Fast Analytical Rank Estimation

Liron David and Prof. Avishai Wool

School of Electrical Engineering
Tel-Aviv University

COSADE 2019

Our Goal

Given an **implementation** of a **symmetric encryption algorithm** and the **secret key**

Our **goal** is:

To estimate the **strength** of the **secret key** **against side channel attacks**

Side Channel Attack

Secret Key

128 bits

$%$#@!@#$%^&*&^%$%^&%^%#@$%^&$#@$%^&#@!#$!~!#%&$*&!^$%^&$#@$%^&#@!#$!~!

Side Channel Attack

Divide-and-Conquer

The attacker **reveals a small part** of bits each time

- Denoted by subkeys

Secret Key

8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits	8 bits
$%$#	@!@#	$%^&*	&^%$	%^&%	^%#@	$%^&	$#@$%	^&#@	!#$!~	!#%&$	*&!^$	%^&$	#@$%	^&#@	!#$!~!

Side Channel Attack

Secret Key

8 bits	8 bits	8 bits	
$%$#@!@#	@$%^#&*!	^*$*$&@%	

The first Subkey

Subkeys	Probabilities
00000000	0.00001
00000001	0.00004
00000010	0.00005
00000011	0.002
00000100	0.004
00000101	0.003
00000110	0.001
00000111	0.003
00001000	0.002
…	
11111110	0.0004
11111111	0.0002

Side Channel Attack

Secret Key

8 bits	8 bits	8 bits
$%$#@!@#	@$%^#&*!	^*$*$&@%

The first Subkey

Subkeys **Probabilities**

00000100 → 0.004
00000101 → 0.003
00001000 → 0.002
00000110 → 0.001
00000111 → 0.003
00000000 → 0.00001
00000010 → 0.00005
11111110 → 0.0004
11111111 → 0.0002
00000001 → 0.00004
00000011 → 0.002

We sort the subkeys according to their probabilities in decreasing order…

Side Channel Attack

Secret Key

8 bits	8 bits	8 bits
$%$#@!@#	@$%^#&*!	^*$*$&@%

The first Subkey

(P_1, K_1)

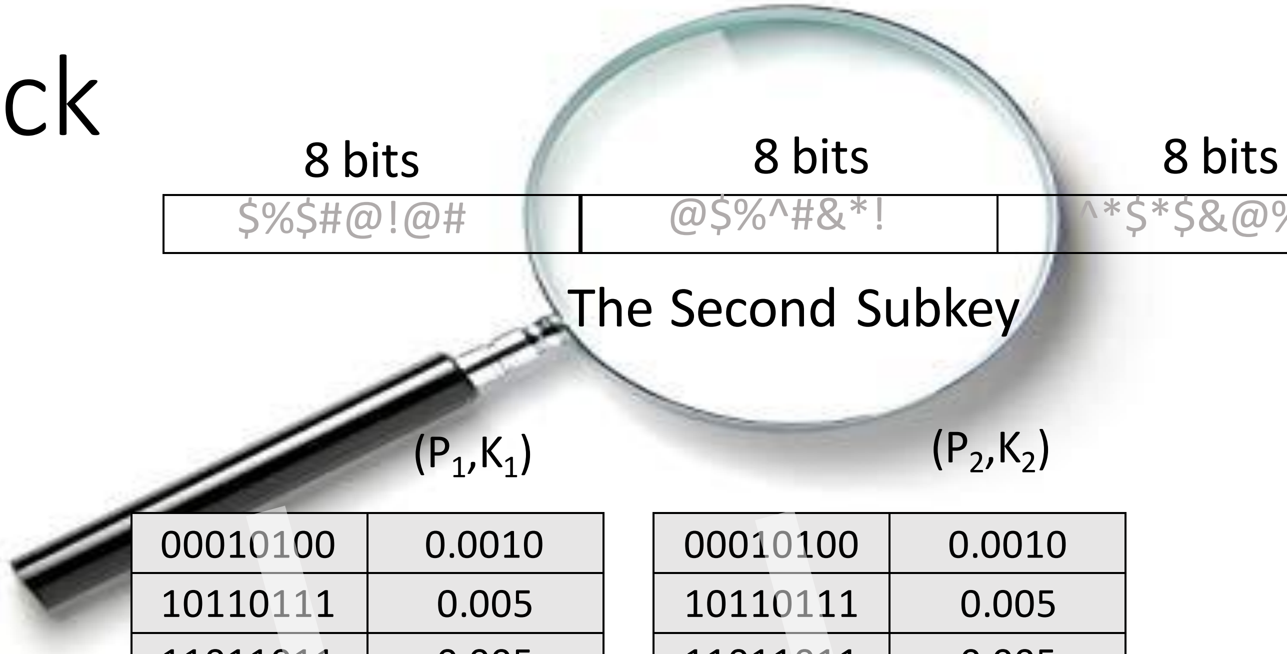
00010100	0.0010
10110111	0.005
11011011	0.005
01000011	0.0045
01110000	0.0043
11011010	0.003
10101110	0.003
01001111	0.002
10100110	0.0015
…	…
00000000	0.000001
11111111	0.000001

Sorted subkeys in decreasing order of probabilites

Side Channel Attack

Secret Key

8 bits	8 bits	8 bits	
$%$#@!@#	@$%^#&*!	^*$*$&@%	



The Second Subkey

Sorted subkeys in decreasing order of probabilites

(P_1,K_1)

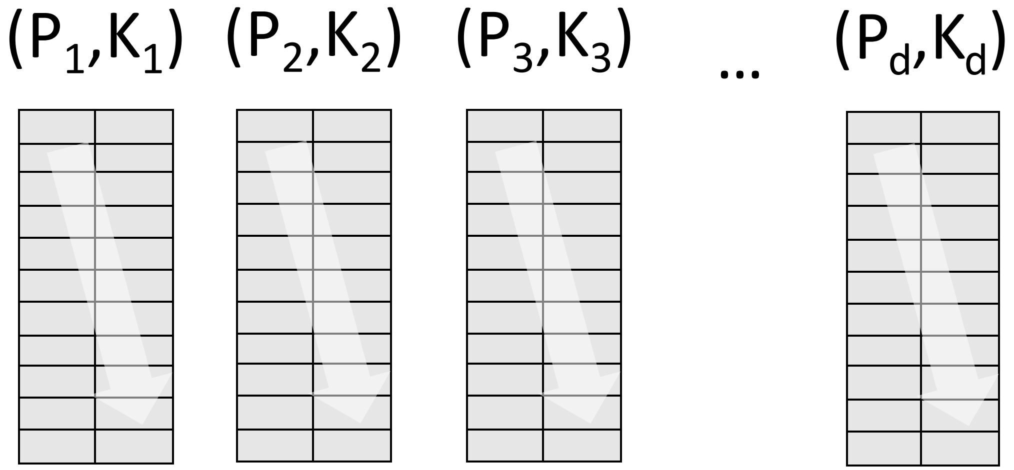
00010100	0.0010
10110111	0.005
11011011	0.005
01000011	0.0045
01110000	0.0043
11011010	0.003
10101110	0.003
01001111	0.002
10100110	0.0015
…	…
00000000	0.000001
11111111	0.000001

(P_2,K_2)

00010100	0.0010
10110111	0.005
11011011	0.005
01000011	0.0045
01110000	0.0043
11011010	0.003
10101110	0.003
01001111	0.002
10100110	0.0015
…	…
00000000	0.000001
11111111	0.000001

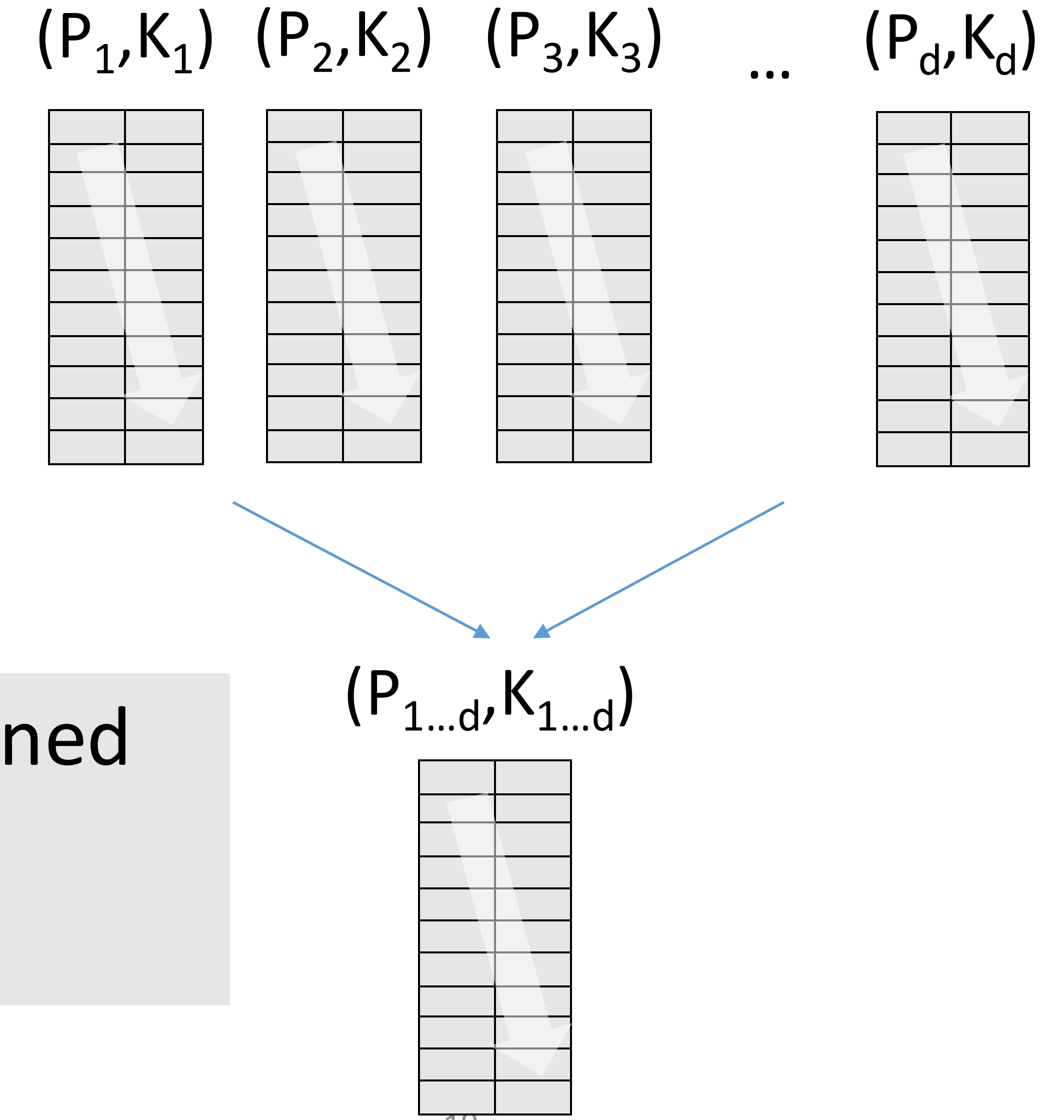
Side Channel Attack

- **d independent subkey spaces** (K_i,P_i) each of size N
- **sorted** in decreasing order of proabilities.



Side Channel Attack

- The attacker **goes over the full keys**
- in **sorted order** from the most likely to the least,
- **till he reaches the correct key**.



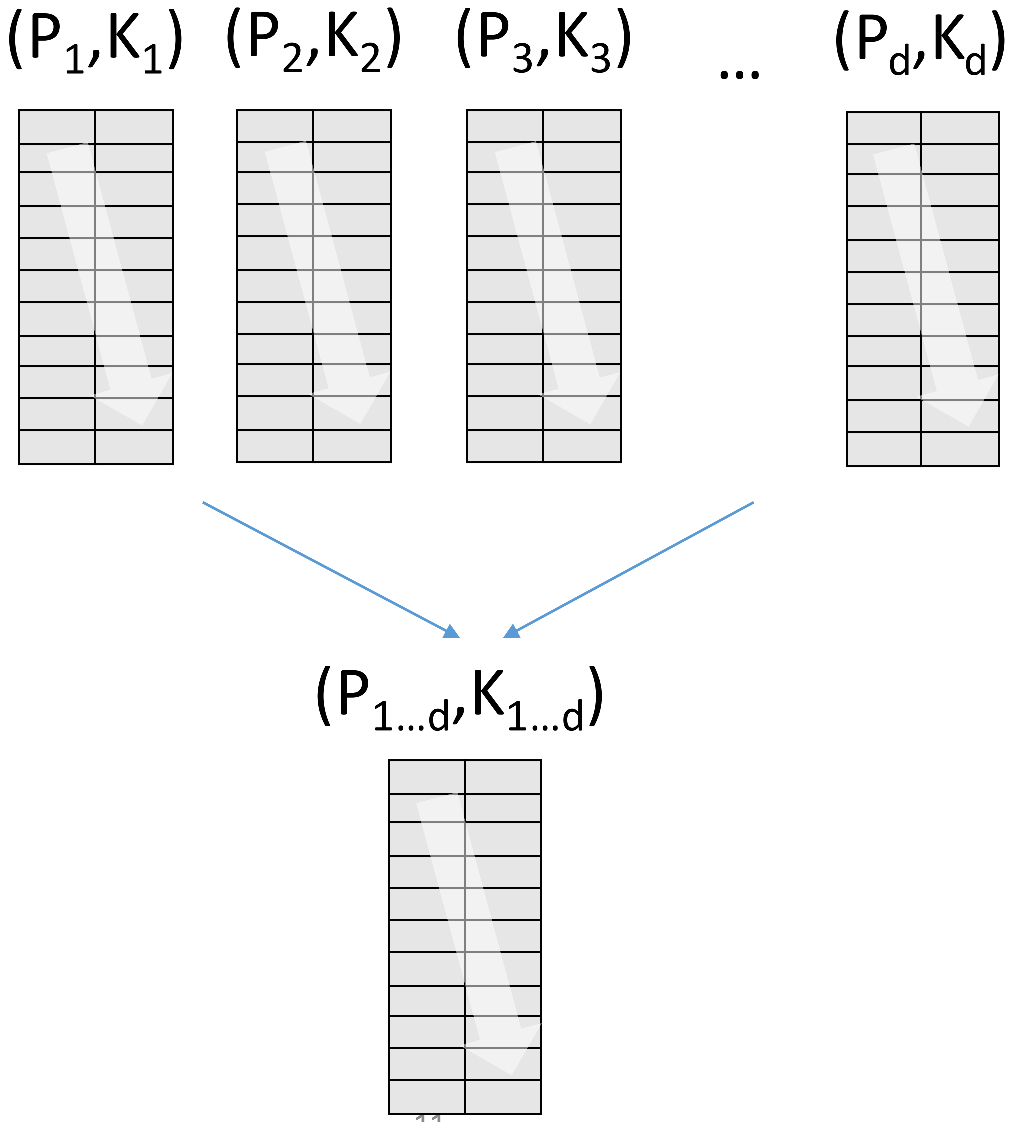
The **probability** of a **full key** is defined as the **product** of its **subkey's probabilities**.

Side Channel Attack

An important question is:

How many full keys the attacker needs to **try before he reaches** the **correct key.**

This allows **estimating** the **strength of the chosen secret key** after an attack has been performed.



Side Channel Attack

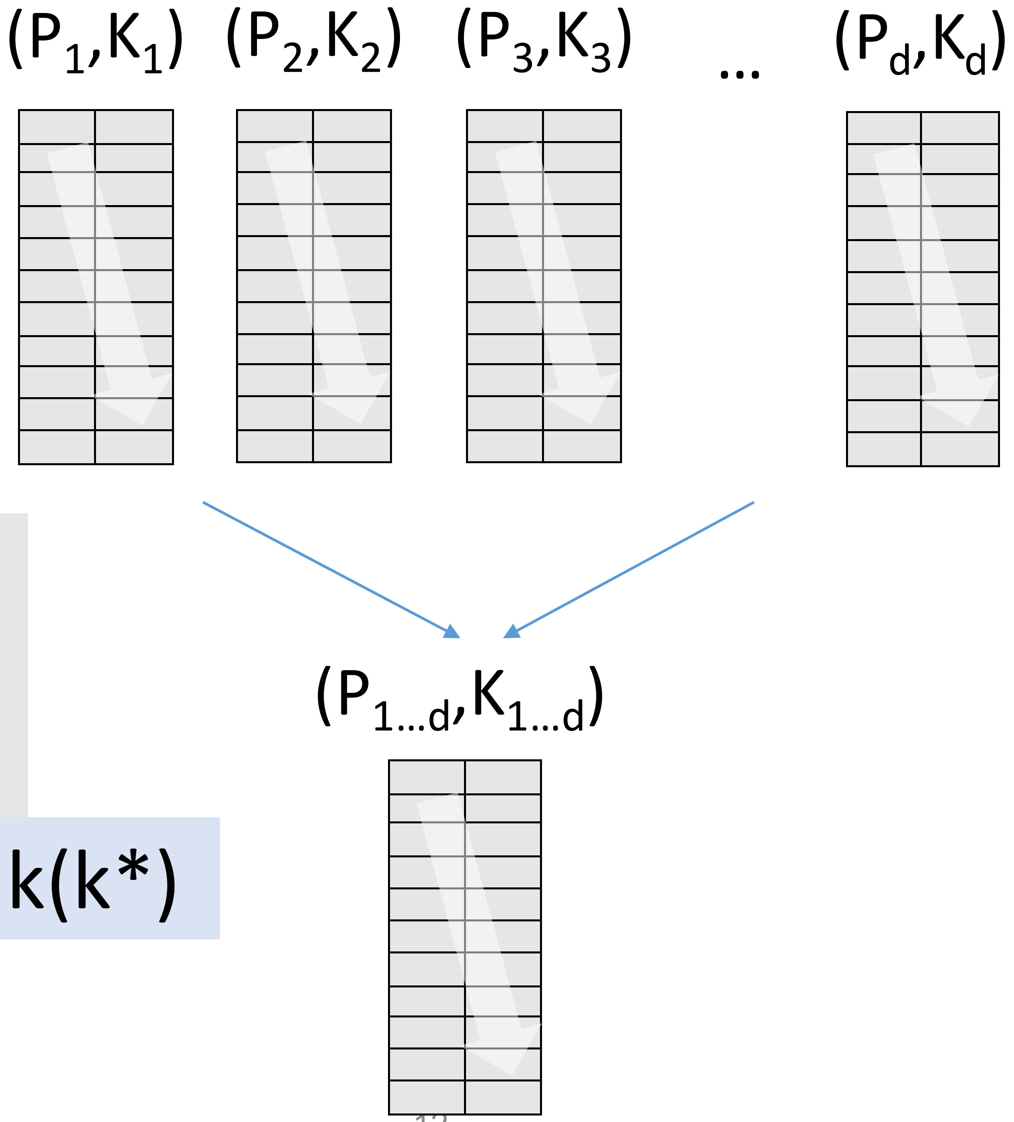
So assume we **know**

- The **correct key k*** and its probability **p***
- The **d subkey spaces** (K_i,P_i)

The goal :

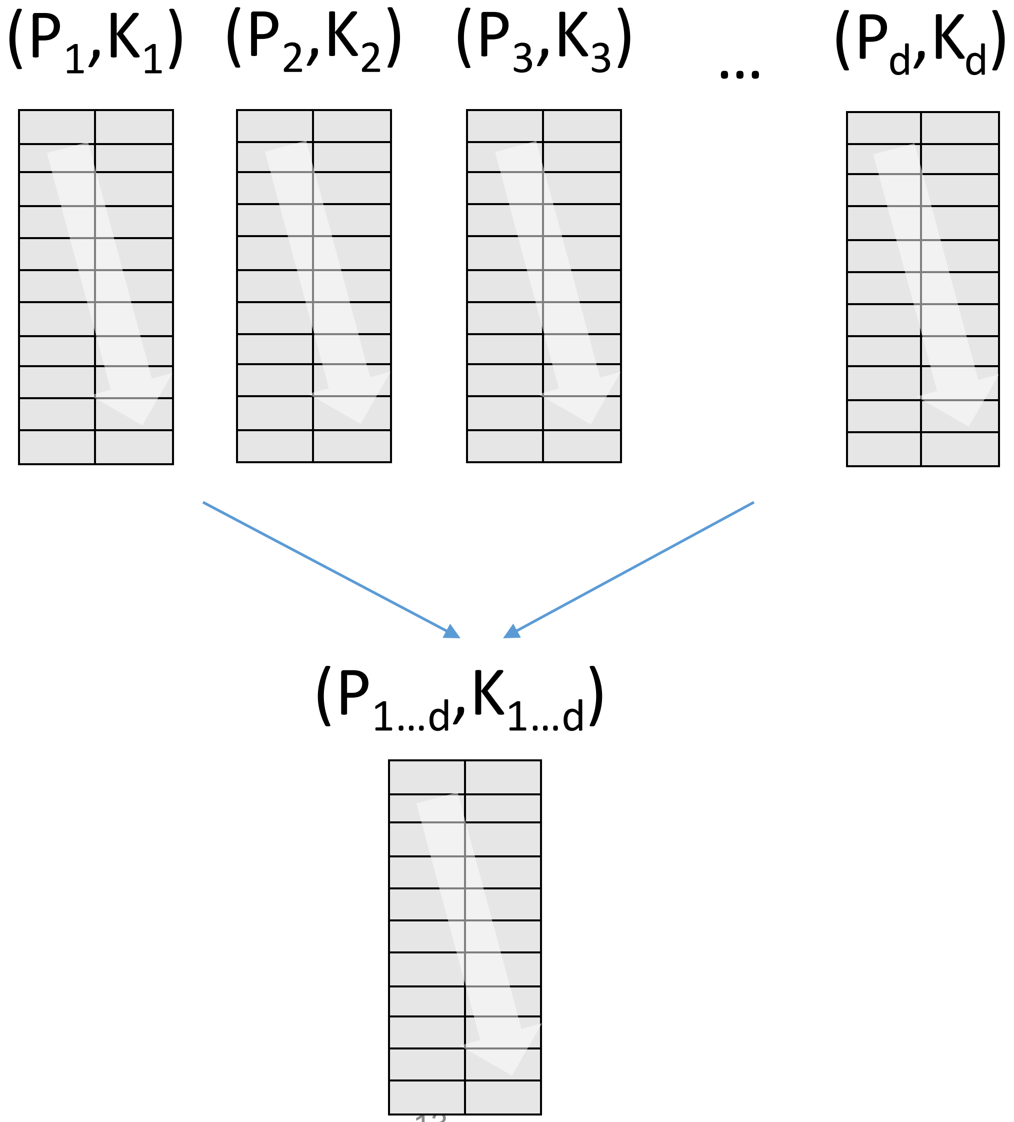
to estimate the **number of full keys with probability higher than p***

This is rank(k*)



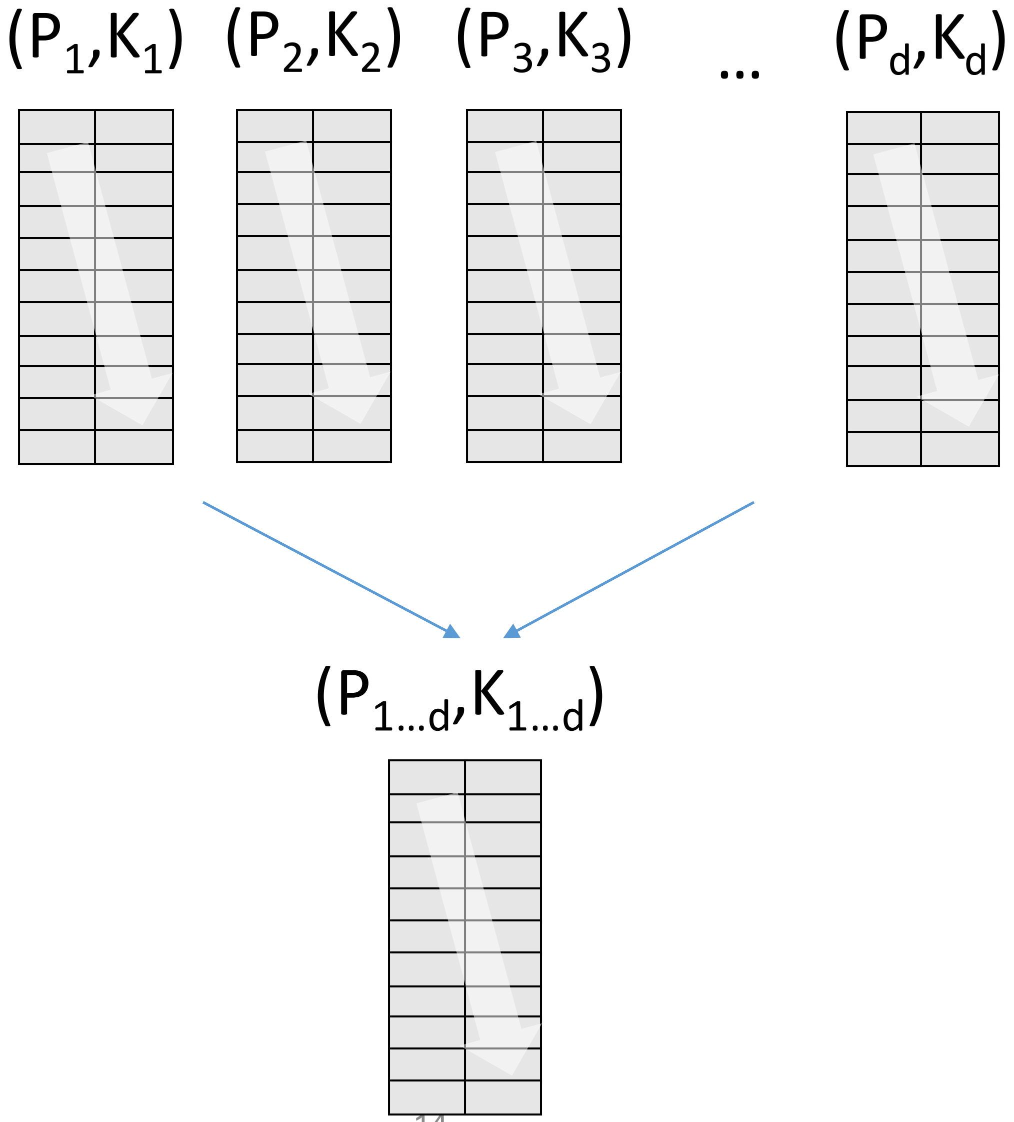
Side Channel Attack

- The optimal solution
- **enumerates** and **counts** the full keys in optimal-order
- till reaches to k*



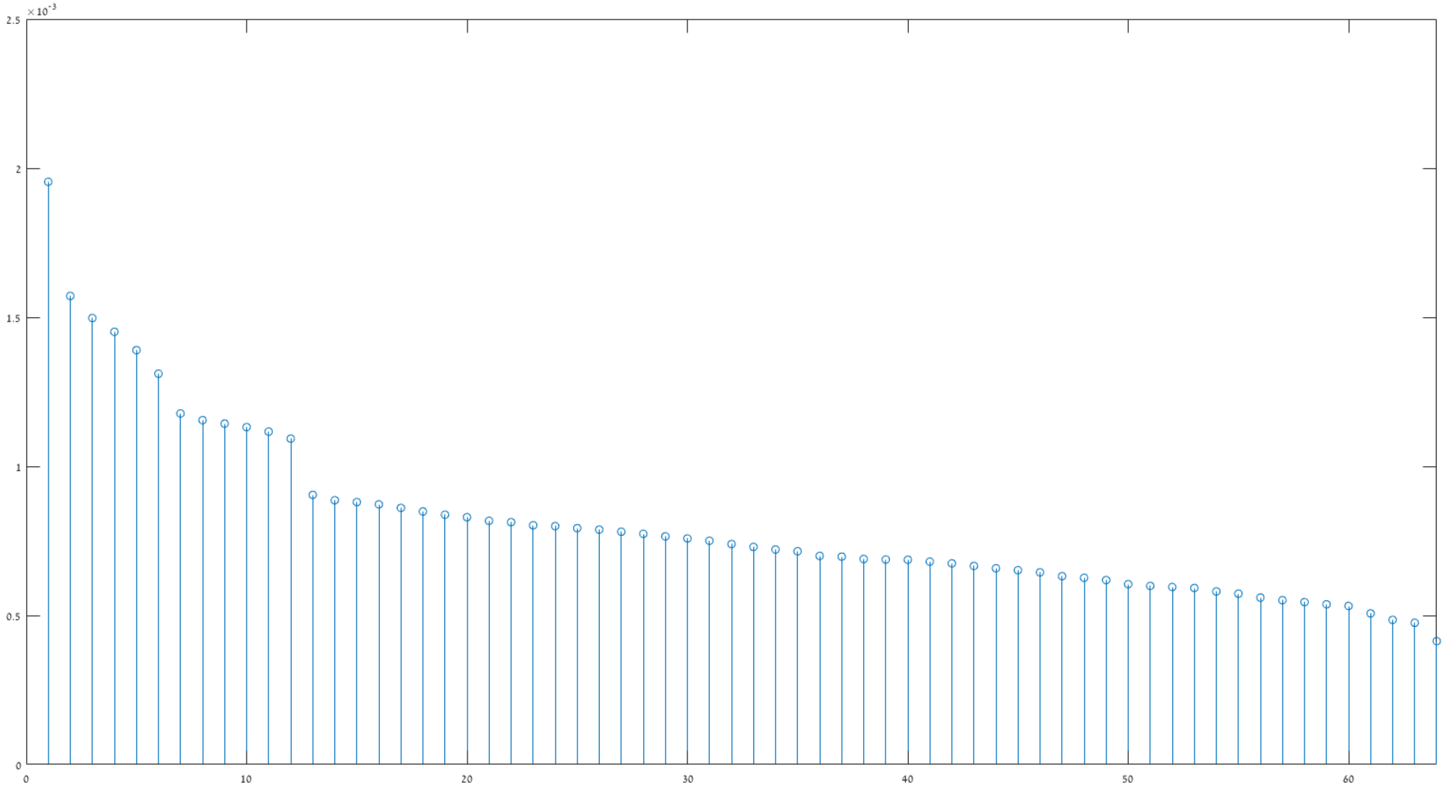
Side Channel Attack

- However, key space size is 2^{128}
- **Enumerating the whole key** space in optimal-order is **impossible**
- Hence, estimating a rank **without enumeration** is of great interest.

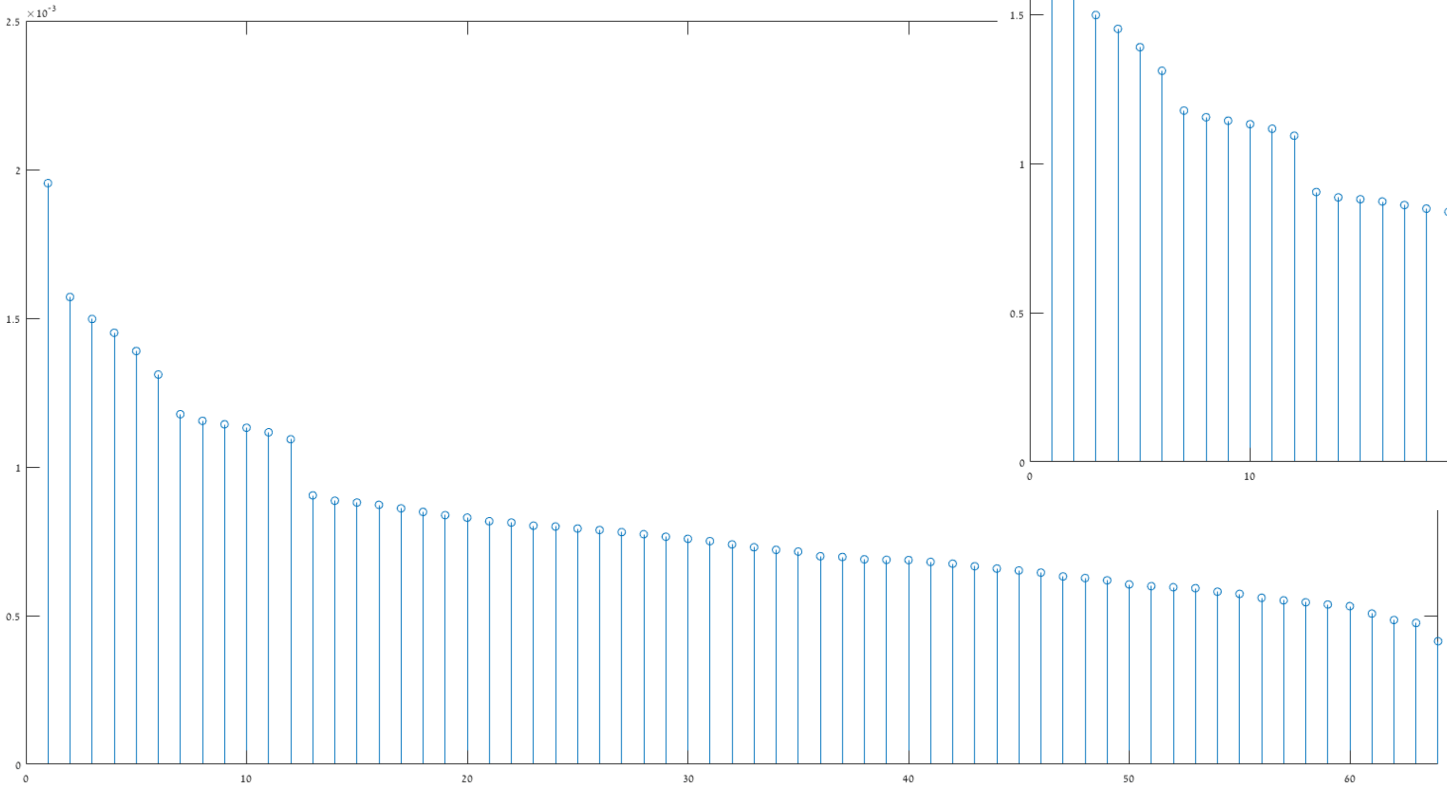


Our Rank Estimation: Motivation for d=2

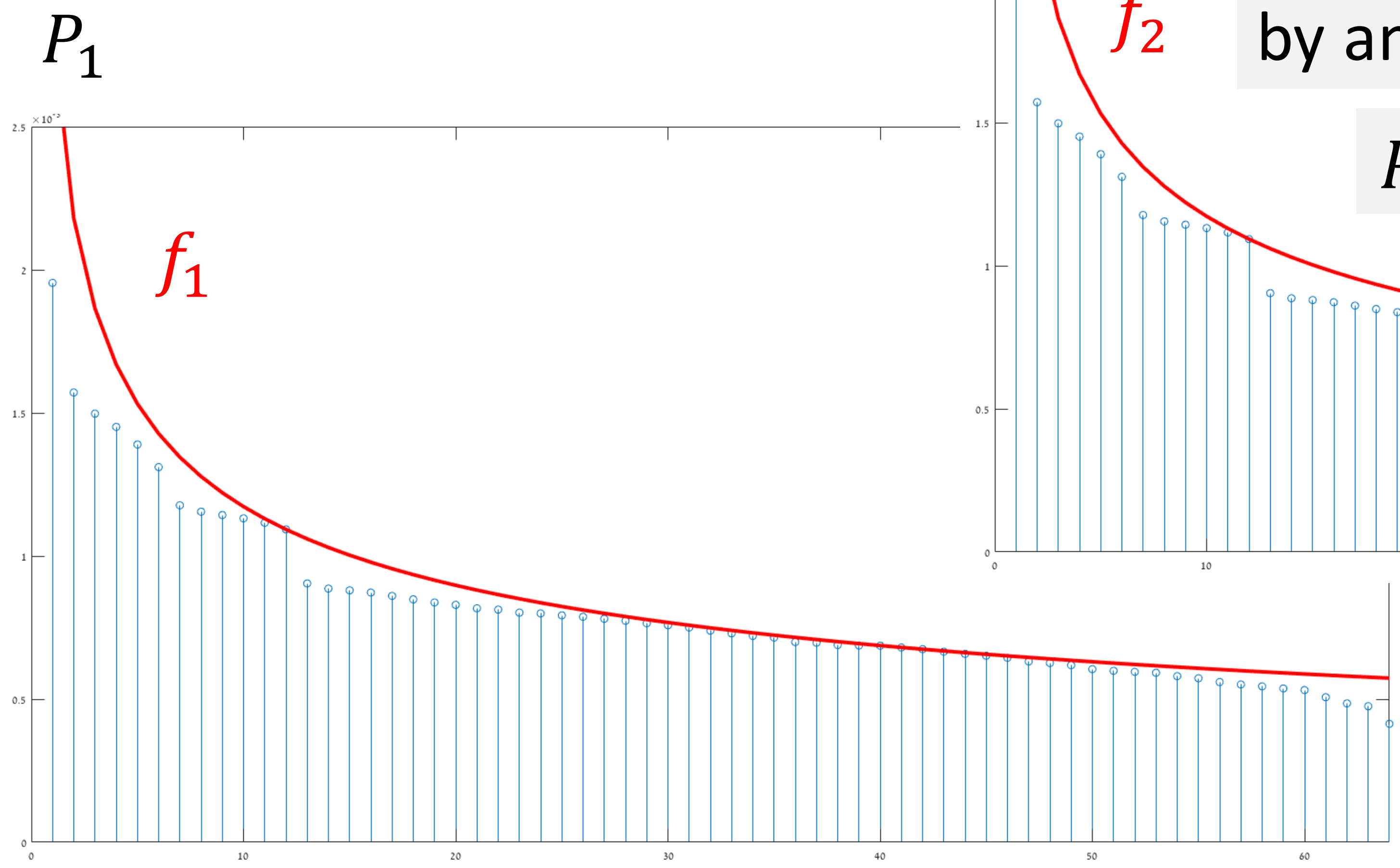
P_2

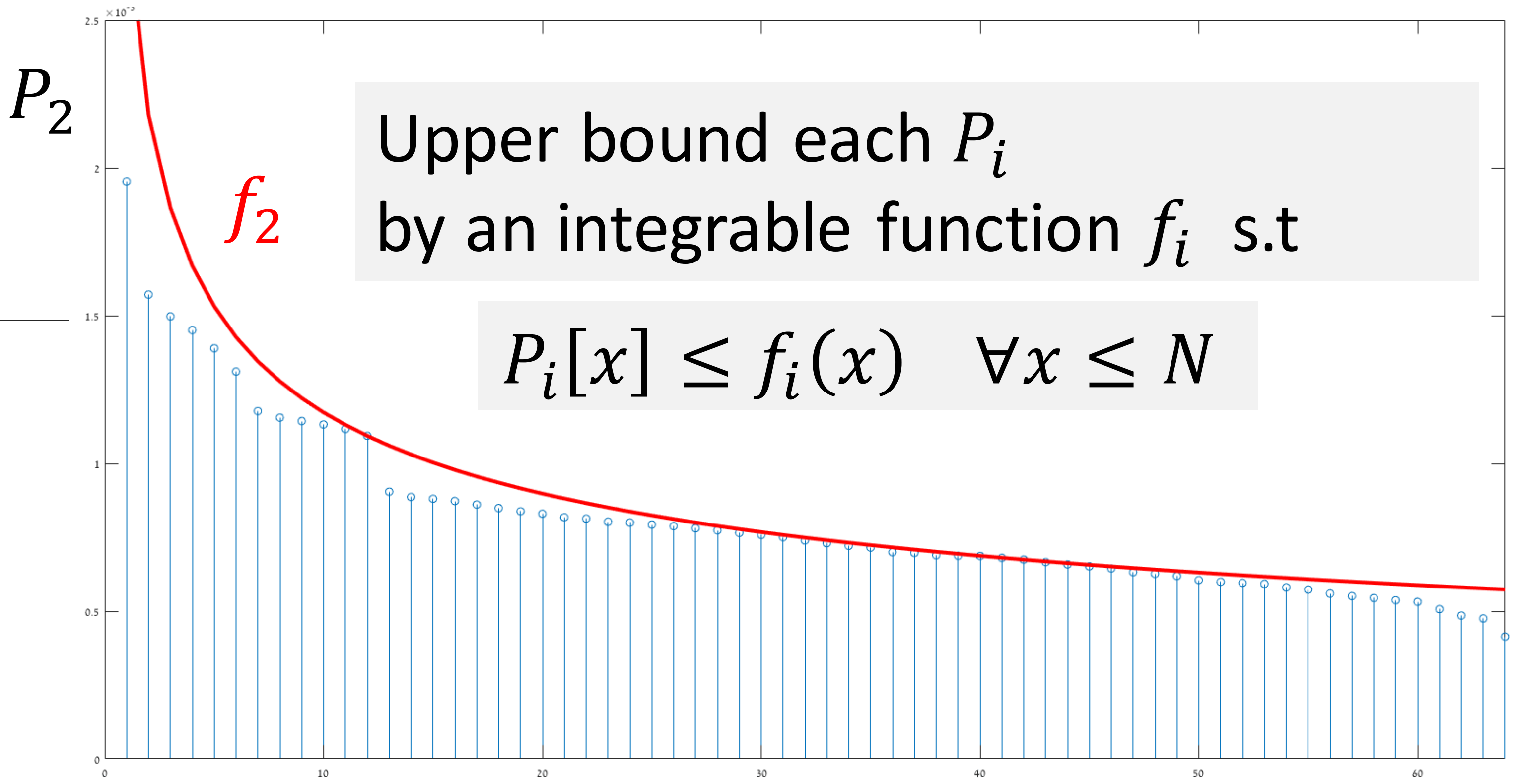


P_1



Our Rank Estimation: Motivation for d=2

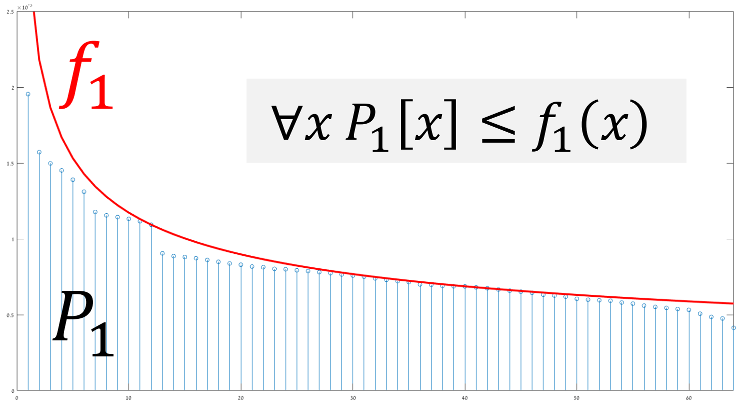


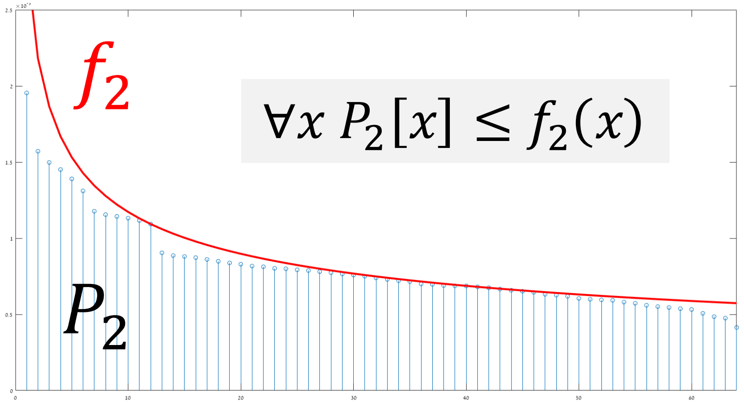


Upper bound each P_i
by an integrable function f_i s.t

$$P_i[x] \leq f_i(x) \quad \forall x \leq N$$

Motivation: d=2





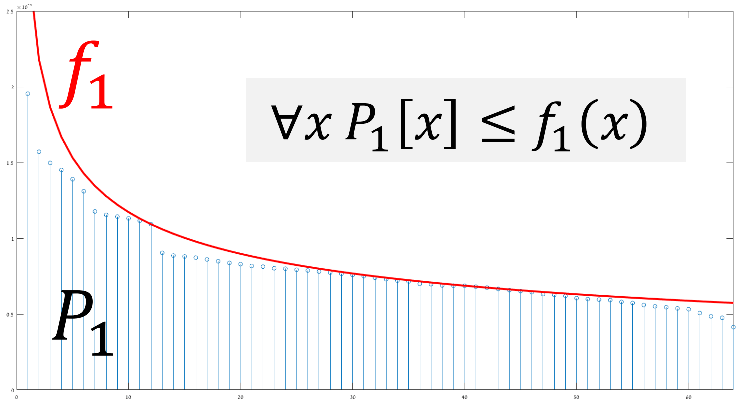
$$\forall x, y \le N$$

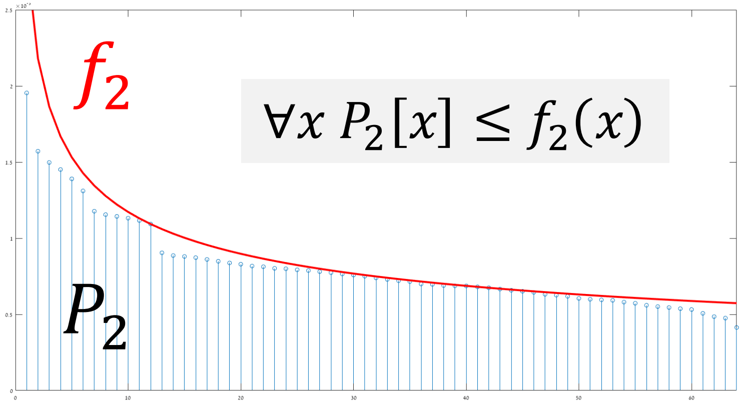
$$P_1[x] \cdot P_2[y] \le f_1(x) \cdot f_2(y)$$

$$\forall x, y \le N$$

$$p^* \le P_1[x] \cdot P_2[y] \Longrightarrow p^* \le f_1(x) \cdot f_2(y)$$

Motivation: d=2





$$\forall x, y \le N$$

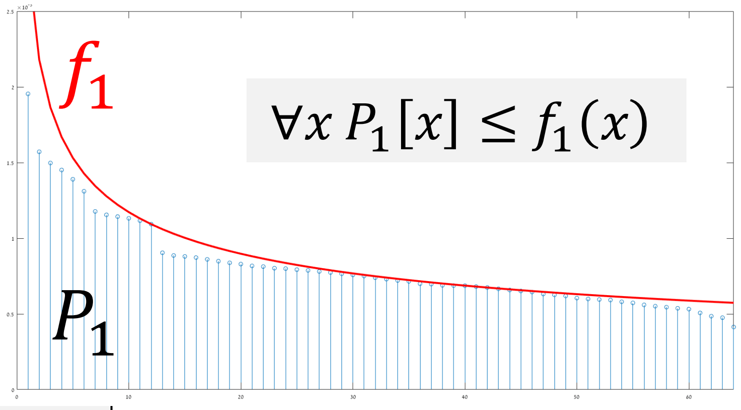
$$P_1[x] \cdot P_2[y] \le f_1(x) \cdot f_2(y)$$

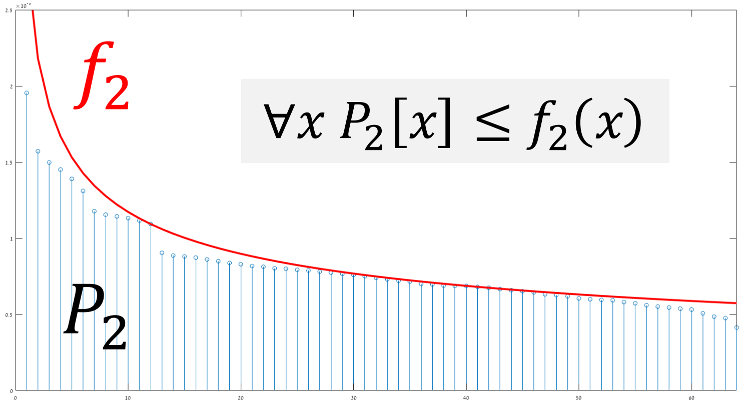
$$\forall x, y \le N$$

$$p^* \le P_1[x] \cdot P_2[y] \Longrightarrow p^* \le f_1(x) \cdot f_2(y)$$

The number of (x, y) s.t $p^* \le P_1[x] \cdot P_2[y]$ $\le$ The number of (x, y) s.t $p^* \le f_1(x) \cdot f_2(y)$

Motivation: d=2





$$\forall x, y \le N$$

$$P_1[x] \cdot P_2[y] \le f_1(x) \cdot f_2(y)$$

$$\forall x, y \le N$$

$$p^* \le P_1[x] \cdot P_2[y] \Longrightarrow p^* \le f_1(x) \cdot f_2(y)$$

The number of (x, y) s.t $p^* \le P_1[x] \cdot P_2[y]$ $\le$ The number of (x, y) s.t $p^* \le f_1(x) \cdot f_2(y)$

$$\text{rank}(k^*) \le \int_0^N \int_0^N 1 \, dx \, dy$$

$f_1(x) \cdot f_2(y) \ge p^*$

Instantiating the framework

For f we select the **Pareto function**:

$$f(x) = \frac{a}{x^{\alpha}}$$

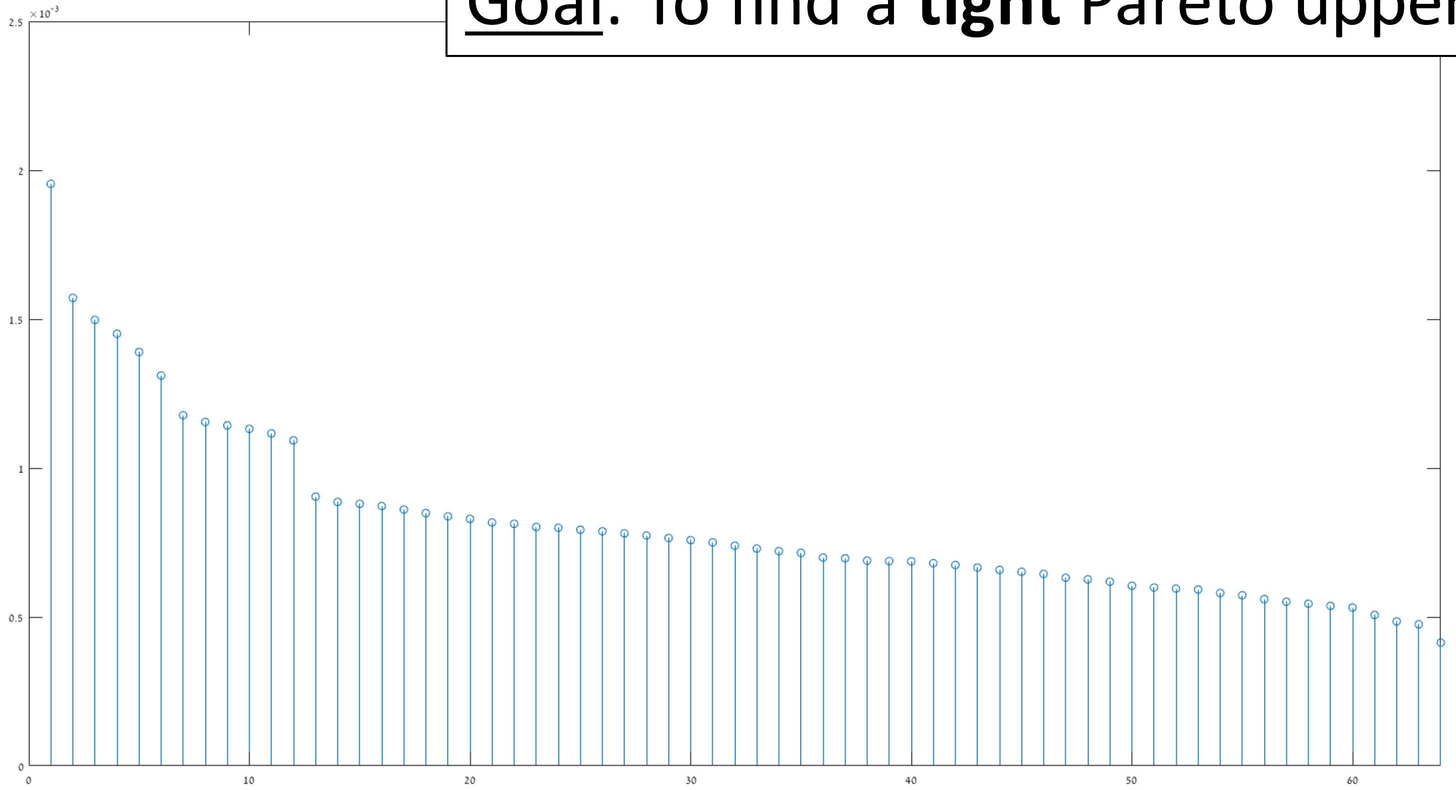
- Long tail
- Easy to calculate mutiple integral

Choosing the best Pareto upper bound

Given a non-increasing probability distribution P

<u>Goal</u>: To find a **tight** Pareto upper bound for P

P



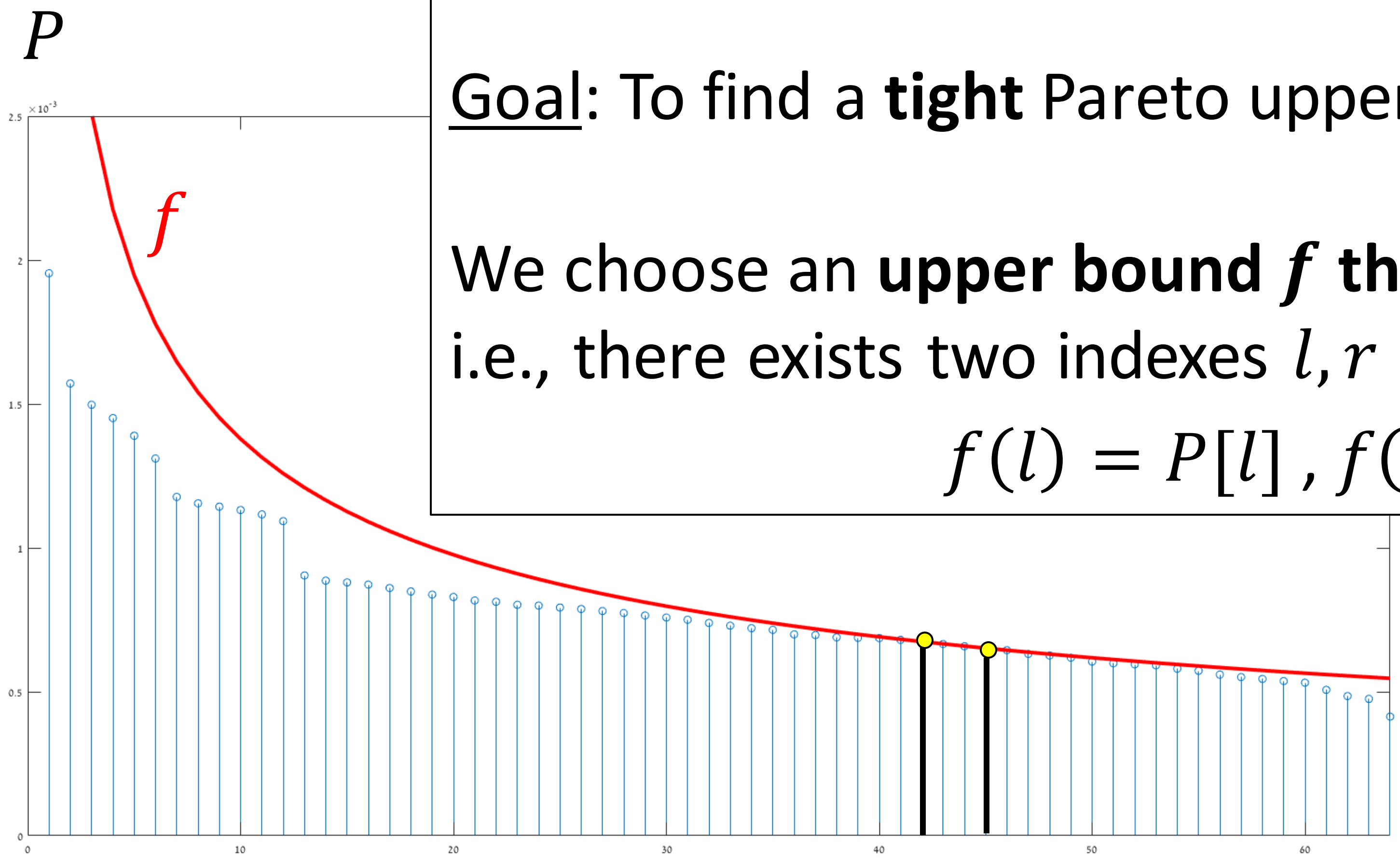
Choosing the best Pareto upper bound

Given a non-increasing probability distribution P

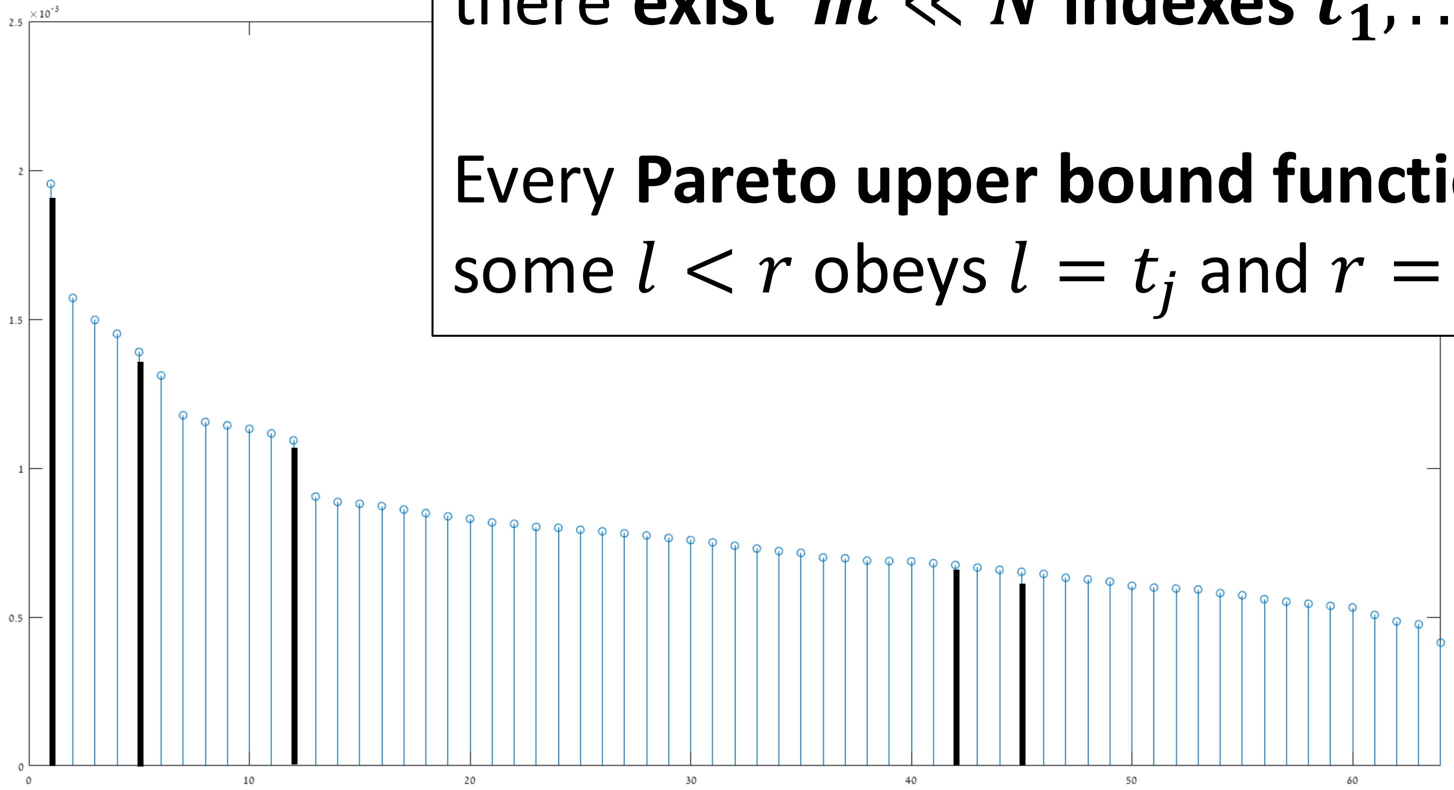
<u>Goal</u>: To find a **tight** Pareto upper bound for P

We choose an **upper bound f that anchors at two indexes,** i.e., there exists two indexes l, r s.t

$$f(l) = P[l]\,,\, f(r) = P[r]$$



Choosing the best Pareto upper bound

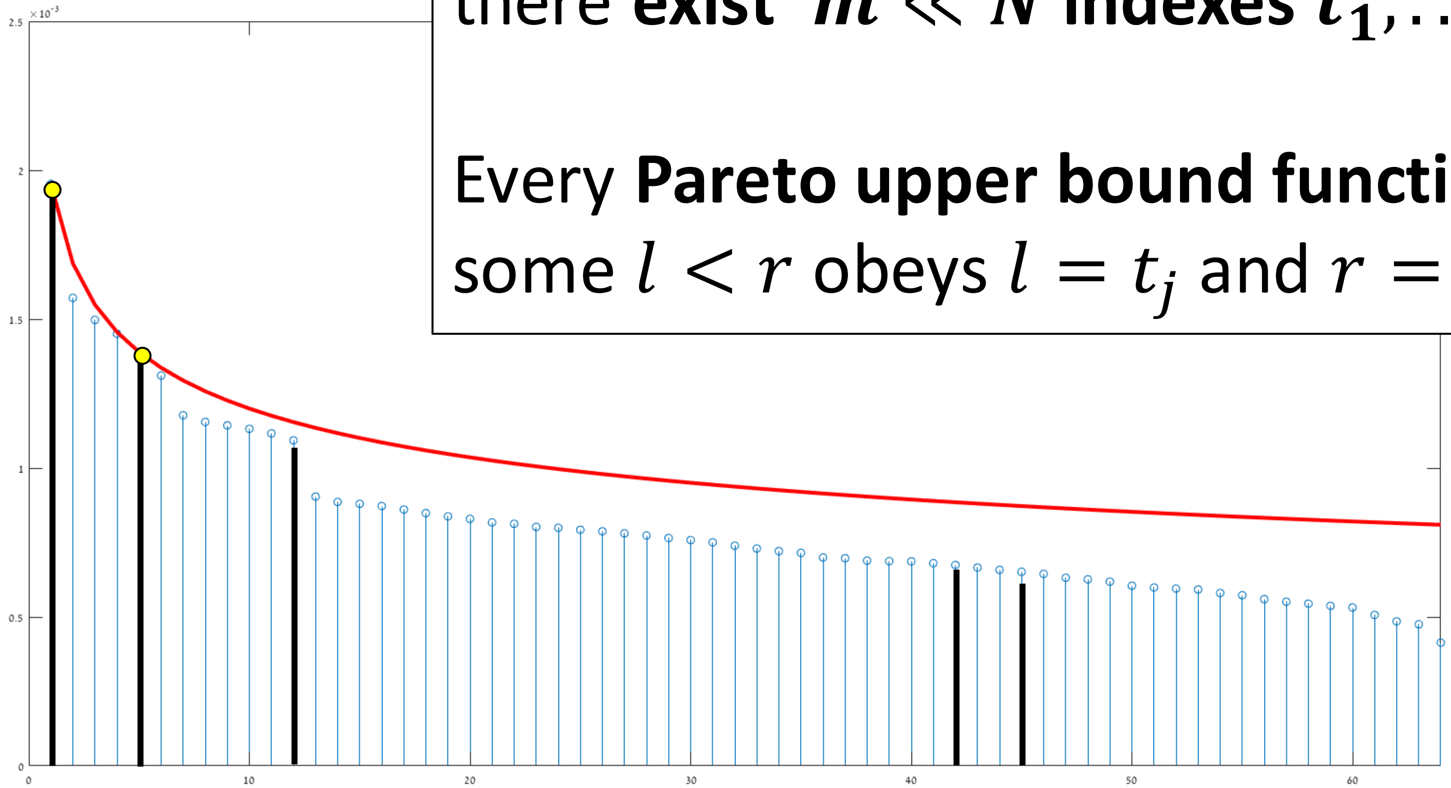


Given a non-increasing probability distribution P

there **exist** $\boldsymbol{m \ll N}$ **indexes** $\boldsymbol{t_1, .., t_m}$ s.t

Every **Pareto upper bound function of** $\boldsymbol{P}$ that is anchored at some $l < r$ obeys $l = t_j$ and $r = t_{j+1}$ for $1 \leq j < m$

Choosing the best Pareto upper bound



Given a non-increasing probability distribution P

there **exist** $\boldsymbol{m} \ll \boldsymbol{N}$ **indexes** $\boldsymbol{t_1, .., t_m}$ s.t

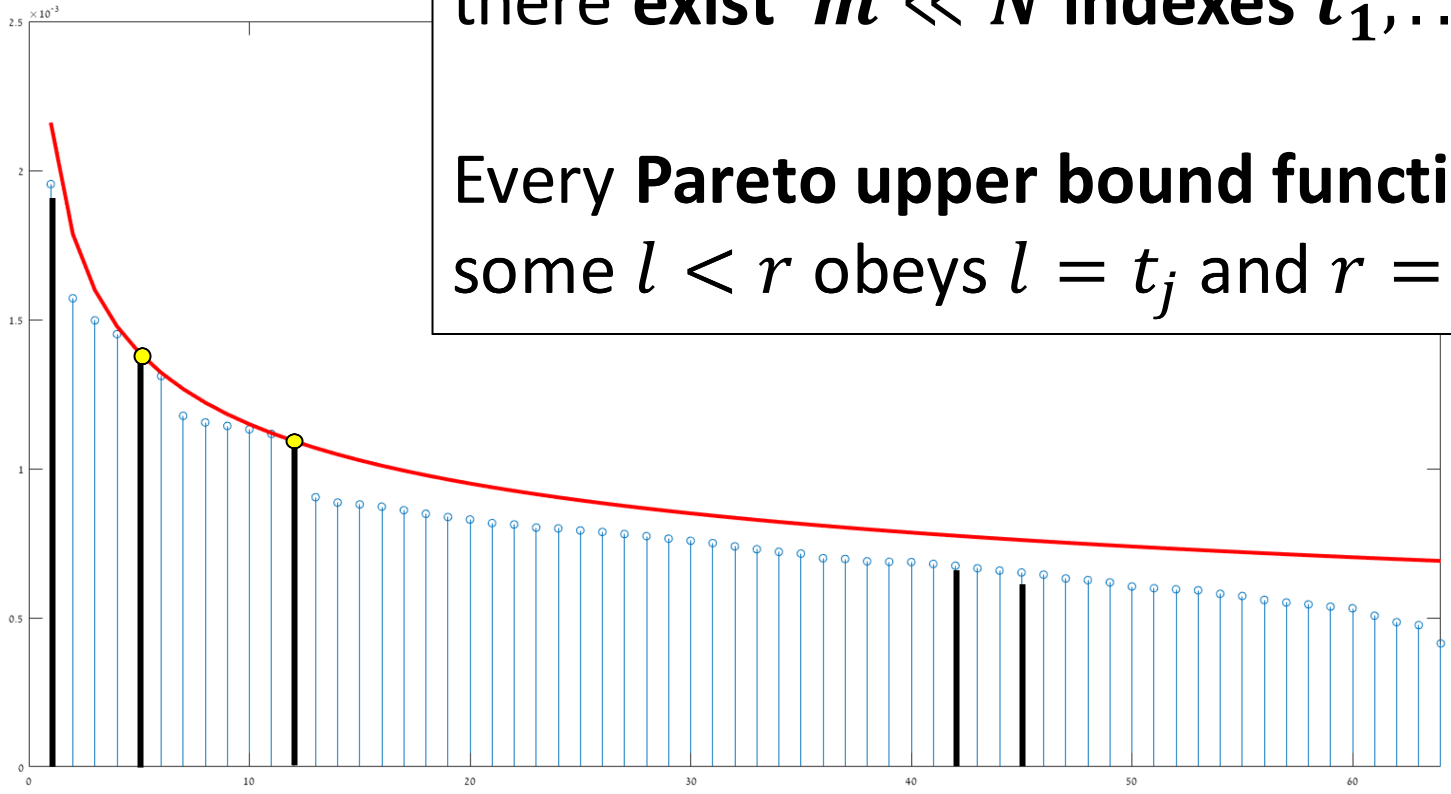
Every **Pareto upper bound function of** $\boldsymbol{P}$ that is anchored at some $l < r$ obeys $l = t_j$ and $r = t_{j+1}$ for $1 \leq j < m$

Choosing the best Pareto upper bound

Given a non-increasing probability distribution P

there **exist** $\boldsymbol{m \ll N}$ **indexes** $\boldsymbol{t_1, .., t_m}$ s.t

Every **Pareto upper bound function of** $\boldsymbol{P}$ that is anchored at some $l < r$ obeys $l = t_j$ and $r = t_{j+1}$ for $1 \leq j < m$

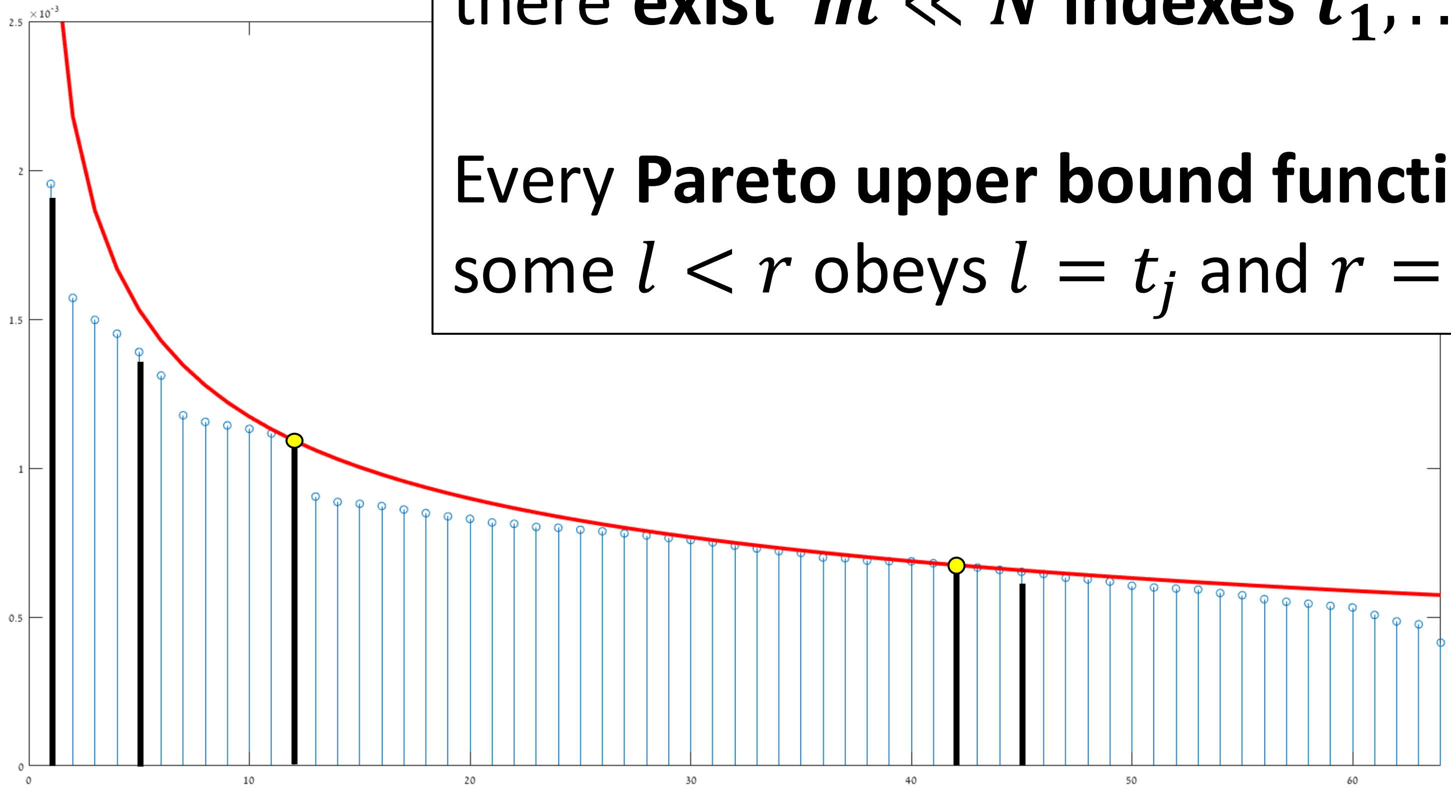


Choosing the best Pareto upper bound

Given a non-increasing probability distribution P

there **exist** $\boldsymbol{m} \ll \boldsymbol{N}$ **indexes** $\boldsymbol{t_1,..,t_m}$ s.t

Every **Pareto upper bound function of $\boldsymbol{P}$** that is anchored at some $l < r$ obeys $l = t_j$ and $r = t_{j+1}$ for $1 \leq j < m$

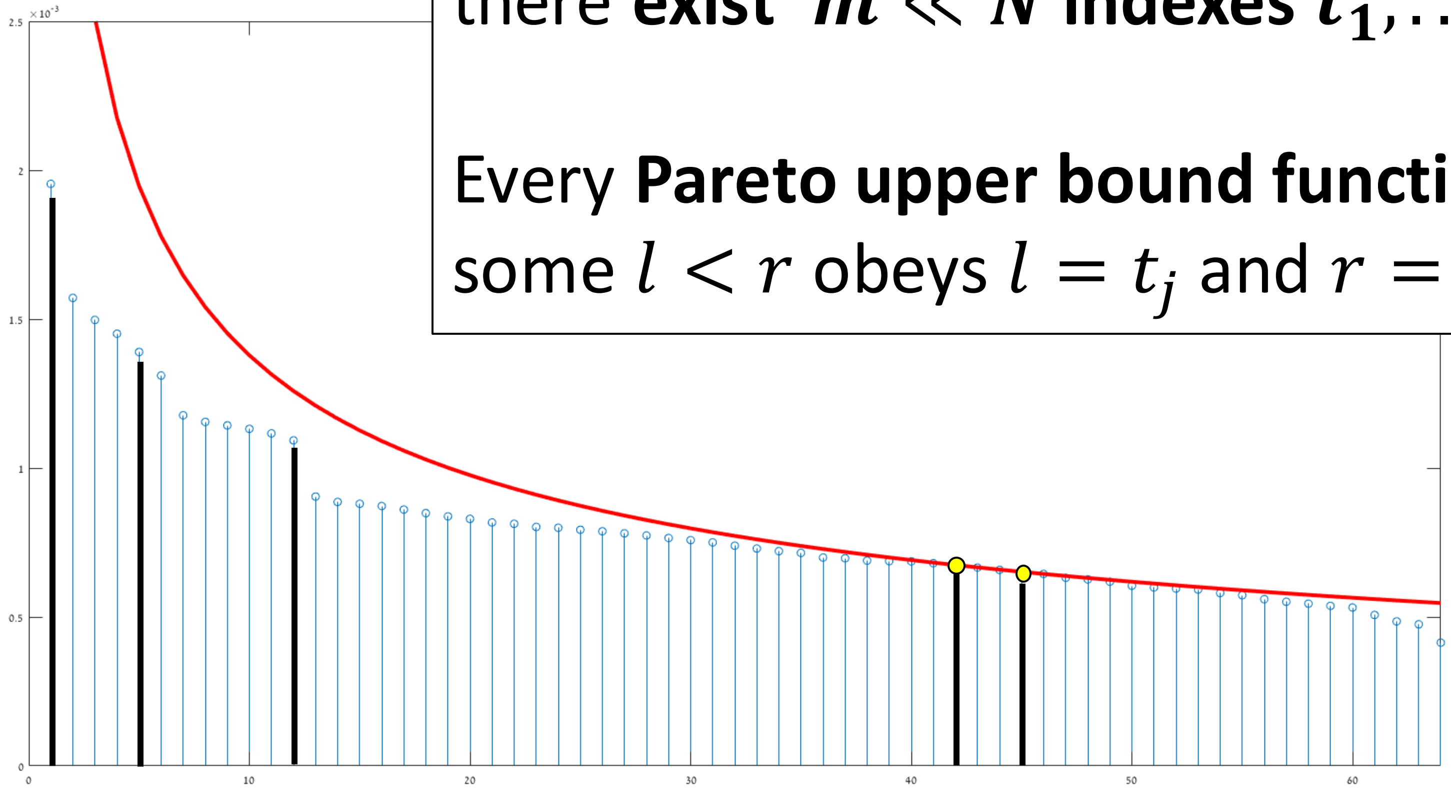


Choosing the best Pareto upper bound

Given a non-increasing probability distribution P

there **exist** $\boldsymbol{m} \ll \boldsymbol{N}$ **indexes** $\boldsymbol{t_1, .., t_m}$ s.t

Every **Pareto upper bound function of** $\boldsymbol{P}$ that is anchored at some $l < r$ obeys $l = t_j$ and $r = t_{j+1}$ for $1 \leq j < m$



Choosing the best Pareto upper bound

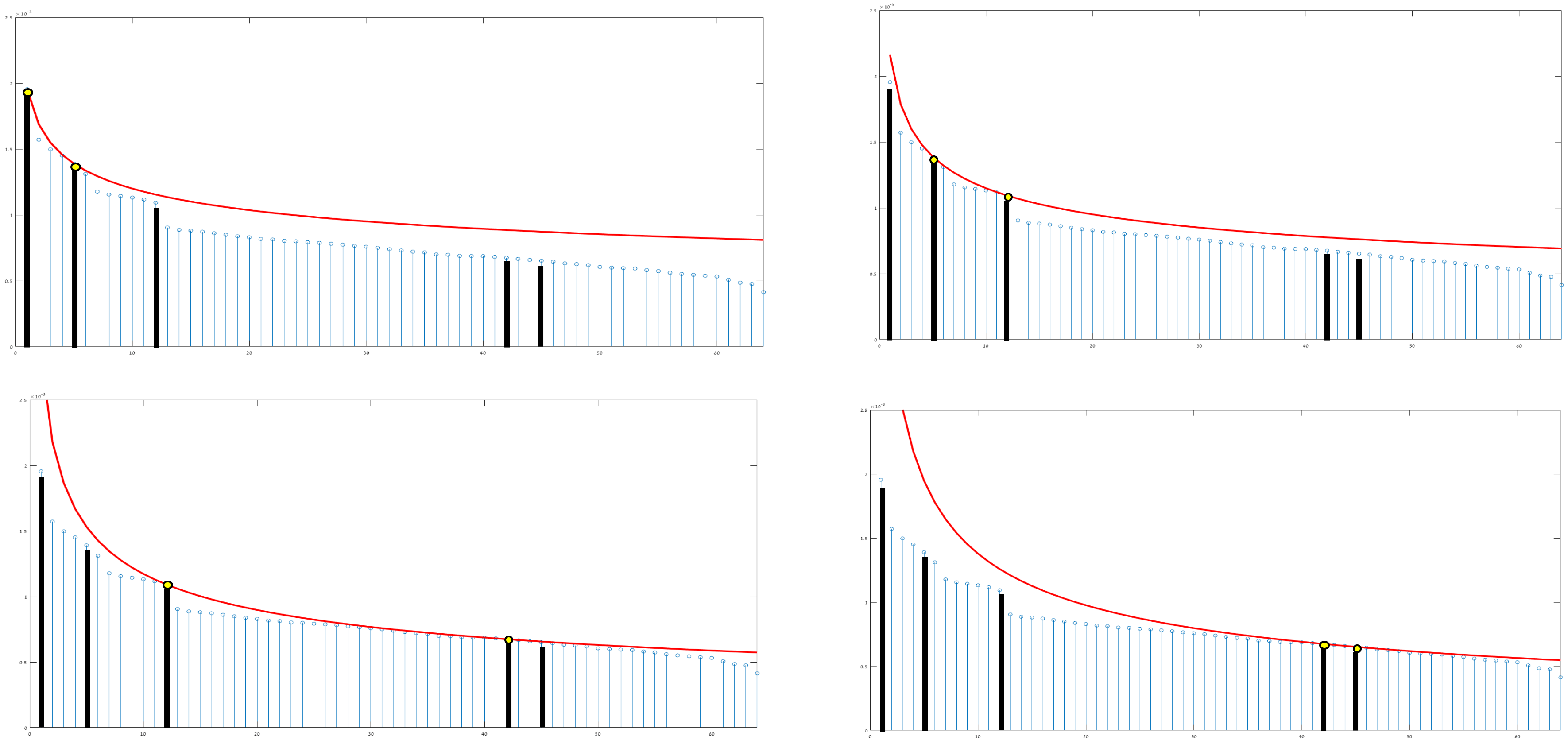
The **asymptotic running time of finding all** the **Pareto** upper bounds of a given P is $\boldsymbol{O}(\boldsymbol{mN})$.

- Since typically $\boldsymbol{m} \ll \boldsymbol{N}$ the algorithm is **almost linear in $\boldsymbol{N}$** and **very quick in practice**.

- Furthermore, **our implementation is very efficient**:
  it allows **skipping** over hundreds of **not relevant candidates**
  which **dramaticly impacts in practice.**

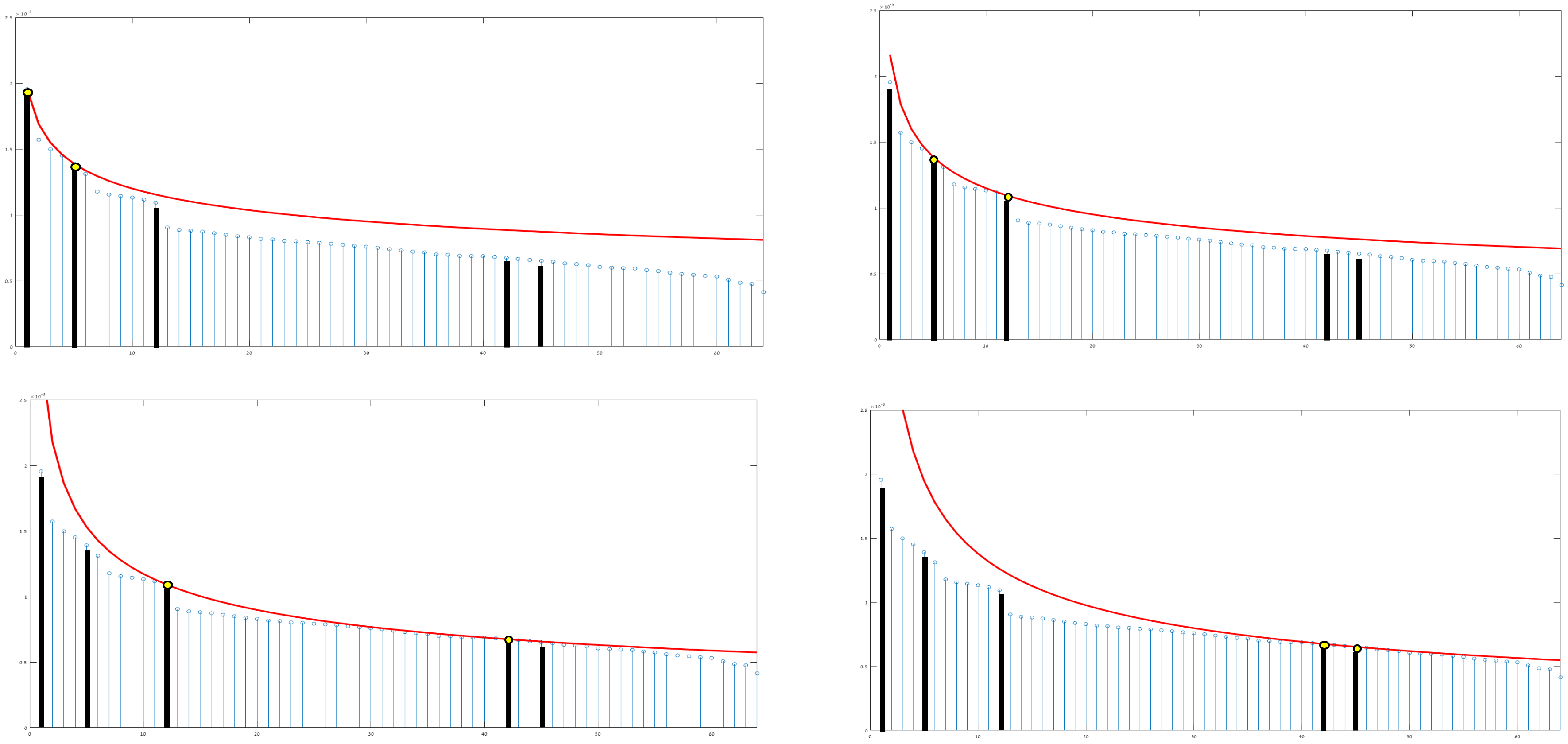
Choosing the best Pareto upper bound

After finding multiple candidates for **Pareto upper bound** of a given P,



Choosing the best Pareto upper bound

We need to **select** the '**best**' function which lead to a **tight bound**.



Choosing the best Pareto upper bound

We chose the following **criteria**:

Given

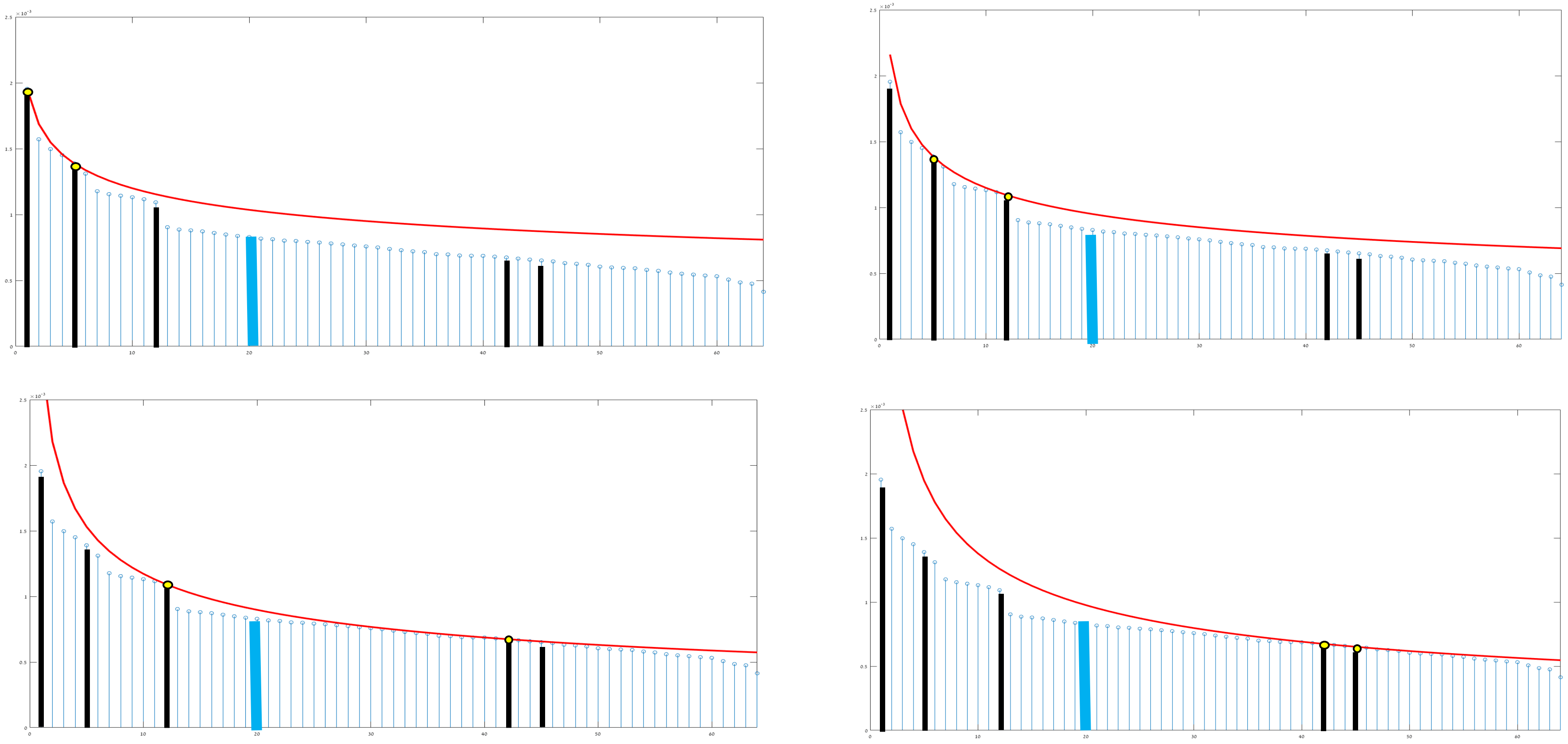
- P - a non-increasing subkey probability distribution
- k - the index of the correct subkey in P

Choose the Pareto upper bound function f s.t

$$\boldsymbol{f(k)} \textbf{ is the closest to } \boldsymbol{P[k]}$$

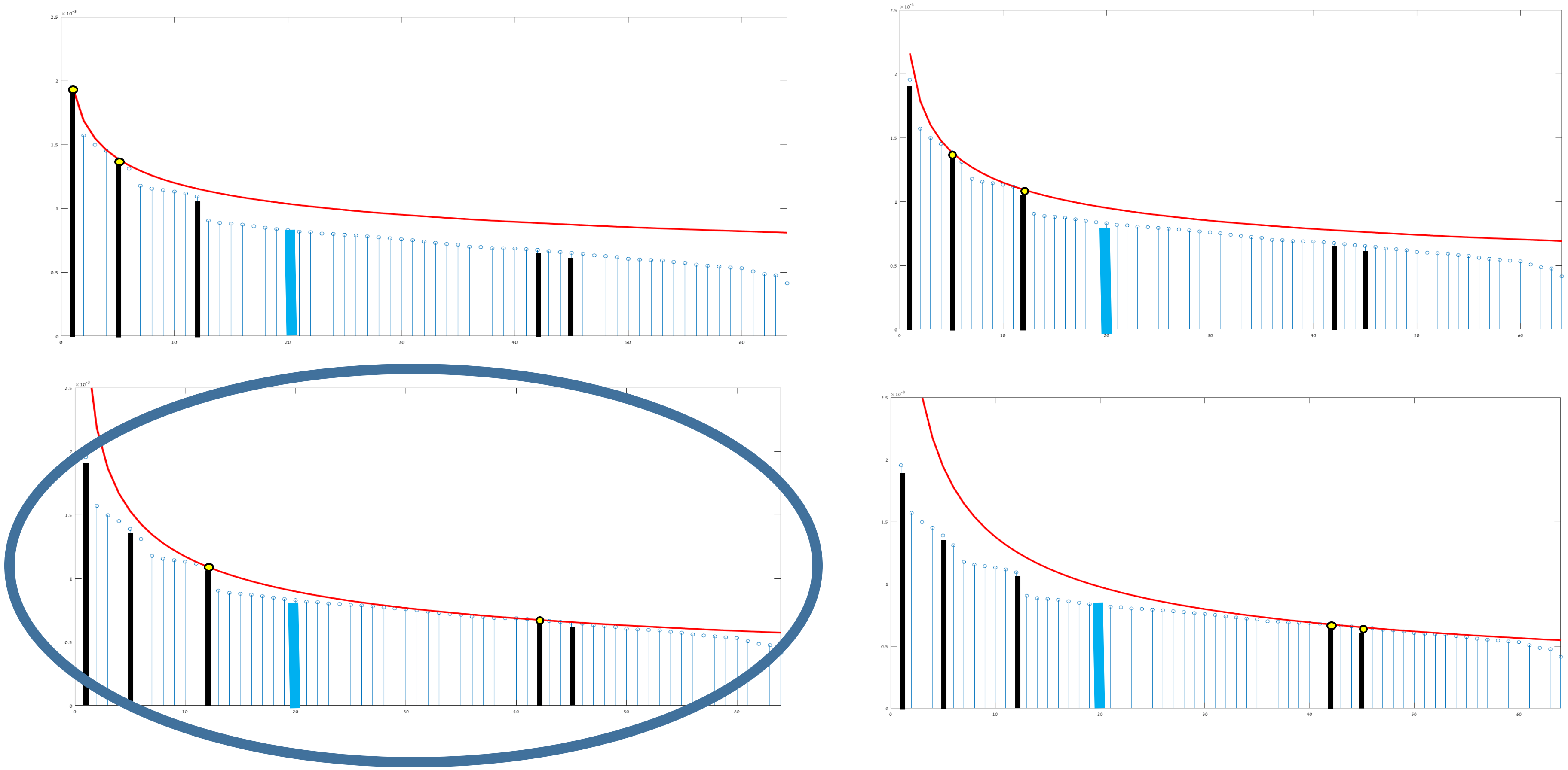
Choosing the best Pareto upper bound

$f(k)$ is the closest to $P[k]$



Choosing the best Pareto upper bound

$f(k)$ is the closest to $P[k]$



Estimating the Volume for d≥2

After we find the 'best' Pareto upper bound function f_i for each P_i

$$\forall x \quad P_i[x] \le f_i(x) = \frac{a_i}{x^{\alpha_i}}$$

We need to calculate the number of $(x_1, x_2, \dots, x_d)$ s.t.

$f_1(x_1) \cdot f_2(x_2) \cdot \dots \cdot f_d(x_d) \ge p^*$

using the multiple integral:

$$\underset{f_1(x_1)\cdot f_2(x_2)\cdot \dots \cdot f_d(x_d) \ge p^*}{\int_0^N \int_0^N \dots \int_0^N 1 \, dx_1 \, dx_2 \dots dx_d}$$

Estimating the Volume for d≥2

We solve the multiple integral: $$\int_0^N \int_0^N \dots \int_0^N 1\, dx_1\, dx_2 \dots dx_d$$

$$f_1(x_1) \cdot f_2(x_2) \cdot \dots \cdot f_d(x_d) \geq p^*$$

using the Pareto upper bound functions $f_i(x) = \frac{a_i}{x^{\alpha_i}}$

We get the following closed formula:

$$\text{rank}(p^*) \leq \sum_{i=1}^{d} \left[\left(\frac{1}{p^*} \cdot \prod_{j=1}^{d} a_j \right)^{\frac{1}{\alpha_i}} \cdot \prod_{j=1, j \neq i}^{d} \left(\frac{\alpha_i}{\alpha_i - \alpha_j} \cdot N^{\frac{\alpha_i - \alpha_j}{\alpha_i}} \right) \right]$$

PRank: The Pareto Rank Estimation Algorithm

Given:

- d probability distiburions $P_1, .., P_d$
- The correct key $k^* = (k_1, \dots, k_d)$ and its probability p^*

Prank Algorithm:

for $i = 1$ to d:

$a_i, \alpha_i \leftarrow$ upper bound P_i by a Pareto upper bound function

compute the closed formula: $$\sum_{i=1}^{d}\left[\left(\frac{1}{p^*}\cdot\prod_{j=1}^{d}a_j\right)^{\frac{1}{\alpha_i}}\cdot\prod_{j=1,j\neq i}^{d}\left(\frac{\alpha_i}{\alpha_i-\alpha_j}\cdot N^{\frac{\alpha_i-\alpha_j}{\alpha_i}}\right)\right]$$

Theoretical Worst-case Performance

Prank Algorithm

for $i = 1$ to d:

$a_i, \alpha_i \leftarrow$ upper bound P_i by a Pareto upper bound function

compute the closed formula: $$\sum_{i=1}^{d}\left[\left(\frac{1}{p^*}\cdot\prod_{j=1}^{d}a_j\right)^{\frac{1}{\alpha_i}}\cdot\prod_{j=1,j\neq i}^{d}\left(\frac{\alpha_i}{\alpha_i-\alpha_j}\cdot N^{\frac{\alpha_i-\alpha_j}{\alpha_i}}\right)\right]$$

Space Complexity:

Only needs to keep a_i, α_i for every $1 \leq i \leq d$

Therefore $\boldsymbol{O(d)}$.

Theoretical Worst-case Performance

Prank Algorithm

for $i = 1$ to d:

$a_i, \alpha_i \leftarrow$ upper bound P_i by a Pareto upper bound function

compute the closed formula: $$\sum_{i=1}^{d}\left[\left(\frac{1}{p^*}\cdot\prod_{j=1}^{d}a_j\right)^{\frac{1}{\alpha_i}}\cdot\prod_{j=1,j\neq i}^{d}\left(\frac{\alpha_i}{\alpha_i-\alpha_j}\cdot N^{\frac{\alpha_i-\alpha_j}{\alpha_i}}\right)\right]$$

Running Time:

Calculating the closed formula: $\boldsymbol{O}(\boldsymbol{d}^{2})$

d additions each consists of d multiplications and d real-value power.

Theoretical Worst-case Performance

Prank Algorithm

for $i = 1$ to d:

$a_i, \alpha_i \leftarrow$ upper bound P_i by a Pareto upper bound function

compute the closed formula: $$\sum_{i=1}^{d}\left[\left(\frac{1}{p^*}\cdot\prod_{j=1}^{d}a_j\right)^{\frac{1}{\alpha_i}}\cdot\prod_{j=1,j\neq i}^{d}\left(\frac{\alpha_i}{\alpha_i-\alpha_j}\cdot N^{\frac{\alpha_i-\alpha_j}{\alpha_i}}\right)\right]$$

Running Time:

Finding the best Pareto upper bound for each P_i is $\boldsymbol{O(m_i \cdot N)}$.

Since typically $\forall i \;\; m_i \ll N$,

the algorithm **is almost linear in $\boldsymbol{dN}$ and very quick in practice.**

Performance Evaluation

- We **compared** our **new PRank algorithm** with the **histogram** algorithm of Glowacz et al. [GGPSS15].
- We **implemented both in Matlab**.
- Our **PRank code is available in gitHub**.

Performance Evaluation

- We run PRank algorithm on 611 traces gathered from a specific SCA.
- The SCA was against **AES with 128-bits keys**.
- Each set in the corpus consists of the correct secret key and **16 distributions**, one per subkey.
- The distributions are **sorted** in **non-increasing** order of probability, **each of length 2^8**.

Performance Evaluation

- We measured the **time** and the **accuracy** for each trace using **PRank** and the **histograms** rank estimation, in two different configurations.
  - d=16 and $n=2^8$
  - d=8 and $n=2^{16}$

We used the **histogram rank** as the **x-axis** in our resulting graphs.

Space Utilization

PRank Histograms

	B=5K		B=50K	
$d = 8$	24 bytes	80KB	24 bytes	800KB
$d = 16$	48 bytes	160KB	48 bytes	1.6MB

The **memory consumption of PRank algorithm is drastically lower** than the histogram space consumption.

The **PRank space consumption is trivial $3d$**
The histogram space requirements are around $2Bd$

Runtime Analysis

The PRank running time consists of:

- **finding** the **Pareto upper bound** function of each probability distribution
- **calculating** the **closed formula** given the secret key.

The histogram running time consists of:

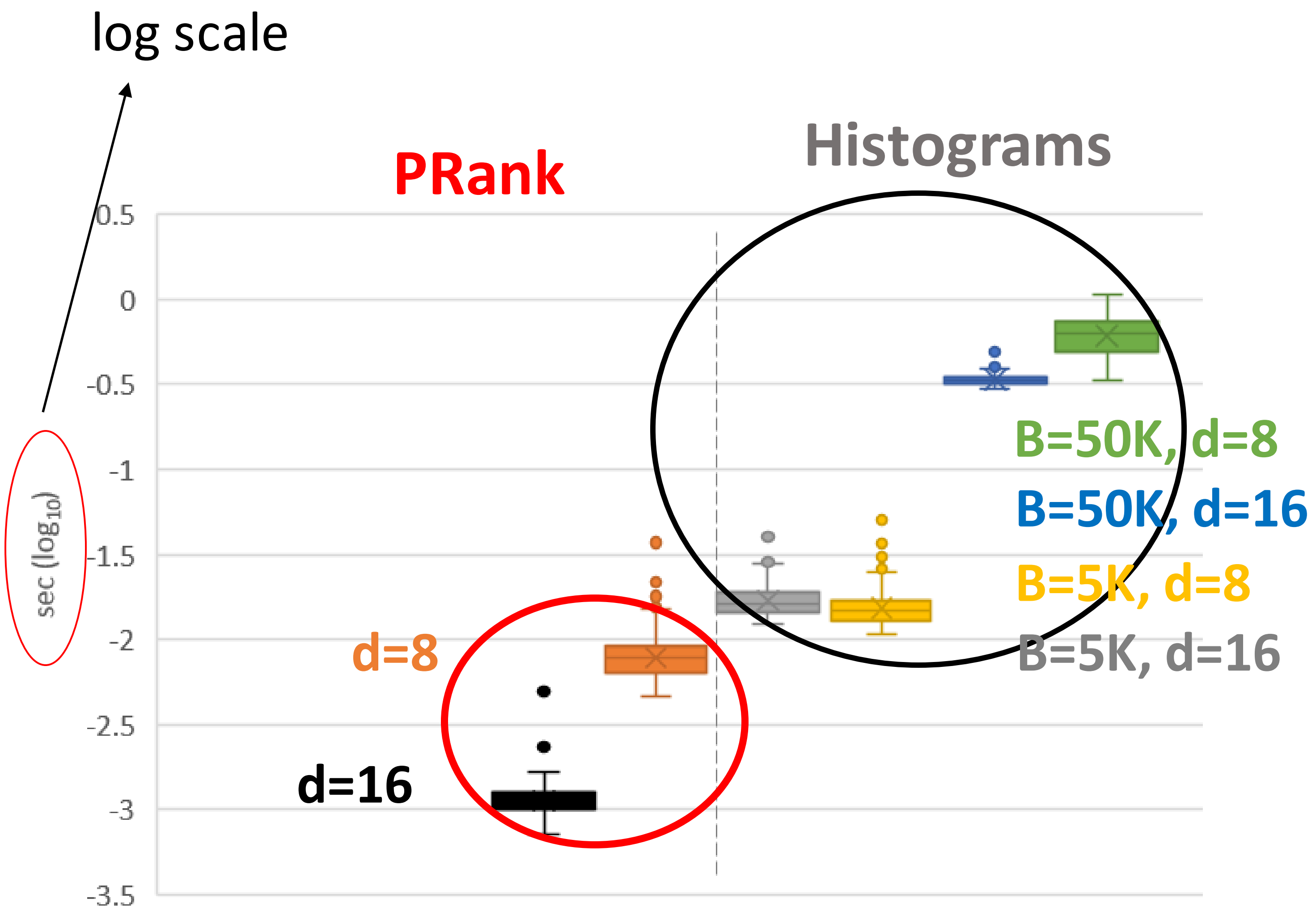
- **converting** each probability distribution **into a histogram**
- **finding** the **sum of the corresponding bins** given the secret key.

Runtime Analysis

PRank, for both d=8 and d=16, typically

takes only a few milliseconds to complete

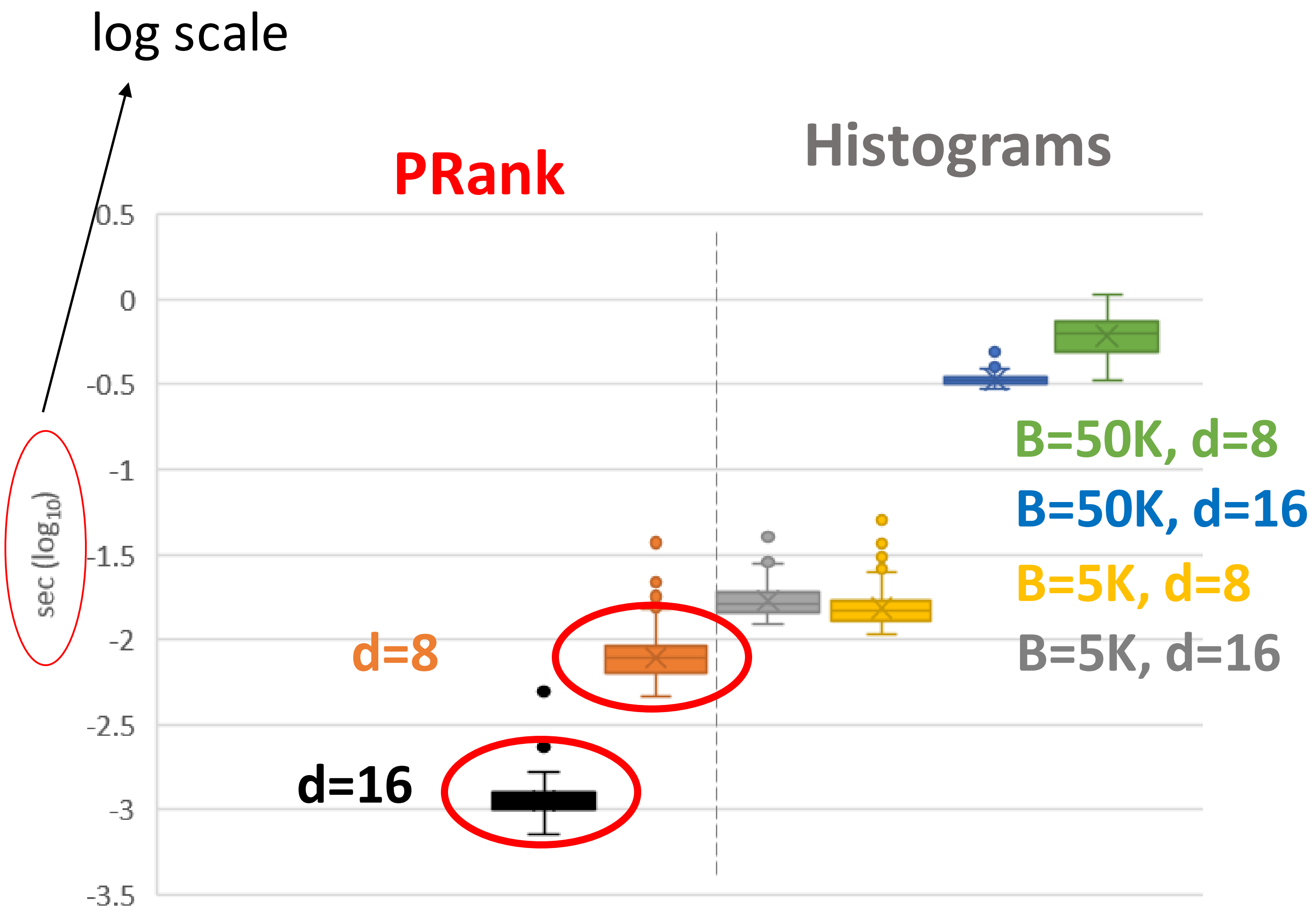
and **runs faster** than the Histograms in its 4 configurations.



Runtime Analysis

Prank with **d=16 runs faster** than PRank with d=8

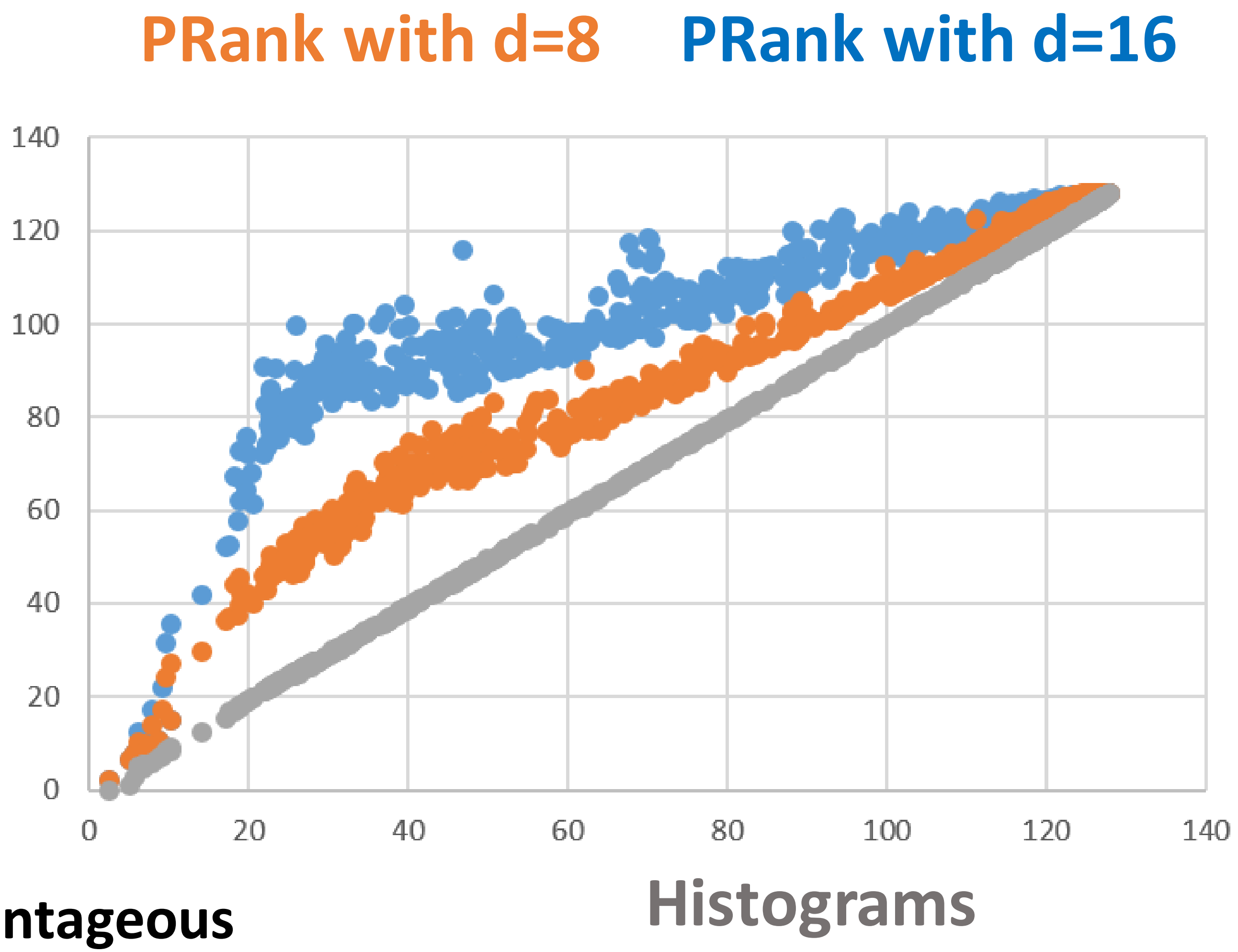
since the **length N** of each distribution is **shorter**.



Bound Tightness

The Figure illustrates the **PRank upper bound** with **d=16**, **d=8** and the **histogram rank**,
all in **number of bits** (log2).

x-axis is the **number of bits** of **histogram** rank, hence its curve is a straight line.

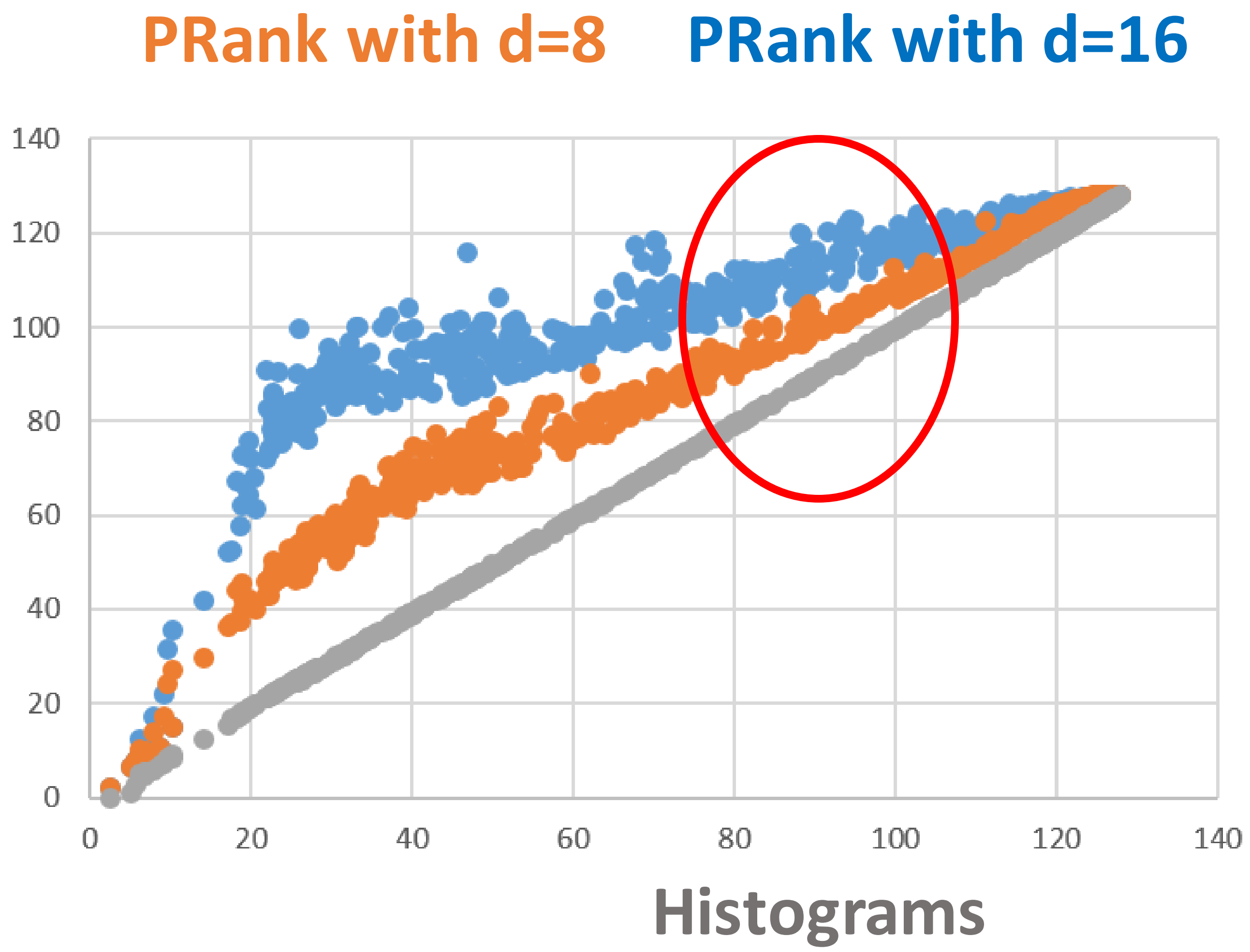


The figure clearly shows that it is **advantageous** to **reduce** the **dimension d.**

Bound Tightness

The **accuracy** of PRank's estimation is **quite good**:

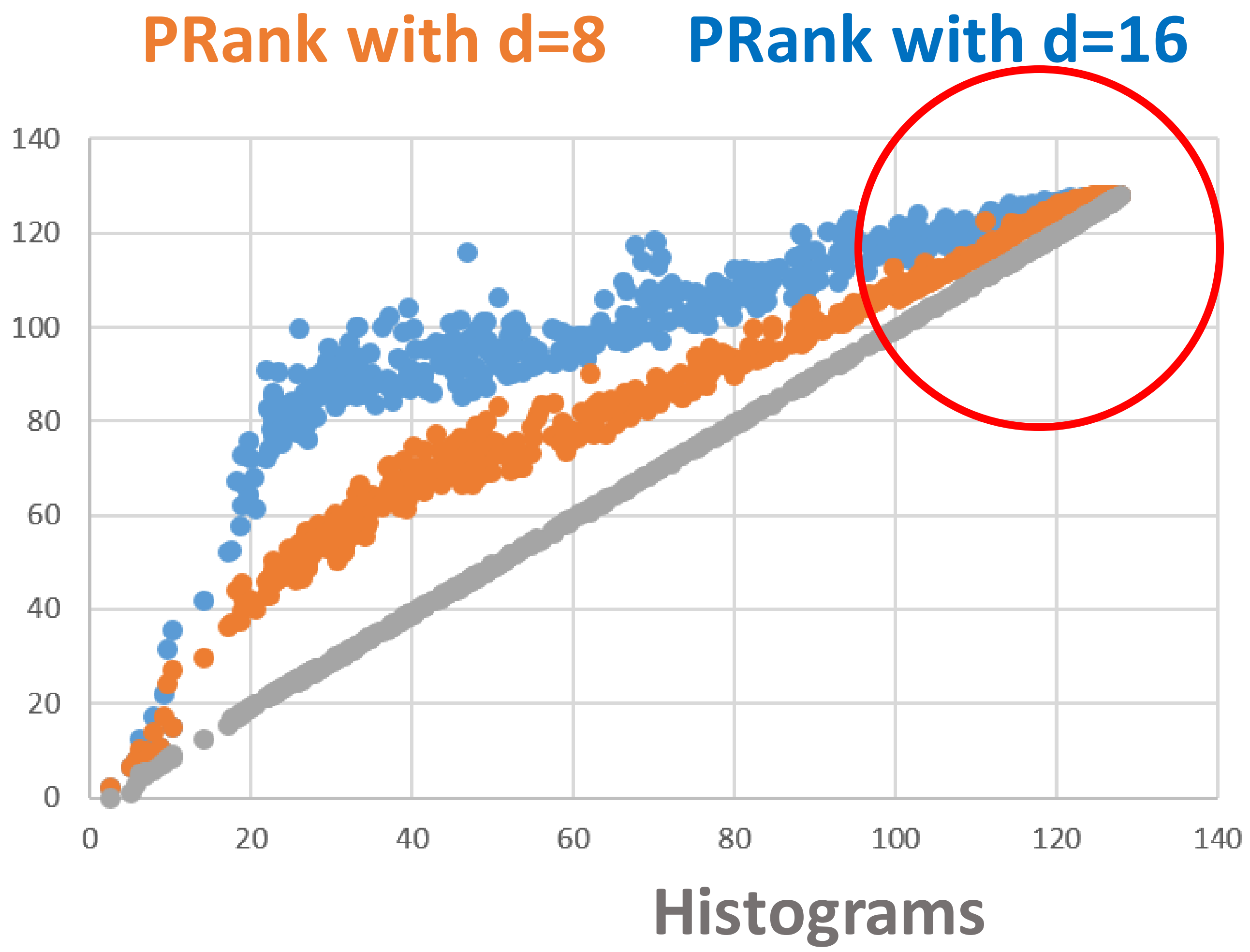
for ranks between 2^{80}–2^{100} :
The median **PRank** bound
is **less than 10 bits** above the
histogram rank.



Bound Tightness

The **accuracy** of PRank's estimation is **quite good**:

for high ranks above 2^{100}:
The median **PRank** bound is **less than 4 bits** more.



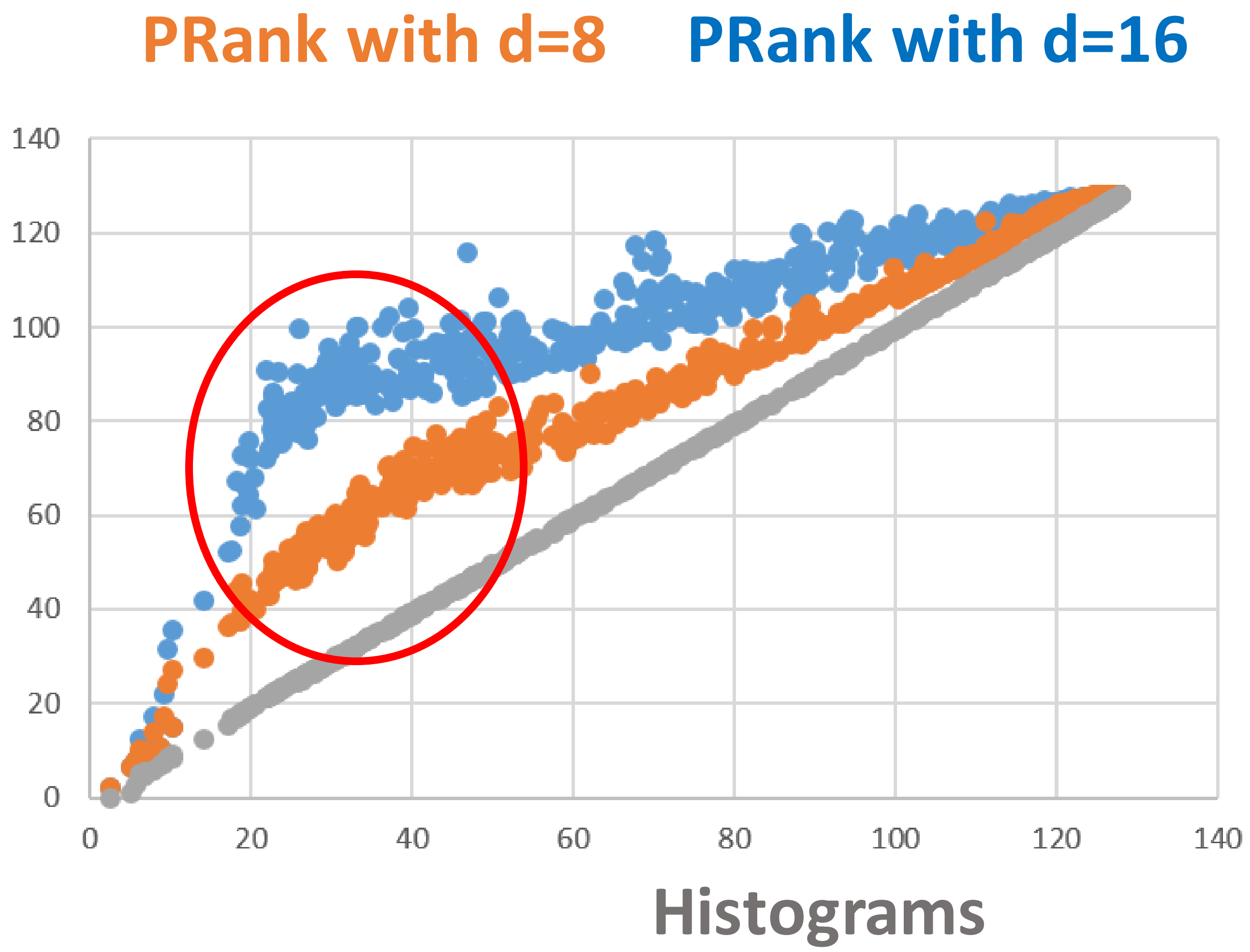
Bound Tightness

The **accuracy** of PRank's estimation is **quite good**:

For small ranks, around 2^{30}:

PRank gave a bound which is roughly 20 bits greater than that of the histogram.

However such ranks are within reach of key enumeration so **rank estimation is not particularly interesting there**.



Bound Tightness

We **chose Pareto** upper bound functions.
This choise clearly **effects the received accuracy**.

However,

- one could **employ** our **framework**
- **with other classes** of **upper-bound functions**
- and possibly **achieve even better results**.

We leave this direction for future research.

Conclusions

- In this paper we proposed a **new framework for rank estimation**, that is **conceptually simple**, **faster and use less memory** than previous proposals.
- Our main idea is to **bound each subkey distribution** by an **analytical function**, and then estimate the rank by a **closed formula**.
- To **instantiate** the framework we use **Pareto functions** to upper-bound the empirical distributions.

Conclusions

- We **fully characterized** such upper-bounding functions and **developed** an **efficient algorithm** to find them.
- We then used Pareto functions to **develop a new explicit closed formula** upper bound on the rank of a given key.
- Combined with the algorithm to find the upper-bounding Pareto functions, we obtained a rank upper-bound estimation algorithm we call **PRank**.