

Accelerating Cryptography in Hardware

- Public Key
- Random Number Generation
- Symmetric Key



Mark Birman
Hi/fn, www.hifn.com
2105 Hamilton Ave, Suite 230
San Jose, CA 95125
August, 1998

Outline

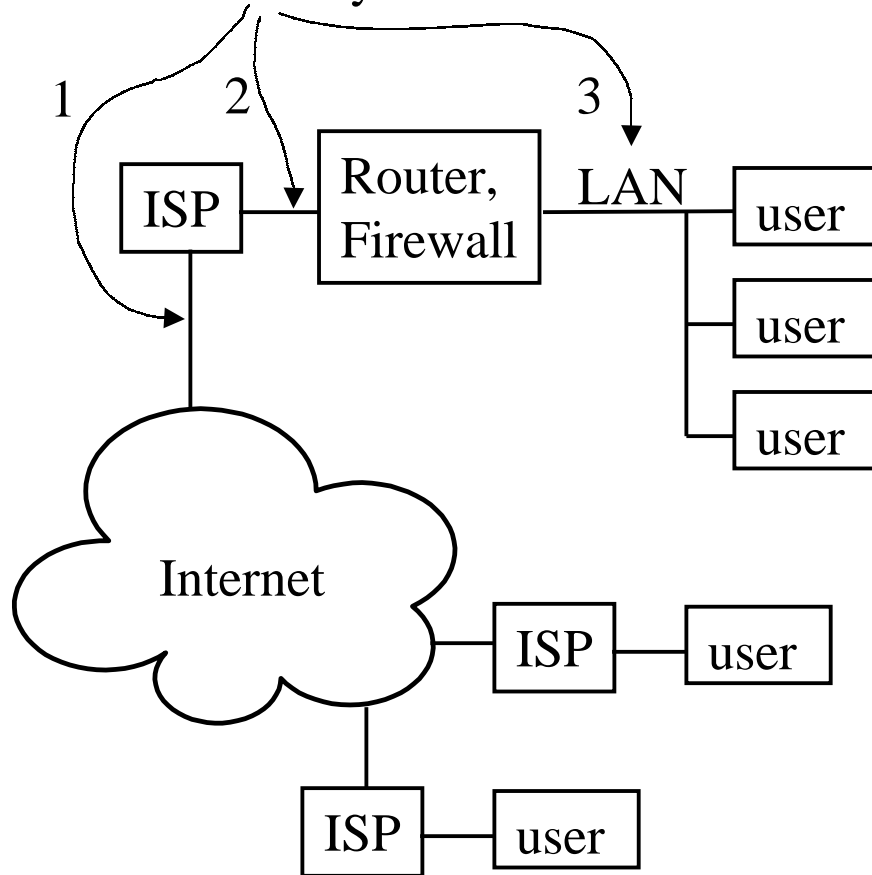
- Overview
 - Why Security?
 - Security Boundaries in Networking
 - Typical Hardware System
- Public Key
 - Technology
 - Computational Requirements
 - Math Properties
 - Math Example
 - Accelerator Block Diagram
 - Instruction Format
 - Instruction Set
 - Public Key Engine Block Diagram
 - Public Key Engine Operation
 - Alternate Public Key Engine
- Random Number Generation
 - Overview
 - Block Diagram
 - Operation
- Symmetric Key
 - Algorithms
 - Accelerator Block Diagram
 - Glossary of Abbreviations
 - Accelerator Operation
- Protocols Supported by Hi/fn Hardware
- Summary

Why security?

- Everybody is getting connected without regard to physical/wiring locality: WAN or LAN; being “neighbors” on the same physical wire, however, is not secure
- Deregulation of the connectivity infrastructure and proliferation of Internet attracts business (vs. using dedicated leased lines), as long as information can be exchanged securely
- On-line “shopping” requires confidentiality of payment/account information and order information, as well as reliable authentication of the parties involved in a transaction.
- Recent standards’ work supporting security deployment
 - Internet Engineering Task Force: IPSEC (IP Security), IPPCP (IP Payload Compression Protocol); support for key exchange, encryption, authentication, and compression
 - SETCo: SET™ (Secure Electronic Transaction) standard; SETCo is started by VISA and MasterCard

Security Boundaries in Networking

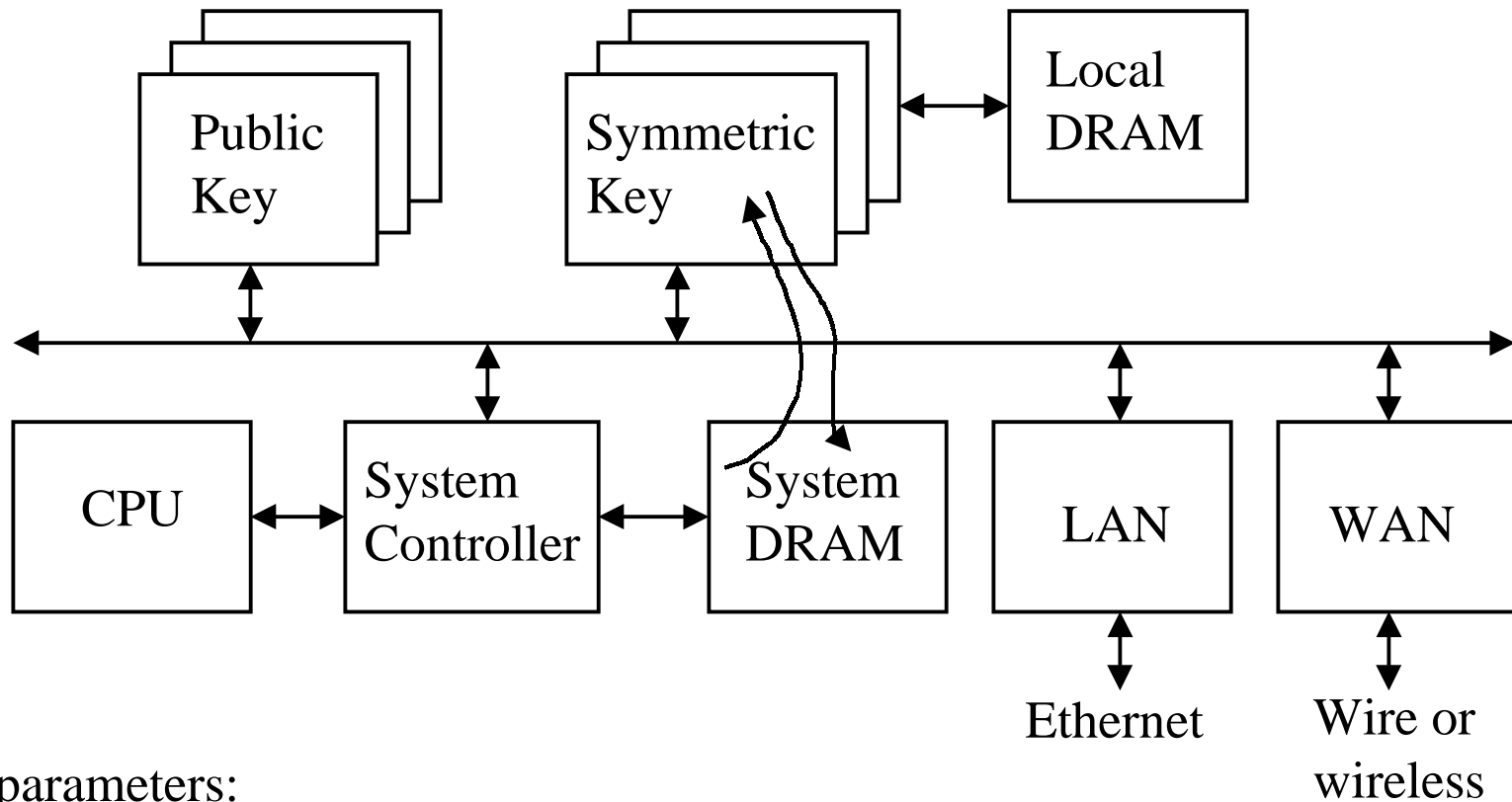
Possible security boundaries



Need to compress before encrypting, since encrypted data does not compress at all!

- Properties of WAN links:
 - point-to-point stateful protocols
 - not as fast as LAN for majority of connections
 - security is essential due to connectivity via Internet
 - compression is widely deployed
- Properties of LAN connections:
 - stateless protocols
 - mostly IP
 - approaching Gbits/sec
 - locality is assumed to be more private, so security is not widely deployed yet
 - compression is not deployed yet

Typical Hardware System

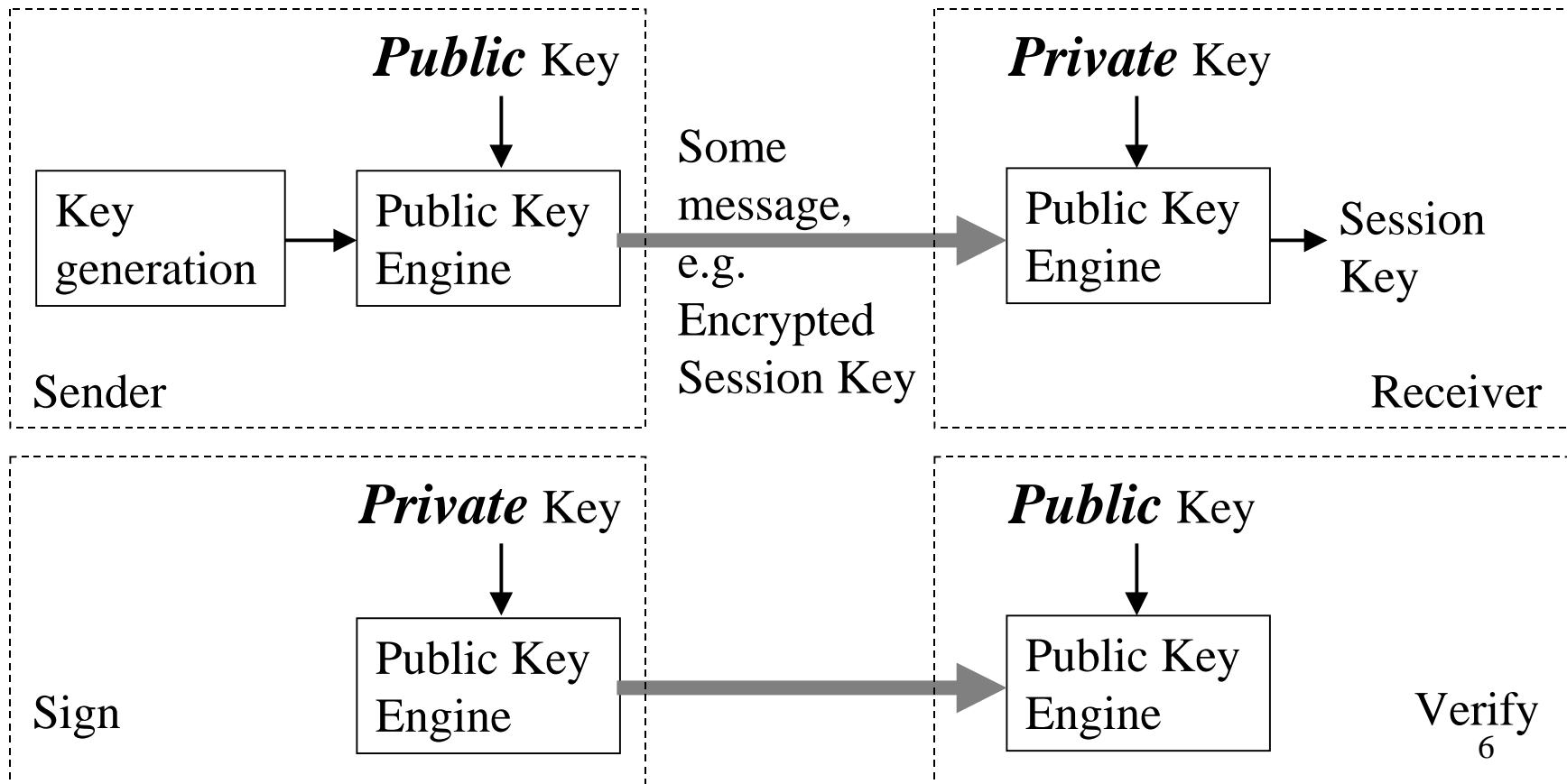


System parameters:

- All-in-one vs. separate public key and symmetric key silicon/performance scalability
- DMA capabilities and bus bandwidth
- Dedicated vs. shared RAM

Public Key Technology

- Asymmetric: computationally easy to encrypt but hard to decrypt; hard to sign but easy to verify the signature
- Key management and trusted certification authority infrastructure problems are complex



Public Key Computational Requirements

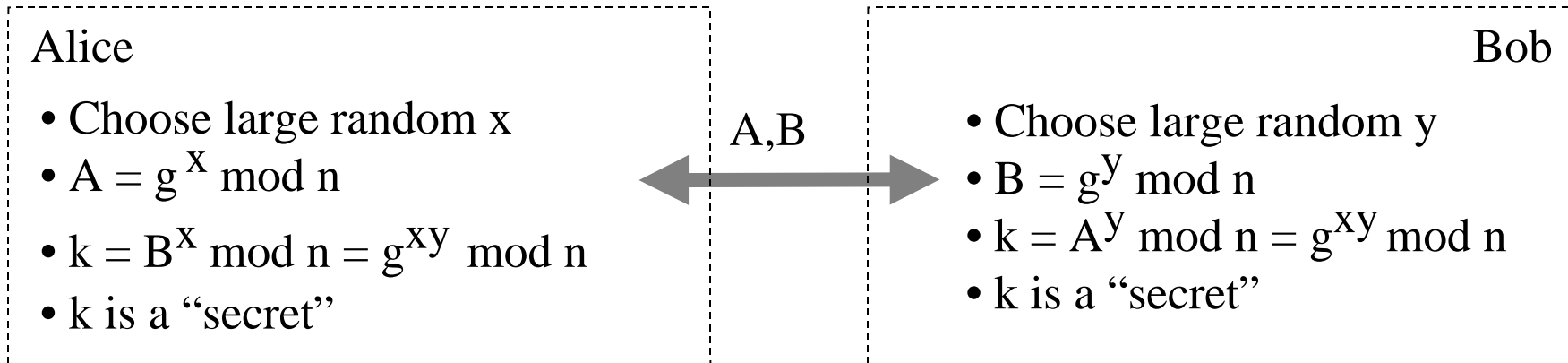
- Modular arithmetic on 1024-bit or larger (e.g. 2048-bit) integers, with the most time consuming operation being:
 - $rd = ra^{re} \bmod rm$, where rd, ra, re, rm are 1024-bit numbers
- 100,000's cycles to compute a private key operation, e.g.
 - 1024-bit and 2048-bit private key RSA operations:
 - 400MHz PentiumII: ~25 msec (40/sec; 2048-bit: 5/sec)
 - 100MHz Hi/fn: 5 msec (200/sec; 2048-bit: 50/sec)
- “Monday morning” phenomenon (everybody calls in on Monday morning) requires 100's computations/sec even for a fairly small communication concentrator, like several 1.5Mbit/sec T1 lines.
- Secure Electronic Transaction servers and higher bandwidth communication concentrators require 1000's computations/sec.

Public Key Math Properties

- Modular arithmetic:
 - $(a * b) \bmod n = (a \bmod n) * (b \bmod n) \bmod n$
 - $(a + b) \bmod n = (a \bmod n) + (b \bmod n) \bmod n$
 - $a^b \bmod n$ can be decomposed into a number of modular multiplies
- Generating primes (testing for primality) and then multiplying them to form a large number is “easy”, but factoring a number is “hard”
- Computing $(a^b \bmod n)$ is “easy”, but finding a discrete logarithm is “hard”
- “Hard” means a $O(c^{f(n)})$, where $f(n)$ is greater than a constant, but less than linear

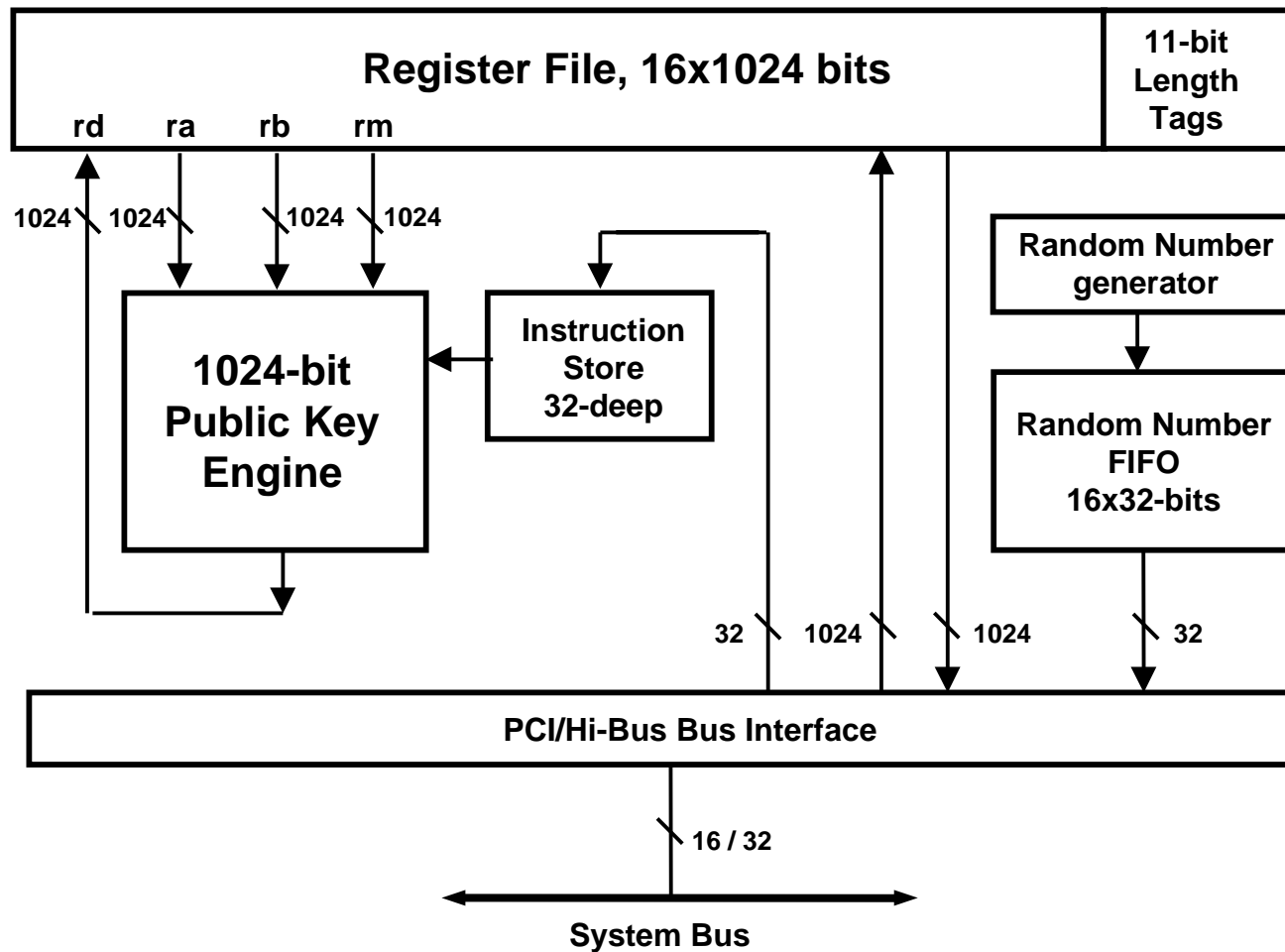
Public Key Math Example (Diffie-Hellman Key Exchange)

- n and g are large known primes
- n and g are publicly known



- only A and B are transmitted
- public knowledge of A and B does not help in determining the secret key, since x and y require computing discrete logarithm

Public Key Accelerator



3.3V, >100MHz, 0.35um, <1.5Watt

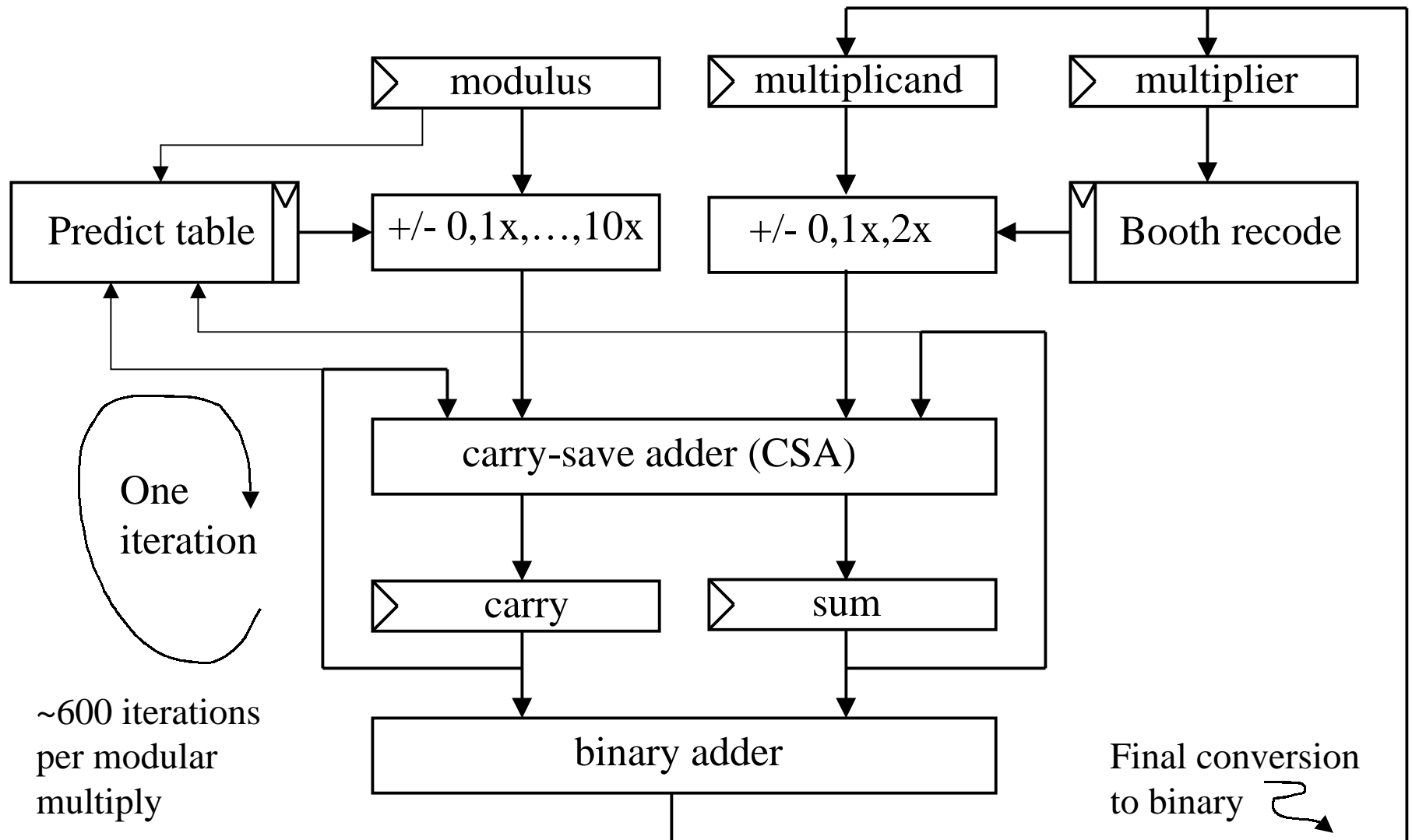
Public Key Accelerator Instruction Format

31 30		26 25		21 20		16 15		11 10		6 5		0							
done	op			rd			ra			rb			rm			reserved			
1		5			5			5			5			5			6		
Bit Field				Description															
Done				Indicates the end of the microprogram. This bit is set on the last instruction of the program. The microprogram terminates after the instruction is executed.															
Op				Instruction opcode.															
ra				Register number containing operand A.															
rb				Register number containing operand B.															
rd				Register number in which to write result															
rm				Register number containing modulus.															
Reserved				Reserved. This field must be written as 0b000000.															

Public Key Accelerator Instruction Set

Name	Opcode	Function	Mnemonic
Modular Exponentiation	0x00	$rd = ra^{re} \bmod rm$	modexp rd ra re rm
Modular Multiplication	0x01	$rd = (ra \times rb) \bmod rm$	modmult rd ra rb rm
Modular Reduction	0x02	$rd = ra \bmod rm$	modred rd ra rm
Modular Addition	0x03	$rd = (ra + rb) \bmod rm$	modadd rd ra rb rm
Modular Subtraction	0x04	$rd = (ra - rb) \bmod rm$	modsub rd ra rb rm
Addition	0x05	$c, rd = ra + rb$	add rd ra rb
Subtraction	0x06	$c, rd = ra - rb$	sub rd ra rb
Addition with carry	0x07	$c, rd = ra + rb + c$	addc rd ra rb
Subtraction with borrow	0x08	$c, rd = ra - rb - c$	subb rd ra rb
Straight Multiplication	0x09	$ru, rl = ra \times rb$	mult ru rl ra rb
Shift Right	0x0A	$rd = ra \gg len$	shr rd ra len
Shift Left	0x0B	$rd = ra \ll len$	shl rd ra len
Increment	0x0C	$c, rd = ra + 1$	inc rd ra
Decrement	0x0D	$c, rd = ra - 1$	dec rd ra
Set Length Tag	0x0E	$rd:tag = len$	settag rd len

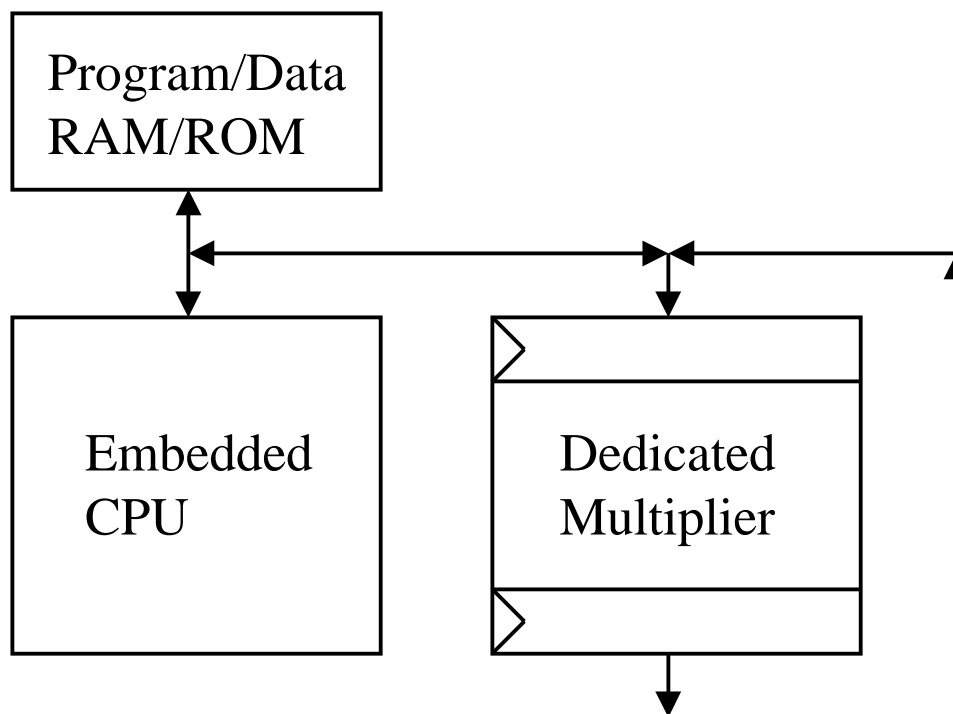
1024-bit Public Key Engine



1024-bit Public Key Engine Operation

- 2-bit per cycle Booth multiplication
- 2-bit per cycle modular reduction (similar to iterative division)
- Both multiplication and reduction steps happen simultaneously, so Carry and Sum registers serve as a redundant representation of both partial product and partial remainder
- In one iteration:
 - Use Booth algorithm to choose $\pm 0, 1, 2$ times multiplicand to be added to the partial product on the next cycle
 - Predict, based on the upper bits of the modulus and the upper bits of the partial remainder, $\pm 0, 1, \dots, 10$ times modulus to be subtracted from the partial remainder on the next cycle
 - Simultaneously add multiplicand multiple to and subtract modulus multiple from the partial product/remainder
- In the end:
 - convert to binary
- Exponentiation consists of a series of multiplies

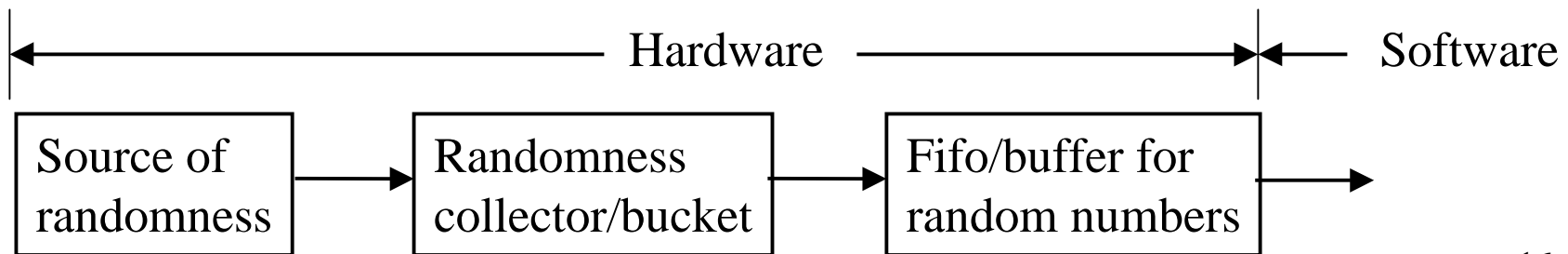
Alternate Public Key Engine Architecture (not implemented)



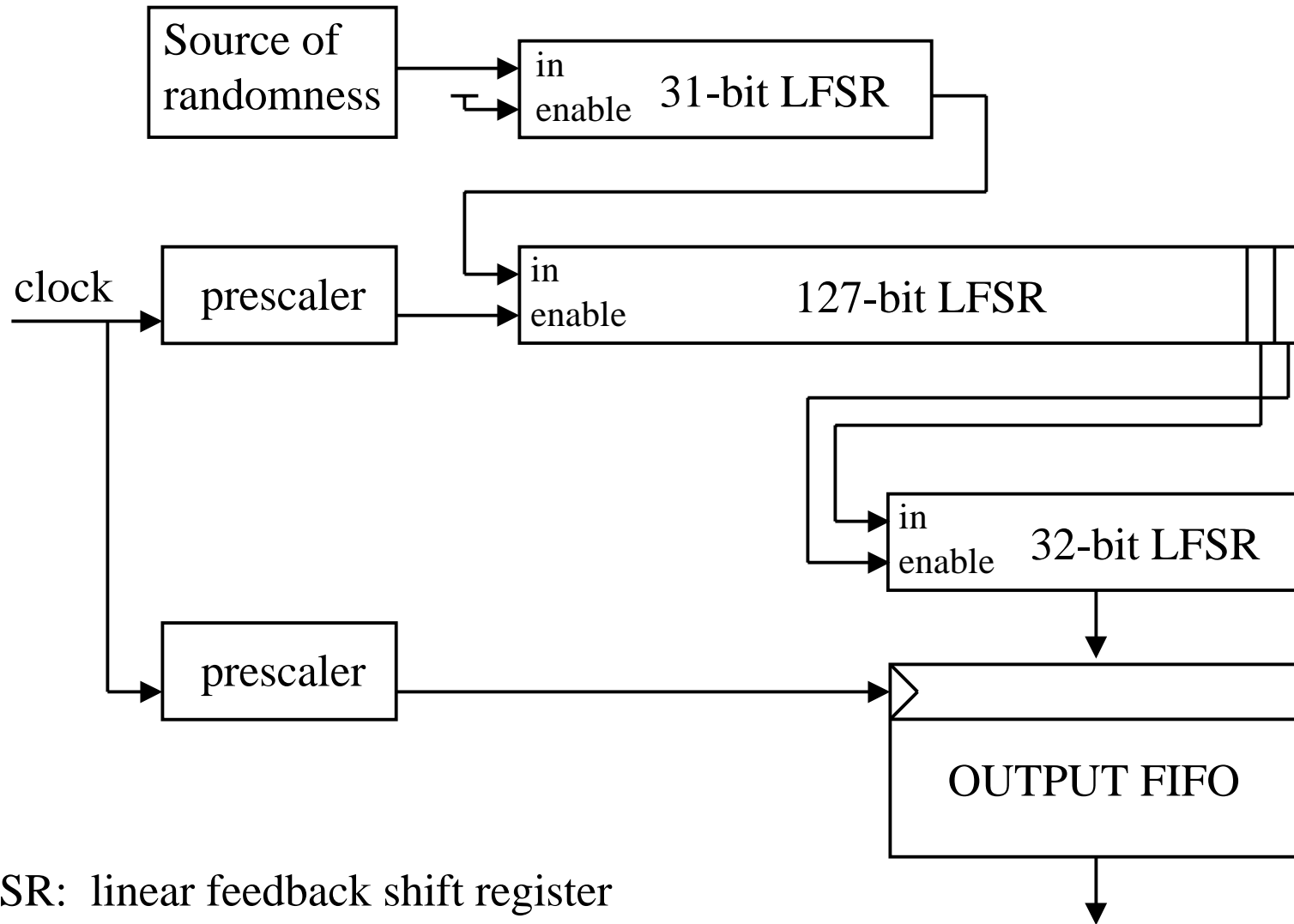
- More flexible
- Less scalable
- Harder to use
- Performance depends on the size and speed of the multiplier, so slower
- Comparable or larger area, depending on the multiplier and the CPU
- Montgomery multiplication algorithm, rather than iterative methods

Random Number Generation

- Random numbers are needed for key generation, for both public key and symmetric key algorithms
- There is no reliable way to generate randomness without hardware
- Randomness can be generated via a noise source of some kind
- Enough randomness has to be collected over a period of time to produce truly random numbers
- One-way hash functions are used to “collect” randomness (one-way means that it’s computationally not feasible to find an inverse)
- Random number generator performance has to match public key hardware performance
- Random keying material needs to be stored securely



Random Number Generator



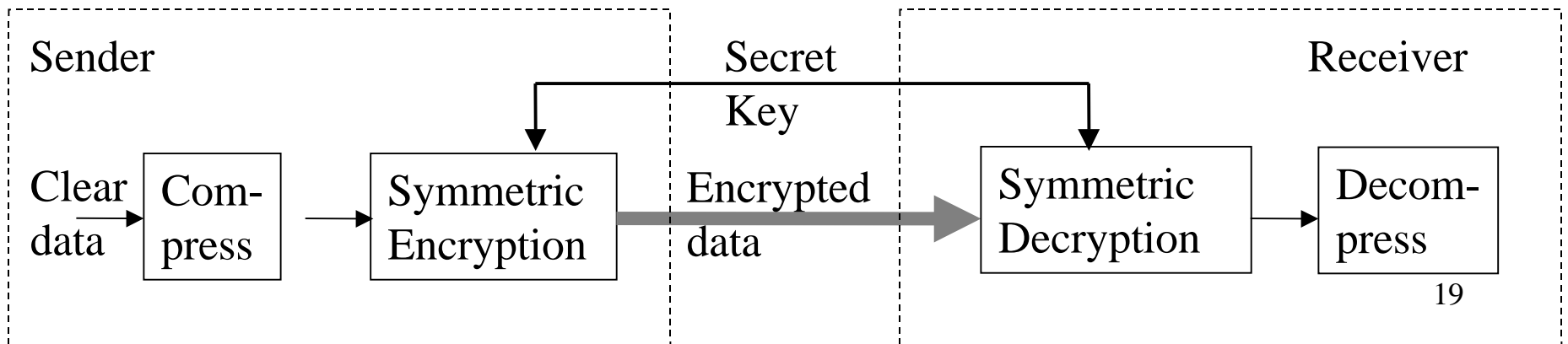
LFSR: linear feedback shift register

Random Number Generator Operation

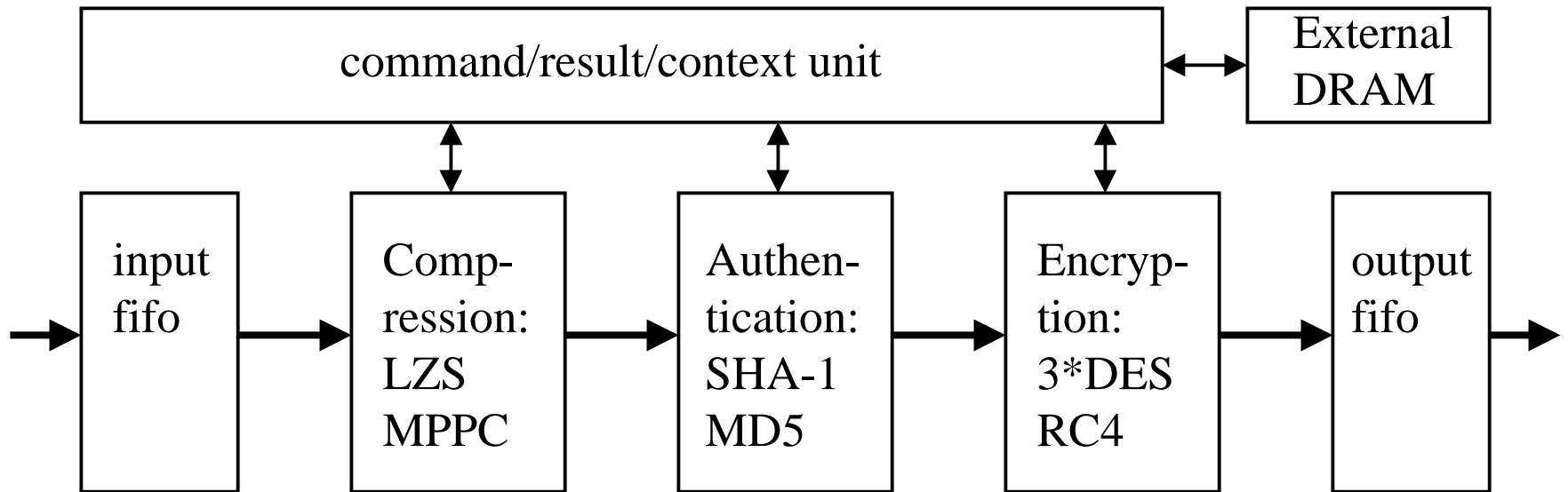
- Use “relatively prime” asynchronous oscillators as a source of randomness; the oscillators are not in any way related to processor clock
- XOR outputs of the oscillators together and synchronize the output: this becomes the input to the 31-bit LFSR; this LFSR is clocked on each clock
- 31-bit LFSR is “holding” randomness from the oscillators
- Every so often (between 2 to 16K cycles), shift in the output from the 31-bit LFSR to the 127-bit LFSR
- Every other clock, and if the least significant bit of the 127-bit LFSR is set, shift in bit-1 of the 127-bit LFSR into the 32-bit output LFSR
- Every so often (between 512 and 1024 cycles) output LFSR is placed into the output FIFO: a new random number is ready
- LFSR lengths, polynomials, and prescaler time delays are selected so that it's computationally infeasible to find any correlation between numbers
- Once the initial 31-bits of randomness is collected, the circuit would work even if no more randomness is supplied

Symmetric Key Algorithms

- Fast in hardware: approaching Gbit speeds!
- Symmetric: same key used by sender and receiver
- Hard to use on its own in communications since may not be able to “share a common secret” in advance of communication
- Need public key algorithms to exchange the secret key (slow...); then use symmetric key encryption for the “payload” (fast!!!)
- Every byte of payload is processed, so it’s convenient to pipeline with other payload processing: compression and authentication
- 3*DES (Triple Data Encryption Standard) is slow in software:
400MHz PentiumII: ~25Mbit/sec; 100MHz Hi/fn: 250Mbit/sec



Symmetric Key Accelerator



- Flexible pipeline ordering
- WAN: T3 (90Mbits/sec), 64-byte packets, stateful, multi-protocol
- LAN: > 200Mbit/sec, 64-byte packets, stateless, single protocol
- If compression cannot compress, pass the data unchanged
- 3.3V, >100MHz, 0.35um, <1.5W

Glossary of Abbreviations

- Compression
 - LZS: Lempel-Ziv-Stac Compression, patented by Hi/fn; hardware-friendly Lempel-Ziv type of compression
 - MPPC: Microsoft Point-to-Point Compression; based on Hi/fn technology; similar to LZS but with a different encoding format
- Authentication
 - MD5: Message Digest; one-way hashing function
 - SHA-1: Secure Hashing Algorithm; one-way hashing function
- Symmetric Encryption
 - DES: US Government Data Encryption Standard; a block-cipher algorithm
 - 3*DES (Triple-DES): DES repeated 3 times (stronger encryption)
 - RC4: Rivest Cipher; stream-cipher; designed by Ron Rivest for RSA Data Security, Inc. Used in Microsoft products

Symmetric Key Accelerator Operation

- Scatter-gather DMA mastering circuit brings in IP packets from system memory and feeds them into the source FIFO
- Each block of the pipeline skips over appropriate header and trailer bytes
- Each block then applies the appropriate algorithm to the payload of the packet, as the payload bytes move through the pipeline
- Scatter-gather DMA mastering circuit places processed packets into system memory
- Each packet of the same communication session is identified by a unique session number, so all packet parameters (e.g. which algorithm to use, header/trailer byte counts, and so on) are selected automatically by the chip
- Stages in the pipeline are automatically reordered based on the algorithm applied to the packet. For example, compression would precede encryption for encoding (transmitting). The opposite for decoding (receiving).
- Dedicated local DRAM is used for keeping packet information and compression and encryption contexts for each session

Protocols Supported by Hi/fn Hardware

- IP Security (IPSEC)
- Point-to-point compression (PPP/CCP)
- Point-to-point tunneling (L2TP/PPTP)
- Secure Socket Layer (SSL/TLS)
- Secure Electronic Transaction (SET)
- Certificate Services (PKIX)

Summary

- Security is a must for private communications over public media and for digital commerce
- Dedicated hardware processing is a must for security implementations to be practical in terms of speed
- Advances in silicon make wide security deployment practical in terms of cost
- Compress before encrypting in order to preserve currently deployed WAN bandwidth that is already relying on compression