An Efficient Method for Random Delay Generation in Embedded Software

Jean-Sébastien Coron   Ilya Kizhvatov

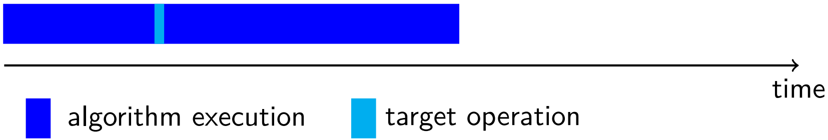
UNIVERSITÉ DU LUXEMBOURG

CHES 2009, Lausanne, Switzerland

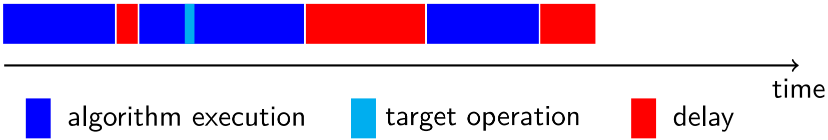
Outline

Outline

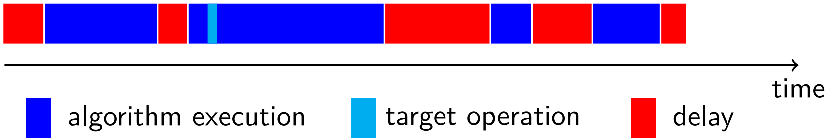
Random Delays: In Brief



Random Delays: In Brief



Random Delays: In Brief



Random Delays: In Brief

Random Delays: In Brief

Effect

- Timing attacks: **noise in time domain**
- DPA attacks: **smeared correlation peak**
  [Clavier et al. CHES'00], [Mangard CT-RSA'04]
- Fault attacks: **decreased fault injection precision**
  [Amiel et al. FDTC'06]

Random Delays: Implementation Levels

Hardware

- **random process interrupts (RPI)** [Clavier et al. CHES'00]
- **gate-level delays** [Bucci *et al.* ISCAS'05], [Lu *et al.* FPT'08]

Software (this work)

- **dummy loops** [Benoit and Tunstall WISTP'07]

```
...
    ld   R0, RND
dummyloop:
    dec  R0
    brne dummyloop
...
```

Outline

Plain Uniform Delays (PU)

$$S_N = \sum_{i=0}^{N} d_i \qquad E(S_N) = N\mu$$

$$d_i \sim \mathcal{U}[0, a] \qquad \mathrm{Var}(S_N) = N\sigma^2$$

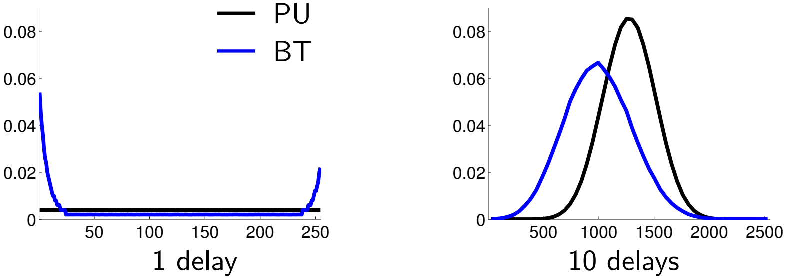
- individual delays are **independent and uniform**
- $\Rightarrow S_N$ has Gaussian distribution

Desired properties of S_N

- **larger variance** to increase the attacker's uncertainty
- **smaller mean** to decrease performance penalty

Method of Benoit and Tunstall [WISTP'07] (BT)

- individual delays: uniform $\longrightarrow$ **pit-shaped** to increase variance
- pit is **asymmetric** to reduce overhead
- individual delays still generated **independently**



In this example: σ^2 33% $\uparrow$, μ 20% $\downarrow$ compared to PU

Limitation of Both Methods

Individual delays are **independent** with mean μ and variance σ^2

$\Downarrow$ **Central Limit Theorem**

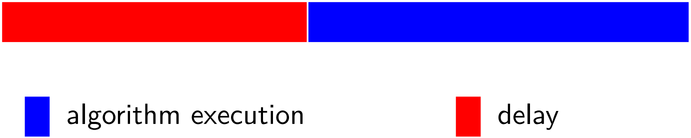
$$S_N \xrightarrow{N} \mathcal{N}(N\mu, N\sigma^2)$$

The **only** way to escape: generate delays **non-independently**

Outline

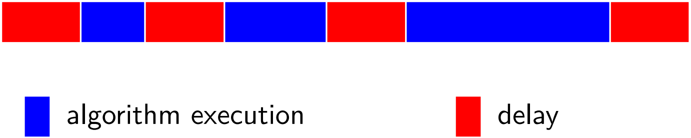
The New Method Step by Step

The New Method Step by Step



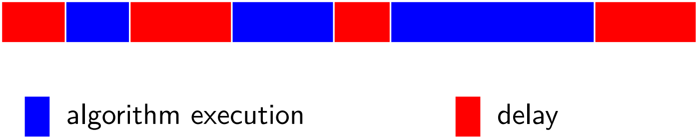
- insert a long uniform delay in the beginning
    - can be removed like in [Nagashima *et al.* ISCAS'07]

The New Method Step by Step



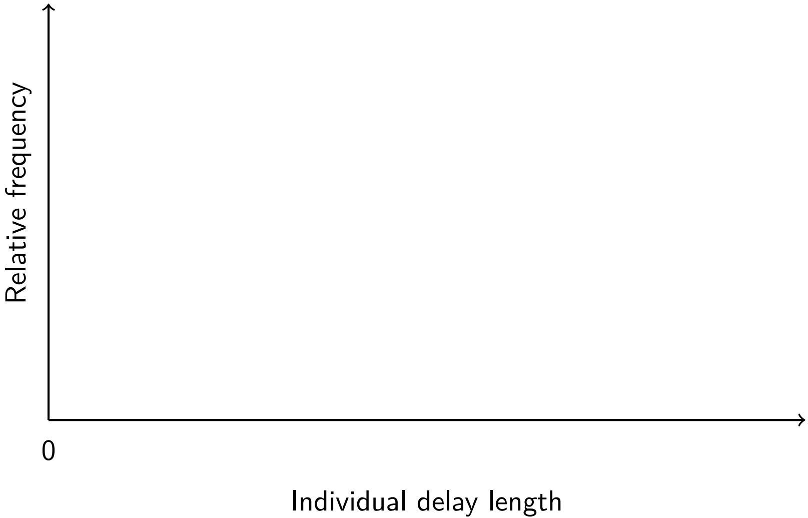
- insert a long uniform delay in the beginning
  - can be removed like in [Nagashima *et al.* ISCAS'07]
- cut it into equal pieces and distribute along the execution
  - the cumulative sum is strictly uniform
  - all delays have identical duration

The New Method Step by Step

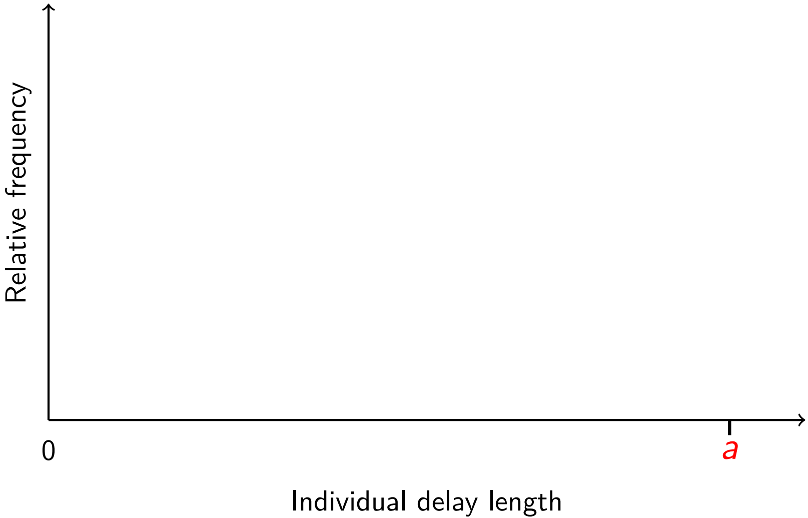


- insert a long uniform delay in the beginning
  - can be removed like in [Nagashima *et al.* ISCAS'07]
- cut it into equal pieces and distribute along the execution
  - the cumulative sum is strictly uniform
  - all delays have identical duration
- add small variation to individual delays
  - the cumulative sum is *almost* uniform

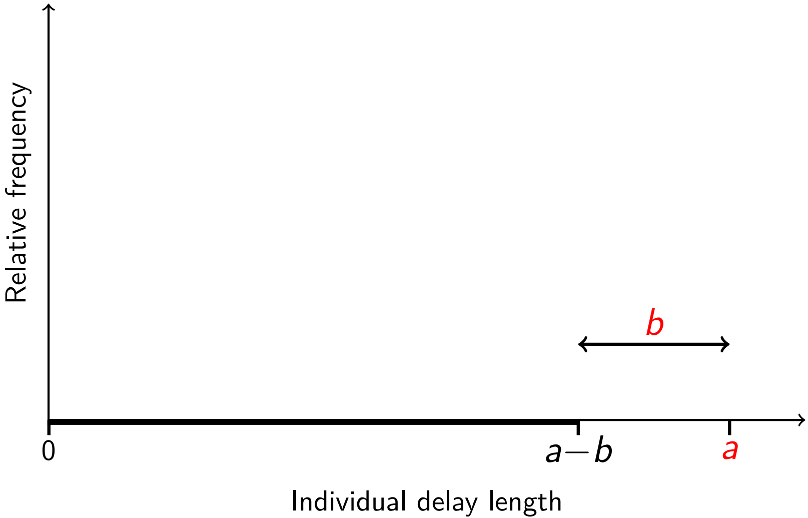
The New Method: More Formally



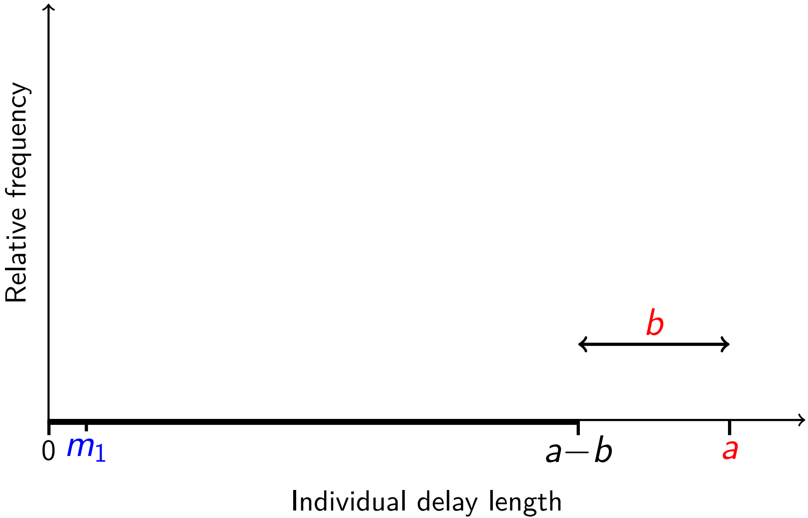
The New Method: More Formally



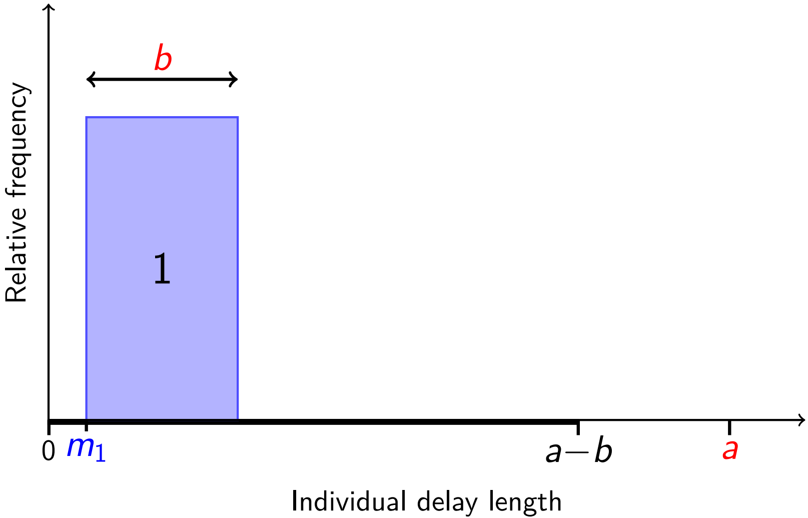
The New Method: More Formally



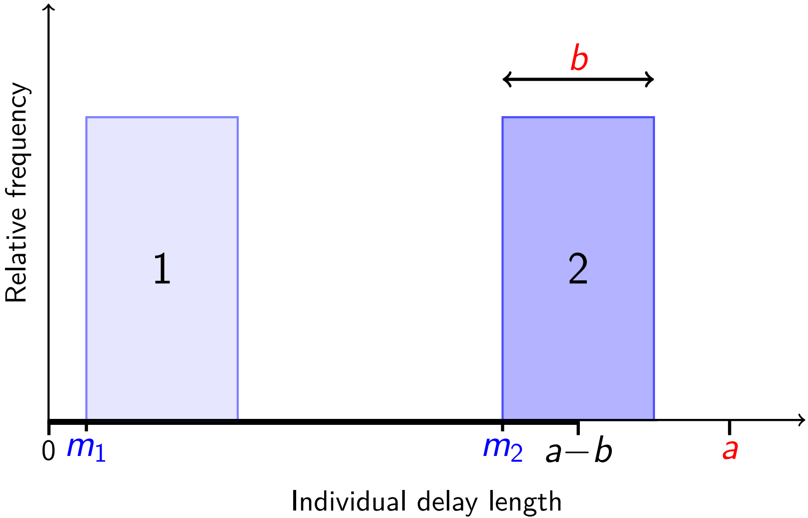
The New Method: More Formally



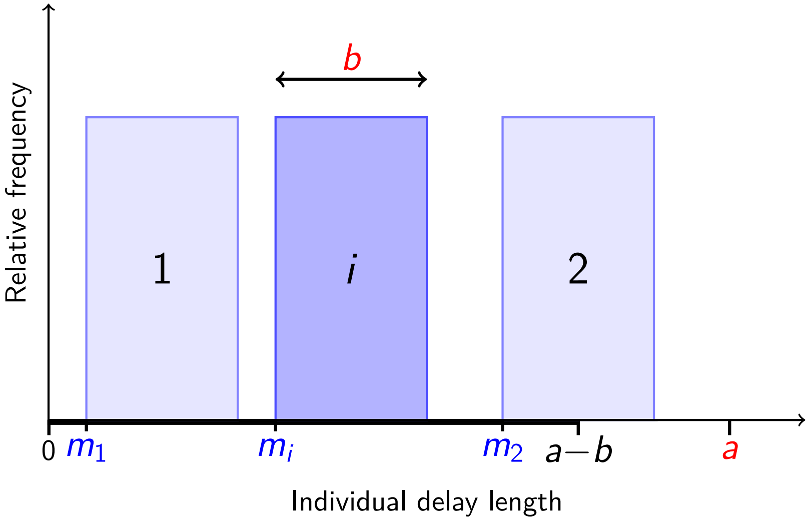
The New Method: More Formally



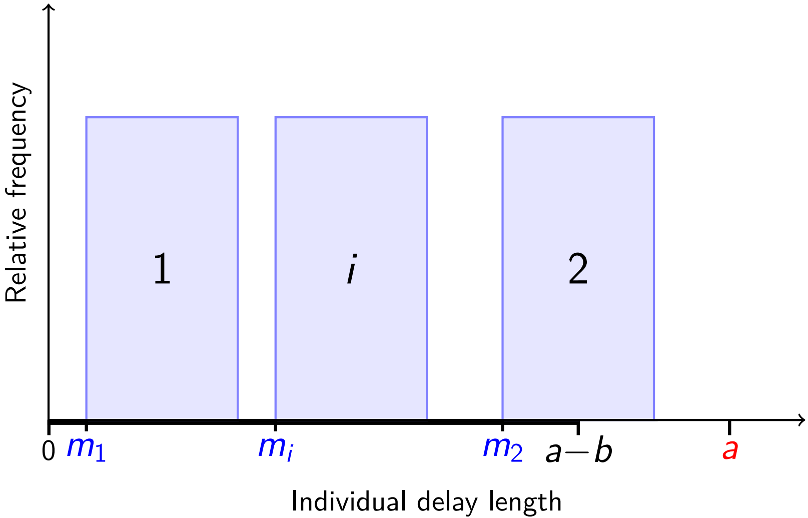
The New Method: More Formally



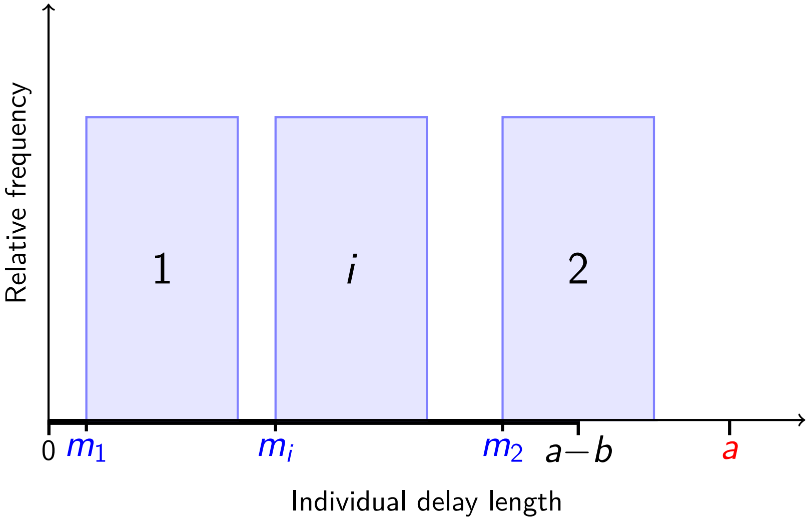
The New Method: More Formally



The New Method: More Formally

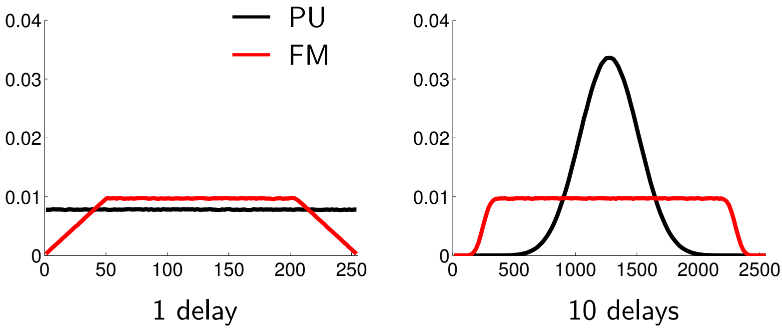


Floating mean: More Formally



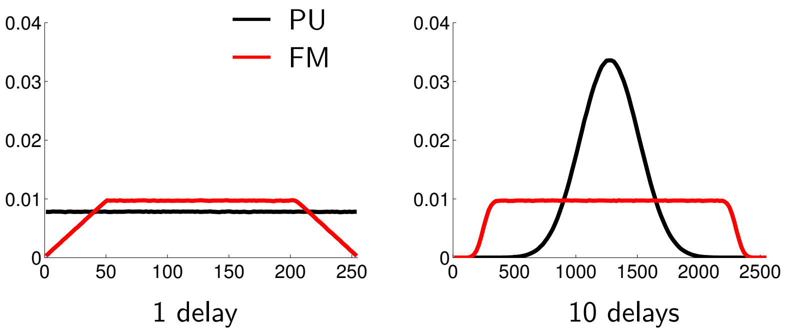
Floating Mean: Distribution

$$E(S_N) = \frac{Na}{2}, \qquad \mathrm{Var}(S_N) = N^2 \cdot \frac{(a-b+1)^2 - 1}{12} + N \cdot \frac{b^2 + 2b}{12}$$



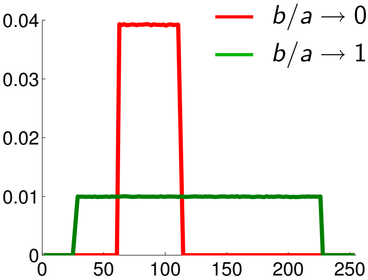
Floating Mean: Distribution

$$E(S_N) = \frac{Na}{2}, \qquad \mathrm{Var}(S_N) = \boxed{N^2} \cdot \frac{(a-b+1)^2 - 1}{12} + N \cdot \frac{b^2 + 2b}{12}$$

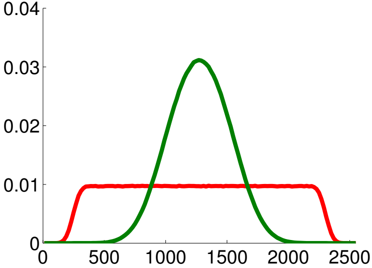


Floating Mean: Tradeoff

- $b/a \to 0$: individual delays within a trace have small variation, cumulative sum is almost uniformly distributed
- $b/a \to 1$: plain uniform delays, cumulative sum tends to normal distribution



1 delay **within an execution**



10 delays

Outline

1 About Random Delays as a Countermeasure

2 Existing Methods for Random Delay Generation in Software

3 The New Method

4 Efficiency Comparison Between the Methods

Comparing Efficiency

Our Criterion

- what performance overhead is required to achieve the given variation of the sum of N delays
- use **coefficient of variation** σ/μ

Plain uniform	Benoit-Tunstall	Floating mean
$\frac{1}{\sqrt{3N}}$	$\frac{\sigma_{\mathrm{BT}}}{\mu_{\mathrm{BT}}} \cdot \frac{1}{\sqrt{N}}$	$\frac{\sqrt{N((a-b+1)^2-1)+b^2+2b}}{a\sqrt{3N}}$

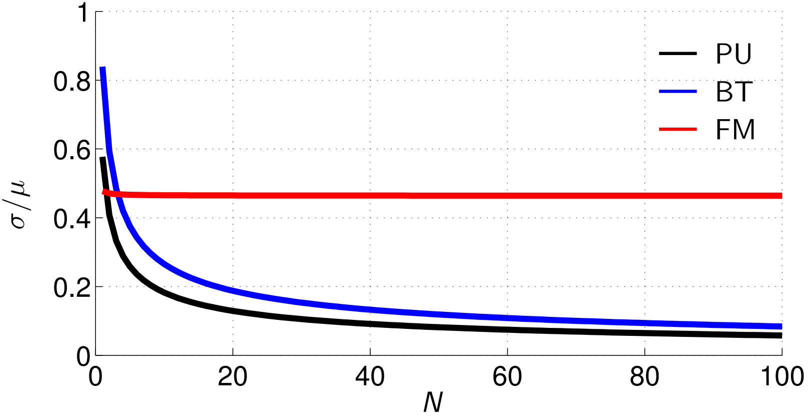
Comparing Efficiency

Our Criterion

- what performance overhead is required to achieve the given variation of the sum of N delays
- use **coefficient of variation** σ/μ

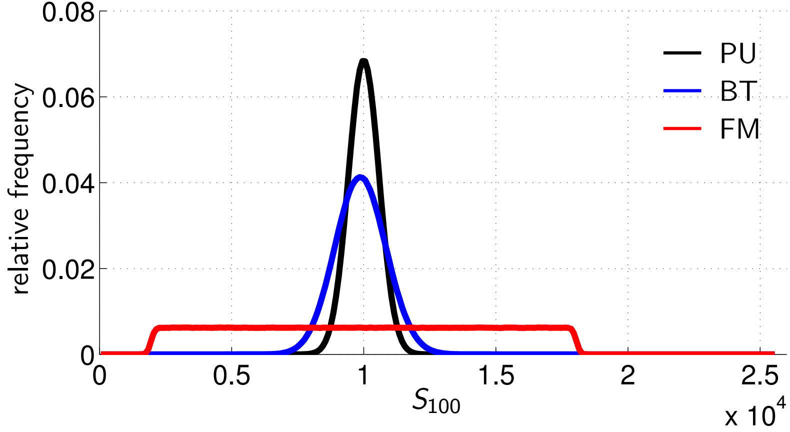
Plain uniform	Benoit-Tunstall	Floating mean
$\Theta\left(\frac{1}{\sqrt{N}}\right)$	$\Theta\left(\frac{1}{\sqrt{N}}\right)$	$\Theta\left(1\right)$

Comparing Efficiency



Efficiency of the methods against the number of delays in S_N

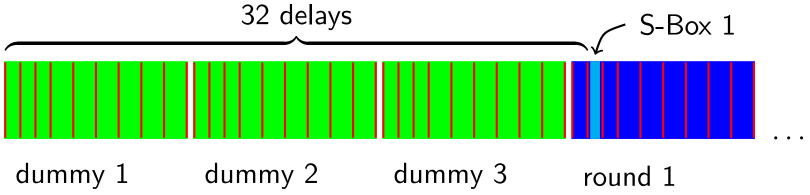
Comparing Efficiency



Distribution of S_{100} for the same performance overhead

Practical Implementation: Details

- AES-128 on Atmel ATmega16
- 10 delays per round, 3 dummy rounds at start/end
- same performance overhead for all methods
- no other countermeasures
- CPA attack [Brier *et al.* CHES'04]



Practical Implementation: Results

	ND	**PU**	**BT**	**FM**
μ, cycles	0	720	860	862
σ, cycles	0	79	129	442
σ/μ	–	**0.11**	**0.15**	**0.51**
CPA, traces	**50**	**2500**	**7000**	**45000**

Conclusion

Our result

- a **new method** for random delay generation in embedded software
- **more efficient and secure** than existing methods

Conclusion

Our result

- a **new method** for random delay generation in embedded software
- **more efficient and secure** than existing methods

Not covered in this talk

- lightweight implementation

Updated version of the paper: **ePrint 2009/419**