RSAConference2019
San Francisco | March 4–8 | Moscone Center

SESSION ID: CRYP-R03

Poly-Logarithmic Side Channel Rank Estimation via Exponential Sampling

Liron David
School of Electrical Engineering
Tel-Aviv University

Prof. Avishai Wool
School of Electrical Engineering
Tel-Aviv University

Side Channel Attack

- **Reveals** the **secret key** of an **encryption algorithm**
- by using the **side information** that **leaks** from its **implementation**.
- For example:
  - Power consumption
  - Electro-magnetic radiation
  - Sound
  - Cache
  - …

Side Channel Attack

Secret Key

128 bits

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

Side Channel Attack

It is difficult to reveal all 128 bits in one try…

Secret Key

128 bits

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

Side Channel Attack

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

- Denoted by subkeys

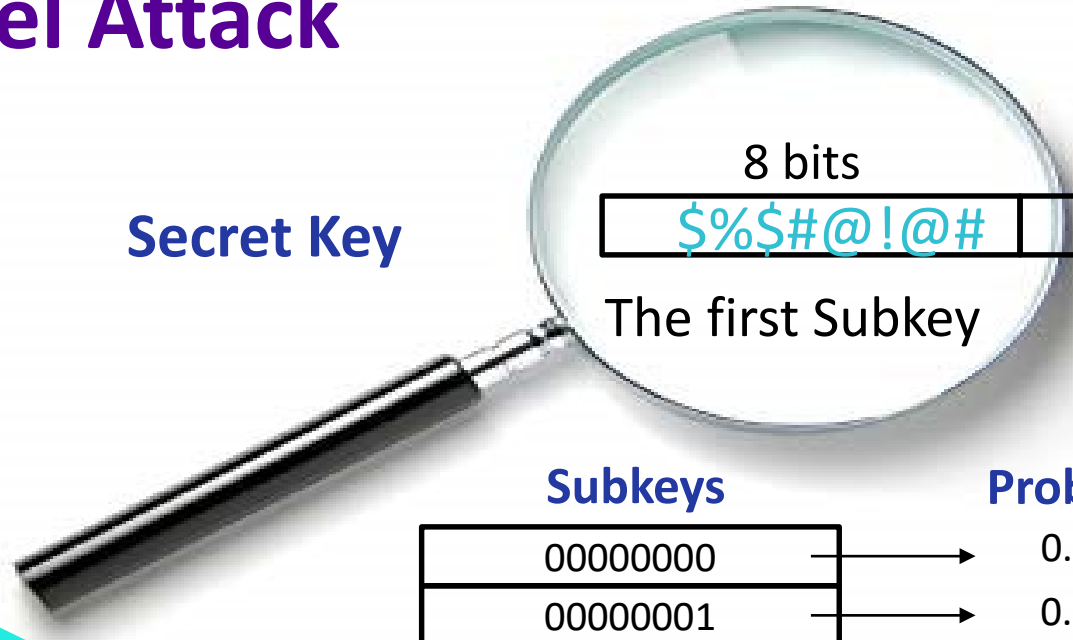
Secret Key

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

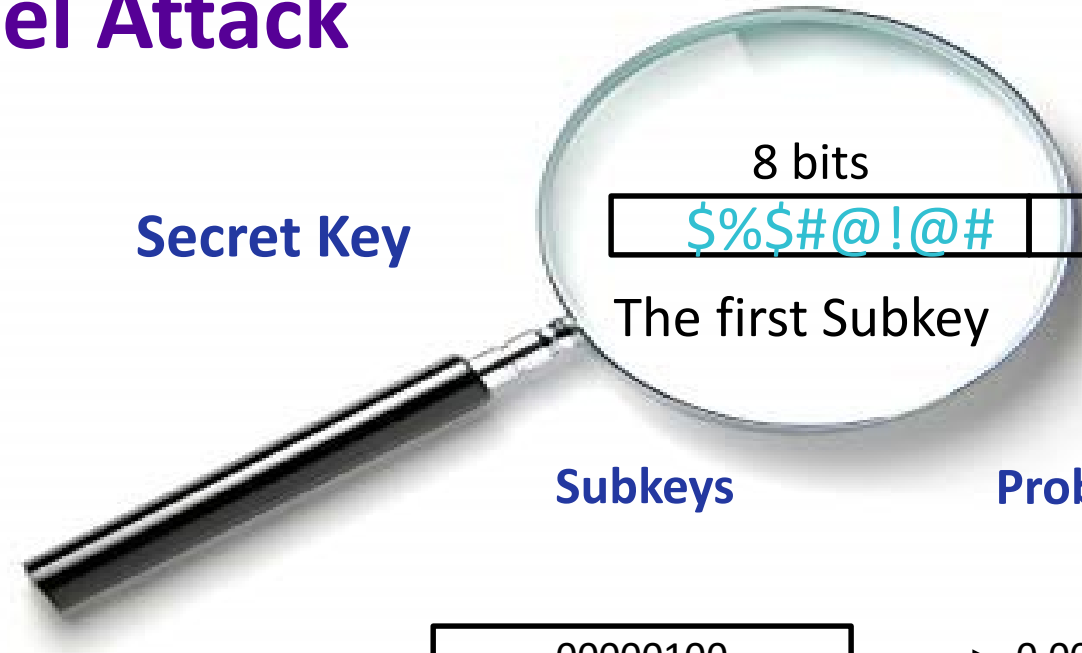


8 bits	8 bits	8 bits	
$%$#@!@#	@$%^#&*!	^*$*$&@%	

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

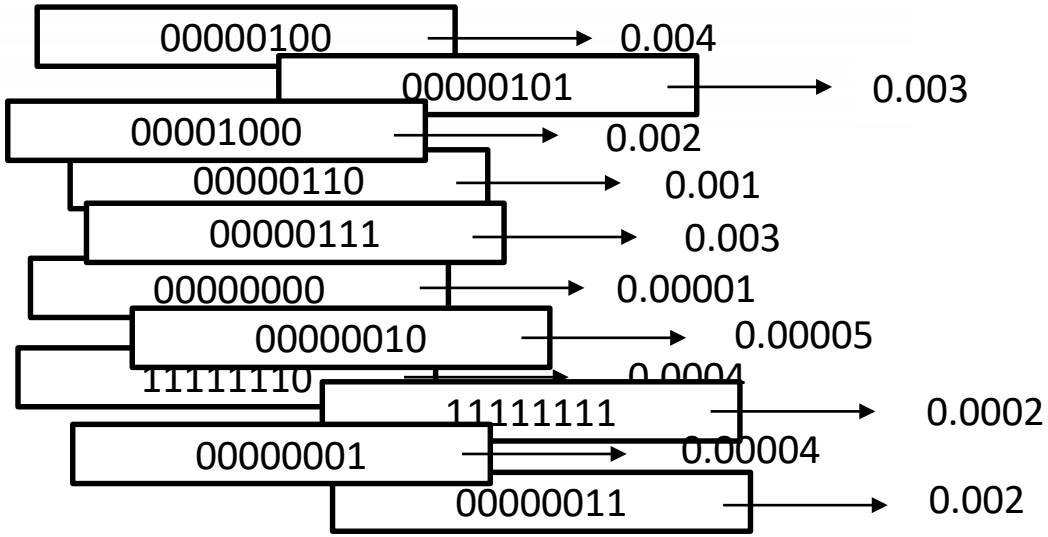


Secret Key

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

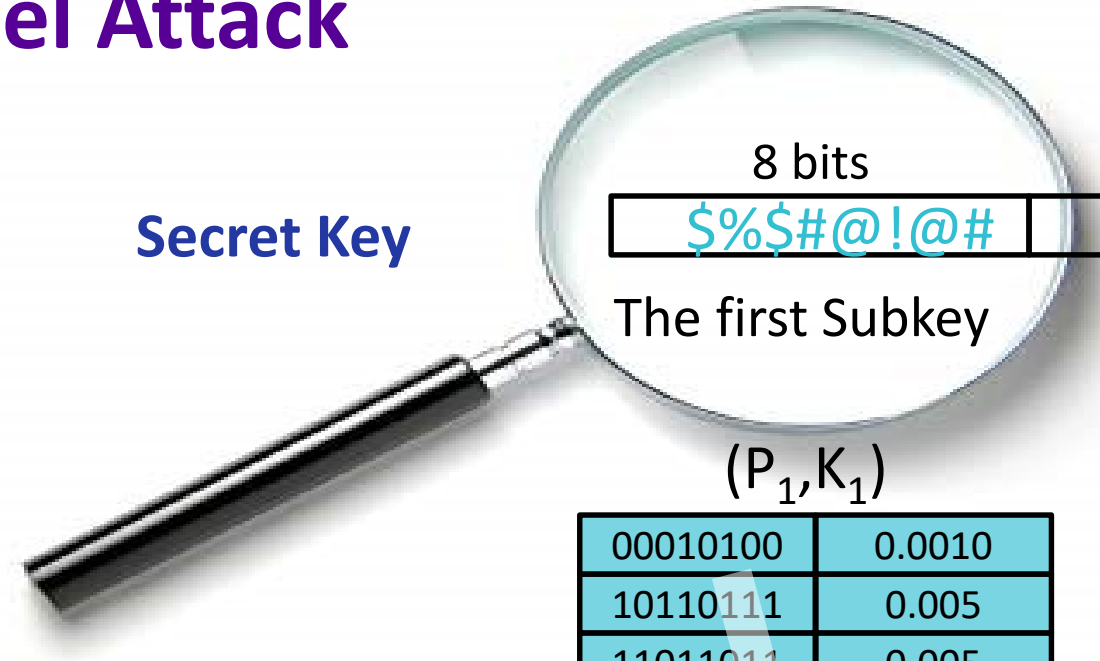
The first Subkey

Subkeys Probabilities



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

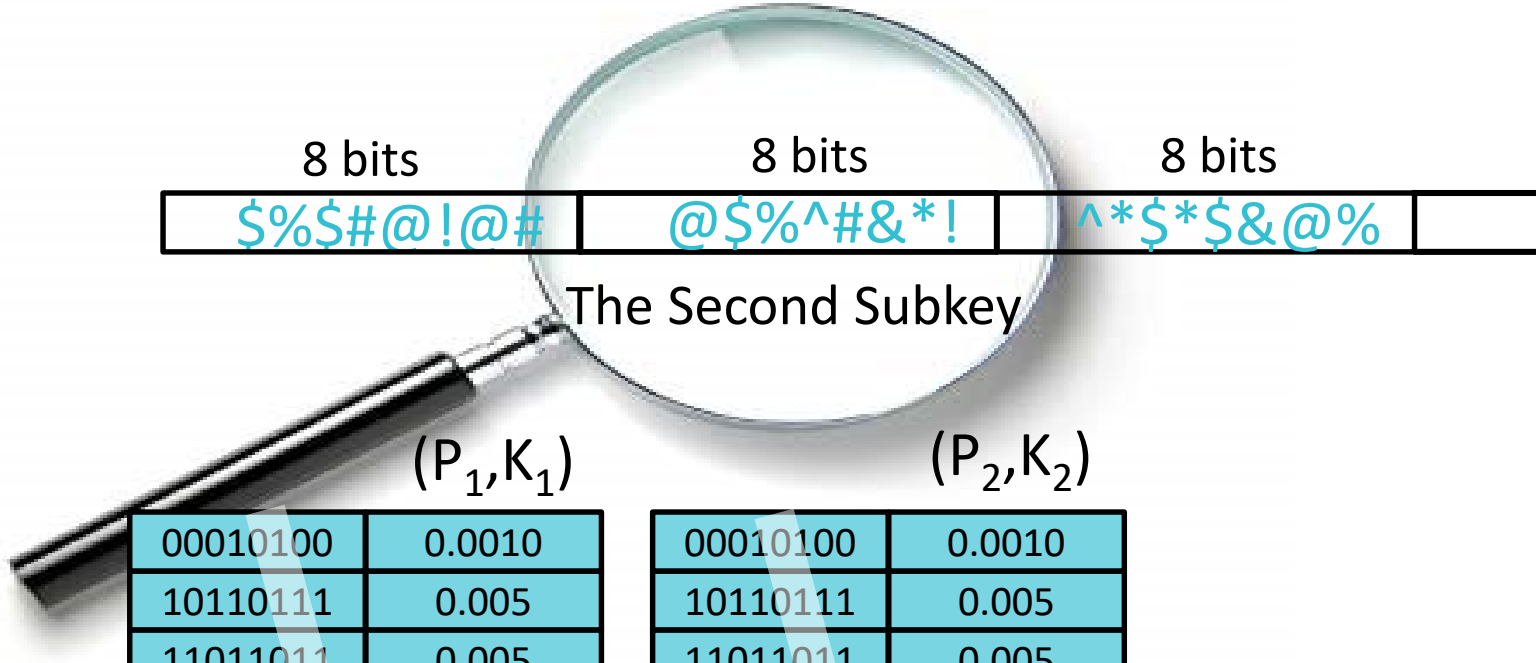
(P_1,K_1)

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



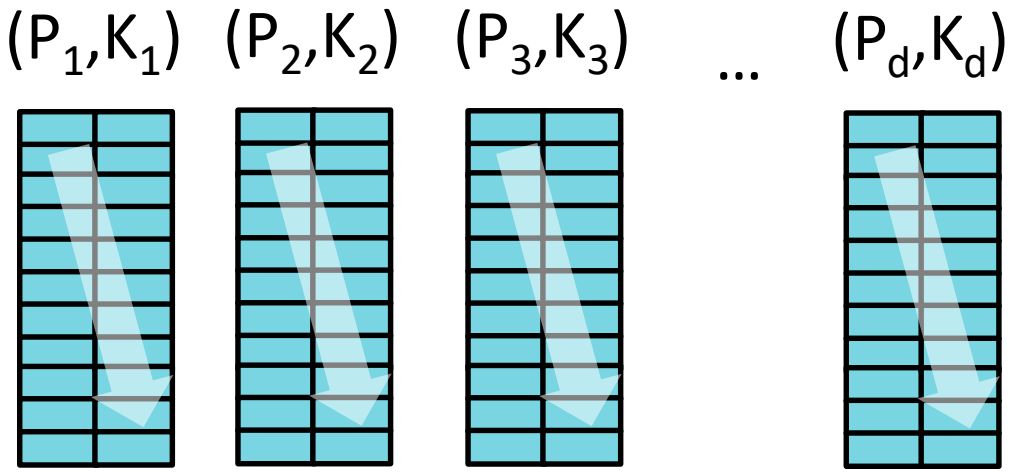
(P_1,K_1)	
00010100	0.0010
10110111	0.005
11011011	0.005
01000011	0.0045
01110000	0.0043
11011010	0.003
10101110	0.003
01001111	0.002
10100110	0.0015
...	...
00000000	0.000001
11111111	0.000001

(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

Sorted subkeys in decreasing order of probabilites

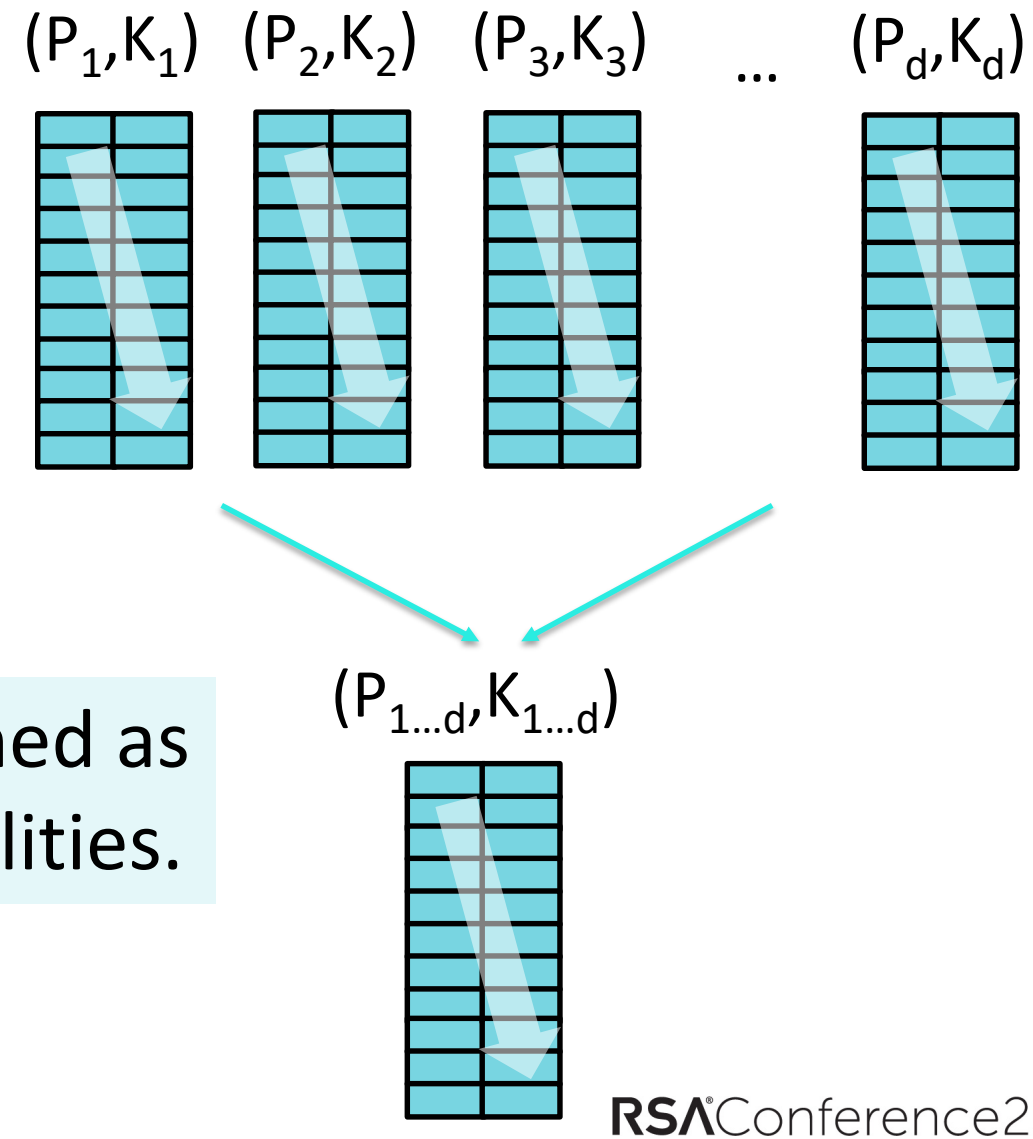
Side Channel Attack

- **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.



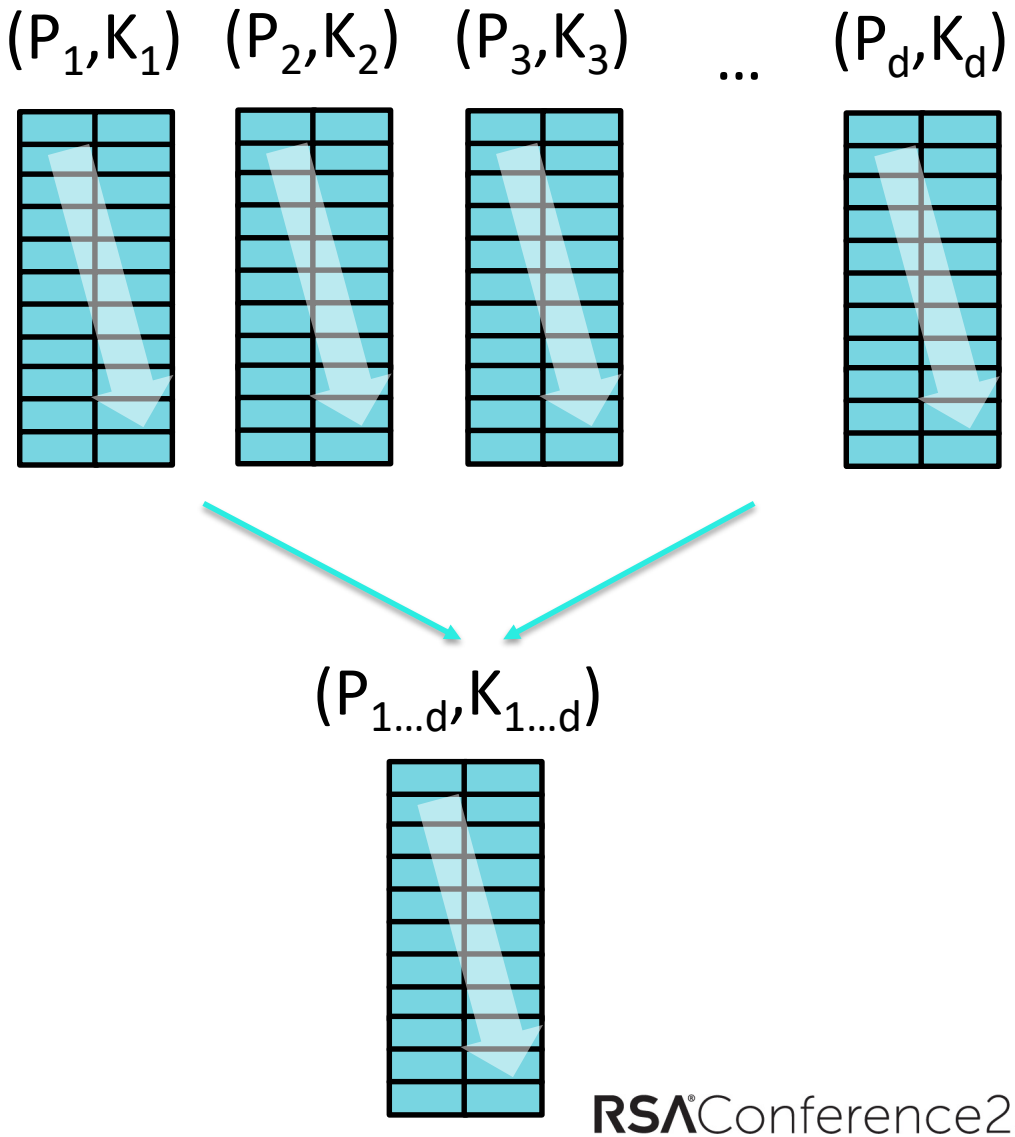
The probability of a full key is defined as the product of its subkey's probabilities.

Side Channel Attack

An important question is:

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

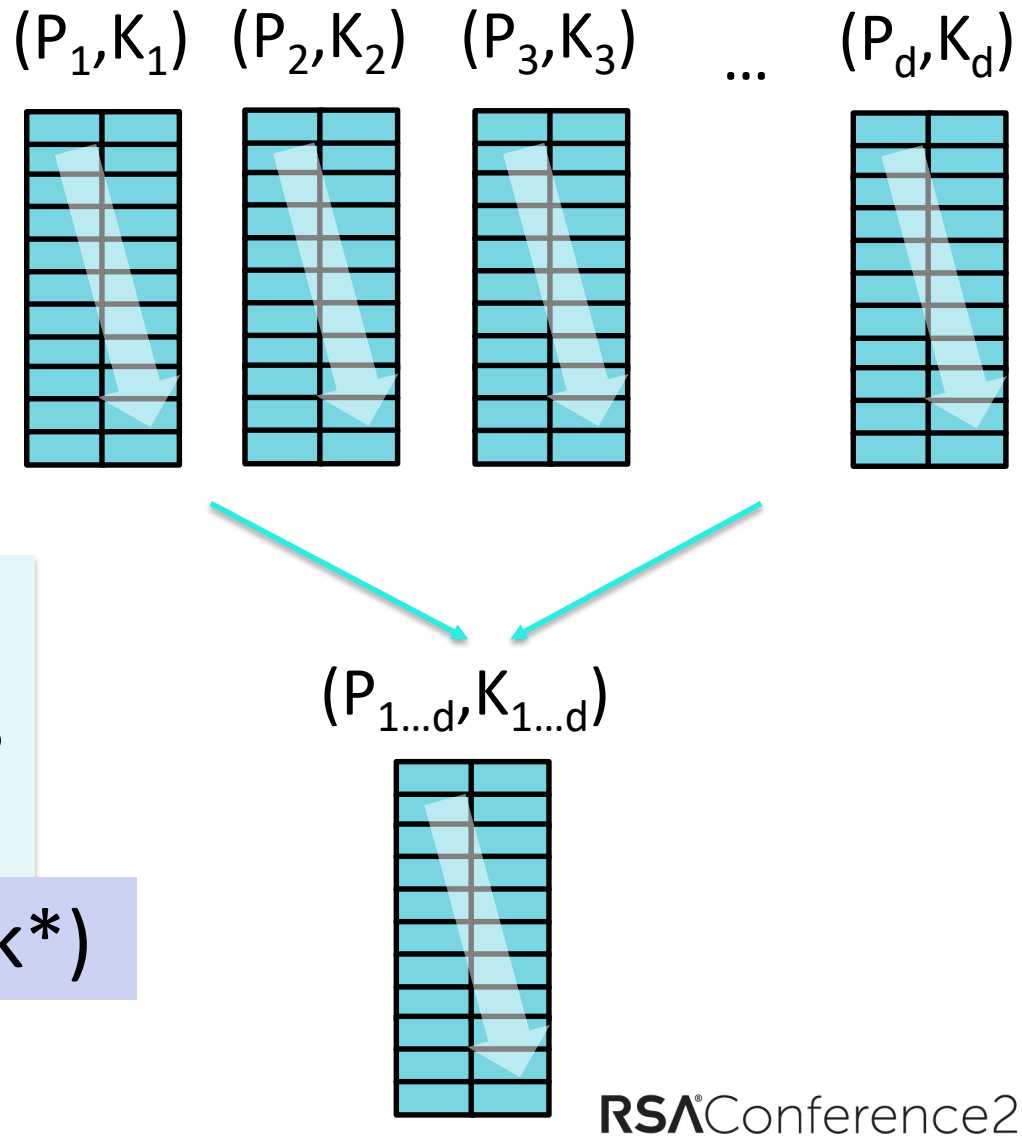
This allows **estimating** the **strength of the chosen secret key** after an attack has been performed.



Side Channel Attack

So assume we **know**

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



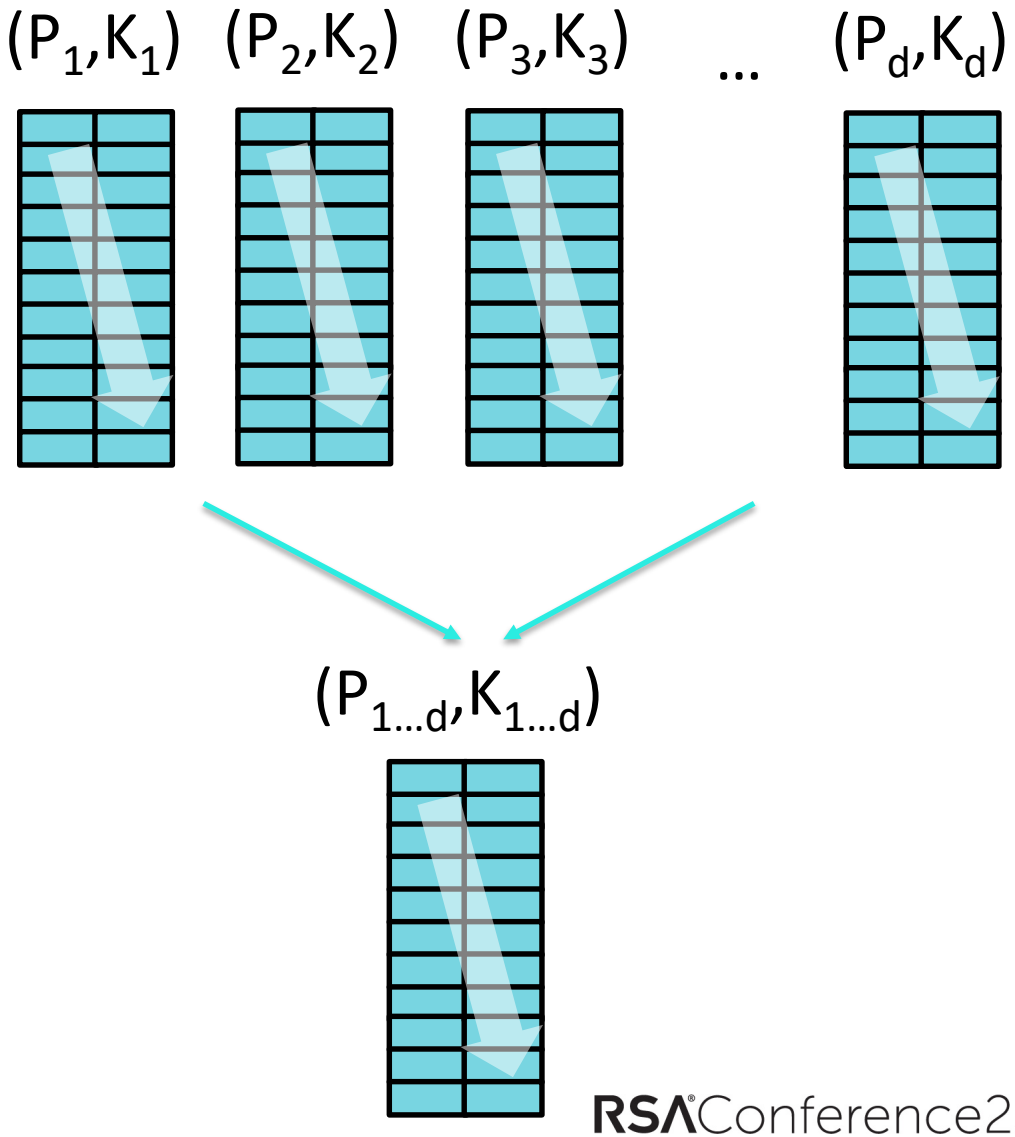
The goal :

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

This is rank(k*)

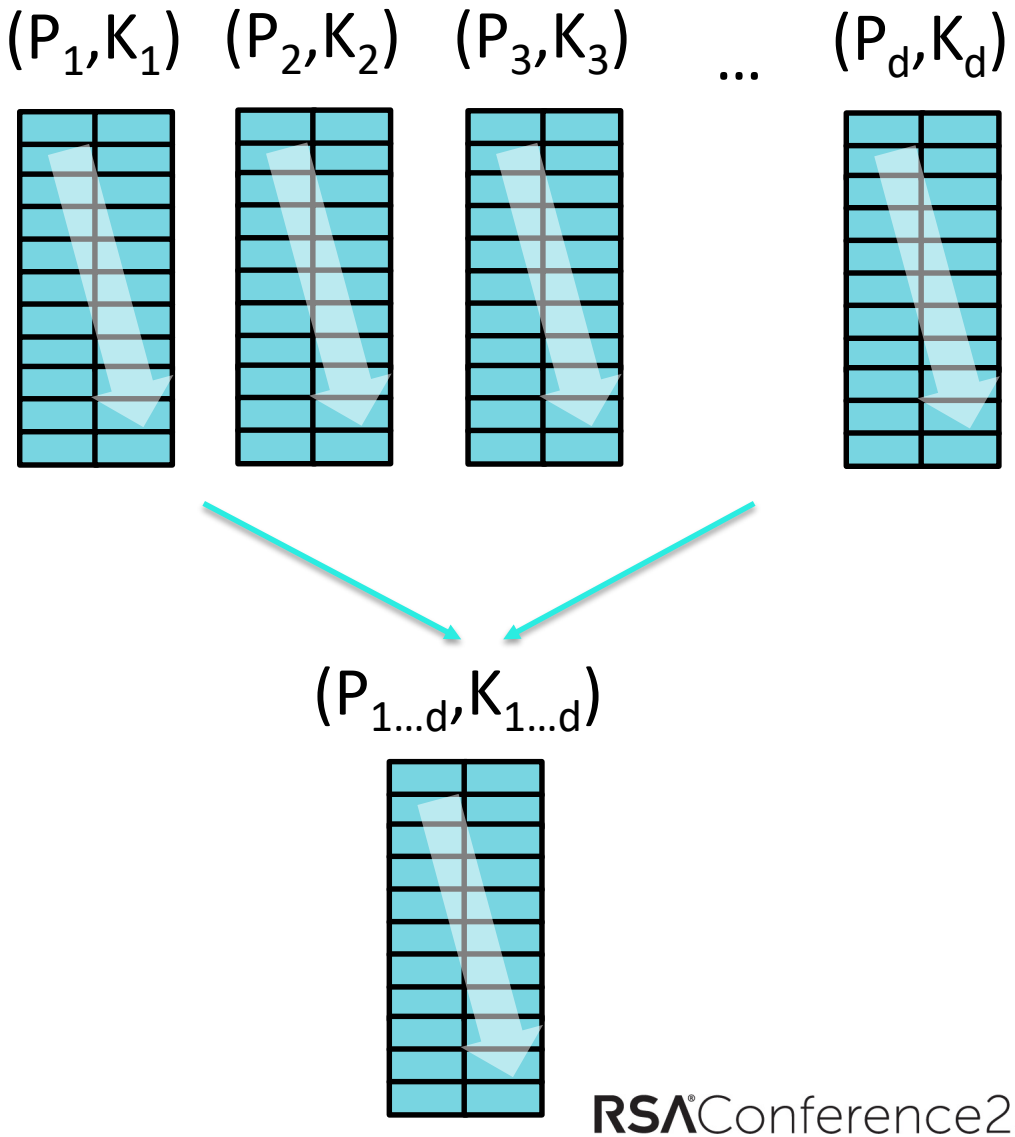
Side Channel Attack

- 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.

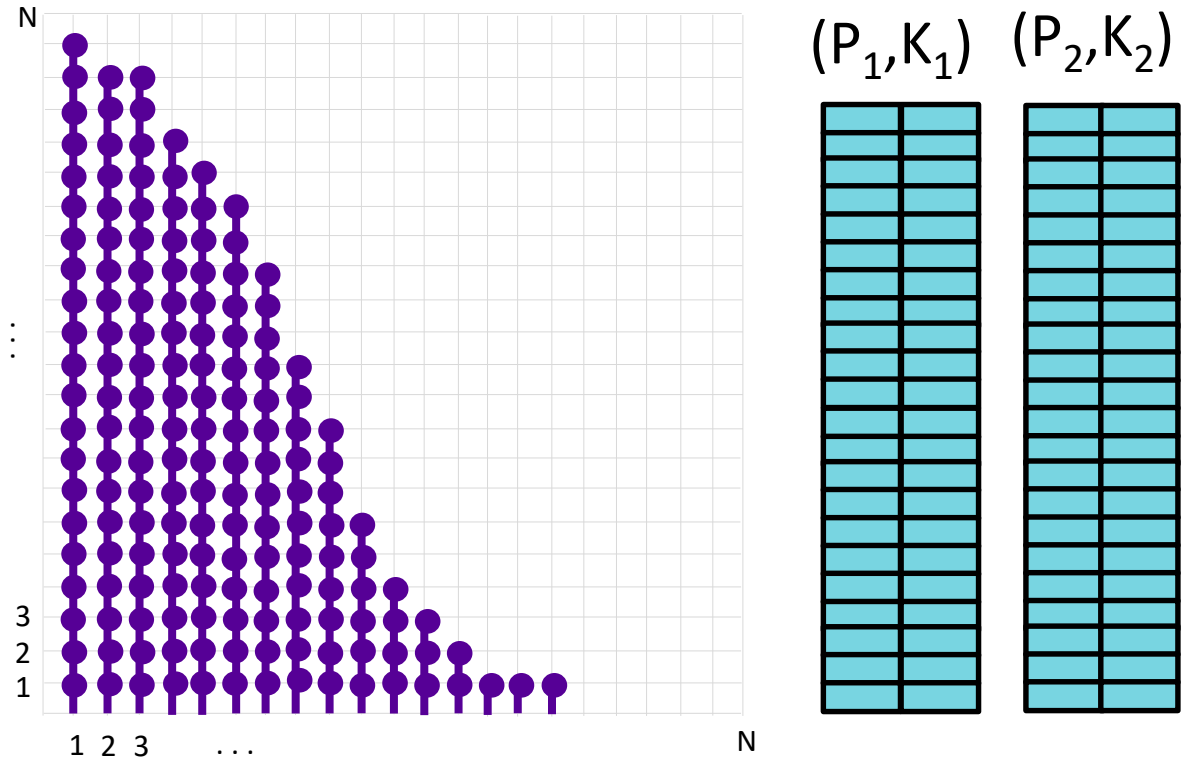


Our ESrank Algorithm: For d=2

Given

- The correct key k* whose probability is p*
- 2 subkeys (P_1,K_1), (P_2,K_2)

rank (k*) is the number of the pairs (i,j) such that
$P_1[i] \cdot P_2[j] \geq p^*$

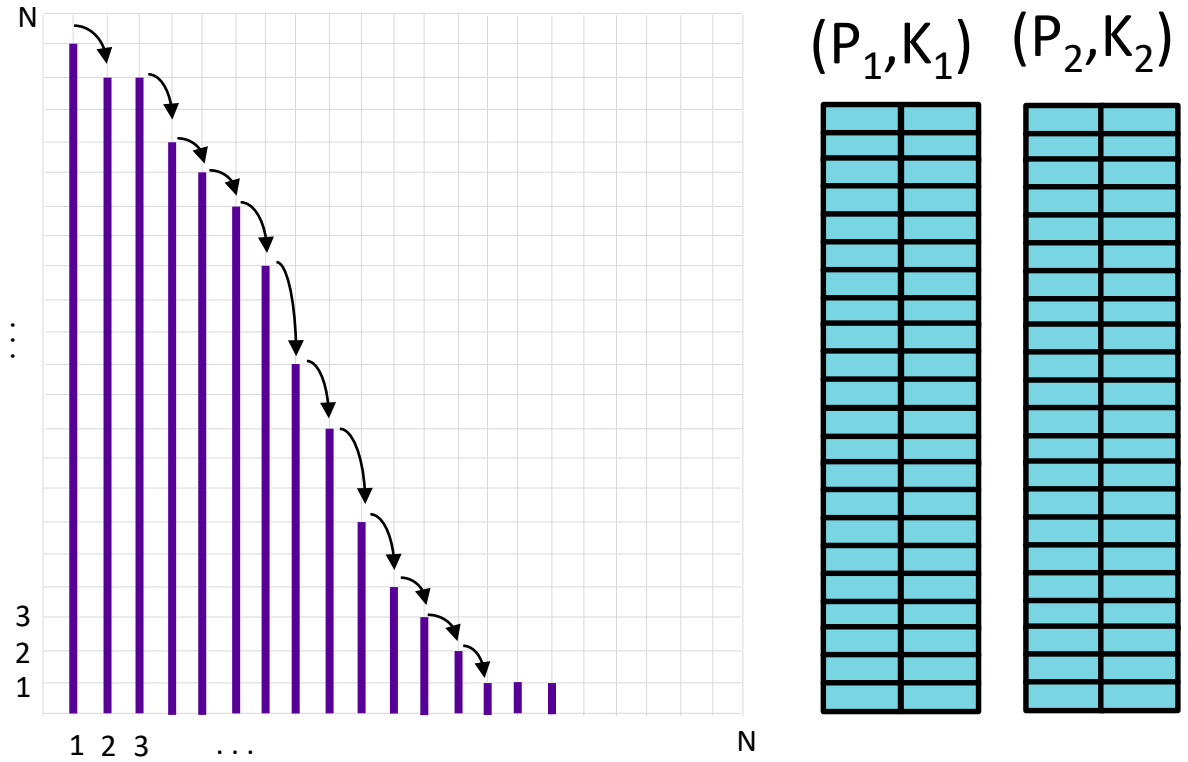


Our ESrank Algorithm: For d=2

We **sum the length of bars** :

Start with i=1, j=N,
decrease j till $P_1[i] \cdot P_2[j] \geq p^*$
then increment i and repeat
till i=N or j=1

O(N) running time.

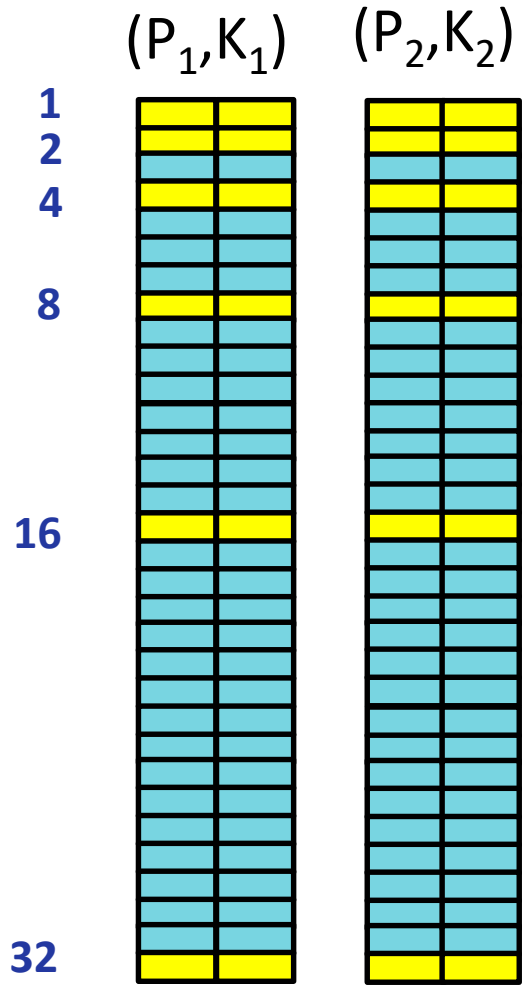


Our ESrank Algorithm: For d=2 with Sampling

To make this algorithm faster,
we use **exponential sampling**.

- **Intuitively, sample indices at powers of $\gamma > 1$.**

For example,
If γ=2 the sampled indices are:
1,2,4,8,16,32,…

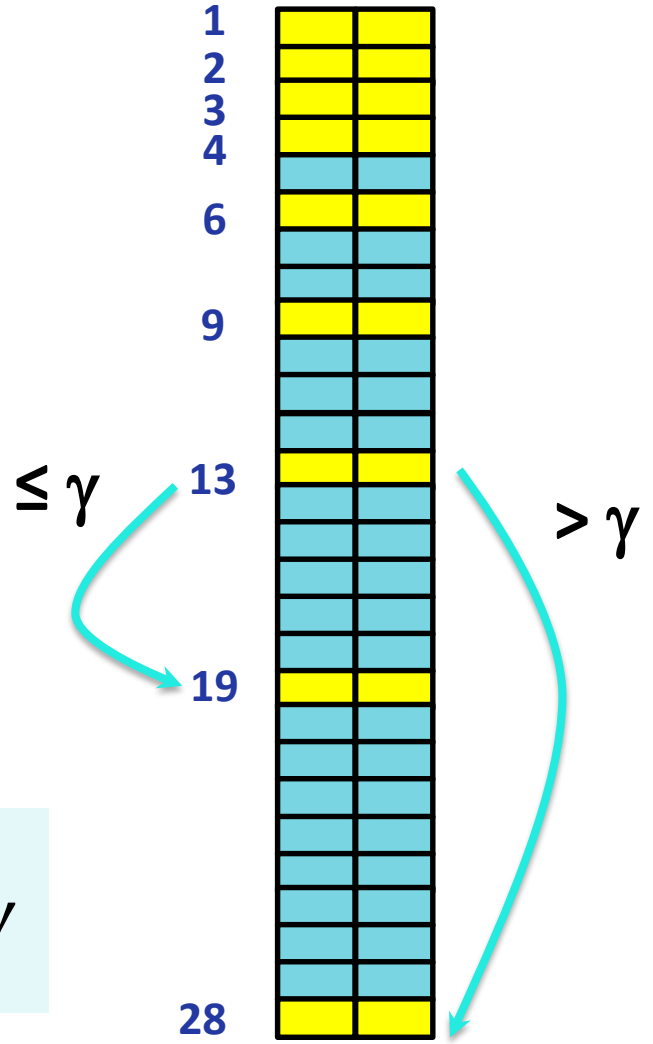


Our ESrank Algorithm: For d=2 with Sampling

Since the power of γ are not necessarily integers

- **maintain** an **invariant**
  on the **ratio between** sampled indices:

$$\frac{\text{i'th sampled index}}{\text{(i−1)'th sampled index}} \leq \gamma \qquad \frac{\text{(i+1)'th sampled index}}{\text{(i−1)'th sampled index}} > \gamma$$

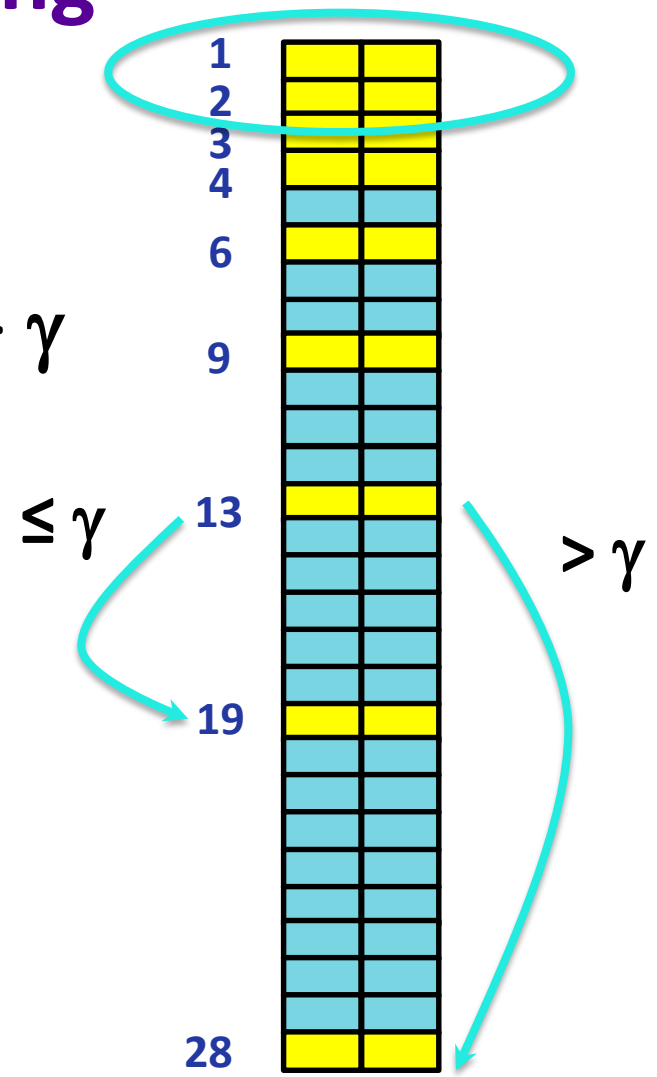


Our ESrank Algorithm: For d=2 with Sampling

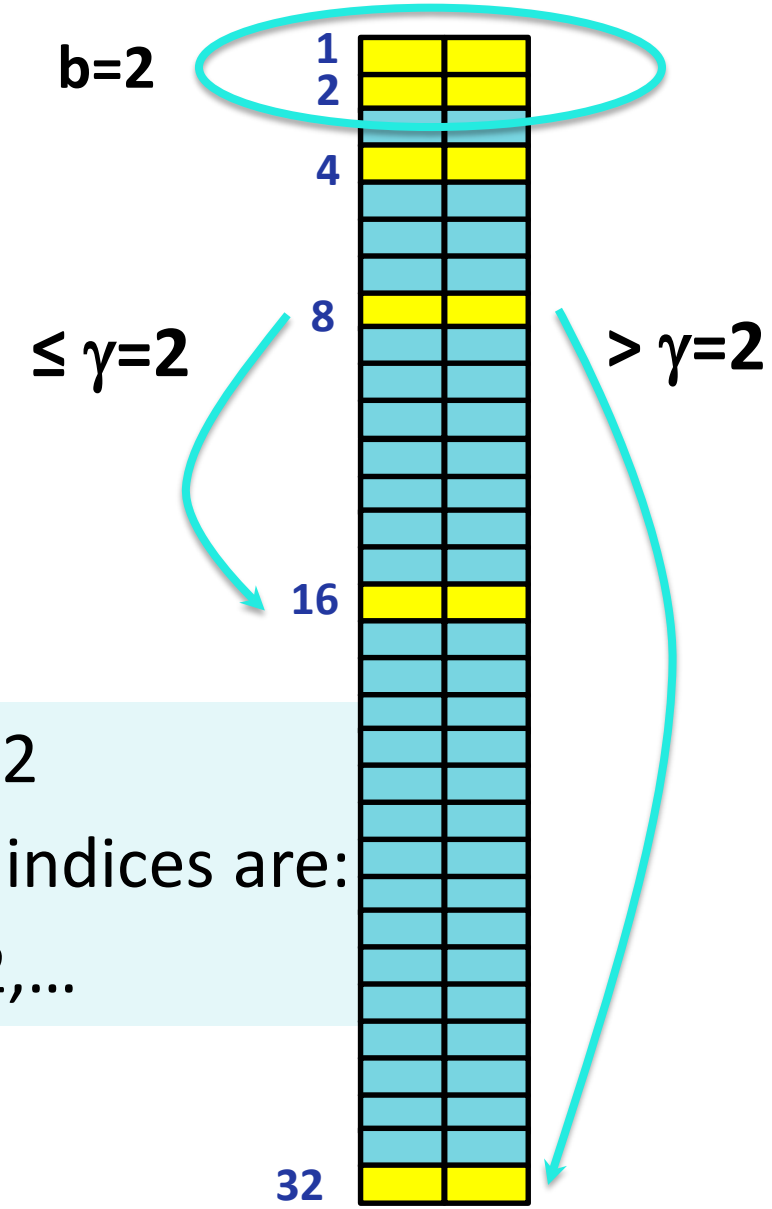
The first indices might not maintain the invariant

- since the ratio between consecutive indices is $> \gamma$

- Let **b** be the **smallest index** that **maintains the invariant**.
- Take the **first b indices without sampling**
- **Sample** the rest **with ratio** γ



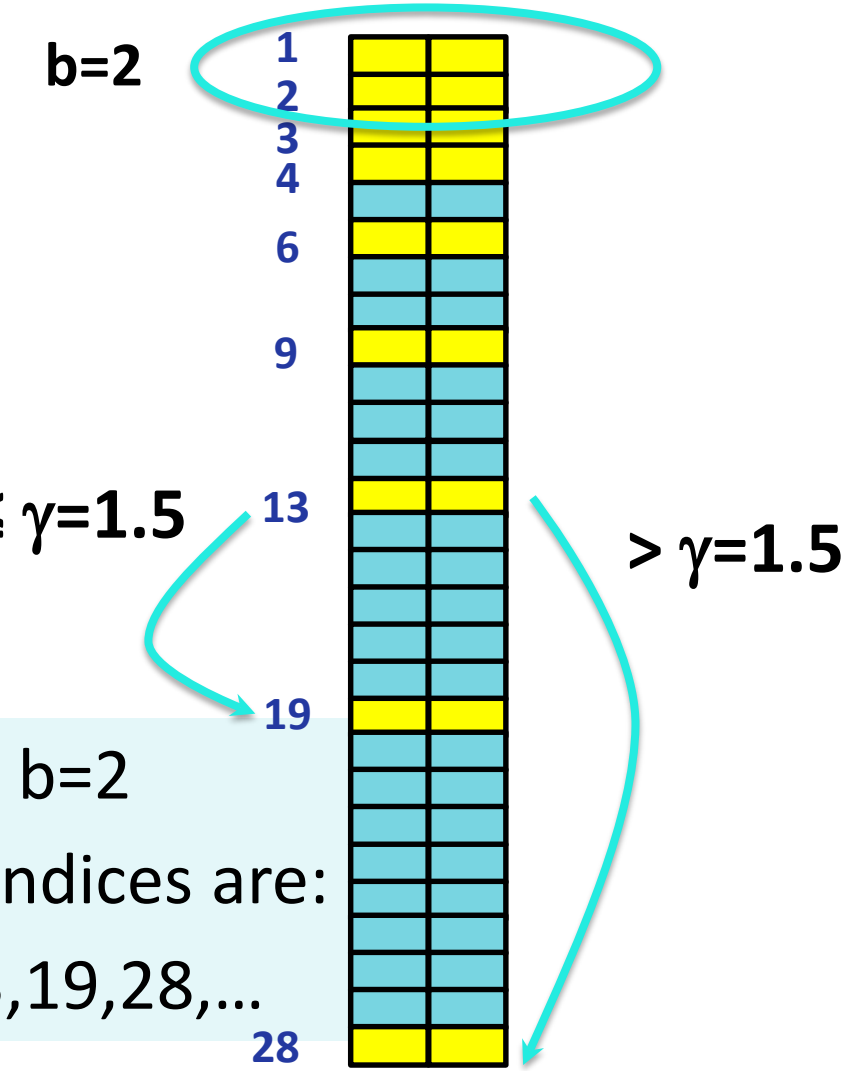
Our ESrank Algorithm: For d=2 with Sampling



If γ=2 and b=2

the sampled indices are:

1,2,4,8,16,32,…



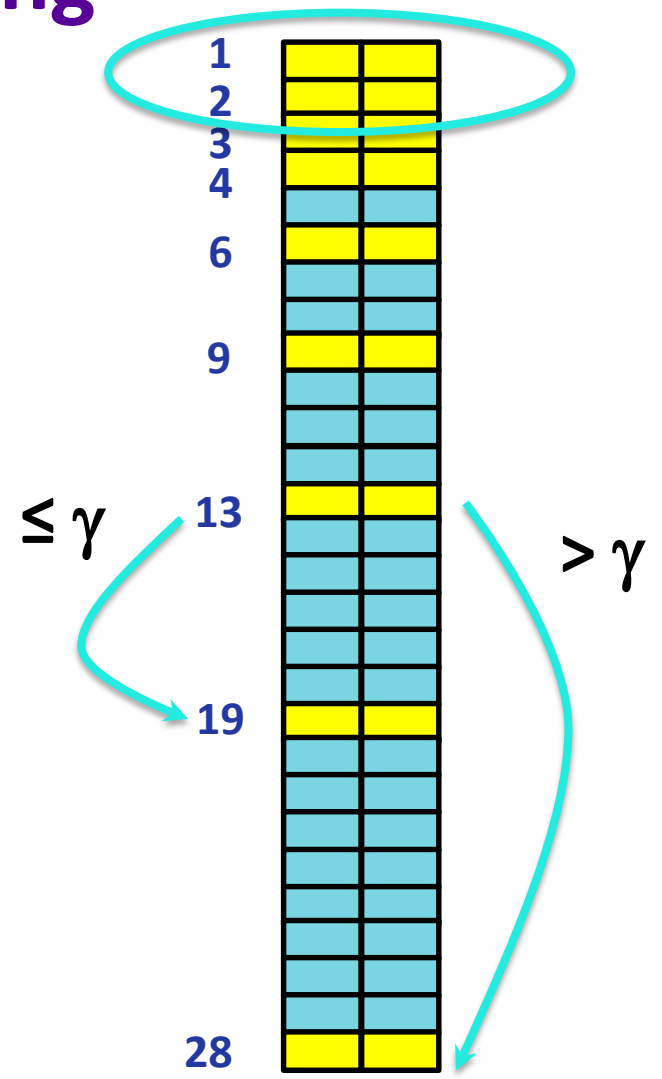
For γ=1.5 and b=2

the sampled indices are:

1,2,3,4,6,9,13,19,28,…

Our ESrank Algorithm: For d=2 with Sampling

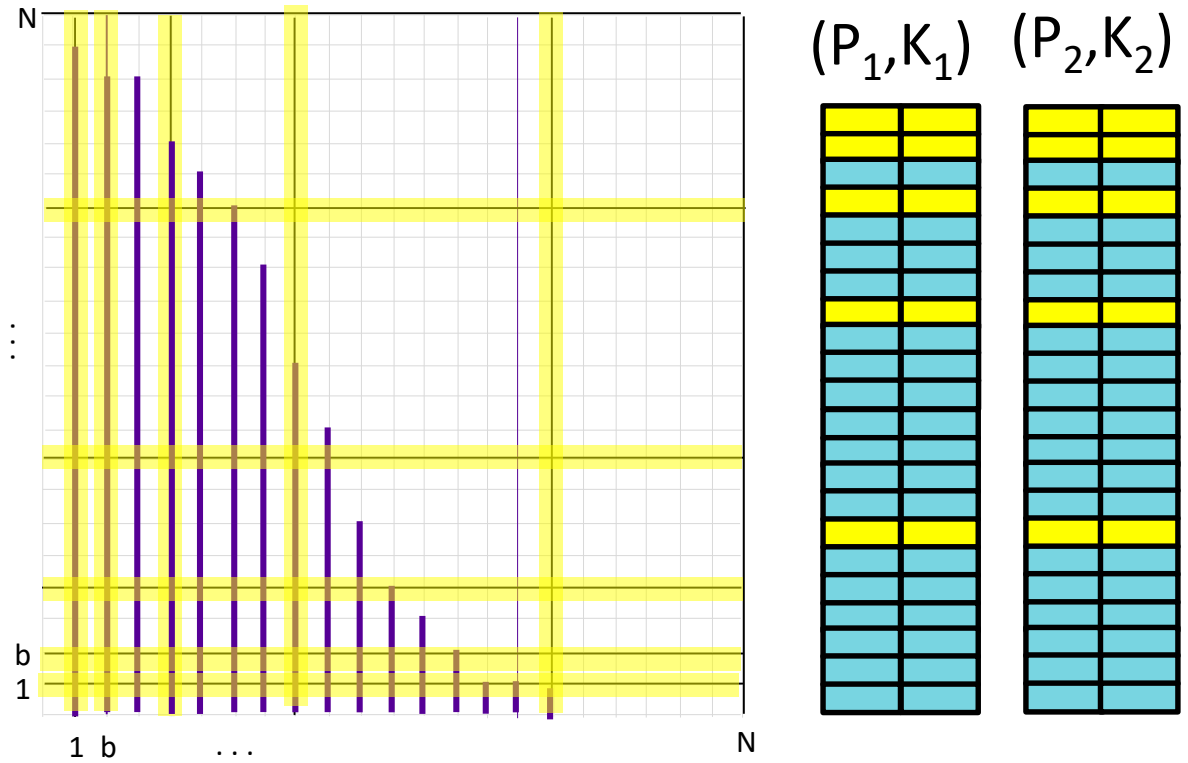
The **sampled** distribution is of **size** $O(\log_\gamma N)$



Our ESrank Algorithm: For d=2 with Sampling

Now, given
two sampled distibutions
and the **correct key k***

The goal:
To calculate **rank(k*)**

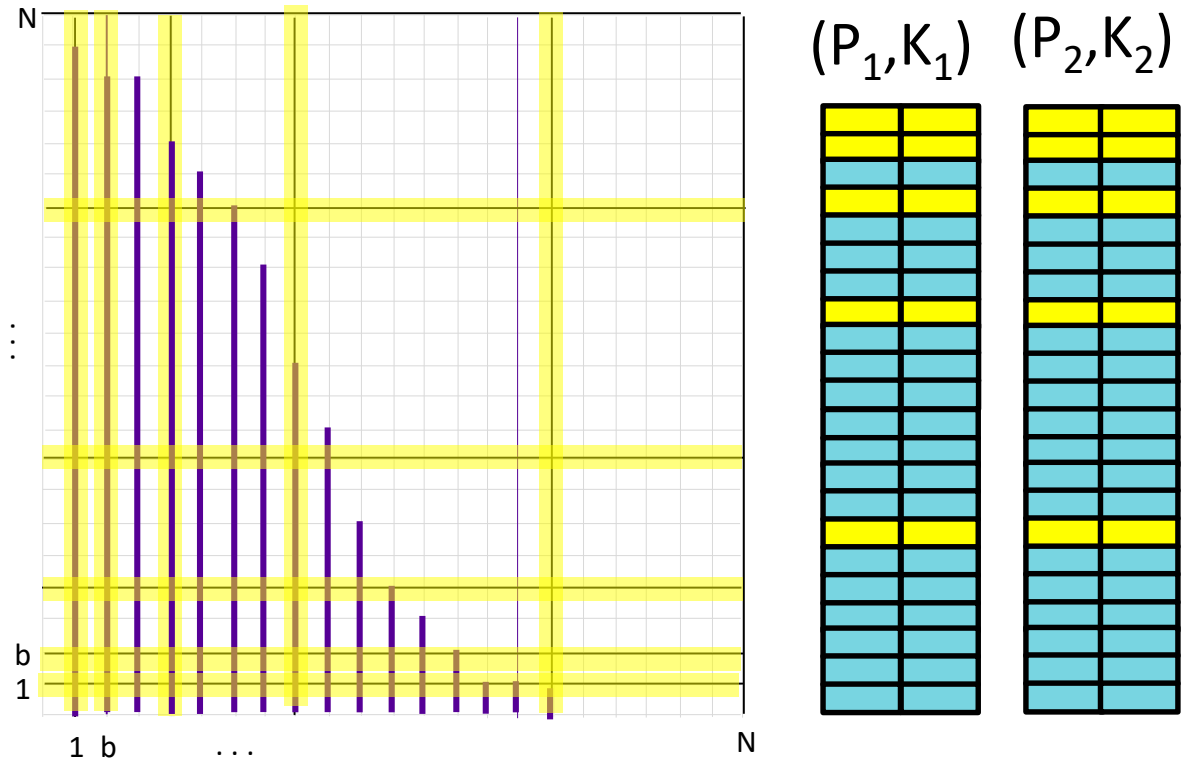


Our ESrank Algorithm: For d=2 with Sampling

We only have an **access** to the **sampled indices**

Therefore, we calculate:

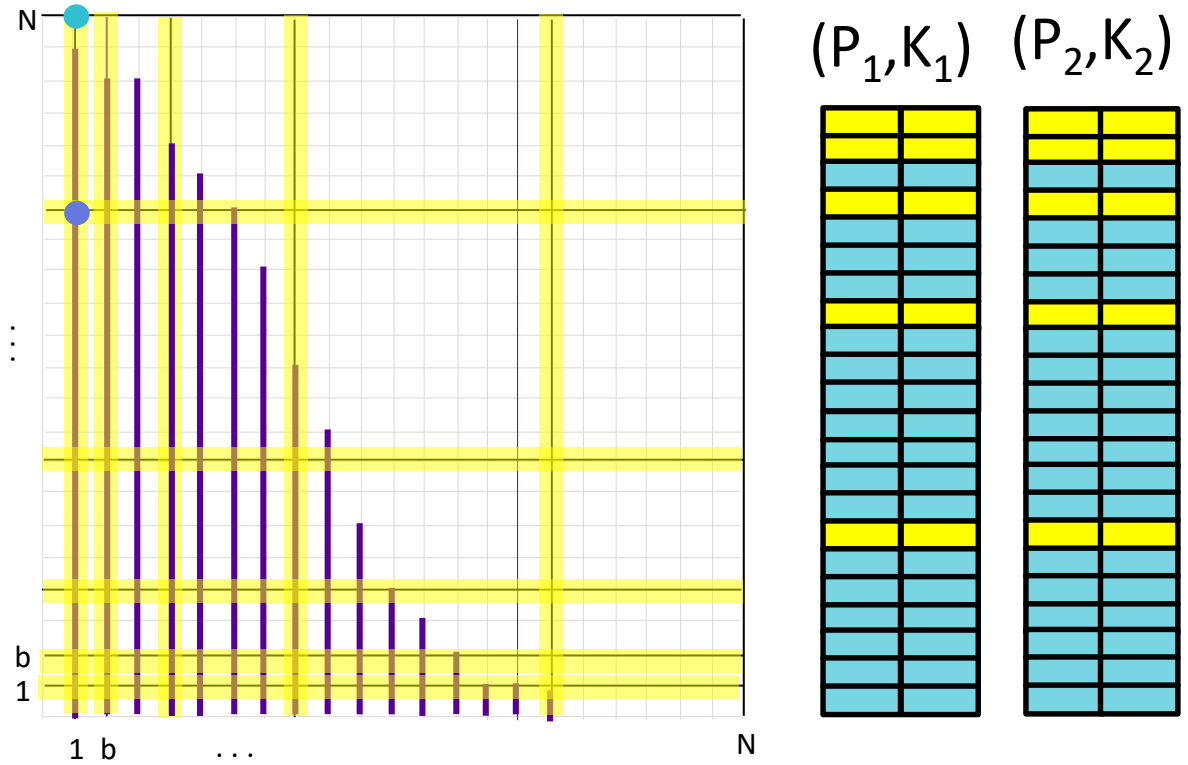
- **upper** bound for rank(k*)
- **lower** bound for rank(k*)



Our ESrank Algorithm: For d=2 with Sampling

We will calculate:

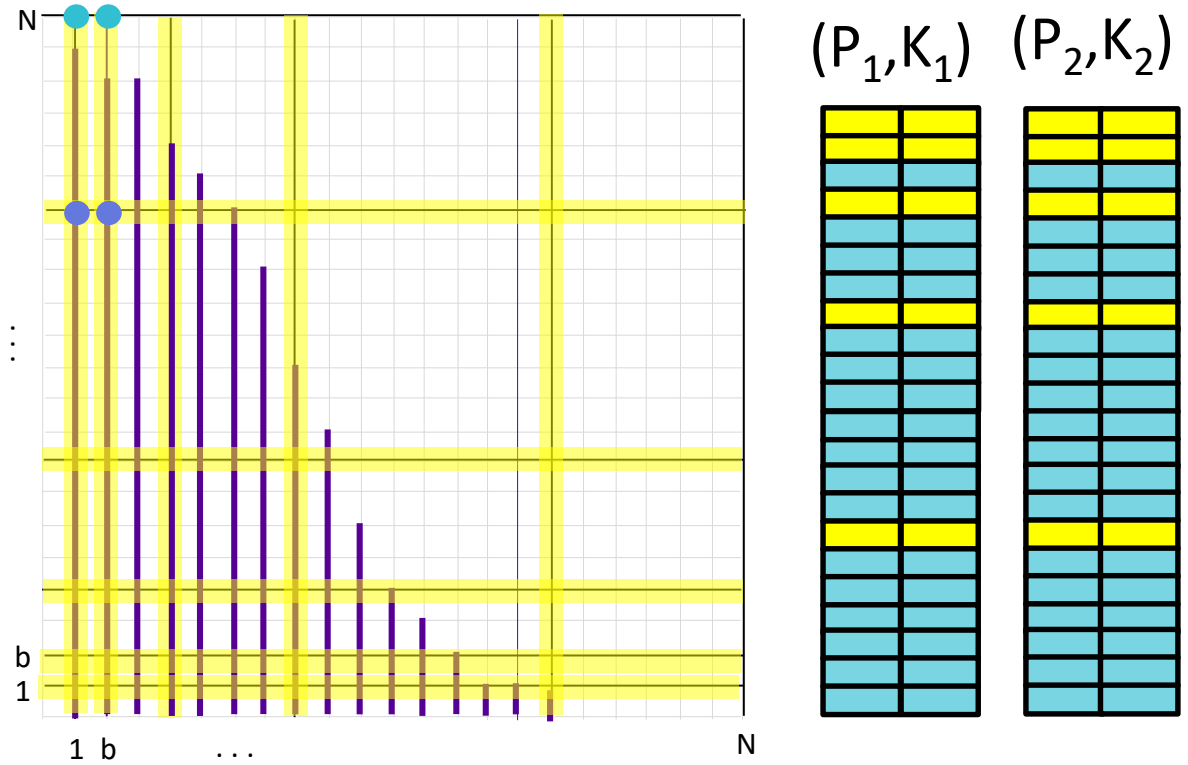
- upper bound for rank(p*)
- lower bound for rank(p*)



Our ESrank Algorithm: For d=2 with Sampling

We will calculate:

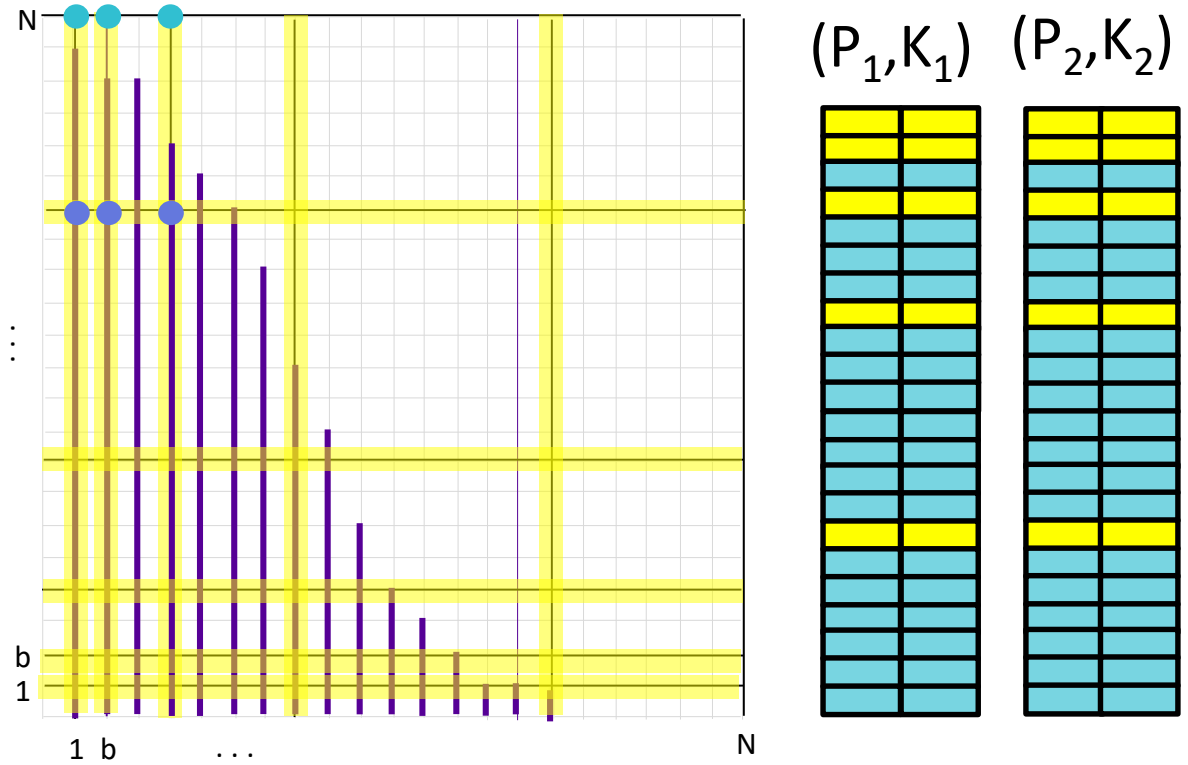
- upper bound for rank(p*)
- lower bound for rank(p*)



Our ESrank Algorithm: For d=2 with Sampling

We will calculate:

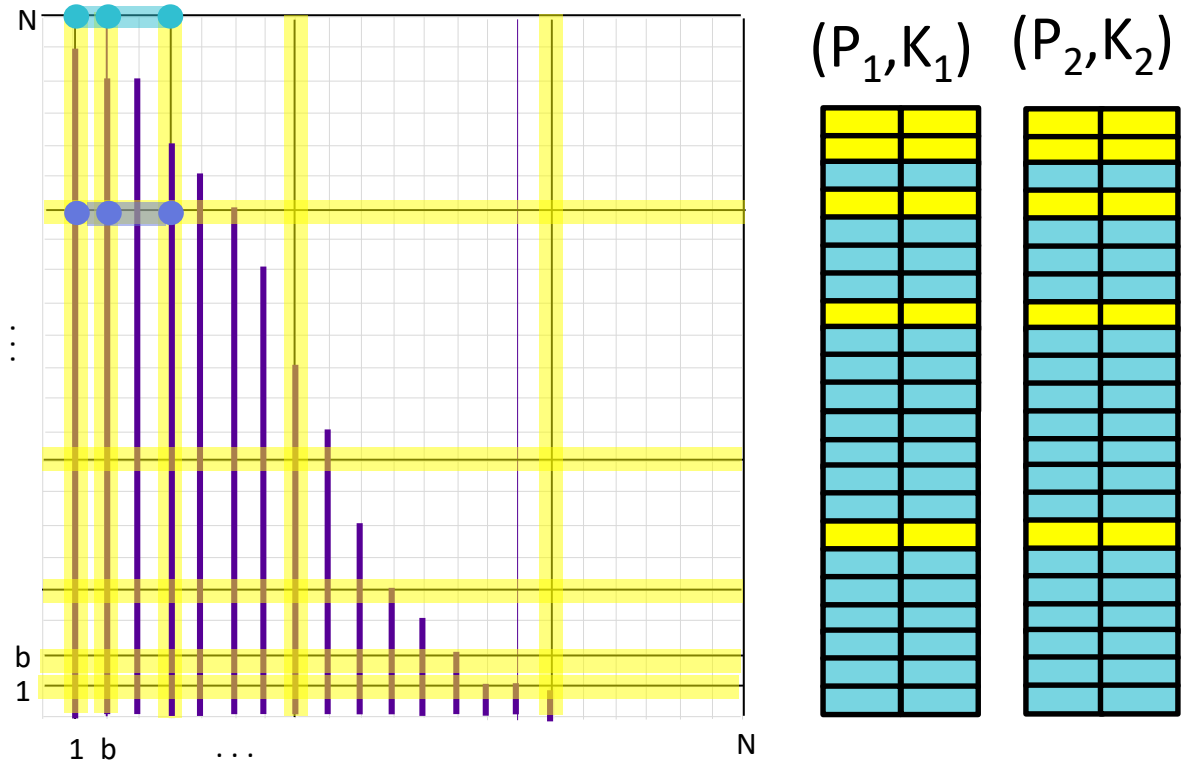
- upper bound for rank(p*)
- lower bound for rank(p*)



Our ESrank Algorithm: For d=2 with Sampling

We will calculate:

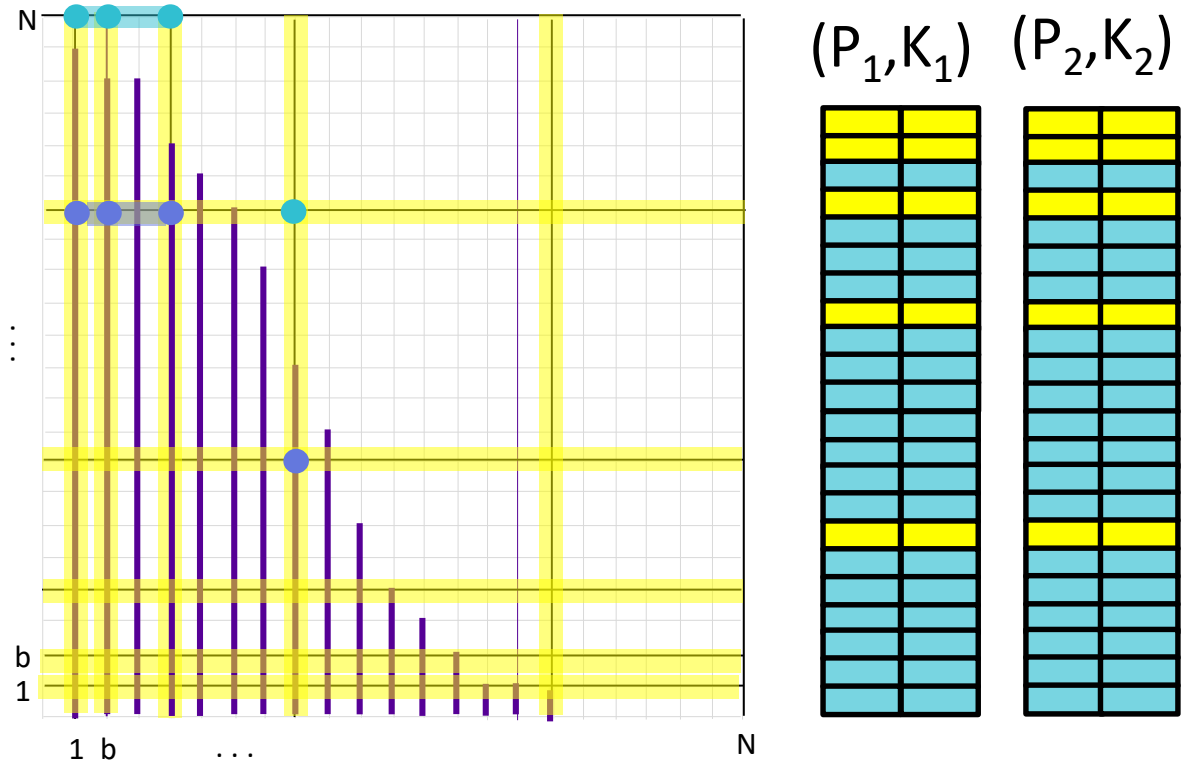
- upper bound for rank(p*)
- lower bound for rank(p*)



Our ESrank Algorithm: For d=2 with Sampling

We will calculate:

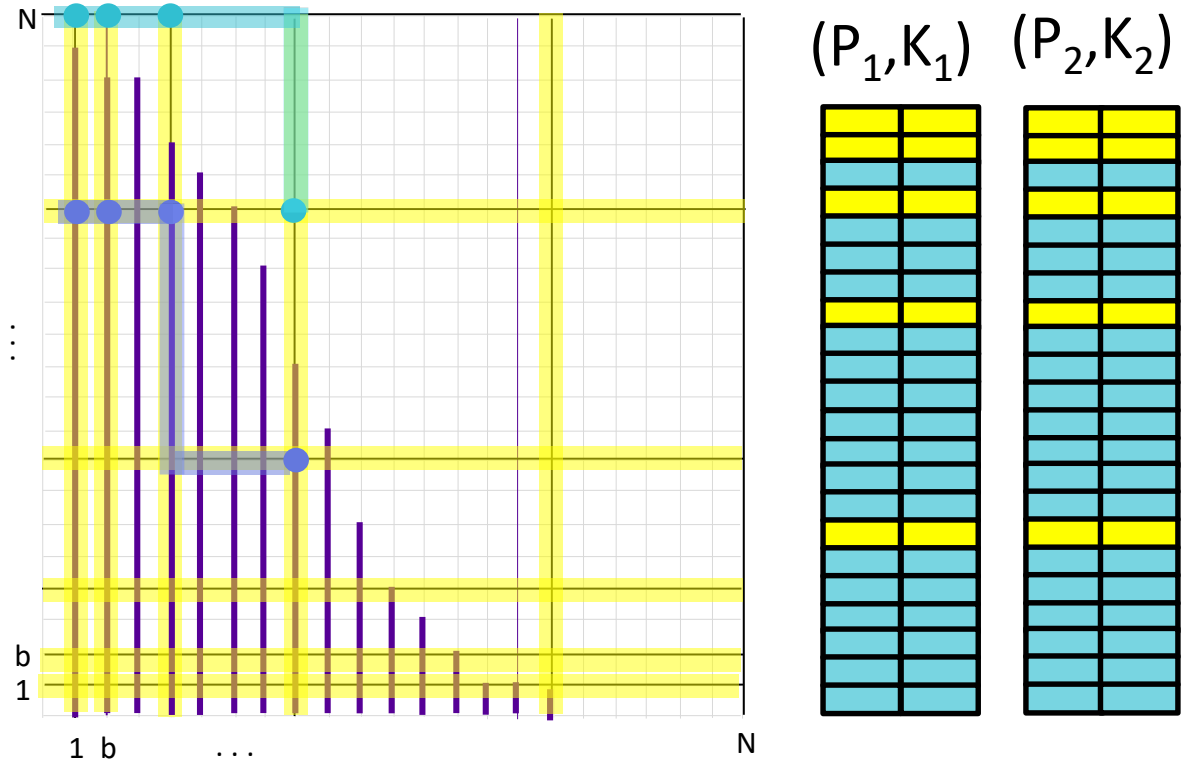
- upper bound for rank(p*)
- lower bound for rank(p*)



Our ESrank Algorithm: For d=2 with Sampling

We will calculate:

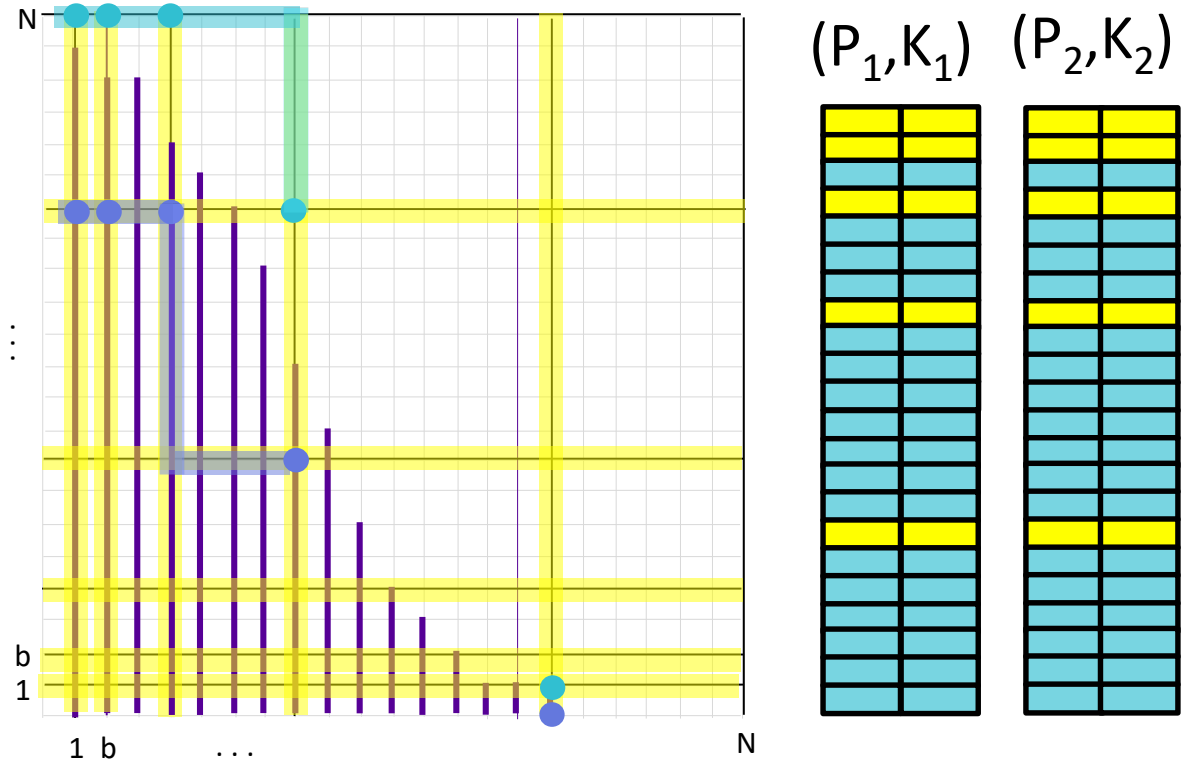
- upper bound for rank(p*)
- lower bound for rank(p*)



Our ESrank Algorithm: For d=2 with Sampling

We will calculate:

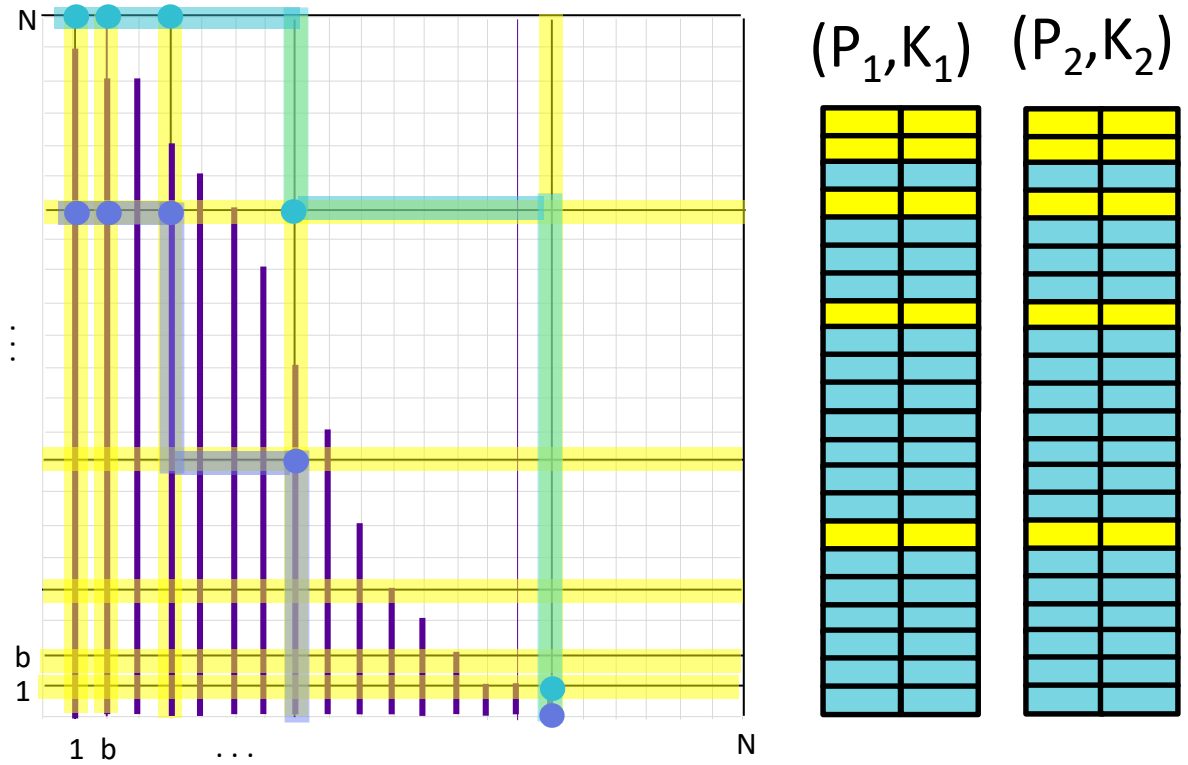
- upper bound for rank(p*)
- lower bound for rank(p*)



Our ESrank Algorithm: For d=2 with Sampling

We will calculate:

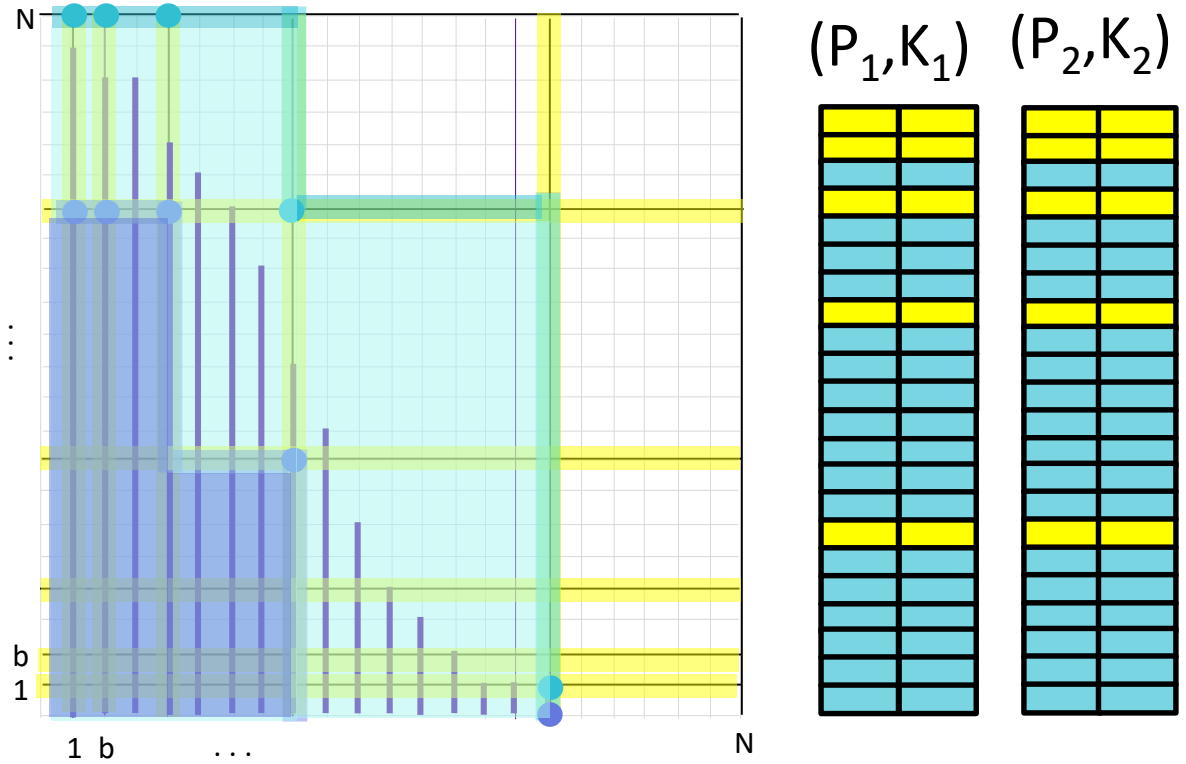
- upper bound for rank(p*)
- lower bound for rank(p*)



Our ESrank Algorithm: For d=2 with Sampling

We will calculate:

- upper bound for rank(p*)
- lower bound for rank(p*)

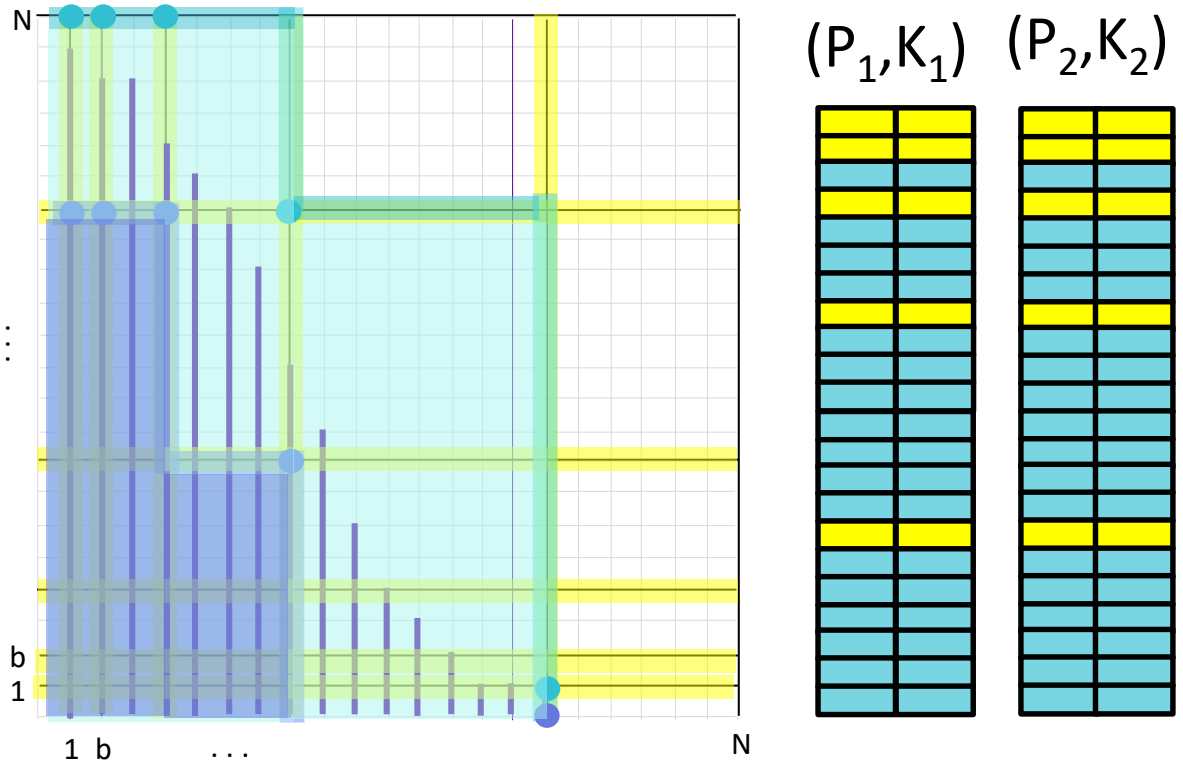


Our ESrank Algorithm: For d=2 with Sampling

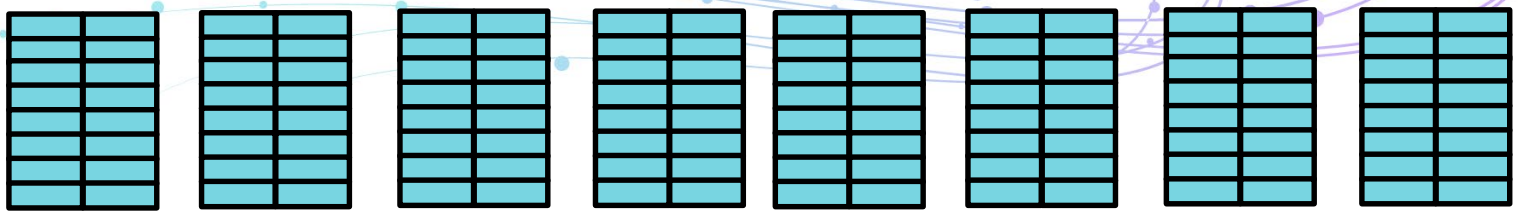
The **running time is linear** in the **size** of the **sampled** distribution,

Therefore $O\left(\log_{\gamma} N\right)$

The **ratio** between the **upper** and the **lower** bounds for a given correct key **is bounded by** $< \gamma^2$

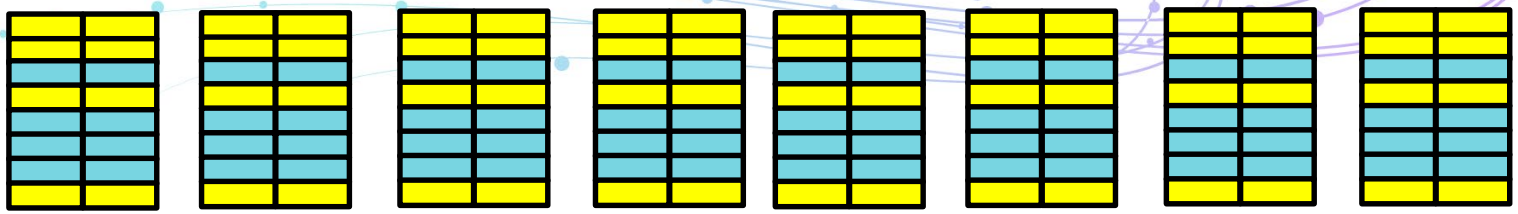


ERank For d>2



d subkey distributions

ERank For d>2

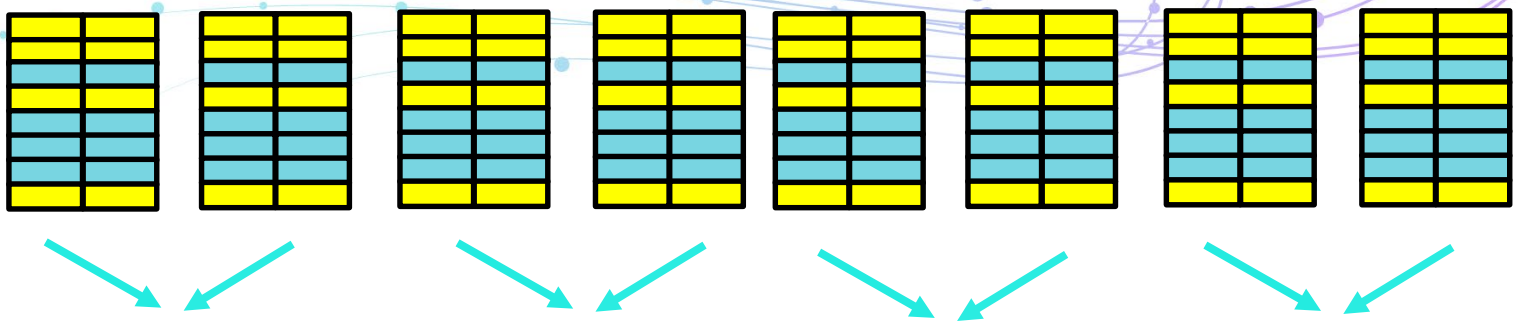


Sample each distribution s.t.

the **sampled indices** maintain
the **invariant with γ and b**

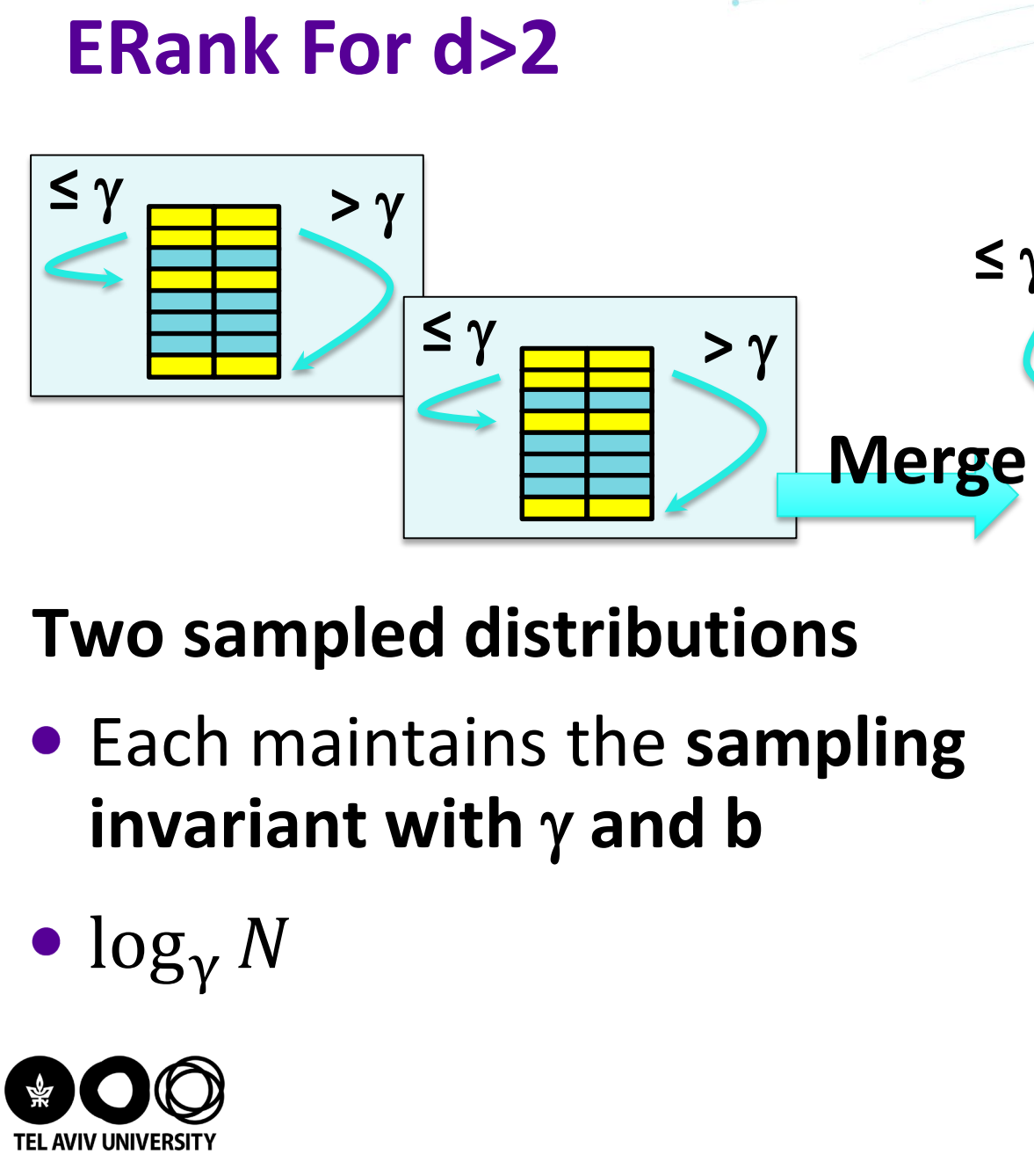
Therefore, each of size $\log_{\gamma} N$

ERank For d>2



Merge each **two sampled** distributions

ERank For d>2



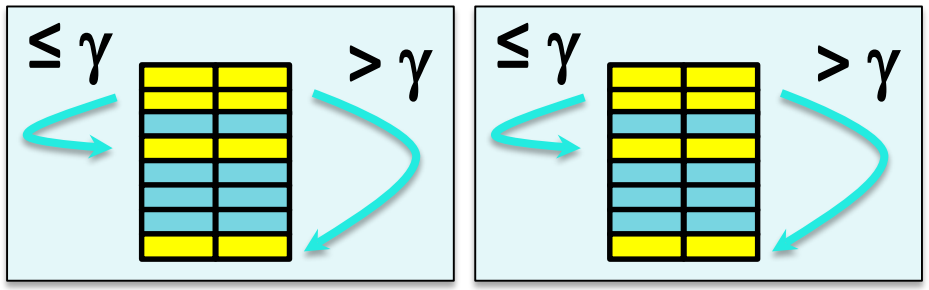
Two sampled distributions

- Each maintains the **sampling invariant with γ and b**
- $\log_\gamma N$

One **sampled distribution**

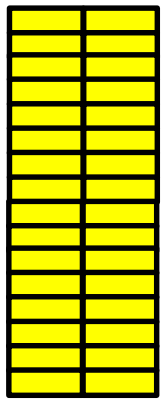
- maintains the **sampling invariant with γ and b**
- $\log_\gamma N^2 = 2\log_\gamma N$

ERank For d>2



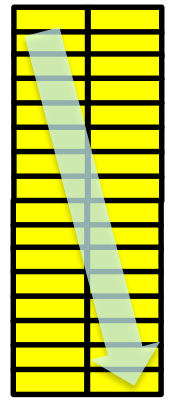
$\log_\gamma N$ $\log_\gamma N$

Calculate the probabilities of all the (x,y) pairs



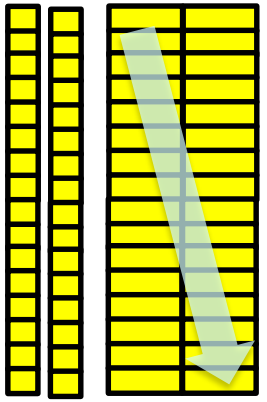
$(\log_\gamma N)^2$

Sort the pairs in decreasing order of probabilities



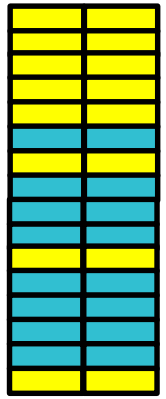
$(\log_\gamma N)^2$

Calculate the **upper** and **lower** bounds of the pairs in **accumulative way,** in **one linear pass over the pairs**



$(\log_\gamma N)^2$

Sample with γ and b



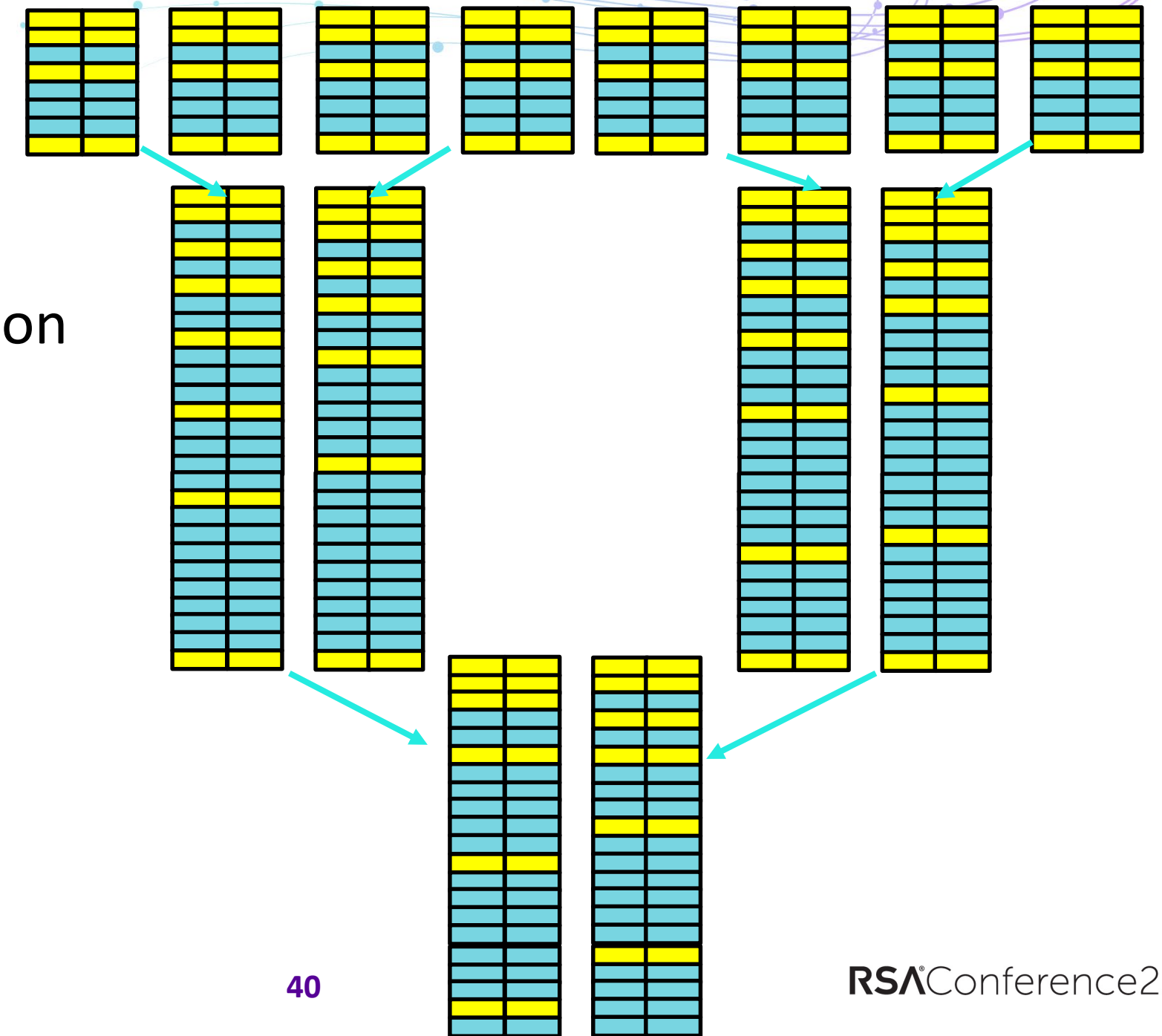
$\log_\gamma N^2 = 2\log_\gamma N$

ERank For d>2

Merge each

two sampled distribution **with γ and b**

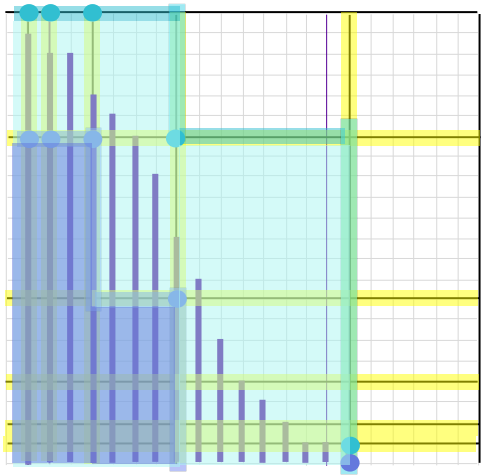
into one sampled merged distribution **with γ and b**



ERank For d>2

The **ratio between** the final **upper** and **lower bounds** of a given key is bounded by

$< \gamma^{2d-2}$



Apply ESRank for two-dimensions

Results

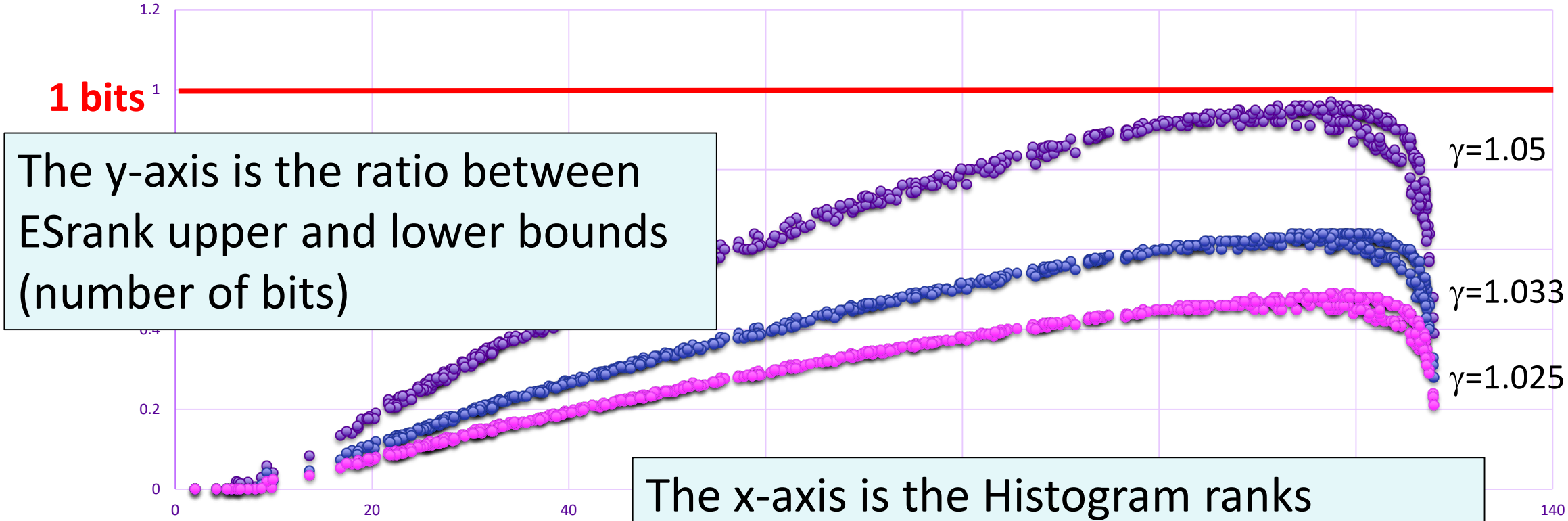
- Our probability distributions are gathered from a specific SCA against AES with 128-bits keys.
- It contains d = 16 probability lists each of size $N = 2^8$
- We merge them into d = 8 lists each of size $N = 2^{16}$
- The distributions are sorted in non-increasing order of probability.

Results

- We measured the **upper bound**, **lower bound**, **time** and **space** for each trace using **ESrank** and the **Histogram** rank estimation of Glowacz et al. [GGPSS15].
- We run with different values of γ and histogram size B.

Results - Accuracy

Excellent accuracy of less than 1 bit margin between ESrank lower and upper bounds



The y-axis is the ratio between ESrank upper and lower bounds (number of bits)

The x-axis is the Histogram ranks (number of bits)

Results – Time and Space

ESrank performance is on-par with the Histogram algorithm:

	Time (Seconds)	Space (MB)	Accuracy < 1 bit (%)
$\gamma = 1.025$	0.59	6.48	100
$\gamma = 1.033$	0.3	3.68	100
$\gamma = 1.05$	0.16	1.60	99.83
$\gamma = 1.065$	0.05	0.96	56.95
$B = 50K$	0.62	3.20	100
$B = 35K$	0.29	2.24	100
$B = 20K$	0.12	1.28	100
$B = 5K$	0.01	0.32	99.83

ESrank algorithm gives:

- **Excellent accuracy of less than 1 bit**
- in **less than 1 second**
- on a standard laptop using **6.5 MB RAM**.

Conclusion

- In this paper we propose a **new**, **simple** and **effective new rank estimation** method called ESrank.
- Our main idea is to use **exponential sampling to drastically reduce** the algorithm's **complexity**.
- We prove ESrank has a **poly-logarithmic time- and space-**complexity.
- ESrank is **simple to build from scratch**, and requires no algorithmic tools beyond a sorting function.