A Description of a Single-Chip Implementation of the RSA Cipher

Ronald L. Rivest, Massachusetts Institute of Technology

In 1976 Diffie and Hellman introduced the revolutionary concept of a *public-key cryptosystem* (Diffie and Hellman, 1976). Unlike classical cryptosystems, in a public-key cryptosystem the encryption and decryption keys are *different*. Moreover, given just one of the two keys, it is computationally prohibitive to calculate the other, although it is simple to create the matched pair of keys in the first place. This paradoxical property enables one to create flexible and powerful key distribution methods and to implement true "digital signatures" that are invaluable in the design of modern information protection and authentication systems.

To illustrate these capabilities, consider the following implementation of "digital signatures." Every user of a public-key cryptosystem publishes a decryption key and keeps the matching encryption key secret. To create a signature S for a message M, he merely encrypts M with his secret encryption key; the resulting ciphertext is S. Anyone else can verify the validity of a message-signature pair (M,S) by checking that the signature S decodes to M under the signer's public encryption key. It is not possible to forge signatures or modify a signed message, given that knowledge of the signer's decryption key does not imply knowledge of the signer's encryption key.

To ensure confidentiality of communications, each user A of a communication system can publish his encryption key EA while keeping the corresponding decryption key D secret. Whenever someone wishes to send A a private message, he can encrypt the message M with A's published key to obtain the ciphertext $C = EA(M)$. Even though an eavesdropper knows what the encryption key is, the nature of a public-key cryptosystem and the fact that the eavesdropper doesn't know the *decryption* key prevent him from reading the mail. When A receives the mail, he can decipher it using DA.

This first practical proposal for a public-key encryption algorithm was by Rivest, Shamir, and Adleman (Rivest, Shamir, and Adleman, 1978). This scheme, based on the difficulty of factoring large integers, has become known as the "RSA method" and has received extensive coverage in the popular and technical press (Gardner, 1977; Lempel, 1979; Diffie and Hellman, 1979; Blakely and Blakely, 1978-79, for example.) No way of "breaking" the code has been found to be quicker than factoring the large composite integer n which is part of the key; yet factoring large composite integers remains one of the "computational impossibilities" of our time.

A potential disadvantage of the RSA method is that encryption and decryption are computationally demanding, requiring up to several hundred multiplication of several hundred bit numbers. A typical microprocessor-based implementation might achieve an encryption rate of ten bits per second, while a costly TTL implementation (e.g., $3K, 160 chips) might reach 6K bits/second. Some interesting "hybrid" RSA/DES schemes have been developed which use RSA for key distribution only and DES for fast data encryption. This paper describes a cost-effective alternative: a special-purpose LSI chip to implement the RSA method. Shamir, Adleman, and I have designed such a "big-number ALU" chip which can support all of the usual big-number operations, including those needed to perform RSA encryption. This "RSA chip" has a 512-bit ALU and eight general-purpose 512-bit registers; it can perform RSA encryption at rates in excess of 1200 bits/second (even faster if keys shorter than the maximum length are used).

The RSA Method

The encryption method is only summarized here; the reader is referred to Rivest, Shamir, and Adleman, 1978, for a full exposition.

An RSA encryption key consists of a pair of positive integers: (e,n). A message M to be encrypted must be an integer in the range 0 to n-1. The ciphertext C is obtained by encrypting M as follows:

$$C \equiv M^e \pmod{n}.$$

That is, C is the remainder obtained when the e-th power of M is divided by n.

Similarly, a decryption key is a pair of positive integers: (d, n). Here, n is the same as in the encryption key. The message M can be obtained by deciphering the ciphertext C:

$$M \equiv C^d \pmod{n}.$$

Note that the encryption and decryption operations have a common form, which simplifies implementation.

The modulus n is chosen to be the product of two large prime numbers, p and q. (As it happens, large prime numbers are relatively common, and testing a large number for primality is not too difficult (Solovay and Strassen, 1977; Adleman, 1980)). A cryptanalyst could attempt to "break" the RSA method by factoring the modulus n. (Recall that in a public-key cryptosystem the encryption key (e,n) may be public knowledge.) However, factoring large integers seems to be a computationally intractable problem.

The integers e and d are chosen to be multiplicative inverses modulo $lcm(p-1, q-1)$ as described in Rivest, Shamir, and Adleman, 1978; they satisfy the equation

$$e \cdot d \equiv 1 (\bmod\ lcm(p-1, q-1)).$$

Context for the Design of the RSA Chip

In the fall of 1979 Shamir, Adleman, and I decided to try to design a single-chip implementation of the RSA method, to be completed by the end of May 1980. During the fall of 1979 we attended a course in VLSI design taught by Jon Allen at MIT, which used the excellent Mead and Conway text (Mead and Conway, 1980).

Our task was to create a computer file in the Caltech Intermediate Form (CIF) language (Mead and Conway, 1980) which described the desired geometry of the wires and transistors on the chip. This file was transmitted to Xerox PARC at the end of May. At that time PARC personnel collected designs from many designers at universities all over the U.S., and subcontracted to have masks made which merged the many designs onto a few wafer types for economical fabrication. Hewlett-Packard fabricated the chips, which were bonded and packaged at PARC; we received copies of the finished chips in mid-August. At the time of writing this paper, the chips had not yet been tested because our testing hardware was still being built.

We estimate that the project required about 5 man-months of effort. This project was primarily a *programming* project, because we almost exclusively wrote programs in LISP which, when executed, created the desired CIF output file. We wrote about 75 pages of LISP code altogether. The three largest pieces were the code that specified the final placement and interconnect of the modules (14 pages), the description of the ALU (11 pages), and the micro-code assembler and simulator (10 pages).

Overall Architecture of the RSA Chip

It was decided at an early stage to use a bit-slice architecture for the ALU. This decision enabled us to proceed rapidly with the design, while leaving a final determination of the actual chip size until the end.

This chip was designed as a general-purpose big-number processor, with a few extra operations which are helpful for RSA key generation. The chip implements the following operations:

(a) +, −, *, /, remainder.

(b) RSA encryption/decryption (i.e., modular exponentiation).

(c) Generating a large prime number, given a random "seed."

(d) Given several random "seeds," generating a complete RSA key-set.

(e) Greatest common divisor (GCD).

(f) Given a number x, find a small divisor of x, if one exists, using Pollard's "rho" method (Pollard, 1975).

(g) Input or output of big numbers by rotating them through an 8-bit "window" that can be examined or modified.

The chip contains the following subsystems:

(i) 512-bit ALU organized in a bit-slice manner, with eight general-purpose 512-bit registers, up-down shifter logic, and multiplier (carry-save) logic.

(ii) Control logic, including a PLA of 224 72-bit micro-code words, and a stack/counter array for subroutines and looping.

(iii) An array of powerful superbuffers to drive the signals that control the ALU.

(iv) An 8-bit "window" for input-output.

(v) Logic to test each subsystem separately.

(vi) Input-output pads for bonding. The chip has 18 pins:

—V_{DD} and GND

—Φ_1 and Φ_2 for two-phase clocking

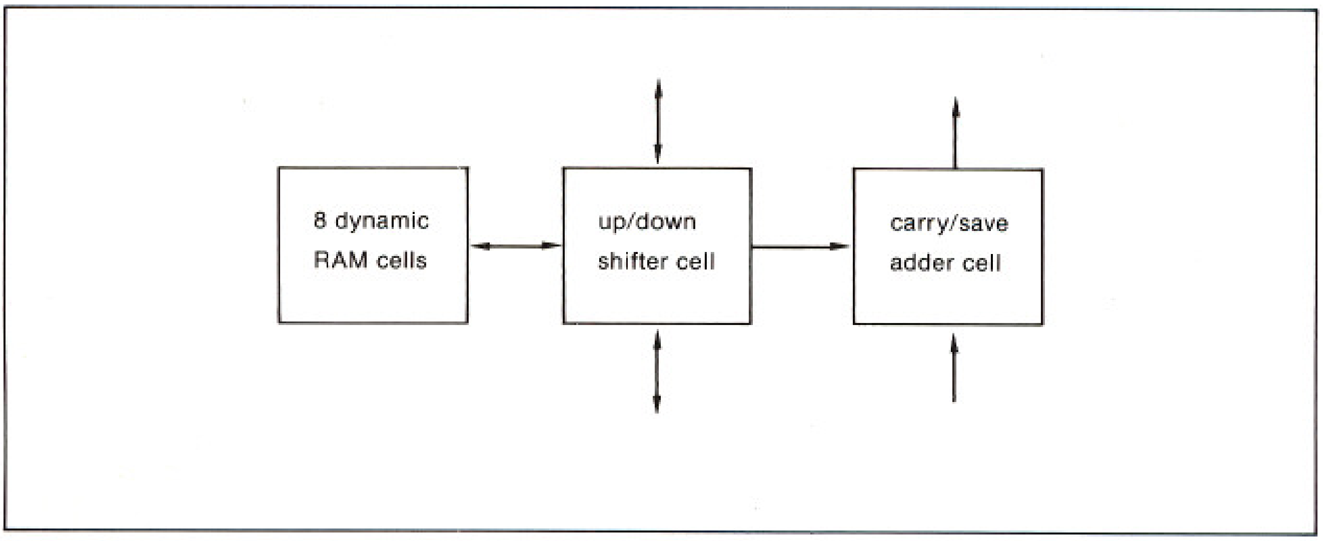


FIGURE 1. Block diagram of the ALU cell.

—10 I/O lines for data or commands

—2 "Address" lines

—one Read/Write line

—one "Done" line (an output signal)

Externally, the chip is configured as a "memory" chip which may be read or written at one of four 8-bit word positions. One such position is the "window" for data I/O. Another position is for receiving "commands" to be executed —such as "add register 1 to register 0" or "encrypt." Such commands cause the Done signal to be raised when they are completed. The other two "memory" positions and two of the data I/O lines are only for debugging.

Internally, the ALU is only capable of performing the operations: $A \cdot B \pm C$, shift-left, shift-right, test least-significant bit. (Note that multiplying two 512-bit numbers yields a 1023-bit product, occupying two registers.) These operations are powerful enough to enable all of the other operations to be implemented using microcode control subroutines. For example, the routine to control the exponentiation can test each exponent bit as needed for the usual "repeated squaring" method of exponentiating to a large power. We sketch the organization of a single ALU cell in Figure 1.

The clock is designed to run at 4MHz. (Since we have not yet been able to test the chip, this estimate is still only an estimate.) The clocking philosophy is a modified two-phase design with some data enables held up during the gap

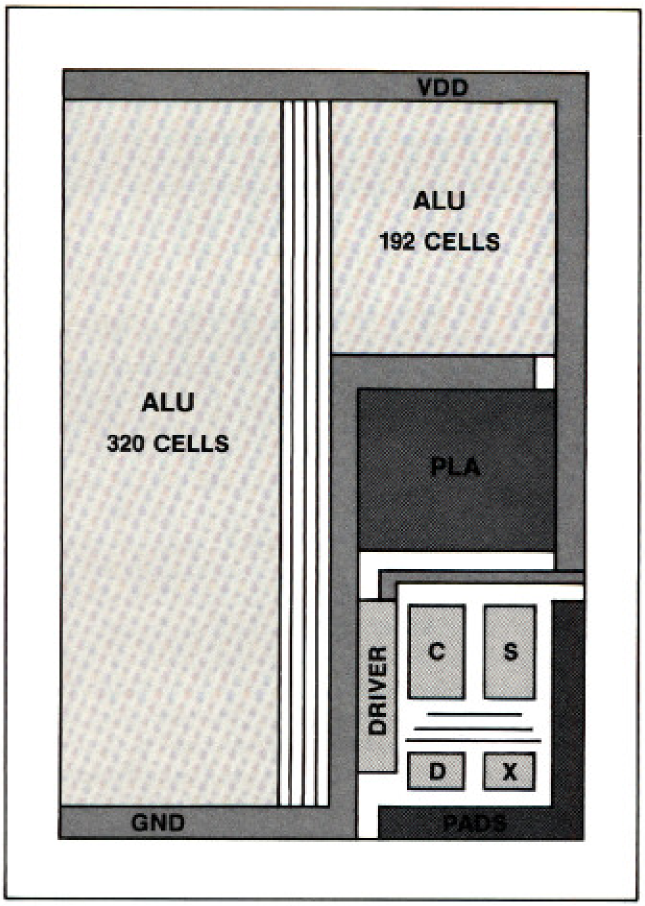


FIGURE 2. Floor plan of the RSA chip.

between Φ_1 and Φ_2, or between Φ_2 and Φ_1, to avoid race conditions between data signals and write pulses. The superbuffer array driving the ALU control lines is on the critical timing path. (Changing the ALU control signals is the most time-consuming operation during a single clock cycle, because there are 512 slice elements to control.)

The stack/counter array contains eight 10-bit words which may be loaded, popped into the program counter, incremented, decremented, or compared to zero. For speed and economy of layout, "counting" uses a period-1023 linear feedback shift register scheme rather than ordinary binary notation.

One microcode instruction is executed every clock cycle. The 72 bits of the instruction word are divided in the following way:

—ALU control: 22 bits

—Stack control: 19 bits

—"True Address": 8 bits

—"False Address": 10 bits

—Branch Control: 2 bits

—Bus Control: 10 bits

—"Done" bit: 1 bit

Although there are only 224 microcode instructions, the "False Address" has 10 bits because it doubles as a data source for loading constants into the stack counter array. At each cycle the address of the next instruction to be executed is selected from the current true or false addresses, or the stack contents. The branch may be conditional upon the low-order bit of a big number or the zero-test of a word in the stack/counter array.

Because many of the data paths in the chip are serial data paths, it was not too difficult to add some logic to improve the testability of the finished chip. We selected eight important data lines and added a "breakpoint cell" in the middle of each one, which allows us to monitor the status of the data on each line, or to "break" the line and feed signals manually into the input end while watching the output data. Each breakpoint cell is individually programmable to be in one of the two modes.

The "floor-plan" of the chip is shown in Figure 2. The left side of the chip contains a block of 320 slices of the ALU, and the upper-right block contains the remaining 192 slices. The central spine of the chip carries control signals to the ALU from the superbuffer driver array at its lower right. The microcode PLA occupies the center right of the chip, and the remaining logic (stack, pads, etc.) occupies the lower right portion. Here *S* denotes the stack, *X* the 8-bit "window," *C* the small bus-control PLA, and *D* some debugging logic.

The entire chip contains about 40,000 actual MOS transistors and uses a little more than one watt of power.

The current design is very cramped. We believe that only modest improvements in fabrication technology will enable the use of different architectural structures which could increase the encryption rate several-fold. We see no intrinsic problems in reaching encryption speeds of 20,000 bits/second with a single-chip implementation within the next few years.

The Design Process

This section briefly describes the steps we took to design the RSA chip.

At an early stage we decided what the basic functionality of an ALU cell should be, and designed the cell. It turned out to be 31λ×365λ in size. (Here λ denotes the Mead-Conway scale factor (Mead and Conway, 1980); we specified λ=2 microns for fabrication.) At this size, we were "just making it"; a larger chip would have had an impractically low yield.

The second step was to write the microcode and debug it with a high-level function simulator. The simulator was a special-purpose LISP program. Concurrently, of course, we designed the functional specifications for the stack, etc.

A special-purpose LISP "chip assembler" was written to enable the components to be described in a high-level manner.

The third step was to enter the ALU cell design into the computer and design the remaining subsystems. A special-purpose LISP "chip assembler" was written to enable these parts to be described in a high-level manner. This assembler allowed parameterized cells, high-level connection commands, and arbitrary LISP code to be used to specify the desired geometries and layouts.

We viewed this portion of the design process as a programming step. Using a high-level language provided a powerful medium for expressing the desired relationships. It was easy to make small changes and let the assembler recalculate the new relative positions of modules and redo the interconnections.

The microcode PLA was then generated by an APL program written by Paul Penfield and Lance Glasser from a bit-matrix input we provided. The PLA design is quite compact and quite fast.

The final layout of the I/O pads and the bus interconnections was then specified; the wiring was routed by hand on large plots and then entered symbolically as symbolic LISP code.

When we believed the design was complete, we used programs written by Clark Baker to perform design-rule and pullup-pulldown ratio checks, and to extract a node-and-switch level circuit from the mask description. Extensive simulation using a program written by Chris Terman helped complete this debugging phase. We found roughly two dozen fatal design errors, but eventually achieved a design that passed all of our tests. This experience underlies our belief that such software design checks are absolutely essential to

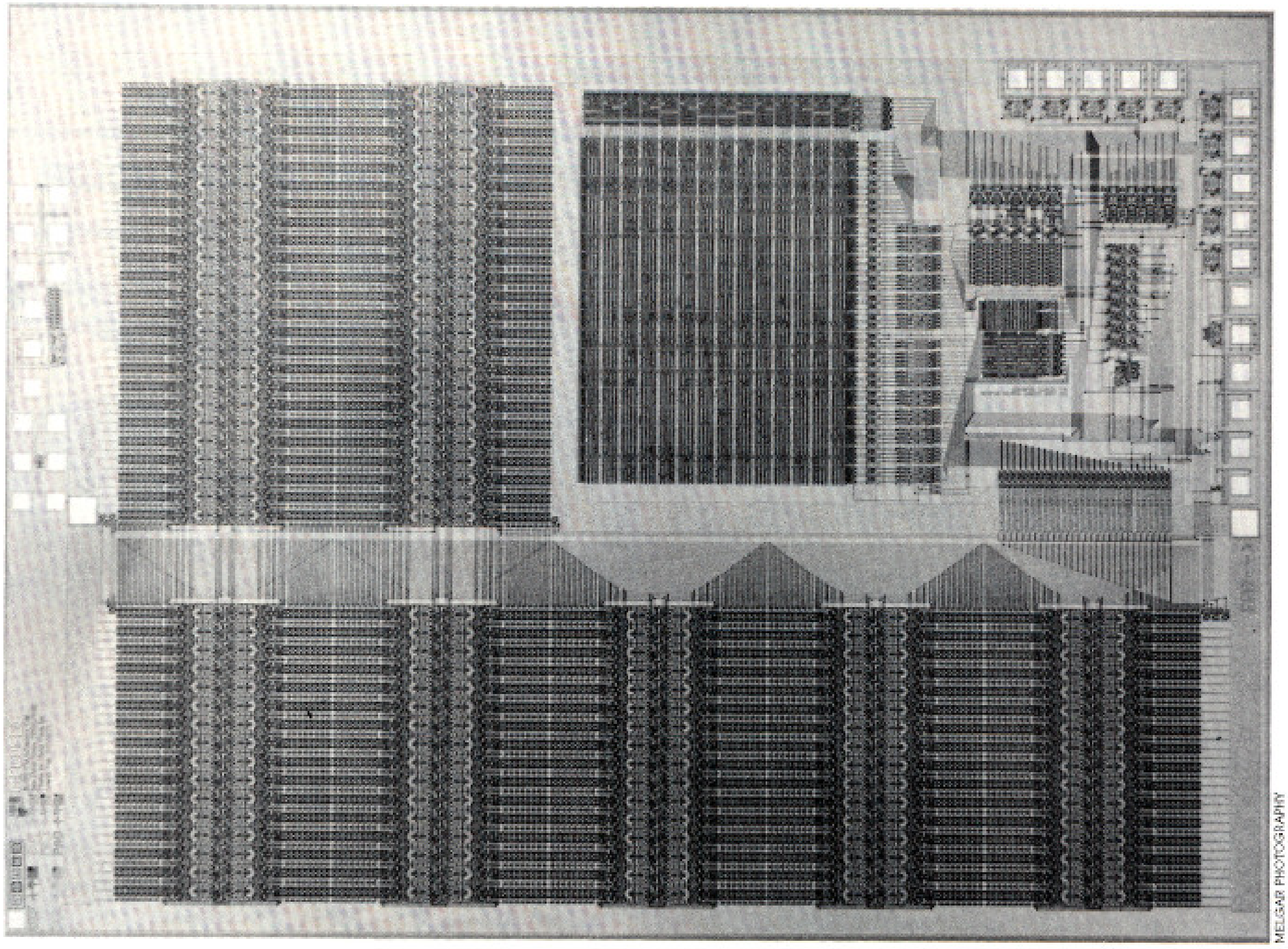


FIGURE 3. The RSA chip contains 40,000 transistors and measures 5.5 mm by 8 mm.

the successful production of VLSI designs.

One useful design strategy was that we really designed two chips: one with a 512-bit ALU and one with a 32-bit ALU. The two designs differ only in the replication factor for the ALU cell and in the constants in the microcode controlling the number of times each loop is executed. Otherwise, wiring is identical. The transistor-level simulation was performed only on the smaller version of the design (which had about 9000 transistors). We presume that the trivial scaling-up operation hasn't introduced errors.

Acknowledgments

We would like to thank the following people for their advice and assistance: Jon Allen, Randy Bryant, Clark Baker, Jim Cherry, Cynthia Cole, Lynn Conway, Lance Glasser, Mark Johnson, Dick Lyon, Paul Penfield, Ted Strollo, Gerry Sussman, Chris Terman, and all of those who have made the MPC (Multi-Project Chips) possible.

The design of the chip architecture was not funded by any grants. The development of the software design-aid tools was funded by grant AFOSR-F49620-80-6-0073.

References

Adleman, L. October 1980. "On Distinguishing Prime Numbers from Composite Numbers." *Proc. 21-st FOCS Conference*, Syracuse, NY.

Blakely, R. and Blakely, G.R. "Security of Number Theoretic Public-Key Cryptosystems Against Random Attack." *Cryptologia*, Part I (October 1978), Part II (January 1979), Part III (April 1979).

Diffie, W. and Hellman, M. November 1976. "New Directions in Cryptography." *IEEE Trans. Inform. Theory* IT-22.

Diffie, W. and Hellman, M. March 1979. "Privacy and Authentication: An Introduction to Cryptography." *Proc. IEEE*, Vol. 67, No. 3.

Gardner, M. August 1977. "Mathematical Games." *Scientific American.*

Kahn, D. Fall 1979. "Cryptology Goes Public." *Foreign Affairs.*

Lempel, A. December 1979. "Cryptology in Transition—A Survey." *Computing Surveys*, Vol. 11, No. 4.

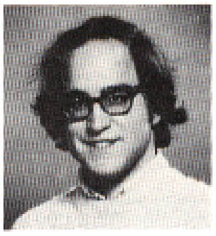
Mead, C. and Conway, L. 1980. *An Introduction to VLSI Systems.* Reading: Addison-Wesley.

Pollard, J.M. 1975. "A Monte Carlo Method for Factorization." *BIT 15.*

Rivest, R.L.; Shamir, A.; and Adleman, L. February 1978. "A Method for Obtaining Digital Signatures and Public-Key Crypto-systems." *CACM*, Vol. 21, No. 2.

Solovay, R. and Strassen, V. March 1977. "A Fast Monte-Carlo Test for Primality. *SIAM J. Computing*, Vol. 6. λ

About the Author



Ronald L. Rivest earned a B.S. in math from Yale in 1969 and a Ph.D. in computer science at Stanford University in 1974. He was a post-doctoral student at IRIA and has been with the M.I.T. Laboratory for Computer Science since 1974. He is currently an Associate Professor in the Computer Science Department.

A UNIQUE SPECTRUM OF DESIGN SERVICES

CUSTOM DESIGN

IC DESIGN TRAINING PROGRAMS

COMPUTER AIDED DESIGN SERVICES

DESIGN/CAD CONSULTING

FOR DETAILS, PLEASE WRITE OR CALL ANTHONY WAINWRIGHT, MANAGER OF ICE DESIGN CENTER

ICE®

INTEGRATED CIRCUIT ENGINEERING CORPORATION
15022 N. 75th Street • Scottsdale, Arizona 85260
Tel: 602-998-9780 • Telex: 165-755 ICE SCOT

DESIGN CENTER