Changing Base without Losing Space

Yevgeniy Dodis

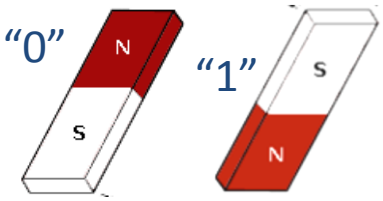
Mihai Pătraşcu

Mikkel Thorup

Cookie Talk, Apr. 8’10

To appear at STOC’10.

Changing Base



Represent a vector $A[1..n] \in \{0,..,9\}^n$ (…on a binary computer)

Support reading and writing $A[i]$

	Space	**Time**
Write a big number (10^n) in binary	$\lceil n \cdot \log_2 10 \rceil$	$O(n)$
Store each digit with 4 bits	$4n$	$O(1)$

Changing Base

Represent a vector $A[1..n] \in \{0,..,9\}^n$ (…on a binary computer)

Support reading and writing $A[i]$

	Space	**Time**
"arithmetic coding"	$\lceil n \cdot \log_2 10 \rceil$	$O(n)$
"Huffman coding"	$4n$	$O(1)$

Changing Base

Represent a vector $A[1..n] \in \{0,..,9\}^n$ (…on a binary computer)

Support reading and writing $A[i]$

	Space	**Time**
"arithmetic coding"	$\lceil n \cdot \log_2 10 \rceil$	O(n)
"Huffman coding"	4n	O(1)
This paper	$\lceil n \cdot \log_2 10 \rceil$	O(1)

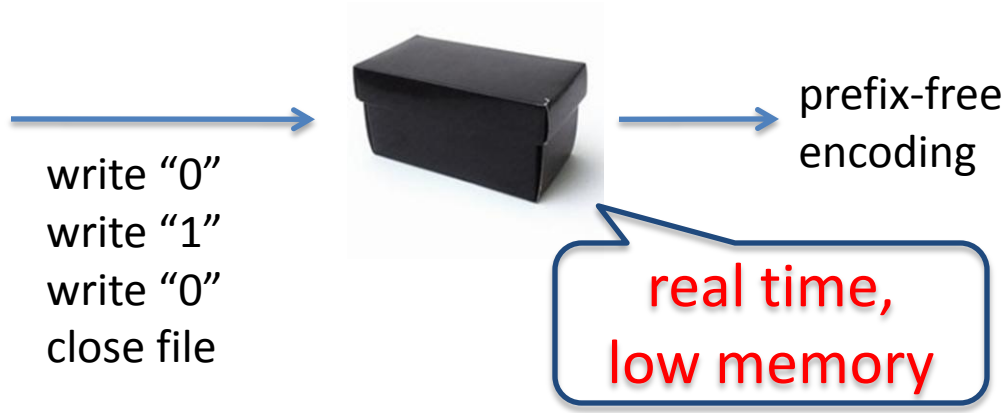
Prefix-Free Coding

Represent $A[1..n] \in \{0,1\}^n, \forall n$ by a *prefix-free* code

Elias codes:

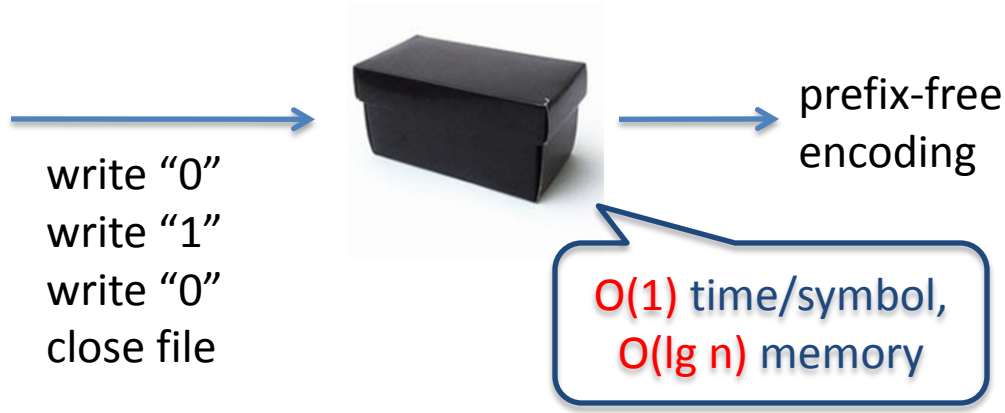
Code length	How
$2n$	append a bit after each $A[i]$ ("more? yes/no")
$n+2\lg n$	at the beginning, write "n" by the code above
$n+\lg n+2\lg\lg n$	...
$n + \lg n + \ldots + O(\lg^* n)$	...

Online Prefix-Free Coding



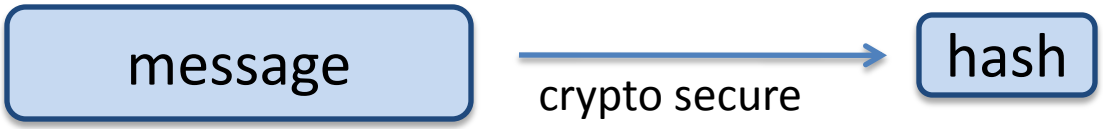
Code length	How
2n	append a bit after each A[i] (“more? yes/no”)
n+2√n	a bit after each block of size √n
^^^	Optimal? [Maurer, Sjödin]

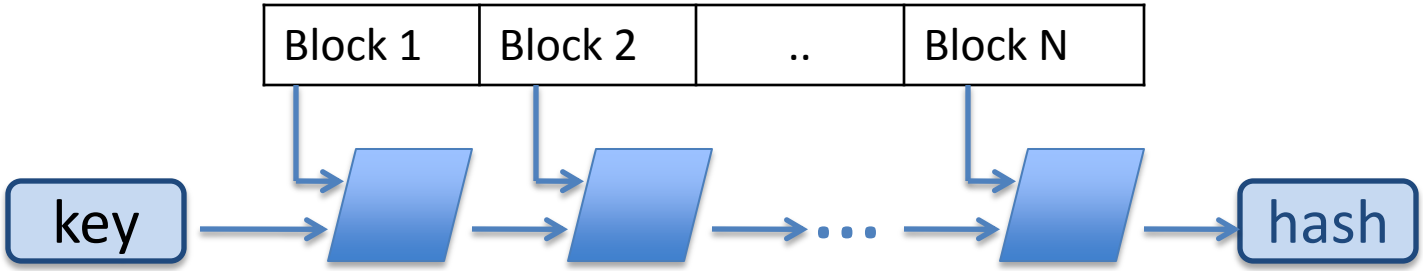
Online Prefix-Free Coding



Code length	How
2n	append a bit after each A[i] (“more? yes/no”)
n+2√n	a bit after each block of size √n
^^^	Optimal? [Maurer, Sjödin]
n + lg n + O(lglg n)	This paper.

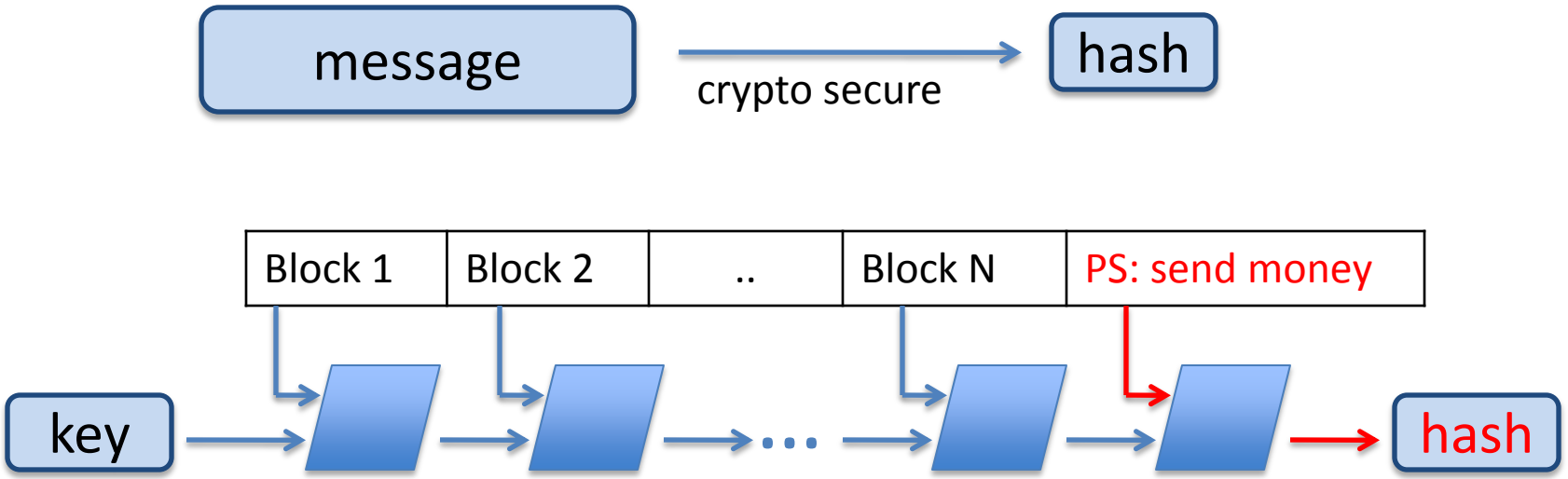
Why Prefix-Free Coding?





Is this secure?

Why Prefix-Free Coding?



Prefix-free ➔ security

Prefix Free ≈ Changing Base

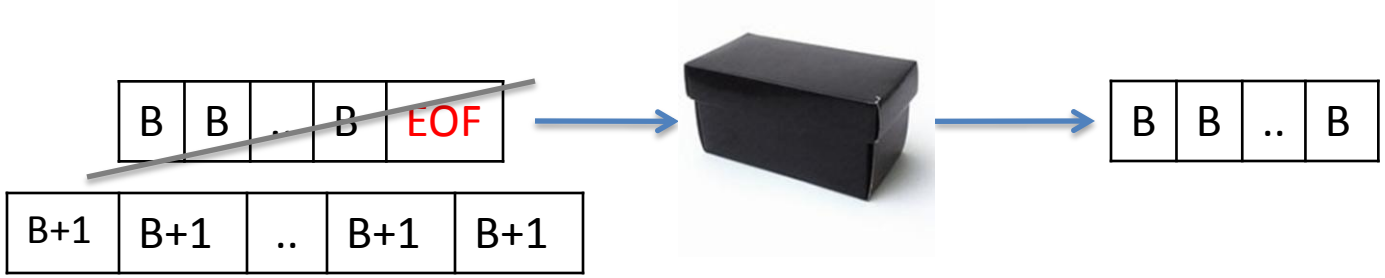
Crypto: big blocks (b=128 bits or more)

Let $B=2^b$

Prefix Free ≈ Changing Base

Crypto: big blocks (b=128 bits or more)

Let $B=2^b$



Entropy loss: $n \cdot \log_2(B+1) - n \cdot \log_2 B = n \cdot \log_2(1+ {}^1/_B) = O(n/B)$

So if $B \gg N$, i.e. $b \gg \log_2 n$, negligible loss…

The Algorithm in One Slide

Block #	1	2	3	4	5	6	…
Input alphabet	B	B	B	B	B	B	…
With EOF	B+1	B+1	B+1	B+1	B+1	B+1	…
Split	B	B+3	B-3	B+6	B-6	B+9	…
Regroup	B	B+3	B-3	B+6	B-6	B+9	…
Merge	B	B	B	B	B	B	…

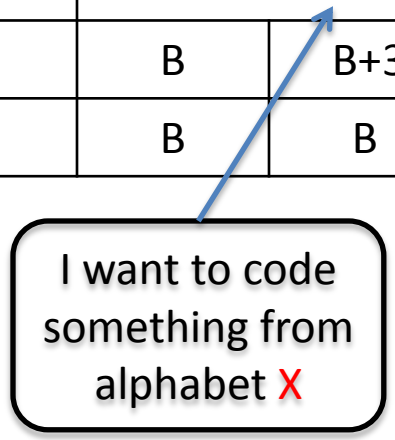
Doing the math:

Merge: $(B+3i)(B-3i) < B^2$

Split: $(B-3i)(B+3i+3) > (B+1)^2$ if $B>4n^2$

What have we learned?

Block #	1	2	3	4	5	6	…
Input alphabet	B	B	B	B	B	B	…
With EOF	B+1	B+1	B+1	B+1	B+1	B+1	…
Split	B	B+3	B-3	B+6	B-6	B+9	…
Regroup	B	B+3	B-3	B+6	B-6	B+9	…
Merge	B	B	B	B	B	B	…



What have we learned?

Block #	1	2	3	4	5	6	…
Input alphabet	B	B	B	B	B	B	…
With EOF	B+1	B+1	B+1	B+1	B+1	B+1	…
Split	B	B+3	B-3	B+6	B-6	B+9	…
Regroup	B	B+3	B-3	B+6	B-6	B+9	…
Merge	B	B	B	B	B	B	…

I want to code something from alphabet X

Combine with some Y such that: $X \cdot Y \approx 2^m$

What have we learned?

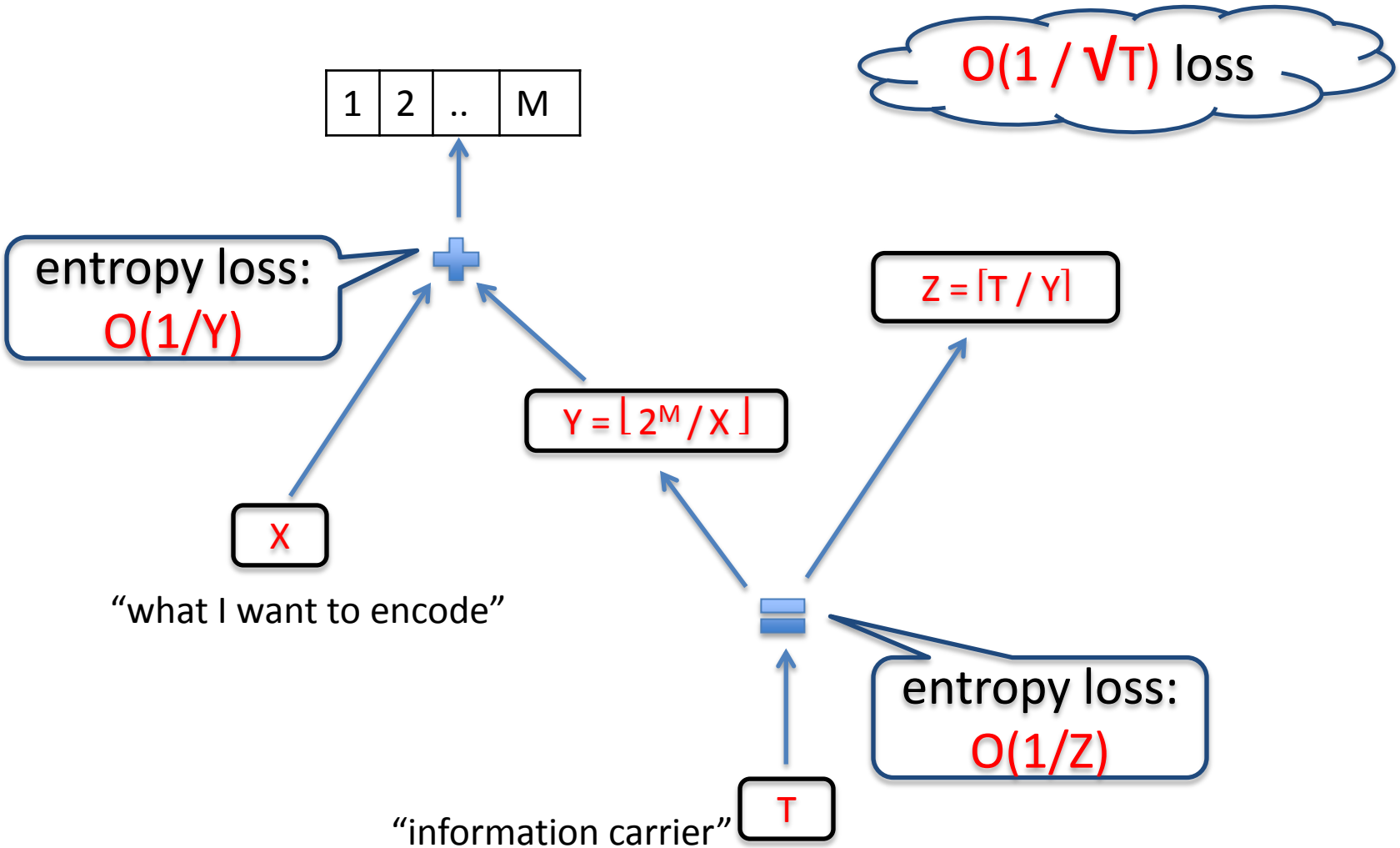
Block #	1	2	3	4	5	6	…
Input alphabet	B	B	B	B	B	B	…
With EOF	B+1	B+1	B+1	B+1	B+1	B+1	…
Split	B	B+3	B-3	B+6	B-6	B+9	…
Regroup	B	B+3	B-3	B+6	B-6	B+9	…
Merge	B	B	B	B	B	B	…

I want to code something from alphabet X

Combine with some Y such that: $X \cdot Y \approx 2^m$

Take a large alphabet T and split into: $T \approx Y \cdot Z$

Information Carrier Codes



The End

Open Problem:

- locally decodable arithmetic codes
  for *non-uniform* distributions