147

Optimising Prøst on ARM11

Thom Wiggers
Institute for Computer and Information Sciences, Radboud University, Nijmegen

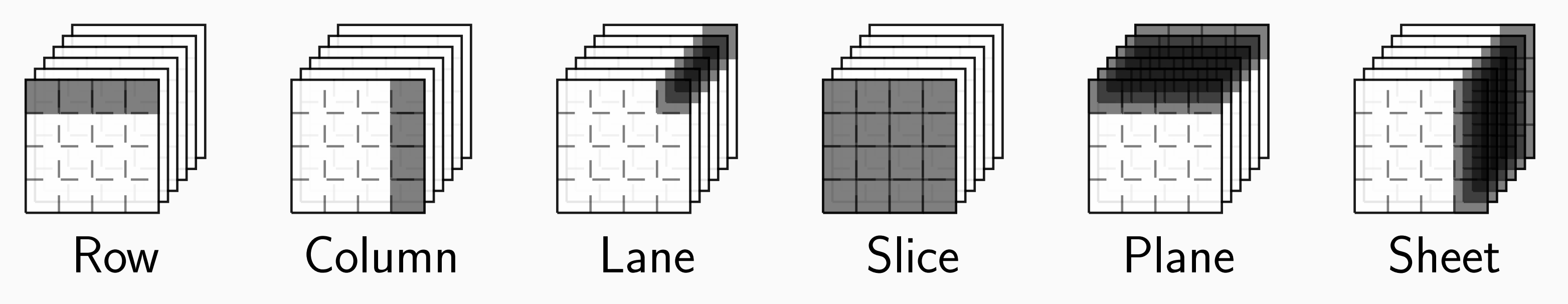
Prøst

PRØST was a contestant in the CAESAR competition for authenticated encryption. Authenticated encryption algorithms provide not only confidentiality, but also authenticity and integrity. High-speed implementations of ciphers competing in these competitions help asses the suitability for wide-spread deployment.
PRØST-128 has a 256-bit state s which is considered as a $4 \times 4 \times 16$ three-dimensional block

$$\mathrm{s} = \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{pmatrix}$$

where each $s_{x,y}$ is a 16-bit value. PRØST's authors call these *lanes*. The terms *row*, *column*, *slice*, *plane* and *sheet* for the other parts of the state are described below.
The permutation consists, in the PRØST-128 case, of 16 rounds. The round function $R_i : \mathbb{F}_2^{256} \to \mathbb{F}_2^{256}$ with $0 \leq i < 16$, can be defined as

$$R_i(x) = (\texttt{AddConstants}_i \circ \texttt{ShiftPlanes}_i \circ \texttt{MixSlices} \circ \texttt{SubRows})(x).$$



Nomenclature for state parts

MixSlices

Mix up the slices according to a matrix, this gives operations such as:

$$\begin{aligned} s'_{0,0} &= s_{0,0} \oplus s_{1,0} \oplus s_{1,3} \oplus s_{2,2} \oplus s_{3,0} \oplus s_{3,2} \oplus s_{3,3} \\ s'_{0,1} &= s_{0,1} \oplus s_{1,0} \oplus s_{2,3} \oplus s_{3,0} \oplus s_{3,3} \\ &\cdots \\ s'_{2,0} &= s_{0,2} \oplus s_{1,0} \oplus s_{1,2} \oplus s_{1,3} \oplus s_{2,0} \oplus s_{3,0} \oplus s_{3,3} \\ &\cdots \end{aligned}$$

This system has 72 xors, using the Boyar-Matthews-Peralta heuristic we were able to get a system with only 48 xors.

Boyar-Matthews-Peralta SLP heuristic

- Consider your program as an input matrix M;
- Initialise matrix S to $([1, 0, \cdots], [0, 1, 0 \cdots])$ to represent your inputs;
- Define $Dist[i]$ as the minimum number of additions of rows in S to get $M[i]$;
- Choose new combination by lowest norm of the $Dist$ vector.

$$\begin{aligned} y_0 &= x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4 \\ y_1 &= x_0 \oplus x_1 \oplus x_2 \oplus x_3 \\ y_2 &= x_0 \oplus x_1 \oplus x_2 \oplus x_4 \\ y_3 &= x_2 \oplus x_3 \oplus x_4 \\ y_4 &= x_0 \oplus x_4 \end{aligned} \qquad M = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad S = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ & & \cdots & & \end{pmatrix} \begin{array}{l} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ t_0 = x_1 \oplus x_3 \\ \cdots \end{array}$$

Finding the true shortest program using SAT

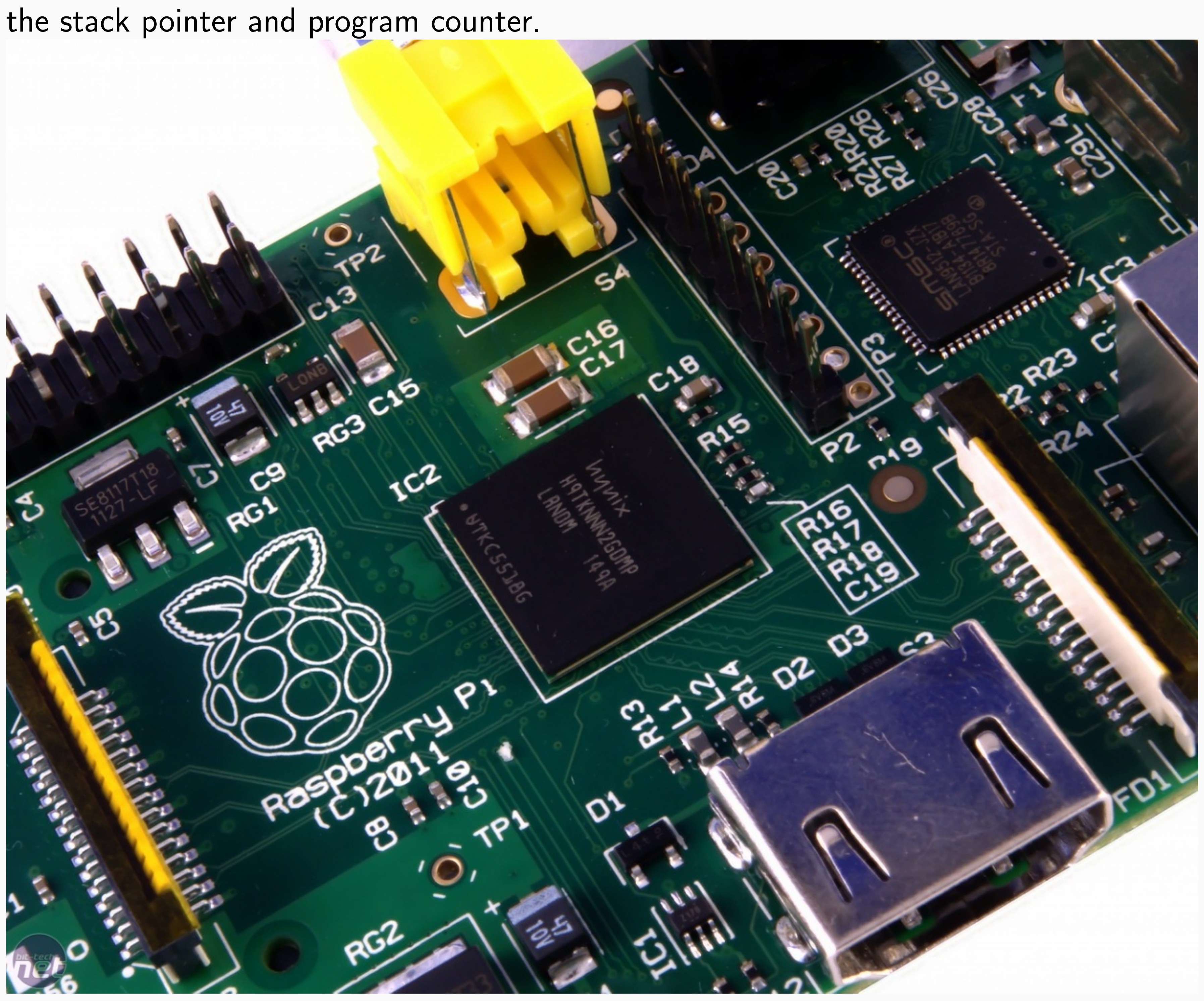
- Input your program as an $(m \times n)$ matrix and decide on a number of lines k;
- Define matrices $B : (m \times k)$ for the inputs used, $C : (k \times k)$ for the intermediates used and mapping f from intermediates to outputs;
- Apply constraints that only can be satisfied by valid programs;
- If the problem is satisfiable, extract the program from B, C, and f.
- Repeat with lower k until UNSAT.

Each line has exactly two inputs: $\beta_1 = \bigvee_{0 \leq i < k} \mathrm{exactly}_2(b_{i,1}, \cdots, b_{i,n}, c_{i,n}, \cdots, c_{i,i-1})$

$$B = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad C = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{aligned} t_0 &= x_0 \oplus x_4 \\ t_1 &= x_1 \oplus t_0 \\ t_2 &= x_2 \oplus t_1 \\ t_3 &= x_3 \oplus t_2 \\ t_4 &= x_4 \oplus t_3 \\ t_5 &= t_2 \oplus t_4 \end{aligned} \qquad f = \begin{cases} 0 \mapsto 3 \\ 1 \mapsto 4 \\ 2 \mapsto 2 \\ 3 \mapsto 5 \\ 4 \mapsto 0 \end{cases}$$

ARM11

ARM11 is range of microprocessors that are commonly found in for example smart phones, the Raspberry Pi and the Nintendo 3DS. Hundreds of millions of these chips are deployed worldwide. ARM11 is an implementation of the ARMv6 architecture. This is a 32-bit architecture that provides the programmer with 16 registers, including the stack pointer and program counter.



Some assembly optimisation tricks

```
x1  = mem16[address_a]
x1 += 10
x2  = mem16[address_b]
x2 += 10
x3  = mem16[address_c]
x3 += 10
```

Latencies slow execution down (12 cycles)

```
x1 = mem16[address_a]
x2 = mem16[address_b]
x3 = mem16[address_c]
x1 += 10
x2 += 10
x3 += 10
```

Hiding load latencies (6 cycles)

Equivalent programs, but different execution times

```
# a = s_{0,0}, b = s_{0,1}, c = s_{0,3}
# a' = c ^ (a & b)
a_and_b, c_and_d = mem64[address_of_s]
newa  = a_and_b & (a_and_b >>> 16)
newa ^= c_and_d
# only write back the lower 16 bits
mem16[address_of_s] = newa
```

An example part of SubRows *using two lanes in one register and wide loads.*

Benchmark results

Implementation	APE	COPA	OTR
Reference (C)[a]	2,976,123	2,402,577	1,569,582
ARM Assembly	1,900,274[a]	1,648,407[a]	848,100[b]
Improvement	36%	28%	46%

[a] Compiled with `gcc -funroll-loops -fno-schedule-insns -O3 -fomit-frame-pointer`
[b] Compiled with `gcc -O3 -fomit-frame-pointer`

More information

Find the developed software and my thesis at `https://thomwiggers.nl/proest/`.

Radboud University