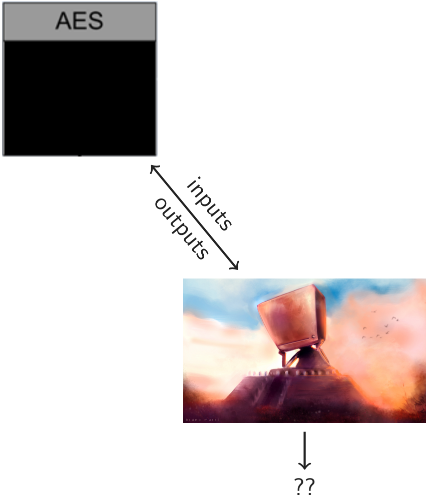
Scalable Key Rank Estimation Algorithm for Large Keys

Vincent Grosso

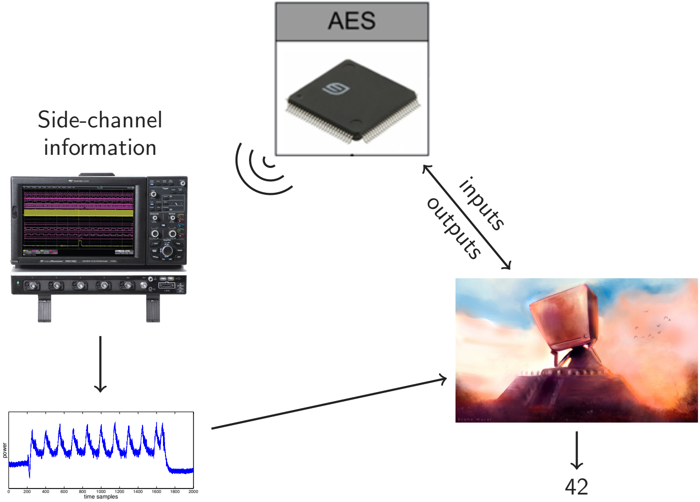
13/11/2018

CNRS/laboratoire Hubert Curien
Université Jean Monnet
Saint-Étienne

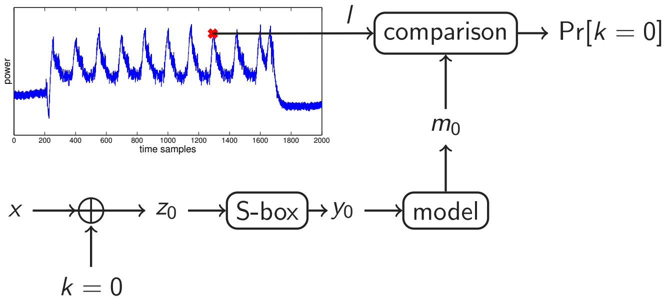
Side-channel attacks: cryptography



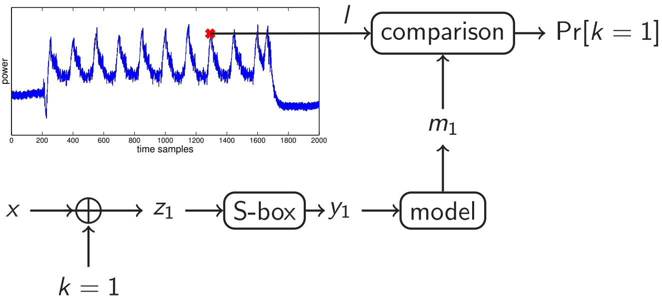
Side-channel attacks: cryptography



Side-channel attacks

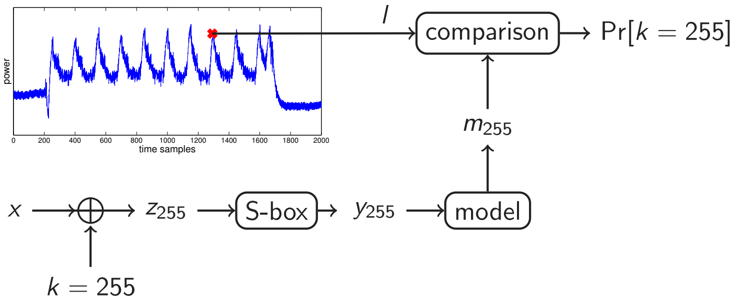


Divide-and-conquer strategy.



Divide-and-conquer strategy.

Side-channel attacks



Divide-and-conquer strategy.

Side-channel attacks: result

k_0	k_1	k_2	$\cdots$	k_{15}
0X2a,0.125	0X23,0.128	0X10,0.325		0X45,0.347
0Xcd,0.100	0X51,0.045	0X01,0.204		0Xdc,0.210
0Xae,0.050	0Xff,0.035	0X13,0.036		0X83,0.151
0X63,0.025	0X2b,0.025	0X58,0.029		0X13,0.035
$\cdots$	$\cdots$	$\cdots$		$\cdots$

Side-channel attacks: result

k_0	k_1	k_2	$\cdots$	k_{15}
0X2a,0.125	0X23,0.128	0X10,0.325		0X45,0.347
0Xcd,0.100	0X51,0.045	0X01,0.204		0Xdc,0.210
0Xae,0.050	0Xff,0.035	0X13,0.036		0X83,0.151
0X63,0.025	0X2b,0.025	0X58,0.029		0X13,0.035
$\cdots$	$\cdots$	$\cdots$		$\cdots$

Enough side-channel information
$\Rightarrow$ direct recovery (attack)

Side-channel attacks: result

k_0	k_1	k_2	$\cdots$	k_{15}
0X2a,0.125	0X23,0.128	0X10,0.325		0X45,0.347
0Xcd,0.100	0X51,0.045	0X01,0.204		0Xdc,0.210
0Xae,0.050	0Xff,0.035	0X13,0.036		0X83,0.151
0X63,0.025	0X2b,0.025	0X58,0.029		0X13,0.035
$\cdots$	$\cdots$	$\cdots$		$\cdots$

Enough side-channel information $\Rightarrow$ direct recovery (attack)

Not Enough side-channel information, enough computational power $\Rightarrow$ enumeration (attack)

Side-channel attacks: result

k_0	k_1	k_2	$\cdots$	k_{15}
0X2a,0.125	0X23,0.128	0X10,0.325		0X45,0.347
0Xcd,0.100	0X51,0.045	0X01,0.204		0Xdc,0.210
0Xae,0.050	0Xff,0.035	0X13,0.036		0X83,0.151
0X63,0.025	0X2b,0.025	0X58,0.029		0X13,0.035
$\cdots$	$\cdots$	$\cdots$		$\cdots$

Enough side-channel information $\Rightarrow$ direct recovery (attack)

Not Enough side-channel information, enough computational power $\Rightarrow$ enumeration (attack)

Side-channel attacks: result

k_0	k_1	k_2	$\cdots$	k_{15}
`0X2a`,0.125	`0X23`,0.128	`0X10`,0.325		`0X45`,0.347
`0Xcd`,0.100	`0X51`,0.045	`0X01`,0.204		`0Xdc`,0.210
`0Xae`,0.050	`0Xff`,0.035	`0X13`,0.036		`0X83`,0.151
`0X63`,0.025	`0X2b`,0.025	`0X58`,0.029		`0X13`,0.035
$\cdots$	$\cdots$	$\cdots$		$\cdots$

Enough side-channel information
$\Rightarrow$ direct recovery
(attack)

Not Enough side-channel information, enough computational power
$\Rightarrow$ enumeration
(attack)

Not Enough side-channel information, not enough computational power
$\Rightarrow$rank estimation
(key needed, evaluation)

Table of contents

Previous solutions

Problem

$$\mathsf{rank}(k) = \#\{k^* \,|\, \mathsf{Pr}[k^*|\mathrm{SCI}] \geq \mathsf{Pr}[k|\mathrm{SCI}]\}.$$

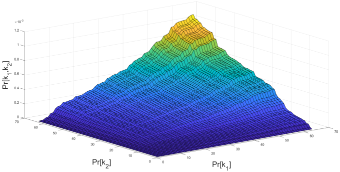
$$\#\{k^*\} \geq 2^{128}$$

Problem

$$\text{rank}(k) = \#\{k^* \,|\, \Pr[k^*|\mathrm{SCI}] \geq \Pr[k|\mathrm{SCI}]\}.$$

$$\#\{k^*\} \geq 2^{128}$$

Divide-and-conquer approach on independent subkeys

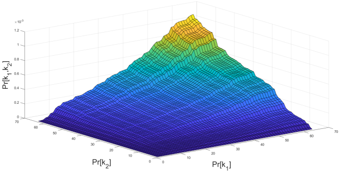


Problem

$$\text{rank}(k) = \#\{k^* \,|\, \Pr[k^*|\mathrm{SCI}] \geq \Pr[k|\mathrm{SCI}]\}.$$

$$\#\{k^*\} \geq 2^{128}$$

Divide-and-conquer approach on independent subkeys



Space carving $\{k^*\}$ into 3 parts:

$$< \Pr[k|\mathrm{SCI}] \qquad ? \qquad \Pr[k|\mathrm{SCI}] <$$

Smaller is the part ? the more accurate is the rank estimation

Rank estimation algorithms zoo

Method	Pros	Cons
Eurocrypt'13	First solution, can compute the exact rank (in theory)	Quite slow, quite loose bounds (in practice)
Pro'15		
FSE'15		
Asiacrypt' 15		
CT-RSA'17		
CHES'17		
This paper		

Rank estimation algorithms zoo

Method	Pros	Cons
Eurocrypt'13	First solution, can compute the exact rank (in theory)	Quite slow, quite loose bounds (in practice)
Pro'15	Efficient and tight solution for small keys	Quite slow for large keys and reasonable tightness
FSE'15	Efficient and tight solution for small keys	Quite slow for large keys and reasonable tightness
Asiacrypt' 15	Efficient and tight solution for small keys	Quite slow for large keys and reasonable tightness
CT-RSA'17		
CHES'17		
This paper		

Rank estimation algorithms zoo

Method	Pros	Cons
Eurocrypt'13	First solution, can compute the exact rank (in theory)	Quite slow, quite loose bounds (in practice)
Pro'15	Efficient and tight solution for small keys	Quite slow for large keys and reasonable tightness
FSE'15	Efficient and tight solution for small keys	Quite slow for large keys and reasonable tightness
Asiacrypt' 15	Efficient and tight solution for small keys	Quite slow for large keys and reasonable tightness
CT-RSA'17	Efficient solution	Loose bound, not exact rank
CHES'17		
This paper		

Rank estimation algorithms zoo

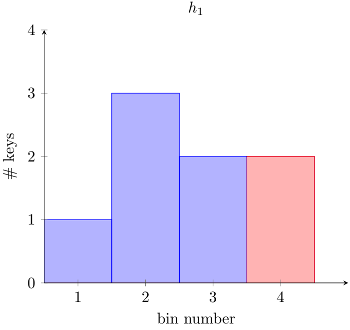
Method	Pros	Cons
Eurocrypt'13	First solution, can compute the exact rank (in theory)	Quite slow, quite loose bounds (in practice)
Pro'15	Efficient and tight solution for small keys	Quite slow for large keys and reasonable tightness
FSE'15	Efficient and tight solution for small keys	Quite slow for large keys and reasonable tightness
Asiacrypt' 15	Efficient and tight solution for small keys	Quite slow for large keys and reasonable tightness
CT-RSA'17	Efficient solution	Loose bound, not exact rank
CHES'17	Really fast even for large key	Expected value of the rank, not rank estimation
This paper		

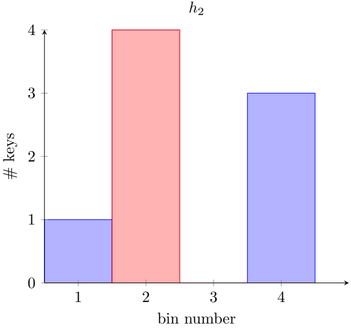
Rank estimation algorithms zoo

Method	Pros	Cons
Eurocrypt'13	First solution, can compute the exact rank (in theory)	Quite slow, quite loose bounds (in practice)
Pro'15	Efficient and tight solution for small keys	Quite slow for large keys and reasonable tightness
FSE'15	Efficient and tight solution for small keys	Quite slow for large keys and reasonable tightness
Asiacrypt' 15	Efficient and tight solution for small keys	Quite slow for large keys and reasonable tightness
CT-RSA'17	Efficient solution	Loose bound, not exact rank
CHES'17	Really fast even for large key	Expected value of the rank, not rank estimation
This paper	Fast, tight estimation of the rank even for large key	Less efficient than CHES'17

The histogram solution (FSE'15)

	k_1			k_2		
Candidate	Pr	log	bin	Pr	log	bin
0	0.6643	-0.5901	1	0.0012	-9.7027	3
1	0.2588	-1.9501	1	0.0011	-9.8283	3
2	0.0313	-4.9977	2	0.3588	-1.4787	1
3	0.0412	-4.6012	2	0.0713	-3.8100	1
4	0.0001	-13.2877	4	0.5643	-0.8255	1
5	0.0020	-8.9658	3	0.0012	-9.7027	3
6	0.0013	-9.5873	3	0.00005	-14.2877	4
7	0.0010	-9.9658	3	0.00205	-8.9302	3

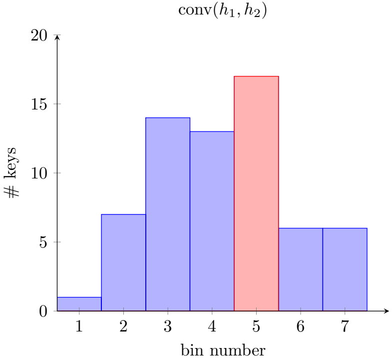


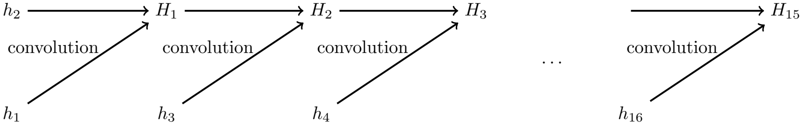


Mix result

Perform convolution of histogram

$$conv(h_1, h_2)[i] = \sum_{j=0}^{i} h_1[j] h_2[i-j]$$

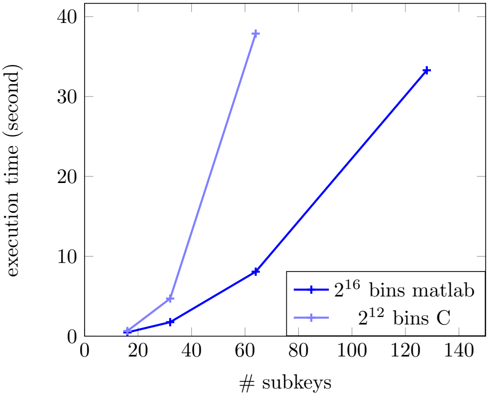




Size of H_i grows

Limitation for larger keys

For large number of dimension we perform convolution on larger and larger histograms: could be costly.

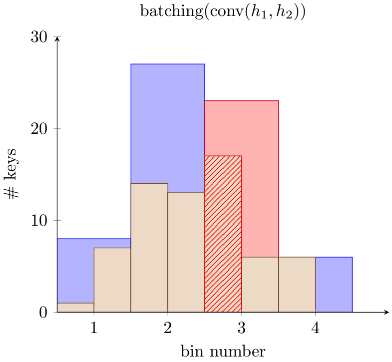


Can the cost be linear in the number of subkeys?

New solution

Batching

Keep the size of the histogram constant

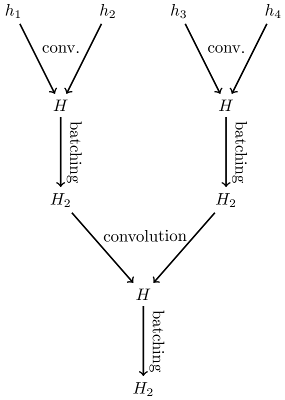


Why does it work?

Similar to classical histogram solution we can keep track of the position (bin of the key)

Similar as dividing the number of bins by the number of subkeys

Convolution need equally bin sized histogram: need a balanced tree structure



Why is more efficient?

Similar as doing classical histogram convolution with large bin

But a better tracking of the estimation error

ν : number of subkeys

ϵ: size of bin $/2$

Method	error	cost
Classical FSE'15	$\nu\epsilon$	quadratic
Reduced FSE'15	$\nu^2\epsilon$	linear
Batching	$(\nu + \log_2(\nu)\frac{\nu}{2})\epsilon$	linear

Experimental results

Experimental setup

- Matlab implementation (limited to 1024-bit key)
- C implementation

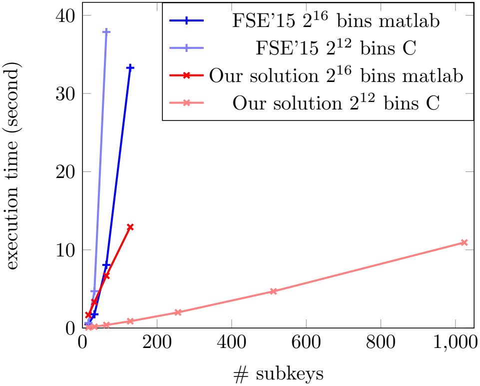
Leakages: subkey+noise (~~S-box~~)

Size of subkey: 8-bit

Number of subkeys: 8-1024

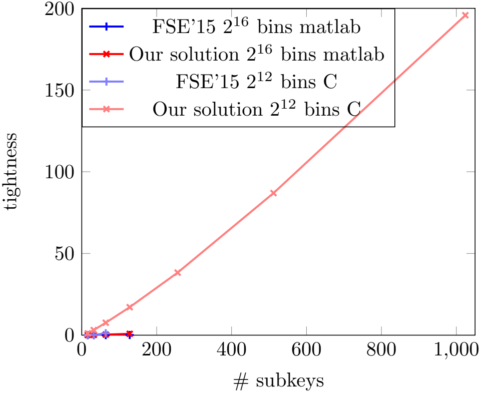
Number of bins: tightness-efficiency parameter

Efficiency



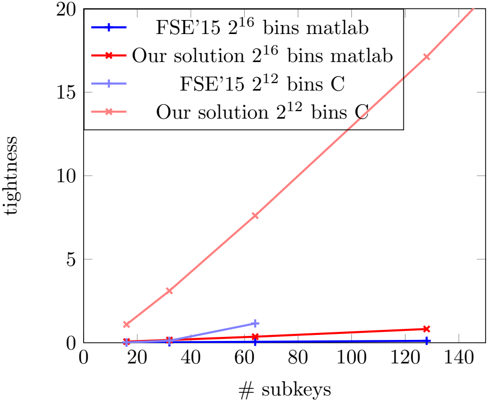
Our solution has a complexity linear in the number of subkeys

Tightness



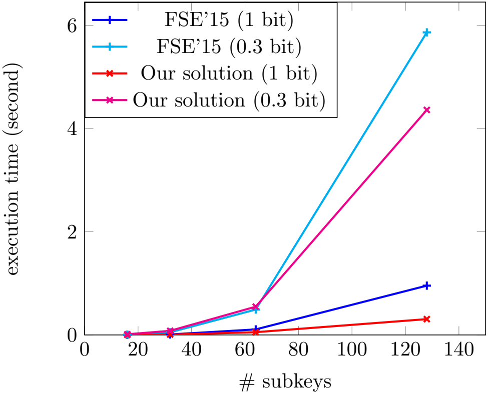
Works for very long key, with tightness damage

Tightness

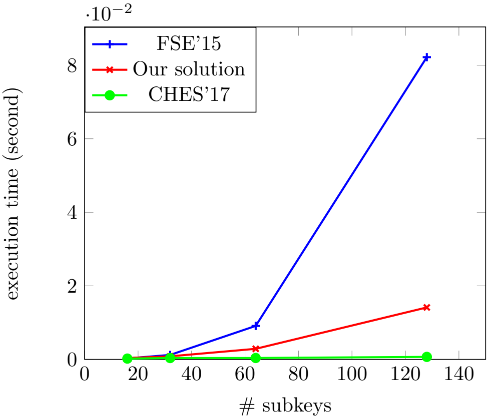


Works for very long key, with tightness damage

Similar tightness



Adapted number of bin to have similar tightness



CHES'17 solution is not so tight (6 bits, and cannot be tightened) all solutions are efficient ($< 0.1s$)

Conclusion

Conclusion & open problems

Trick for rank estimation for large keys

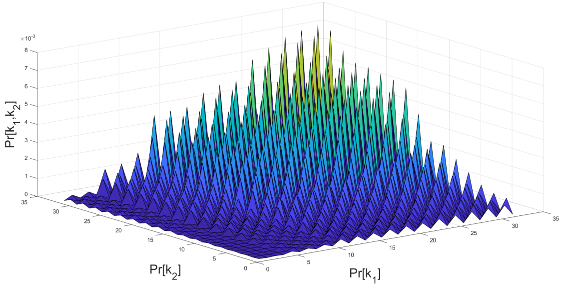
Tight and efficient method

Conclusion & open problems

Trick for rank estimation for large keys

Tight and efficient method

Limited to independent attack



Thanks!

Questions?

Comments?