Efficient Use of Random Delays in Embedded Software

Michael Tunstall & Olivier Benoit

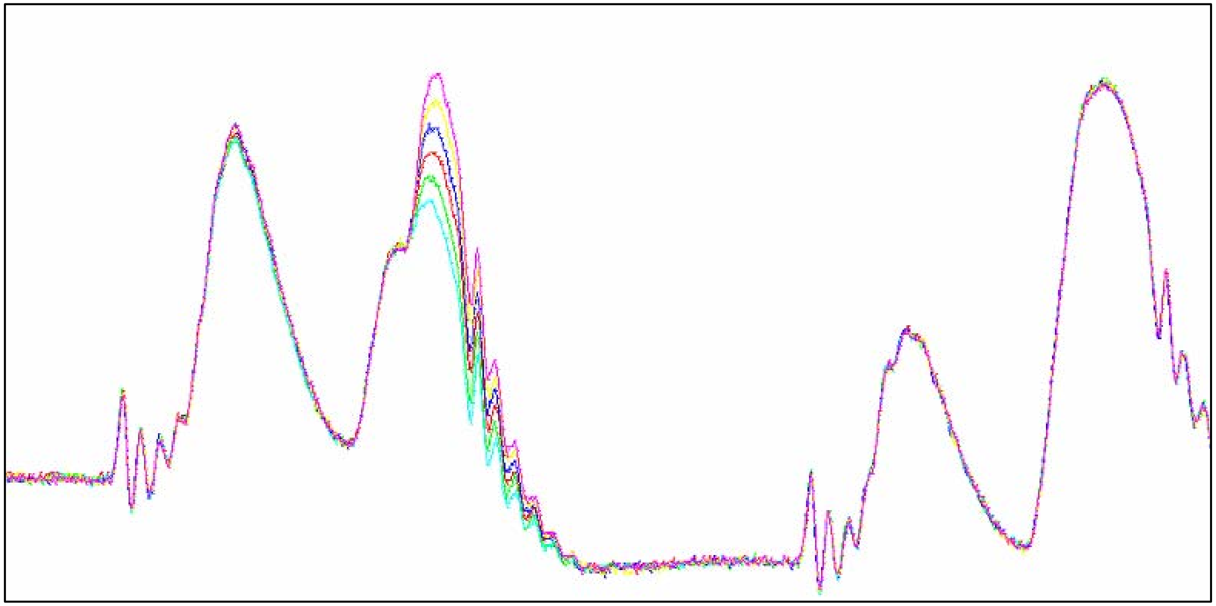
What are Random Delays?

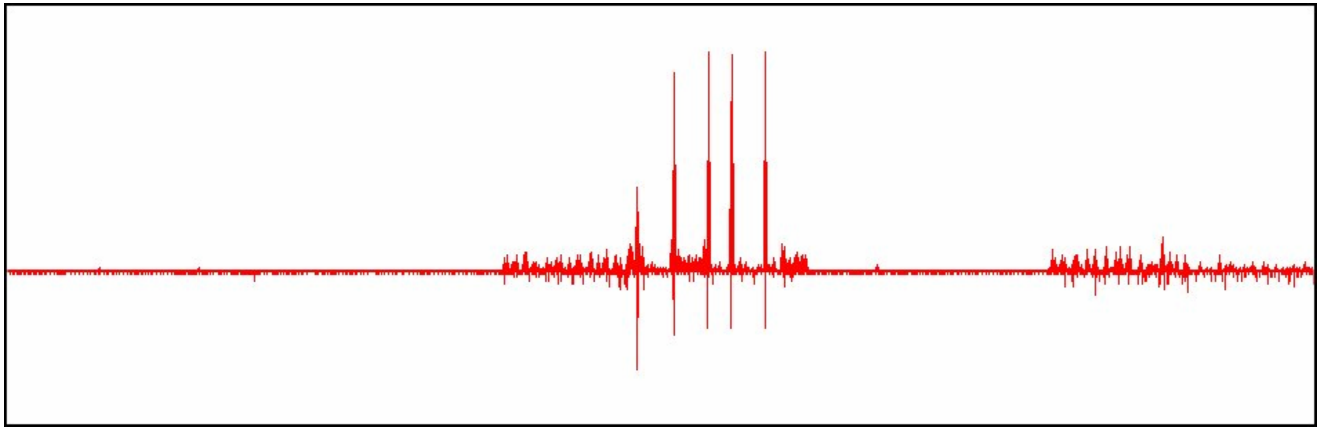
- Dummy functions inserted into embedded software that have no purpose other than to pause computation for a random period of time. E.g.

```
    mov   a, RND
    mov   r0, a
 Delay_Label:
    djnz  r0, Delay_Label
```

- Software Random delays will be considered in this presentation.
- Hardware versions exist that will ignore or repeat CPU instructions (see Clavier *et al.*, 2000).

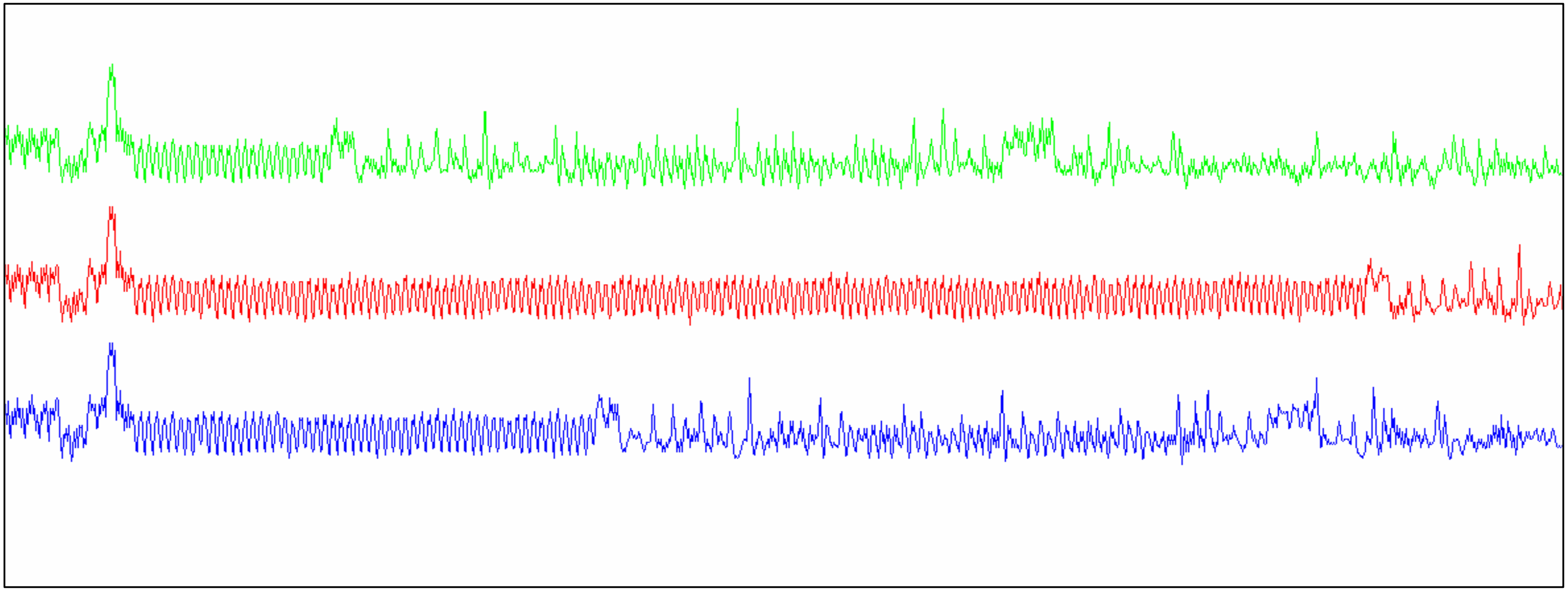
Power Attack Countermeasure





- Statistical analysis of small differences in power consumption to predict intermediate states.

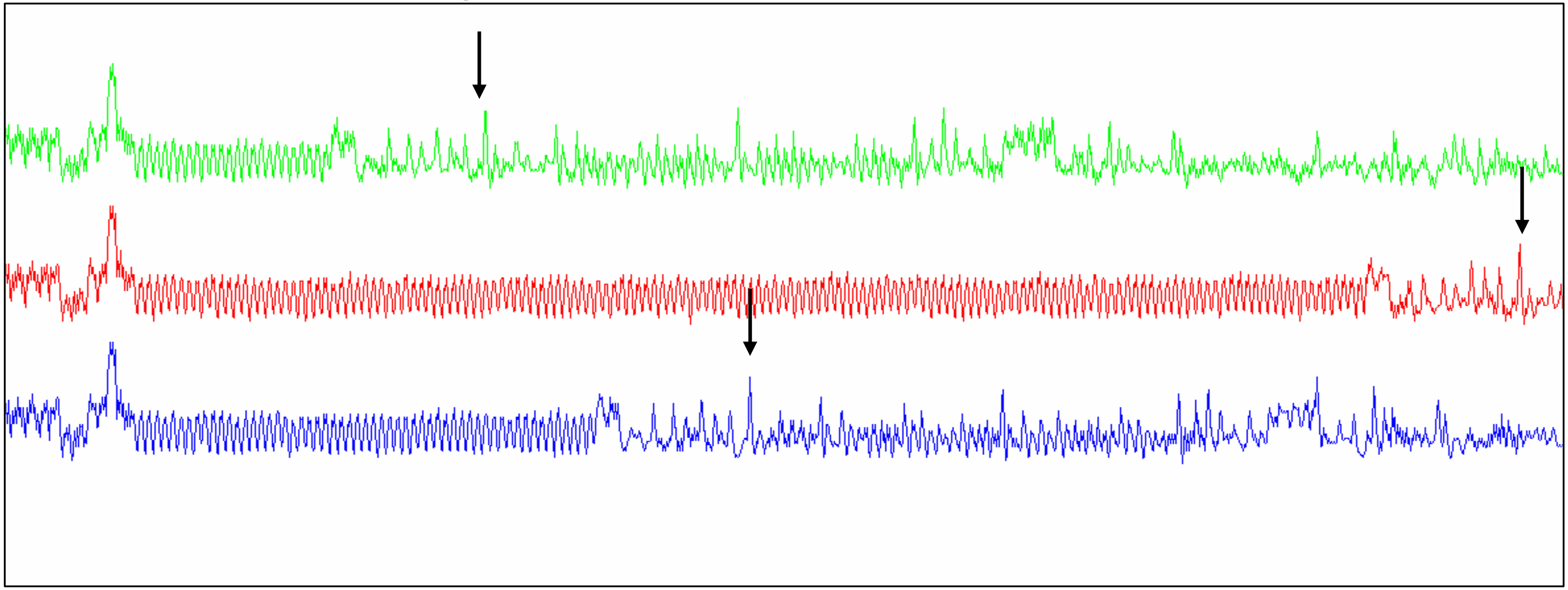
Random Delays



- Random delays in software desynchronise events.
- An attacker is obliged to resynchronise events *a posteriori*.

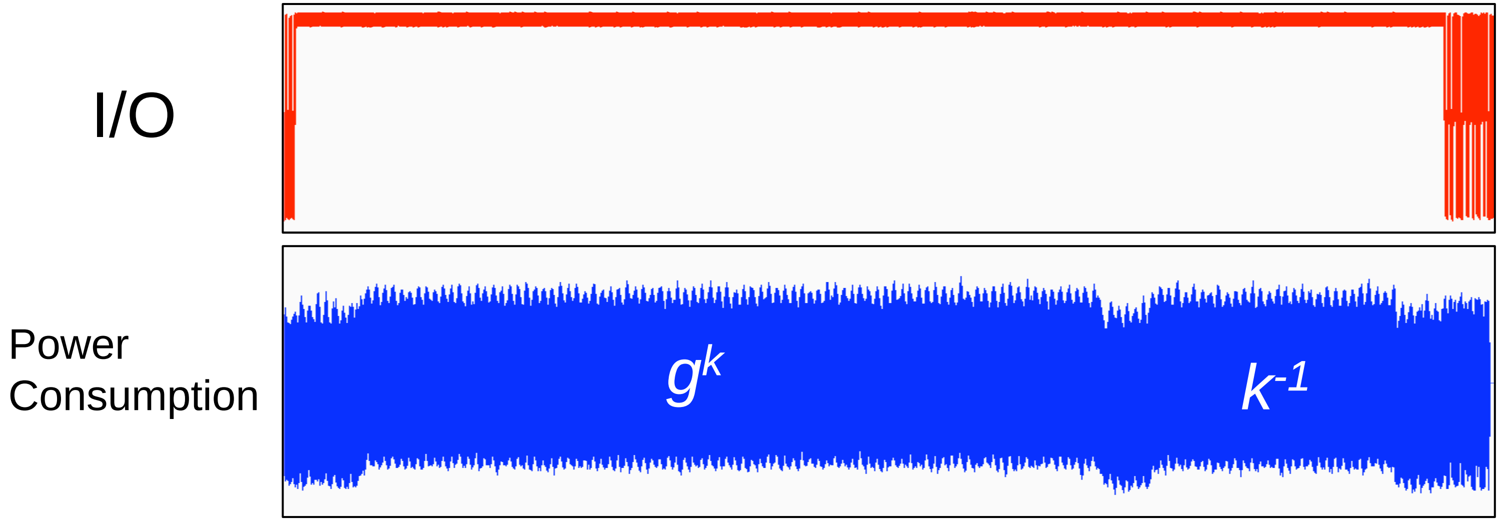
Fault Injection

- Injecting Faults to retrieve information on secret/private keys or to compromise the security of the operating system.
- Random delays provide a moving target, an attacker must wait until fault and target coincide.



Dynamic Resynchronisation

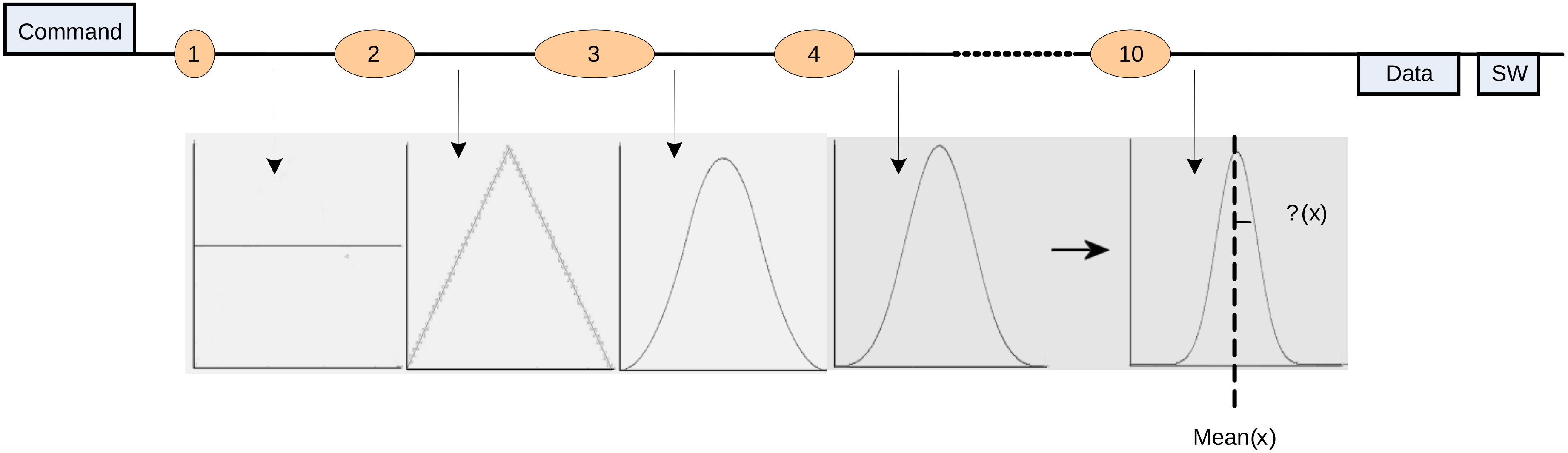
- Events such as I/O, programming EEPROM, coprocessor usage can allow dynamic resynchronisation. E.g. a DSA on a smart card:



- It is considered prudent to include a random delay whose length is distributed over a large interval after each event.

Random Delays Within a Command

- Random delays occur at numerous points within a command.
- Typically, the length of each random delay is uniformly distributed.
- To prevent DPA it is prudent to include a delay between every sub-function of a round function in a block cipher, so only a local resynchronisation is possible.



Proposed Optimisation

- Modify the distribution of the lengths of random delay to increase the standard deviation and decrease the mean cumulative delay.
  - The delays are an overhead to the computation.

- Random values generated on chip will be uniformly distributed.
- This can be modified by using a uniformly distributed random to look up values in a table. The frequency of entries in a table dictates the distribution, i.e.

```
{ 0, 0, 0, 0, 0, 1, 1, 1, 1, 2, 2, 2, …}
```

- Number of entries needs to be a power of 2 for ease of use.

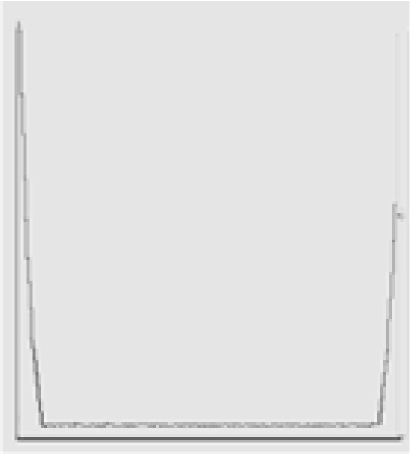
Proposed Optimisation

- The number of entries in look up table was given by the formula:

$$y = \lceil ak^x + bk^{N-x} \rceil$$

  where the resulting lengths are distributed in [0,*N*].

- After simulating the possibilities of this formula for *N*=255, a heuristic optimum was found for *k*=0.92 for any *a* and *b*.



- Different values of *a* and *b* produce differing effects

Examples – for lengths in [0,255]

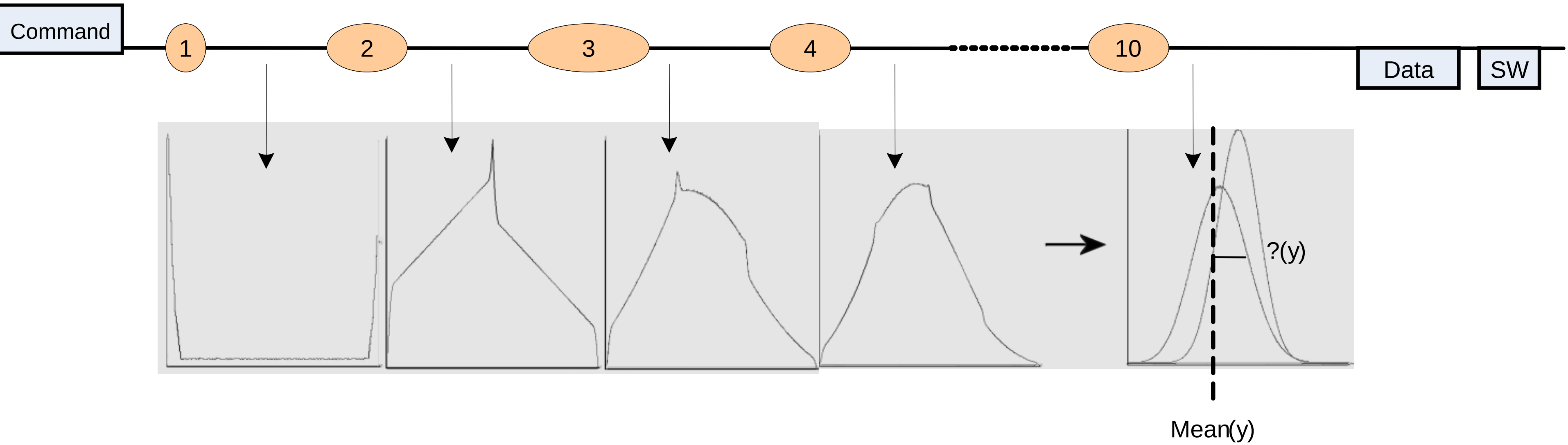


Table 1. Some Parameter Characteristics for Tables of 2^9 Entries

a	b	k	Mean % decrease	σ % increase
25	10	0.88	22.6	30.7
26	6	0.89	32.8	23.5
26	12	0.87	19.7	32.5
32	8	0.86	31.4	25.8

Attack Scenarios

- The proposed optimisation should not be used if:
  - An attacker can dynamically resynchronise after an event, and
  - There is a potential fault attack that can be conducted after one random delay.
- In all other attack scenarios an attacker will be faced with the sum of several (at least) random delays.
- An attacker can determine that the lengths of random delays are distributed in the proposed manner using a chi-squared test.
  - Somewhat lengthy procedure.
  - Less effort to ignore the modified distribution.
  - Only of interest if an attacker can isolate one random delay.

Conclusion

- The proposed modification the distribution of lengths of random delays can be used to increase the desynchronisation produced by random delays in embedded software.
  - This also reduces the time lost because of the use of software random delays.

- However, the lengths of random delays used to hinder dynamic synchronisation should be uniformly distributed.