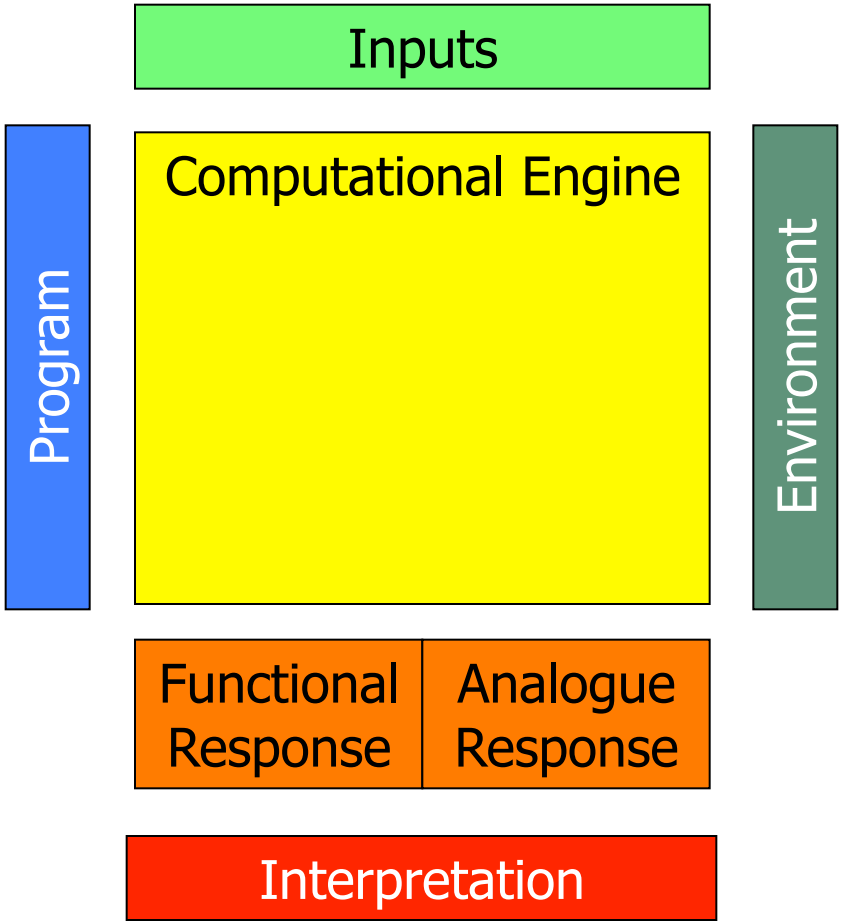
THE UNIVERSITY of York

DISTAR Computing
Digital Stimulus Analogue Response
(inspired mostly by crypto)

John A Clark

Outline Model of Computation

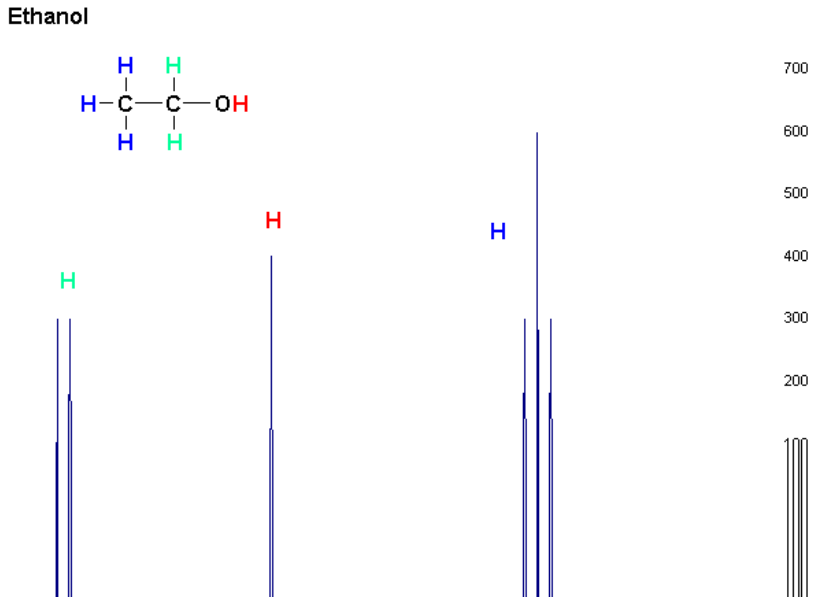


This is an outline model of computation which allows us to identify where to put effort.

There are many choices over what to seek control.

Going to take a general view of analogue: radio frequency, timing, power, heat, …

Genetic algorithms and NMR

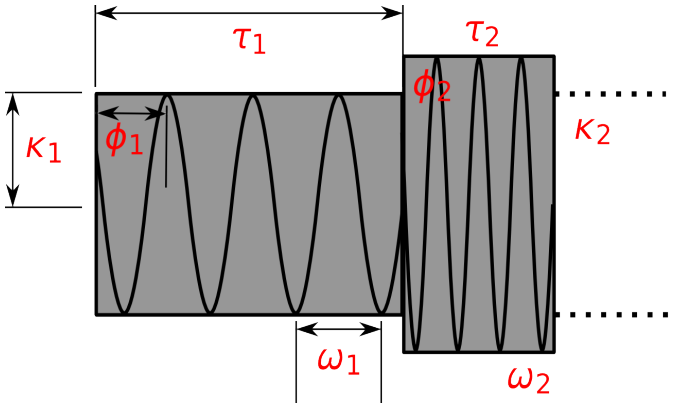


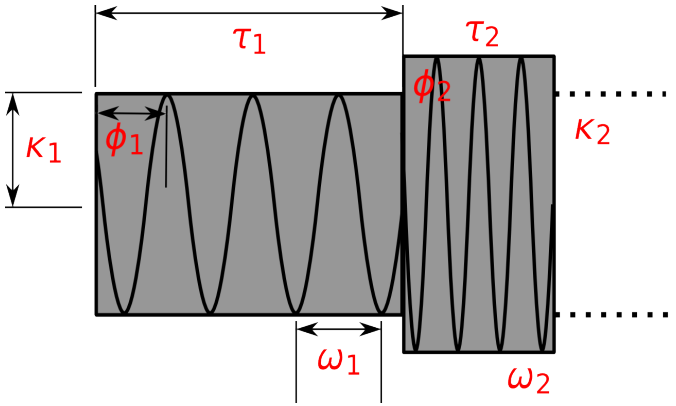
What happens if you RF pulse a substance in magnetic field?

Over various pulsing frequencies you get an associated RF response from the substance depending what it contains.

Usually easy to identify substance composition there is a single molecule type but if there are several the composition is more complicated.

Genetic algorithms and solid-state NMR





κ1	τ1	ω1	κ2	τ2	ω2	● ● ●	κr	τr	ωr

Genome (individual) here is decoded as a program to generate the indicated RF pulse sequence.

Powdered substrate responds to the pulse sequence RF pulse sequence with its own RF response in a way we hope is revealing in some way (i.e. characterises its composition).

This is an example of evolving a program to induce analogue responses of a desired form (BTW: we have broken existing theory.)

Seeking Control Over Timing Outputs

- David reported earlier on timing avalanches and PRNGs: this an attempt to control both:
  - Functional outputs (does it work like a good PRNG, e.g. pass randomness tests?)
  - Timing properties – to the extent that the execution times look 'random': the idea here is that NO (little) information should leak via these times.
    - Here it is simulated time but this is still a timing property of **a** system – you would get different programs if you ran this with real time measurements on real processors – but the principle is the same.
- It does so by evolving a program seeking measurable functional properties with desirable induced timing responses properties.

Seeking Control Over Timing Outputs

- But can you find a program that solves a problem using only the timing properties.
- Let's consider a pattern classification problem.

Loosely

Take two sets of data A={r_1, r_2,…r_n} B={s_1, s_2, …, s_n}.

Can you find a program P(data) such that

Timing (P(r_j)) < Timing (P(s_k)) for all j, k

Effectively, can timing act as an efficient and effective classifier?

Seeking Control Over Timing Outputs

- Program space is limited subset of expressions using integers with a primitive simulated timing model.

Instruction	Timing Model
MUL(a,b)	Hamming(a)*Hamming(b)
ADD(a,b)	Hamming(a)+Hamming(b)
SUB(a,b)	Hamming(a)-Hamming(b)
SHIFTL	1
SHIFTR	1

Problem: A={0,…,127} B={128,…,255}

Seeking Control Over Timing Outputs

- Example program evolved……

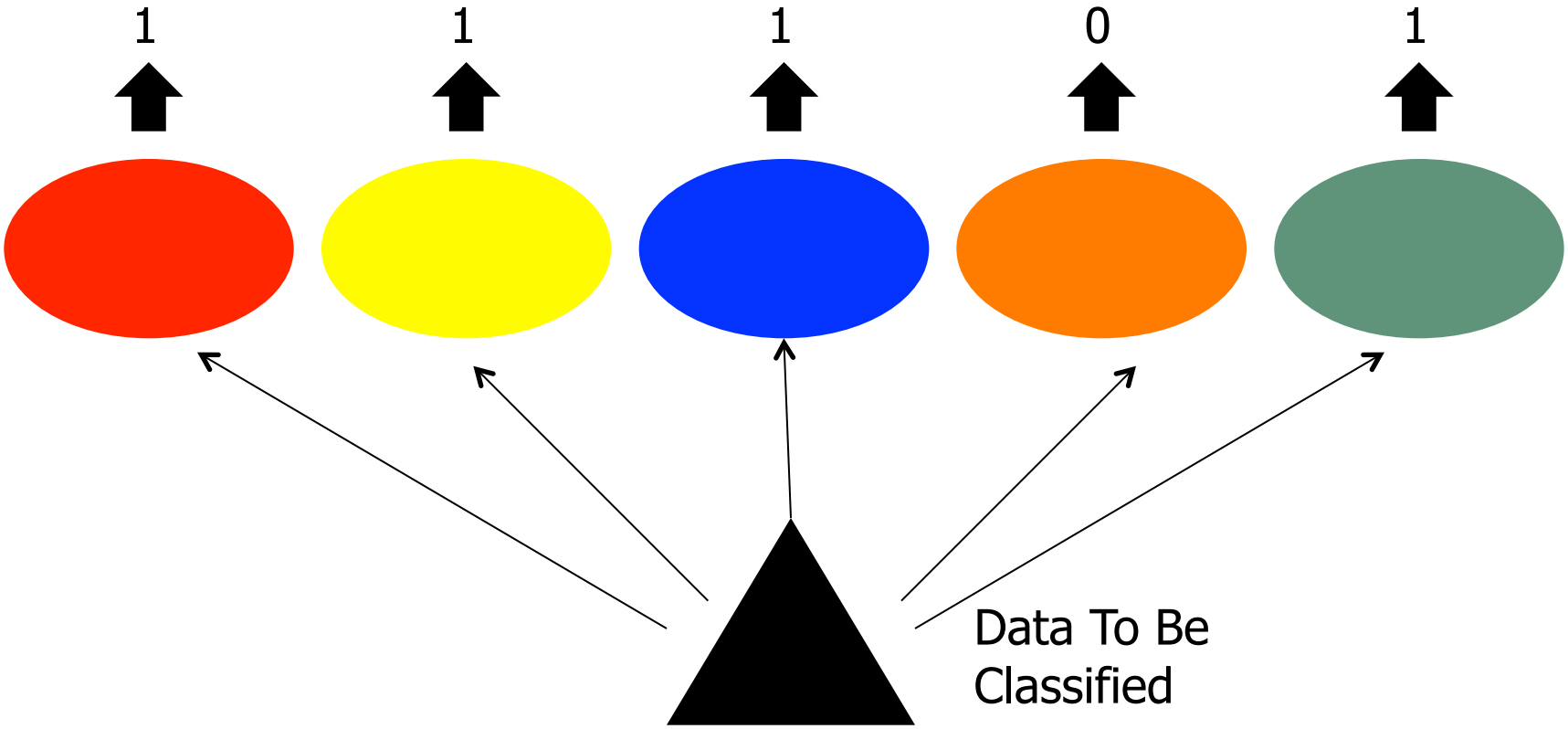
```
Best Individual of Run:
Subpopulation 0:
Evaluated: true
Fitness: Standardized=914.0 Adjusted=0.0010928961748633888
Hits=255
Tree 0:
 (* (* (* (* (* (SHIFTR  (SHIFTR  (SHIFTR
    (SHIFTR  (SHIFTR  (SHIFTR  (SHIFTR  x)))))))
    x) x) x) x) x)
```

May also be interesting things happening functionally regarding overflow.

Problem: A={0,…,127} B={128,…,255}

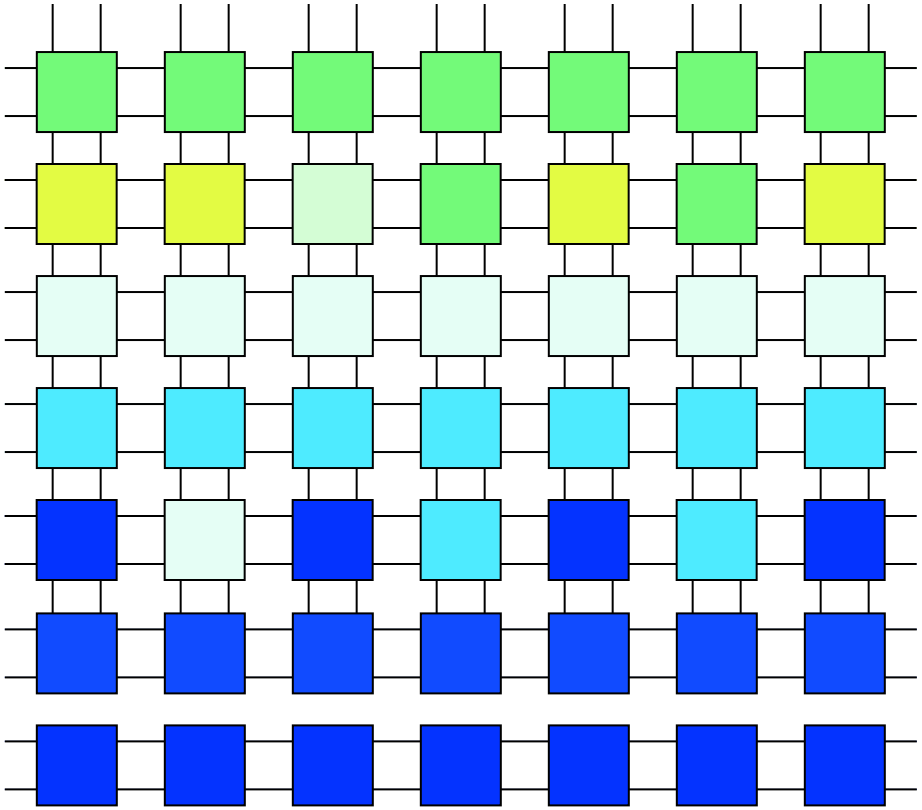
Seeking Control Over Timing Outputs

Possible that for complex tasks an ensemble of timing oriented classifiers may be best



Possible that this approach may also be power efficient, if it works…

Heat Profile as IO



Can we find a program P such that when you run it on data D classifies D as either A or B via the heat profile of the chip. ….

Top Hotter Than Bottom=>A

Bottom hotter than Top => B

Toggle frequency as a proxy for heat.

Idea from 2004 tried in 2007 and it failed. But really this is an extraordinarily bizarre goal. Why not have (evolve) a more sophisticated interpretation of the heat profile? (See also more recent Cambrideg work on TOR system.)

IDS in MANETs (Sevil Sen)

Table 1. GP parameter settings

Objective	Find a computer program to detect flooding and route disruption attacks against MANETs
Function set	+,-,*, /, pow, min, max, percent sin, cos, log, ln, sqrt, abs, exp, ceil, floor, and, or, comparison operators
Terminal set	The feature set in Appendix A
Populations Size	100
Generations	1000
Crossover Probability	0.9
Reproduction Probability	0.1
Tournament Size	7

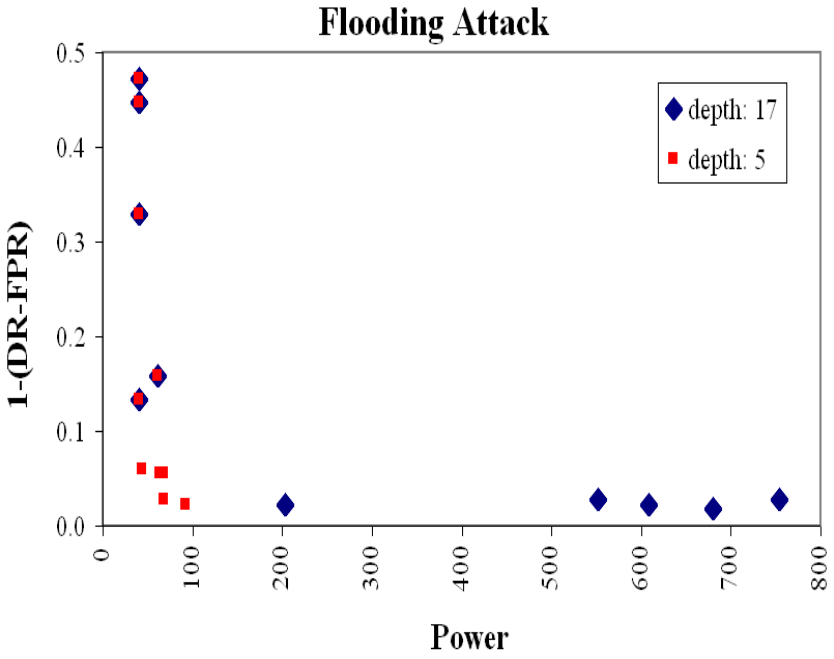
$$Fitness = detection\ rate - false\ positive\ rate \quad (1)$$

IDS in MANETs (Sevil Sen)

Table 2. Performance of the Genetic Programming technique on simulated networks

Network Scenarios	**Flooding Attack**		**Route Disruption Attack**	
	DR	FPR	DR	FPR
low mobility low traffic	99.81%	0.34%	100%	0.51%
low mobility medium traffic	99.24%	1.94%	100%	0.99%
medium mobility low traffic	99.95%	0.36%	97.06%	0.46%
medium mobility medium traffic	99.89%	1.88%	100%	0.88%
high mobility low traffic	99.79%	0.66%	100%	0.52%
high mobility medium traffic	98.62%	1.83%	100%	0.84%

IDS in MANETs (Sevil Sen)



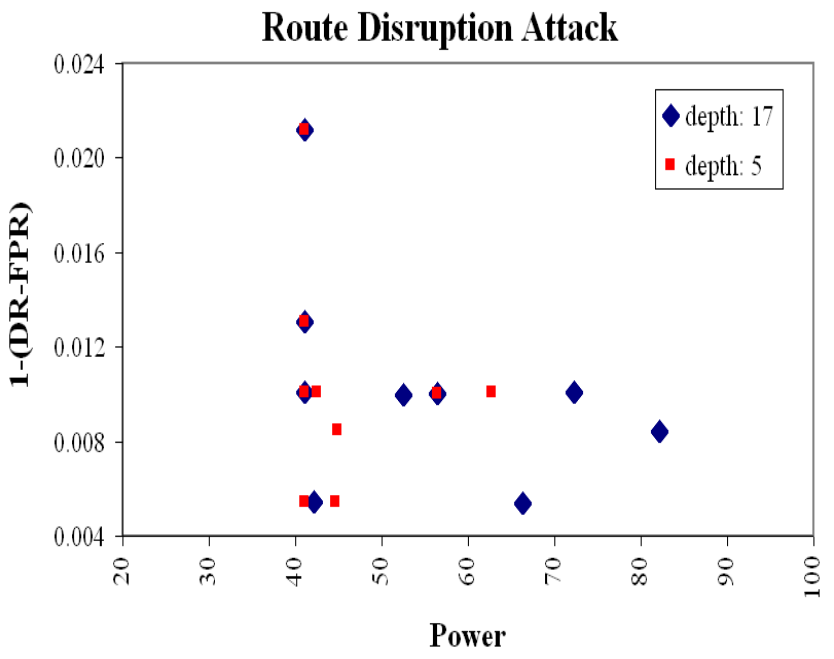


Fig. 2. Classification accuracy and energy consumption of the optimal evolved programs

IDS in MANETs (Sevil Sen)

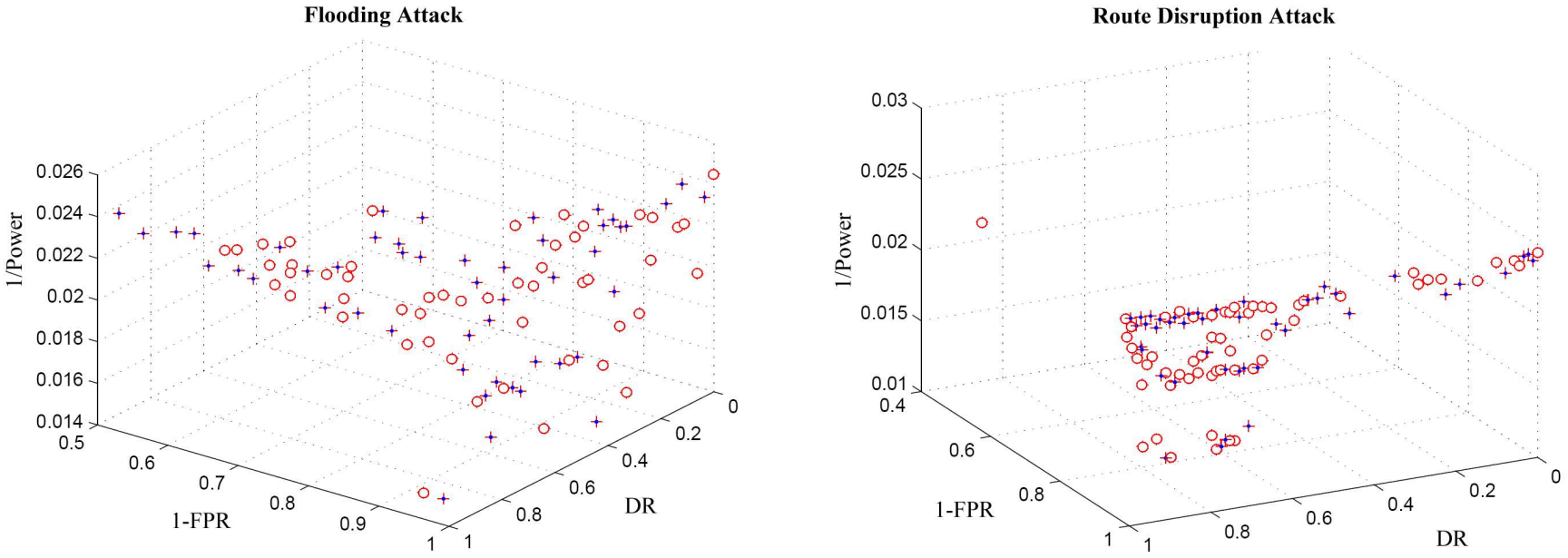


Fig. 3. 3D-Pareto front for detection of each attack with the three objectives: detection rate, false positive rate and energy consumption

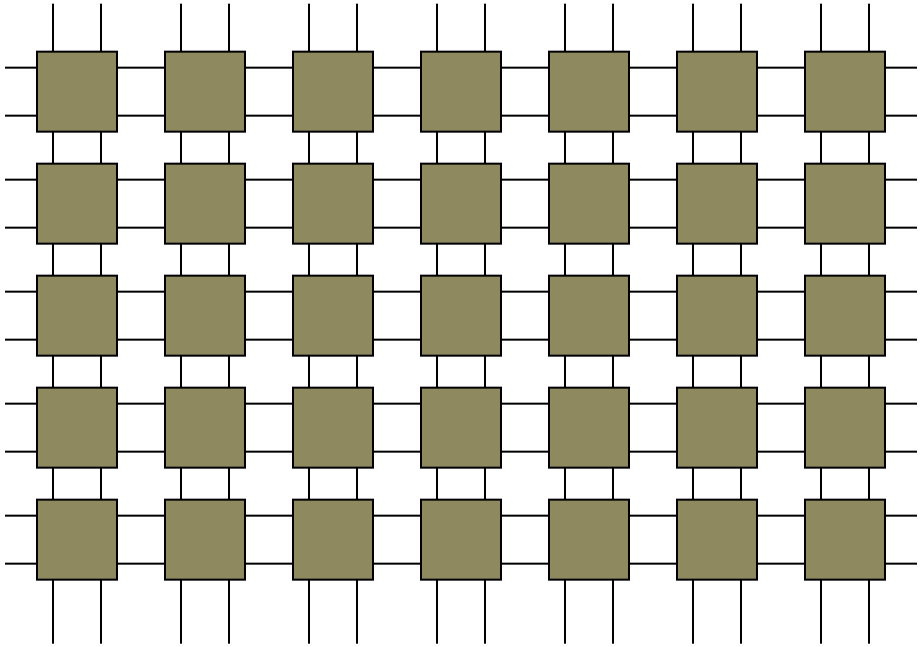
IDS in MANETs (Sevil Sen)

Table 3. Example programs evolved by MOEA for each attack

Attack Type	**Evolved Program**	**DR**	**FPR**	**Energy Usage**
Flooding	(frw_aodvPs * frw_aodvPs) > (4log(neighbours) + 5updated_routes)	98.65%	1.23%	65.42
Route Disruption	((2updated_routes - 2recv_aodvPs + active_routes) * recv_rrepPs > (recv_aodvPs + updated_routes)	100%	0.63%	43.05
Both	(((updated_routes * init_aodvPs) $\leq$ frw_rreqPs) && (init_rrepPs $\neq$ recv_rrepPs) && (exp(updated_routes) $\neq$ recv_rrepPs)) $\|\|$ (updated_routes < frw_rreqPs)	93.29%	4.65%	50.14

Environment Manipulation

- Adrian Thompson did some really cool (or hot) stuff in the late 1990s by evolving FPGA programs (cell matrix configurations) using Genetic Algorithms.
- Evolved programs to distinguish 1kz and 10 kHz signals using the unconstrained dynamics of the chip (switch off lock step).
- Program worked for around 20 minutes until chip got hot!!!!



Environment Manipulation

- Consider RAM chips.
- We tell lies about how they work to our students.
- We tell them that if we remove the power then the contents disappear.

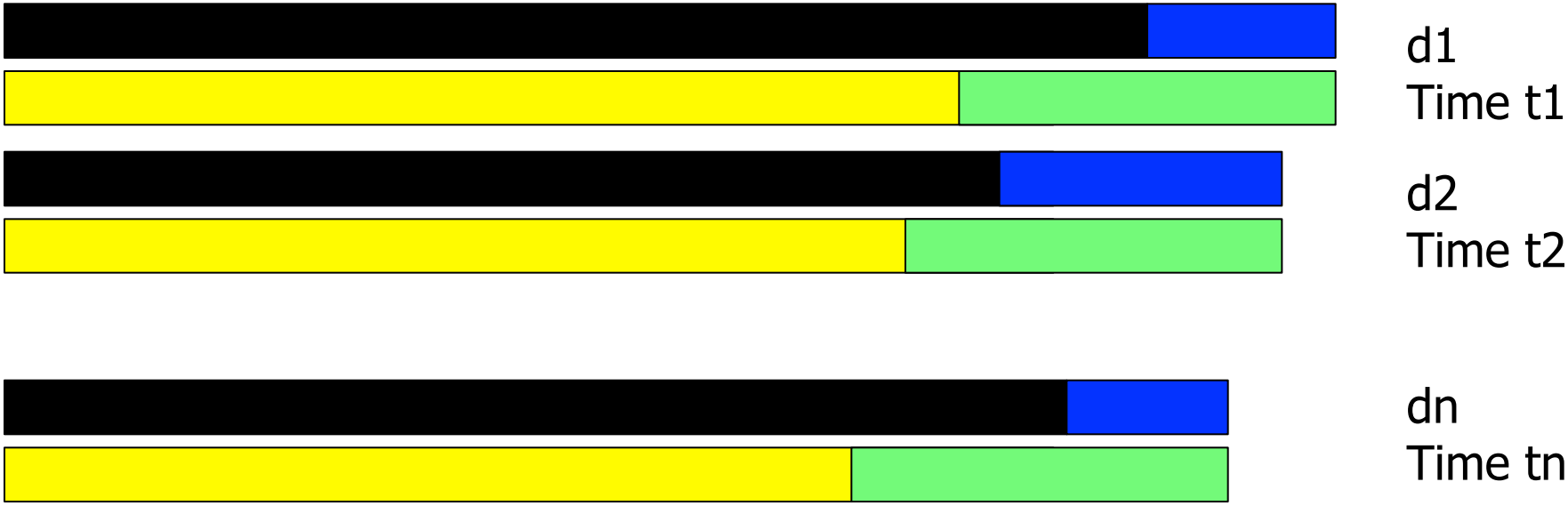
- But for some memory chips if you reduce the temperature to say -40 C and then remove the power, it powers up in *almost* the state it was in before you remove the power.
- This could allow you to bypass security mechanisms that boil down to “pulling the plug if you detect tampering”.
- More general point is that the info properties of hardware are different under different environmental conditions.

Interpretation Needed

- Square and multiply with key (exponent) $k_0k_1k_2$ etc.

```
s0 := 1
for i = 0 to n-1
   Ri := (if ki = 1  then (si * y) mod m  else si)
   si+1 := (Ri * Ri) mod m
endfor
return Rn-1
```

Kocher's Timing Attack



Suppose we have the total times for exponentiation t1, t2,…,tn for the identified data items d1, d2, …, dn.

Assume you can calculate the time for the first round under the assumption that the first key bit is 0 (blue) and under the assumption that the first key bit is 1 (green). The time for the remaining rounds is then calculated (black and yellow respectively

Kocher's Timing Attack

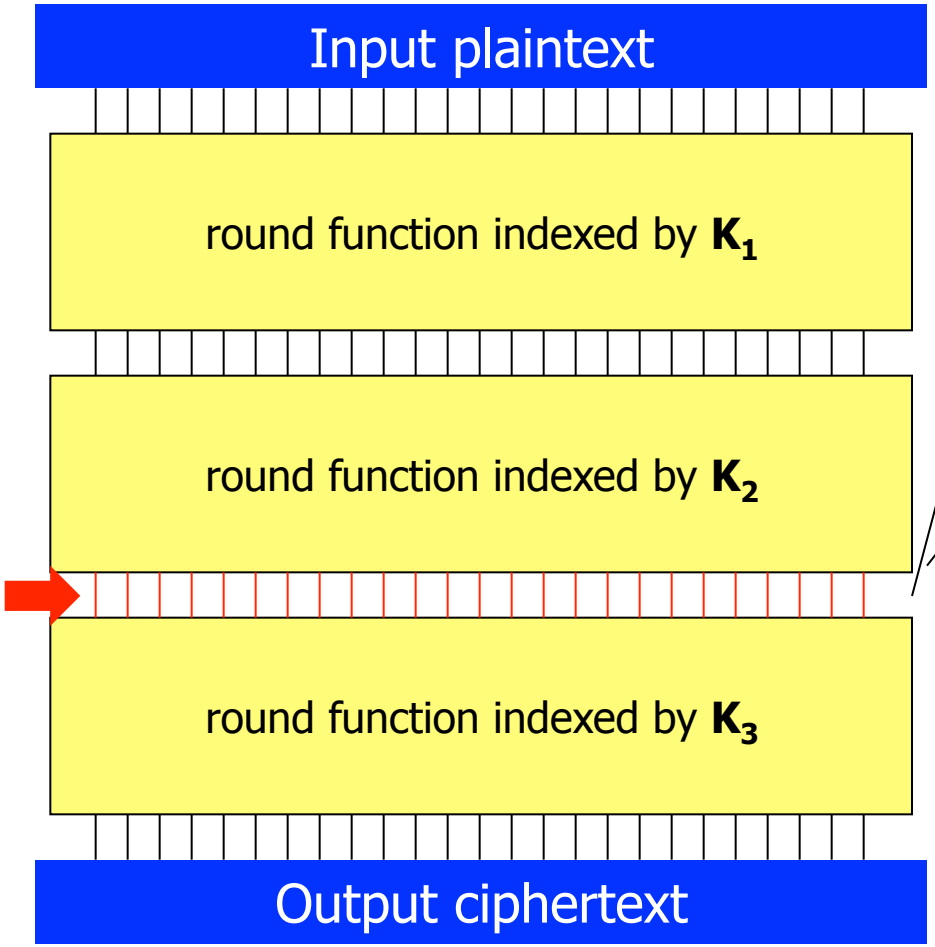
If the variance of the BLACK remaining times is less than the variance of the YELLOW remaining times then the first bit WAS actually a 0. Otherwise the first bit WAS actually a 1. Now repeat the process for the next round (in the context of the choice you have now made)….

Strictly this can go wrong (detectably) and some degree of backtracking is needed.

This is an example of INTERPRETATION OF THE TIMING MEASUREMENTS.

Let's Do the Time Warp Again

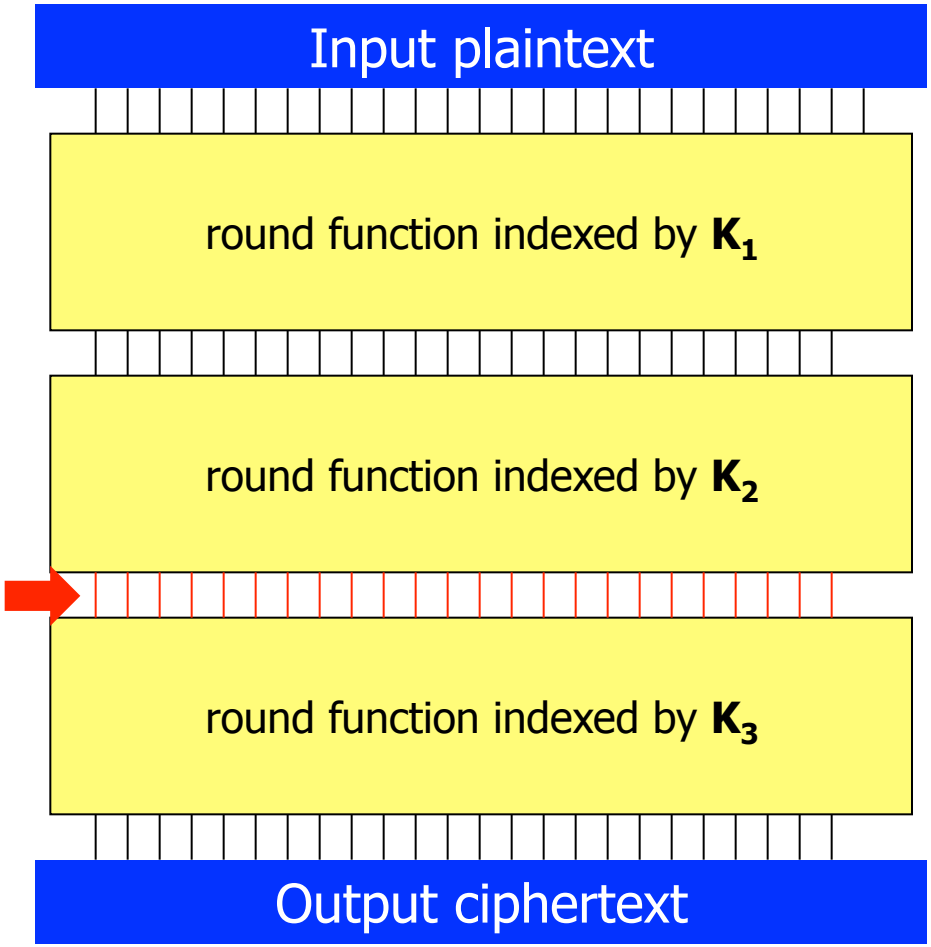
- Simulations of this attack work even when the timing model for multiplication is randomly generated lookup table (e.g. mean 1000ns with a small variance) Thanks to Susan Stepney).
- So why not EVOLVE THE TIMING MODEL?
  - This is a fairly radical step, but we can leverage the fact that we can simulate: we are not beholden to actual hardware.
  - With earlier example we could evolve the program and the timing model together.



If you know K_3 then you know all the intermediate text here, because you can invert the round precisely.

If you know a subset of the key K_3 then you know a subset of the the intermediate text here.

Suppose if you know the final 6 bits of K_3 you can reverse engineer the FIRST intermediate bit value.

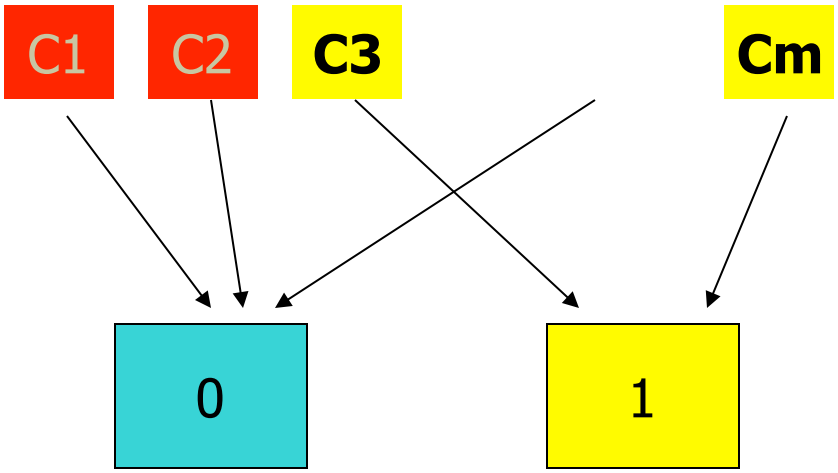


So for each choice of final 6 bits you get a predictor for the value of that bit given a particular ciphertext.

For each such guess of 6 key bits if you guess the 6 bits correctly then the predicted bit for each ciphertext ACTUALLY TAKES THE VALUE its had during the encryption.

If there is an error in the key guess this process essentially randomises the result (half right and half wrong).

Predictor acts as partitioner



$$\Delta_D[j] = \left(\frac{\sum_{i=1}^{m} D(C_i, K_s) T_i[j]}{\sum_{i=1}^{m} D(C_i, K_s)} \right) - \left(\frac{\sum_{i=1}^{m} (1 - D(C_i, K_s)) T_i[j]}{\sum_{i=1}^{m} (1 - D(C_i, K_s))} \right)$$

Monitor power traces

C_1: $T_1[1]$ $T_1[2]$ $T_1[3]$ $T_1[n]$

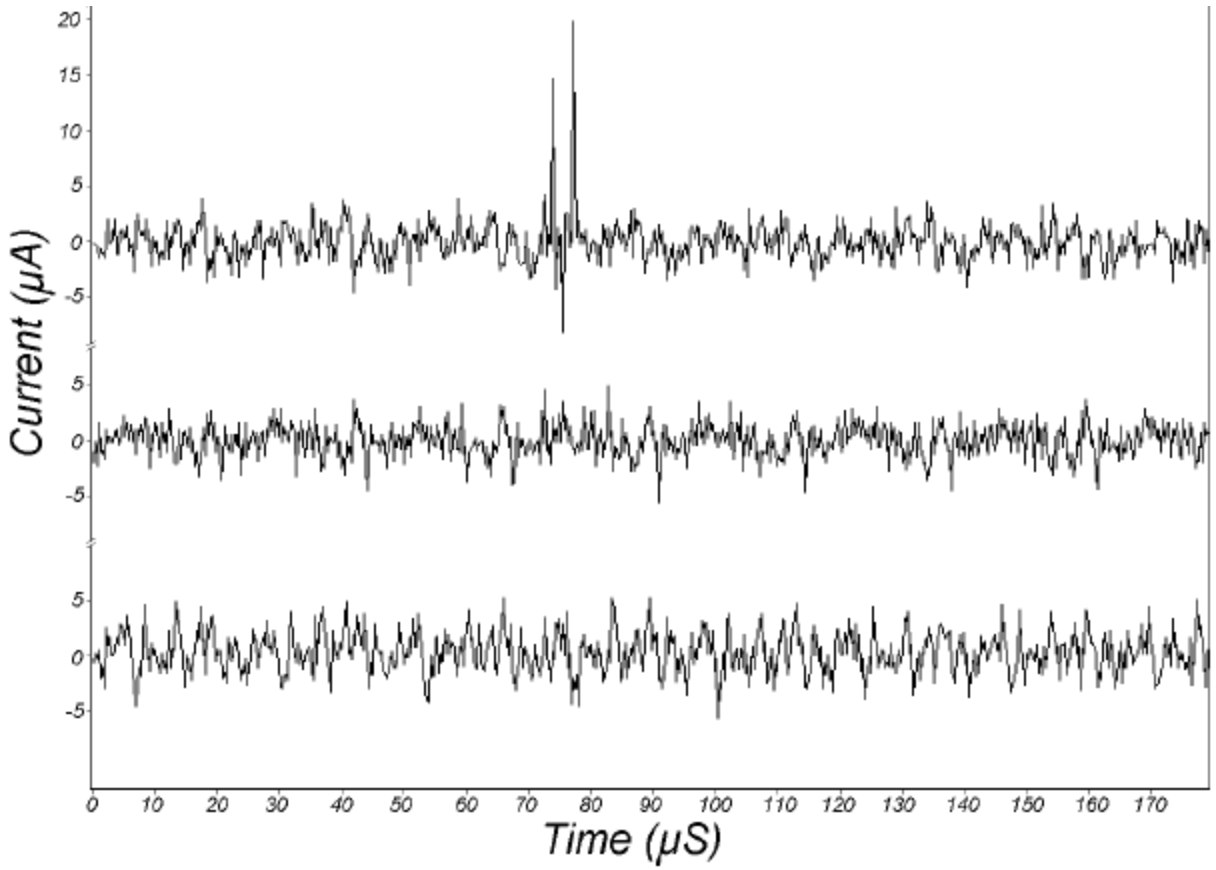
C_2: $T_2[1]$ $T_2[2]$ $T_2[3]$ $T_2[n]$

C_m: $T_m[1]$ $T_m[2]$ $T_m[3]$ $T_m[n]$

$$\Delta_D[j] = \left(\frac{\sum_{i=1}^{m} D(C_i, K_s) T_i[j]}{\sum_{i=1}^{m} D(C_i, K_s)} \right) - \left(\frac{\sum_{i=1}^{m} (1 - D(C_i, K_s)) T_i[j]}{\sum_{i=1}^{m} (1 - D(C_i, K_s))} \right)$$

Plotting the correlations

Utter genius!!!!!



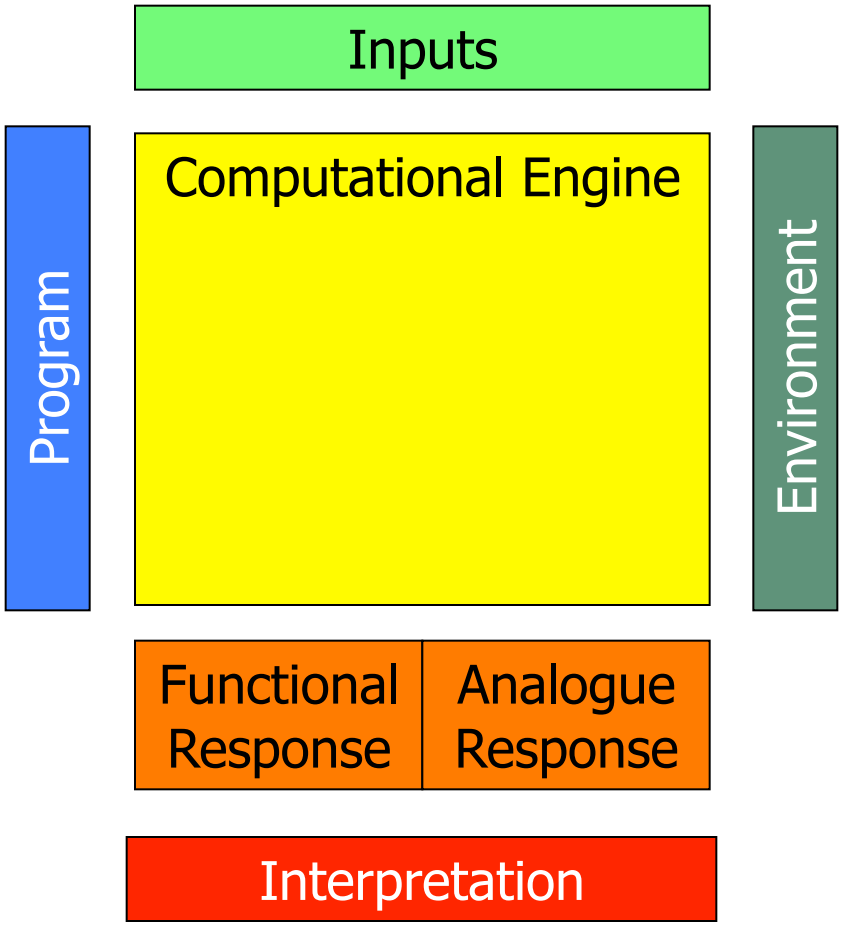
correct subkey guess

incorrect subkey guess

incorrect subkey guess

[Kocher 1999, fig 4]

Outline Model of Computation



So if we are to exploit analogue phenomena we may need to be eclectic and radical in what we seek control over.

It would not be outrageous to seek to control simultaneously the inputs, the program, the timing model and the interpretation function for example.

Breaking the Model: finalisation and a taxonomy of security attacks. John A. Clark, Susan Stepney, Howard Chivers. REFINE 2005