Cryptanalytic Hardware Architecture Optimized for Full Cost

Michael J. Wiener

2005 February 25

Overview

- Multiprocessor h/w architecture
  - Connects many processors to common large memory
  - Useful for broad range of cryptanalytic attacks
  - Asymptotically optimal

Constant factor of improvement possible

When all h/w costs considered, "full cost"

Overview (cont'd)

- Define "full cost" of an algorithm
  - Contrast with traditional run-time measure
    - "processor cost"
- Describe multiprocessor h/w architecture
  - Optimality proof outline
- Applications
  - Discrete log
  - Factoring
  - Multiple Encryption
  - Hash collisions

Processor Cost of an Algorithm

- Traditional cost measure
  - # processor steps or run-time
- If multiple processors used:

$$\left(\text{\# processors} \right) \times \left(\begin{array}{c} \text{\# steps} \\ \text{or} \\ \text{run-time} \\ \text{per processor} \end{array} \right)$$

Full Cost

$$\left(\begin{array}{c}\text{\# h/w components}\\ \text{or}\\ \text{total h/w \$}\end{array}\right) \times \left(\text{run-time}\right)$$

- Used by
  - [Amirazizi, Hellman, 1981, 1988]
  - [Bernstein, 2001]
  - [Lenstra, Shamir, Tomlinson, Tromer, 2002]
    - called it "throughput cost"

Comparing Cost Measures

- Only difference:
  - Traditional measure ignores all h/w components except processors
- Which measure is correct?
  - Right or wrong discussions pointless
  - Useful or not useful
  - Processor cost is simpler
  - Full cost gives more useful answers

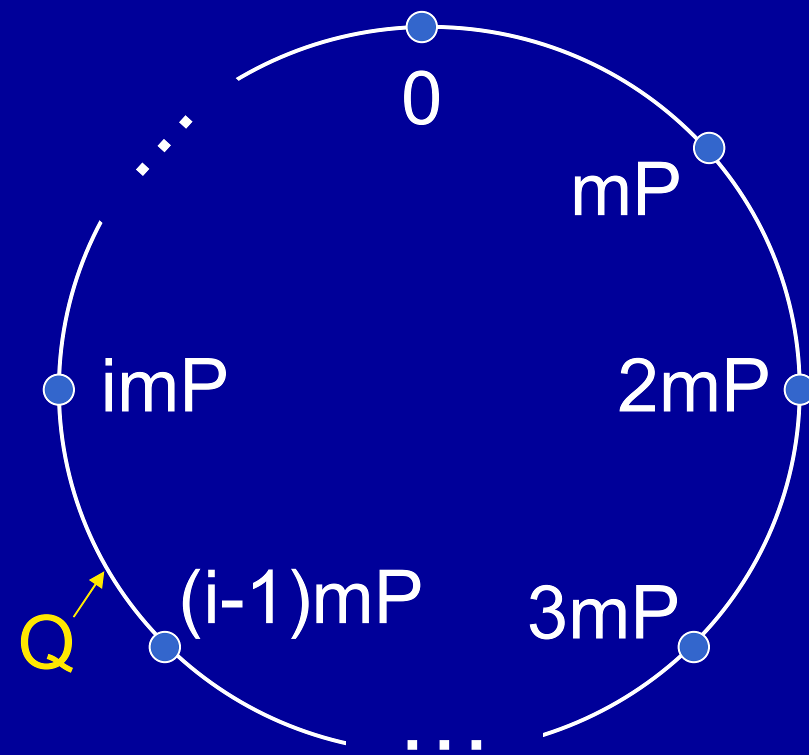
Full Cost Example: Discrete Log

- EC log
  - Given P, Q = kP, find k
  - Prime group order $n \approx 2^{80}$
- Use one PC + memory
- Compare
  - Shanks' method
  - Pollard's ρ-method

Shanks' Method

Also called baby-step giant-step

- Let $m = \lceil n^{1/2} \rceil \approx 2^{40}$
- Store mP, 2mP, …, $\lceil n/m \rceil$mP
- $Q \leftarrow Q + P$ until Q is in table
- If Q = imP after j steps: k = im - j



Shanks' Method Costs

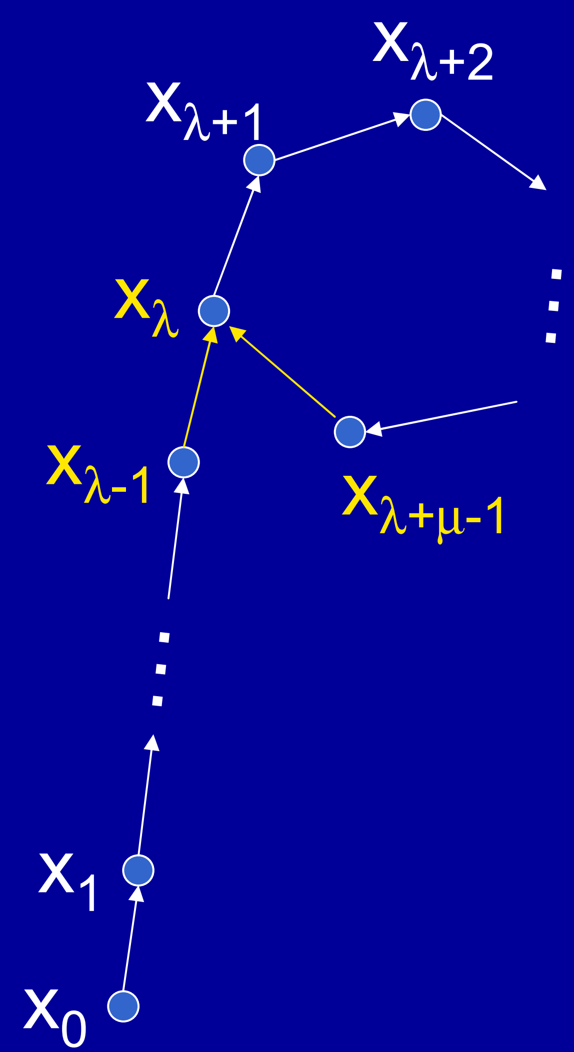
- Hardware
  - Must store 2^{40} points
    - x coordinate only (10 bytes)
  - 10 Tbytes
  - $1M assuming $100/Gbyte
  - Additional cost of PC is insignificant
- Time
  - Expect 1.5 x 2^{40} EC adds

Pollard's ρ-Method

- Choose an iterating function f [Teske, 1998]
- Choose some starting EC point x_0
- Iterate $x_{i+1} = f(x_i)$
- f is constructed so that collision

  $f(x_{\lambda-1}) = f(x_{\lambda+\mu-1}) = x_\lambda$

  reveals key k



ρ-Method Costs

- Hardware
  - Negligible memory using distinguished points [Rivest]
    - Only store points with 30 trailing zero bits
  - \$1000 for PC
- Time
  - Expect $(\pi/2)^{1/2} \times 2^{40}$ EC adds

Comparison

Processor cost:
about the same

	# processors	EC adds	h/w $
Shanks	1	$\approx 2^{40}$	$1M
ρ-method	1	$\approx 2^{40}$	$1000

Full cost:
ρ 1000x better

Which Cost Measure Makes Sense?

- ρ-method really is 1000x better
  - Traditional measure makes no sense here
  - Full cost gives a useful answer

Was this a Fair Comparison?

- Between processor cost and full cost?
  - Yes
  - Hardware design for Shanks is very poor
    - Deserves 1000x poorer rating than ρ-method
- Between Shanks' and ρ-methods?
  - No
  - Shanks with only one processor is inefficient
  - For fair comparison, optimize designs
    - As Lenstra et al. did for factoring

Improving Shanks Approach

- Use 100 PCs in parallel
  - Connect all PCs to one large memory
  - Expensive logic required to maintain high-speed memory access
  - Estimate of total cost: $2M

New Comparison

Processor cost:
all the same

	# processors	EC adds	h/w $
Shanks	1	$\approx 2^{40}$	$1M
Parallel Shanks	100	$\approx 2^{40}/100$	$2M
ρ-method	1	$\approx 2^{40}$	$1000

Full cost: parallel Shanks 50x improved
ρ still 20x better

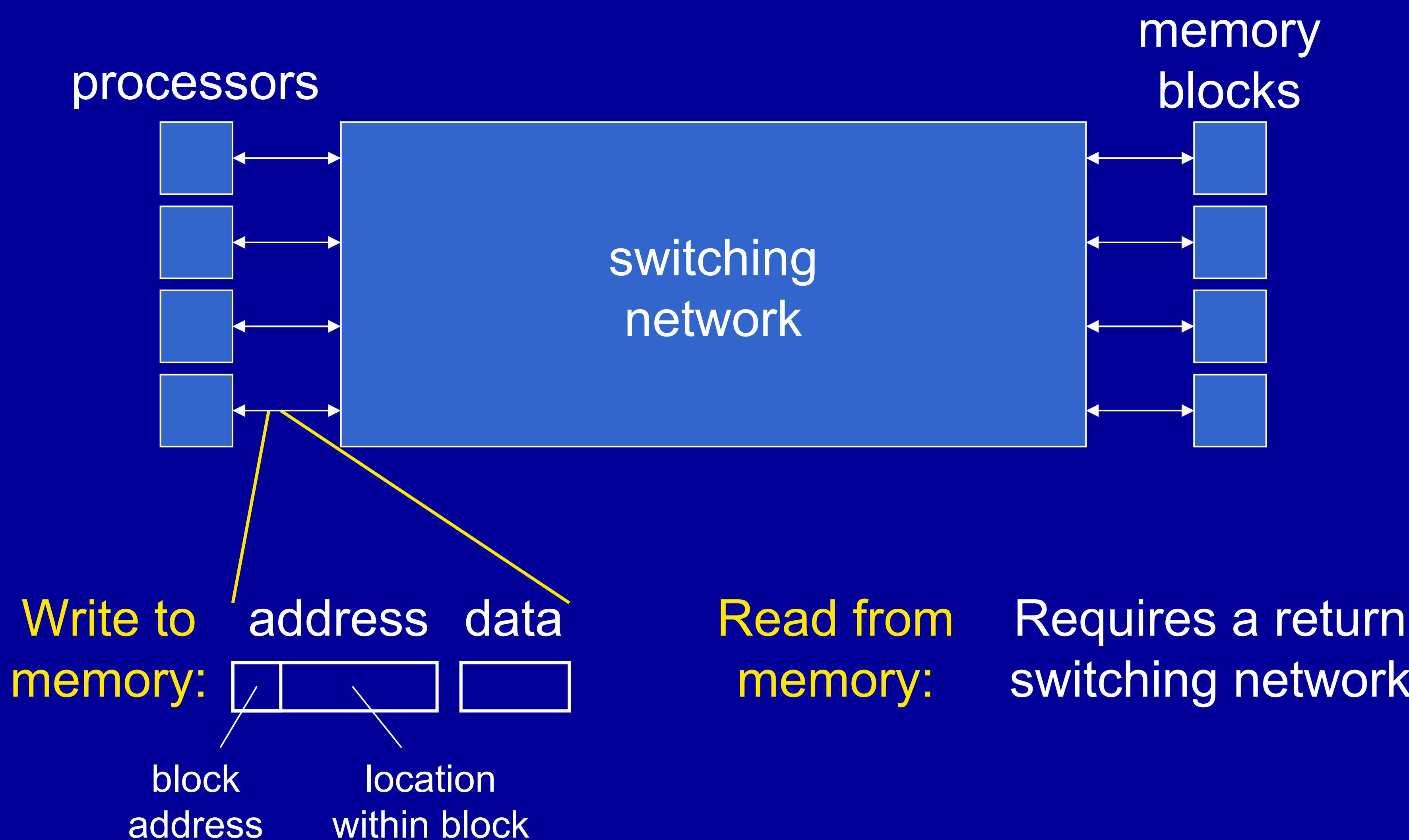
Even More Parallelism?

- Can we use more parallelism to reduce Shanks' full cost to that of ρ?
  - No
  - Cost of connections to one large memory too high
  - More about this later

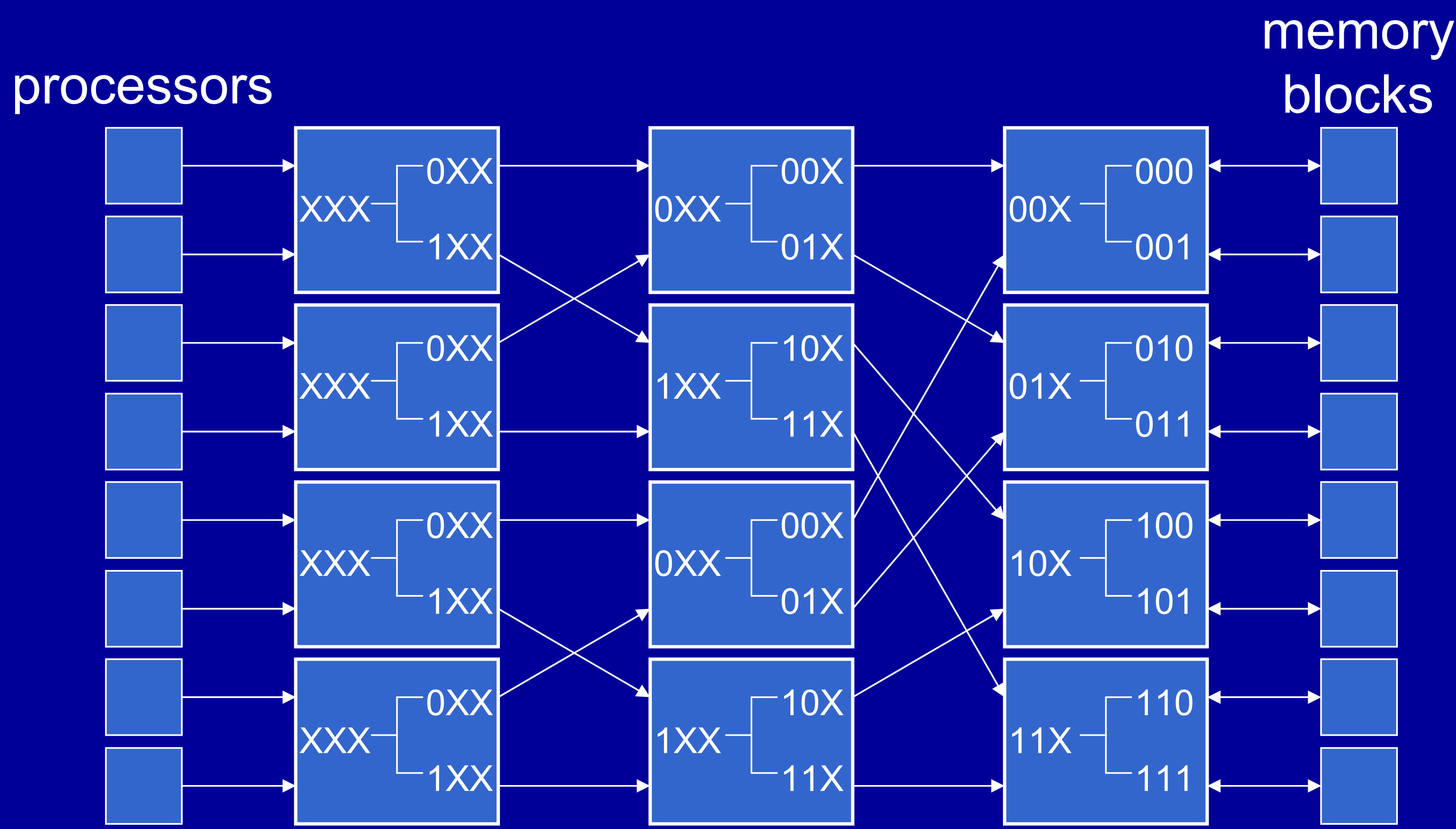
Requirements for Many Attacks

- Large memory
- Multiple processors needed to reduce full cost
- Need simultaneous high-speed access to the large memory
  - Expensive

Requirements (cont'd)



Multiprocessor H/W Architecture



Long connections to last stage are a problem.

Switching Network Cost

- n processors, n memory blocks
- $\Theta(n \log n)$ switching elements
- Total wire length
  - Hardware design achieves $\Theta(n^{3/2})$
  - This is the best that can be done
- $\therefore$ costs are dominated by wires in the switching network!
  - Answers open question [Amirazizi, Hellman]

Wire Length Proof Outline

- In multiprocessor architecture
  - With processors, memory blocks in 2-D grid
  - $\Theta(n \log n)$ wires, average length $\Theta(n^{1/2}/\log n)$
  - Establishes upper bound $O(n^{3/2})$
- Full paper proves
  - Must have $\Theta(n)$ processors separated from $\Theta(n)$ memory blocks by distance $\Omega(n^{1/2})$
  - Establishes lower bound $\Omega(n^{3/2})$

Wires Dominate Costs – Is this Reasonable?

- Yes
- Misleading to call it “wire”
  - By this definition, internet is mostly just wire
- PCs access internal memory at Gbyte/s rates
  - Maintaining this speed for 4 or 8 processors is fairly easy
  - For 10,000 processors, say, must maintain this speed over distances (costly)

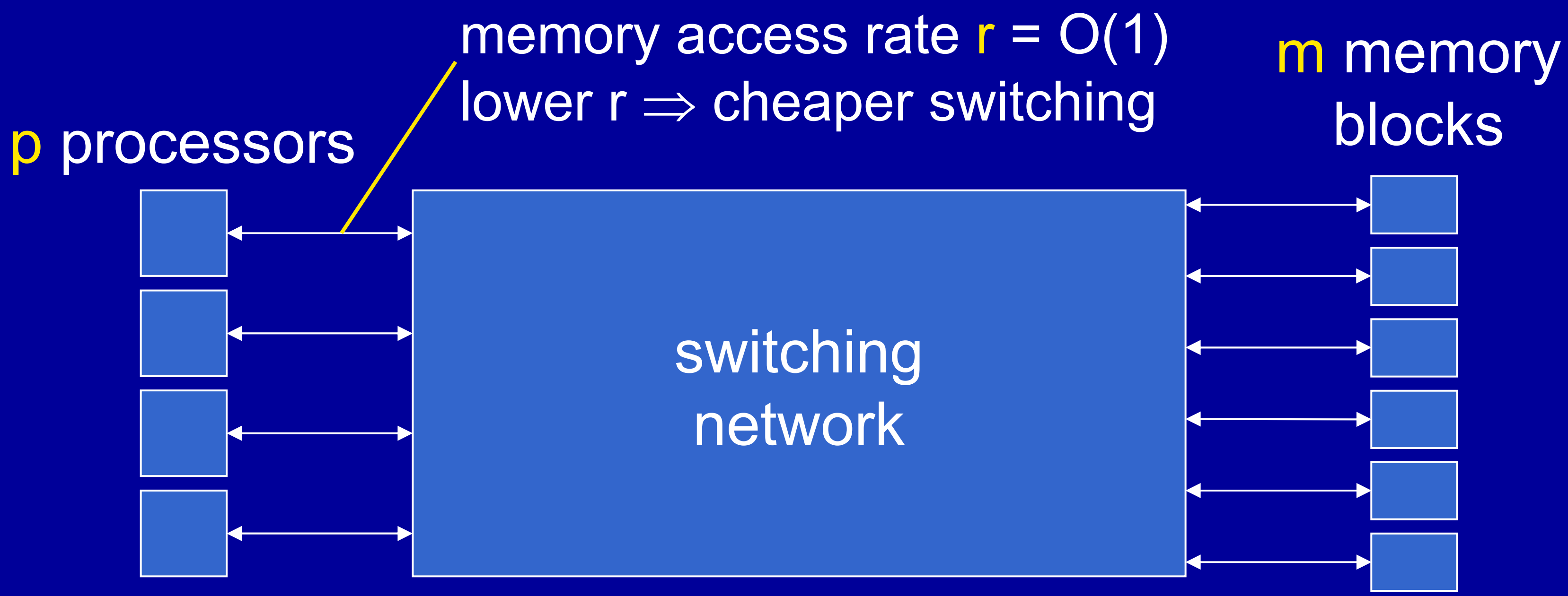
Internet Example

- Processors are PCs
- Large memory is collective RAM in PCs
- Switching network is the internet
- PCs need to read/write data to other PCs' memories millions of times per second
  - Internet is far too slow for this
  - A new internet to handle this demand would cost orders of magnitude more than all PCs together

Switching Network Delays

- Is latency through switching network a problem?
  - Usually no
- For most attacks,
  - Can continue to push out more memory access requests before current one gets to memory
  - Filling up memory with Shanks' giant steps illustrates this

More General Case



- Total h/w cost (with switching network + wires)
  - $\Theta(p + m + (pr)^{3/2})$
  - Proof in full paper

Full Cost

- Let T be total processor cost of an attack
  - Spread across p processors
- Full cost is time T/p x hardware cost:

$$F = \Theta((T/p)(p + m + (pr)^{3/2}))$$

Full Cost vs. Processor Cost

- Rewrite equation as

$$F = \Theta(T(1 + m/p + p^{1/2}r^{3/2}))$$

- So $F=\Omega(T)$, and

$$F=\Theta(T) \text{ iff } m = O(p) \text{ and } r = O(p^{-1/3})$$

Applications of the Multiprocessor Architecture

- Discrete log
  - Shanks' method
  - Parallel collision search
- Factoring
- Double encryption
- Three-key triple encryption
- Hash collisions

Notation

- Throughout discussion of applications
  - Ignore constant factors
  - Ignore log factors
    - Distraction
    - Full paper tracks log factors
- E.g., $3n^2(\log n)^{3/2} \longrightarrow n^{2+o(1)}$

Discrete Log

- Group with no index calculus attack
  - E.g., elliptic curves
- Prime group order n
- Both Shanks and collision search require

  $T = n^{1/2+o(1)}$ processor steps

Full Cost (Shanks)

Proc. time	$T = n^{1/2+o(1)}$
Memory	$m = n^{1/2+o(1)}$
Access rate	$r = n^{o(1)}$ (high)
Processors	$p = ?$ (too be optimized)
Full cost	$F = \Theta(T(1 + m/p + p^{1/2}r^{3/2}))$ $= n^{1/2+o(1)}(1 + n^{1/2}/p + p^{1/2})$ Minimum: $F = n^{2/3+o(1)}$ when $p = n^{1/3+o(1)}$

Full Cost (Parallel Collision Search)

Proc. time	$T = n^{1/2+o(1)}$
Memory	$m = pn^{o(1)}$ (each processor stores constant # of distinguished points)
Access rate	$r = m/T = p/n^{1/2-o(1)}$ (low)
Processors	$p = ?$ (too be optimized)
Full cost	$F = \Theta(T(1 + m/p + p^{1/2}r^{3/2}))$ $= n^{1/2+o(1)}(1 + 1 + p^2/n^{3/4})$ Minimum: $F = n^{1/2+o(1)}$ when $p = n^{3/8+o(1)}$ or less

Comparison

- Shanks full cost: $n^{2/3+o(1)}$
- Collision search full cost: $n^{1/2+o(1)}$
- Full cost analysis reveals
  - Collision search is better than Shanks
  - Even though same processor cost
- Shanks advantage:
  - Deterministic time
  - Not important in practice, but mathematically pleasing

Factoring with NFS

- NFS parameters are chosen to trade-off costs of
  - Relation collection step
  - Matrix step
- For standard trade-off based on processor cost,
  - Matrix step has higher full cost
- Rebalancing required to minimize full cost [Lenstra, Shamir, Tomlinson, Tromer]

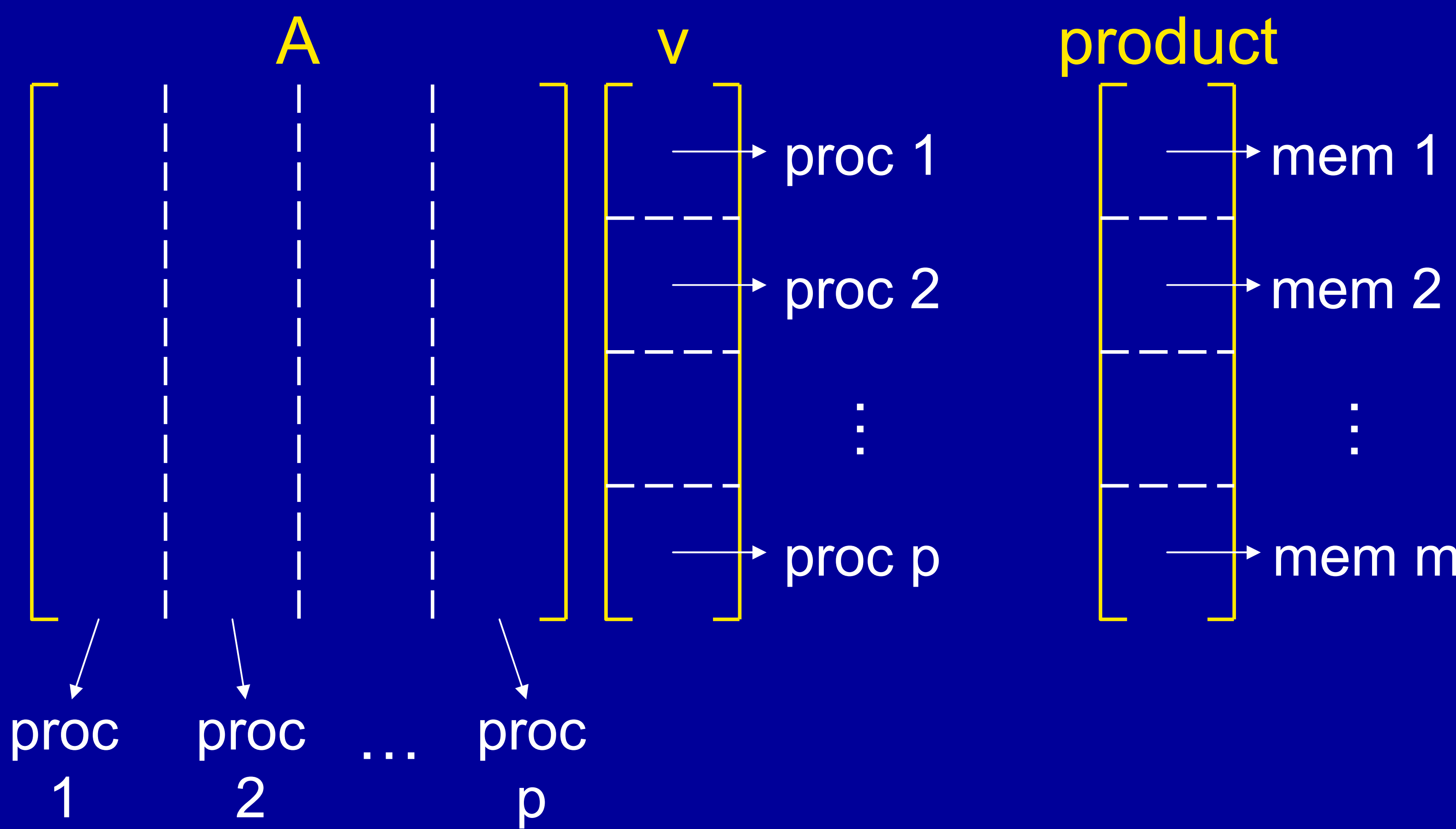
Matrix Step

- Run-time dominated by matrix-vector products (over GF(2))

$$\begin{array}{cccc} A & v & & \text{product} \\ \begin{bmatrix} \\ \text{sparse} \\ \text{columns} \\ \\ \end{bmatrix} & \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix} & = & \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix} \end{array}$$

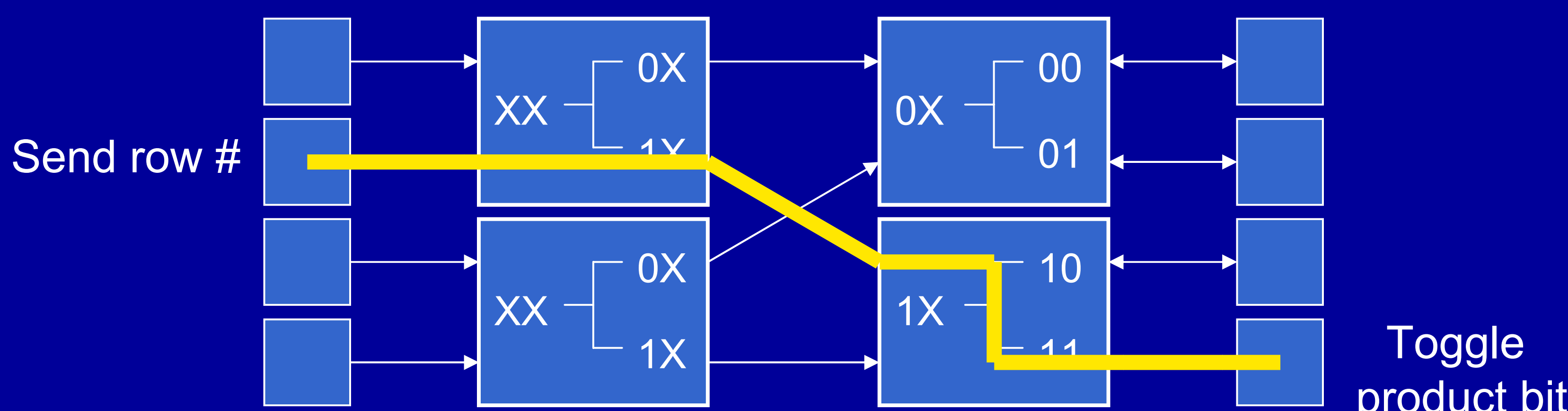
- Represent columns of A as a list of row indices for the 1 entries

Multiprocessor Matrix Step



Matrix-Vector Multiply

- For columns with corresponding v bit 1,
  - Send row numbers to memory
  - Row numbers serve as memory addresses
- Memory toggles product bit each time row number arrives

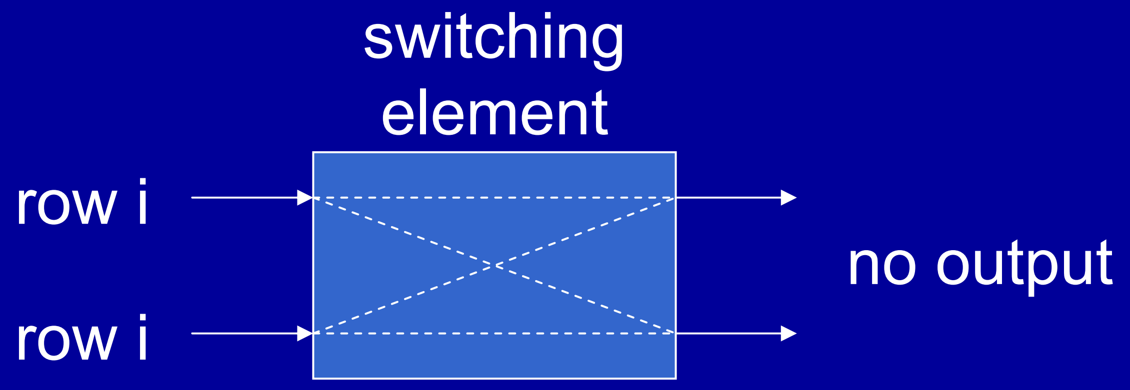


A Problem

- For most attacks, memory accesses have uniformly random addresses
  - Tends to balance load on memory blocks
- Top rows of matrix much denser than lower rows
  - Memory blocks for low row numbers will be swamped

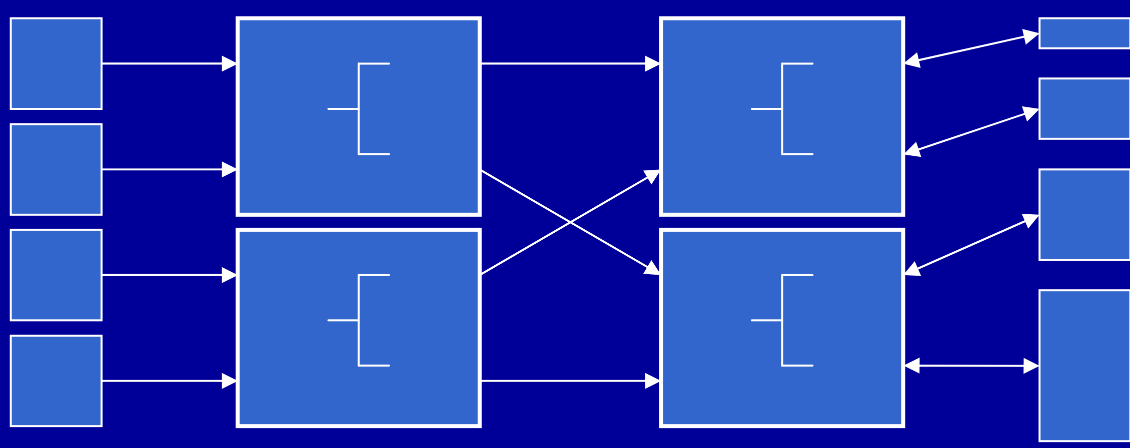
Partial Solution

- Switching elements take in two row numbers at a time
- If both equal, they cancel
  - Toggling output bit twice = do nothing
  - Just drop equal row numbers



Full Solution

- Top memory blocks still swamped
- Solution:
  - Make top memory blocks smaller
  - Change address decisions in switching elements to compensate



Performance

- Asymptotically optimal matrix step
  - Without a fundamentally different approach, can only be beaten by a constant factor
  - But maybe a large constant
- I have not tried doing a detained design to compare to other designs:
  - [Bernstein]
  - [Lenstra, Shamir, Tomlinson, Tromer]

Double Encryption

- $E_{k_2}(E_{k_1}(P)) = C$
- $|\text{key space}| = n$
- Given (P, C) find (k_1, k_2)
  – Assume enough other (P, C) pairs to uniquely identify key pair
- Attack approaches
  – Standard meet-in-the-middle
  – Parallel collision search

Meet-in-the-Middle Attack

- Let $D_k(\cdot)$ denote decryption
- Observe that $E_{k_1}(P) = D_{k_2}(C)$

For each candidate k_1

- Store $(k_1, E_{k_1}(P))$

large memory

For each candidate k_2

- Look up $D_{k_2}(C)$ to get possible k_1
- Test (k_1, k_2) on other (P, C) pairs

Multiprocessor Meet-in-the-Middle

Proc. time	$T = n^{1+o(1)}$
Memory	$m = n^{1+o(1)}$
Access rate	$r = n^{o(1)}$ (high)
Processors	$p = ?$ (too be optimized)
Full cost	$F = \Theta(T(1 + m/p + p^{1/2}r^{3/2}))$ $= n^{1+o(1)}(1 + n/p + p^{1/2})$ Minimum: $F = n^{4/3+o(1)}$ when $p = n^{2/3+o(1)}$ Full cost > processor cost

Parallel Collision Search Attack

- Uses collision search to generate many candidate key pairs
  - Tests each key pair on multiple (P, C) pairs until (k_1, k_2) found

Parallel Collision Search Cost

Memory	$m = ?$ (to be optimized)
Proc. time	$T = n^{3/2+o(1)}/m^{1/2}$
Access rate	$r = m^{1/2}/n^{1/2-o(1)}$ (low)
Processors	$p = ?$ (too be optimized)
Full cost	$F = \Theta(T(1 + m/p + p^{1/2}r^{3/2}))$ $= n^{3/2+o(1)}/m^{1/2}(1 + m/p + p^{1/2}m^{3/4}/n^{3/4})$ Minimum: $F = n^{6/5+o(1)}$ when $p = n^{3/5+o(1)}$ and $m = n^{3/5+o(1)}$

Double Encryption Results

- Parallel collision search is better than standard meet-in-the-middle attack
  - $n^{6/5+o(1)}$ vs. $n^{4/3+o(1)}$
- Double encryption is widely believed to be no better than single encryption
  - Not true
  - Small advantage: $n^{1/5+o(1)}$

Three-Key Triple Encryption

- $E_{k_3}(D_{k_2}(E_{k_1}(P))) = C$
- Best attack known:

For each candidate k_1

- Perform double encryption attack using
  - $E_{k_1}(P)$ as plaintext
  - C as ciphertext

Triple Encryption Attack Cost

- Just n times double encryption attack
- Full cost: $F = n^{11/5+o(1)}$
- Ignoring the o(1),
  - 3-key triple-DES: 123 bits of security

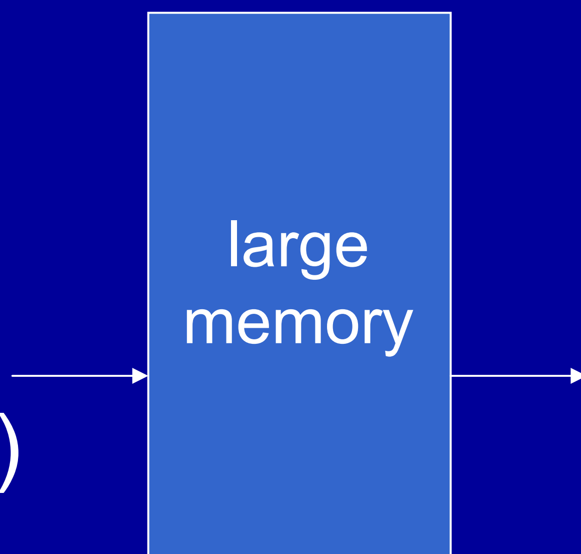
Hash Collisions

- Find two texts, x and y, such that
  - $x \neq y$
  - $H(x) = H(y)$
- |output space| = n
- Approaches:
  - Simple table-based attack
  - ρ collision search

Table-Based Attack

Choose unique texts x

- Store (x, H(x))



Check for duplicate hash outputs in table

Duplicate
$\Rightarrow$ H(x) = H(y)

Multiprocessor Table Attack

Proc. time	$T = n^{1/2+o(1)}$
Memory	$m = n^{1/2+o(1)}$
Access rate	$r = n^{o(1)}$ (high)
Processors	p = ? (too be optimized)
Full cost	$F = \Theta(T(1 + m/p + p^{1/2}r^{3/2}))$ $= n^{1/2+o(1)}(1 + n^{1/2}/p + p^{1/2})$ Minimum: $F = n^{2/3+o(1)}$ when $p = n^{1/3+o(1)}$

ρ Collision Search

Proc. time	$T = n^{1/2+o(1)}$
Memory	$m = pn^{o(1)}$ (each processor stores constant # of distinguished points)
Access rate	$r = m/T = p/n^{1/2-o(1)}$ (low)
Processors	$p = ?$ (too be optimized)
Full cost	$F = \Theta(T(1 + m/p + p^{1/2}r^{3/2}))$ $= n^{1/2+o(1)}(1 + 1 + p^2/n^{3/4})$ Minimum: $F = n^{1/2+o(1)}$ when $p = n^{3/8+o(1)}$ or less Better than table-based attack

Conclusion

- Multiprocessor h/w architecture is asymptotically optimal for full cost
- Full cost better reflects reality than traditional processor cost
- Collision search techniques give best attack for several problems

Conclusion (cont'd)

Attack	Method	Processor steps	Full Cost
Discrete log	Shanks Parallel coll. search	$n^{1/2+o(1)}$ $n^{1/2+o(1)}$	$n^{2/3+o(1)}$ $n^{1/2+o(1)}$
Double encryption	Meet-in-the-middle Parallel coll. search	$n^{1+o(1)}$ $n^{1+o(1)}$	$n^{4/3+o(1)}$ $n^{6/5+o(1)}$
Triple encryption	Meet-in-the-middle Parallel coll. search	$n^{2+o(1)}$ $n^{2+o(1)}$	$n^{7/3+o(1)}$ $n^{11/5+o(1)}$
Hash collision	Table-based method Parallel coll. search	$n^{1/2+o(1)}$ $n^{1/2+o(1)}$	$n^{2/3+o(1)}$ $n^{1/2+o(1)}$