CORSAIR: A Smart Card for Public Key Cryptosystems

Dominique de Waleffe & Jean-Jacques Quisquater

Philips Research Laboratory
Avenue Albert Einstein, 4
B–1348 Louvain-la-Neuve, Belgium
E-mail: {ddw, jjq}@prlb.philips.be

Abstract. *Algorithms best suited for flexible smart card applications are based on public key cryptosystems — RSA, zero-knowledge protocols ... Their practical implementation (execution in ≈1 second) entails a computing power beyond the reach of classical smart cards, since large integers (512 bits) have to be manipulated in complex ways (exponentiation). CORSAIR achieves up to 40 (8 bit) MIPS with a clock speed of 6 Mhz. This allows to compute X^E mod M, with 512 bit operands, in less than 1.5 second (0.4 sec for a signature). The new smart card is in the final design stage; the first test chips should be available by the end of 1990.*

Keywords: smart card, public key algorithms, RSA, digital signature, zero-knowledge protocols.

1 Introduction

A large number of security problems can be solved by correct use of cryptographic methods. However, all methods found to date are more or less computationally intensive.

- DES works by applying a complex multiround algorithm on medium size numbers.
- Diffie-Hellman key exchange protocol is based on modular exponentiation of large integers.
- RSA is based on the same exponentiation and needs large exponents.
- Zero-knowledge protocols like those of Fiat-Shamir [9] or Guillou-Quisquater [10] use large number exponentiation but the exponents are not as large as in RSA.
- Many identity-based systems also rely on modular exponentiation of large numbers.

Public key techniques are the most promising for the future as they provide more flexible solutions and impose less burden both on users and security management. Most practical techniques rely on large integer arithmetic.

the correct result at the end. This allows one to reduce the size of manipulated numbers without doing modulo reductions before the very last step.

Novel algorithms [1] exploit that property and the reduction step is a simple subtraction of an adequate multiple of the modulus. As is explained in [1], one can do a multiplication step with a 24 bit y then follow by a reduction step where the modulus is multiplied by a 32 bit number before the subtraction. Further, the algorithm guarantees that intermediate values are always positive.

When the modulus is stored in 2's complement form, both multiplication and reduction steps are based on the operation

$$B \leftarrow y \cdot X + A$$

where all operands are positive numbers. This formula is the basic operation performed by the special cell.

3 First model

The width of data words used in storage and handling of information in current smart cards is one major architectural constraint. Most cards are based on 8 bit CPU's and memories and there is no likely radical change in the foreseeable future, primarily due to the chip size limit of 25 mm^2 of the ISO standard.

Adding a cell based on a different word size would further imply a major redesign of many systems components and may not bring the best performance due to the discrepancy in size of items.

A consequence of these facts is a coprocessor which manipulates 8 bit quantities, both internally (size, complexity, delays issues) and externally (interfacing issues).

The needed arithmetic operations can be performed quite fast with random logic; thus the limiting factor of the coprocessor's performance appears to be its ability to read input operands and write back results, which all are large integers. One goal of the architecture was thus to optimize the use of the available bus bandwidth.

Once it was shown that arithmetic operations could be performed in less than one memory access cycle, a direct connection between the cell and the memory system was introduced. This interface also comprises a number of auto-(inc)decrement pointers which give the cell some independence. These features allow the cell to access memory 6 million times per second.

Let us leave these interfacing considerations and examine the internal architecture. Using 8 bit data naturally leads to an 8 by 8 multiplier; if one also remembers that $(b-1)^2+2(b-1) < b^2$, one knows that adding two 8 bit numbers to the multiplication result is still a 16 bit number. The architecture in figure 2 exploits this property.

Note that, y considered as an 8 bit constant, the operation performed, $B \leftarrow y \cdot X + A$, necessitates 3 memory accesses for each multiplication (two "read", one "write").

This cell contains 3 pointers, one for each large integer operand. These pointers have auto-decrementers, hence they are updated automatically after each access. There are a cycle counter, command and status registers. The cycle counter, y and the pointers are setup by the CPU which then writes the start bit of the command register, allowing the cell to start.

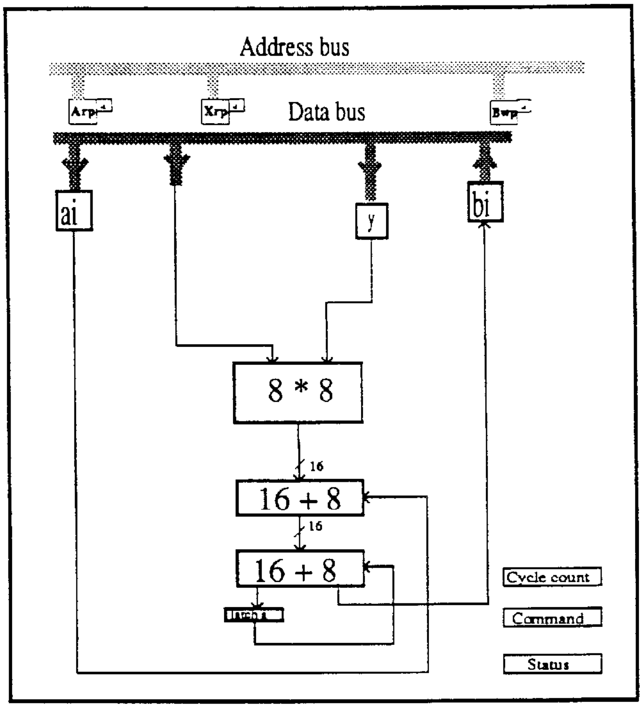

Figure 2: Simple cell

The sequencer for this cell can be summarized by the following algorithm[1]:

```
$for(; cyclecount--;)\{$
    $/* bus\ cycle\ 0 */$
    $ai = *Arp--;$
    $/* bus\ cycle\ 1 */$
    $parbegin$
        $tmp = *Xrp-- \cdot yi + latcha + ai;$
        $latcha = highbyte(tmp); bi = lowbyte(tmp);$
    $parend$
    $/* bus\ cycle\ 2 */$
    $*Bwp-- = bi;$
$\}$
```

Analysis of the sequencer shows that one memory access is made during each available bus

[1] A C-like syntax is adopted with *parbegin* and *parend* used to show the operations performed in parallel during one bus cycle. A temporary variable is introduced for simplicity. Pointers represent large integers, hence they start by an uppercase letter.

cycle. However, only one cycle out of three is used for the multiplication and accumulation. This is not an optimal use of the hardware.

Also, input operands must be padded with one null byte to allow for a correct last result byte.

Exploiting this cell for a complete algorithm is also quite cumbersome.

Despite its problems, this architecture is already a 40 fold improvement over a software implementation of the primitive on the 8051 CPU.

4 Final architecture

Using the multiply and accumulate hardware during one third of the cycles is the most important drawback of the simple model. However there is a remedy: let us multiply the large operand X by a 3 bytes y operand. Looking at hand multiplication, one can see that this amounts to computing three partial products and accumulating them simultaneously. Of course, more hardware is needed to do so. Figure 3 shows this extended model.

Three y registers are necessary to hold 24 bits, those are initialized when the cell starts its operations. Also, to improve matters, the y registers have an associated pointer (Yrp) which is used and auto-incremented[2] at each access.

A register xi to hold the current byte of X must be introduced since the same byte will be used 3 cycles in a row, once for each y byte.

The multiplication of a large integer by a 24 bit number is simulated by the proper interleaving of three multiplications of the said large number by three 8 bit numbers. This interleaving requires internal memory and datapaths which allow to recycle the temporary results into the final result. A pipeline structure for input and output also allows the interleave the memory accesses with the multiplications.

The sequencer is also quite easily described by a small algorithm, the reset cycles during which the y operand registers are loaded have been omitted (see also figure 4):

[2] The incrementer allows to easily implement the English method of multiplication, without reloading the Yrp register at each step.

```
for(; cyclecount − −; ){
    parbegin / ∗ bus cycle 0 ∗ /
        $ai' = *Arp - -;$
        $tmp = xi \cdot y[0] + latchd + ai;$
        $latchb = latcha;$
        $latcha = highbyte(tmp); bi = lowbyte(tmp);$
    parend
    parbegin / ∗ bus cycle 1 ∗ /
        $xi' = *Xrp - -;$
        $tmp = xi \cdot y[1] + latchd + latcha;$
        $latchd = latchb;$
        $latcha = highbyte(tmp); latchb = lowbyte(tmp);$
    parend
    parbegin / ∗ bus cycle 2 ∗ /
        $*Bwp - - = bi;$
        $tmp = xi \cdot y[2] + latchd + latcha;$
        $latchd = latchb;$
        $latcha = highbyte(tmp); latchb = lowbyte(tmp);$
        $xi = xi'; ai = ai';$
    parend
}
```

Examining the sequencer (see figure 4 for a graphical view of the sequencer) and going through an example shows that all registers are necessary, and thus make up an optimal implementation of the algorithm.

Further, one also sees that during every available bus cycle, a multiplication step is performed as well as a memory access (read or write), the scheme is thus optimal in its use of the available hardware.

Unfortunately, the system as designed now prevents the CPU from doing anything while the cell is computing, turning it the other way around, the cell can not be used while the CPU works. If, somehow, both units could be active at the same time, the full capability of the novel algorithms could be used.

In these new algorithms there are no tests except for the number of loops to performs. Therefore, while one step, say multiplication, is being run by the cell, the next operation, reduction step, can be prepared by the CPU. This preparation consists in computing operand pointers, number of cycles, ...

A dedicated bus was thus introduced and is reserved to the cell. This allows the CPU to access ROM and EEPROM while the cell access the RAM at full speed.

This is not sufficient though, because the cell must receive new parameters at each step. By providing double control registers in the cell and making them accessible as internal CPU registers, the CPU can prepare the parameters, load them into the cell. Then as soon as the cell completes the current operation, the CPU can start it again. With this trick, the cell can be kept busy almost all the time during an exponentiation step.

As said in section 2, the algorithm also needs a multiplication of a large integer by a 32 bit number. The model has then been extended at the expense of optimality to provide such

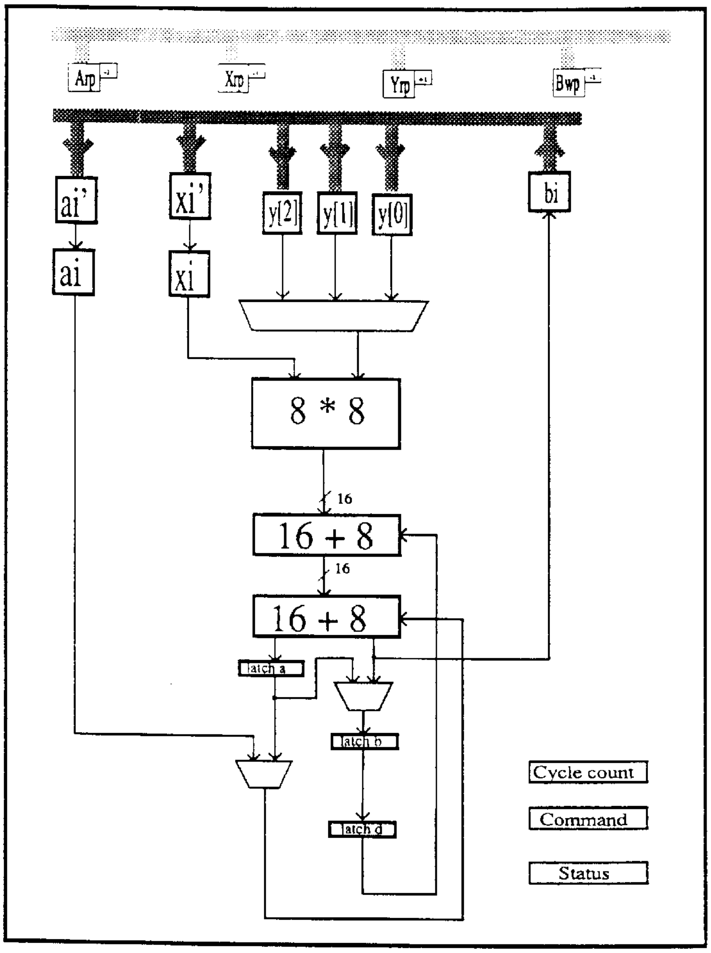

Figure 3: Intermediate architecture

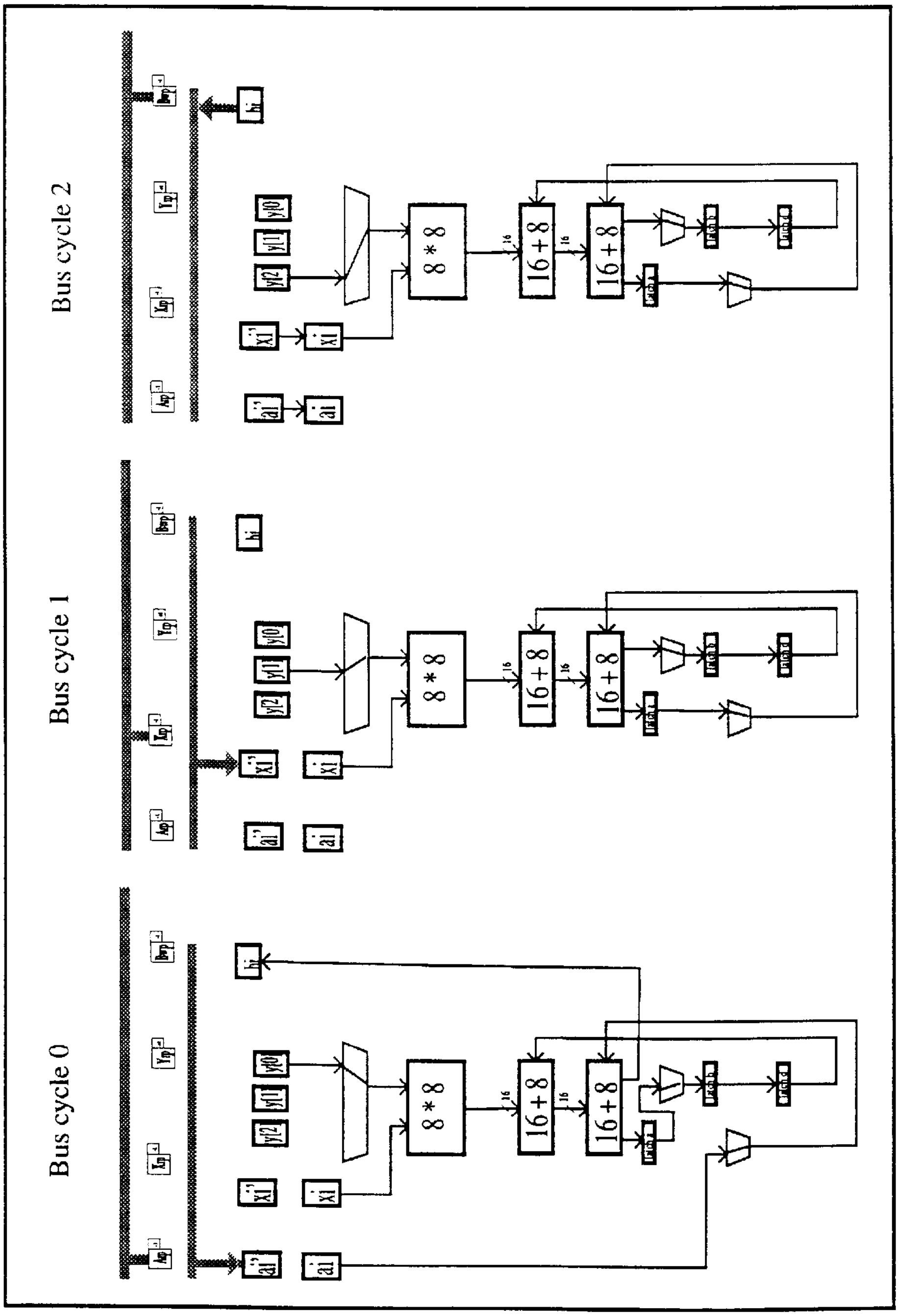

Figure 4: Activity during bus cycles

	trivial $X^E \bmod N$	signature (CRT)
8051 software	$\approx$ 160 sec	$\approx$ 40 sec
8051 + CORSAIR	$\approx$ 1.5 sec	$\approx$ 0.4 sec
6805 software + 8*8	$\approx$ 36 sec	$\approx$ 9 sec

Table 1: Performance comparison for 512 bit operands

capability. The detailed and complete architecture is shown in figure 5. Several important changes have been introduced to allow :

1. a second operating mode for multiplication by a 32 bit number (y[3..0], *latchc*), this mode also needs fourth bus cycle during which no memory access can be made,
2. to empty the pipeline without padding operands (*read_A_lim* and *read_X_lim*),
3. shifts of up to 4 bytes (*write_B_lim*, b[3..0]),
4. the CPU to preload operation parameters while the cell runs (pipeline for control registers),
5. selective disabling of reading or writing operands by setting command bits,
6. allow large integers exclusive-or operations.

With all those features, the cell can also do very fast block moves, memory initializations, shifts, reuse a 4 byte result as y operand for the next operation. These operations can be used to improve the performance of many primitives manipulating large integers on smart cards.

5 Performance evaluation

An implementation of the modular exponentiation algorithm on currently available smart cards takes between 1 and 4 minutes to compute[3] $X^E \bmod M$, with 512 bit operands. This delay is not acceptable for many smart card applications because they require interactions between the user and system.

CORSAIR performance is due to several reasons:

- The base cycle time is one bus cycle, instead of several if assembly language is used.
- Multiply and double accumulate in one cycle.
- A high degree of parallelism:
  - With the 2 busses, the CPU can still execute instructions while the cell is running, for instance it can prepare the next cell activation;

[3] Our estimation for the 8051 CPU is close to 3 minutes without any optimization, while Amos Fiat (rump session of Crypto '90 and private communication) reported 9 seconds for a signature (CRT) on a 6805 enhanced by a 8 bit multiplier. Extrapolating gives a time of 36 seconds for the general case on the 6805.

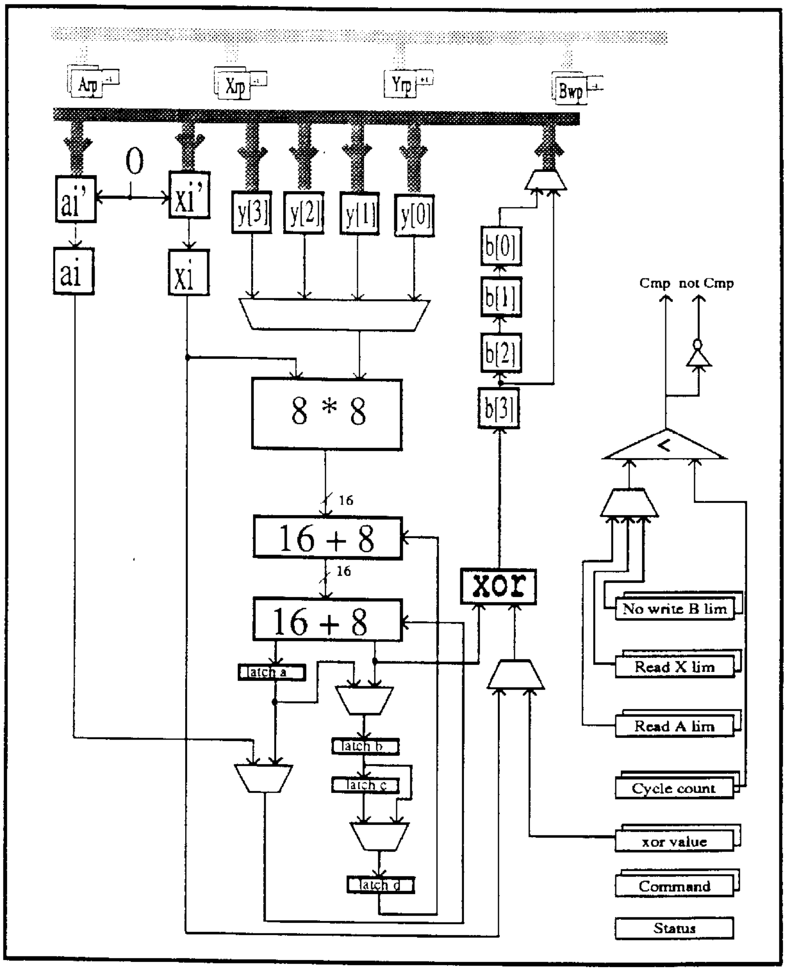

Figure 5: Final architecture

  - Inside the cell, many operations are performed in parallel: accessing RAM or EEPROM, updating pointers, computing and moving data.

- Dedicated bus bandwidth is fully used.

As one knows that it takes on the average a little more than 8 million multiplications (8 by 8) for an exponentiation with 512 bit operands, CORSAIR only needs a little more than 8 million bus cycles for the computation, that is close to 1 second at 8 Mhz or less than 1.5 second at 6 Mhz.

Such a performance not only allows RSA operations but also makes possible very fast implementations of zero-knowledge protocols on the smart cards.

It must be noted that the architecture is not limited to 512 bit numbers, using the Chinese Remainder Theorem should allow signatures with numbers up to 800 bits with the same quantity of RAM (256 bytes).

6 Conclusion

CORSAIR will soon be the first smart card (complying to standards) which allows public key system to be practically implemented.

Providing such a high performance to smart cards is only one step in the right direction. The complete card must also include physical protection in the form of frequency and voltage detectors, ...

Security applications must be designed to securely exploit the capabilities of the card. Zero-knowledge protocols, signature schemes implemented on CORSAIR form the next step towards more manageable secure environments.

Acknowledgments: We are indebted to J.-P. Boumas (Philips Components, Paris) for his fruitful collaboration.

References

[1] J.-J. Quisquater, *"Fast modular exponentiation without division"*, Rump session of EUROCRYPT '90.

[2] J.-J. Quisquater, L. Guillou, *"Des procédés d'authentification basés sur une publication de problèmes complexes et personnalisés dont les solutions maintenues secrètes constituent autant d'accréditations"*, SECURICOM 89, Paris, April 1989.

[3] D. Denning, *"Cryptography and Data Security"*, Addison-Wesley Publishing Company, 1985.

[4] R. Rivest, A. Shamir, L. Adleman, *"A method for obtaining digital signatures and public-key cryptosystems"*, C. ACM, Vol. 21, No. 2, pp. 120–126, 1978.

[5] M. Gardner, *"A new kind of cipher that would take millions of years to break"*, Scientific American, Vol. 237, No. 2, pp. 120–124, Aug. 1977.

[6] R. Bright, *"Smart cards. Principles, practice, applications"*, Ellis Horwood Limited, 1988.

[7] W. Diffie, M. Hellman, *"New directions in cryptography"*, IEEE Trans. Informat. Theory, Vol. IT-22, pp. 644–654, Nov. 1976.

[8] J.-J. Quisquater, D. de Waleffe and J.-P. Bournas, *"CORSAIR: A chip card with fast RSA capability"*, Proceedings of Smart Card 2000, Amsterdam, 1989, to appear.

[9] A. Fiat, A. Shamir, *"How to prove yourself: practical solutions to identification and signature problems"*, Proc. of CRYPTO '86, Lecture notes in Computer Science, Springer Verlag, Vol. 263, pp. 186–194, 1987.

[10] L. C. Guillou, J.-J. Quisquater, *"A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory"*, Proc. EUROCRYPT '88, Lecture notes in Computer Science, Springer Verlag, Vol. 330, pp. 123–128.

[11] L. C. Guillou, M. Ugon, *"Smart Card: a highly reliable and portable security device"*, Proc. of CRYPTO '86, Lecture notes in Computer Science, Springer Verlag, Vol. 263, pp. 464–489, 1987.