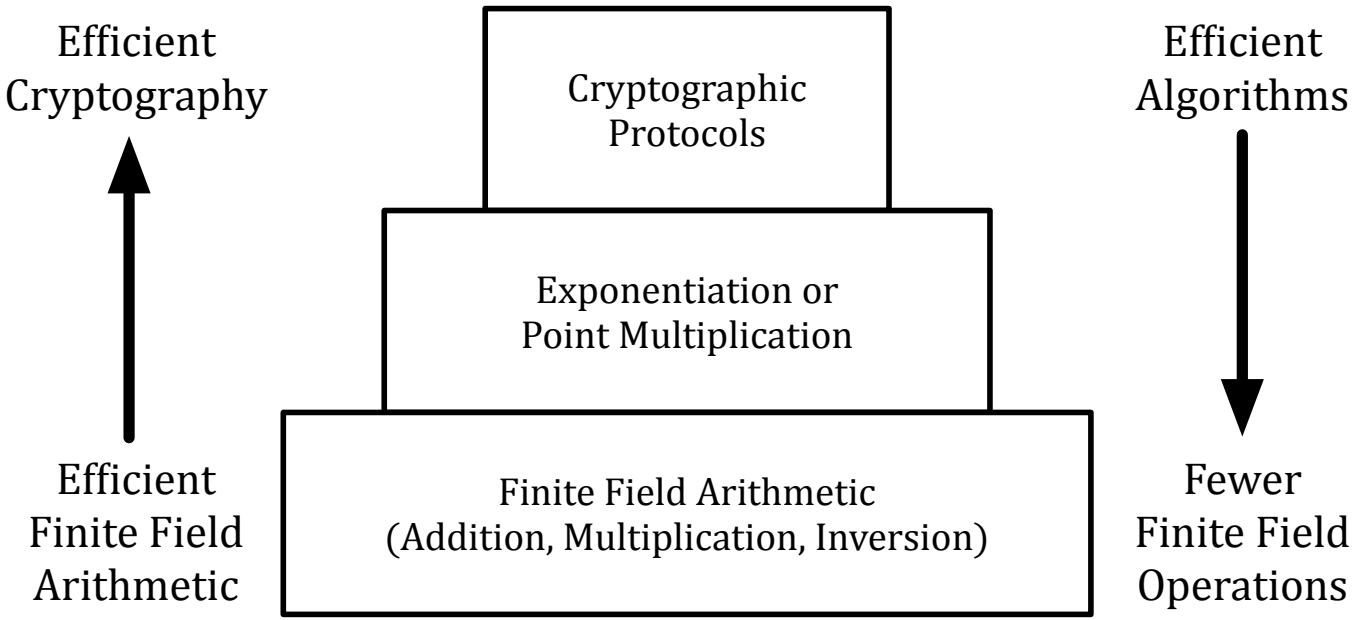
Computational Requirements of PKC



Computational Requirements for PKC

- All public-key cryptographic functions, such as public-key encryption and decryption operations, digital signature generation and verification operations, homomorphic functions, variety of cryptographic protocols, and post-quantum cryptographic functions, are based on the arithmetic of number and polynomial groups, rings and fields
- Mathematical properties of these structures, efficient representations of their elements, and algorithms for arithmetic operations need to be well understood in order to design suitable software and hardware
- This endeavor is in a way similar to the design and implementation of error-detecting and error-correcting codes, however, in cryptography the lengths of the elements are significantly larger

Computational Requirements for RSA

- The RSA public-key cryptographic algorithm is based on the arithmetic of the ring $\mathcal{Z}_n$ where n is the product of two primes p and q
- The fundamental RSA operation is the exponentiation, defined as

$$s \leftarrow m^d \pmod{n}$$

- Operands in RSA are in the range $[0, n)$
- They are represented as k-bit integers, where $k = \log_2 n$
- The operand ranges are usually $k \in [1024, 4096]$

Computational Requirements for RSA

- The computation of $m^d \pmod{n}$ is accomplished by performing multiplication and squaring operations in $\mathcal{Z}_n$, which require additions and subtractions in $\mathcal{Z}_n$

$$\begin{aligned} t &\leftarrow a \cdot b \pmod{n} \\ u &\leftarrow a \cdot a \pmod{n} \\ v &\leftarrow a + b \pmod{n} \\ w &\leftarrow a - b \pmod{n} \end{aligned}$$

- The exponentiation operation is accomplished using addition chains
- There are sophisticated algorithms for modular multiplication and addition, for example, the Montgomery multiplication algorithm, the Karatsuba algorithm, and the spectral methods

Computational Requirements for RSA

- Since $n = p \cdot q$, the RSA operations may use the Chinese Remainder Algorithm, which require computations in $\mathcal{Z}_p$ and $\mathcal{Z}_q$
- For example, the exponentiation $s = m^d \pmod{n}$ is performed as

$$s \leftarrow \text{CRT}(s_p, s_q; p, q) \begin{cases} s_p \leftarrow m^{d_p} \pmod{p} \\ s_q \leftarrow m^{d_q} \pmod{q} \end{cases}$$

- The sizes of d_p and d_q are half of the size of d, which are defined as

$$\begin{aligned} d_p &\leftarrow d \pmod{p-1} \\ d_q &\leftarrow d \pmod{q-1} \end{aligned}$$

- The CRT operation requires modular additions and multiplications, and we also need the multiplicative inverse of p mod q

Computational Requirements for ECC

- Elliptic curves used in cryptography are Weierstrass and Edwards curves over finite fields GF(p) and GF(2^k)
- The points on the curve are pairs of (x, y) where x and y are elements of the finite fields GF(p) or GF(2^k), satisfying the curve equation
- The points on the curve with the neutral element $\mathcal{O}$ and the point addition operation $\oplus$ form an additive group of order n denoted as $\mathcal{E}$
- The fundamental operation of the ECC is the point multiplication operation for a given integer $d \in [0, n)$

$$Q \leftarrow [d]P$$

- The point multiplication operation is accomplished using addition or addition-subtraction chains

Computational Requirements for ECC

- The point multiplication operation $Q \leftarrow [d]P$ is accomplished by performing point addition, doubling, and subtraction operations

$$\begin{aligned} R &\leftarrow P \oplus Q \\ T &\leftarrow P \oplus P \\ U &\leftarrow P \ominus Q \end{aligned}$$

- These point operations in the elliptic curve group $\mathcal{E}$ requires the computation of field additions, subtractions, multiplication, and inversion operations in the finite fields GF(p) or GF(2^k)

$$\begin{aligned} t &\leftarrow a \mp b \\ u &\leftarrow a \cdot b \\ v &\leftarrow a^{-1} \end{aligned}$$

Computational Requirements for DH, ElGamal, and DSA

- Public-key cryptographic algorithms based on the discrete logarithm problem in the multiplicative group $\mathcal{Z}_p$ have their fundamental operation as the exponentiation

$$s \leftarrow g^r \pmod{p}$$

- The ranges of the prime p and the other parameters are usually $k \in [1024, 4096]$, where $k = \log_2 p$
- g is a primitive element mod p
- These algorithms including their ECC variants require deterministic or true random numbers

Computational Requirements for DH, ElGamal, and DSA

- The computation of $g^r \pmod p$ is accomplished by performing multiplication and squaring operations in $\mathcal{Z}_p$, which may require additions in $\mathcal{Z}_p$ or GF(p)

$$\begin{aligned} t &\leftarrow a \cdot b \pmod p \\ u &\leftarrow a \cdot a \pmod p \\ v &\leftarrow a + b \pmod n \end{aligned}$$

- The exponentiation operation is accomplished using addition chains
- There are sophisticated algorithms for modular multiplication and addition operations, for example, the Montgomery multiplication algorithm, the Karatsuba algorithm, and the spectral methods

Computational Requirements of Cryptographic Protocols

- Cryptographic protocols, such as coin flipping, secret sharing, and zero knowledge require a diverse set of fundamental operations
- Protocols based on the number-theoretic concepts require multiplication, squaring, gcd and inverse computation, square roots modulo a composite or prime number, and the CRT computation

$$
\begin{aligned}
n &\leftarrow p \cdot q \\
t &\leftarrow u^2 \pmod{n} \\
u &\leftarrow \gcd(n, a \mp b) \\
v &\leftarrow \sqrt{a} \pmod{p} \\
a &\leftarrow \mathrm{CRT}(a_1, a_2, \ldots, a_k; m_1, m_2, \ldots, m_k) \\
a_i &\leftarrow a \pmod{m_i} \\
y &\leftarrow r_1 \cdot r_2 \pmod{n} \\
y &\leftarrow x^2 \pmod{n}
\end{aligned}
$$

Representations of Polynomials

- The elements of GF(2^k) or various other structures used in post-quantum cryptography use polynomials as their operands
- Often the coefficients of these polynomials are in GF(2), however, they may also be in a ring $\mathcal{Z}_n$
- Polynomials with binary coefficients can be represented as if they are binary integers, however, the arithmetic functions using these operands perform appropriate arithmetic operations
- Elements of GF(2^k) may also be represented in a normal basis, which in fact also uses a binary representation
- The ranges of the lengths of polynomials used in ECC are generally in $[163, 571]$ however some post-quantum algorithms use long operands

Representations of Integers

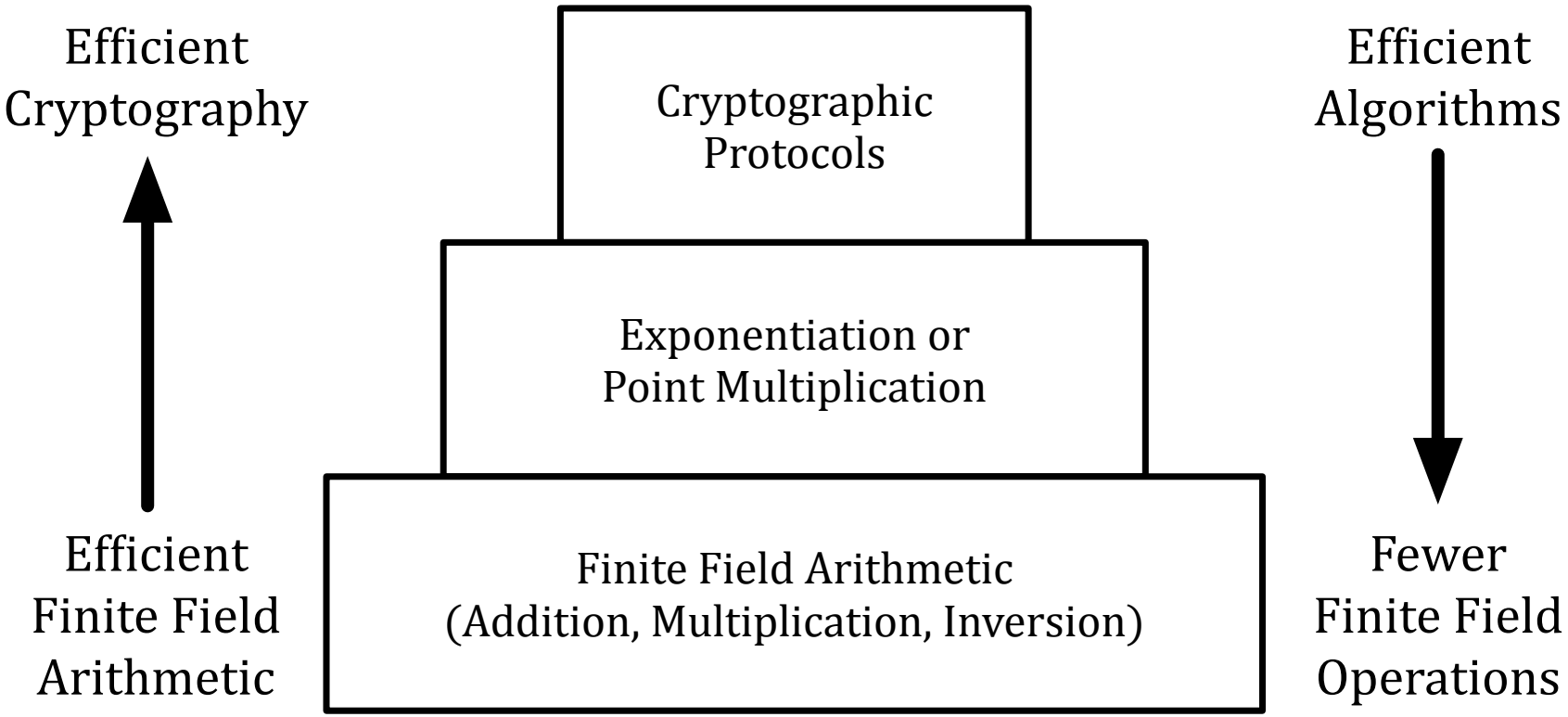
- Integer operands in PKC range from 160 to 4096 bits
- Integers as elements of $\mathcal{Z}_n$ and GF(p) are assumed to be k bits in binary with $k \in [160, 4096]$
- In software implementations, the word size w of the computer, and similarly, in hardware implementations, the width w of the data path should be taken into account
- An **efficient** way to represent large integers is to break the k-bit number ($k \geq 160$) into w-bit words
- For example, a 256-bit integer a can be represented using an 8-word vector such that each vector element A_i is $w = 32$ bits

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0

Interplay of Finite Fields and PKC

- All public-key cryptographic algorithms are based on computationally intensive finite field and ring arithmetic
- The primary operation for the PKC algorithms is the exponentiation or point multiplication operation in the group built upon the finite field or ring arithmetic
- Cryptographic protocols aim to minimize field and ring operations for efficiency, without sacrificing security
- On the other hand, efficient finite field and ring arithmetic leads to efficient public-key cryptography
- This is an interdisciplinary research area, involving
  - Mathematics
  - Computer science
  - Electrical engineering

PKC Computational Pyramid



PKC ALU Design

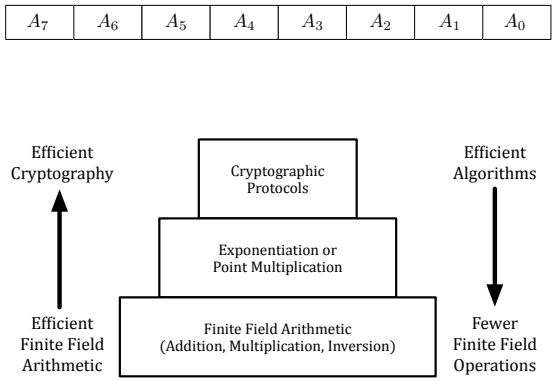
- The basic arithmetic operations used in PKC are addition, subtraction and multiplication operations
- These basic functions can be designed into an ALU so that they are available for use in various cryptographic algorithms and protocols
- By designing a data path and an instruction set architecture, we can create a cryptographic processor or co-processor on a given computational platform
- More complicated functions such as modular multiplication, elliptic curve point addition, modular exponentiation, and elliptic curve point multiplications can use these basic functions repeatedly with the help of finite state machines

PKC ALU Design

- The complexity of the PKC ALU design is determined by the **variety of the basic functions** and the **width of the data path**
- We need to build a computational **pyramid**

The processor architecture provides a constant width data path which is the building block of longer operands

The basic functions are the computational elements of more complicated functions and cryptographic algorithms



- A design-at-once approach or short-cutting almost never works

First RSA Chip

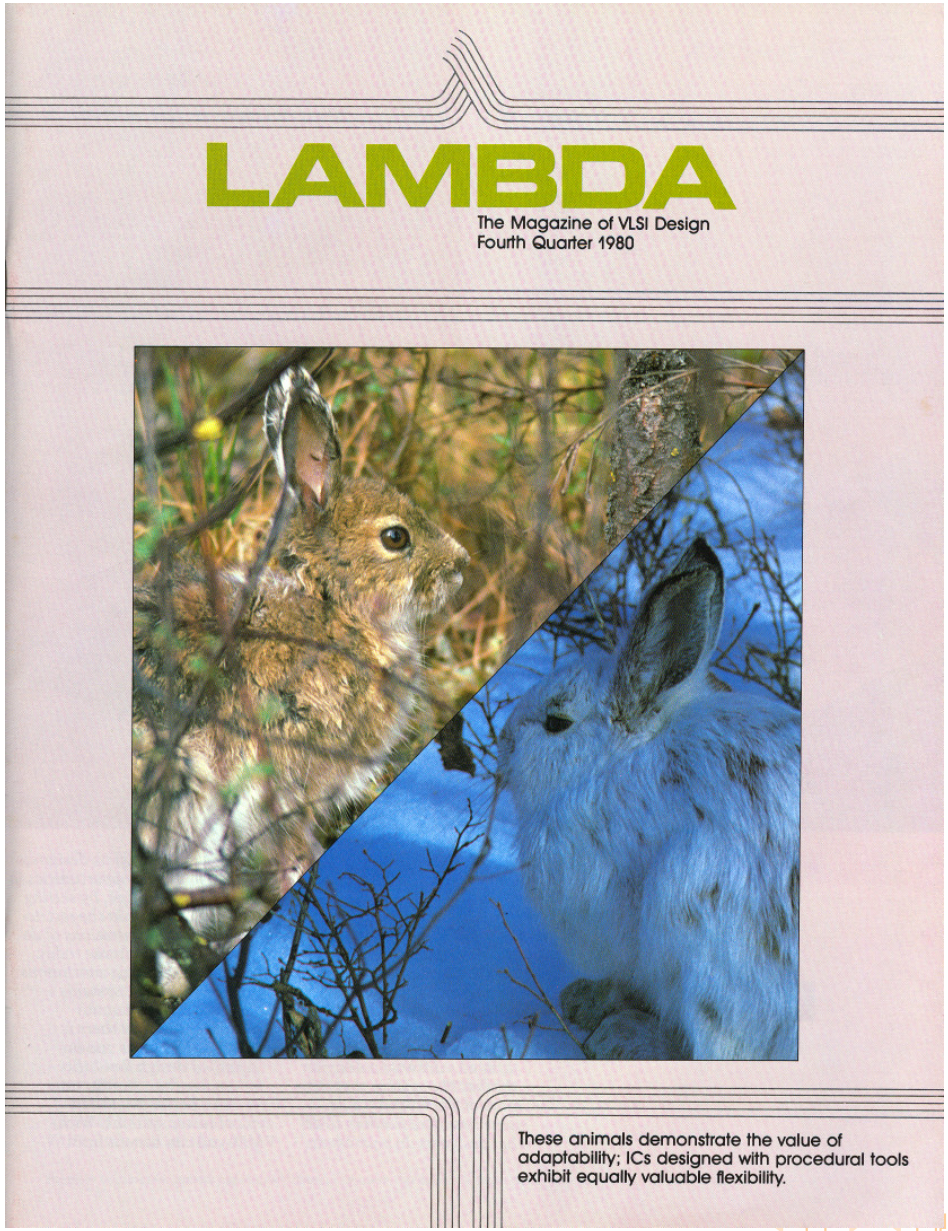
- R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, vol. 21, pages 120-126, 1978.

First RSA Chip

- R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, vol. 21, pages 120-126, 1978.

- R. L. Rivest. A Description of a Single-Chip Implementation of the RSA Cipher. *Lambda*, vol. 1, no. 3, pages 14-18, Fourth Quarter 1980.

First RSA Chip



First RSA Chip

Articles

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

Ronald L. Rivest, *Massachusetts Institute of Technology*

This public-key encryption/decryption chip features a 512-bit ALU.

First RSA Chip

Articles

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

Ronald L. Rivest, *Massachusetts Institute of Technology*

This public-key encryption/decryption chip features a 512-bit ALU.

First RSA Chip

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

First RSA Chip

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).

First RSA Chip

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).

First RSA Chip

A SHORT REPORT ON THE RSA CHIP

Ronald L. Rivest

MIT Laboratory for Computer Science
Cambridge, Mass. 02139

The nMOS "RSA chip" described in our article [1] was initially fabricated by Hewlett-Packard. Testing revealed that while the control portion of the chip worked correctly, the arithmetic section suffered from transient errors and was usually too unreliable to complete a full encryption. We tested a number of chips and found the same problem, enough to convonce us that the cause was probably a design error and not a fabrication problem.

- R. L. Rivest. A short report on the RSA Chip. *Crypto*, pages 327-327, 1982.

First RSA Chip

RSA Chips (Past/Present/Future)*

(Extended abstract)

Ronald L. Rivest
MIT Laboratory for Computer Science
Cambridge, Mass. 02139

Brief Abstract We review the issues involved in building a special-purpose chip for performing RSA encryption/decryption, and review a few of the current implementation efforts.

- R. L. Rivest. RSA Chips (Past/Present/Future). *Eurocrypt*, pages 159-165, LNCS Nr. 209, 1984.

First RSA Chip

IV.A. The "first" RSA chip

This chip was designed by Rivest, Shamir, and Adleman, and is described in [Ri80].

It was a single-chip nMOS design; using 4-micron design rules, the chip occupied 42 mm^2. It contained a 512-bit ALU in bit-slice design with eight 512-bit registers for storage of intermediate results, carry-save adder logic, and up-down shifter logic. The 224-word microcode ROM contained control routines for encryption, decryption, finding large primes, gcd, etc. It used a 5V supply, and drew approximately 1 watt of power. It contained approximately 40,000 transistors. It communicated with a host microprocessor using an 8-bit I/O port. The encryption rate was designed to be slightly in excess of 1200 bits/second. Due to an as yet undiagnosed error in the memory cell design, this chip never worked reliably.

- R. L. Rivest. RSA Chips (Past/Present/Future). *Eurocrypt*, pages 159-165, LNCS Nr. 209, 1984.

First RSA Chip

Evolution of Intel Microprocessors: 1971 to 2007

Family	Trade Name (Code Name for Future Chips)	Clock Frequency in MegaHertz**	Millions of Instructions per Second	Date of Introduction	Number of Transistors	Design Rule (Pixel Size)	Address Bus Bits
80986	Projected Roadmap	24,000.0 MHz	+125,000. MIPS	2007	1 billion	0.045 micron	64 bit
80886	(Northwood)	3,000.0 MHz	TBA	2003	TBA	0.13 micron	64 bit
80886	(Madison)	TBA	TBA	2003	TBA	0.13 micron	64 bit
80886	(Deerfield)***	TBA	TBA	2002Q2	TBA	0.13 micron	64 bit
80886	(McKinley)	1,000.0 MHz	TBA	2002Q1	TBA	0.18 micron	64 bit
80786	Itanium (Merced)	800.0 MHz	+2,500.00 MIPS	May 29, 2001	30 / 300 M	0.18 micron	64 bit
80686	Pentium 4	1,500.0 MHz	*1,500.00 MIPS	November 20, 2000	42 million	0.18 micron	32 bit
80686	Pentium III	1,000.0 MHz	*1,000.00 MIPS	March 1, 2000	28.1 million	0.18 micron	32 bit
80686	P III Xeon	733.0 MHz	*733.00 MIPS	October 25, 1999	28.1 million	0.18 micron	32 bit
80686	Mobile P II	400.0 MHz	*400.00 MIPS	June 14, 1999	27.4 million	0.18 micron	32 bit
80686	P III Xeon	550.0 MHz	*550.00 MIPS	March 17, 1999	9.5 million	0.25 micron	32 bit
80686	Pentium III	500.0 MHz	*500.00 MIPS	February 26, 1999	9.5 million	0.25 micron	32 bit
80686	P II Xeon	400.0 MHz	*400.00 MIPS	June 29, 1998	7.5 million	0.25 micron	32 bit
80686	Pentium II	333.0 MHz	*333.00 MIPS	January 26, 1998	7.5 million	0.25 micron	32 bit
80686	Pentium II	300.0 MHz	*300.00 MIPS	May 7, 1997	7.5 million	0.35 micron	32 bit
80586	Pentium Pro	200.0 MHz	*200.00 MIPS	November 1, 1995	5.5 million	0.35 micron	32 bit
90586	Pentium	133.0 MHz	*133.00 MIPS	June 1995	3.3 million	0.35 micron	32 bit
80586	Pentium	90.0 MHz	*90.00 MIPS	March 7, 1994	3.2 million	0.60 micron	32 bit
80586	Pentium	60.0 MHz	*60.00 MIPS	March 22, 1993	3.1 million	0.80 micron	32 bit
80486	80486 DX2	50.0 MHz	*50.00 MIPS	March 3, 1992	1.2 million	0.80 micron	32 bit
80486	486 DX	25.0 MHz	20.00 MIPS	April 10, 1989	1.2 million	1.00 micron	32 bit
80386	386 DX	16.0 MHz	5.00 MIPS	October 17, 1985	275,000	1.50 micron	16 bit
80286	80286	6.0 MHz	0.90 MIPS	February 1982	134,000	1.50 micron	16 bit
8086	8086	5.0 MHz	0.33 MIPS	June 8, 1978	29,000	3.00 micron	16 bit
8080	8080	2.0 MHz	0.64 MIPS	April 1974	6,000	6.00 micron	8 bit
8008	8008	.2 MHz	0.06 MIPS	April 1972	3,500	10.00 micron	8 bit
4004	4004	.1 MHz	0.06 MIPS	November 15, 1971	2,300	10.00 micron	4 bit

Timeline of PKC Hardware Design

- **Naive algorithms**, 1978-1985
- Classical Montgomery algorithm, 1985
- Fast exponentiation and multiplication, 1990
- Advanced Montgomery algorithms, 1996
- Montgomery algorithm in $GF(2^k)$, 1998
- Scalable arithmetic, 1999
- Dual-field arithmetic, 2000
- RNS arithmetic, 2000
- Spectral arithmetic, 2006