AES on Sharemind

Riivo Talviste, Jan Willemson
{riivo,janwil}@cyber.ee

Estonian Computer Science Theory Days
Kubija, January 27-29, 2012

This research was, in part, funded by the U.S. Government. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. "A" (Approved for Public Release, Distribution Unlimited)

This research was supported by European Social Fund's Doctoral Studies and Internationalisation Programme DoRa.

CYBERNETICA EXCS ESTONIAN COMPUTING Europa Liit Europa Sotsiaalfond Eesti tuleviku heaks DoRa UNIVERSITY OF TARTU 1632

Motivation

- Common benchmarking test
- Can be used as a cryptographic primitive
  - E.g. Database JOIN operation

Advanced Encryption Standard

- Symmetric block cipher
  - Using 128-bit blocks
  - 128, 192 or 256-bit keys
    - We use 128-bit keys in our implementation

AES-128 on Sharemind

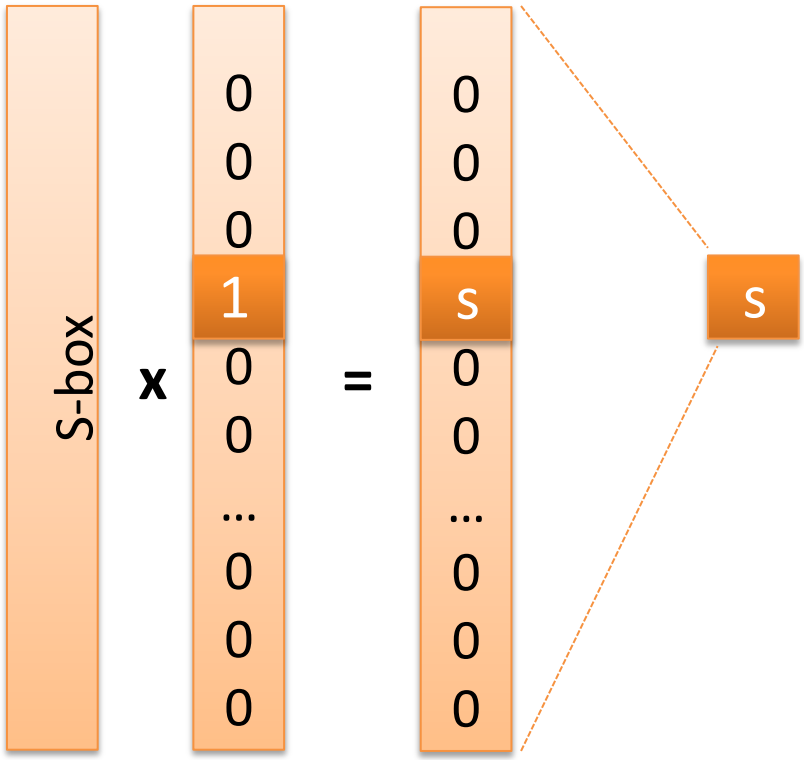
- Straightforward, following the NIST specification
- Plaintext and key are bitwise secret shared

  $s = s_1 \oplus s_2 \oplus s_3$, where $\oplus$ is XOR
- Four 8-bit bytes are packed into a single 32-bit integer (word)
- Most computations are local
  - Additions, bitshifts, multiplications by constant in $GF(2^8)$

AES-128 on Sharemind (2)

- **S-box**: non-linear byte-for-byte substitution table
  - Has algebraic definition
    - Usually pre-computed and given as a 16x16 byte table
    - We use it as a 256-byte vector
  - Byte `b` is replaced with `S-box[b]`
  - Requires communication

Computing S-box: characteristic vector



Computing S-box: characteristic vector (2)

- Substitute byte b = $b_7b_6b_5b_4b_3b_2b_1b_0$
- S-box[0] $(1-b_7)(1-b_6)(1-b_5)(1-b_4)(1-b_3)(1-b_2)(1-b_1)(1-b_0)$

  [1] $(1-b_7)(1-b_6)(1-b_5)(1-b_4)(1-b_3)(1-b_2)(1-b_1)b_0$

  [2] $(1-b_7)(1-b_6)(1-b_5)(1-b_4)(1-b_3)(1-b_2)b_1(1-b_0)$

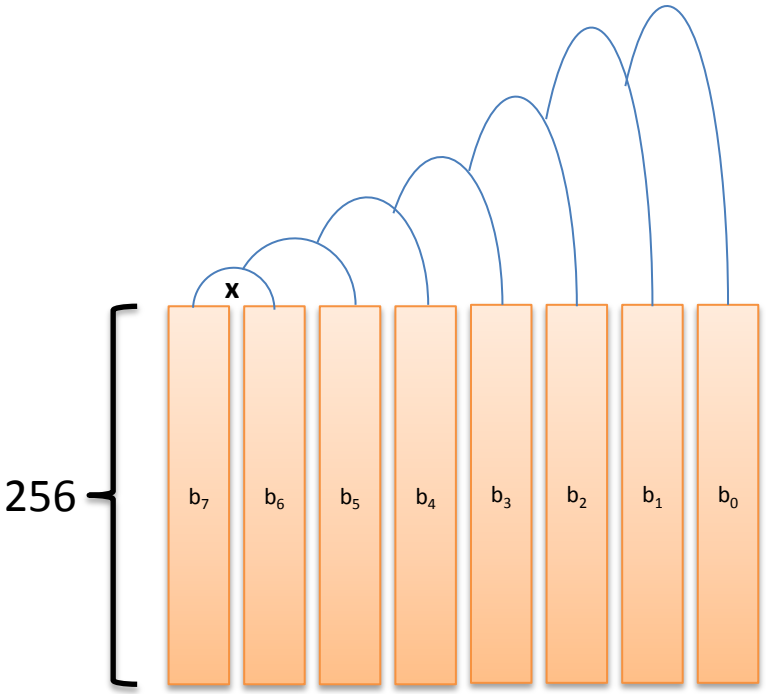
  ...

  [254] $b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1(1-b_0)$

  [255] $b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$

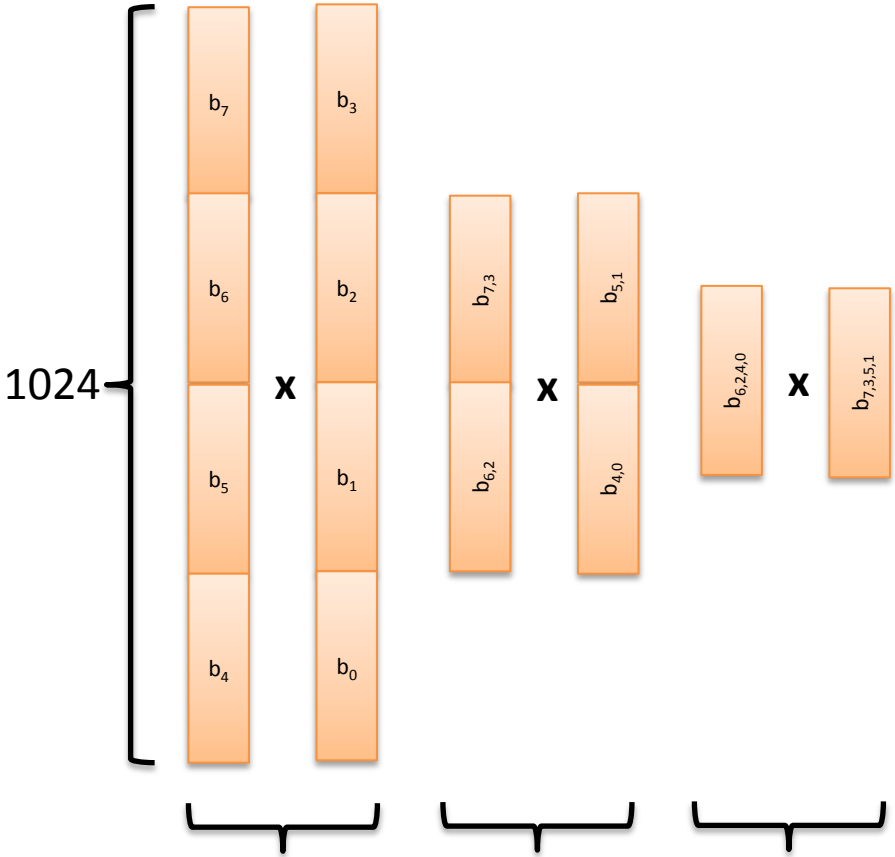
Characteristic vector: multiplication

Option 1:



Total: 7 rounds of multiplications

Option 2:



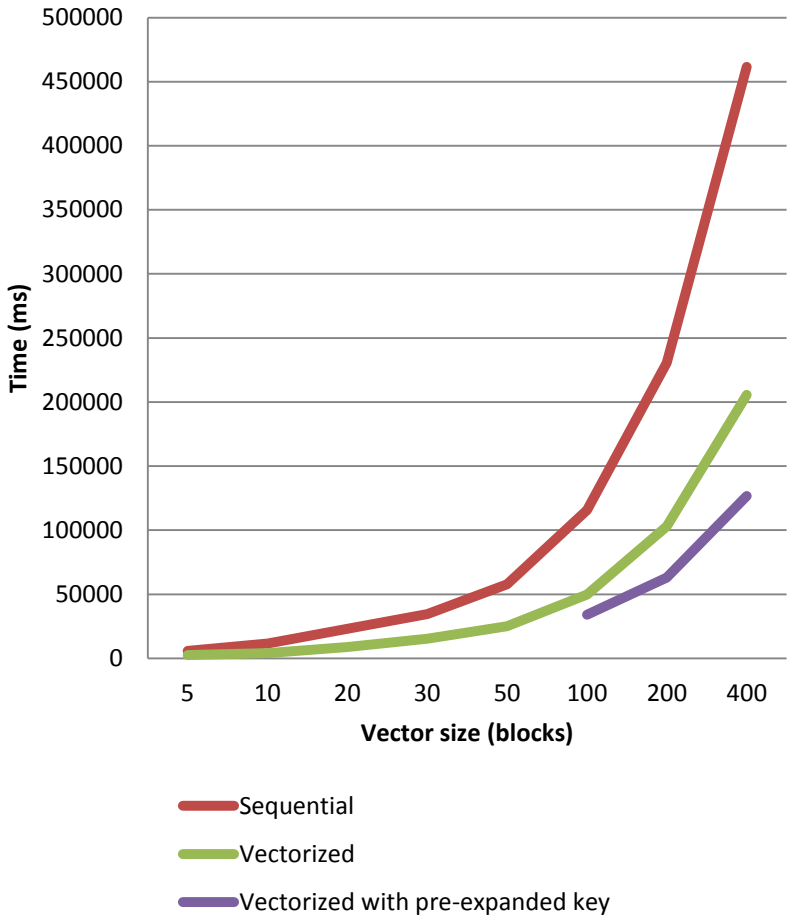
Vectorization

- Several plaintexts (128-bit blocks)
  - Each block encrypted separately
- Sharemind is highly optimized for vector operations
- Idea: Process several plaintext blocks in parallel
  - Vector lengths increase by the factor of #(blocks), but #(communication rounds) stays the same

Pre-expanded key

- S-box is used in key expansion phase:
  - Cipher key is used to generate ten 128-bit round keys
- Secret shared cipher key has to be known to miners before AES can be executed
- Idea: We can move key expansion to pre-processing phase and provide miners with the secret shared pre-expanded key instead

Benchmarking



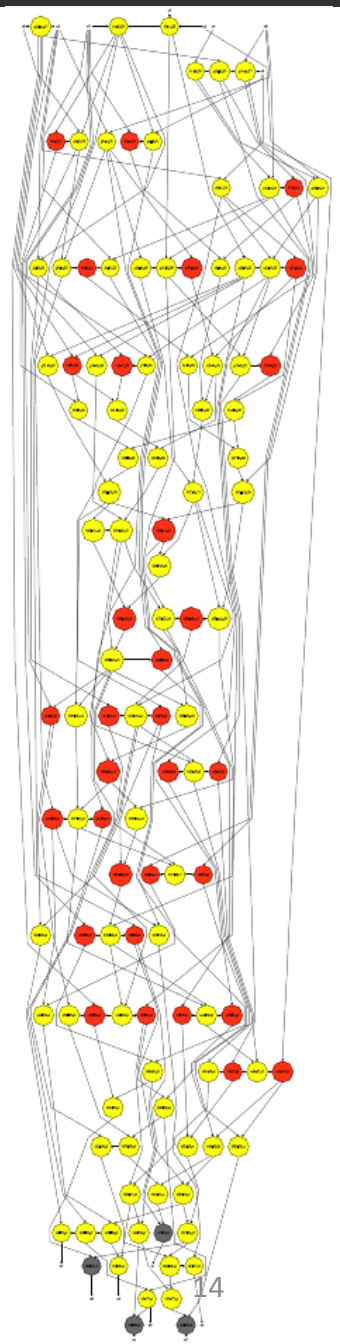
- 1 Gbit LAN
- Results:
  - 13.9 B/s
  - 33.2 B/s
  - 49.5 B/s
- Hoped for larger speedup :(

What happened?

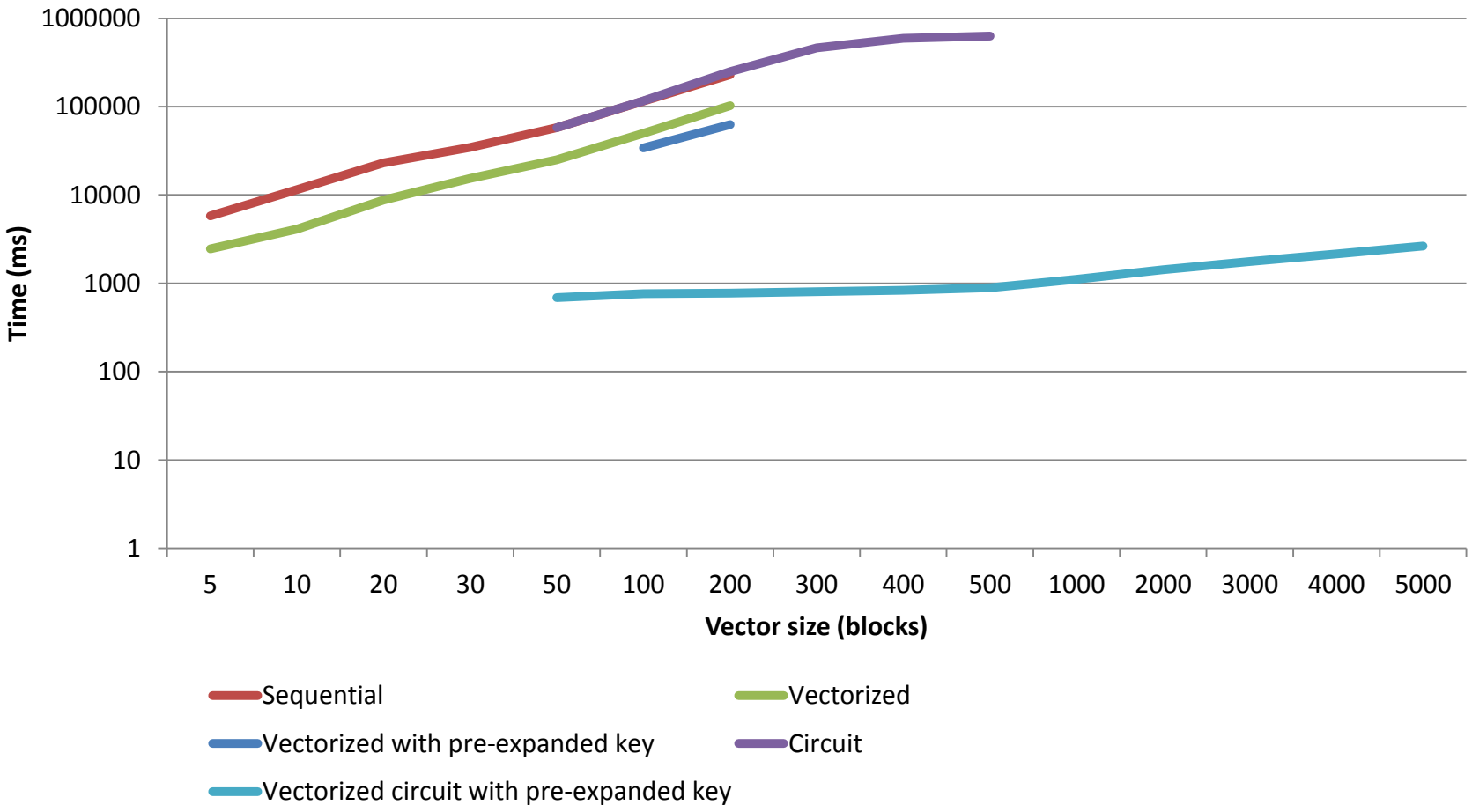
- Vectors are too large for the network layer
  - E.g. SubBytes() multiplies vectors of size up to #(blocks)×4096 words
  - Saturation point for multiplication protocol depends on bandwidth
    - In our scenario, it is ca. 10 000
  - Hence, encrypting more than 2 blocks at once, clogs the network layer!
- Optimizing only communication rounds is wrong
  - Large amount of data kills parallelization

S-box with circuits

- Boyar and Peralta [2010, 2011] have come up with several circuits for AES S-box
  - Using only AND, XOR, XNOR
  - Minimal depth
- In multi-party computing, XOR is free (local), AND (multiplication) costs
  - We want minimal number of AND gates



Benchmarking, again



Benchmarking, again (2)

- Vectorized circuit with pre-expanded key
  - Maximum vector length: #(blocks)×18 words
  - Reach saturation point with ca. 550 blocks
  - Average throughput: 12.6 kB/s

Compare to others

Team	Model	sec/ block
Damgård, Keller [2009]	3-party, w/o pre-expanded key, 10 blocks in parallel	2
Huang et al. [2011]	2-party, pre-expanded key	0.06
Launchbury et al. [2011]	3-party, 1 block, no pipelining	0.015
	3-party, 64 blocks in parallel, pipelined	0.007
Us [2011]	3-party, pre-expanded key, 10 blocks in parallel	0.07
	3-party, pre-expanded key, 100 blocks in parallel	0.007
	3-party, pre-expanded key, 1000 blocks in parallel	0.001
	3-party, pre-expanded key, 5000 blocks in parallel	0.0005

Conclusions

- AES on secret shared data can be done
- Optimizing only communication rounds is wrong
  - Large amount of data kills parallelization
- Circuits help to lower the amount of data
- In future, we use it to implement oblivious database JOIN operation