$i\mathcal{O}$ from PRGs in NC^0, LPN, and Bilinear Maps
Or:
On The Power of Lattice-Free Cryptography

Aayush Jain (NTT Research and CMU)

Joint work with:

Crypto from LWE/Lattices

Homomorphic Encryption [Gen 09, BV 11, BGV 12…]

Attribute-Based Encryption/Predicate Encryption [GVW 12, BGG+13, GVW 15]

MKFHE [CM 15, MW 16]

Succinct Functional Encryption [GKPVZ 13]

Universal Thresholdizers [BGG+18]

Lockable Obfuscation [GKW 17, WZ 17]

CI Hash/ NIZK [CCHLRW 19,PS 19]

SNARGS for P [CJJ 21]

Quantum FHE [M 18]

CVQC [M 18]

…….

Crypto from LWE/Lattices

Homomorphic Encryption [Gen 09, BV 11, BGV 12…]

Attribute-Based Encryption/Predicate Encryption [GVW 12, BGG+13, GVW 15]

Fundamental Question

Are Lattice based hardness assumptions essential to building these primitives?

CI Hash/ NIZK [CCHLRW 19,PS 19]

SNARGS for P [CJJ 21]

Quantum FHE [M 18]

CVQC [M 18]

…….

Our Result in a Nutshell

We can base not only these primitives but almost all known cryptographic primitives from sub exponential security of following "trio":

Decisional Linear assumption over Symmetric Bilinear Maps

Field LPN with noise rate $n^{-\delta}$

PRGs in NC^0 with stretch $\kappa^{1+\epsilon}$

Question 1: Are these assumptions connected to lattices?

Question 2: How do you show such a result?

Are the assumptions truly incomparable to Lattices?

Decisional Linear assumption over Symmetric Bilinear Maps

Field LPN with noise rate $n^{-\delta}$

PRGs in NC^0 with stretch $\kappa^{1+\epsilon}$

LPN and PRGs aren't even known to imply PKE.
Don't know if they are in coAM

DLIN has no known reductions to/from Lattices.

Exciting questions in themselves!

Our Result in a Nutshell

We can base not only these primitives but almost all known cryptographic primitives from sub exponential security of following "trio":

Decisional Linear assumption over Symmetric Bilinear Maps

Field LPN with noise rate $n^{-\delta}$

PRGs in NC^0 with stretch $\kappa^{1+\epsilon}$

Question 1: Are these assumptions connected to lattices? ✓

Question 2: How do you show such a result?

Main result

Main result

In fact, construct $i\mathcal{O}$ based on 3 well studied “non-lattice” problems.

Previous work [JLS 21] additionally assumes LWE.

Main result

In fact, construct $i\mathcal{O}$ based on 3 well studied "non-lattice" problems.

Previous work [JLS 21] additionally assumes LWE.

FHE follows from $i\mathcal{O}$
and
Perfectly re-randomizable encryption
[Canetti-Lin-Tessaro-Vaikunthanathan]

Main result

In fact, construct $i\mathcal{O}$ based on 3 well studied "non-lattice" problems.

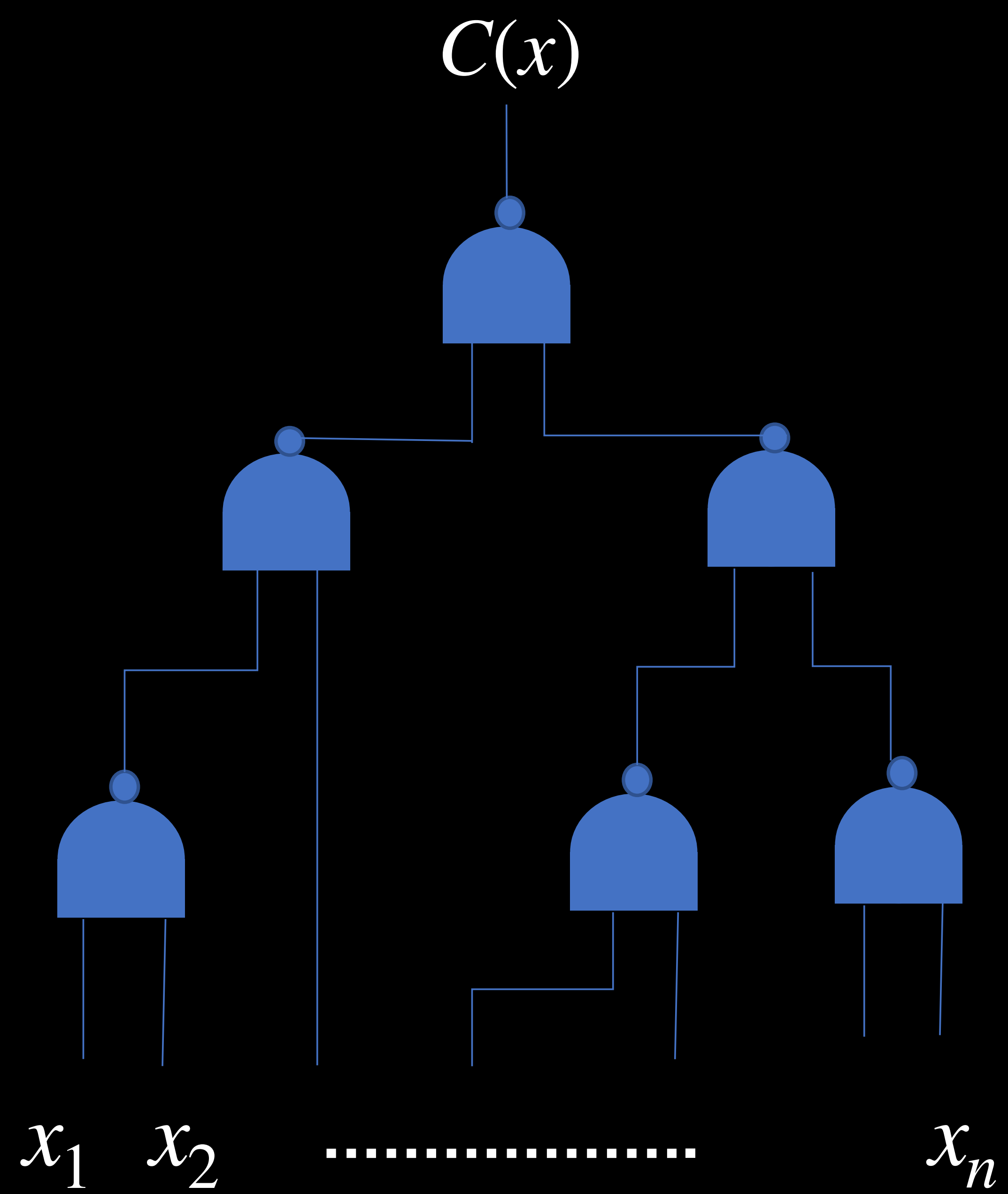
Previous work [JLS 21] additionally assumes LWE.

FHE follows from $i\mathcal{O}$
and
Perfectly re-randomizable encryption
[Canetti-Lin-Tessaro-Vaikunthanathan]

And, a number of other applications previously known only via lattices.

Obfuscation Goal

Let $C : [N = 2^n] \rightarrow \{0,1\}$ be a boolean circuit to be obfuscated.



Obfuscation Goal

Let $C : [N = 2^n] \rightarrow \{0,1\}$ be a boolean circuit to be obfuscated.

Obfuscation Goal

Let $C : [N = 2^n] \to \{0,1\}$ be a boolean circuit to be obfuscated.

A "Trivial" $i\mathcal{O}$ Scheme:

Obfuscation Goal

Let $C : [N = 2^n] \to \{0,1\}$ be a boolean circuit to be obfuscated.

A "Trivial" $i\mathcal{O}$ Scheme:

Input:	1	2	3	....	$N-1$	N
Output:	$C(1)$	$C(2)$	$C(3)$	....	$C(N-1)$	$C(N)$

The Truth-Table!

Obfuscation Goal

Let $C : [N = 2^n] \rightarrow \{0,1\}$ be a boolean circuit to be obfuscated.

A "Trivial" $i\mathcal{O}$ Scheme:

Input:	1	2	3	....	$N-1$	N
Output:	$C(1)$	$C(2)$	$C(3)$	....	$C(N-1)$	$C(N)$

The Truth-Table!

Problem:

Obfuscation takes time! $|T_{i\mathcal{O}(C)}| \propto N$

Obfuscation Goal

Let $C : [N = 2^n] \rightarrow \{0,1\}$ be a boolean circuit to be obfuscated.

The Truth-Table!

$$|T_{i\mathcal{O}(C)}| \propto N$$

Obfuscation Goal

Let $C : [N = 2^n] \rightarrow \{0,1\}$ **be a boolean circuit to be obfuscated.**

Obfuscation

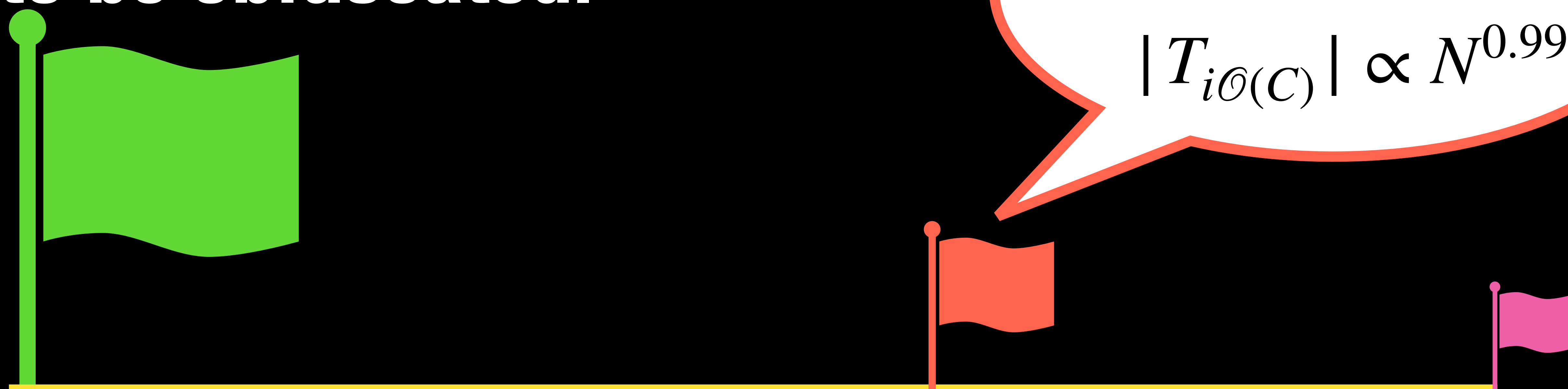
$|T_{i\mathcal{O}}(C)| \propto \text{poly}(|C|)$

The Truth-Table!

$|T_{i\mathcal{O}(C)}| \propto N$

Obfuscation Goal

Let $C : [N = 2^n] \to \{0{,}1\}$ be a b... it to be obfuscated.



Non-Trivial $i\mathcal{O}$

$|T_{i\mathcal{O}(C)}| \propto N^{0.99}$

Obfuscation

$|T_{i\mathcal{O}}(C)| \propto \mathrm{poly}(|C|)$

The Truth-Table!

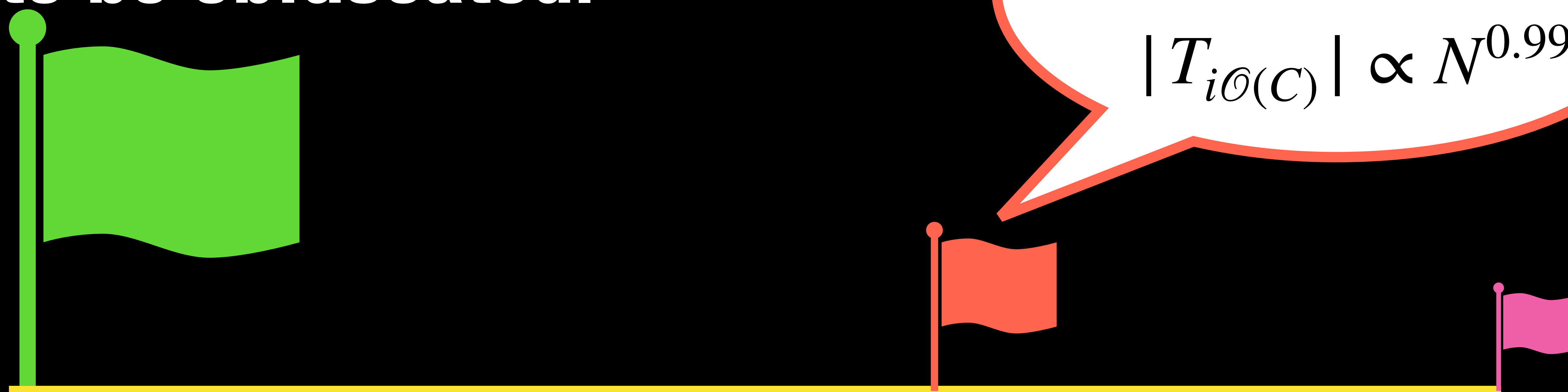
$|T_{i\mathcal{O}(C)}| \propto N$

Obfuscation Goal

Let $C : [N = 2^n] \to \{0,1\}$ be a b
it
to be obfuscated.

Non-Trivial $i\mathcal{O}$

$|T_{i\mathcal{O}(C)}| \propto N^{0.99}$



Obfuscation

$|T_{i\mathcal{O}}(C)| \propto \text{poly}(|C|)$

The Truth-Table!

$|T_{i\mathcal{O}(C)}| \propto N$

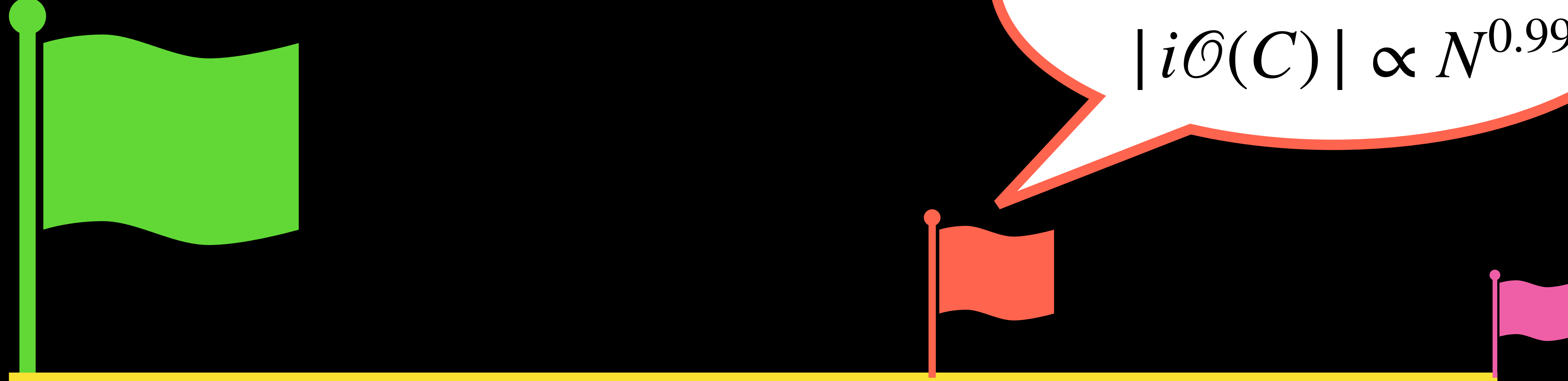
Non-Trivial $i\mathcal{O} + \text{DLIN} \Longrightarrow i\mathcal{O}$**!**

[BV 15, AJ 15, LPST 16, BNPW 17]

Obfuscation Goal

Let $C : [N = 2^n] \to \{0,1\}$ be a b ... it to be obfuscated.

Previous work:
$|i\mathcal{O}(C)| \propto N^{0.99}$



Obfuscation
$|T_{i\mathcal{O}}(C)| \propto \text{poly}(|C|)$

The Truth-Table!
$|T_{i\mathcal{O}(C)}| \propto N$

Non-Trivial $i\mathcal{O} + \text{LWE} \Longrightarrow i\mathcal{O}$**!**

[BV 15, AJ 15, LPST 16, BNPW 17]

Our Approach

Our Approach

Special Encryption Scheme

Encrypt($\tilde{C} = (C, r)$)

Size of $\tilde{C}$:

$|\tilde{C}| \propto N^{0.99}$

Our Approach

Special Encryption Scheme

Encrypt($\tilde{C} = (C, r)$)

Can learn functions

$\{U_x(\tilde{C}) = C(x)\}_{x \in [N]}$

$U_1(\tilde{C})\ U_2(\tilde{C}) \cdots\cdots\cdots\cdots U_x(\tilde{C}) \cdots U_N(\tilde{C})$

Size of $\tilde{C}$:

$|\tilde{C}| \propto N^{0.99}$

Our Approach

Special Encryption Scheme

Encrypt($\tilde{C} = (C, r)$)

Size of $\tilde{C}$:

$|\tilde{C}| \propto N^{0.99}$

Can learn functions

$\{U_x(\tilde{C}) = C(x)\}_{x \in [N]}$

$U_1(\tilde{C})\ U_2(\tilde{C}) \cdots\cdots U_x(\tilde{C}) \cdots U_N(\tilde{C})$

Truth-Table!

Learn Nothing Else

Our Approach

Special Encryption Scheme

Encrypt($\tilde{C} = (C, r)$)

Size of $\tilde{C}$:

$|\tilde{C}| \propto N^{0.99}$

Can learn functions

$\{U_x(\tilde{C}) = C(x)\}_{x \in [N]}$

$U_1(\tilde{C})\ U_2(\tilde{C}) \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\ U_x(\tilde{C}) \cdots U_N(\tilde{C})$

Problem:

$U_x(C, r) = C(x)$ is too complex!

Truth-Table!

Learn Nothing Else

Our Approach

How simple can U_x be?

Application of [Yao 86, AIK 04, L 17, AS 17]:

If PRGs with locality d exist

$3d+1$**- Local:** $U_x(\tilde{C})$ **depends on** $3d+1$ **bits of** $\tilde{C}$**.**

Degree-16 polynomial!

Nothing Else

Our Approach

Special Encryption Scheme

Encrypt($\tilde{C} = (C, r)$)

Can learn "degree-16" functions

$$\{U_x(\tilde{C})\}_{x\in[N]}$$

$U_1(\tilde{C})$ $U_2(\tilde{C})$ .... $U_x(\tilde{C}) \cdots U_N(\tilde{C})$

Size of $\tilde{C}$:

$|\tilde{C}| \propto N^{0.99}$

Our Approach

Special Encryption Scheme

Encrypt($\tilde{C} = (C, r)$)

Size of $\tilde{C}$:

$|\tilde{C}| \propto N^{0.99}$

Can learn "degree-16" functions

$\{U_x(\tilde{C})\}_{x \in [N]}$

$U_1(\tilde{C})\ U_2(\tilde{C})\ \cdots\cdots\ U_x(\tilde{C}) \cdots U_N(\tilde{C})$

Recover($\{U_x(\tilde{C})\}_{x \in [N]}$)

Truth-Table: $(C(1), \ldots, C(N))$

Our Approach

Special Encryption Scheme

Encrypt($\tilde{C} = (C, r)$)

Size of $\tilde{C}$:

$|\tilde{C}| \propto N^{0.99}$

Can learn "degree-16" functions

$\{U_x(\tilde{C})\}_{x \in [N]}$

$U_1(\tilde{C})\ U_2(\tilde{C}) \cdots\cdots\cdots\cdots\ U_x(\tilde{C}) \cdots U_N(\tilde{C})$

Recover($\{U_x(\tilde{C})\}_{x \in [N]}$)

Truth-Table: $(C(1), \ldots, C(N))$

Learn Nothing Else

Our Approach

Encrypt($\tilde{C} = (C, r)$)

Can learn "degree-16" functions

$$\{U_x(\tilde{C})\}_{x \in [N]}$$

$U_1(\tilde{C})$ $U_2(\tilde{C})$ .... $U_x(\tilde{C})$ ... $U_N(\tilde{C})$

Our Approach

Good News:

Can handle quadratic functions

[Lin 17, AJLMS 19, JLMS 19, GJLS 20, Wee 21].

Based on DLIN

Encrypt($\tilde{C} = (C, r)$)

Can learn "degree-16" functions

$\{U_x(\tilde{C})\}_{x \in [N]}$

$U_1(\tilde{C})\ U_2(\tilde{C}) \cdots\cdots\cdots\ U_x(\tilde{C}) \cdots U_N(\tilde{C})$

$$U_x(\tilde{C}) = \sum_{i,j} q_{x,i,j} \tilde{C}_i \cdot \tilde{C}_j \mod p$$

Our Approach

Good News:

Can handle quadratic functions

[Lin 17, AJLMS 19, JLMS 19, GJLS 20, Wee 21].

Based on DLIN

$$U_x(\tilde{C}) = \sum_{i,j} q_{x,i,j} \tilde{C}_i \cdot \tilde{C}_j \mod p$$

Encrypt$(\tilde{C} = (C, r))$

Can learn "degree-16" functions

$$\{U_x(\tilde{C})\}_{x \in [N]}$$

$U_1(\tilde{C})$ $U_2(\tilde{C})$ .... $U_x(\tilde{C}) \cdots U_N(\tilde{C})$

Our Approach

Good News:

Can handle quadratic functions

[Lin 17, AJLMS 19, JLMS 19, GJLS 20, Wee 21].

Based on DLIN

$$U_x(\tilde{C}) = \sum_{i,j} q_{x,i,j}\tilde{C}_i \cdot \tilde{C}_j \mod p$$

Encrypt($\tilde{C} = (C, r)$)

Can learn "degree-16" functions $\{U_x(\tilde{C})\}_{x\in[N]}$

$U_1(\tilde{C})$ $U_2(\tilde{C})$ .... $U_x(\tilde{C}) \cdots U_N(\tilde{C})$

Problem: U_x is degree - 16!!

Our Approach

Good News:

Can handle quadratic functions

[Lin 17, AJLMS 19, JLMS 19, GJLS 20, Wee 21].

Based on DLIN

Encrypt($\tilde{C} = (C, r)$)

Can learn "degree-16" functions

$$\{U_x(\tilde{C})\}_{x \in [N]}$$

$$U_x(\tilde{C}) = \sum_{i,j} q_{x,i,j} \tilde{C}_i \cdot \tilde{C}_j \mod p$$

$U_1(\tilde{C})$ $U_2(\tilde{C})$ .... $U_x(\tilde{C}) \cdots U_N(\tilde{C})$

Problem: U_x is degree - 16!!

Goal: Replace U_x by quadratic functions.

Our Approach

Good N

Can ha

quadratic functi

[Lin 17, AJLMS 1

Wee 21].

Bas

DLIN

Public $P(\tilde{C})$

Encrypt($S(\tilde{C})$)

Coefficients constant degree polynomial over P

n "degree-16" functions

$\{U_x(\tilde{C})\}_{x \in [N]}$

$$U_x(\tilde{C}) = \sum_{i,j} q_{x,i,j} S_i \cdot S_j \mod p$$

$U_1(\tilde{C})$ $U_2(\tilde{C})$ .... $U_x(\tilde{C}) \cdots U_N(\tilde{C})$

Problem: U_x is degree - 16!!

Goal: Replace U_x by quadratic functions.

Use of LPN

Goal: Replace U_x by quadratic functions.

Use of LPN

Goal: Replace U_x by quadratic functions.

Step 1: Approximate $U_x(\tilde{C})$ by quadratic $f_x(S)$,

$$f_x(S) = U_x(\tilde{C}) \quad \text{for most inputs } x \in [N]$$

Main Idea: Variable change via LPN.

Use of LPN

Goal: Replace U_x by quadratic functions.

Step 1: Approximate $U_x(\tilde{C})$ by quadratic $f_x(S)$,

$$f_x(S) = U_x(\tilde{C}) \quad \text{for most inputs } x \in [N]$$

Main Idea: Variable change via LPN.

Step 2: Error correction via quadratic $\mathsf{Corr}_x(M)$

$$h_x(S, M) = f_x(S) + \mathsf{Corr}_x(M) = U_x(\tilde{C}) \qquad \forall\ x \in [N]$$

Main Idea: Compression via Matrix Factorization.

Use of LPN

Goal: Replace U_x by quadratic functions.

Step 1: Approximate $U_x(\tilde{C})$ by quadratic $f_x(S)$,

$$f_x(S) = U_x(\tilde{C}) \quad \text{for most inputs } x \in [N]$$

Main Idea: Variable change via LPN.

Step 2: Error correction via quadratic $\mathsf{Corr}_x(M)$

$$h_x(S, M) = f_x(S) + \mathsf{Corr}_x(M) = U_x(\tilde{C}) \qquad \forall\ x \in [N]$$

Main Idea: Compression via Matrix Factorization.

Applying LPN

Applying LPN

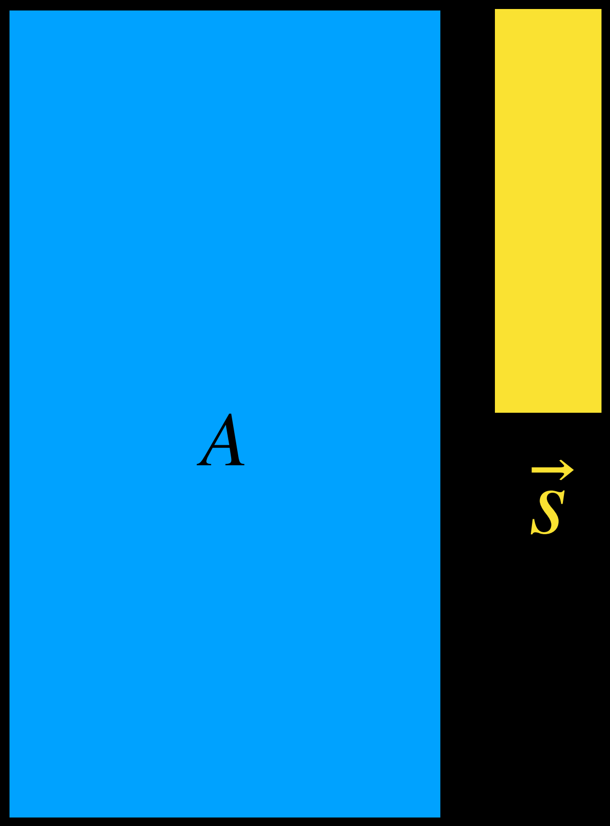
1. **Write** $\tilde{C} = (C, r)$

Applying LPN

1. **Write** $\tilde{C} = (C, r)$
2. **Encode** $\tilde{C}$ **into** $\overrightarrow{b}$

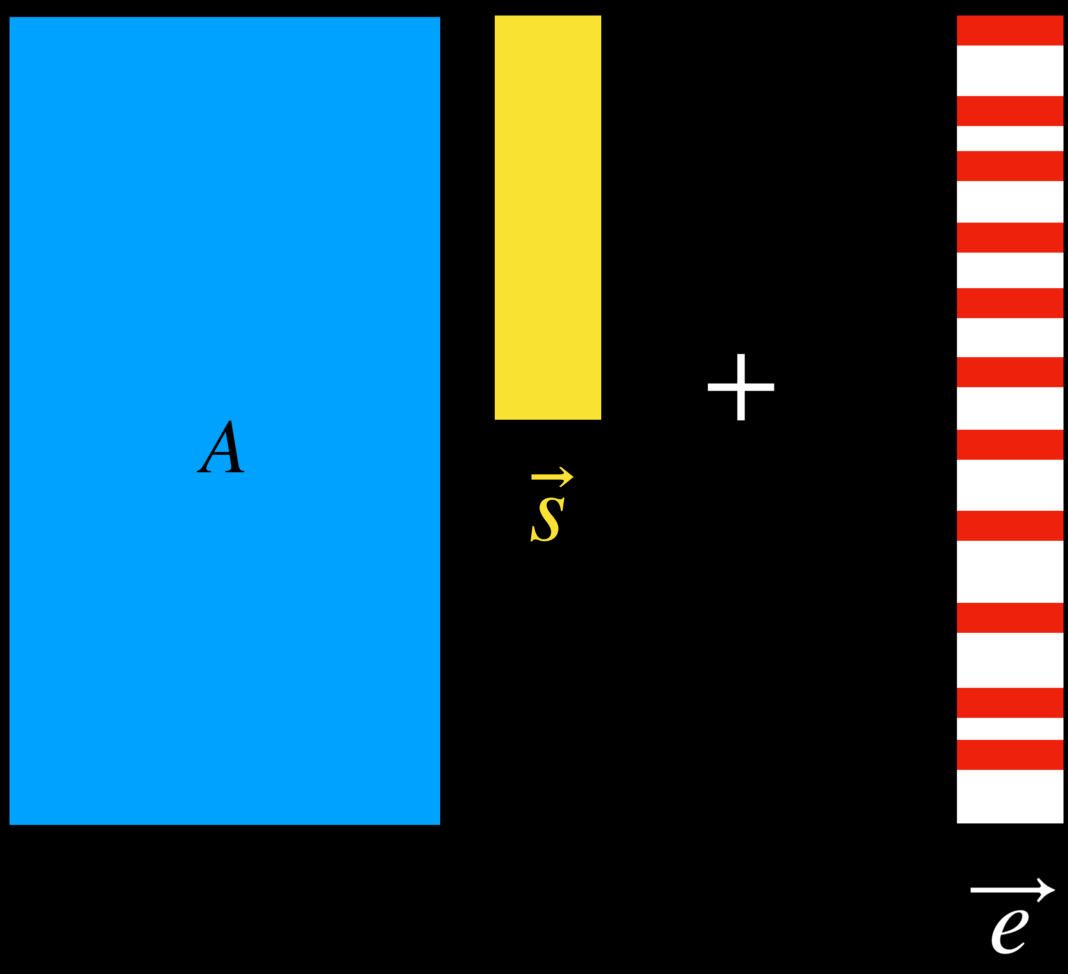
Applying LPN

1. Write $\tilde{C} = (C, r)$
2. Encode $\tilde{C}$ into $\overrightarrow{b}$



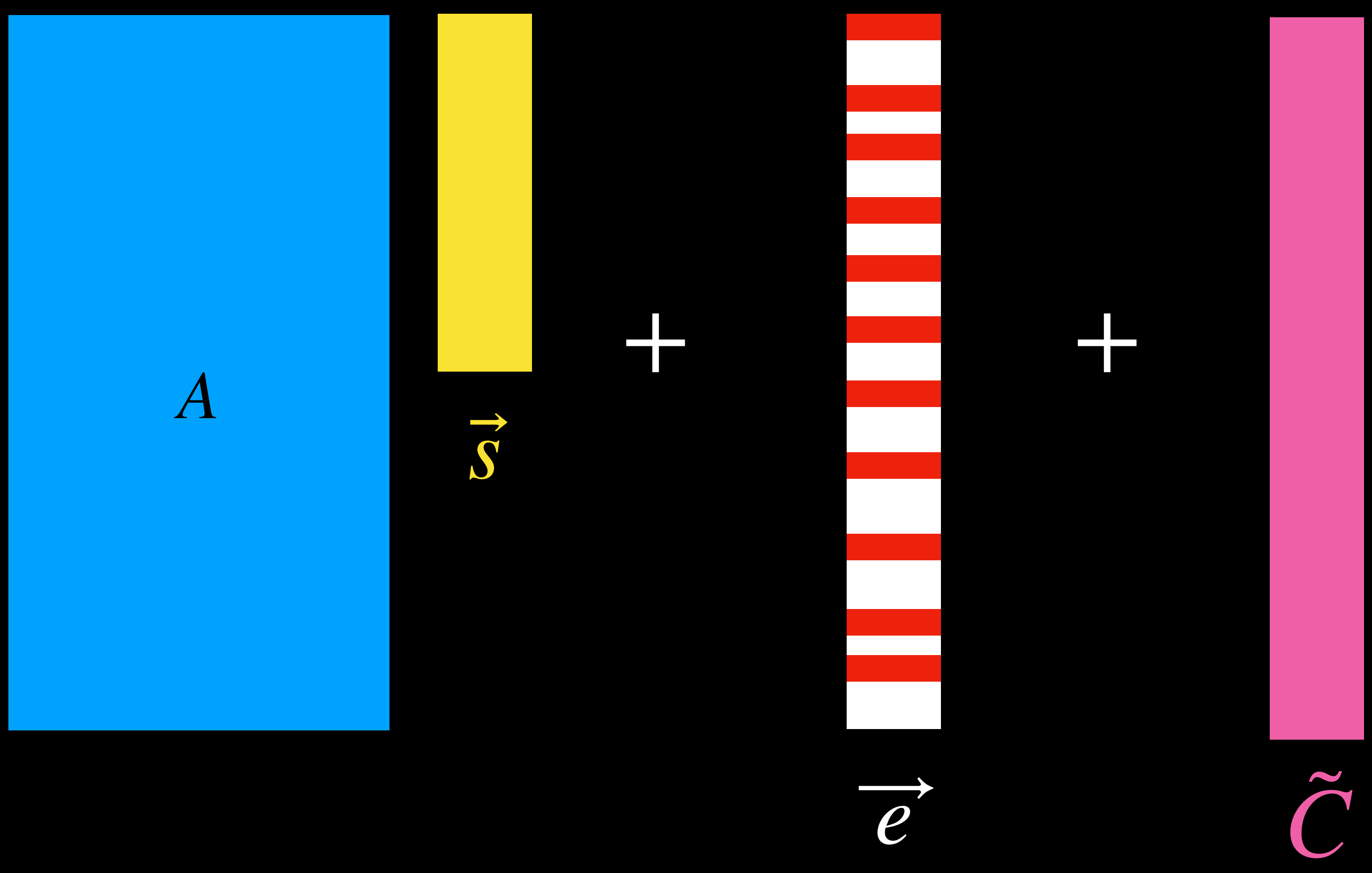
Applying LPN

1. **Write** $\tilde{C} = (C, r)$
2. **Encode** $\tilde{C}$ **into** $\overrightarrow{b}$



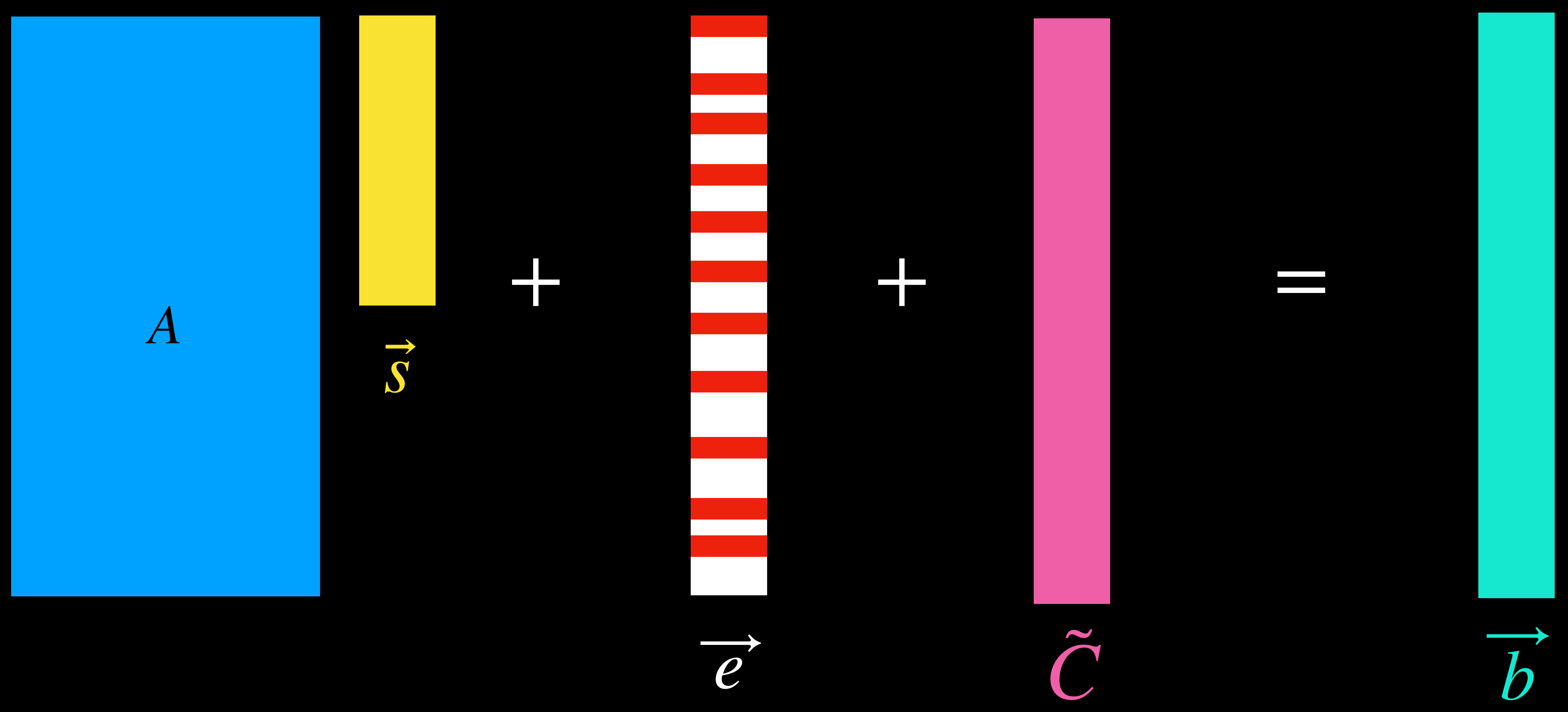
Applying LPN

1. Write $\tilde{C} = (C, r)$
2. Encode $\tilde{C}$ into $\overrightarrow{b}$



Applying LPN

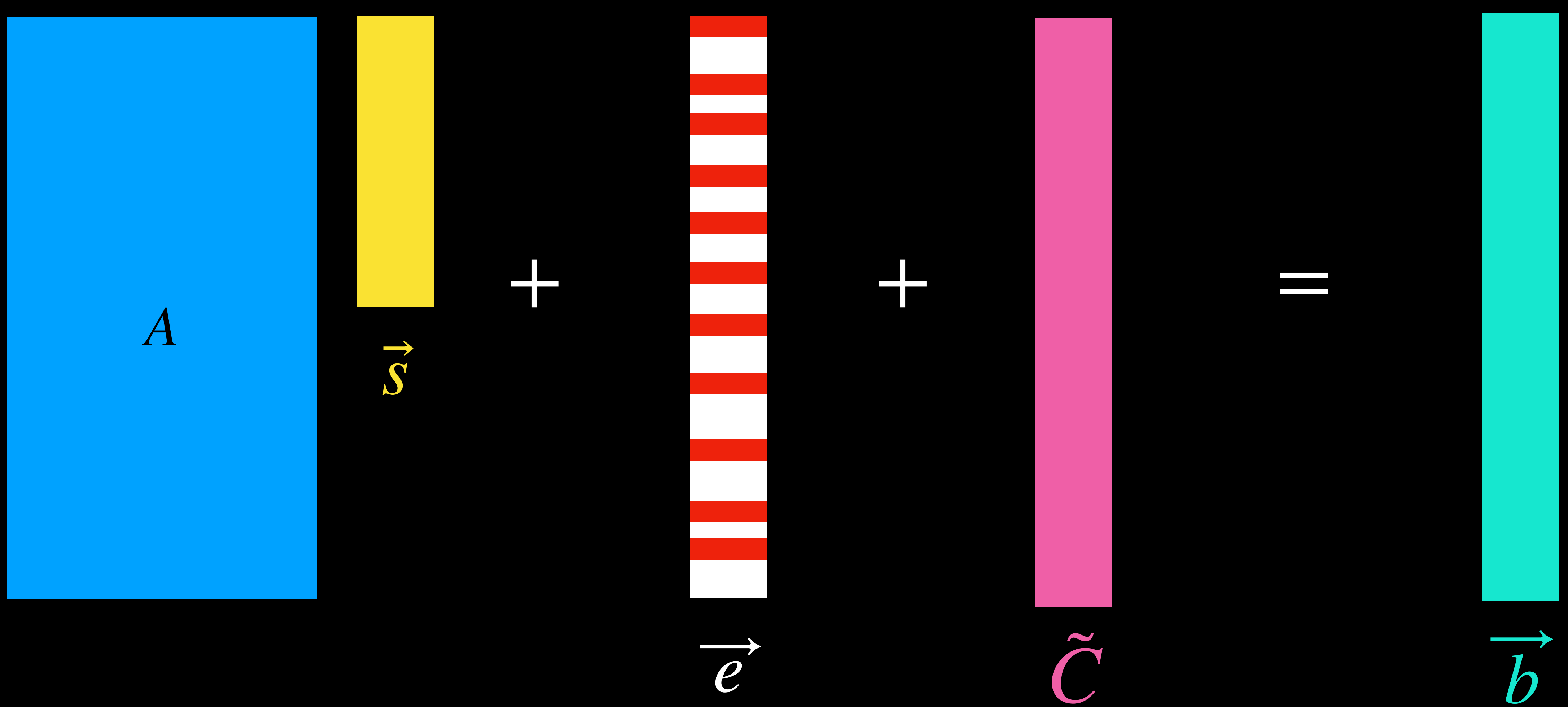
1. Write $\tilde{C} = (C, r)$
2. Encode $\tilde{C}$ into $\overrightarrow{b}$



Applying LPN

1. **Write** $\tilde{C} = (C, r)$
2. **Encode** $\tilde{C}$ **into** $\overrightarrow{b}$

$\overrightarrow{A}, \overrightarrow{b}$ **encrypts** $\tilde{C}$ **!**



Applying LPN

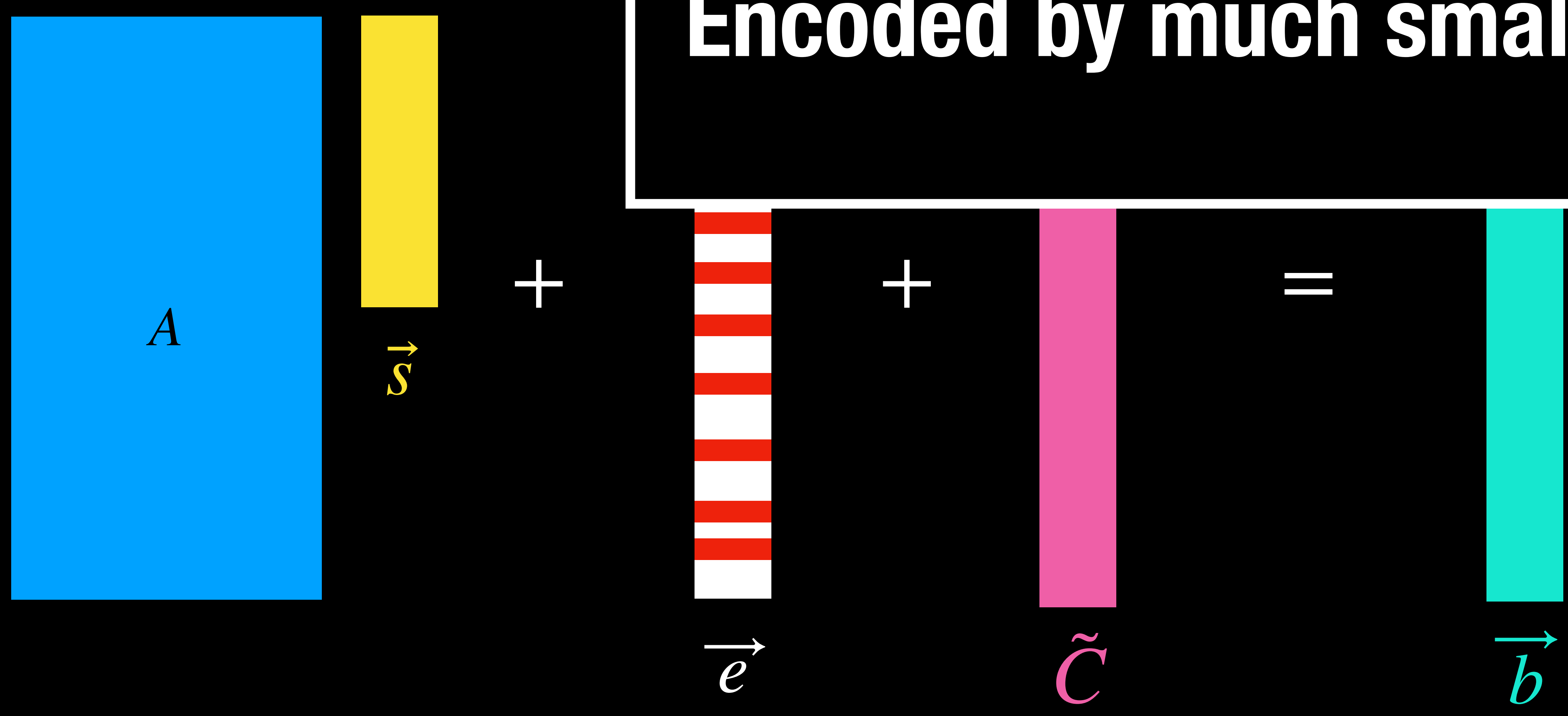
1. Write $\tilde{C} = (C,$

2. Encode $\tilde{C}$ int

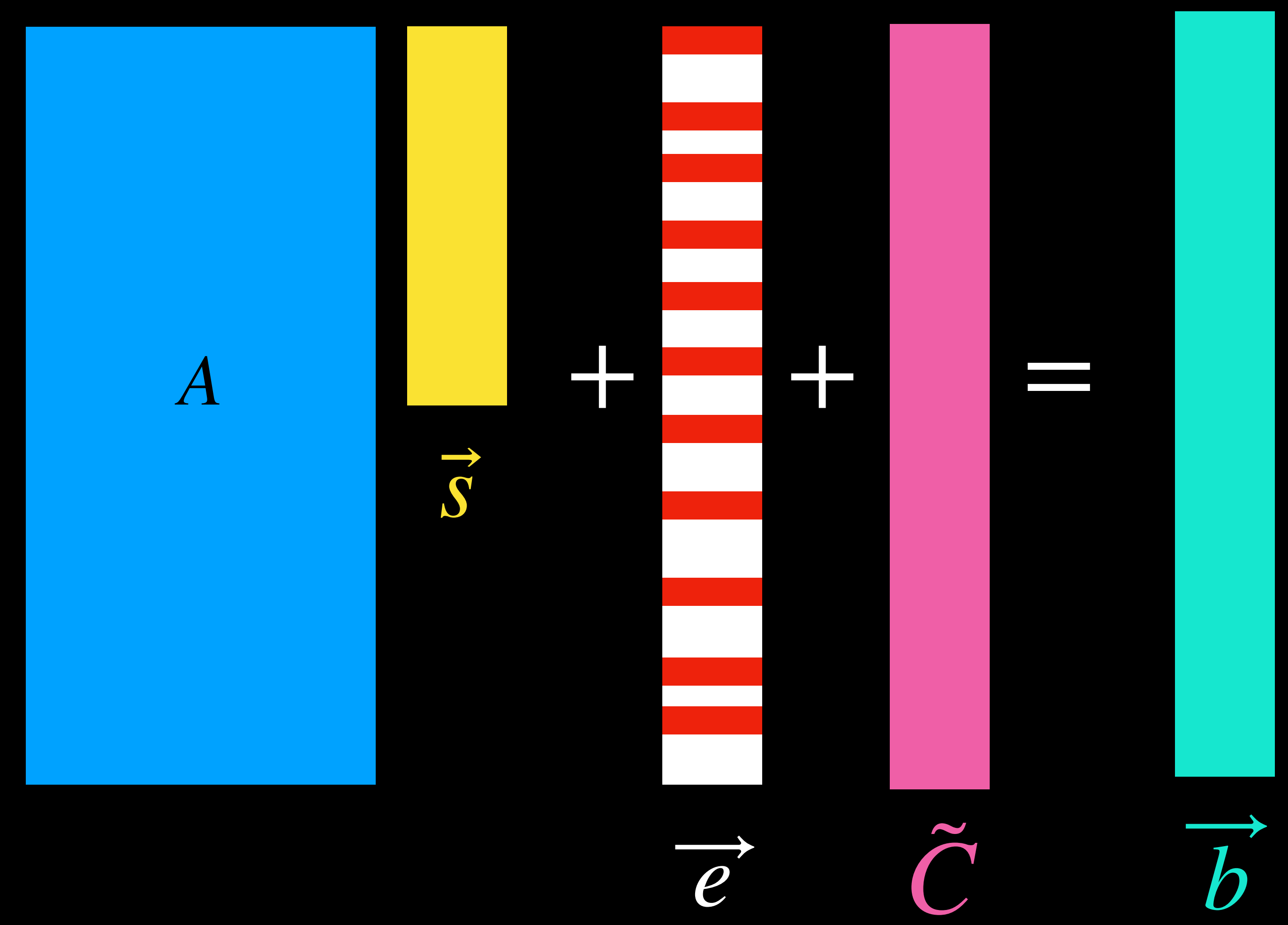
$\boxed{\overrightarrow{A}, \overrightarrow{b} \text{ encrypts } \tilde{C}\,!}$

Can be made public!

Encoded by much smaller $\vec{s}$



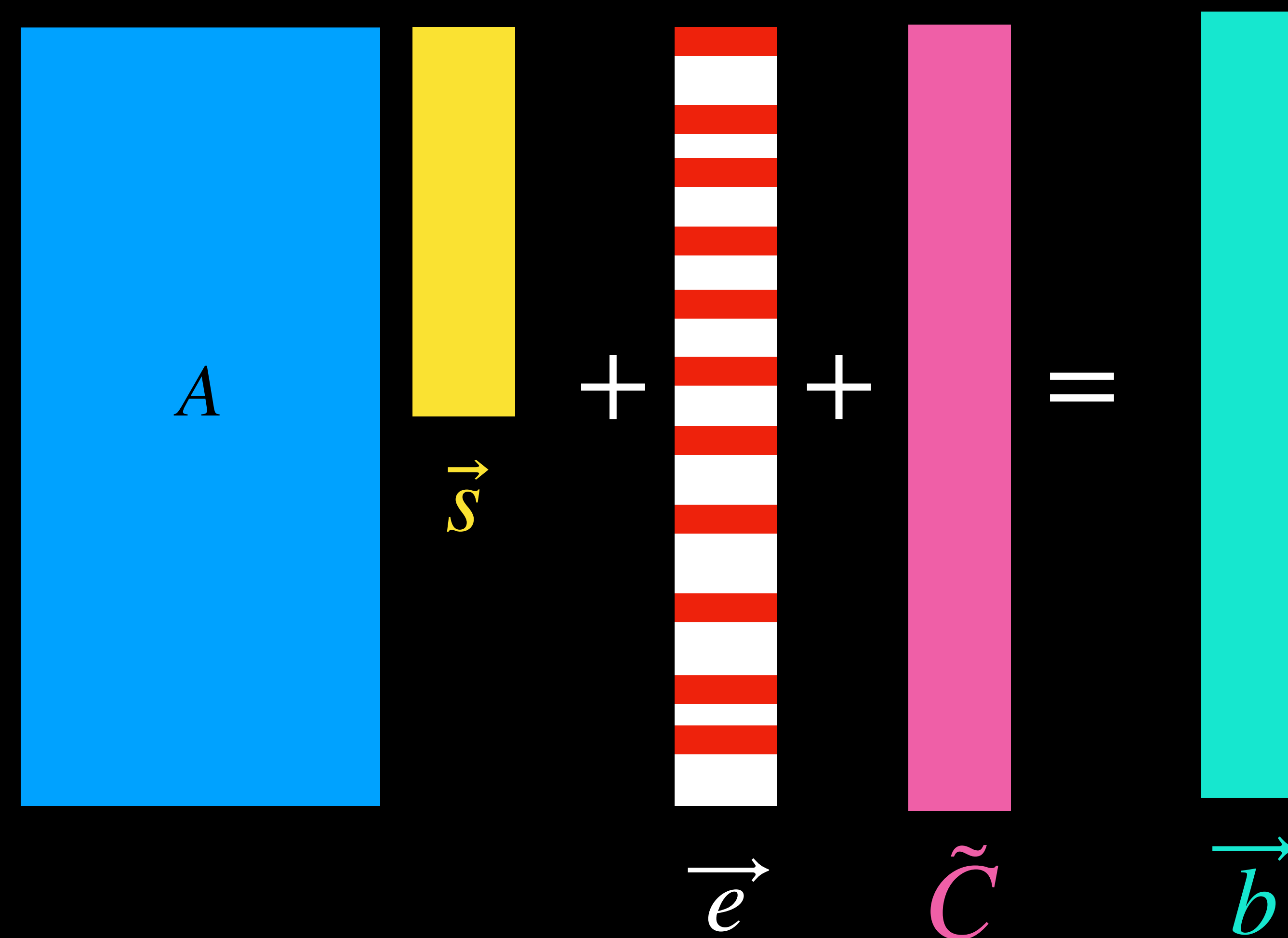
Applying LPN



Applying LPN

Goal: Find f_x that is:

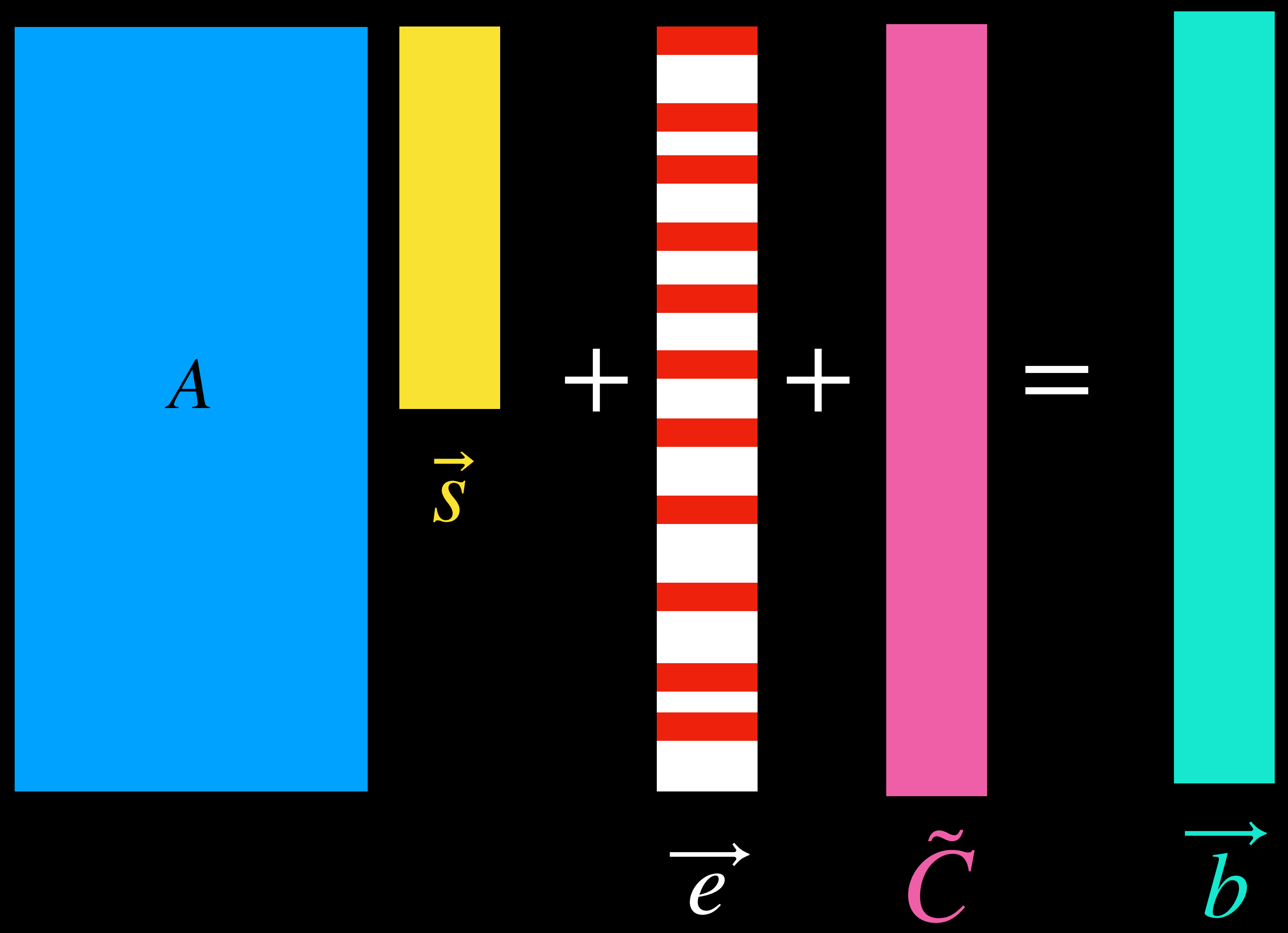
- Quadratic in short S,
- For most x, $f_x(S) = U_x(\tilde{C})$



Applying LPN

Goal: Find f_x that is:

- Quadratic in short S,
- For most x, $f_x(S) = U_x(\tilde{C})$

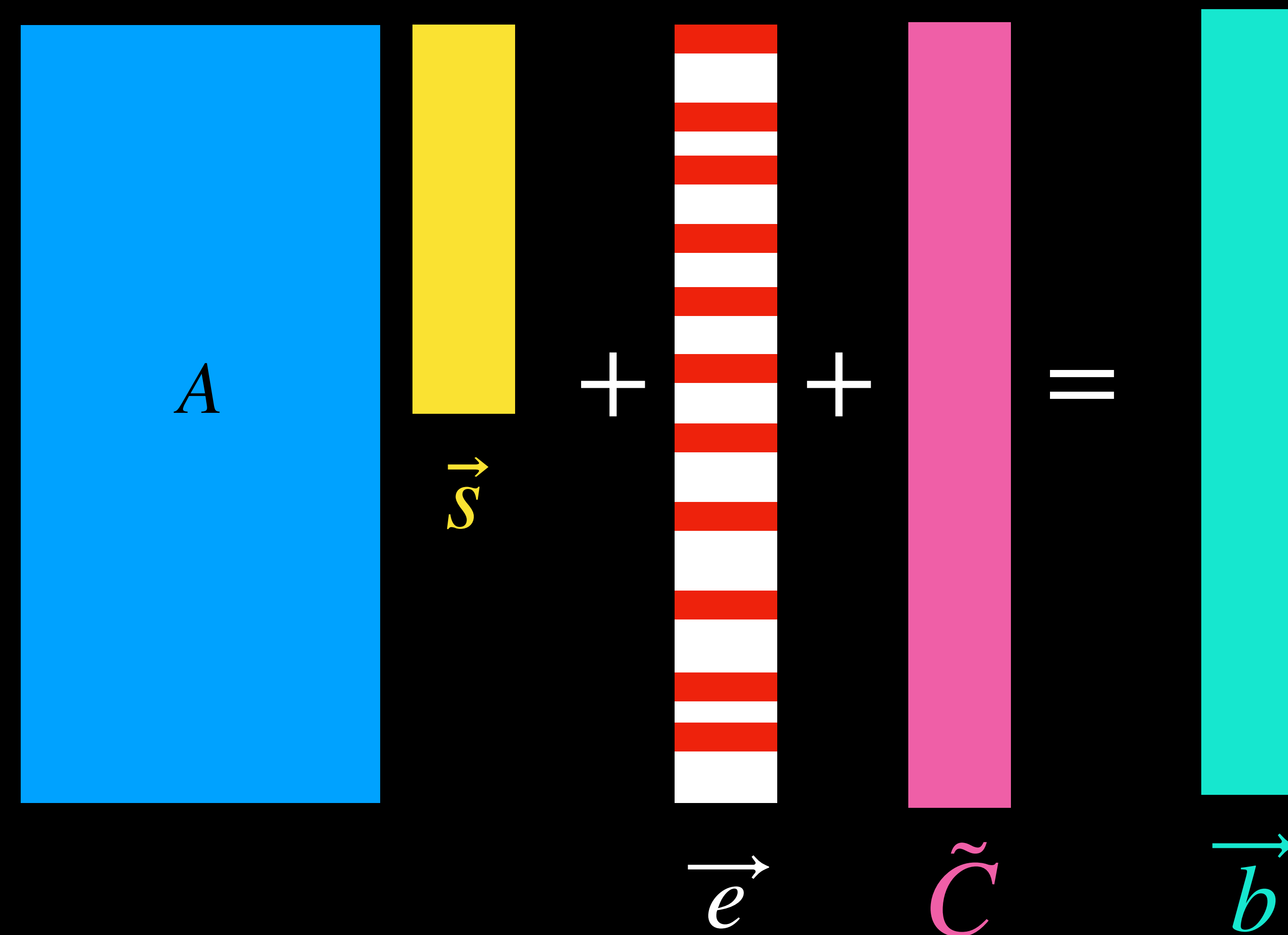


Consider:

Applying LPN

Goal: Find f_x that is:

- Quadratic in short S,
- For most x, $f_x(S) = U_x(\tilde{C})$

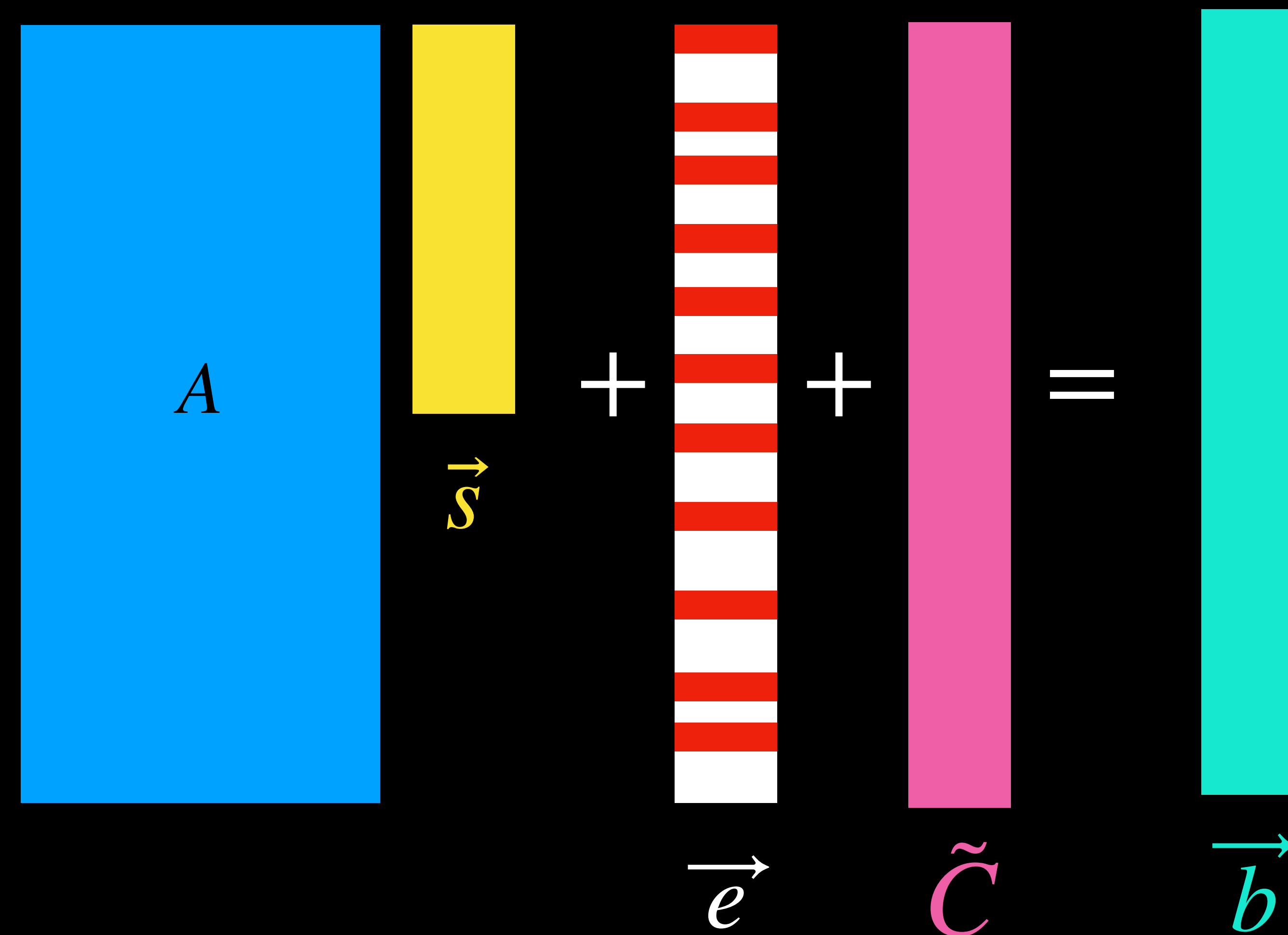


Consider:

$$U_x(\vec{b} - A\vec{s})$$

$\boxed{\text{Degree - 16 in } \vec{s}}$

Applying LPN



Goal: Find f_x that is:

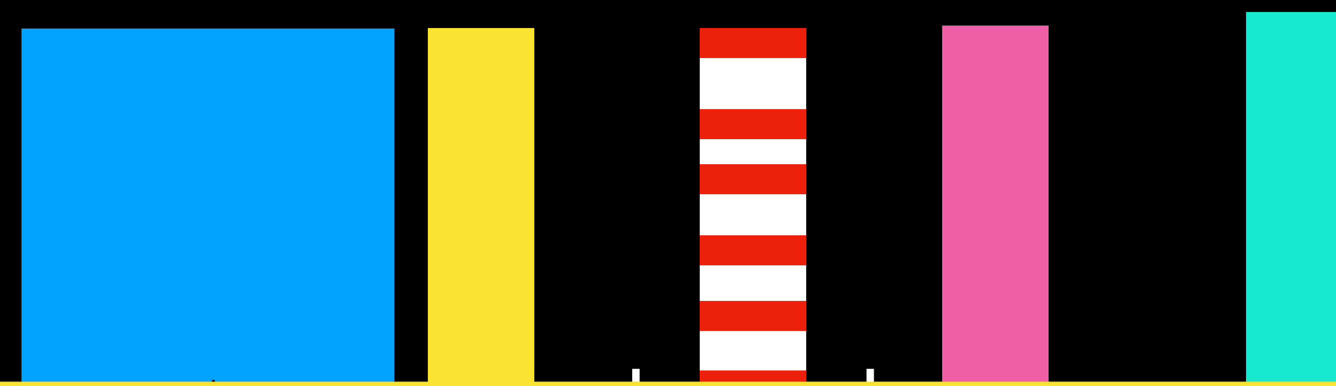
- Quadratic in short S,
- For most x, $f_x(S) = U_x(\tilde{C})$

Consider:

$$U_x(\tilde{C} + \overrightarrow{e}) = U_x(\overrightarrow{b} - A\vec{s})$$

Degree - 16 in $\vec{s}$

Applying LPN



Goal: Find f_x that is:

- Quadratic in short S.

$\boxed{\{f_x\}_x \text{ approximates } \{U_x\}_x}$

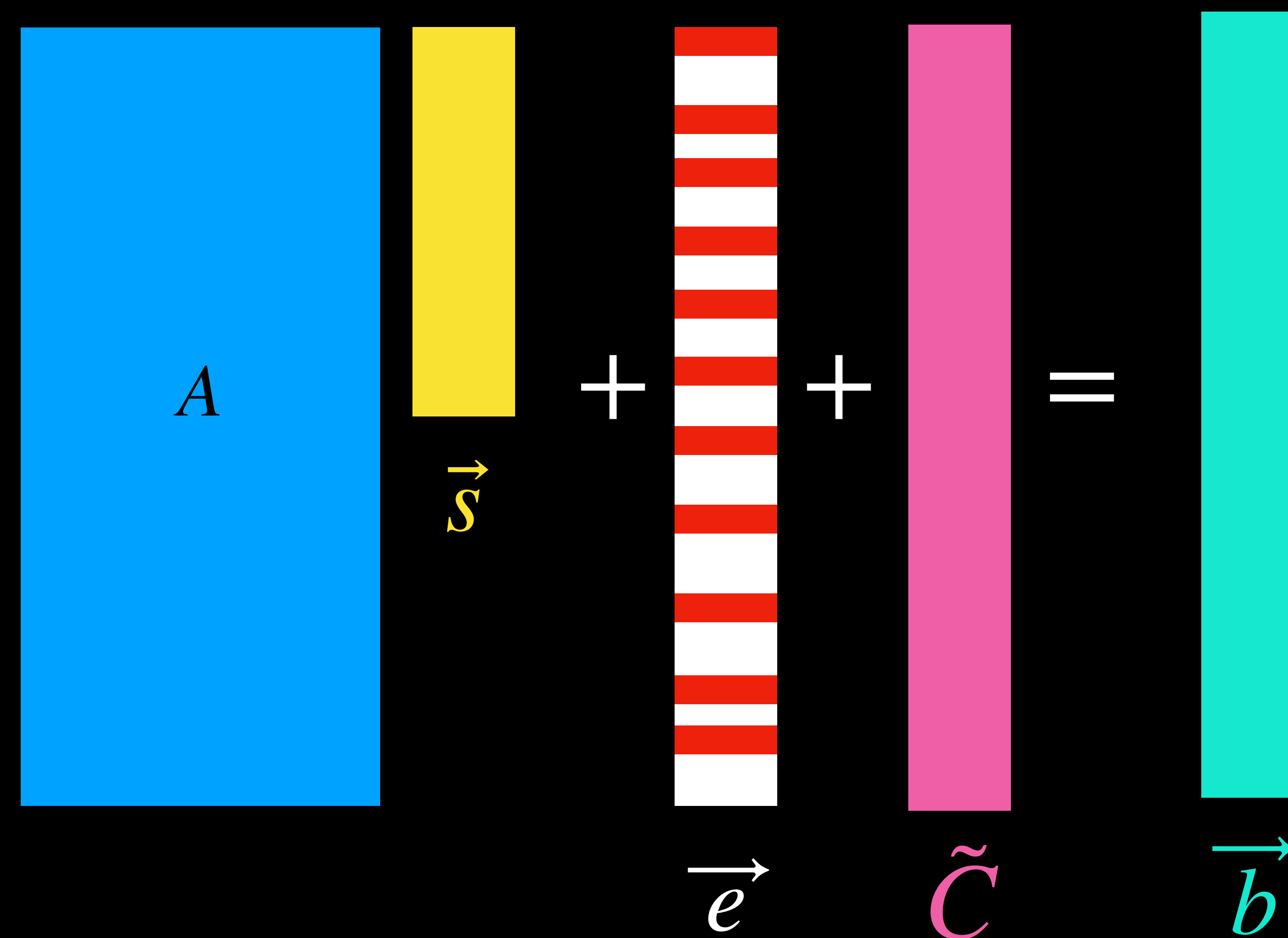
$\vec{e}$ **is sparse,** U_x **depends on 16 bits, for any** x

$$U_x(\tilde{C} + \vec{e}) = U_x(\tilde{C})$$

with high probability (over $\vec{e}$**).**

Degree - 16 in s

Applying LPN



Goal: Find f_x that is:

- Quadratic in short S,
- For most x, $f_x(S) = U_x(\tilde{C})$

Consider:

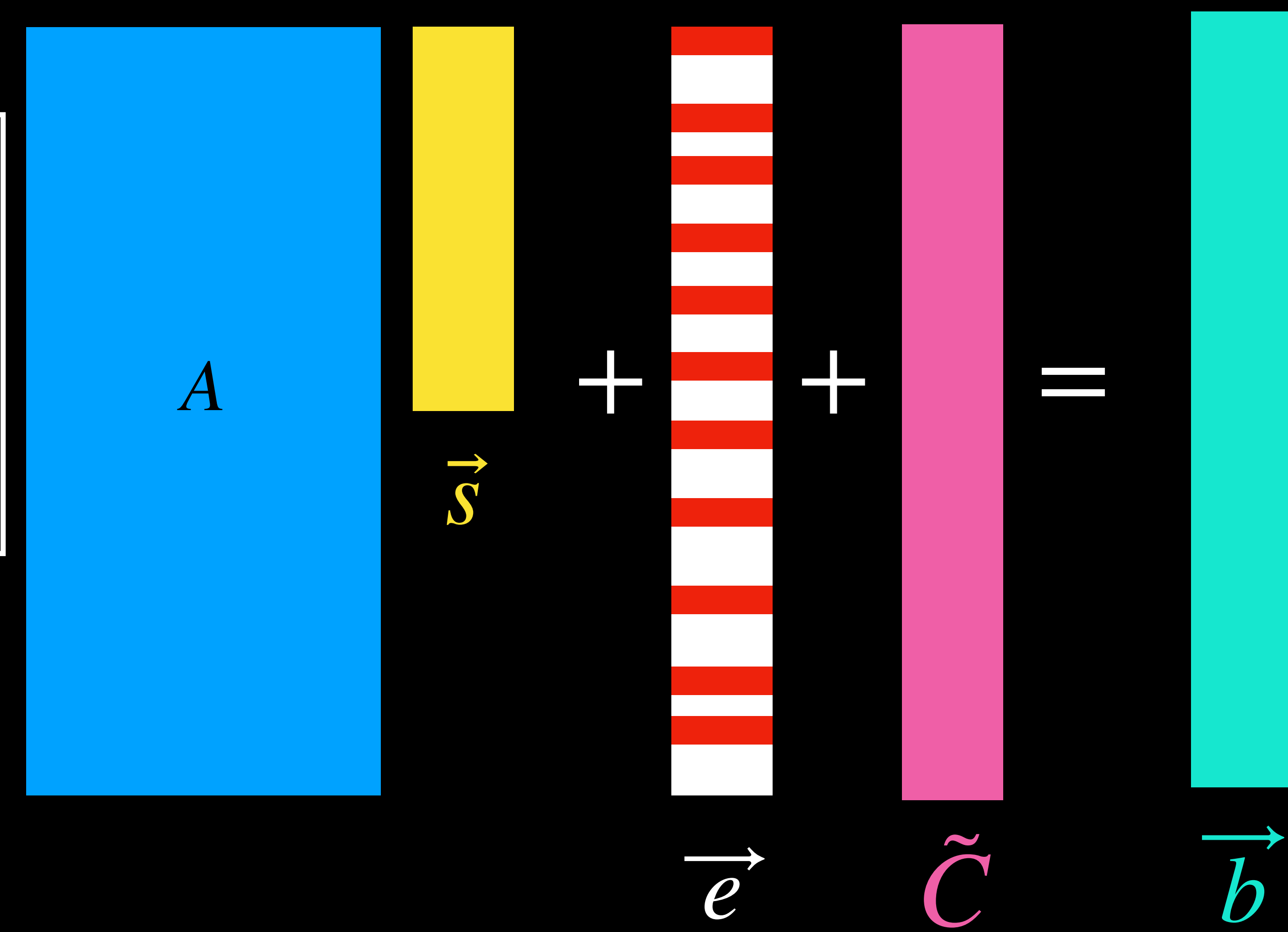
$$U_x(\tilde{C} + \vec{e}) = U_x(\vec{b} - A\vec{s})$$

Degree - 16 in $\vec{s}$

Applying LPN

Goal: Find f_x that is:

- Quadratic in short S,
- For most x, $f_x(S) = U_x(\tilde{C})$



Consider:

$$U_x(\tilde{C} + \vec{e}) = U_x(\vec{b} - A\vec{s})$$

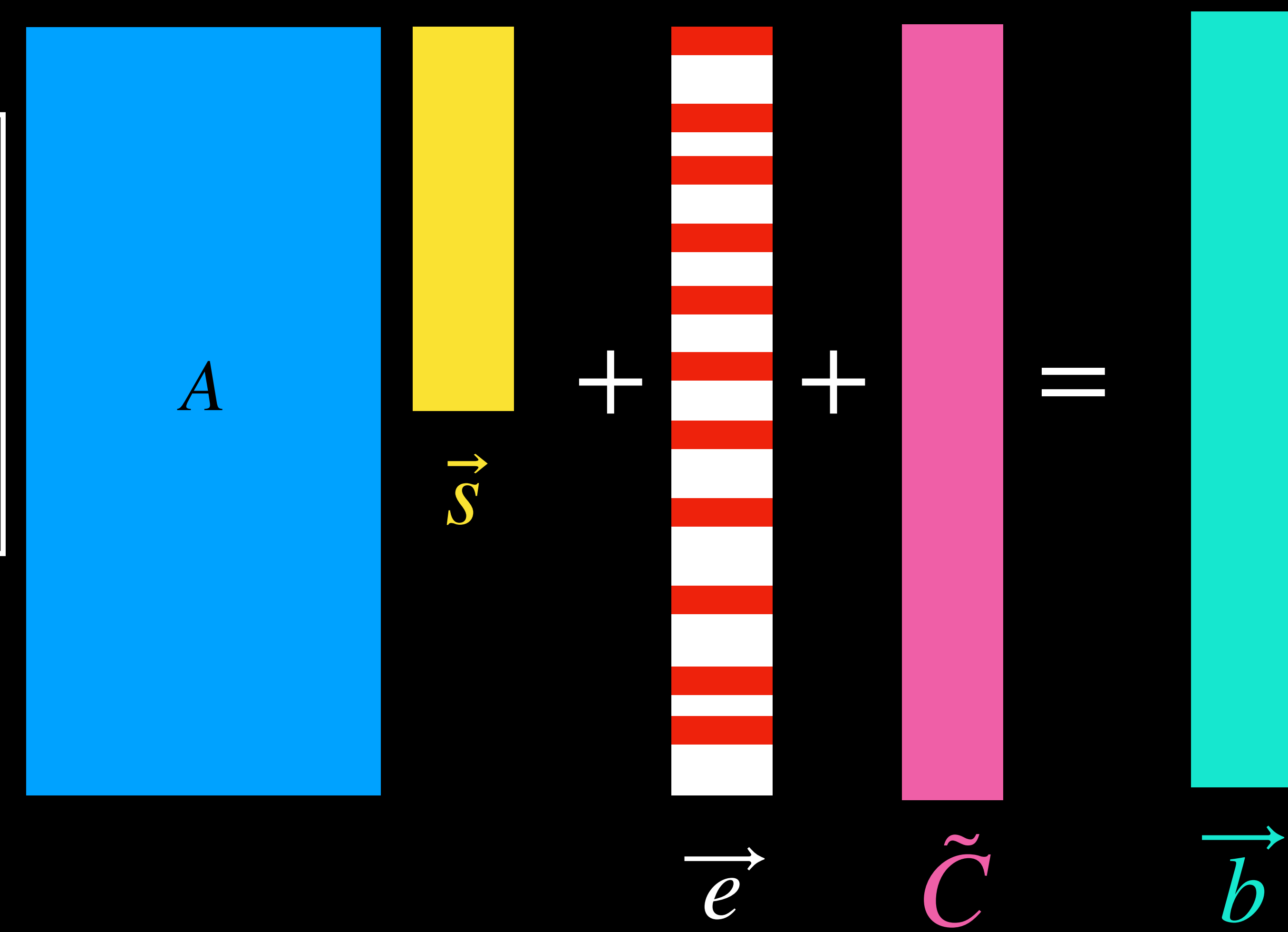
Degree - 16 in $\vec{s}$

$|\vec{s}|$ **is very small**

Applying LPN

Goal: **Find f_x that is:**

- **Quadratic in short S,**
- **For most x, $f_x(S) = U_x(\tilde{C})$**



Consider:

$$U_x(\tilde{C} + \vec{e}) = U_x(\vec{b} - A\vec{s}) = f_x(S)$$

Degree - 16 in $\vec{s}$

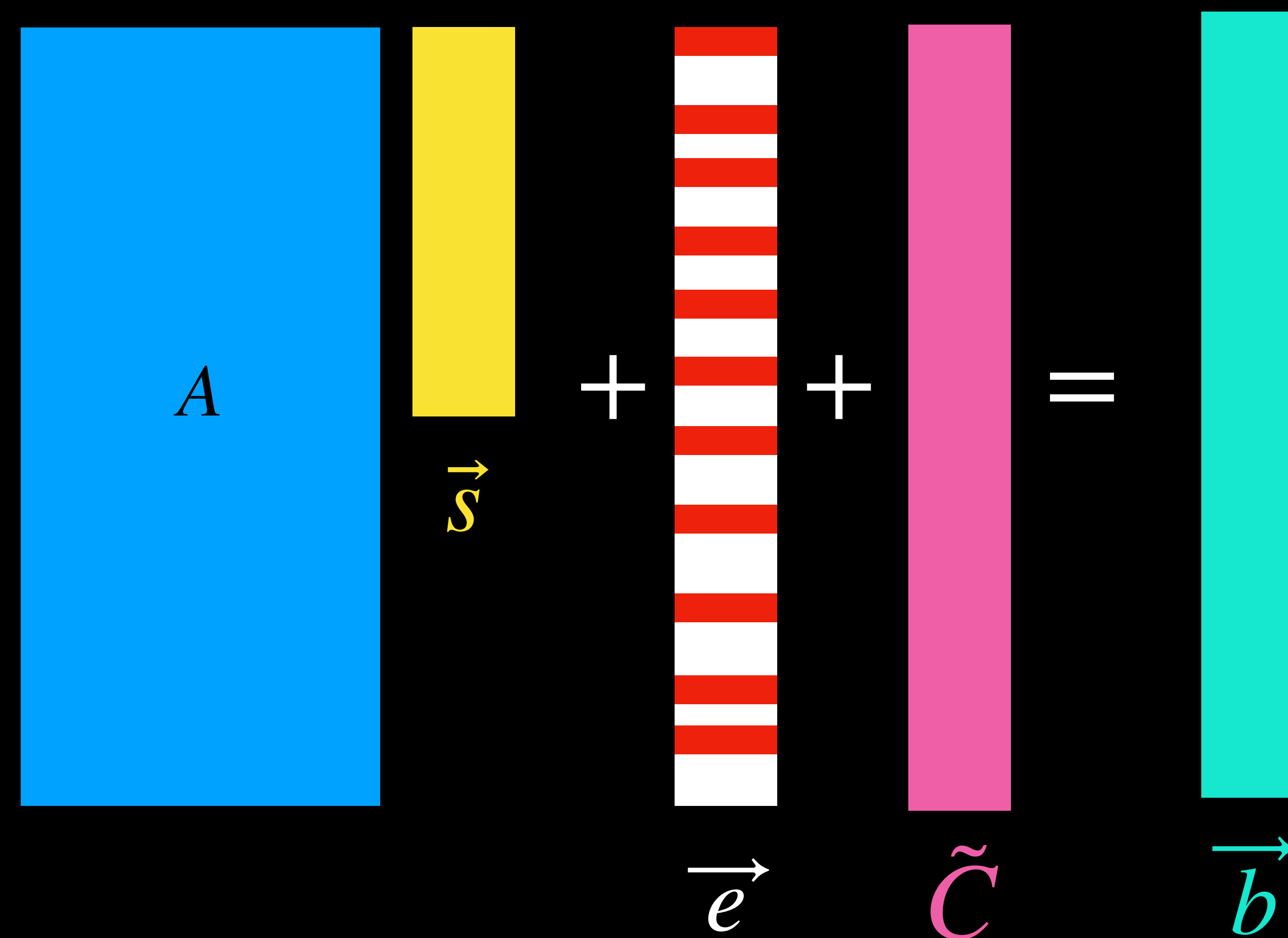
Quadratic in $S = (\vec{s}, 1)^{\otimes 8}$

$|\vec{s}|$ is very small

Applying LPN

Goal: Find f_x that is:

- Quadratic in short S,
- For most x, $f_x(S) = U_x(\tilde{C})$



Consider:

$$U_x(\tilde{C} + \vec{e}) = U_x(\vec{b} - A\vec{s}) = f_x(S)$$

Degree - 16 in $\vec{s}$

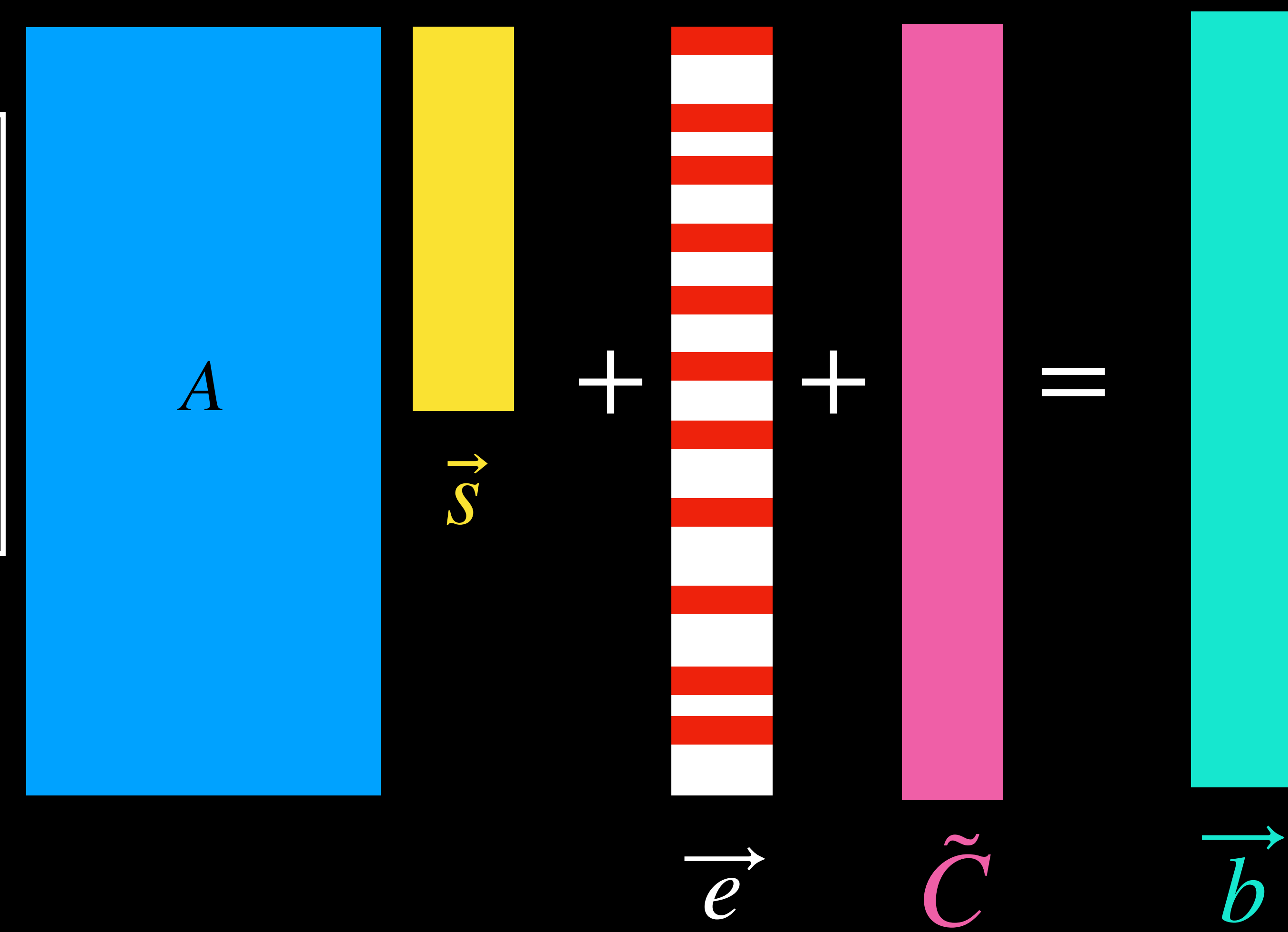
Quadratic in $S = (\vec{s}, 1)^{\otimes 8}$

$|\vec{s}|$ is very small $\Longrightarrow$ $|S|$ is small

Applying LPN

Goal: Find f_x that is:

- Quadratic in short S,
- For most x, $f_x(S) = U_x(\tilde{C})$



Consider:

$$U_x(\tilde{C} + \vec{e}) = U_x(\vec{b} - A\vec{s}) = f_x(S)$$

$\boxed{\textbf{Degree - 16 in } \vec{s}}$ $\boxed{\textbf{Quadratic in } S = (\vec{s}, 1)^{\otimes 8}}$

$|\vec{s}|$ **is very small** $\Longrightarrow$ $|S|$ **is small**

$|\vec{s}| \ll N^{0.10}$ $|S| \ll N^{0.80}$

Applying LPN

Goal: Find f_x that is:
- Quadratic in short S,
- For most x, $f_x(S) = U_x(\tilde{C})$

$$A \vec{s} + \vec{e} + \tilde{C} = \vec{b}$$

Consider:

$$U_x(\tilde{C} + \vec{e}) = U_x(\vec{b} - A\vec{s}) = f_x(S)$$

Degree - 16 in $\vec{s}$

Quadratic in $S = (\vec{s}, 1)^{\otimes 8}$

$|\vec{s}|$ **is very small** $\Longrightarrow$ $|S|$ **is small**

$|\vec{s}| \ll N^{0.10}$

$|S| \ll N^{0.80}$

Use of LPN

Goal: Replace U_x by quadratic functions.

Step 1: Approximate $U_x(\tilde{C})$ by quadratic $f_x(S)$,

$$f_x(S) = U_x(\tilde{C}) \quad \text{for most inputs } x \in [N]$$

Main Idea: Variable change via Random Linear Codes.

Step 2: Error correction via quadratic $\mathsf{Corr}_x(M)$

$$h_x(S, M) = f_x(S) + \mathsf{Corr}_x(M) = U_x(\tilde{C}) \qquad \forall\ x \in [N]$$

Main Idea: Compression via Matrix Factorization.

Use of LPN

Goal: Replace U_x by quadratic functions.

Step 1: Approximate $U_x(\tilde{C})$ by quadratic $f_x(S)$,

$$f_x(S) = U_x(\tilde{C}) \quad \text{for most inputs } x \in [N]$$

Main Idea: Variable change via Random Linear Codes.

Step 2: Error correction via quadratic $\mathsf{Corr}_x(M)$

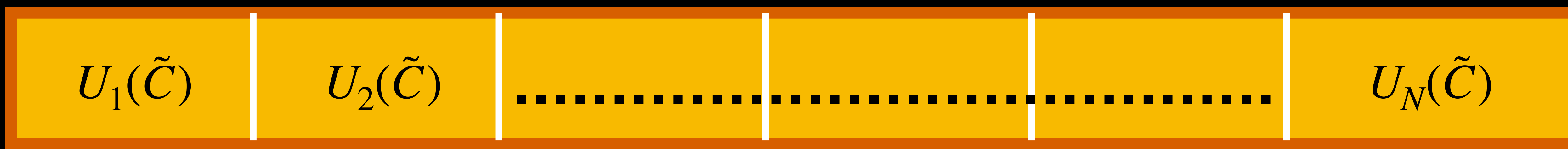
$$h_x(S, M) = f_x(S) + \mathsf{Corr}_x(M) = U_x(\tilde{C}) \qquad \forall\ x \in [N]$$

Main Idea: Compression via Matrix Factorization.

Step 2: Error Correction

Step 2: Error Correction

Target



Step 2: Error Correction

Target

$U_1(\tilde{C})$ | $U_2(\tilde{C})$ | .... | $U_N(\tilde{C})$

Actual

$f_1(S)$ | $f_2(S)$ | .... | $f_N(S)$

Step 2: Error Correction

Target

$U_1(\tilde{C})$	$U_2(\tilde{C})$	....	$U_N(\tilde{C})$

Actual

-

$f_1(S)$	$f_2(S)$	....	$f_N(S)$

Correction vector = Target - Actual

Step 2: Error Correction

Target

$U_1(\tilde{C})$	$U_2(\tilde{C})$	....	....	....	$U_N(\tilde{C})$

Actual

-

$f_1(S)$	$f_2(S)$	....	....	....	$f_N(S)$

Correction vector = **Target** - **Actual**

0		0	0	0	0

Takeaway: Correction vector is sparse!!

Amortization

Previously we showed that for any circuit C

Map: $\tilde{C} \to P, S$

Size = $\tilde{O}(N^{0.99})$

Time = $\tilde{O}(N)$

Requires LWE [GKPVZ 13, BV 15, AJ 15, LPST 16, BNPW 16]

Amortization

Previously we showed that for any circuit C

Map: $\tilde{C} \to P, S$

Size = $\tilde{O}(N^{0.99})$

Time = $\tilde{O}(N)$

Requires LWE [GKPVZ 13, BV 15, AJ 15, LPST 16, BNPW 16]

Main Lemma:

Map: $(\tilde{C}_1, \ldots, \tilde{C}_k) \to (P_1, P_2, \ldots, P_k, S_1, \ldots, S_k)$

Time = $\tilde{O}(Nk^{1-\epsilon} + k^c)$
Sublinear in Nk

Time Succinctness

Show that this suffices for $i\mathcal{O}$

Efficient circuit implementations for special RAM prorgams such as lookups and sorting.

Thank you!

Thank you!

FHE directly from these assumptions?

Thank you!

FHE directly from these assumptions?

Complexity/algorithm questions:

Reductions to LWE/GAP-SVP for these assumptions?

Thank you!

FHE directly from these assumptions?

Complexity/algorithm questions:

Reductions to LWE/GAP-SVP for these assumptions?

LPN/PRG: Build PKE/ show that they are in CoAM