M_{w'}$. The exact values of these cases are listed in Tab. 2.

allow to compute the best case $X_b$, the expected case $X_e$ and the worst case $X_w$ of execution (Tab. 1), they can be used to construct a new PFSM which characterizes the dynamic and not only the static behaviour of the CP. Based on these new machines one is able to compute the required functions (Fig. 4, 5). During this computation, the probability is represented not only as a scalar but as a vector due to the PFSM. The functions return as result the sum of all vector *components*.

The function $f_t$ depends on the implementation of the full adder. In the CP this 780 *bit* adder is realized as a chain of 20 *bit* sized adders, called *block adders*. Each block adder has an attached *block CLA* element. Generally, both together are able to compute in one processor cycle of 32 *ns* the 20 *bit* result and early enough the *block carry bit* so that the next higher block adder is able to compute its result in attention to the mentioned carry bit. So the complete addition of 780 *bit* sized numbers takes only one processor cycle.

But what happens if this carry block bit depends on the carry block bit of the next lower block adder? Now there is not enough time to calculate the block carry bit early enough for the next higher block adder. Therefore this block CLA gets into *panic*! But there is a magician in the CP who avoids the failing of computation. He notices the panic of this unlucky block adder, flourishes his magic wand and suddenly there is even enough time for traveling the block carry bit from the lowest block adder to the highest one. In a more technical language this wonderful magician is simply an interrupter which notices the normally unused *block carry propagate bit* and, when activated by this bit, it disconnects for a certain amount of cycles the triggering clock lines of the *Ciphering Units* (CU). The probability of this scenario is coupled to the probability that one block adder gets into panic which is

$$\mathcal{P}(\text{One block adder gets into panic}) = 2^{-20}.$$

Due to the acting look-ahead algorithms of multiplication and modulo operation this probability is not as obvious as

it seems to be. The algorithms cause the binary *0*s and *1*s to appear not with the same probability at the input of the binary adder and even at one input line of the full adder (Fig. 3). But as in [9] deduced this inequality does not influence the appearence of *0*s and *1*s in the addition result $Z$, just as the carry propagate signal of each bit slice is not affected. Because only these carry propagate signals determine the block carry propagate bit the derived probability is correct. Hence the probability of one CU geting into panic is (considering that the lowest block adder cannot get into panic)

$$\begin{aligned}
\mathcal{P}(\text{Panic}) &= 1-(1-\mathcal{P}(\text{One block adder gets into panic}))^{\#(\text{block adders of CP})-1} \\
&= 1-(1-2^{-20})^{\lceil \frac{1}{20}\log_2(N)-1\rceil} \\
&= 1-\sum_{i=0}^{\lceil \frac{1}{20}\log_2(N)-1\rceil}\binom{\lceil \frac{1}{20}\log_2(N)-1\rceil}{i}(-2^{-20})^i \\
&= 2^{-20}\left\lceil \frac{1}{20}\log_2(N)-1\right\rceil - \sum_{i=2}^{\lceil \frac{1}{20}\log_2(N)-1\rceil}\binom{\lceil \frac{1}{20}\log_2(N)-1\rceil}{i}(-2^{-20})^i \\
&= 2^{-20}\left\lceil \frac{1}{20}\log_2(N)-1\right\rceil - \mathrm{O}(\varepsilon) \\
&\simeq 2^{-20}\left\lceil \frac{1}{20}\log_2(N)-1\right\rceil .
\end{aligned}$$

Now it is easy to derive the required function $f_t$ (the block CLA chain is able to travel a block carry bit over four 20 *bit* CLA elements in one processor cycle)

$$f_t(x)=\begin{cases} 1-\mathcal{P}(\text{Panic}) & \text{if } x=1; \\ \mathcal{P}(\text{Panic}) & \text{if } x=\left\lceil \frac{1}{4\times 20}\log_2(N)\right\rceil; \\ 0 & \text{otherwise.}\end{cases}$$

For the computation of $f_m$ (Fig. 5), only the corresponding PFSM representing the look-ahead algorithm of exponentiation is used as base (cp. [4], [9]). This function and the above-mentioned $f_s$ determine $f_e$ as follows. Required is the probability that one exponentiation takes $x$ cycles of time which is $f_e(x)$. The exponentiation is divided into an indefinite number of multiplications assuming the exponent is statistically distributed. This probability distribution of the number of multiplications is described by $f_m$. Each of these multiplications is independent of the other ones. The number of cycles needed to perform every multiplication is therefore independent, too. Furthermore, the number of cycles to perform one multiplication is unknown and only described by the probability distribution $f_s$. The sum of all cycles for performing all multiplications has to be $x$. Thus the derivation from $f_e$ is

$$\begin{aligned}
f_e(x)=\mathcal{P}(\mathrm{E}=x) &= \sum_{m=1}^{\infty}\left\{\mathcal{P}(\mathrm{M}=m)*\mathcal{P}(\sum_{i=1}^{m}\mathrm{S}_i=x)\right\} \\
&= \sum_{m=\mathrm{M}_b}^{\mathrm{M}_w}\left\{\mathcal{P}(\mathrm{M}=m)*\sum_{(s_1,\ldots,s_{m-1})\in\mathcal{N}^{m-1}}\left\{\mathcal{P}(\mathrm{S}_m=x-\sum_{i=1}^{m-1}s_i)*\prod_{i=1}^{m-1}\mathcal{P}(\mathrm{S}_i=s_i)\right\}\right\} \\
&= \sum_{m=\mathrm{M}_b}^{\mathrm{M}_w}\left\{f_m(m)*\sum_{(s_1,\ldots,s_{m-1})\in[\mathrm{S}_b,\mathrm{S}_w]^{m-1}}\left\{f_s(x-\sum_{i=1}^{m-1}s_i)*\prod_{i=1}^{m-1}f_s(s_i)\right\}\right\}.
\end{aligned}$$

The curve of $f_e$ is very funny, it looks like a cookscomb (Fig. 6). This is true for a statistically independent distribution of the exponent. For a selected key the exponent is static and therefore the curve reduces to one peak of the cookscomb.

The above-mentioned functions $f_a$ and $f_s$ are shown in Fig. 4 whereby $f_a$ is computed in the described way of using as base only the PFSM representing the look-ahead algorithm of multiplication. An addition time of one cycle per step is assumed, too. On the other hand, for the computation of $f_s$ the influence of flow buffer bounds (Fig. 2) and the functions $f_t$ and $f_a$ are used in addition to the PFSM of modulo operation. Amazingly, the standard devi-

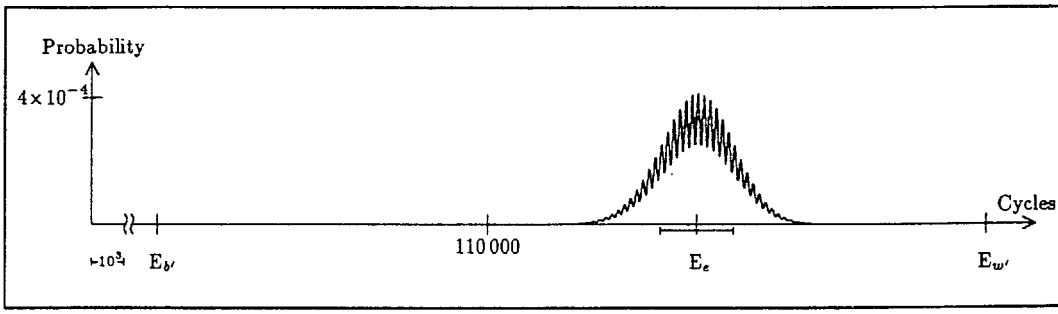


*Figure* 6: The probability distribution of $f_e$ : *Cycles needed for modular Exponentiation on CP.* Rendered prominent are again the best case $E_{b'}$, the expexted case (with the standard deviation) $E_e$ and the worst case $E_{w'}$. The exact values of these cases are listed in table 2.

| $\lceil \log_2(N) \rceil = 440; \quad k = 3; \quad B_L = 20$ | | | |
|---|---|---|---|
| Function $f_x$ | Best Case $X_{b'}$ | Expected Case $X_e$ | Worst Case $X_{w'}$ |
| $f_t$ | 1 | 1.00013 | 6 |
| $f_a$ | 164 | 189.102 | 213 |
| $f_s$ | 179 | 199.693 | 250 |
| $f_m$ | 499 | 584.778 | 631 |
| $f_e$ | 99 660 | 116 776.181 | 125 955 |

*Table* 2: The values are computed by analyzing the probability distributions and defining the worst and the best case in a more technical than in the former mathematical manner (see the text). The new values of $X_{b'}$, $X_{w'}$ and specially $E_{w'}$ are very suitable to characterize the abilities of the CP.

ation of $f_s$ is smaller than the standard deviation of $f_a$ although assuming not an addition time of one cycle per step due to $f_t$. This aspect results in the flow buffer and the statistical independence of multiplication steps and modulo operation steps. If the CP gets a fast multiplication the probability that the intermediate result is shifted toward the upper bound of flow buffer is high so in average the CP has to perform some additional modulo operation steps. If on the other hand the CP gets a slow multiplication, the probability that the intermediate result is near the lower bound of flow buffer is high, so in average the CP has to perform much less additional modulo operation steps than before. This coherence effects the decrease of standard deviation.

Now one has all suppositions to compute the ciphering rate. However, it is important in which way the worst case is defined because it determine the ciphering rate. From the view of complexity theory it is necessary to use the values of Tab. 1. But these values do not characterize very well the abilities of CP in a practical employment. Of course the absolute worst case of execution is possible. But in every transmission line a protocol is performed to detect several different situations in which one of the communication partners is not able to transmit or receive data. Also this protocol is used to detect transmission errors. If the CP is not able to compute the required value fast enough it can be conceived as a transmission error by the receiver. Thus the sender is solely invited to transmit the value again.

However, it is important to fix the probability of this scenario to a value much less than the normal error rate of transmission lines. The latter probability is assumed to be $10^{-12}$ (a fantasticly good value). Thus for the computation of the values in Tab. 2, the technically best case $X_{b'}$ and worst case $X_{w'}$ are defined as

$$X_{b'} : \mathcal{P}(x \leq X_{b'}) = 10^{-18}, \text{ and}$$
$$X_{w'} : \mathcal{P}(x \geq X_{w'}) = 10^{-18}.$$

As described later, the CP consist of two independent CUs. The larger one has a size of 440 *bit*. They are able to decode parallelly a 780 *bit* sized number. Therefore the technical ciphering rate of CP is

$$\text{Ciphering Rate (CP)} = \frac{\text{Size (Code)}\,[Kbit]}{\text{E}_{w'}[cycle] * 32\,[ns/cycle]} \simeq 195\,[Kbit/sec].$$

4. Features of the RSA Cryptography Processor

All RSA implementations known to the author suffer from one or more of the following restrictions: the ciphering rate is too low (e.g. the "Security Processor C.R.I.P.T." in [5] with a rate of less than $10\,Kbit/sec$); the key length is too short; or several chips are required (e.g. the "NEC/Miyaguchi" design in [4] with a rate of $29\,Kbit/sec$, a key length of $660\,bit$ and a solution of 333 chips).

In contrast to all of them is the CP. The heart of it consists of two independent CUs with a size of 340 and $440\,bit$. For encoding, both CUs cooperate as a single $780\,bit$ CU, for decoding each of them works independently using the advantages of the Chinese Remainder Theorem. This results in an encoding rate of $3\,Mbit/sec$ when using as the exponent Fermat's 4th number and in a decoding rate of $0.2\,Mbit/sec$ if the two primes involved do not exceed the size of the CUs.

Furthermore, the CP has a shell around its kernel. So it is not only a very fast ciphering processor but a "Cryptograhic Area" which means that

- "Cryptographic Actions" like "Load a new key and accept it only if the certificate is correct" can only be started but not manipulated from outside the chip,
- signatures (certificates) are automatically generated respectively checked,
- secret memory is neither readable nor writeable from outside the chip, and
- it exists the possibility of generating new keys.

The generation and checking of signatures requires a hash function with the abilities of a One-Way function. Such a function is implemented. It is the one discussed at ISO ([6], [7]) and not as formerly described the one developed at SEPT [5]:

1. Insert between every four bits of $M$ four binary 1's,
2. Divide $M'$ into $m$ blocks $B_i$ less than $N$,
3. Perform
$$H_0 = 0,$$
$$H_i = (H_{i-1} \oplus_2 B_i)^2 \bmod N,$$
4. $H_m$ is the result.

The key generation is done by the Monte Carlo method using only the strong pseudoprime test. The two primes $p$ and $q$ were computed in the known way: $p-1$ has a large factor $p'$, $p'-1$ has a large factor $p''$, and the same for $q$, respectively. But in addition to that caution, $q$ is calculated such that

$$\gcd(N-1, E) = \gcd(p \times q - 1, E) = E = 2^{2^4} + 1.$$

This property prevents an illegal installation of a key $N_{Attacker}$ which is not certificated by the secret key $D_{Authority}$ but by the manipulated "secret" key

$$D'_{Attacker} \equiv E^{-1} \bmod (N_{Authority} - 1).$$

This attack is possible if $N_{Authority}$ is a pseudoprime to the base $H(N_{Attacker})$ and if $\gcd(N_{Authority} - 1, E) = 1$. The possibility of the last condition is very high if the two primes are computed in the known way. Although the first condition of finding such a pseudoprime is very low the presented way of key generation is implemented in the CP to prevent this attack.

For users, the following features of the CP are of interest:

- protected "Cryptographic Area",

- key generation takes an average time of 2 *sec*,
- operation as a co-processor,
- one DMA channel for a data stream of 200 $Kbit/sec$, or
- three DMA channels for three independent data streams of 64 $Kbit/sec$,
- 8 *bit* wide data bus,
- independent operation of the CU and the I/O Unit,
- about 160 000 transistors in 1.5$\mu$-CMOS technology, and
- chip size of approximately $5mm \times 4.8mm$.

## 5. Conclusions

So far, a prototype consisting of about 5000 5$\mu$-NMOS transistors has been realized. This chip contains all major parts of the CP in a small version. Furthermore, based on the experience with the 5$\mu$-NMOS process, the performance rates of the 1.5$\mu$-CMOS process were obtained by theoretical considerations as well as by full length simulations. The development of a 1.5$\mu$-CMOS design of the CP is now completed and therefore, first prototypes will be available in summer 1988.

## 6. References


[1] Hellmann,M.E., Die Mathematik neuer Verschlüsselungssysteme, Spektrum der Wissenschaft 10, 1979, 92-101.

[2] Meyer,C.H., Matyas,S.M., Cryptography - A new Dimension in Computer Data Security, Cryptography Competence Center, IBM, Kingston, New York, Wiley-Interscience Publication, 1982.

[3] Pomerance,C., Recent Developments in Primality Testing, The Mathematical Intelligencer 3, 1981, 97-104.

[4] Rivest,R.L., RSA Chips (Past/Present/Future), in "Advances in Cryptology", Proc. Eurocrypt 84, Paris, France, 1984, 160-164.

[5] Pailles,J.C., Girault,M., The security processor C.R.I.P.T., Information Security: The Challenge, Preprints of the Fourth IFIP Security on Information Systems Security, Monte-Carlo, December 2-4, 1986.

[6] A Hash function for use in digital signature and authentication (Exclusive OR and squaring), Working Document - First Draft, Secrétariat ISO/TC 97/SC 20/WG 2 N75, Paris, France, 1986.

[7] TeleTrusT/OSIS, The TeleTrusT/OSIS Hash-Function, TeleTrusT/OSIS - WG1 Security Aspects, GMD, Darmstadt, Germany, 1986.

[8] Sedlak,H., Konzept und Entwurf eines Public-Key-Code Kryptographie-Prozessors, Institut für Theoretische und Praktische Informatik, Technische Universität Braunschweig, Germany, 1985.

[9] Sedlak,H., Theory and Single-Chip Implementation of a Cryptography Processor, Institut für Theoretische Informatik, Technische Universität Braunschweig, Germany, to be submitted as a PhD-Thesis.