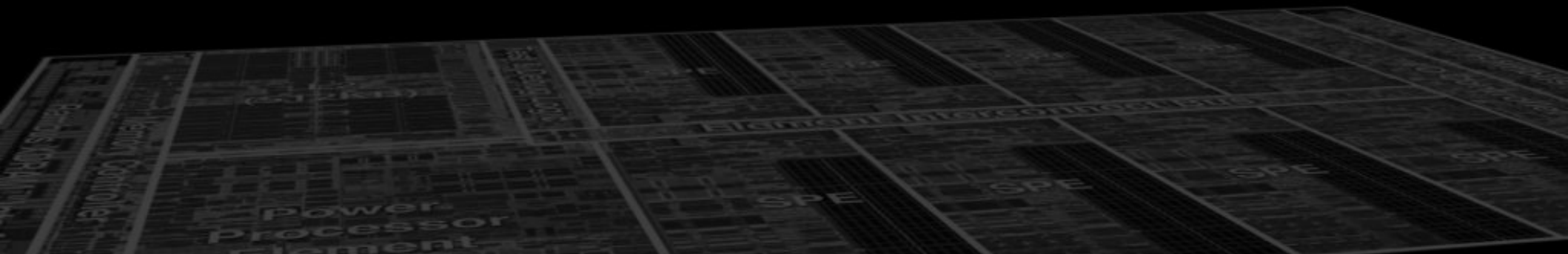


# Breaking UNIX crypt() on the PlayStation 3

Marc Bevand

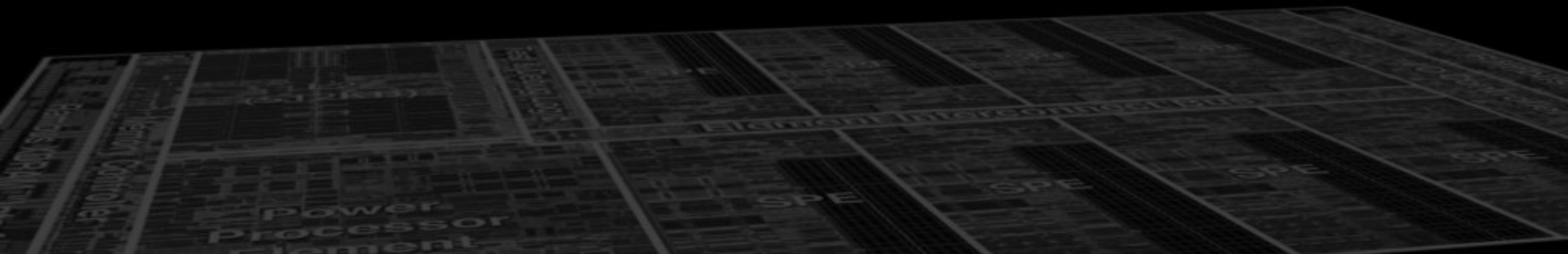
[m.bevand@gmail.com](mailto:m.bevand@gmail.com)

[marc.bevand@rapid7.com](mailto:marc.bevand@rapid7.com)



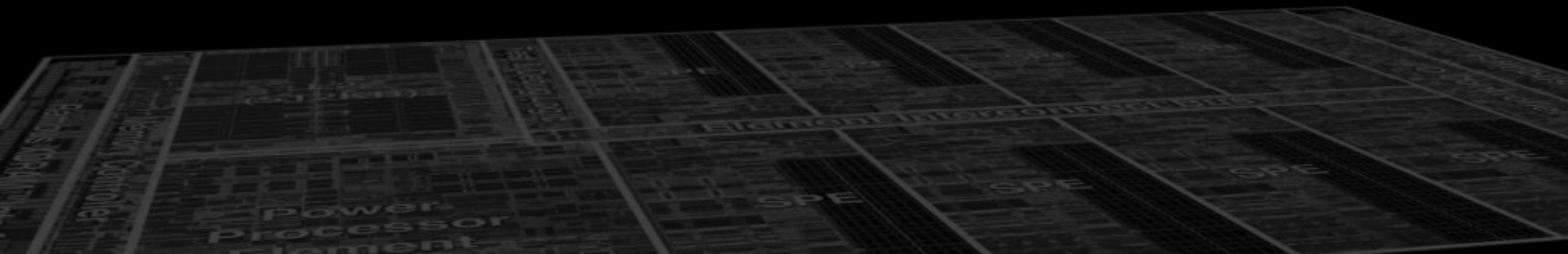
# Plan

- DES-based UNIX crypt() password hashes
- Bitslice DES
- PlayStation 3's Cell B.E. processor
- DES S-Boxes optimized for the Cell
- Results in numbers: perf/\$, perf/W



Isn't UNIX crypt() 30+ years old ?

**WTF ?**



# DES-based UNIX crypt()

- Frequently used in large, legacy UNIX environments
- Still the default in Solaris 10 as well as latest Nevada builds (see CRYPT\_DEFAULT in /etc/security/policy.conf)
- Even in 2008, there is value in optimizing the bruteforcing of UNIX crypt() password hashes

# Quick reminder

- UNIX crypt() algorithm:
  - Build a 56-bit key from the passwd
  - Pick a random 12-bit salt to perturb the E box of the DES algorithm
  - DES-encrypt 25 times an all-bits-zero block
- Passwords limited to 8 chars
- Salt makes pre-computed attacks 4096 times costlier

# Best bruteforcing tool (so far)

- Bitslice DES implemented by Dr Eli Biham in 1997
  - Represents the most computing intensive part of DES (S-Boxes) as series of logical instructions on 1-bit values
  - On N-bit processors, a bitslice implementation performs N encryptions/decryptions in parallel
- Implemented in John the Ripper (S-Boxes from Matthew Kwan): 8.8 Mpw/s on quad-core Q6700 @ 2.66 Ghz (2.2 Mpw/s/core, "only one salt")

# Bitslice DES

- One S-box maps a 6-bit input to a 4-bit output
- 8 S-boxes total: S1..S8
- Each S-box can be represented by a logical circuit using gates such as AND, OR, XOR, etc
- Find the circuits with the smallest number of gates
- No good algorithm is known to find the best circuits, bruteforcing is the only solution

# Bitslice DES on the Cell B.E.

- PlayStation 3 processor is the Cell B.E.
- Particularly well adapted to bitslice DES
- 7 usable cores @ 3.2 GHz: 1 PPU + 6 SPUs
- Each SPU: 128 registers, 128-bit each (128-way bitslicing !)
- The ISA defines all the integer and boolean instructions needed, in particular: selb (mux), orc (or not), andc (and not)

# Bruteforcing the S-box circuits

- A tool has been implemented based on ideas presented by Kwan:
  - Construct iteratively a data structure representing the circuit with its lines and gates (initially, it contains only 6 input lines representing the 6-bit input)
  - Randomly add simple series of gates to try to find the output bits
  - If it doesn't work, add a selb gate to reduce the pb to two 5-bit problems, then 4-bit pbs, etc
- This tool will be released as open source

# Optimized S-box circuits

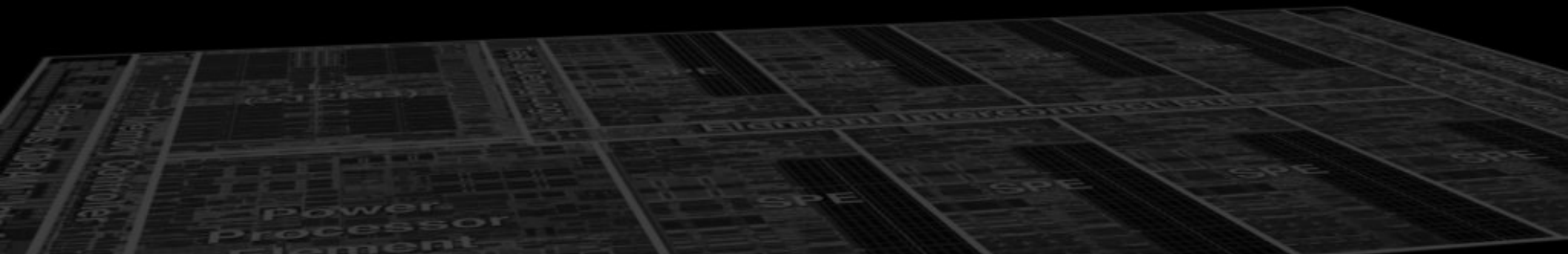
- Takes ~2 hours of bruteforcing to finally find 8 circuits with an average of 45.5 gates:
  - S1, ..., S8: 50, 46, 46, 34, 50, 47, 46, 45 gates
- 10.7% theoretical improvement over Kwan's best result of 51 gates
- Note: unverified claim of "less than 40 gates" by Dag Arne Osvik and Eran Tromer (don't reply to email): [http://www.hyperelliptic.org/SPEED/slides/Osvik\\_cell-speed.pdf](http://www.hyperelliptic.org/SPEED/slides/Osvik_cell-speed.pdf)

# Implementation details

- Each SPU has 2 pipelines (0: even, 1: odd) and can issue and complete up to 2 instructions/clock
- `spu_timing`: static timing analysis tool
- All logical instructions: pipeline 0, 2-clock latency
- For good results:
  - Instruction N should not depend on N-1 (validate with `spu_timing`)
  - Execute `hbr` (Hint for Branch) for return instr. `"bi $0"`

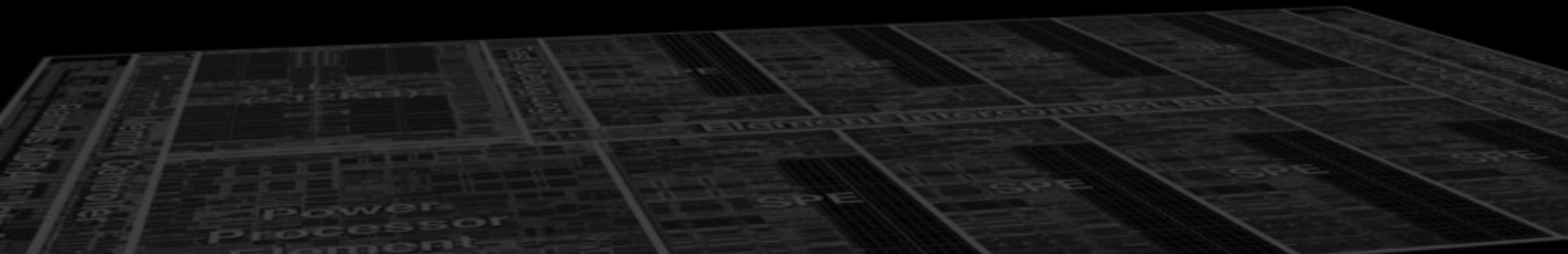
# Now what ?

- ...
- Profit ?
- Yes ! Bitslice DES is all about the S-boxes, nothing else needs to be optimized
- Key schedule, E-box, P-box, initial and final permutations, etc: all of these can be hardcoded and are zero-cost operations in a bitslice implementation



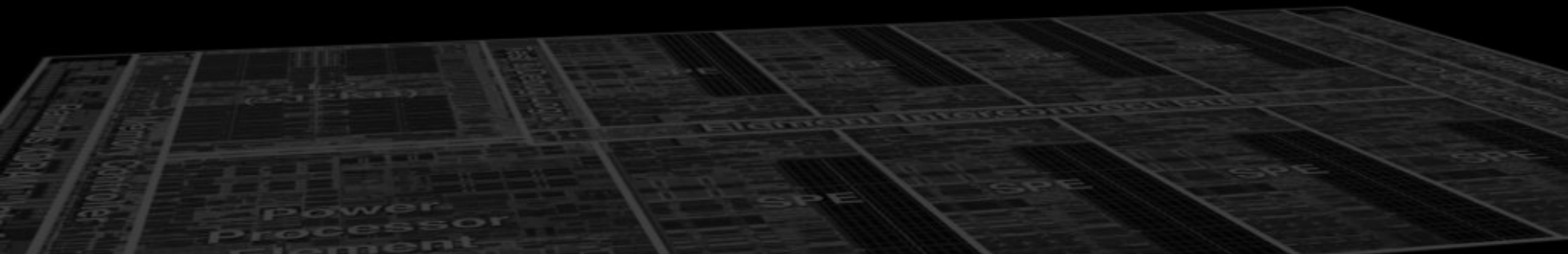
# What about the salt ?

- Used to swap some of the E-box output bits
- Again, can be hardcoded and is a zero-cost operation



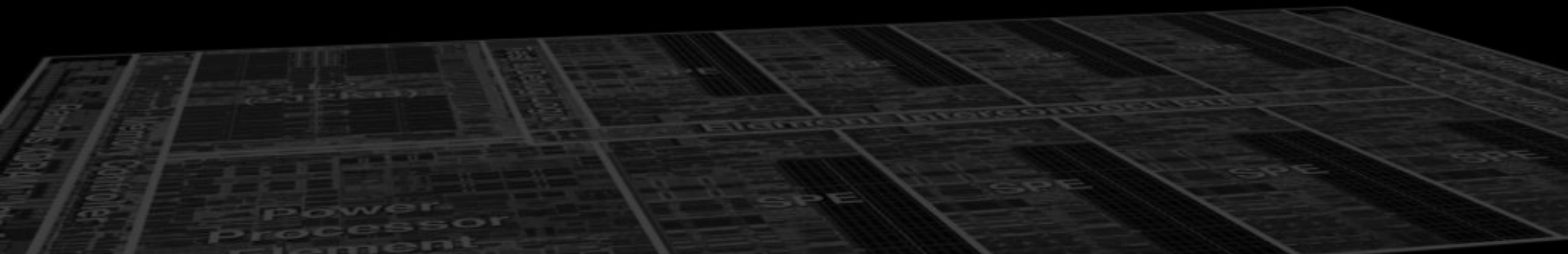
# Results

- PlayStation 3, Cell's 6 SPUs @ 3.2 Ghz:
- Bruteforcing speed =  
11.1 million password/sec
- Compare to 8.8 Mpw/sec for quad-core Q6700



# Performance / price

- Perf/price ratio 1.6x better than current best implementation:
  - PS3: street price US\$400: 27.8k passwd/sec/\$
  - JtR on Q6700: about US\$500 (stripped-down cluster node: mobo, CPU, RAM, PSU only): 17.6k passwd/sec/\$



# Performance / watt

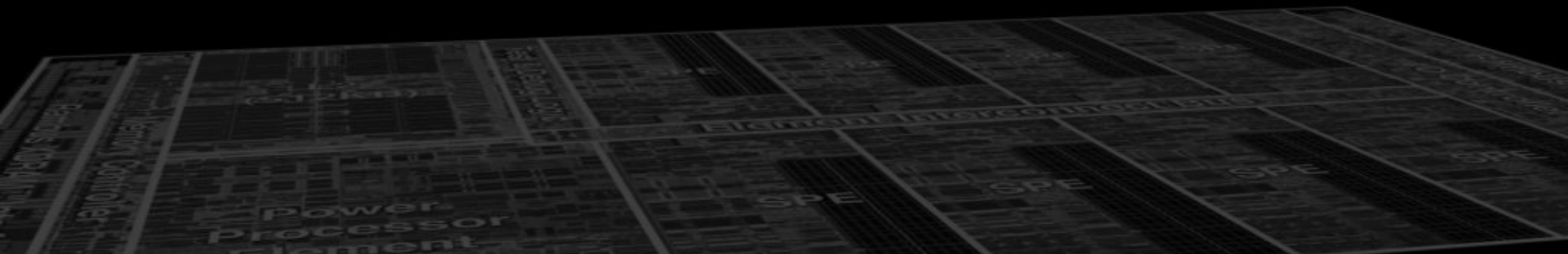
- Don't forget perf/watt – hardware prices are becoming negligible compared to energy costs
- Perf/watt ratio 1.5x better than current best:
  - PS3: about 130W (with 65nm Cell): 85.4k passwd/sec/W
  - JtR on Q6700: about 150W (stripped-down cluster node): 58.7k passwd/sec/W

# Conclusion

- Full printable password space =  $95^{**8} + \dots + 95^{**0}$
- A \$20k investment in 50 PS3s can bruteforce any password in 140 days worst case, 70 days average
- Assuming \$0.1/kWh, the energy cost would be \$2200 worst case, \$1100 average
- Possible further improvements: use the PPU (+17%), get down to 40 gates (another +12%)

# See also

- Slides + source code will be available at:  
[http://epitech.net/~bevand\\_m](http://epitech.net/~bevand_m)



EOF

