RUB

RUHR-UNIVERSITÄT BOCHUM

Dissection-BKW

CRYPTO 2018, Santa Barbara, August 20th 2018

Andre Esser, Felix Heuer, Robert Kübler,
Alexander May, Christian Sohler
Horst Görtz Institute for IT Security
Ruhr University Bochum

hgi
Horst Görtz Institut
für IT-Sicherheit

What is LPN?

Learning Parity with Noise (LPN) Problem

Given: $(\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i),\ \mathbf{a}_i \xleftarrow{\$} \mathbb{F}_2^k,\ \Pr[e_i = 1] = \tau < \frac{1}{2}$

Find: $\mathbf{s} \in \mathbb{F}_2^k$

What is LPN?

Learning Parity with Noise (LPN) Problem

Given: $(\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i),\ \mathbf{a}_i \xleftarrow{\$} \mathbb{F}_2^k,\ \Pr[e_i = 1] = \tau < \frac{1}{2}$
Find: $\mathbf{s} \in \mathbb{F}_2^k$

- Cryptographic applications [HB01, Ale03, HKL$^+$12, DV13]

What is LPN?

Learning Parity with Noise (LPN) Problem

Given: $(\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i),\ \mathbf{a}_i \xleftarrow{\$} \mathbb{F}_2^k,\ \Pr[e_i = 1] = \tau < \frac{1}{2}$
Find: $\mathbf{s} \in \mathbb{F}_2^k$

- Cryptographic applications [HB01, Ale03, HKL+12, DV13]
- Solve LPN: BKW algorithm [BKW00]
  - Time = **Memory** = **Samples**, slightly subexponential
  - only small experiments [BTV16, EKM17]

What is LPN?

Learning Parity with Noise (LPN) Problem

Given: $(\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i),\ \mathbf{a}_i \xleftarrow{\$} \mathbb{F}_2^k,\ \Pr[e_i = 1] = \tau < \frac{1}{2}$
Find: $\mathbf{s} \in \mathbb{F}_2^k$

- Cryptographic applications [HB01, Ale03, HKL+12, DV13]
- Solve LPN: BKW algorithm [BKW00]
  - Time = **Memory** = **Samples**, slightly subexponential
  - only small experiments [BTV16, EKM17]
- **Goal:** BKW-variant applicable for any given memory

Illustration of “BKW”

$$(\mathbf{a}_1, \langle \mathbf{a}_1, \mathbf{s} \rangle + e_1)$$
$$(\mathbf{a}_2, \langle \mathbf{a}_2, \mathbf{s} \rangle + e_2)$$

Illustration of “BKW”

$$
\begin{aligned}
&(\mathbf{a}_1, \langle \mathbf{a}_1, \mathbf{s} \rangle + e_1) \\
+\ &(\mathbf{a}_2, \langle \mathbf{a}_2, \mathbf{s} \rangle + e_2) \\
\hline
= (&\underbrace{\mathbf{a}_1 + \mathbf{a}_2}_{\mathbf{a}'}, \underbrace{\langle \mathbf{a}_1 + \mathbf{a}_2, \mathbf{s} \rangle}_{\langle \mathbf{a}', \mathbf{s} \rangle} + \underbrace{e_1 + e_2}_{\mathbf{e}'})
\end{aligned}
$$

Illustration of “BKW”

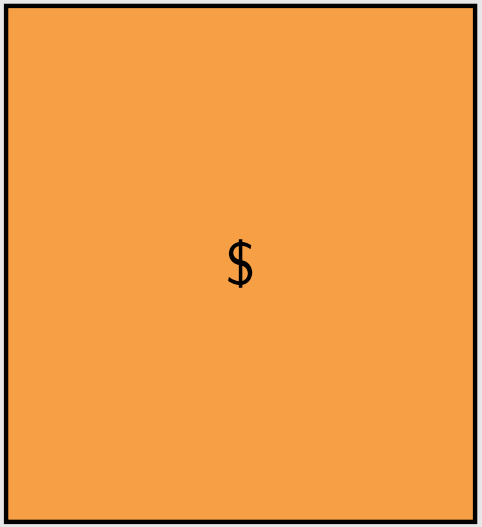


Illustration of “BKW”

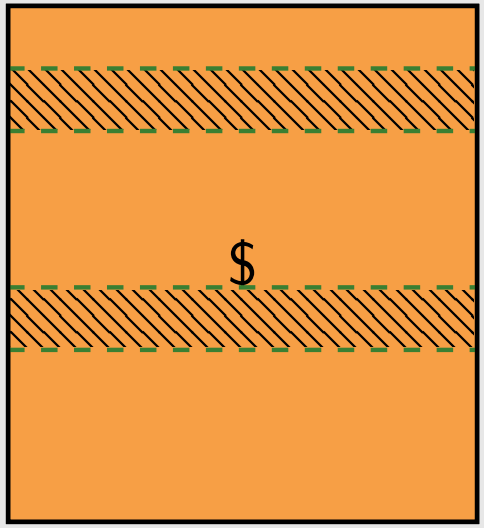


Illustration of “BKW”

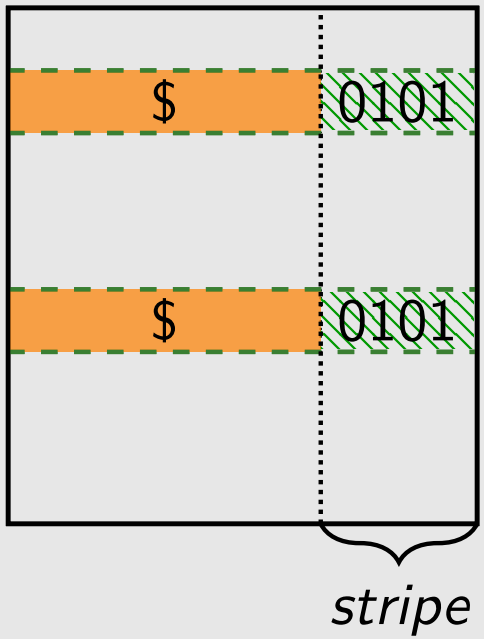


Illustration of “BKW”

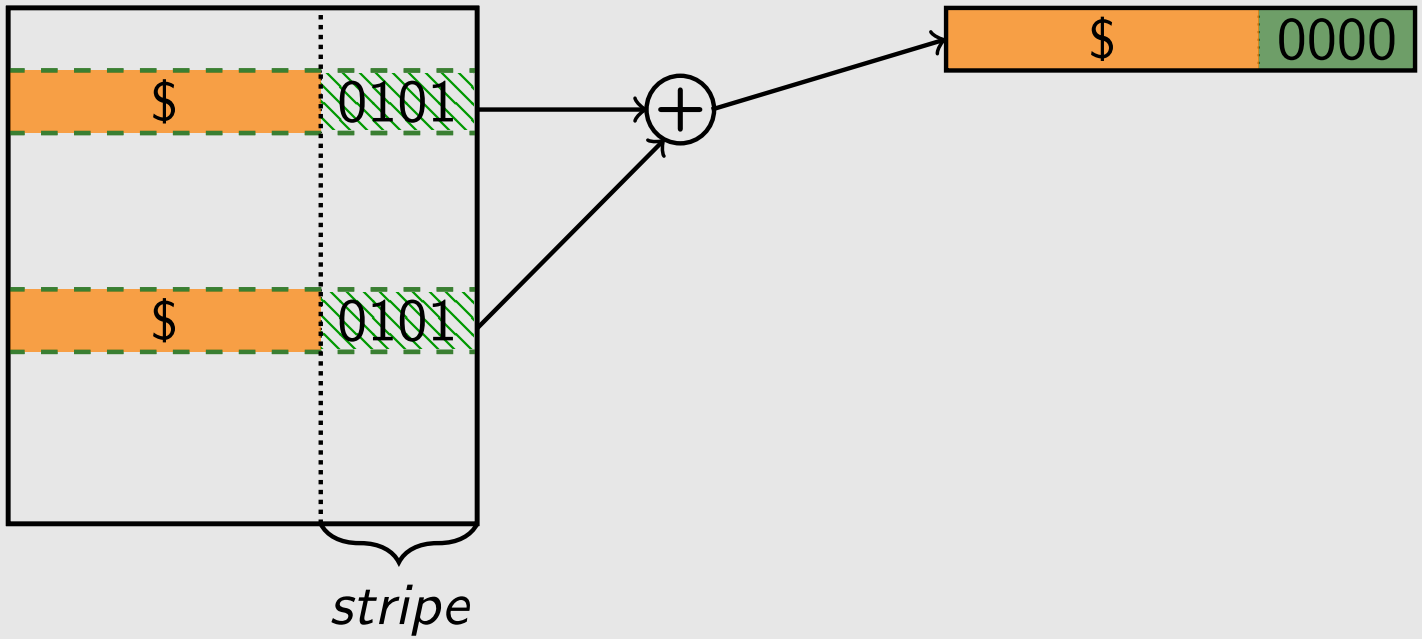


Illustration of “BKW”

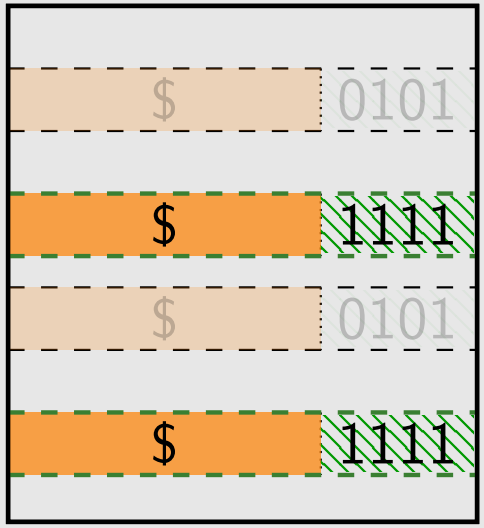


Illustration of “BKW”

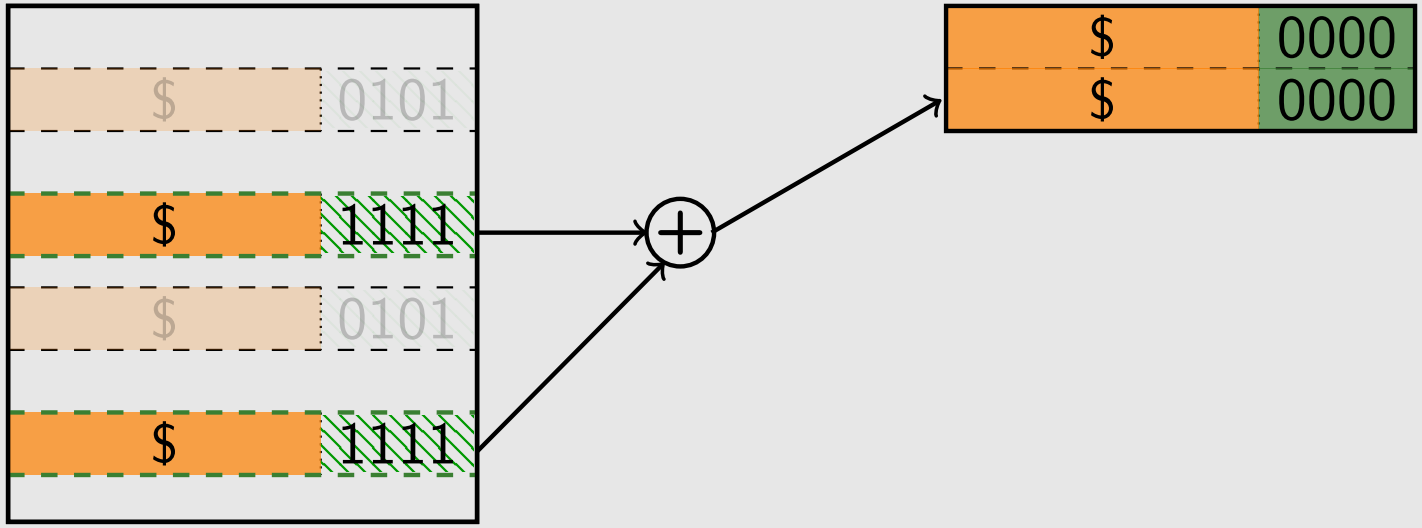
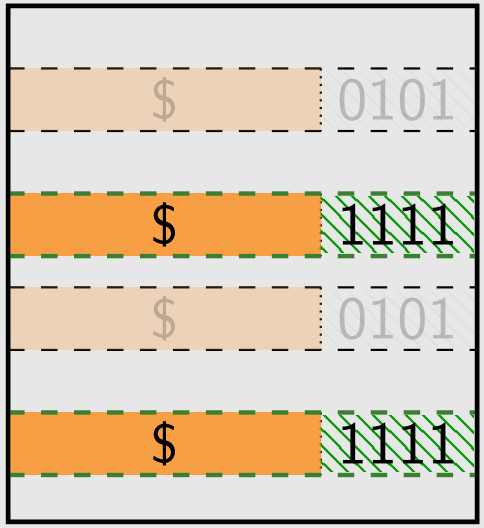


Illustration of “BKW”



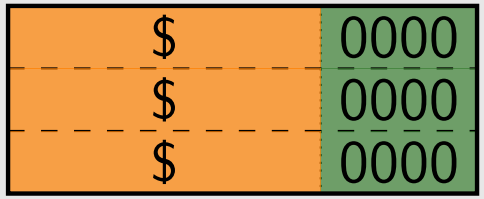
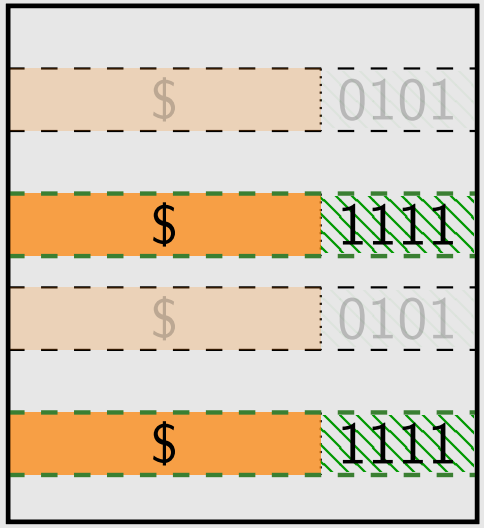


Illustration of “BKW”



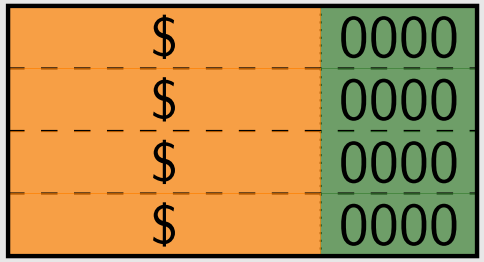
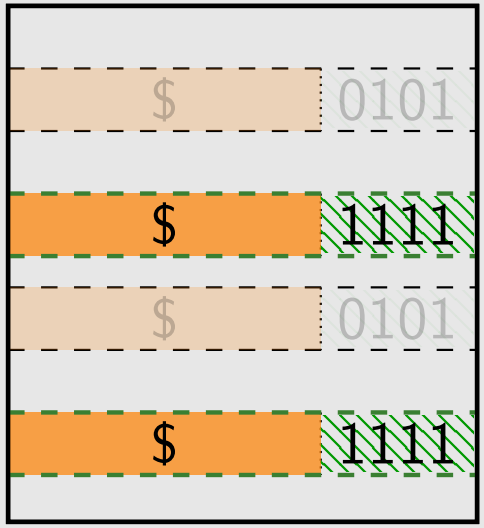


Illustration of “BKW”



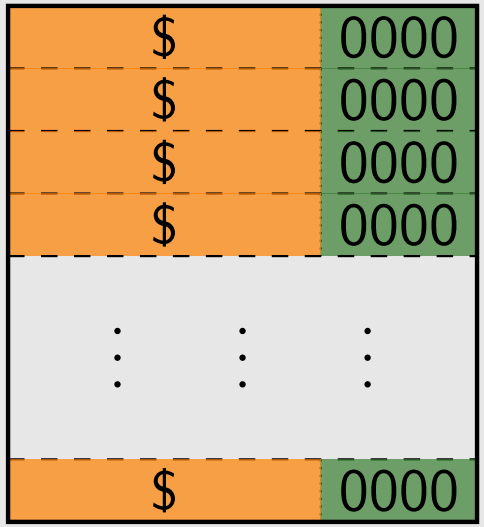


Illustration of “BKW”

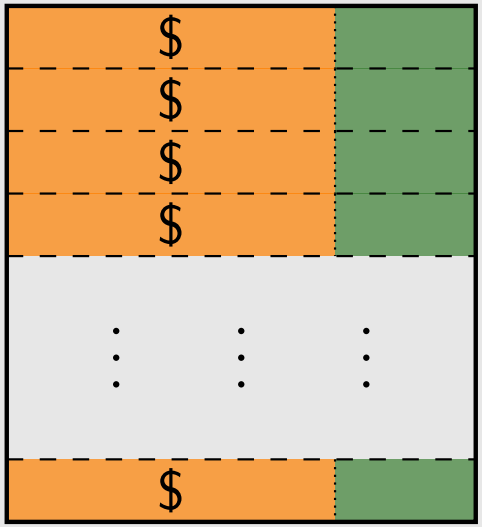


Illustration of “BKW”

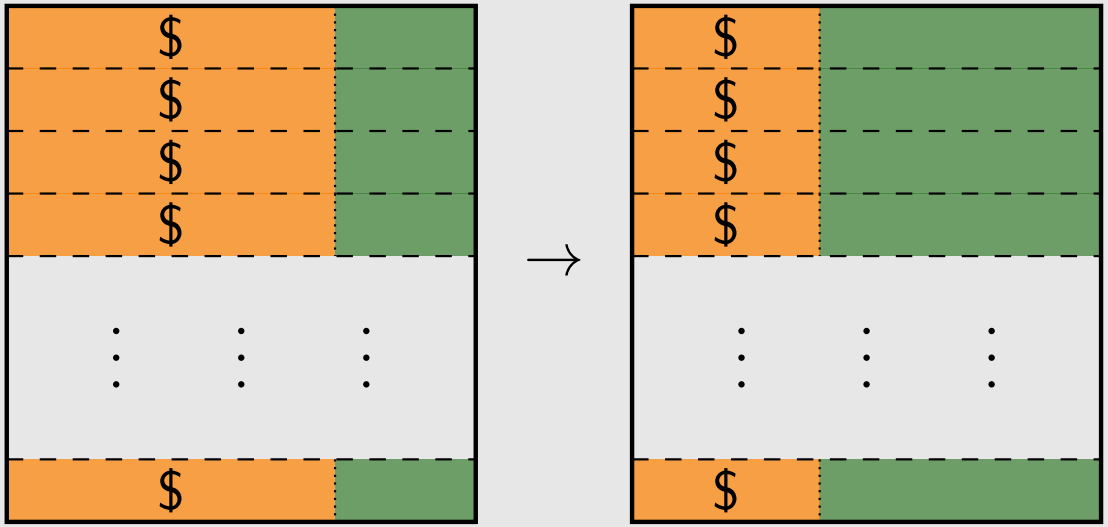


Illustration of “BKW”

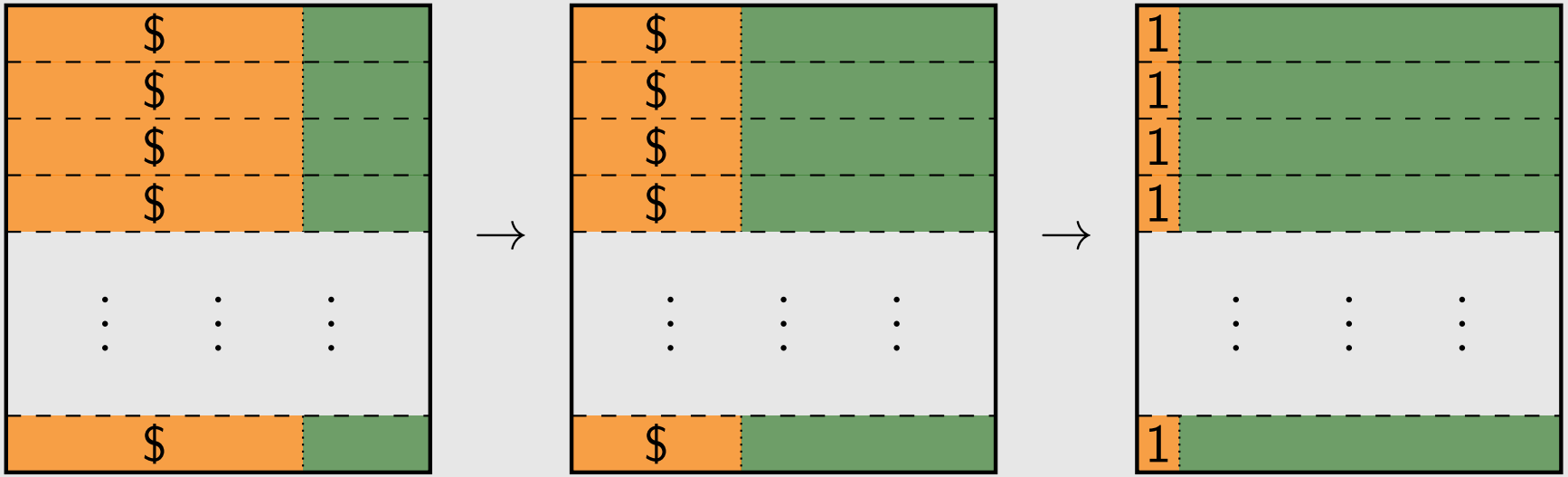
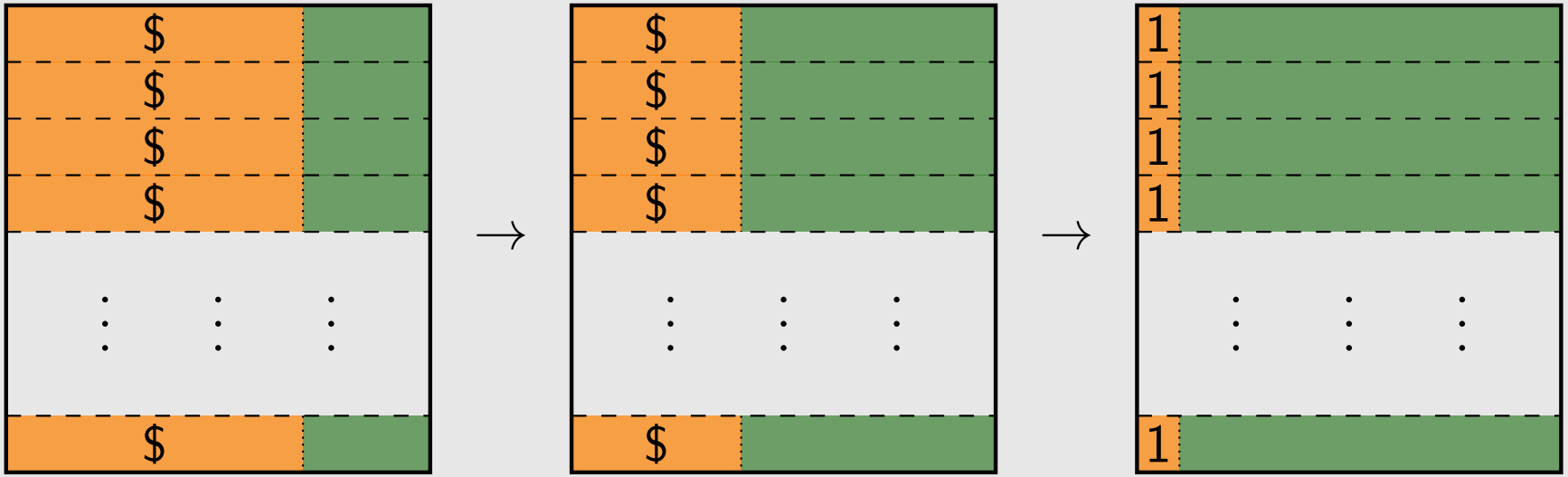


Illustration of "BKW"

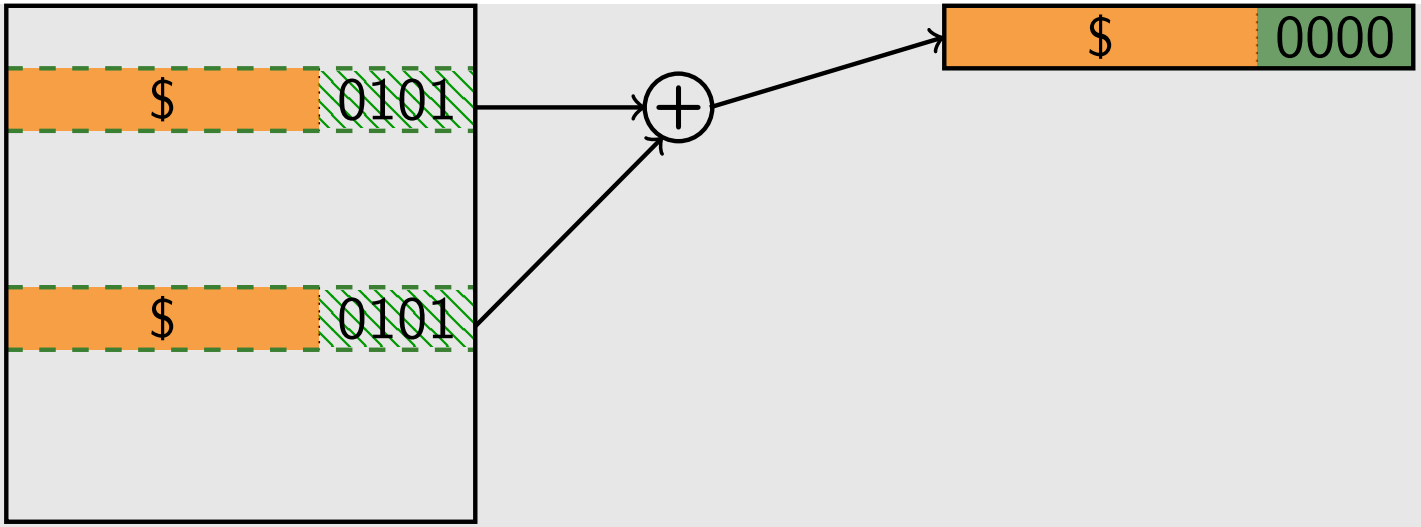


- $\mathbf{a}_i = (1, 0, 0, \ldots, 0) \Rightarrow (\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i) = (\mathbf{a}_i, s_1 + e_i)$
- Majority vote!

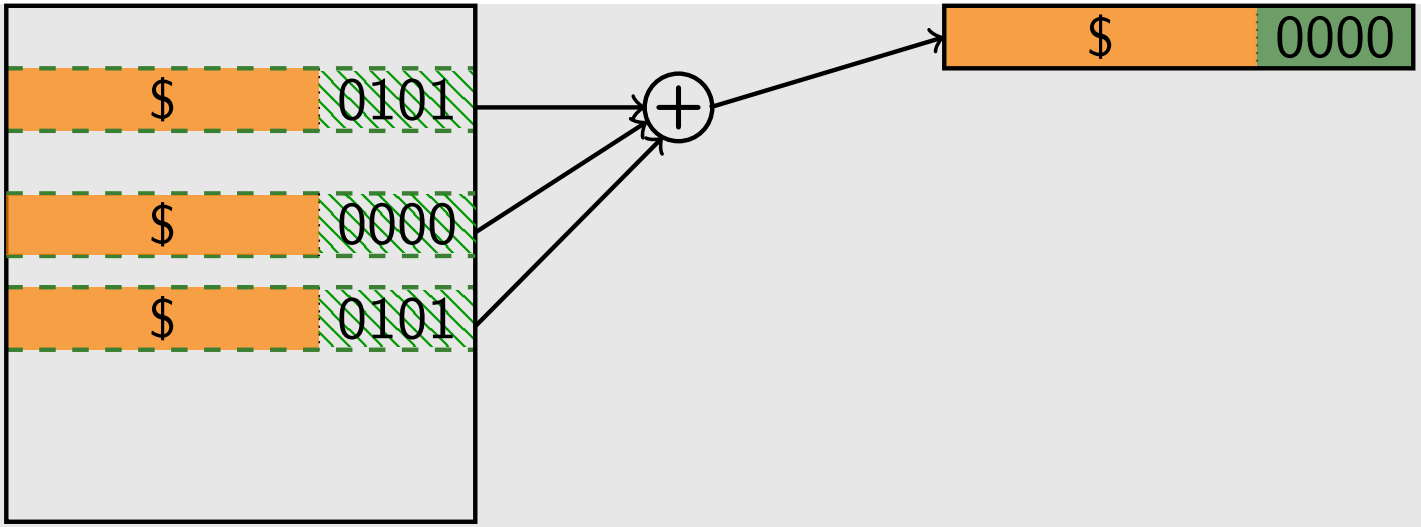
BKW Theorem [BKW00, LF06]

`BKW` solves LPN in time, memory and sample complexity $2^{k/\log k}$.

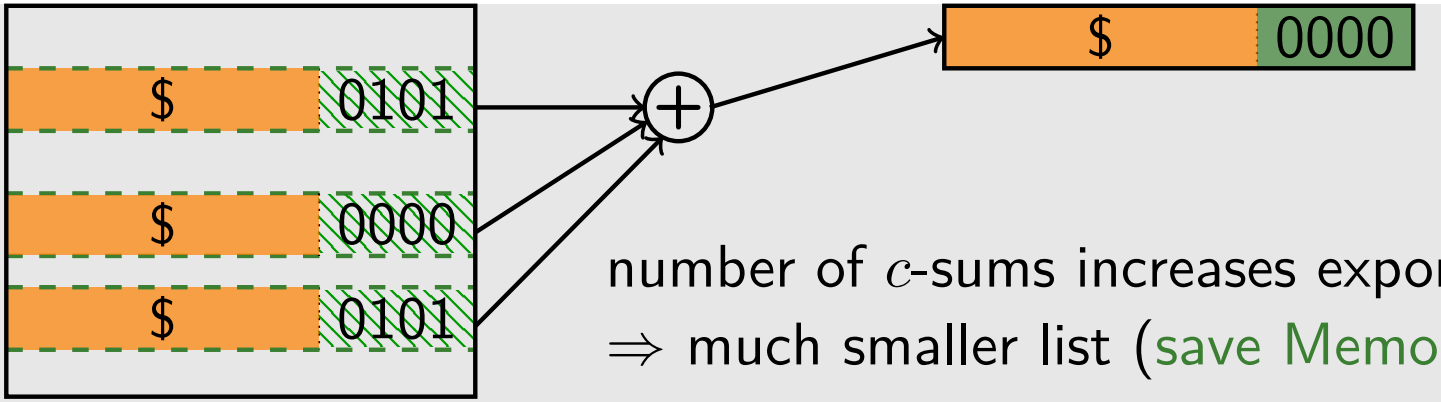
c-sum Observation



c-sum Observation



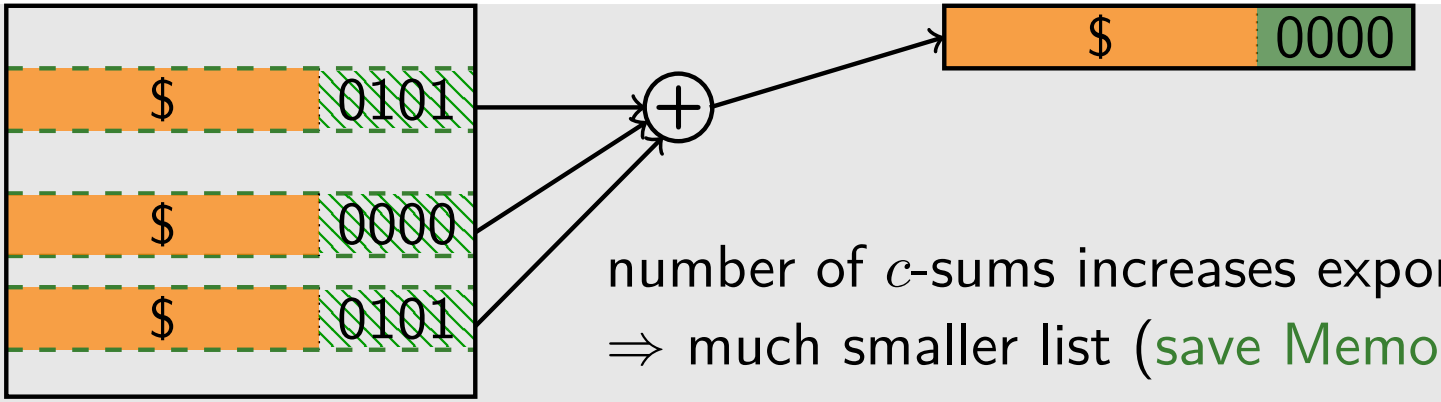
c-sum Observation



number of c-sums increases exponentially in c

$\Rightarrow$ much smaller list (save Memory & Samples)

c-sum Observation

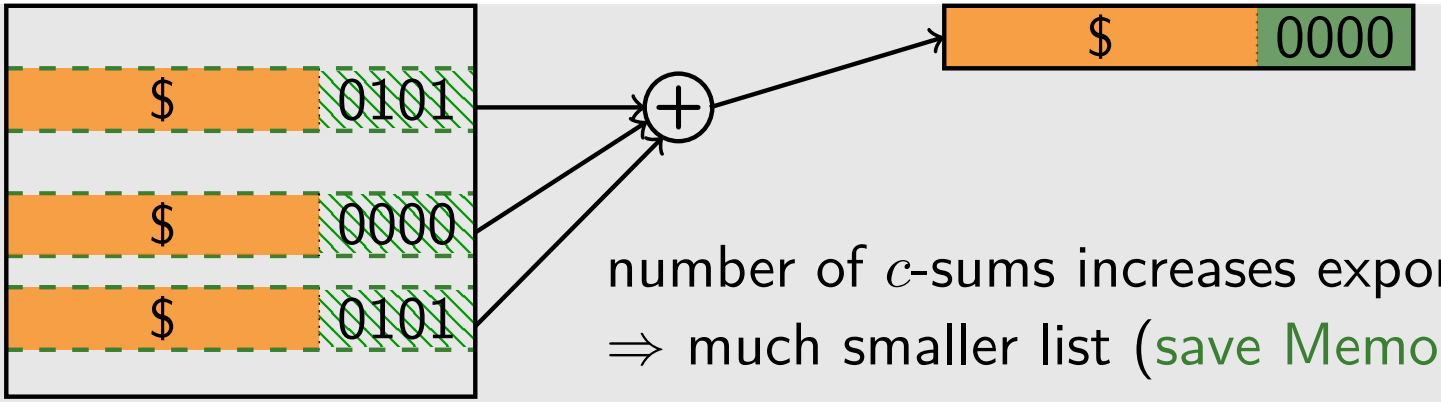


number of c-sums increases exponentially in c
$\Rightarrow$ much smaller list (save Memory & Samples)

c-sum-Problem (c-SP)

Given a list L of N uniformly distributed elements from $\mathbb{F}_2^b$.
Find N combinations of c elements from L that each add up to 0^b.

c-sum Observation



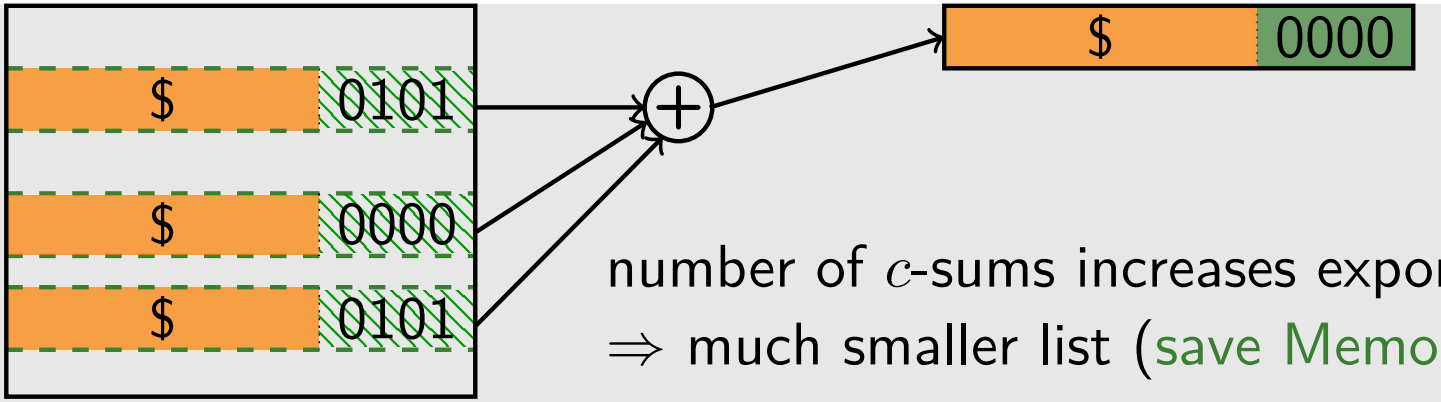
number of c-sums increases exponentially in c
$\Rightarrow$ much smaller list (save Memory & Samples)

$$\binom{N}{c}/2^b \stackrel{!}{\geq} N$$

c-sum-Problem (c-SP)

Given a list L of N uniformly distributed elements from $\mathbb{F}_2^b$.
Find N combinations of c elements from L that each add up to 0^b.

c-sum Observation



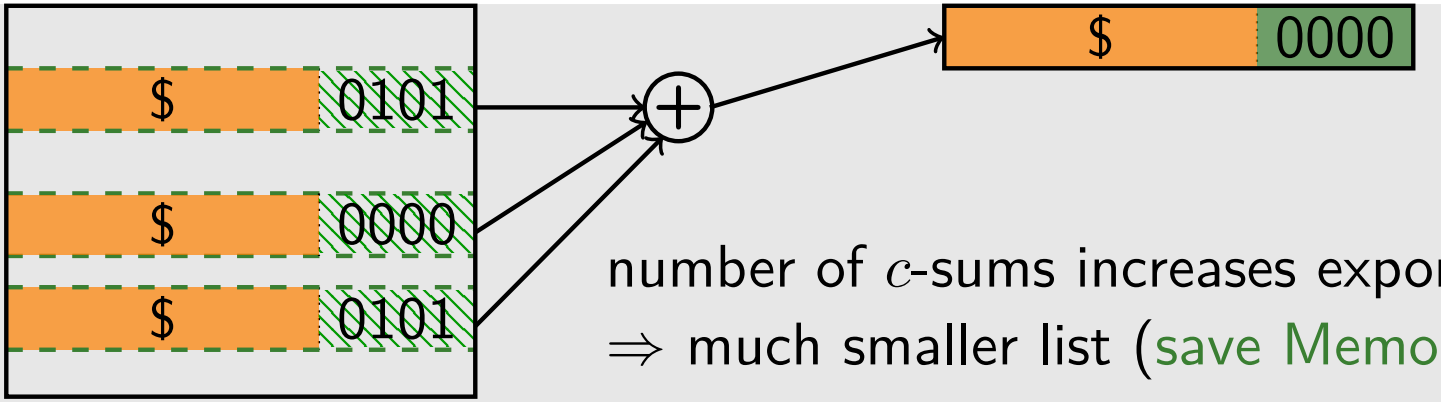
number of c-sums increases exponentially in c
$\Rightarrow$ much smaller list (save Memory & Samples)

$$N \geq 2^{b/(c-1)}$$

c-sum-Problem (c-SP)

Given a list L of N uniformly distributed elements from $\mathbb{F}_2^b$.
Find N combinations of c elements from L that each add up to 0^b.

c-sum Observation



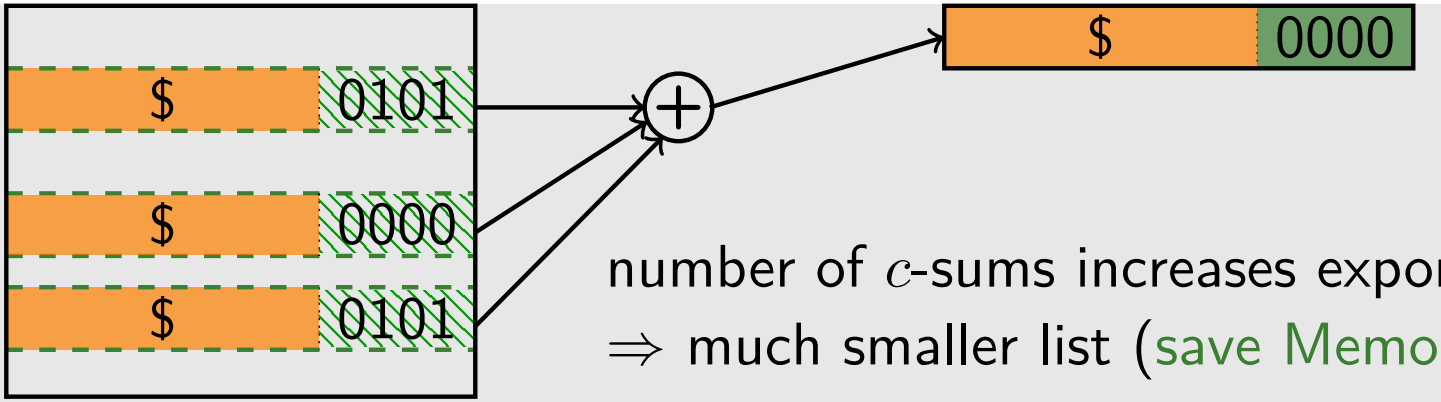
number of c-sums increases exponentially in c
$\Rightarrow$ much smaller list (save Memory & Samples)

$$N = 2^{b/(c-1)}$$

c-sum-Problem (c-SP)

Given a list L of N uniformly distributed elements from $\mathbb{F}_2^b$.
Find N combinations of c elements from L that each add up to 0^b.

c-sum Observation



number of c-sums increases exponentially in c
$\Rightarrow$ much smaller list (save Memory & Samples)

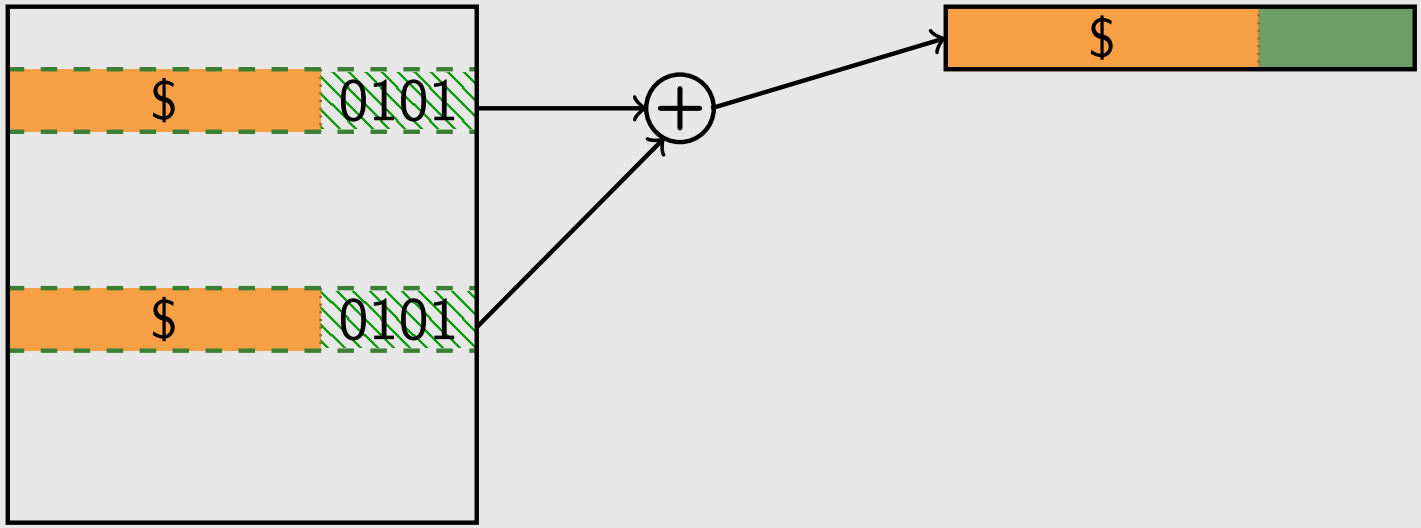
Main Idea: solve c-SP repeatedly on stripes

$$N = 2^{b/(c-1)}$$

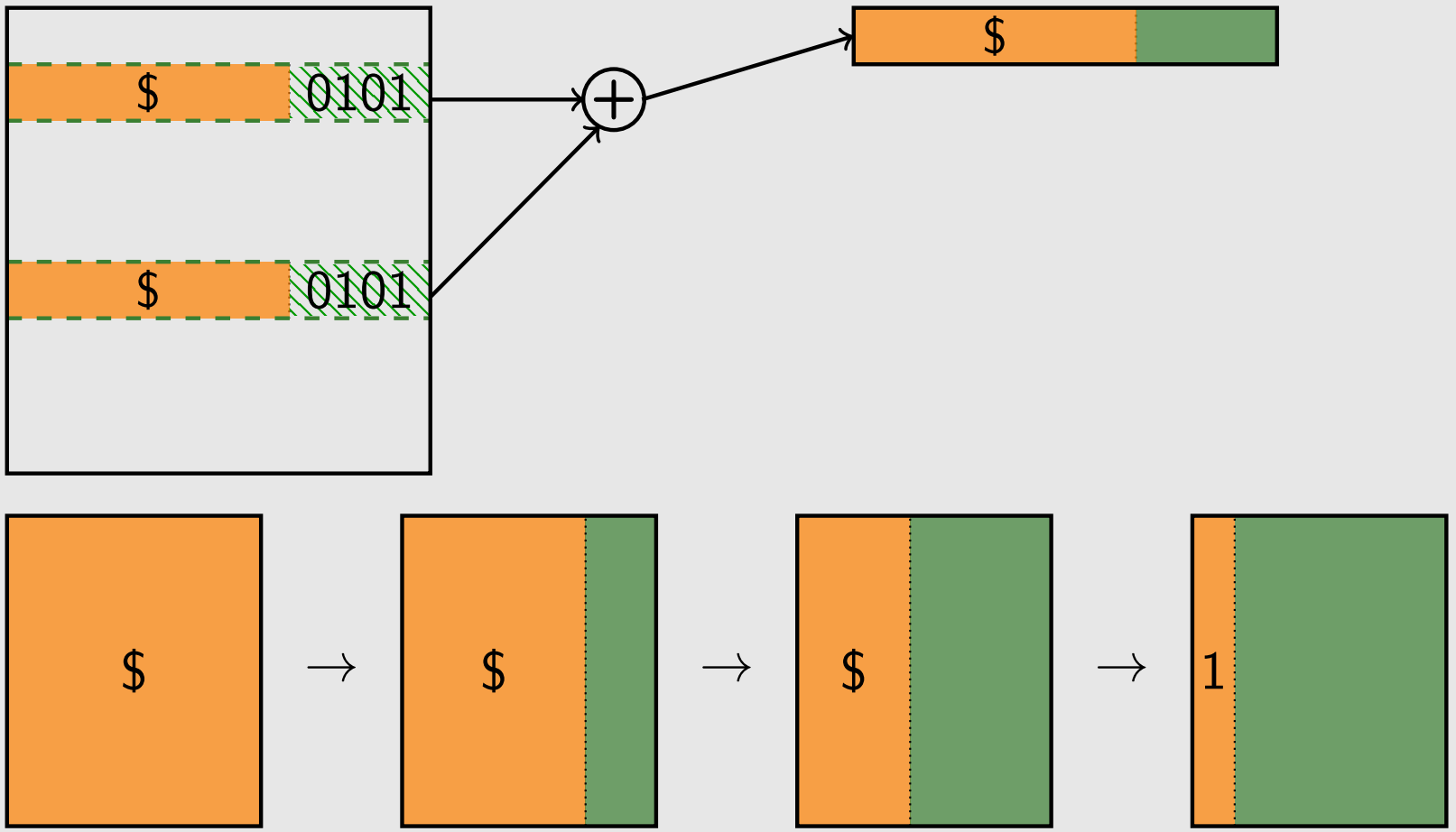
c-sum-Problem (c-SP)

Given a list L of N uniformly distributed elements from $\mathbb{F}_2^b$.
Find N combinations of c elements from L that each add up to 0^b.

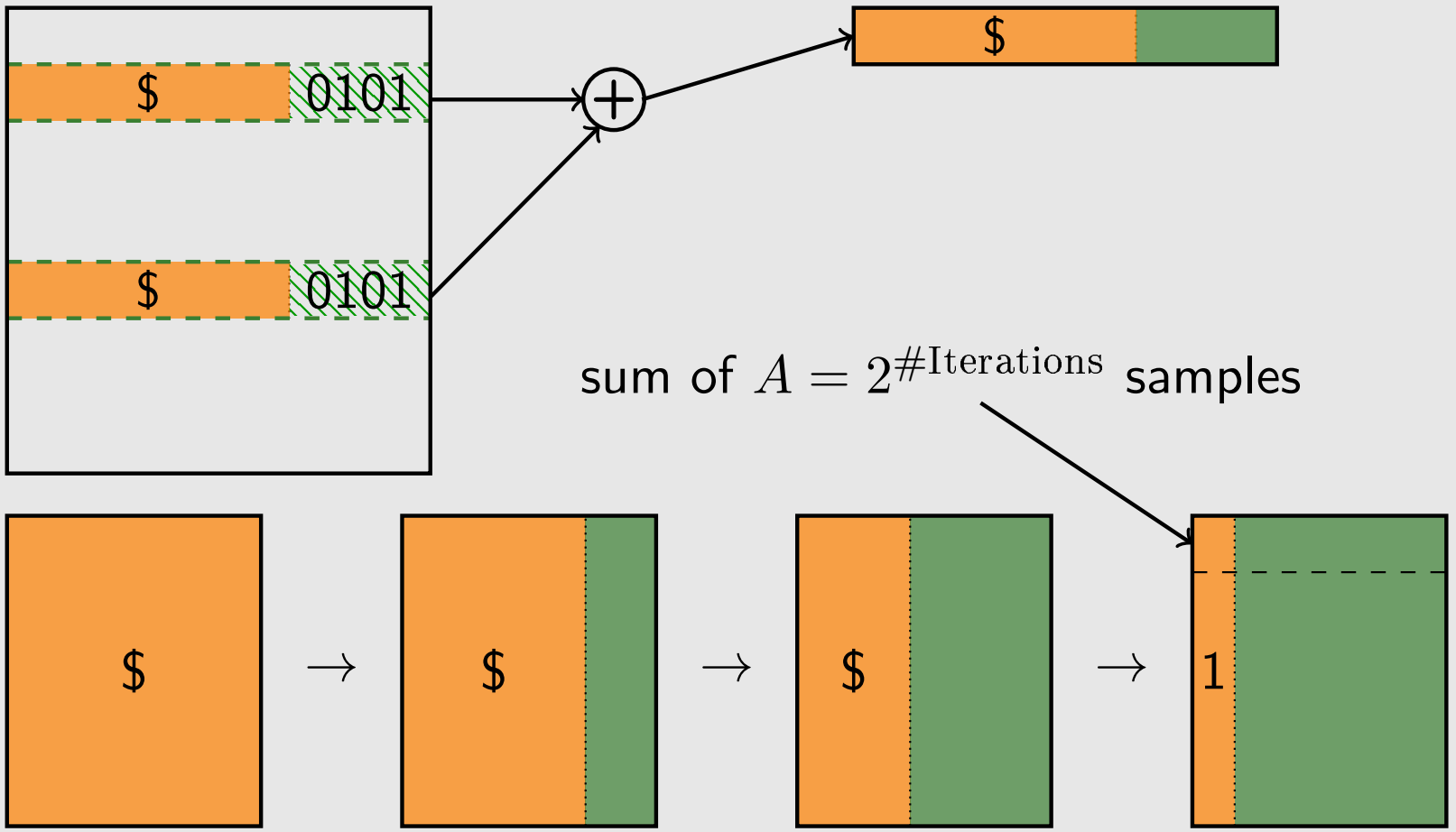
Not just a memory reduction technique



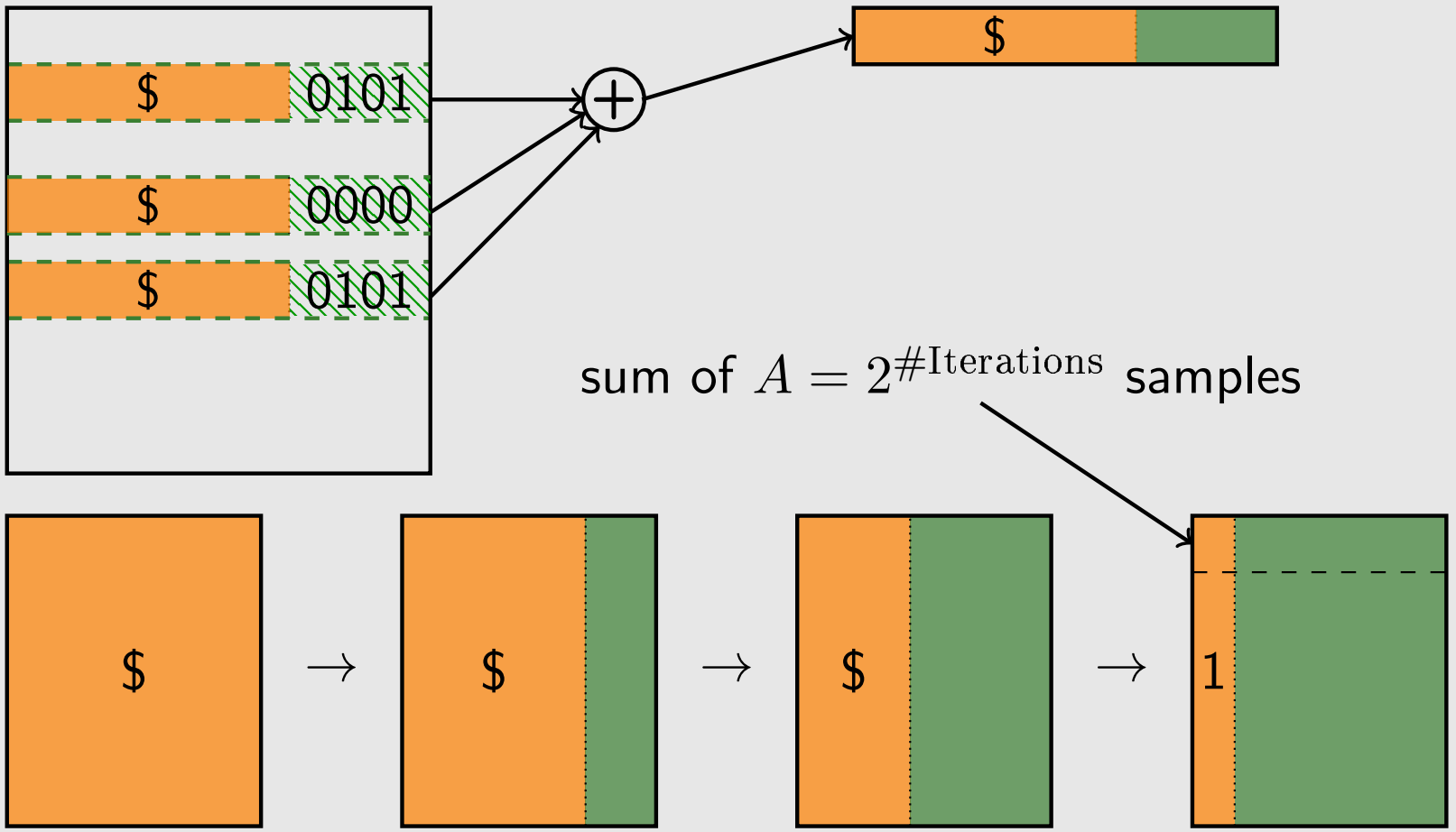
Not just a memory reduction technique



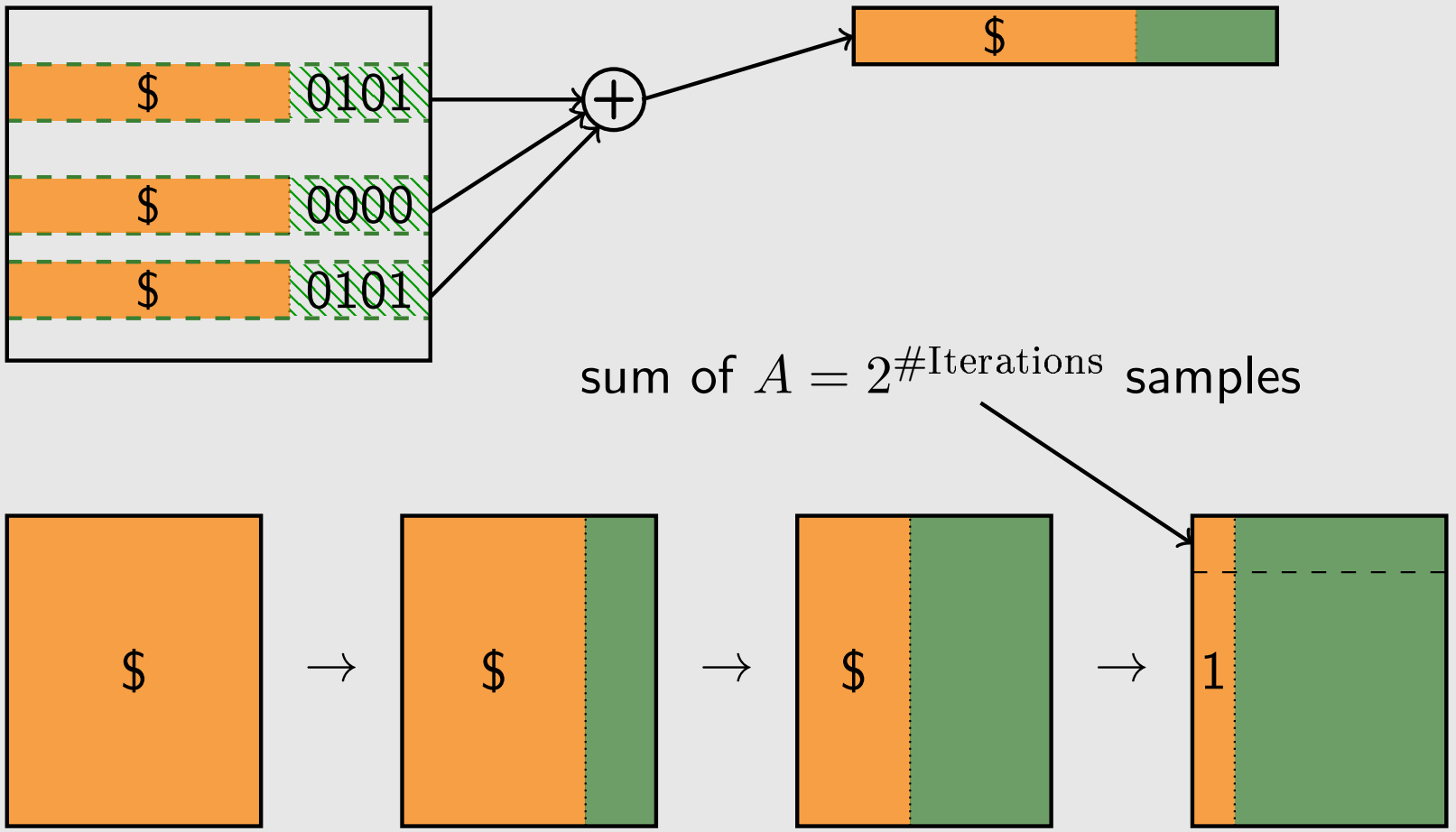
Not just a memory reduction technique



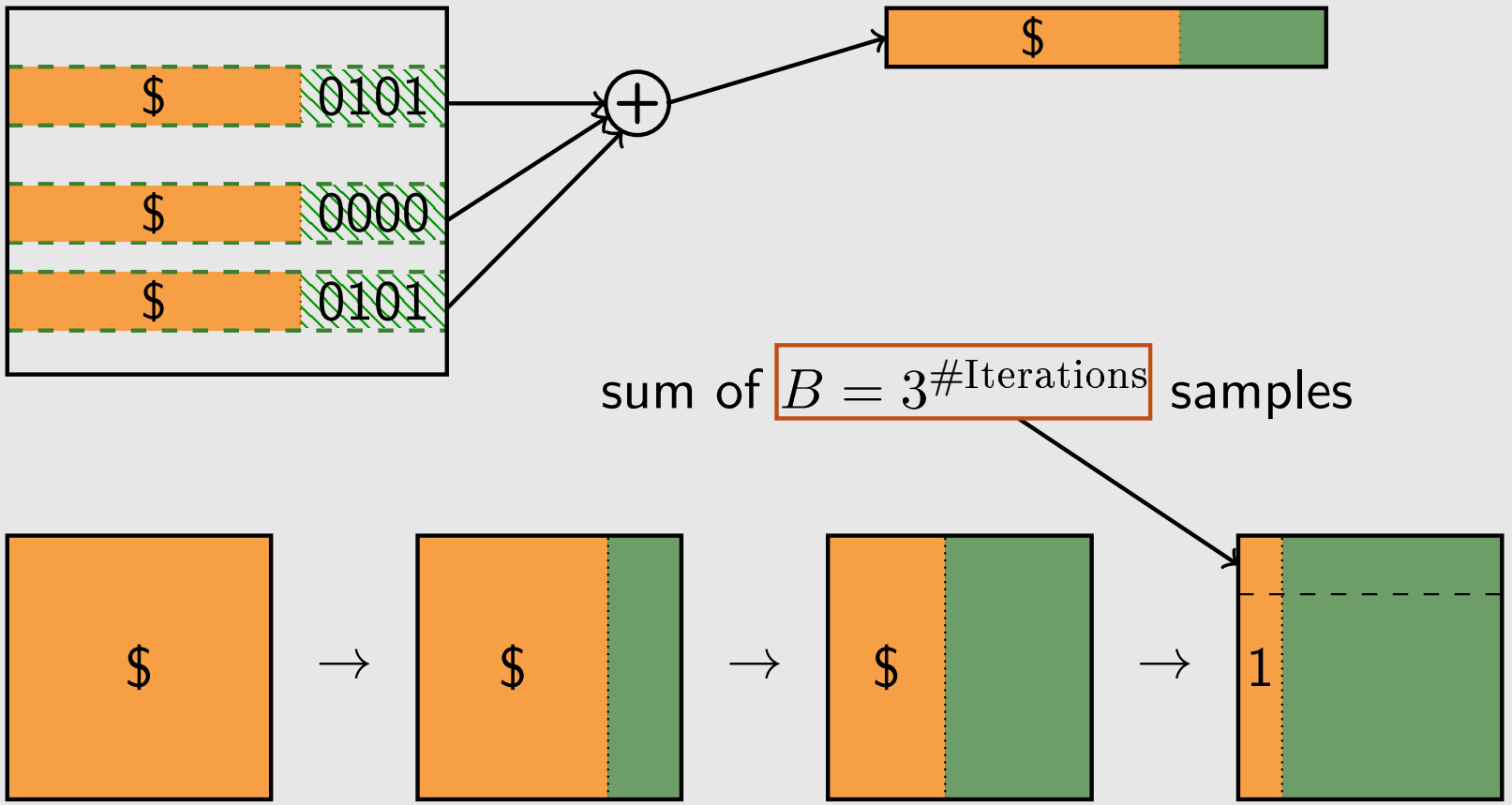
Not just a memory reduction technique



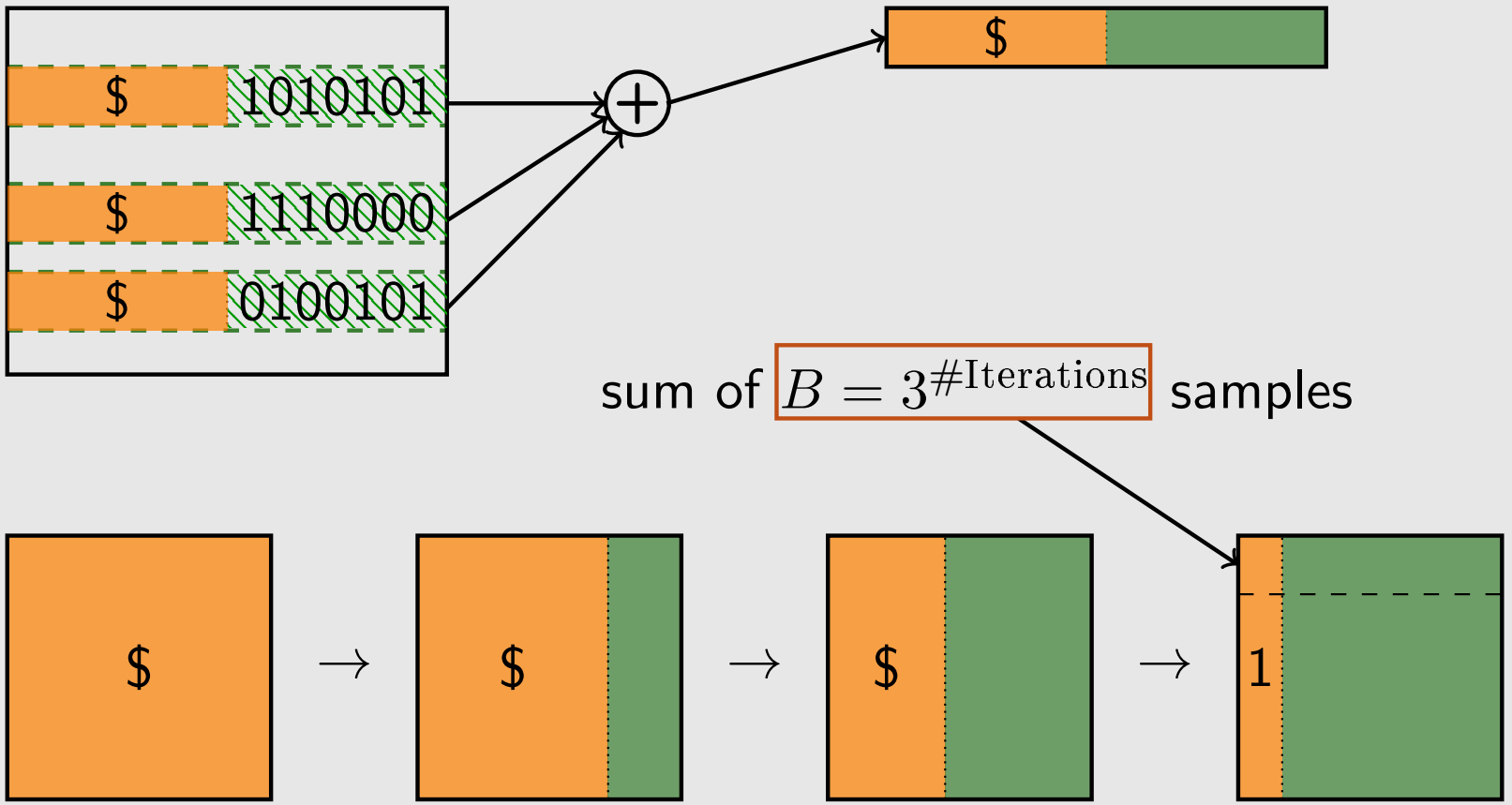
Not just a memory reduction technique



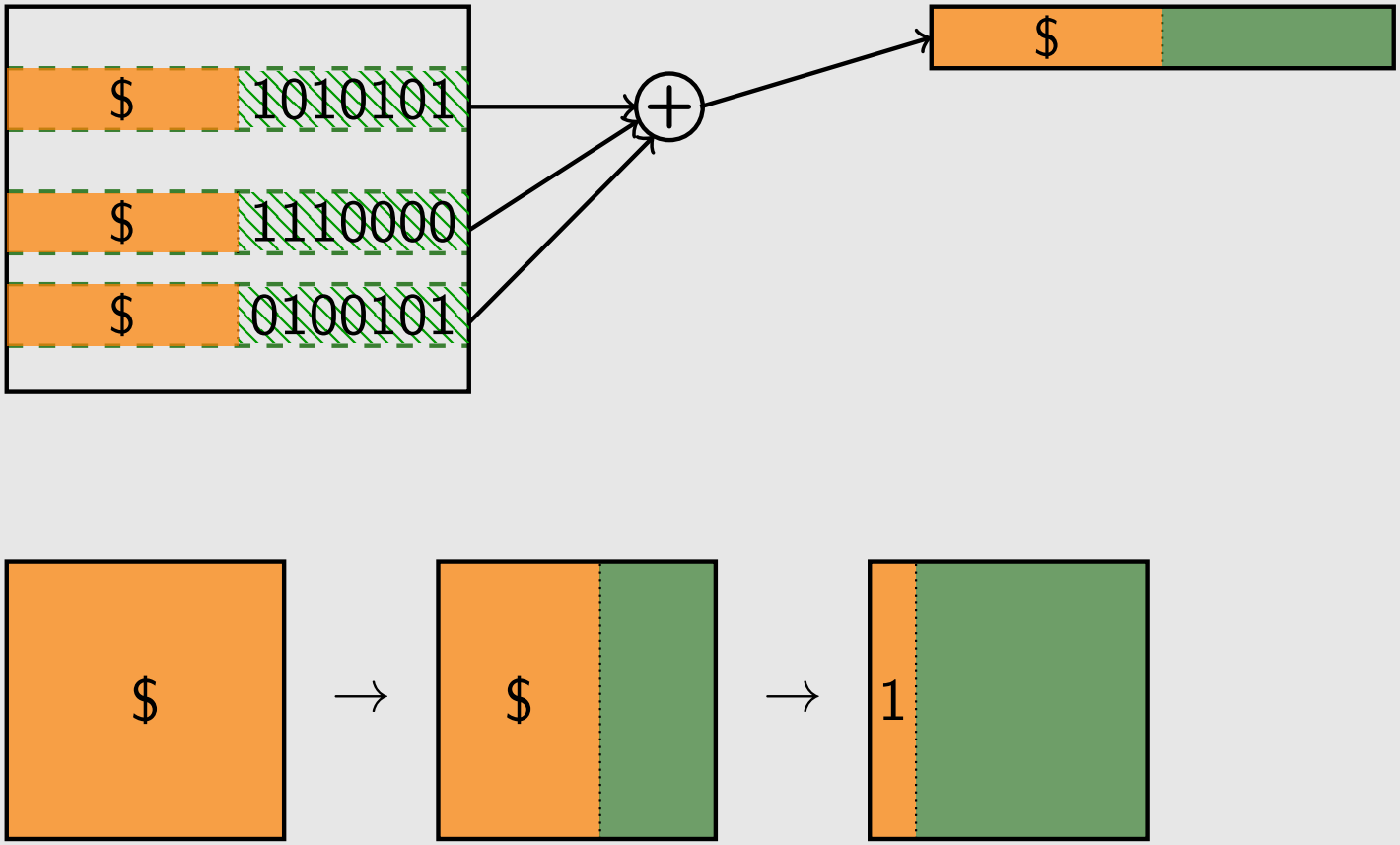
Not just a memory reduction technique



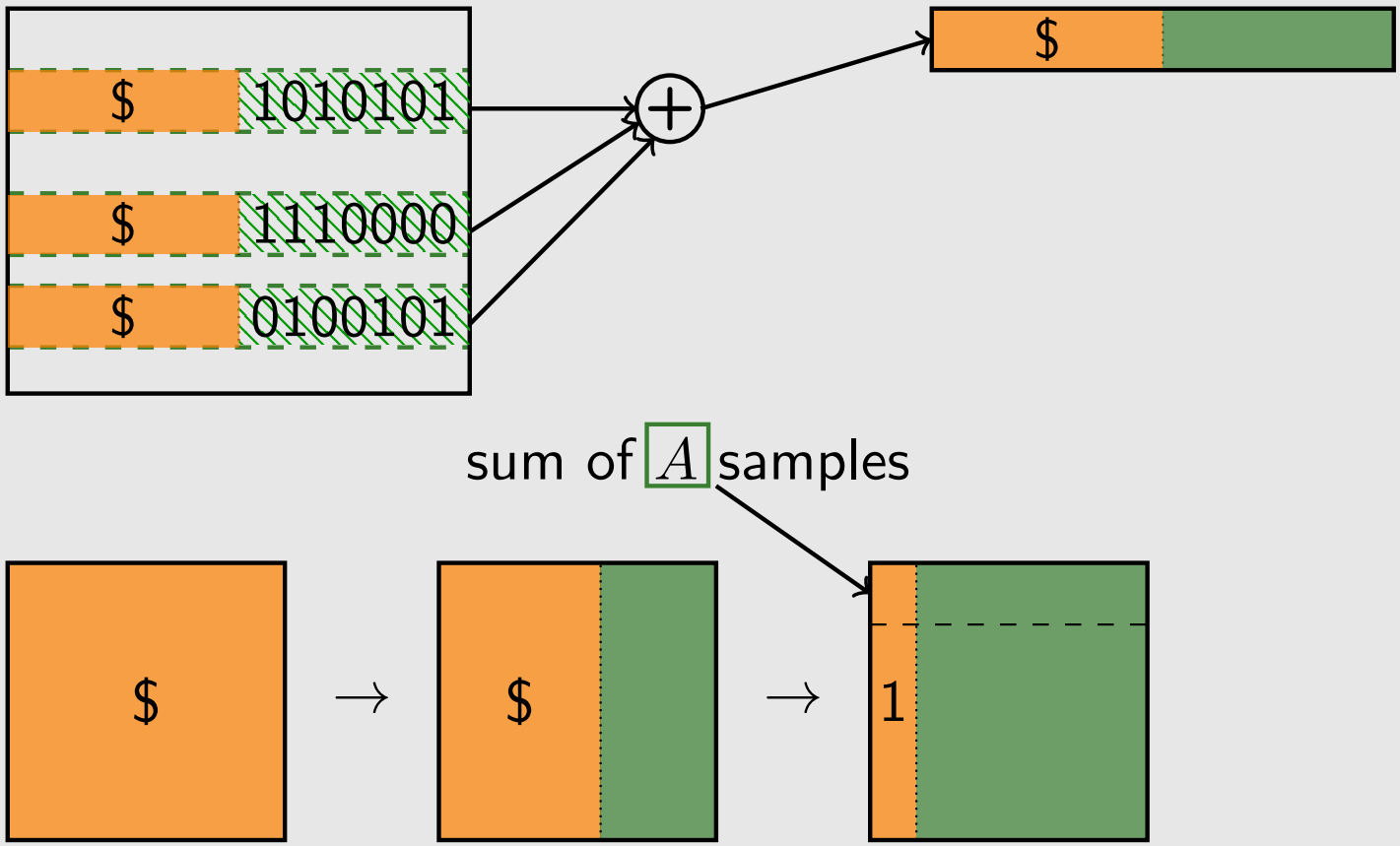
Not just a memory reduction technique



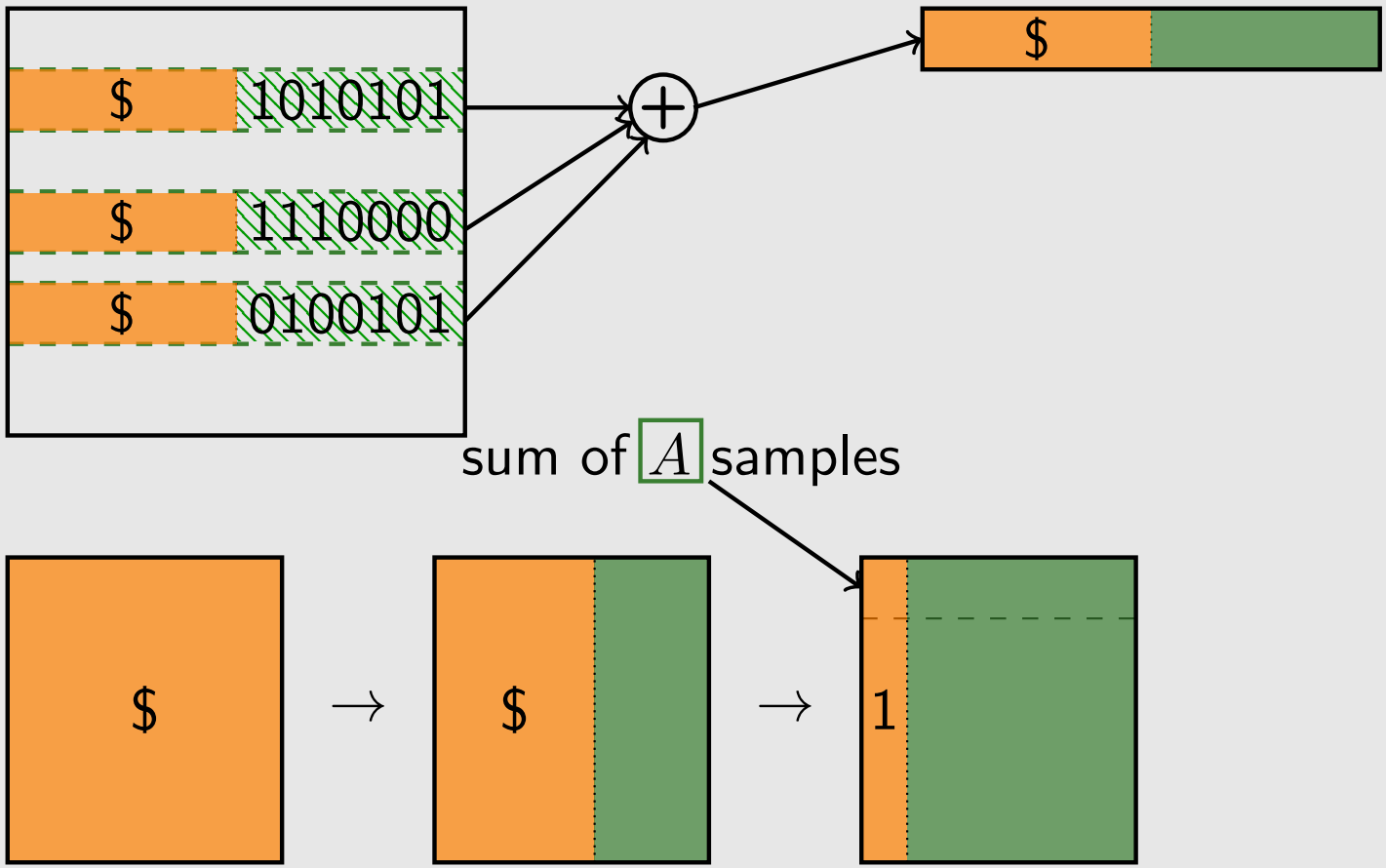
Not just a memory reduction technique

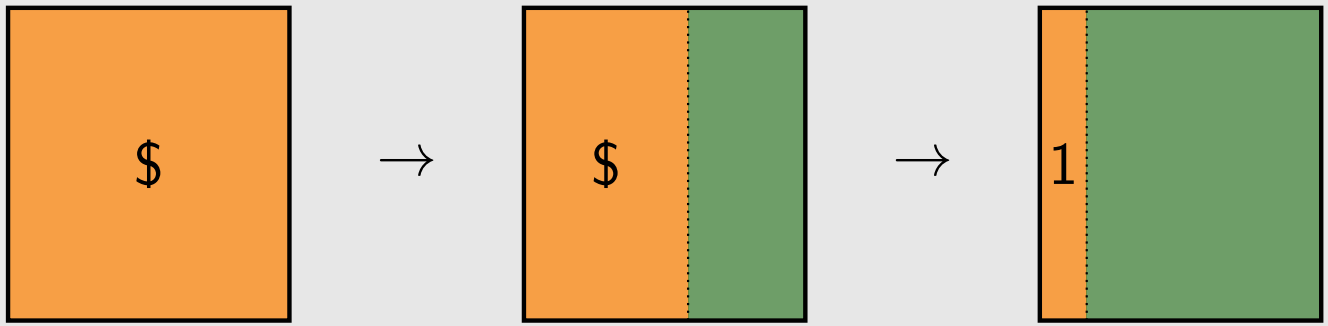


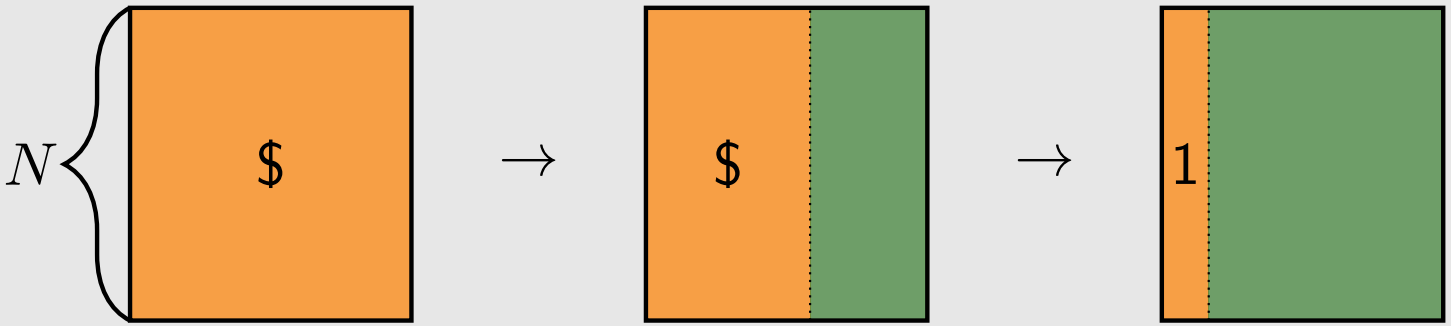
Not just a memory reduction technique

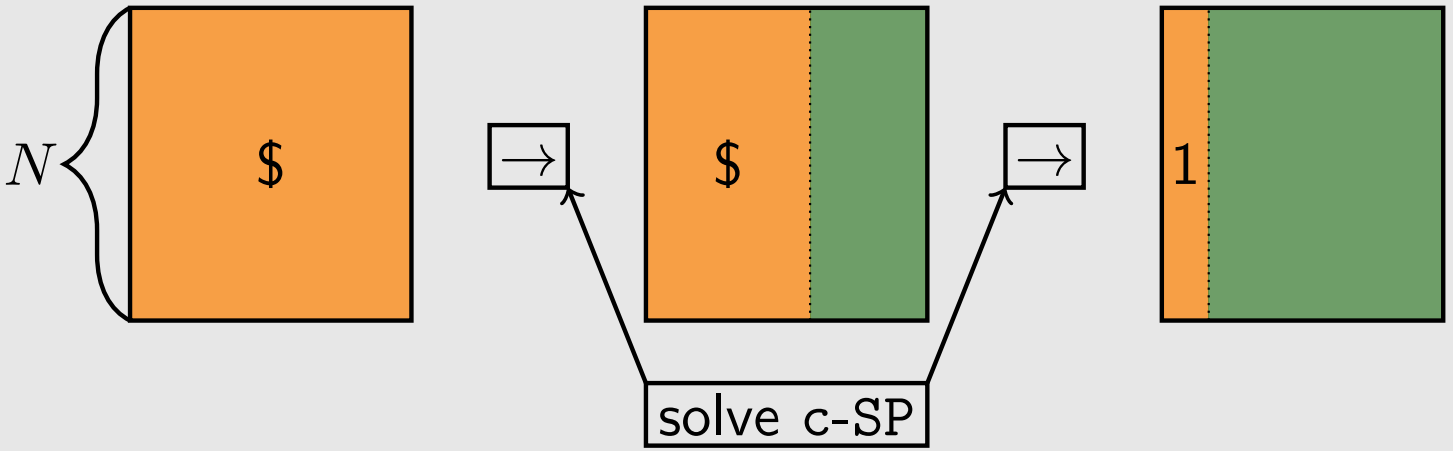


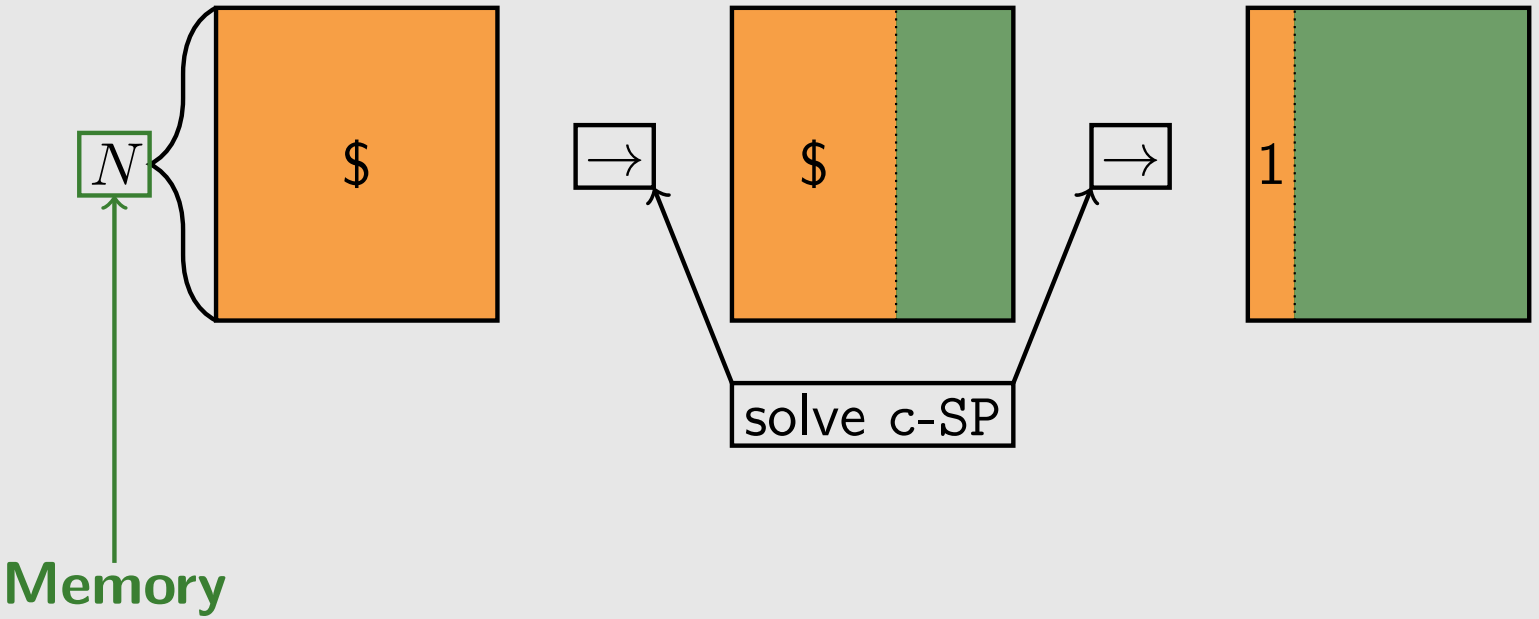
Not just a memory reduction technique

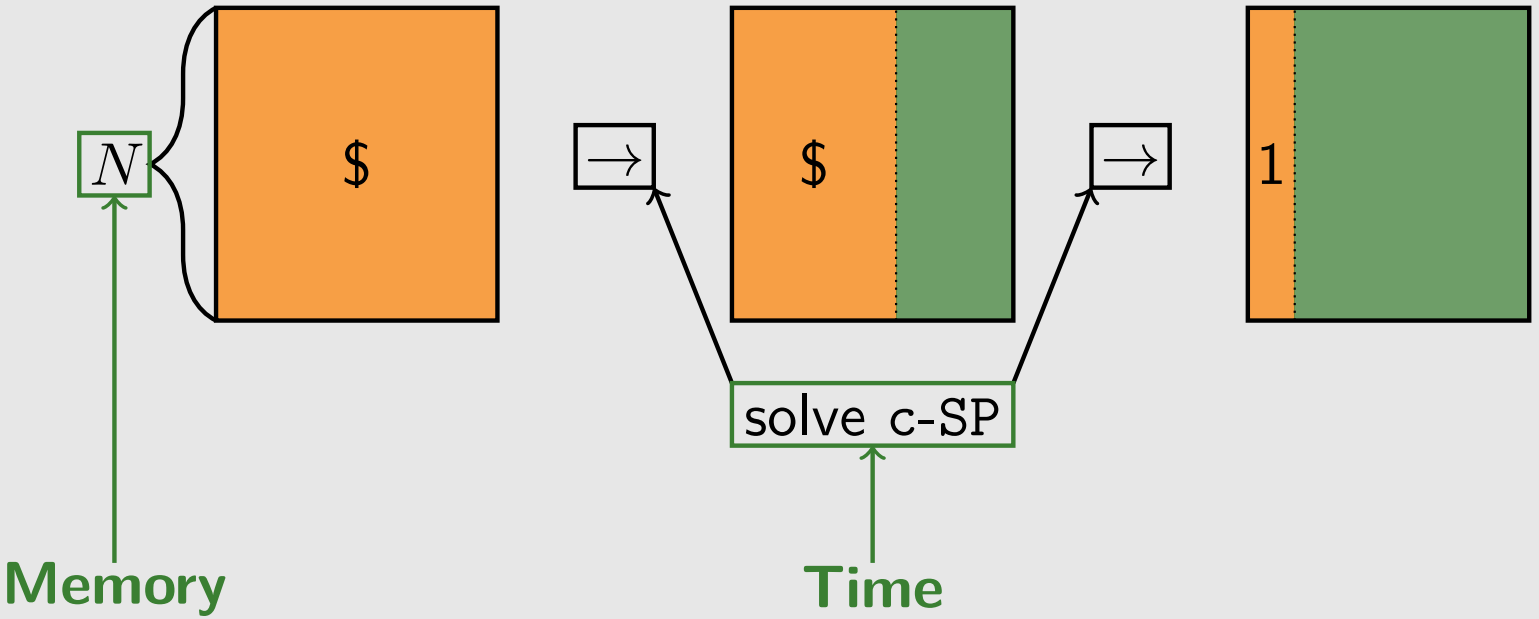


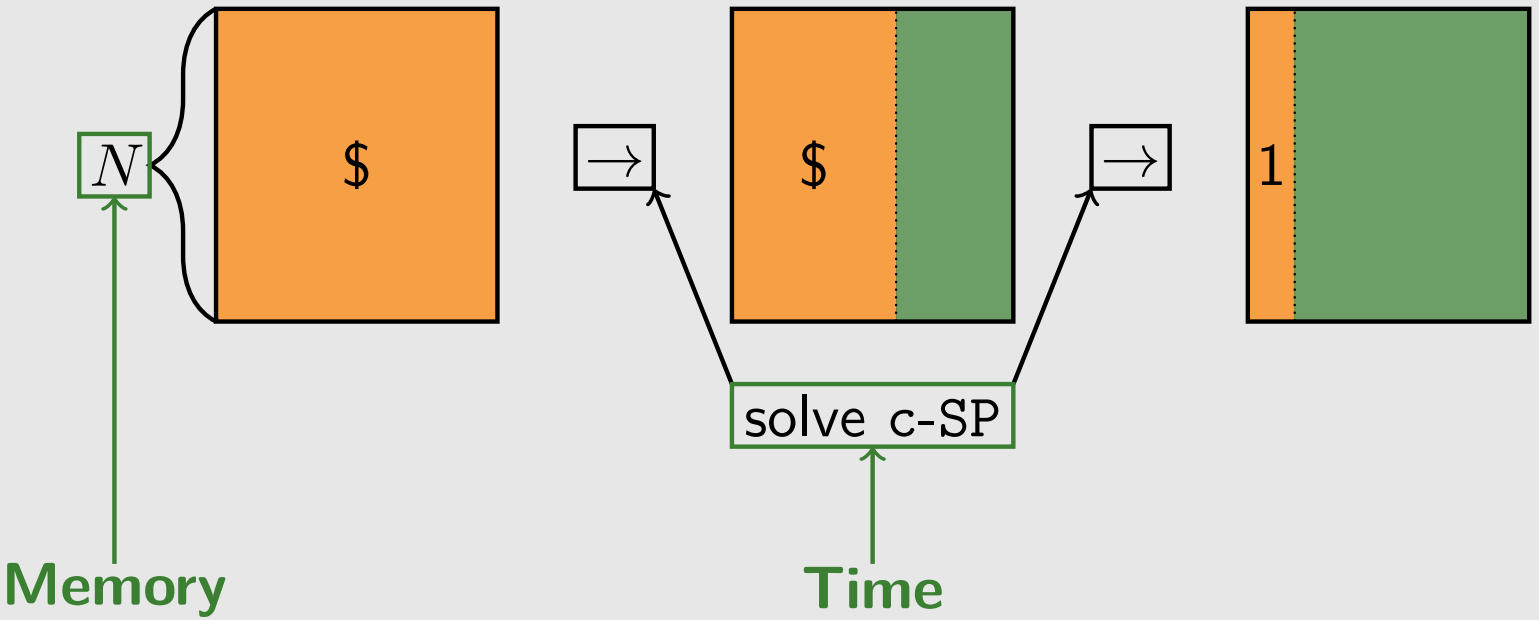

$
$
1


N
$
$
1


N
$
$
1
solve c-SP


N
$
$
1
solve c-SP
Memory


N
$
$
1
solve c-SP
Memory
Time



c-sum-BKW Theorem

LPN can be solved in time T and memory/samples M:

$$T = T_{\text{c-SP}} \text{ and } M = N$$

A naive Algorithm solving the c-sum-Problem

c-sum-naive Algorithm

Input : list L of size N
foreach $(c-1)$-*sum* x *of* L **do**
  if $x \in L$ **then**
    save c-sum

A naive Algorithm solving the c-sum-Problem

c-sum-naive Algorithm

Input : list L of size N
foreach $(c-1)$-*sum* x *of* L **do**
  if $x \in L$ **then**
    save c-sum

c-sum-naive Theorem

c-`sum-naive` solves the c-sum-Problem in time T and Memory M:

$$T = N^{c-1} \text{ and } M = N$$

First TMTO for BKW

c-sum-naive-BKW Theorem

`c-sum-naive-BKW` solves LPN in time T and memory/samples M:

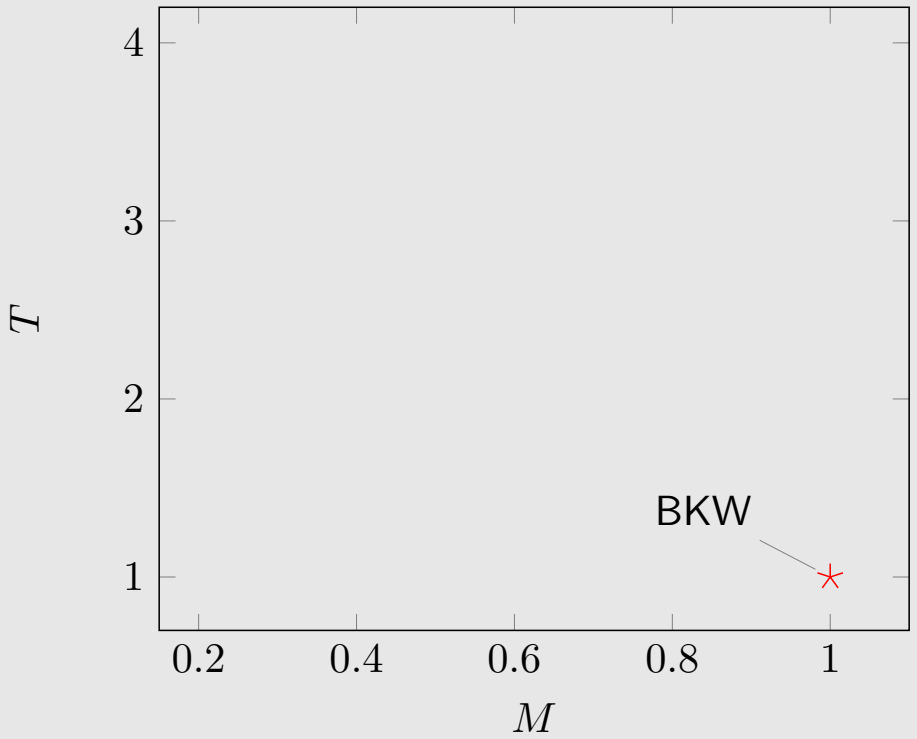
$$\log T = \log c \cdot \frac{k}{\log k} \quad \text{and} \quad \log M = \frac{\log c}{c-1} \cdot \frac{k}{\log k}$$

First TMTO for BKW

c-sum-naive-BKW Theorem

`c-sum-naive-BKW` solves LPN in time T and memory/samples M:

$$\log T = \log c \cdot \frac{k}{\log k} \quad \text{and} \quad \log M = \frac{\log c}{c-1} \cdot \frac{k}{\log k}$$

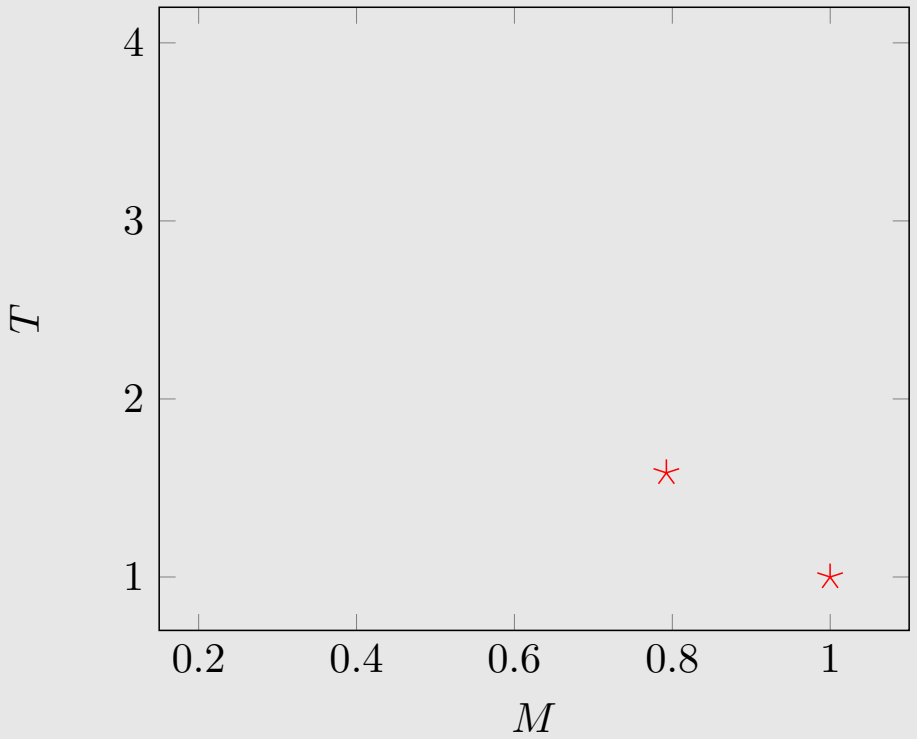


First TMTO for BKW

c-sum-naive-BKW Theorem

`c-sum-naive-BKW` solves LPN in time T and memory/samples M:

$$\log T = \log c \cdot \frac{k}{\log k} \quad \text{and} \quad \log M = \frac{\log c}{c-1} \cdot \frac{k}{\log k}$$

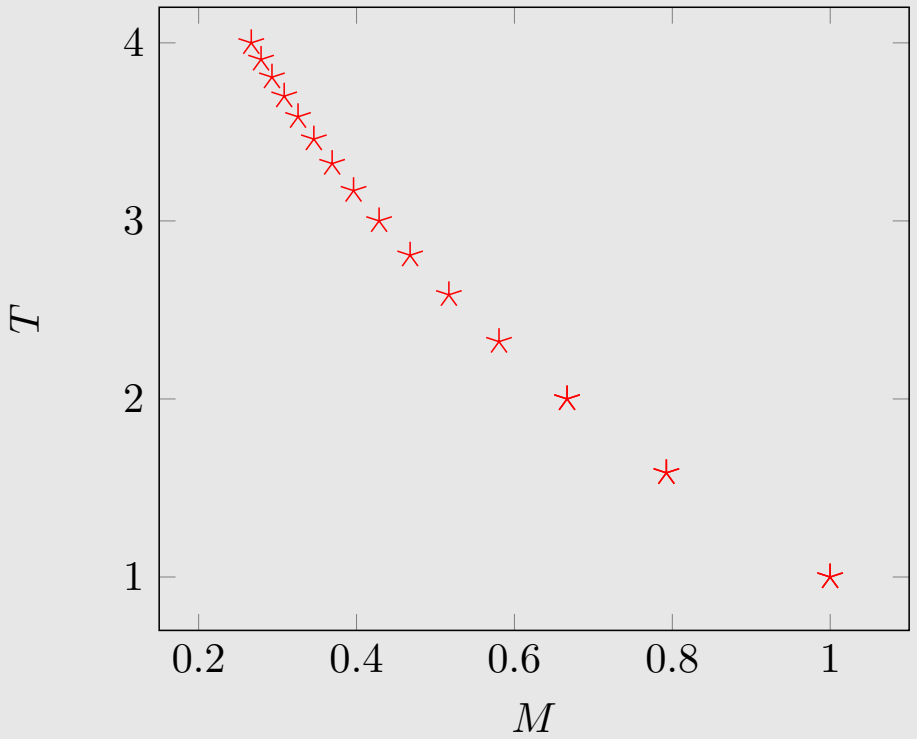


First TMTO for BKW

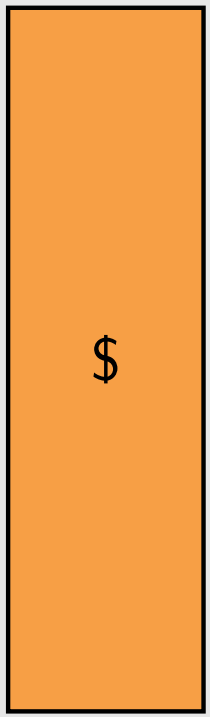
c-sum-naive-BKW Theorem

`c-sum-naive-BKW` solves LPN in time T and memory/samples M:

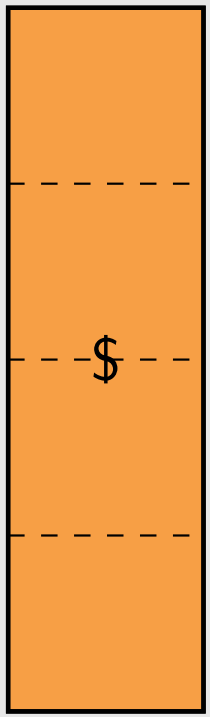
$$\log T = \log c \cdot \frac{k}{\log k} \quad \text{and} \quad \log M = \frac{\log c}{c-1} \cdot \frac{k}{\log k}$$



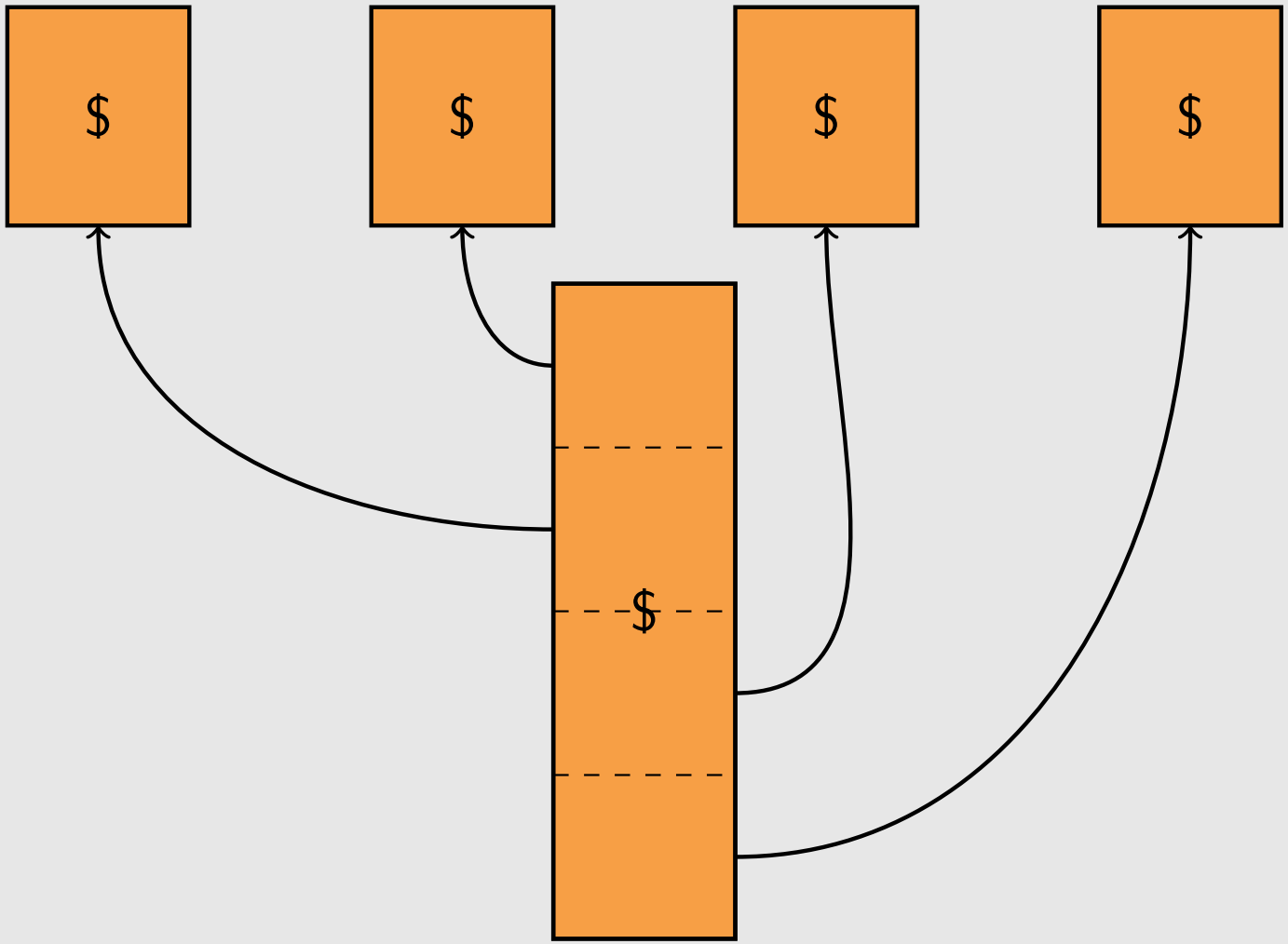
The Idea of Schroeppel-Shamir



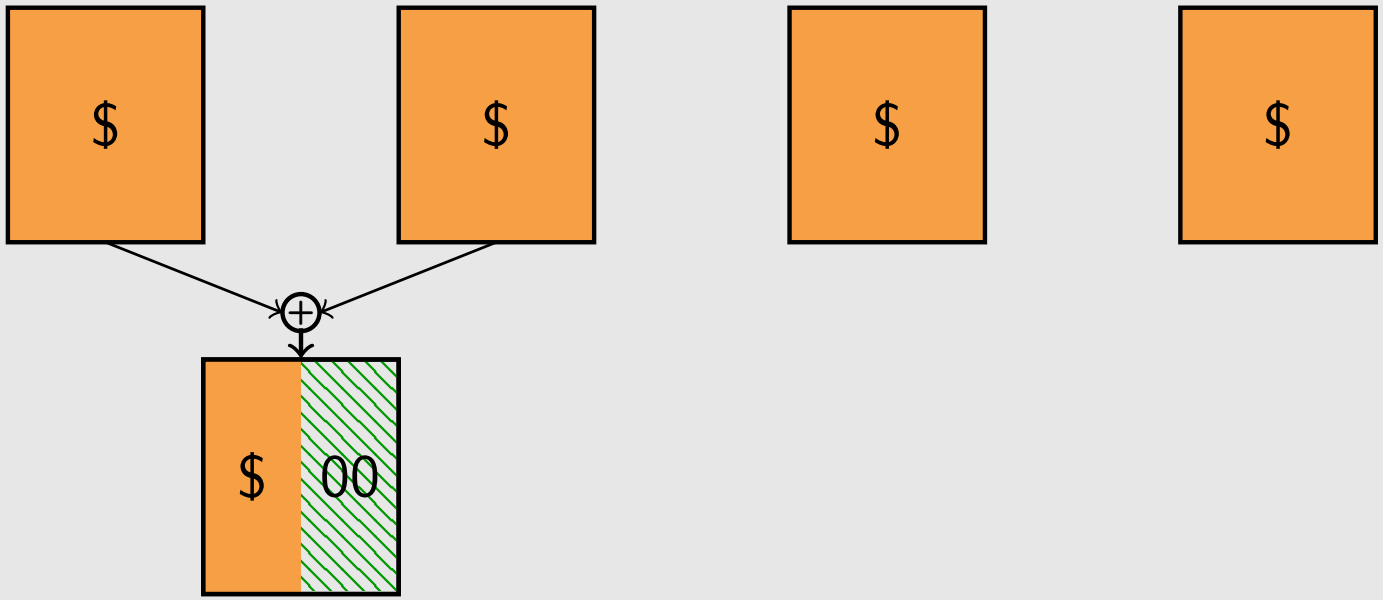
The Idea of Schroeppel-Shamir



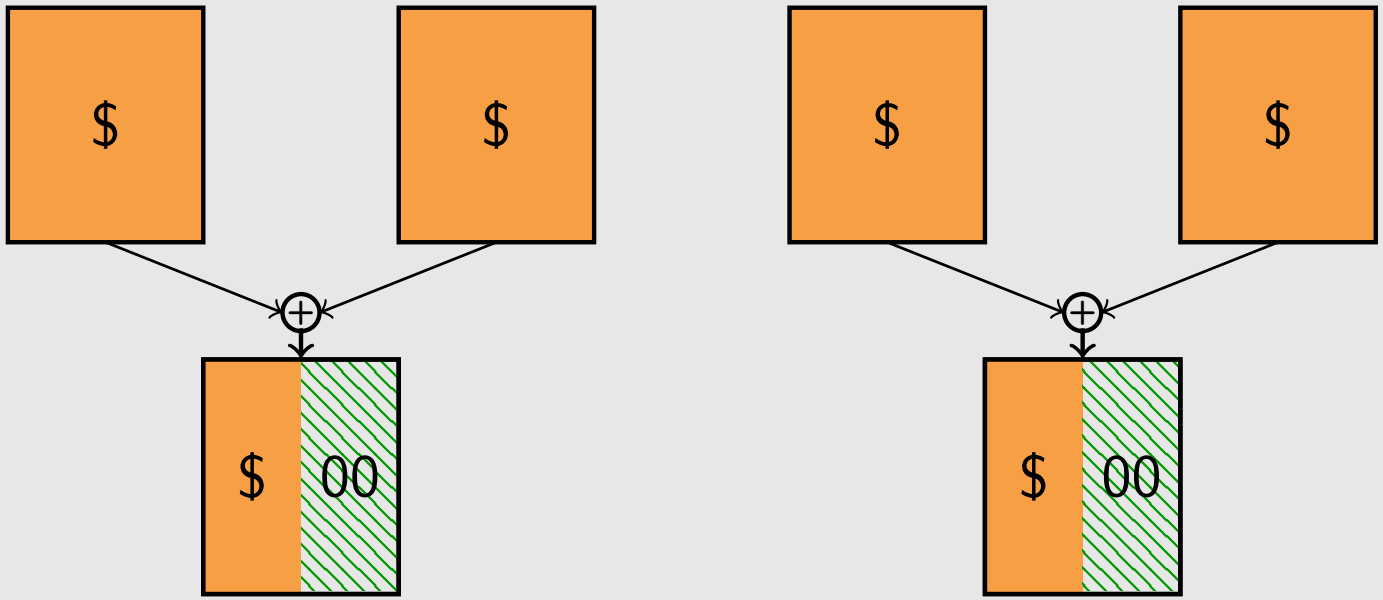
The Idea of Schroeppel-Shamir



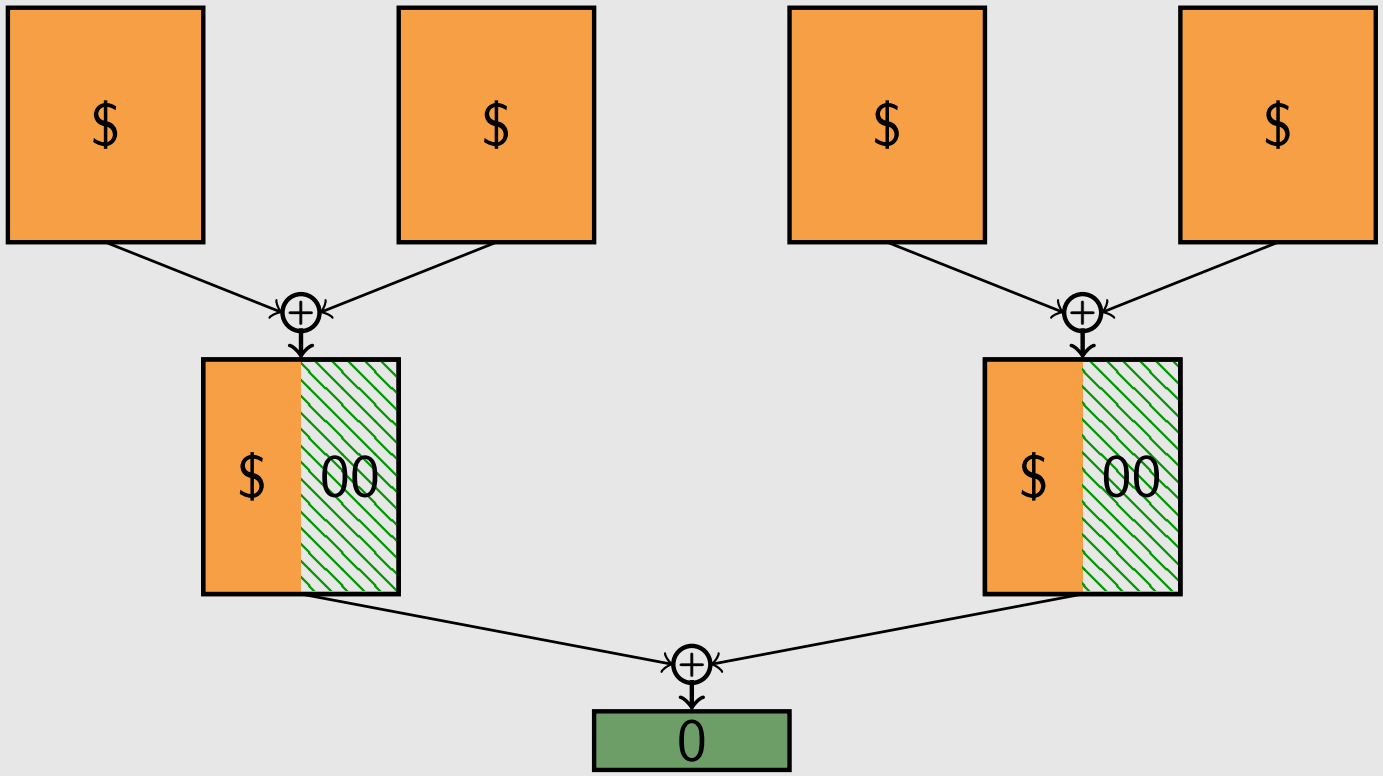
The Idea of Schroeppel-Shamir



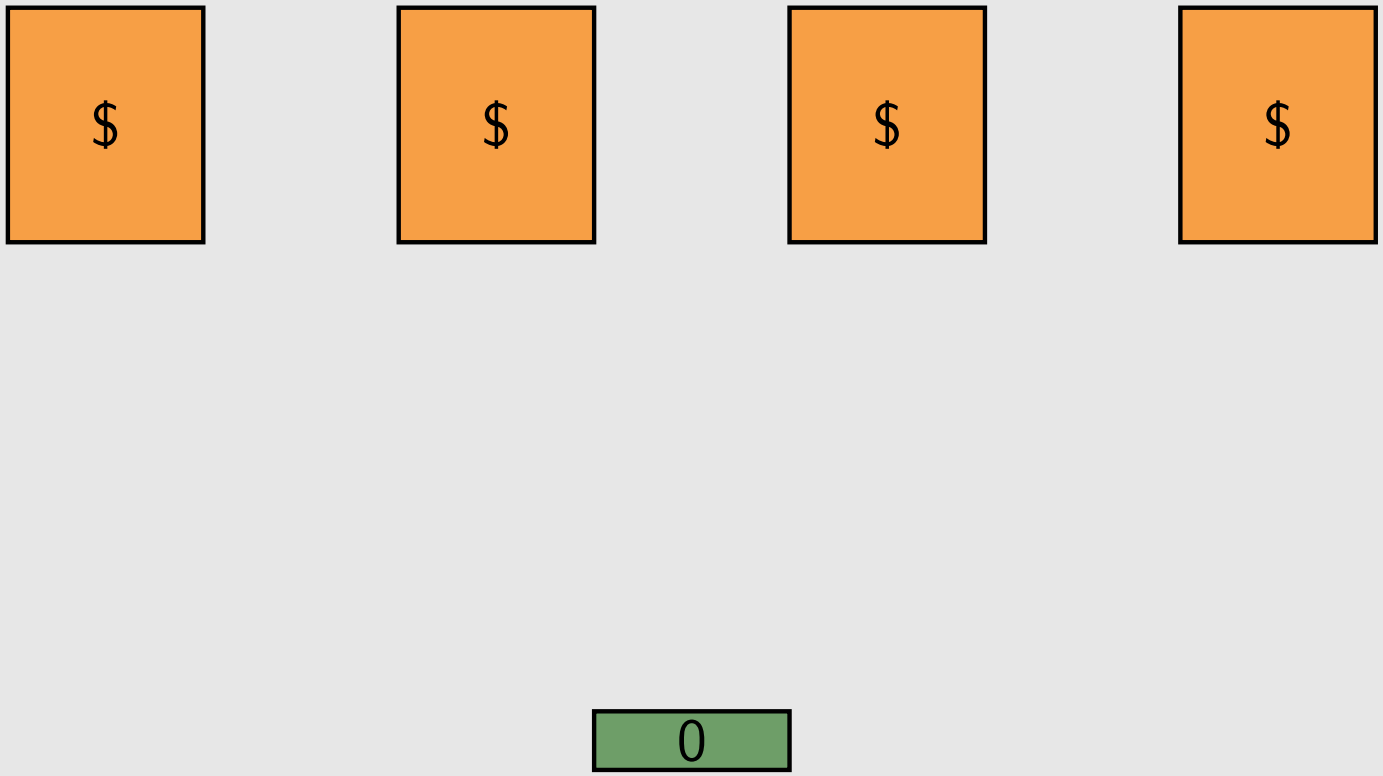
The Idea of Schroeppel-Shamir



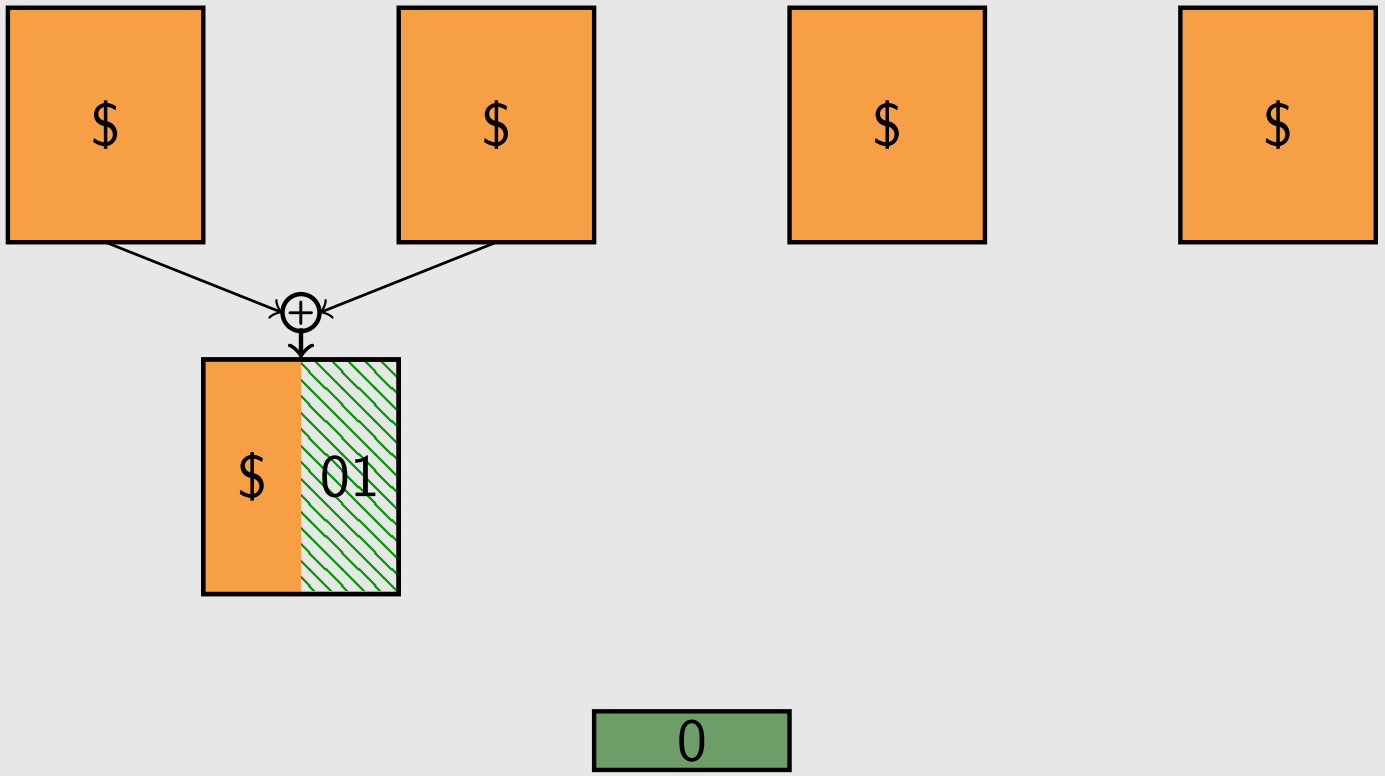
The Idea of Schroeppel-Shamir



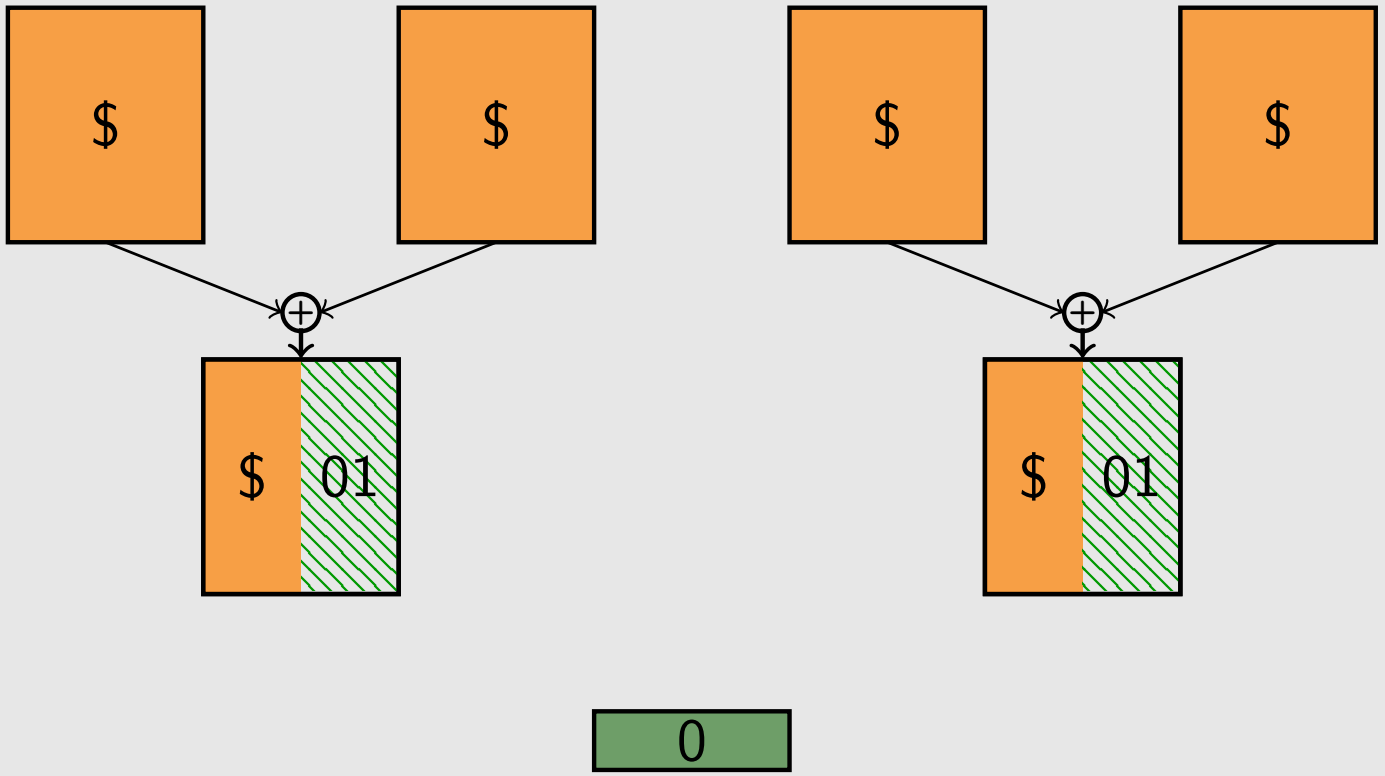
The Idea of Schroeppel-Shamir



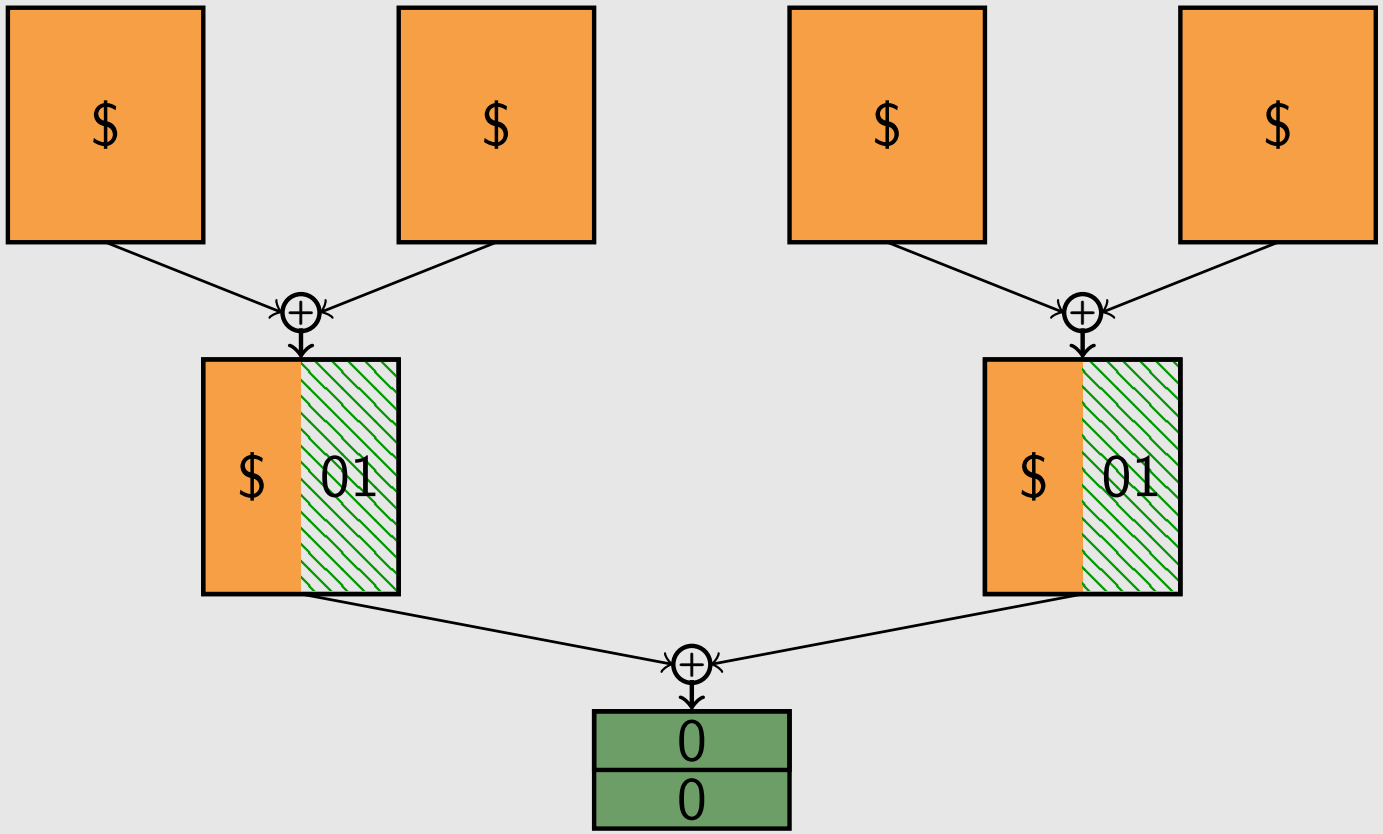
The Idea of Schroeppel-Shamir



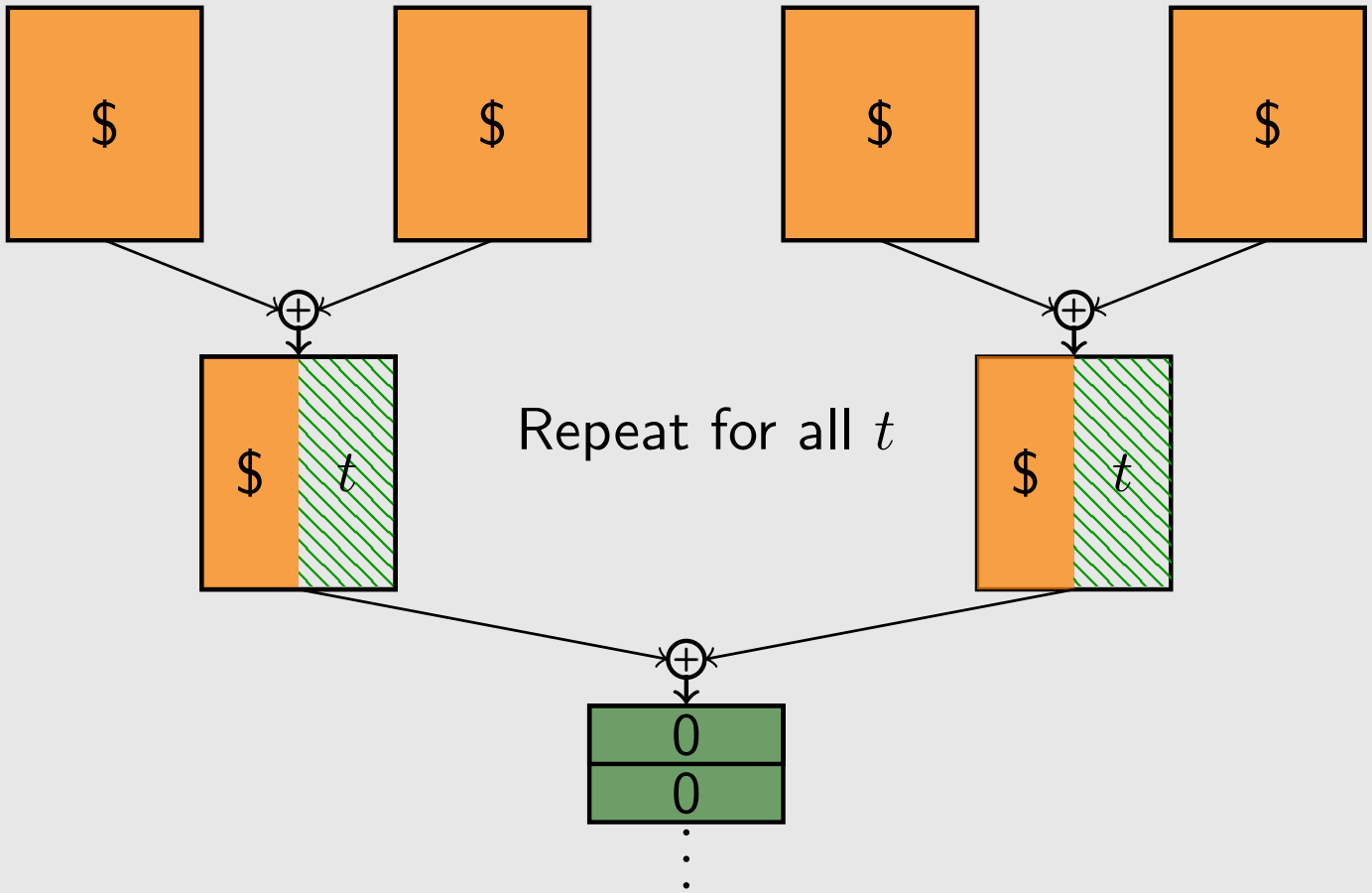
The Idea of Schroeppel-Shamir



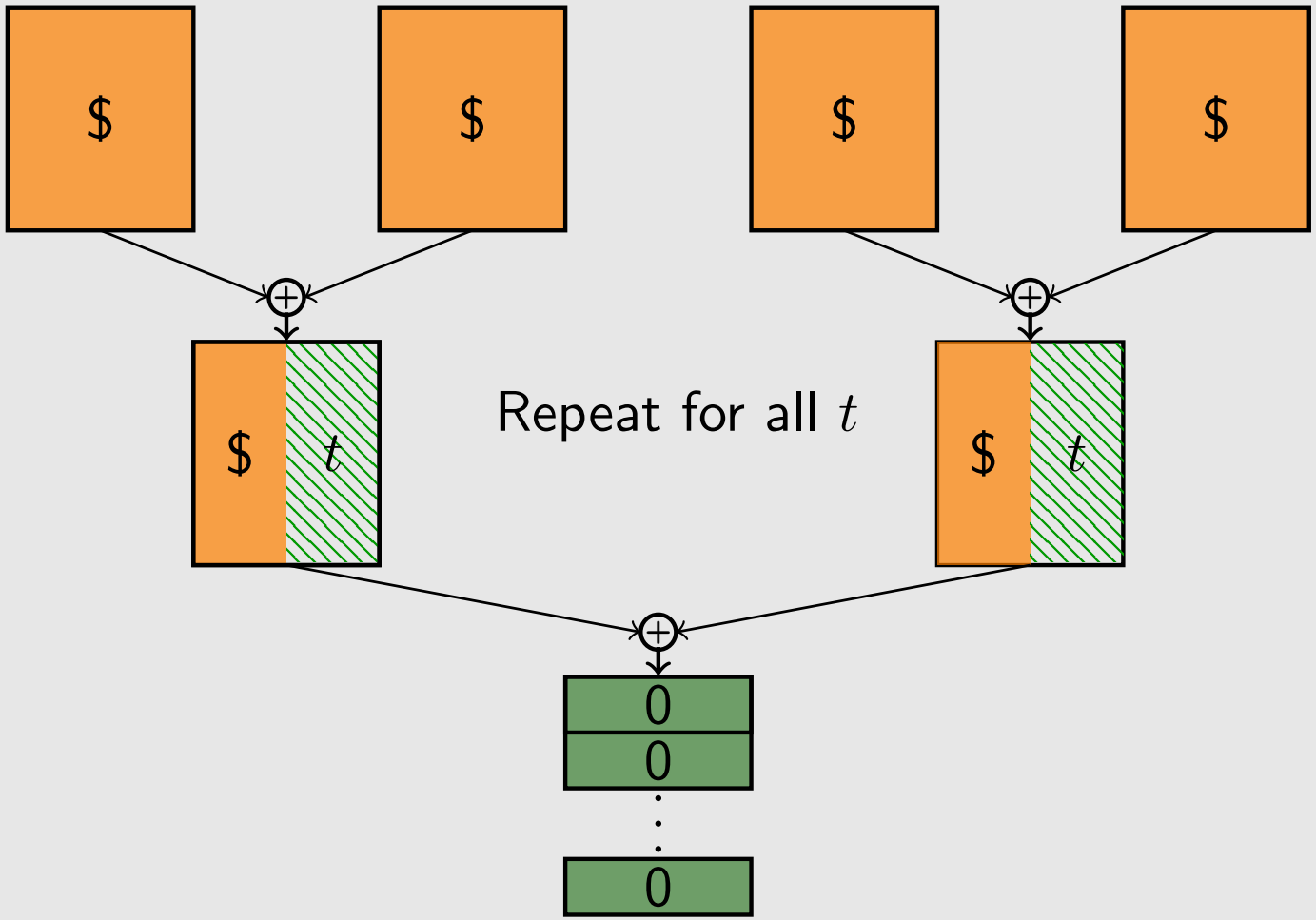
The Idea of Schroeppel-Shamir



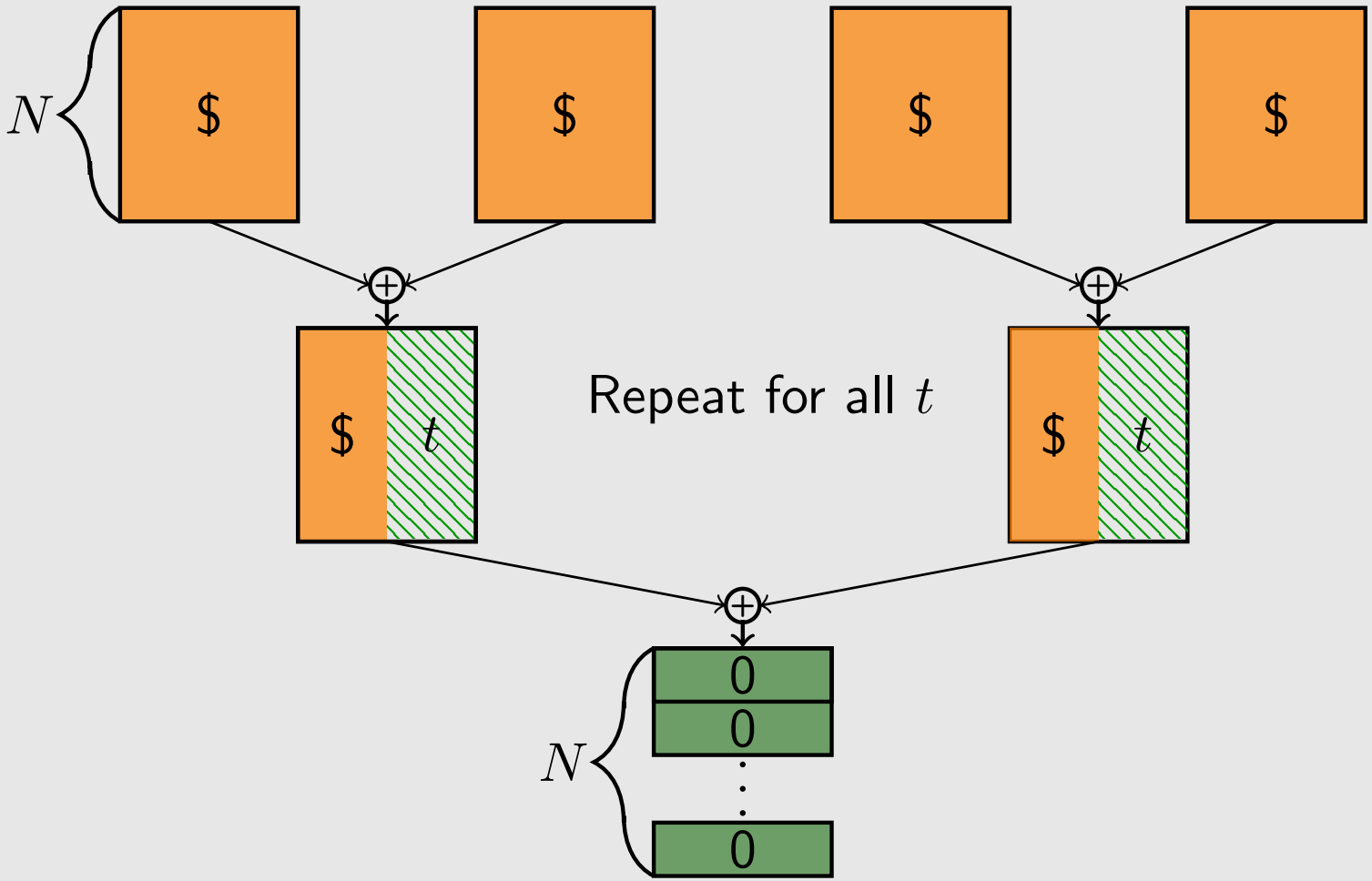
The Idea of Schroeppel-Shamir



The Idea of Schroeppel-Shamir



The Idea of Schroeppel-Shamir



Better TMTO for BKW

Schroeppel-Shamir Theorem [SS81, HGJ10]

Schroeppel-Shamir solves the 4-sum Problem in time T, memory M:

$$T = N^2 \text{ and } M = N$$

Better TMTO for BKW

Schroeppel-Shamir Theorem [SS81, HGJ10]

Schroeppel-Shamir solves the 4-sum Problem in time T, memory M:

$$T = N^2 \text{ and } M = N$$

naive: $T = N^3$

Better TMTO for BKW

Schroeppel-Shamir Theorem [SS81, HGJ10]

Schroeppel-Shamir solves the 4-sum Problem in time T, memory M:

$$T = N^2 \text{ and } M = N$$

naive: $T = N^3$

Dissection Theorem [DDKS12]

Dissection solves the c-sum Problem in time T and memory M:

$$T = N^{c-\sqrt{c}} \text{ and } M = N$$

Better TMTO for BKW

Schroeppel-Shamir Theorem [SS81, HGJ10]

Schroeppel-Shamir solves the 4-sum Problem in time T, memory M:

$$T = N^2 \text{ and } M = N$$

naive: $T = N^3$

Dissection Theorem [DDKS12]

Dissection solves the c-sum Problem in time T and memory M:

$$T = N^{c-\sqrt{c}} \text{ and } M = N$$

naive: $T = N^{c-1}$

Better TMTO for BKW

Dissection-BKW Theorem

`Dissection-BKW` solves LPN in time T and memory/samples M:

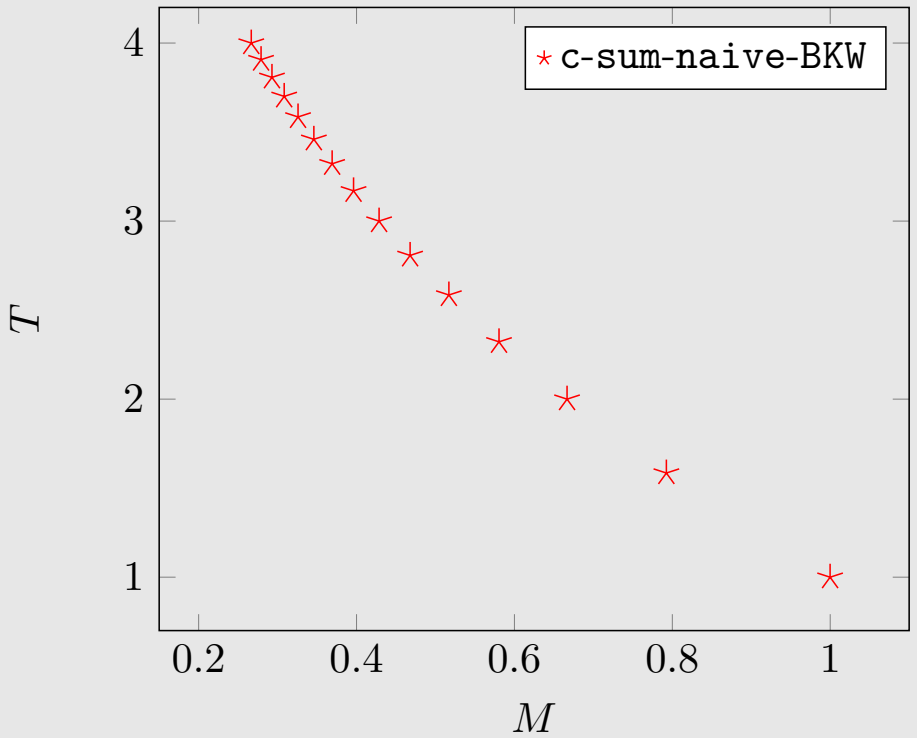
$$\log T = \left(\mathbf{1 - 2/\sqrt{c}}\right) \cdot \log c \cdot \frac{k}{\log k} \quad \text{and} \quad \log M = \frac{\log c}{c-1} \cdot \frac{k}{\log k}$$

Better TMTO for BKW

Dissection-BKW Theorem

`Dissection-BKW` solves LPN in time T and memory/samples M:

$$\log T = \left(\mathbf{1 - 2/\sqrt{c}}\right) \cdot \log c \cdot \frac{k}{\log k} \quad \text{and} \quad \log M = \frac{\log c}{c-1} \cdot \frac{k}{\log k}$$

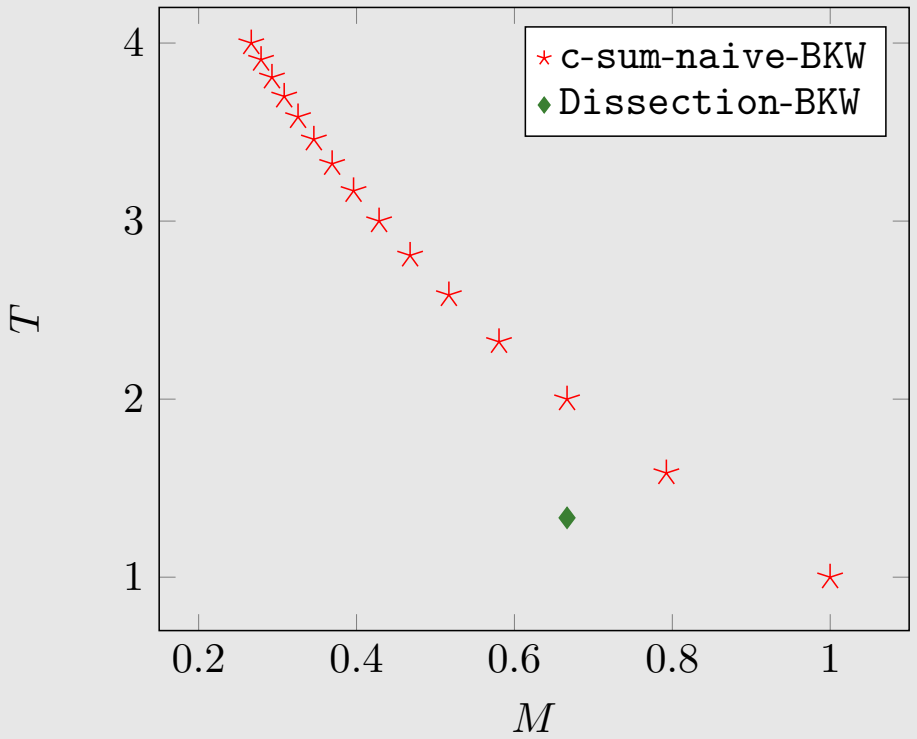


Better TMTO for BKW

Dissection-BKW Theorem

`Dissection-BKW` solves LPN in time T and memory/samples M:

$$\log T = \left(1 - 2/\sqrt{c}\right) \cdot \log c \cdot \frac{k}{\log k} \quad \text{and} \quad \log M = \frac{\log c}{c-1} \cdot \frac{k}{\log k}$$

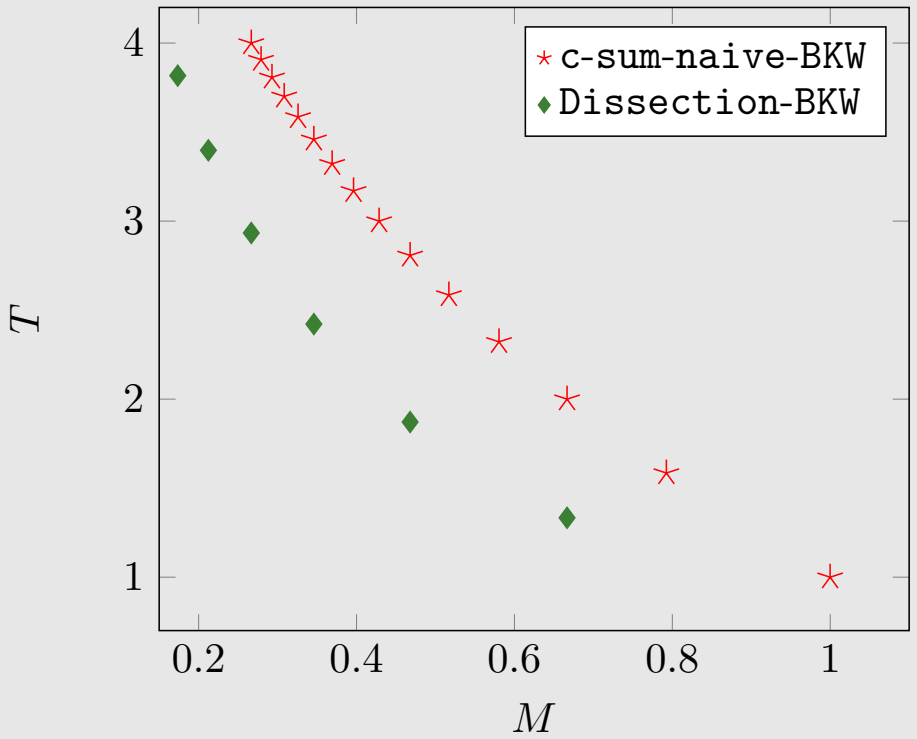


Better TMTO for BKW

Dissection-BKW Theorem

`Dissection-BKW` solves LPN in time T and memory/samples M:

$$\log T = \left(1 - 2/\sqrt{c}\right) \cdot \log c \cdot \frac{k}{\log k} \quad \text{and} \quad \log M = \frac{\log c}{c-1} \cdot \frac{k}{\log k}$$

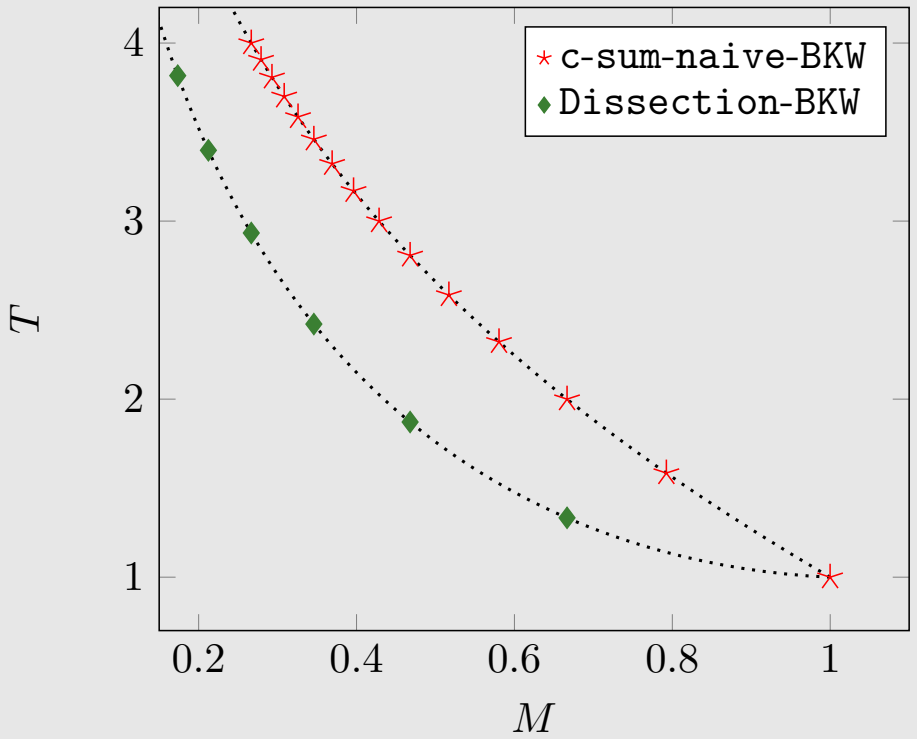


Better TMTO for BKW

Dissection-BKW Theorem

`Dissection-BKW` solves LPN in time T and memory/samples M:

$$\log T = \left(1 - 2/\sqrt{c}\right) \cdot \log c \cdot \frac{k}{\log k} \quad \text{and} \quad \log M = \frac{\log c}{c-1} \cdot \frac{k}{\log k}$$

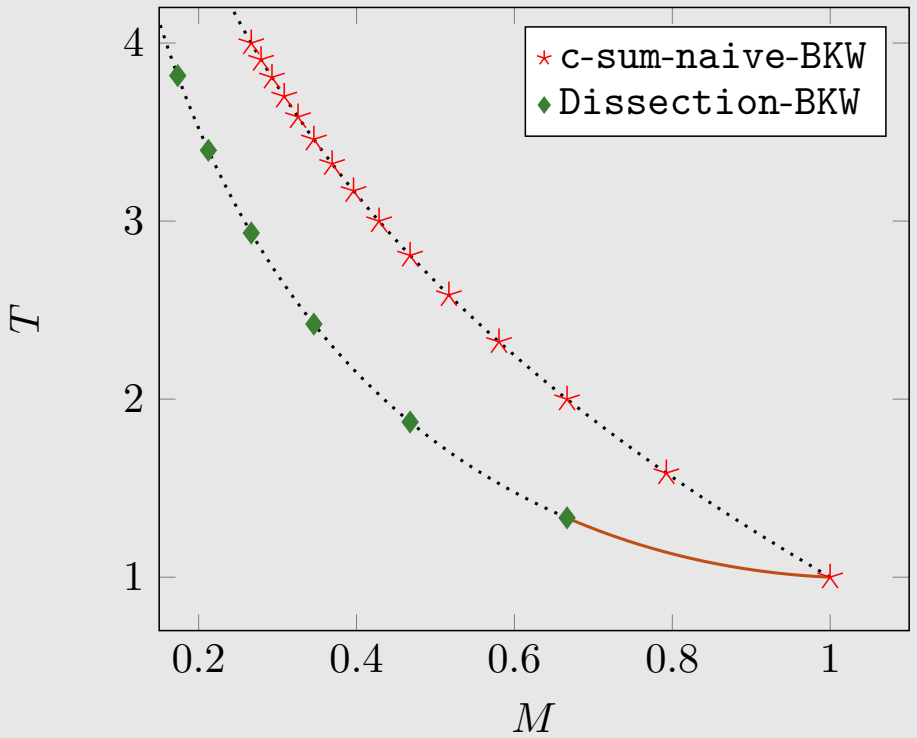


Better TMTO for BKW

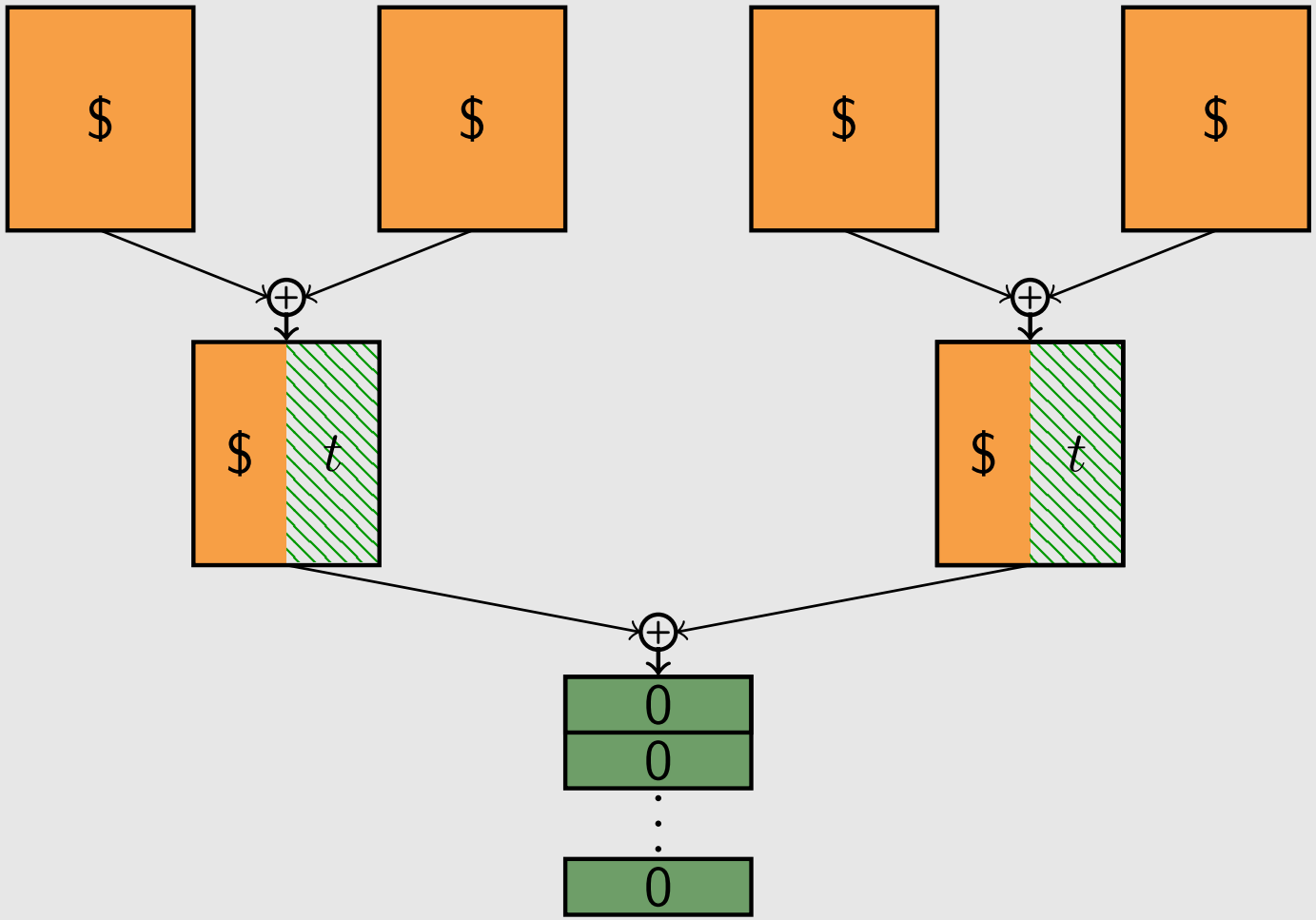
Dissection-BKW Theorem

`Dissection-BKW` solves LPN in time T and memory/samples M:

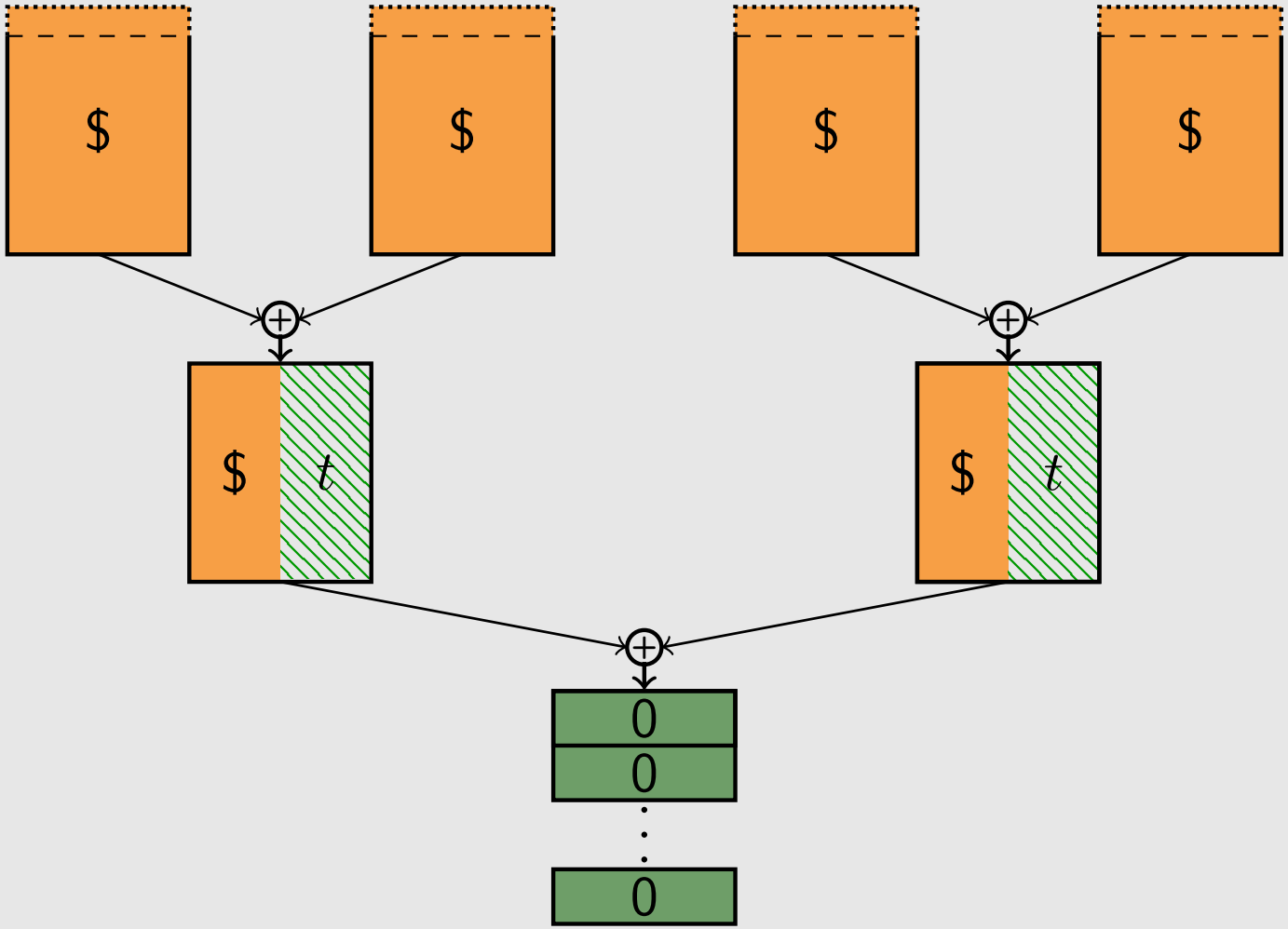
$$\log T = \left(1 - 2/\sqrt{c}\right) \cdot \log c \cdot \frac{k}{\log k} \quad \text{and} \quad \log M = \frac{\log c}{c-1} \cdot \frac{k}{\log k}$$



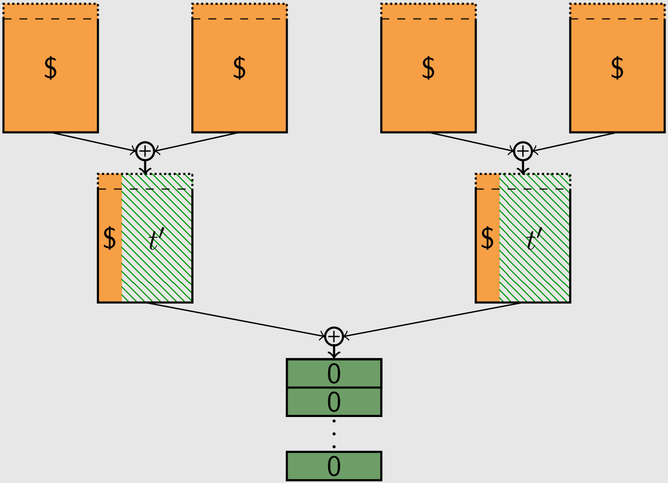
Tailored Schroeppel-Shamir



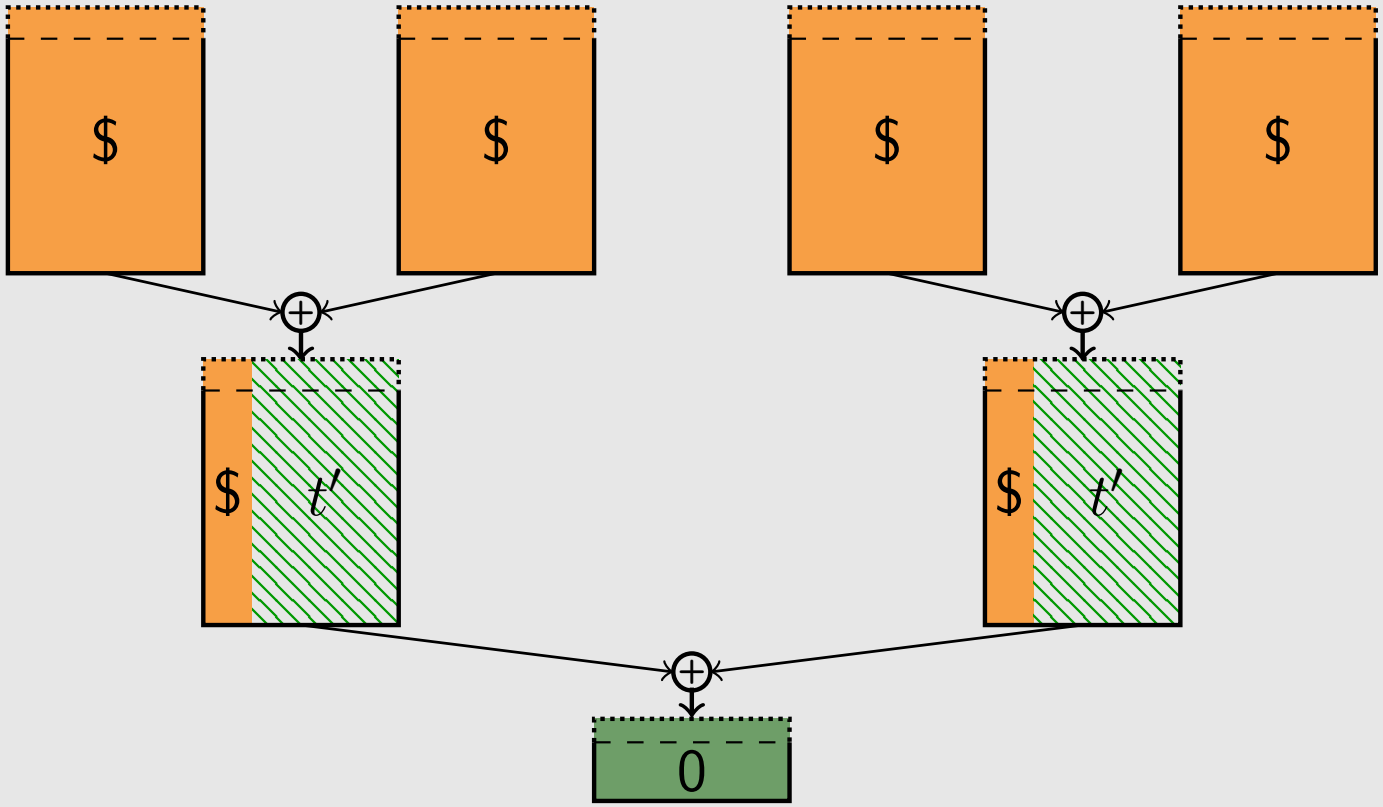
Tailored Schroeppel-Shamir



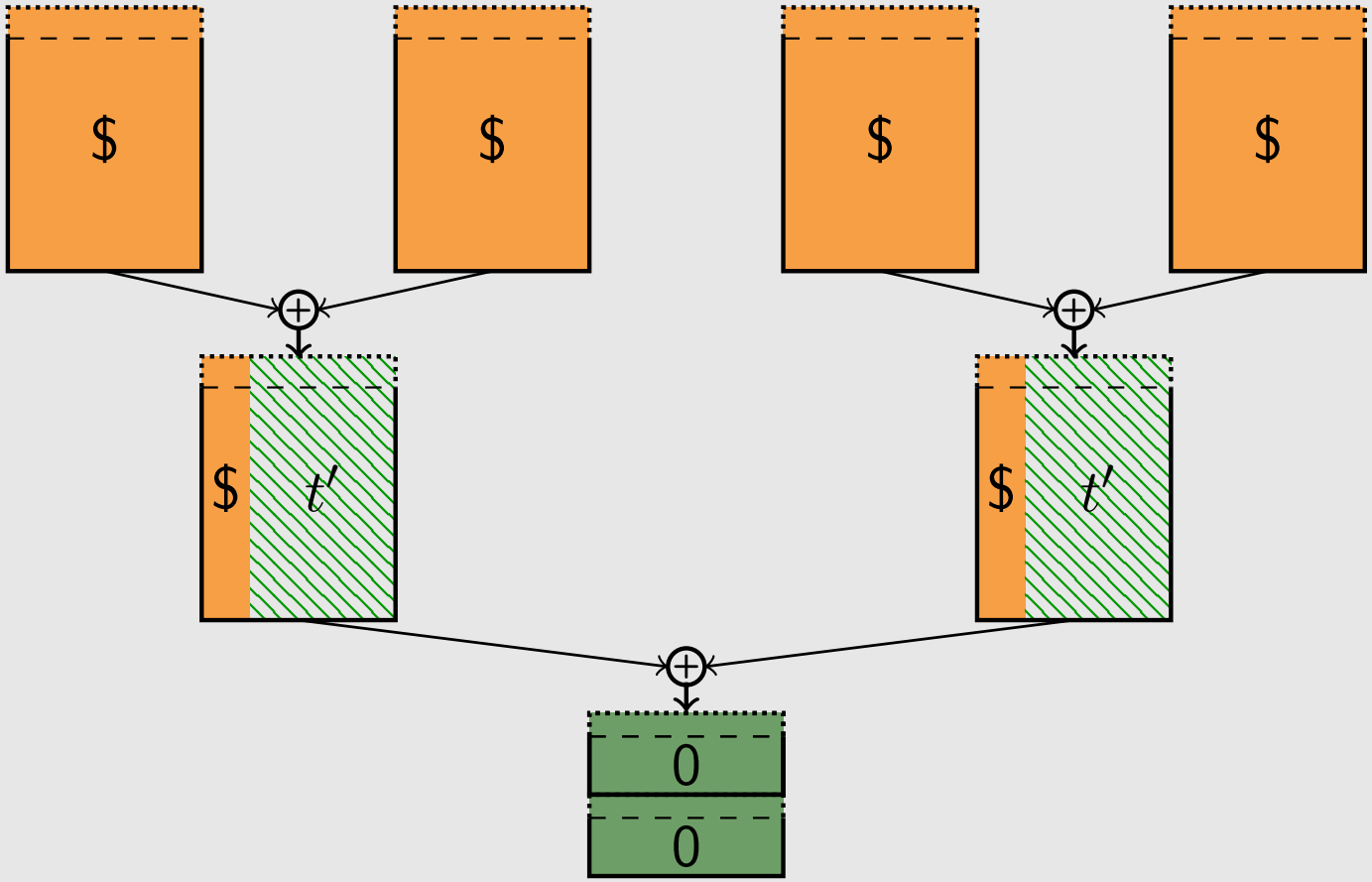
Tailored Schroeppel-Shamir



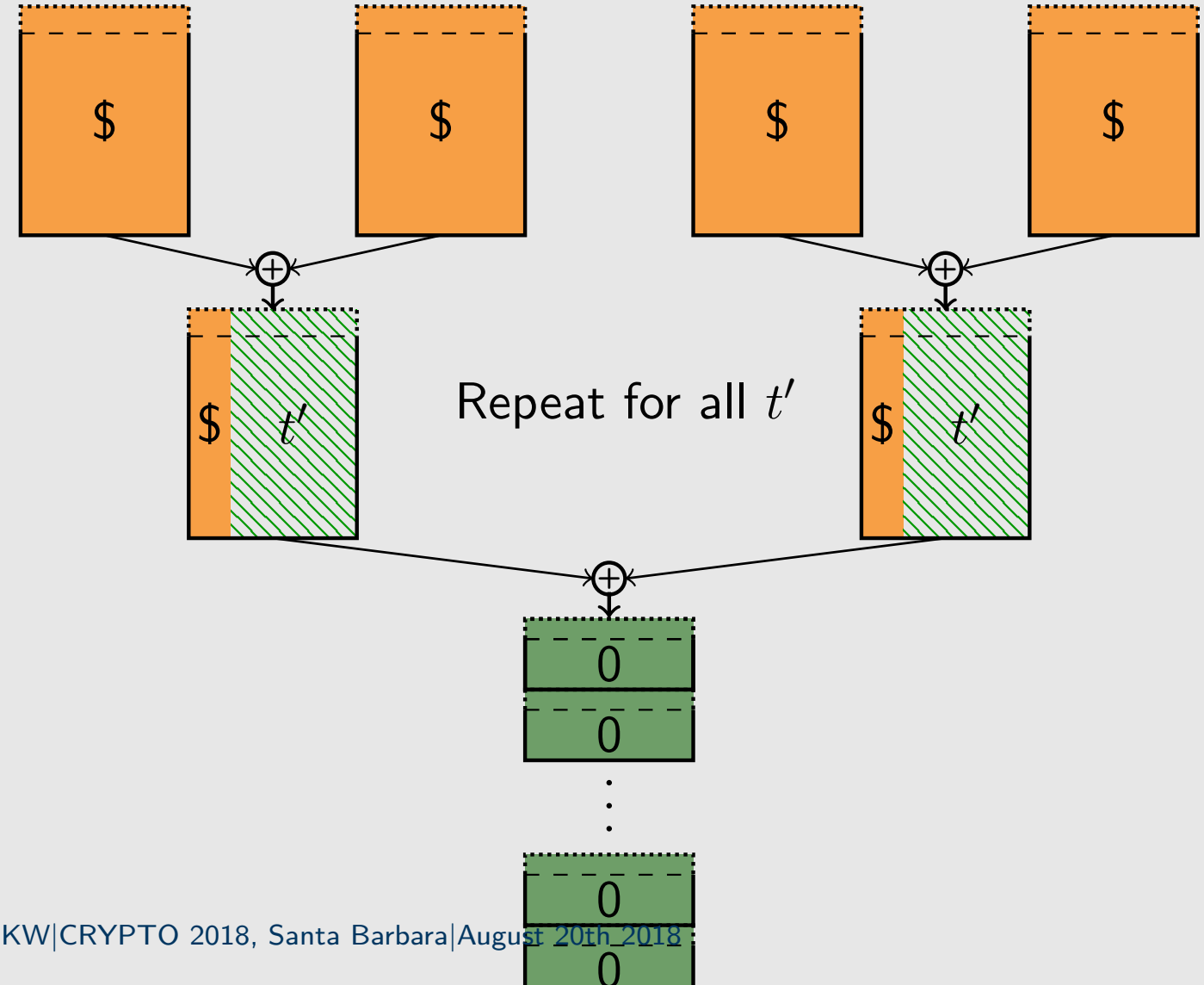
Tailored Schroeppel-Shamir



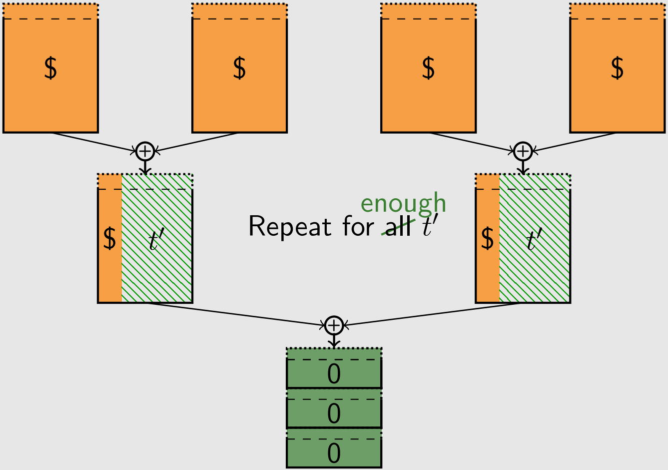
Tailored Schroeppel-Shamir



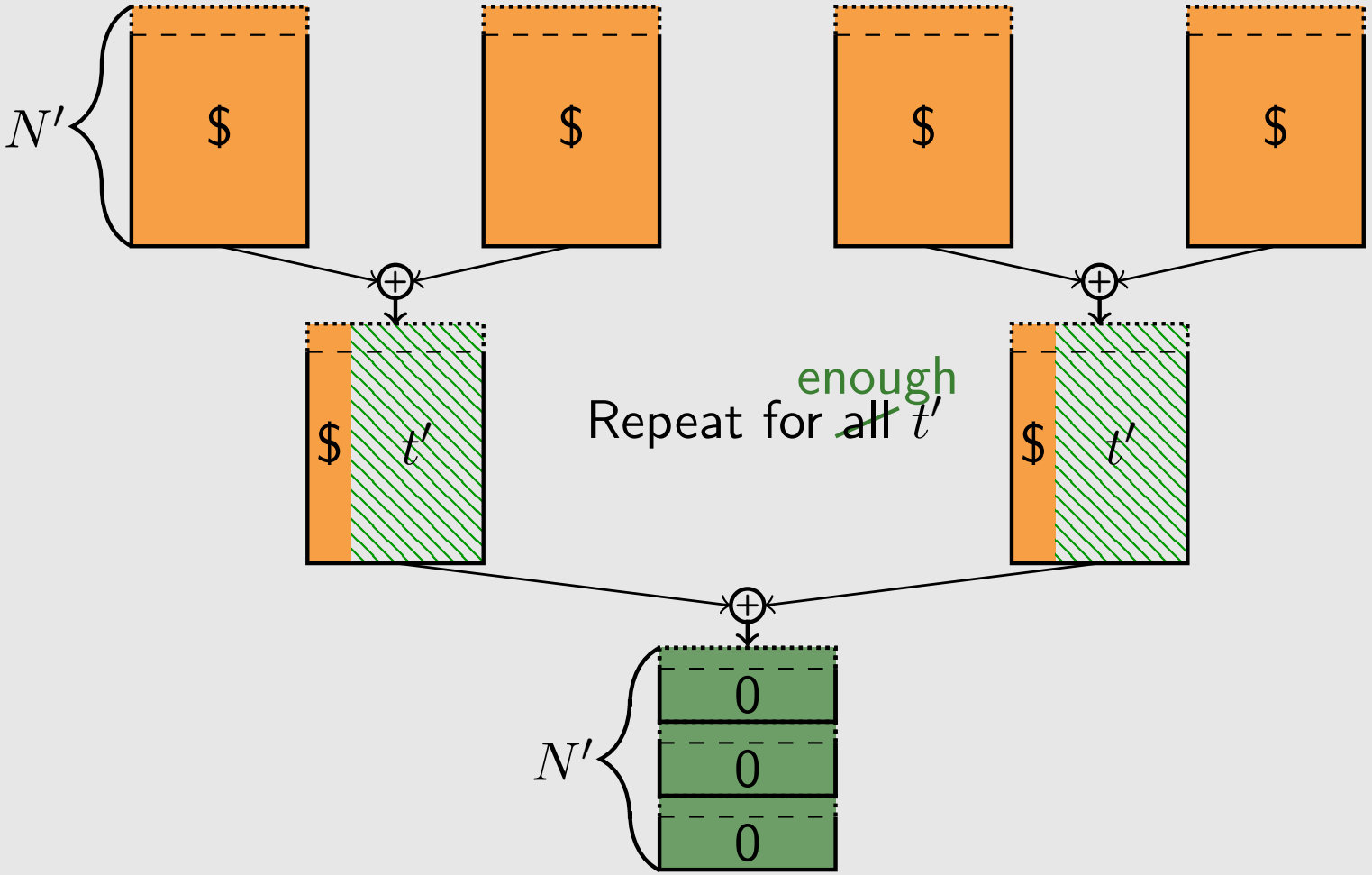
Tailored Schroeppel-Shamir



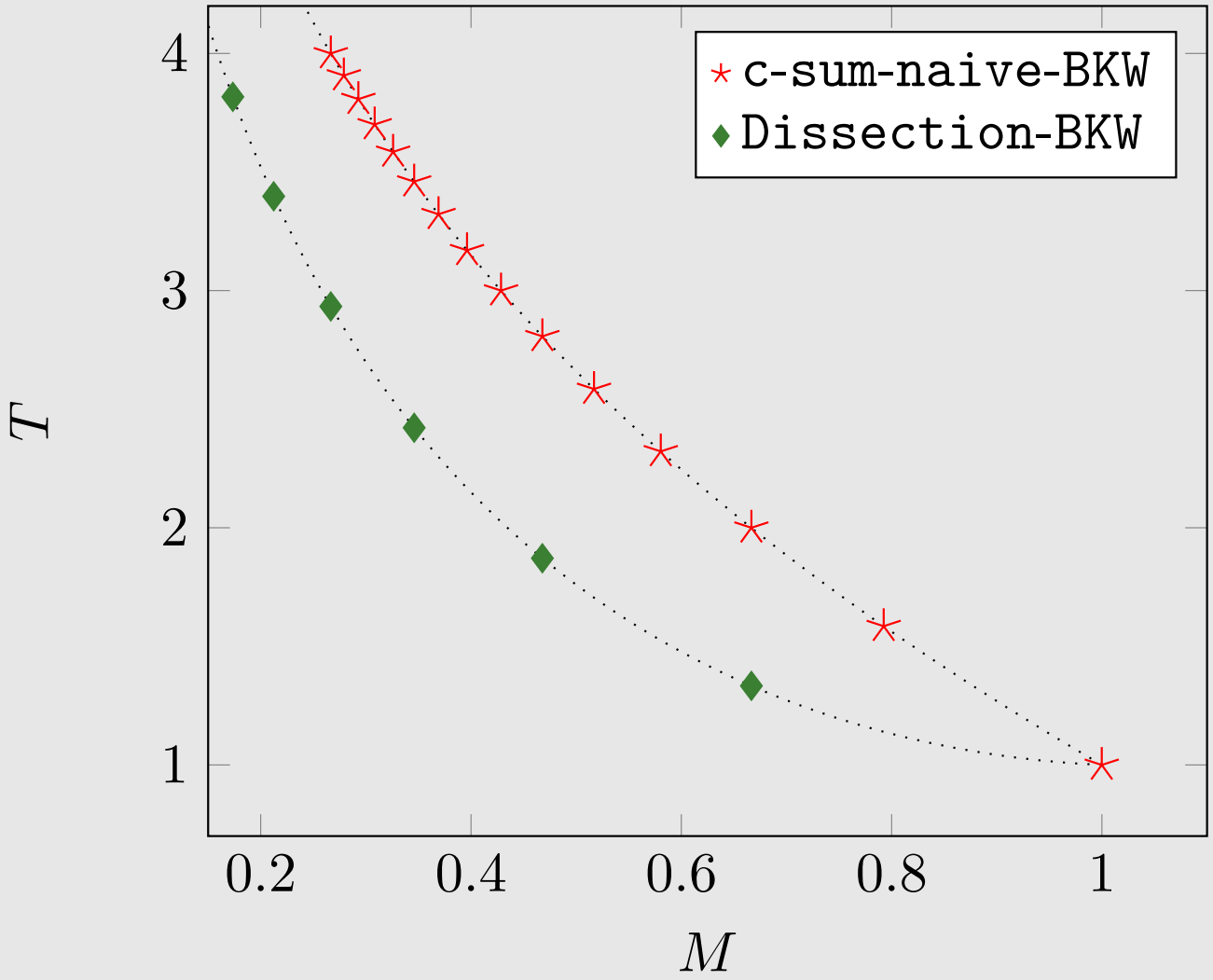
Tailored Schroeppel-Shamir



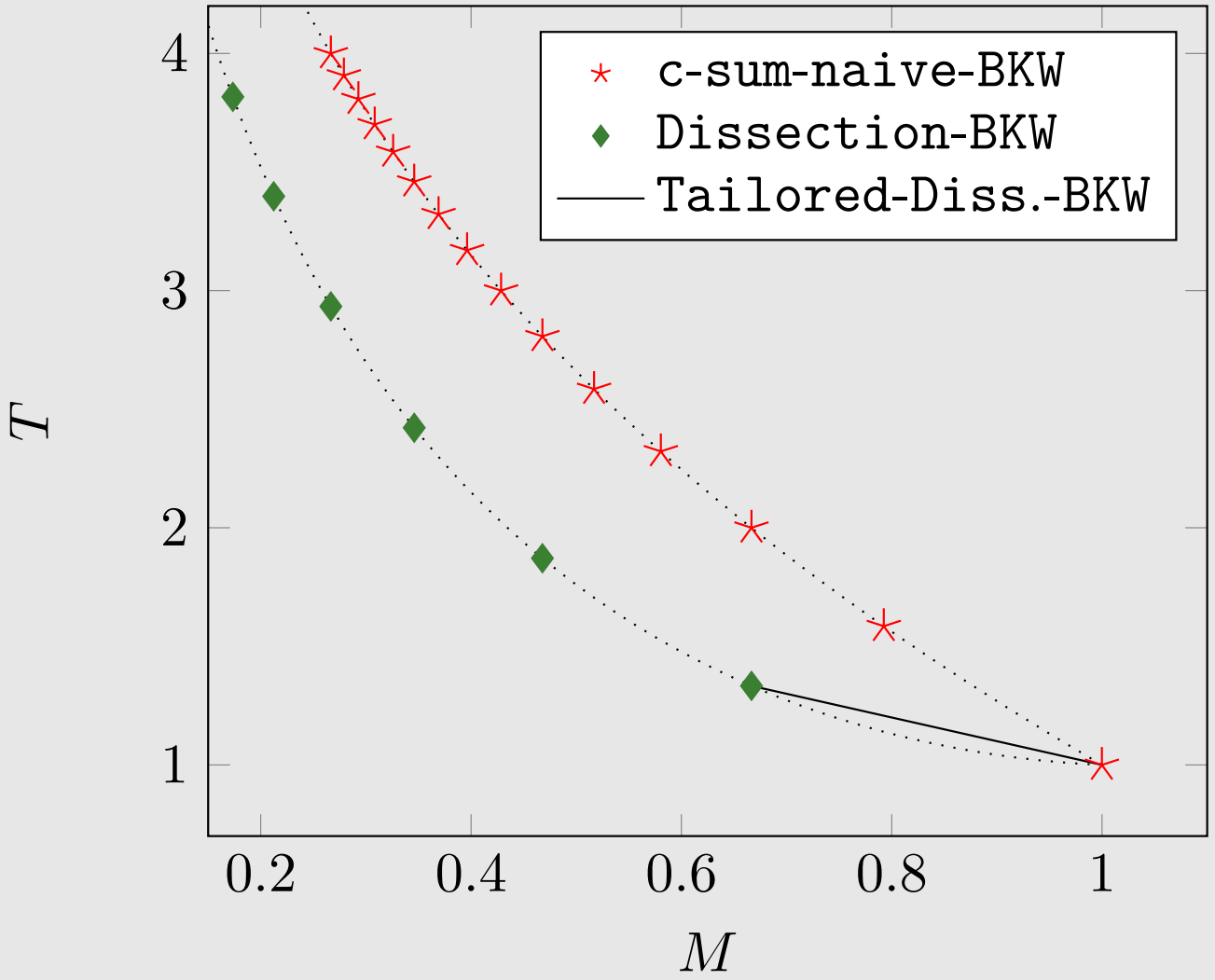
Tailored Schroeppel-Shamir



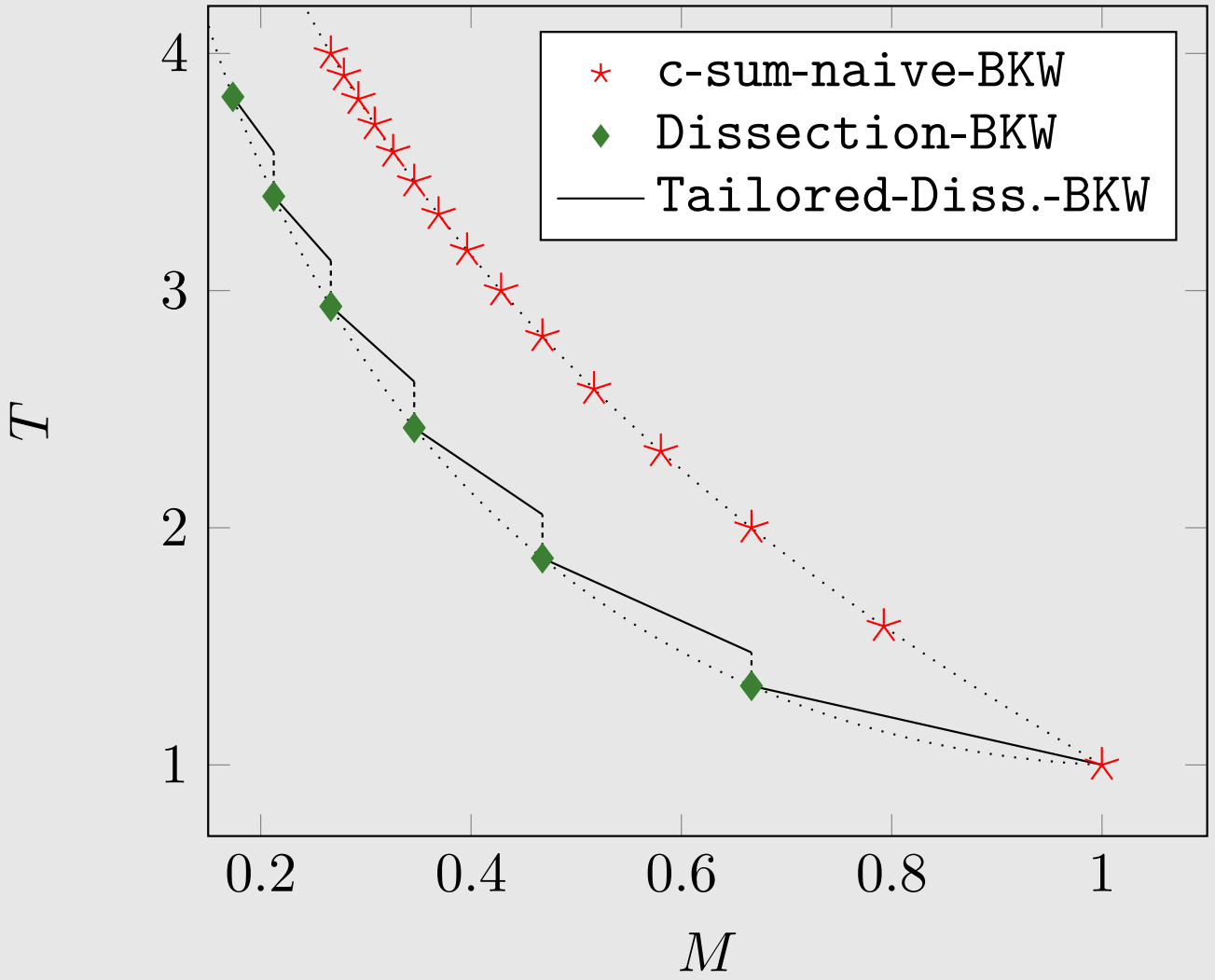
A continous TMTO via tailored Dissection



A continous TMTO via tailored Dissection



A continous TMTO via tailored Dissection



Results

- BKW-variant applicable for any given amount of memory

Results

- BKW-variant applicable for any given amount of memory
- Tailored Dissection

Results

- BKW-variant applicable for any given amount of memory
- Tailored Dissection
- Quantum tradeoff

Results

- BKW-variant applicable for any given amount of memory
- Tailored Dissection
- Quantum tradeoff
- Tradeoffs for LWE

Results

- BKW-variant applicable for any given amount of memory
- Tailored Dissection
- Quantum tradeoff
- Tradeoffs for LWE

References I

[Ale03] Michael Alekhnovich. More on average case vs approximation complexity. In *44th FOCS*, pages 298–307. IEEE Computer Society Press, October 2003.

[BKW00] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *32nd ACM STOC*, pages 435–440. ACM Press, May 2000.

[BTV16] Sonia Bogos, Florian Tramer, and Serge Vaudenay. On solving lpn using bkw and variants. *Cryptography and Communications*, 8(3):331–369, 2016.

[DDKS12] Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 719–740. Springer, Heidelberg, August 2012.

[DV13] Alexandre Duc and Serge Vaudenay. HELEN: A public-key cryptosystem based on the LPN and the decisional minimal distance problems. In Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien, editors, *AFRICACRYPT 13*, volume 7918 of *LNCS*, pages 107–126. Springer, Heidelberg, June 2013.

References II

[EKM17] Andre Esser, Robert Kübler, and Alexander May. LPN decoded. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 486–514. Springer, Heidelberg, August 2017.

[HB01] Nicholas J. Hopper and Manuel Blum. Secure human identification protocols. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 52–66. Springer, Heidelberg, December 2001.

[HGJ10] Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 235–256. Springer, Heidelberg, May / June 2010.

[HKL+12] Stefan Heyse, Eike Kiltz, Vadim Lyubashevsky, Christof Paar, and Krzysztof Pietrzak. Lapin: An efficient authentication protocol based on ring-LPN. In Anne Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 346–365. Springer, Heidelberg, March 2012.

[LF06] Éric Levieil and Pierre-Alain Fouque. An improved LPN algorithm. In Roberto De Prisco and Moti Yung, editors, *SCN 06*, volume 4116 of *LNCS*, pages 348–359. Springer, Heidelberg, September 2006.

[SS81] Richard Schroeppel and Adi Shamir. A t=o(2ˆn/2), s=o(2ˆn/4) algorithm for certain np-complete problems. *SIAM journal on Computing*, 10(3):456–464, 1981.