



# How to Shuffle in Public

Ben Adida

Harvard

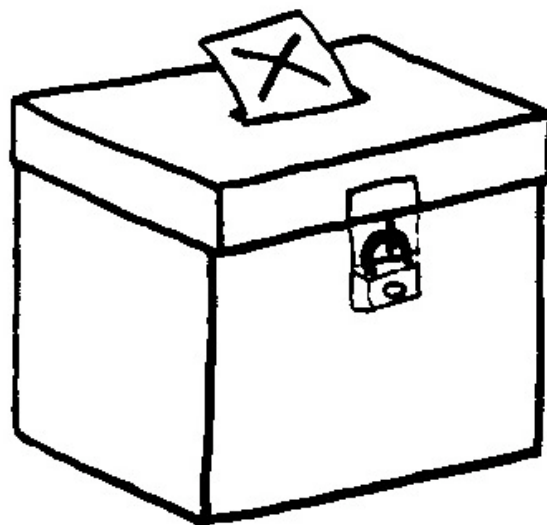
(work done at MIT)

Douglas Wikström

ETH Zürich

**TCC 2007**

*February 24th, 2007*



# How to Shuffle in Public

Ben Adida

Harvard

(work done at MIT)

Douglas Wikström

ETH Zürich

**TCC 2007**

*February 24th, 2007*



Rogers precinct, with more than 100 percent voter turnout, alarmed both of them.

Rogers precinct, with more than 100 percent voter turnout, alarmed both of them.

**Thief grabs voting machine from election official's car**

By ROGER H. AYLWORTH - Staff Writer

Article Launched:11/07/2006 12:00:00 AM PST

Rogers precinct, with more than 100 percent voter turnout, alarmed both of them.

**Thief grabs voting machine from election official's car**

By ROGER H. AYLWORTH - Staff Writer

Article Launched:11/07/2006 12:00:00 AM PST

Last Updated: November 7, 2006 - 2:19 PM EST

**Voter smashes touch-screen machine in Allentown**



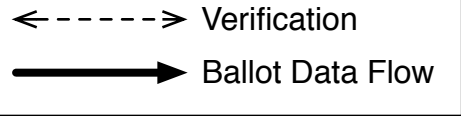
Last Updated: November 7, 2006 - 2:19 PM EST

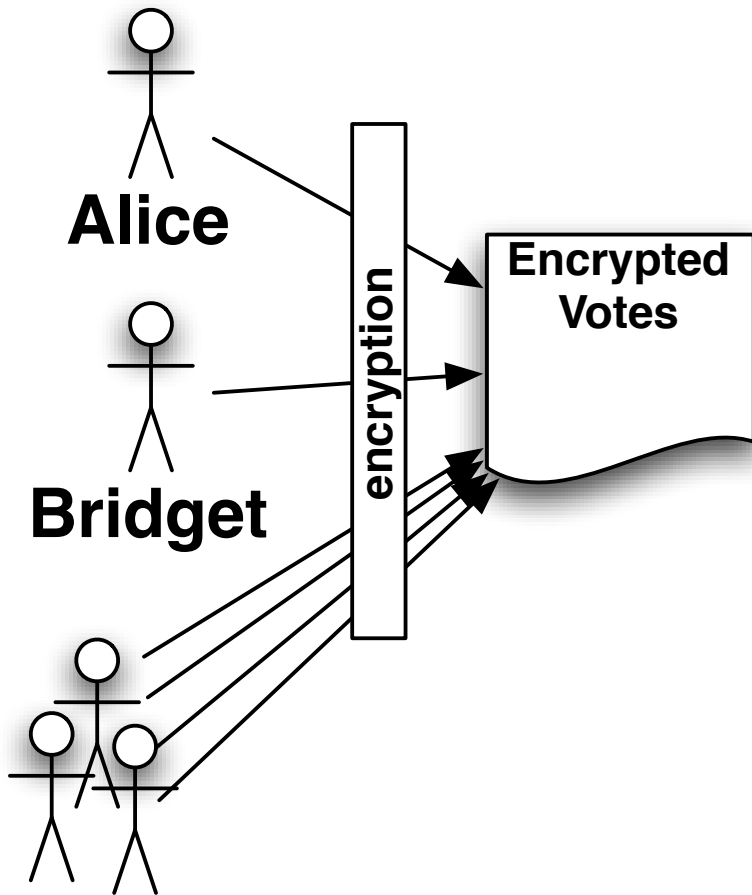
## **Voter smashes touch-screen machine in Allentown**

# Voting with Cryptographic Verification

- (1) **Alice** verifies **her vote**.
- (2) **Everyone** verifies **tallying**.
- (3) Alice **cannot be coerced** by Eve.

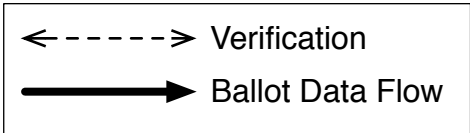
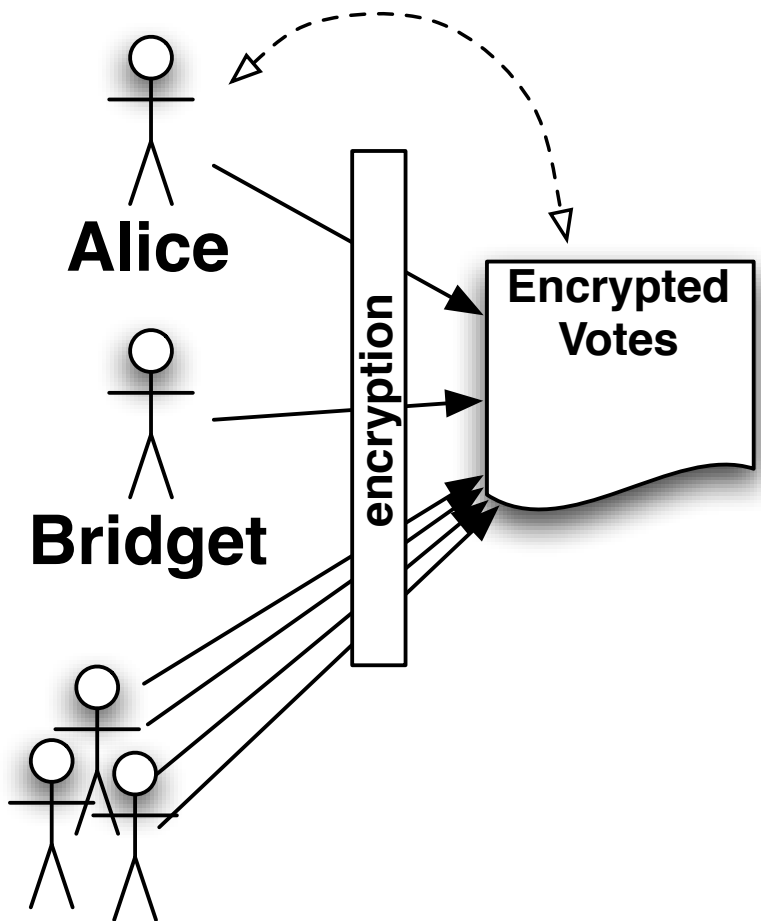
[Chaum81], [Benaloh85], [PIK93], [BenalohTuinstra92], [SK94], [Abe98],  
[CFSY96], [CGS97], [BFPSP2001], [Neff2001], [FS2001], [Chaum2004],  
[Neff2004], [Ryan2004], [Chaum2005], [W2004],  
[W2005], [WG2005], [MN2006]

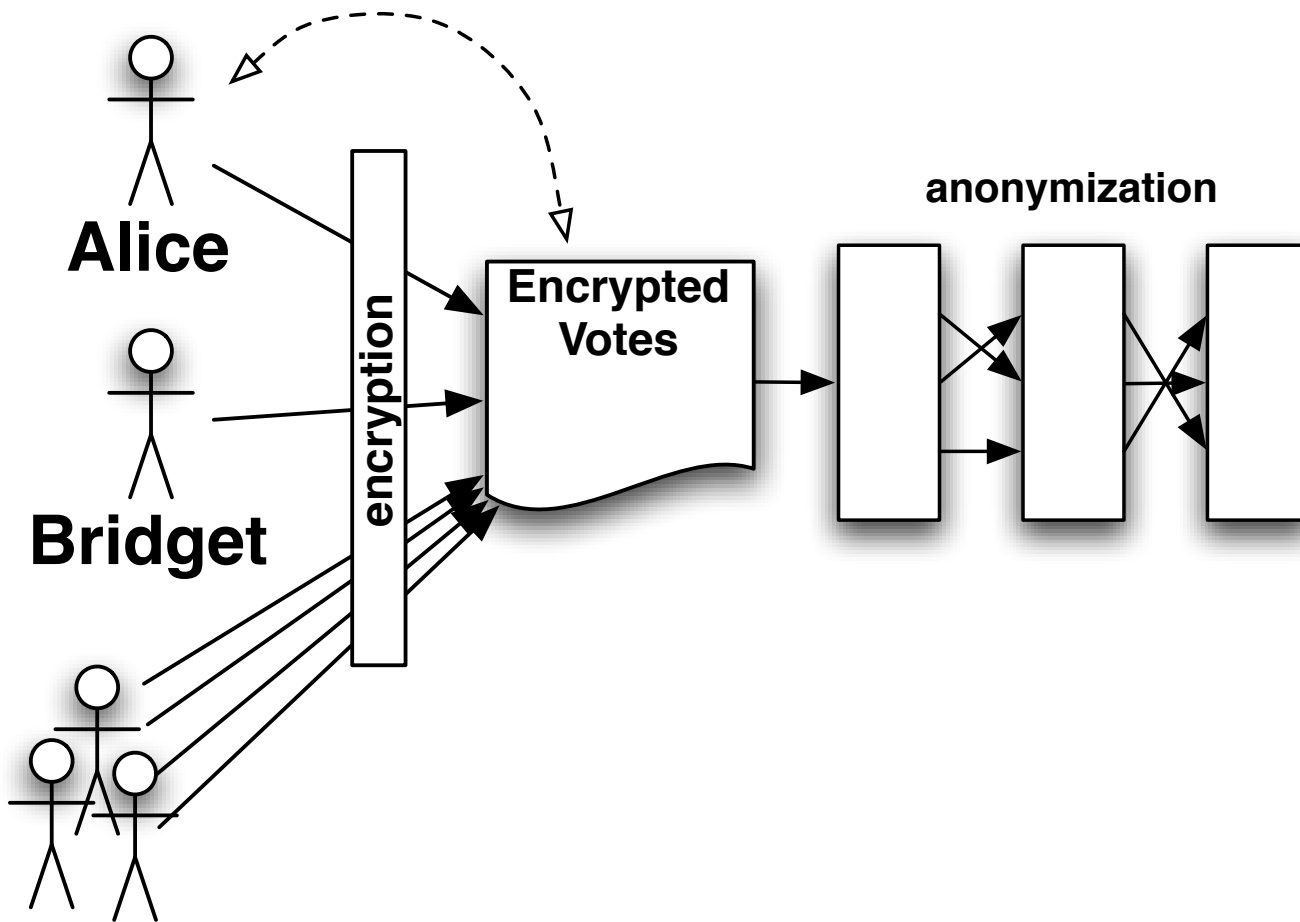




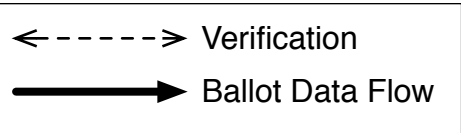
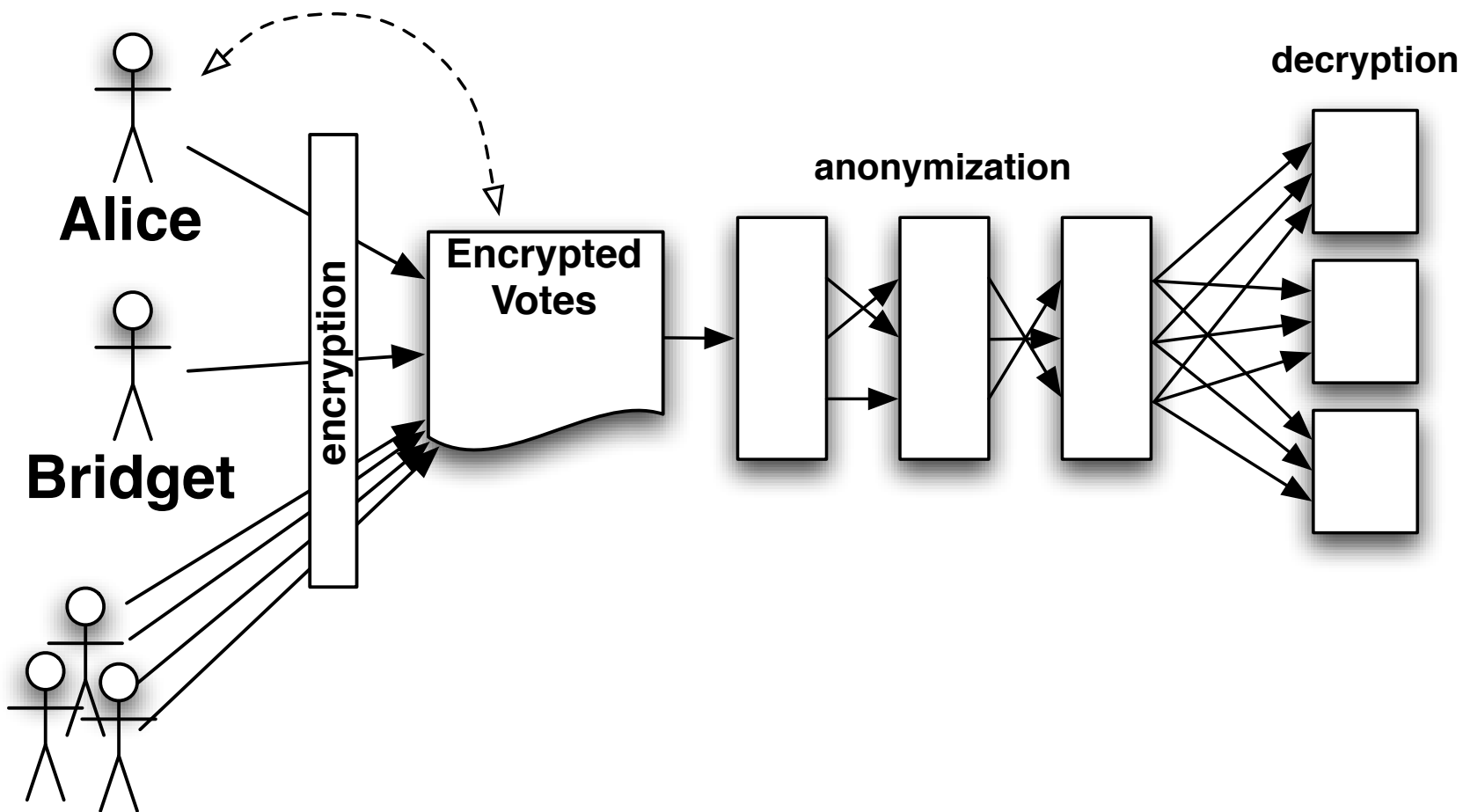
← - - - - - > Verification

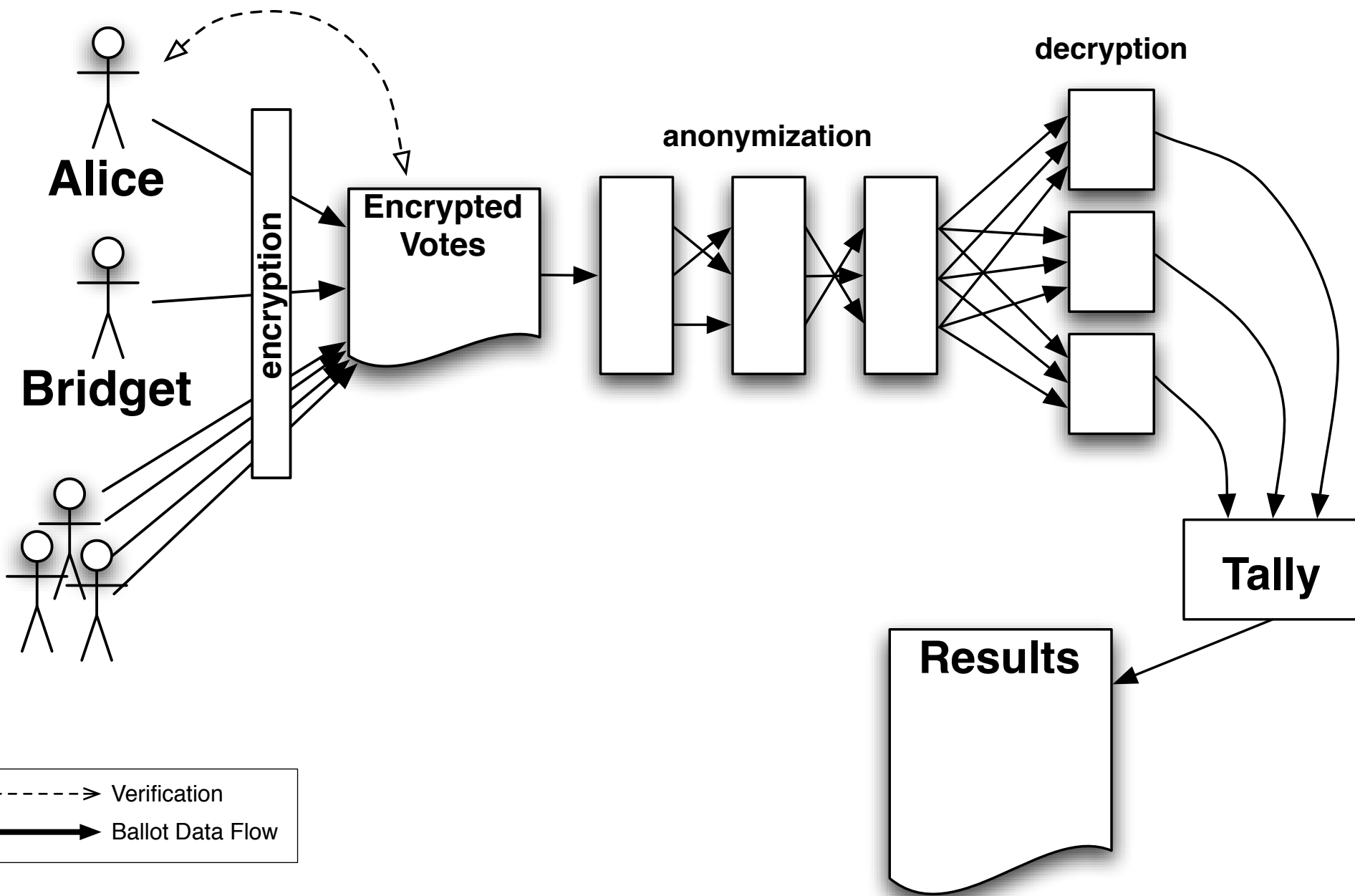
→ Ballot Data Flow



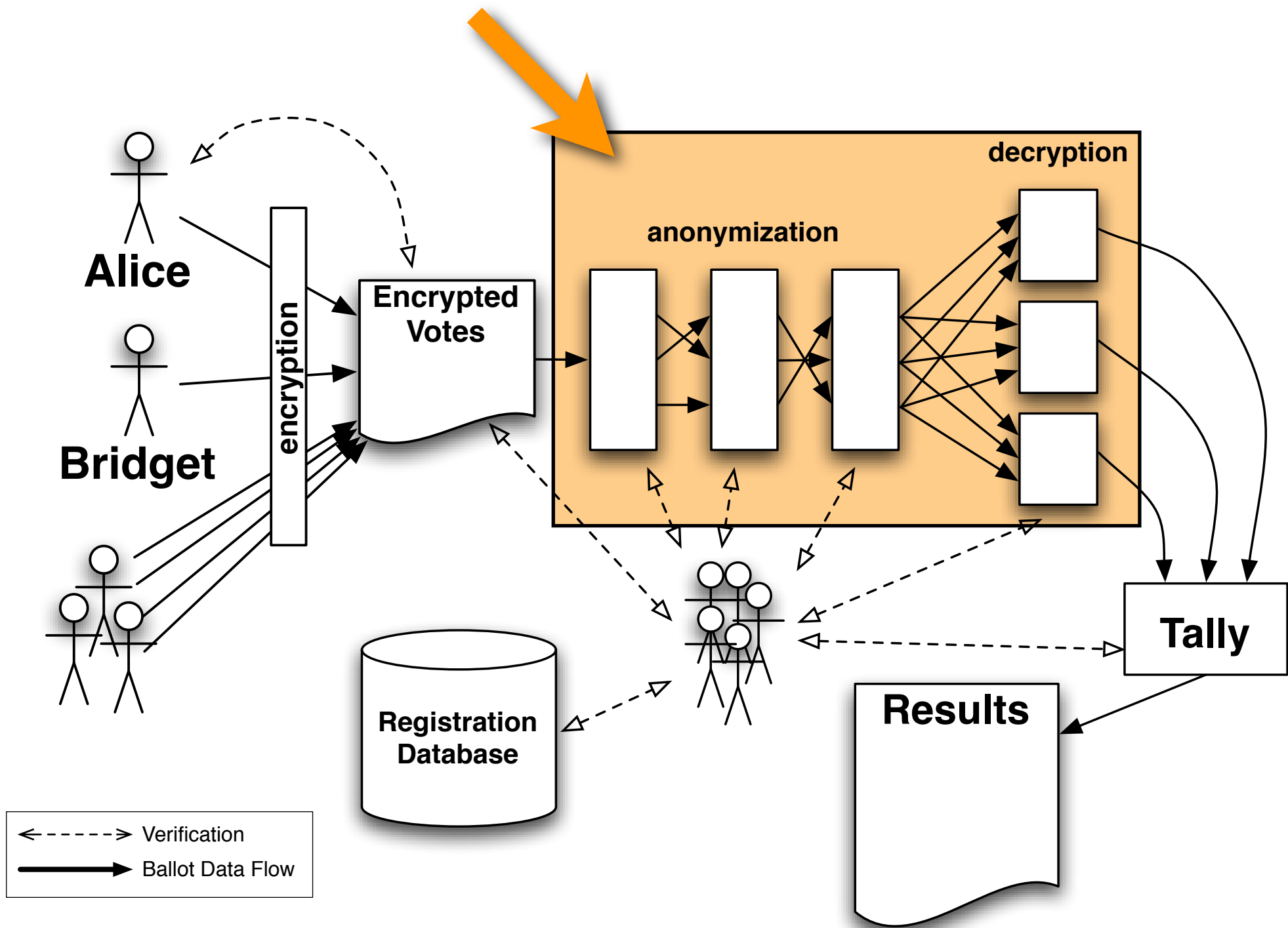


← - - - - - > Verification  
— — — — —> Ballot Data Flow



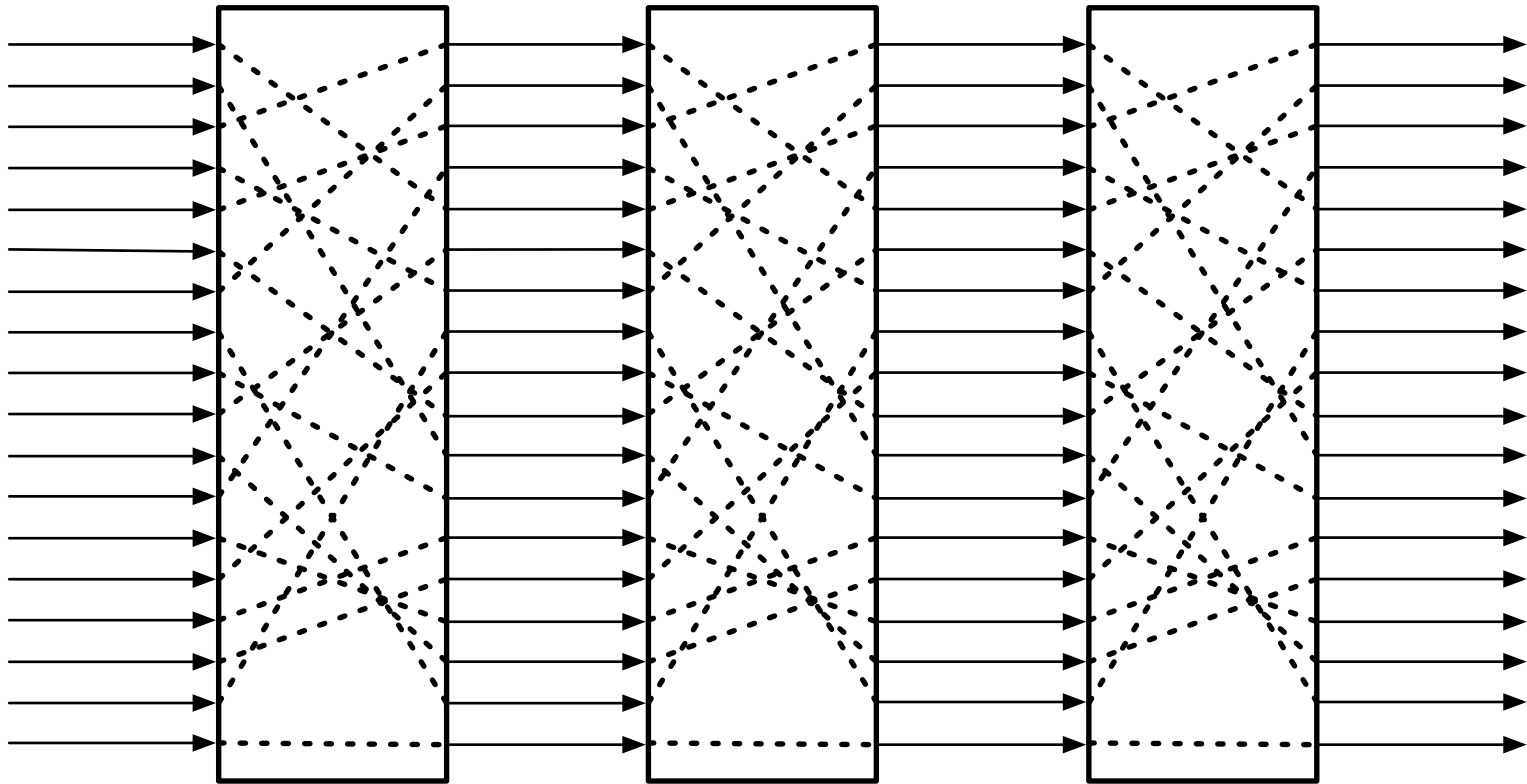






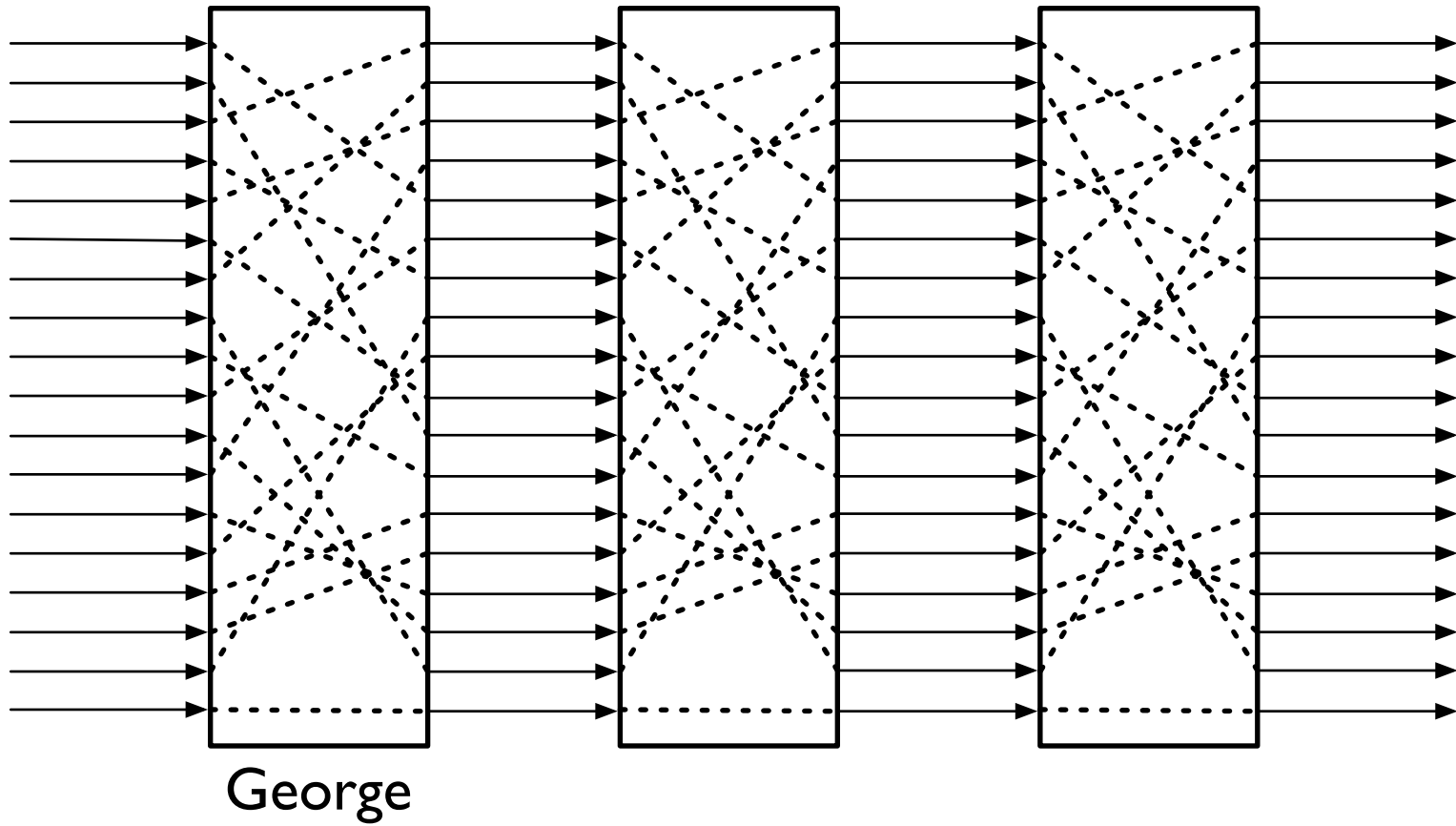
[Chaum81, PP90,  
PIK93, SK94,..]

# Mixnet



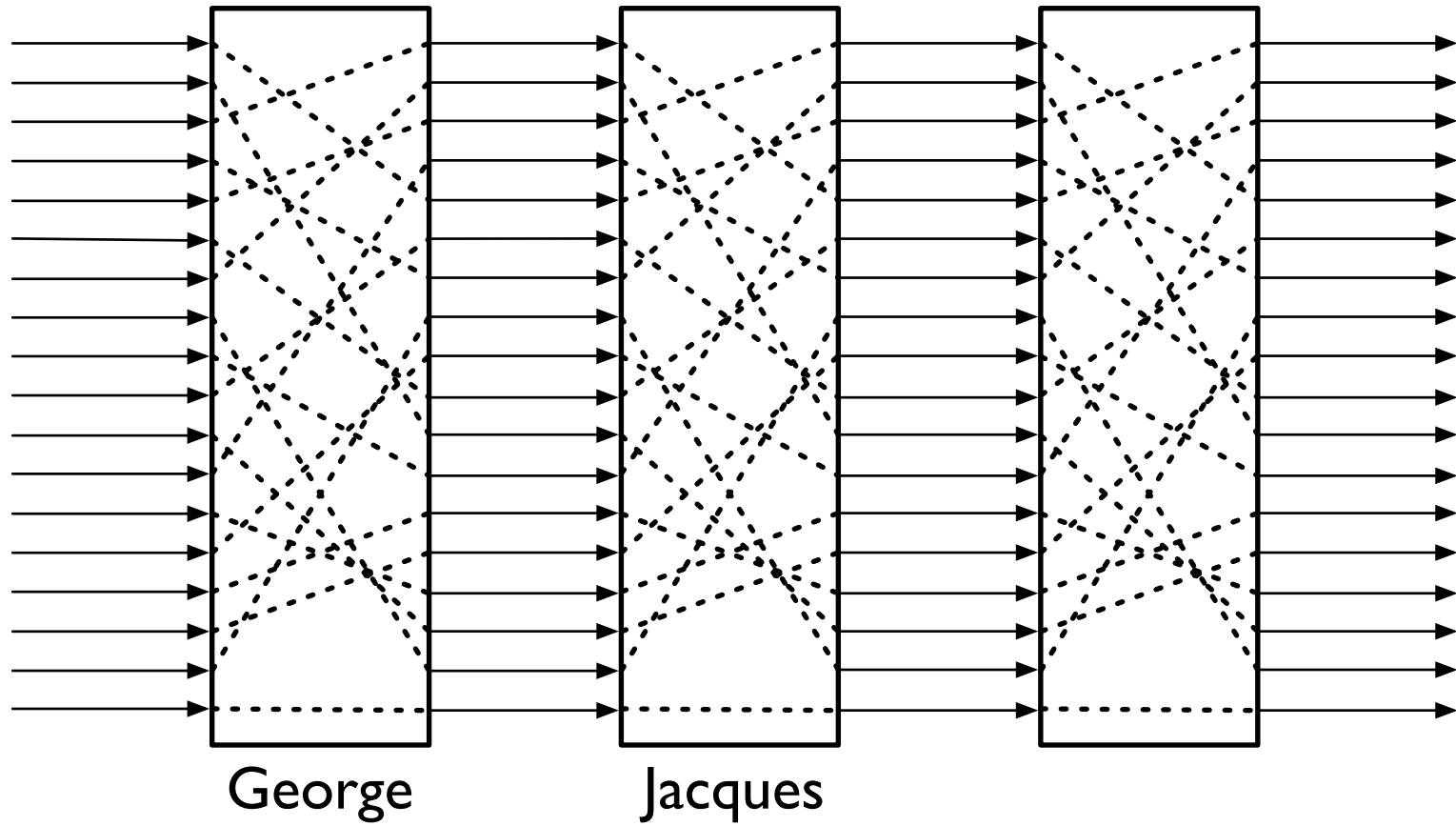
[Chaum81, PP90,  
PIK93, SK94,..]

# Mixnet



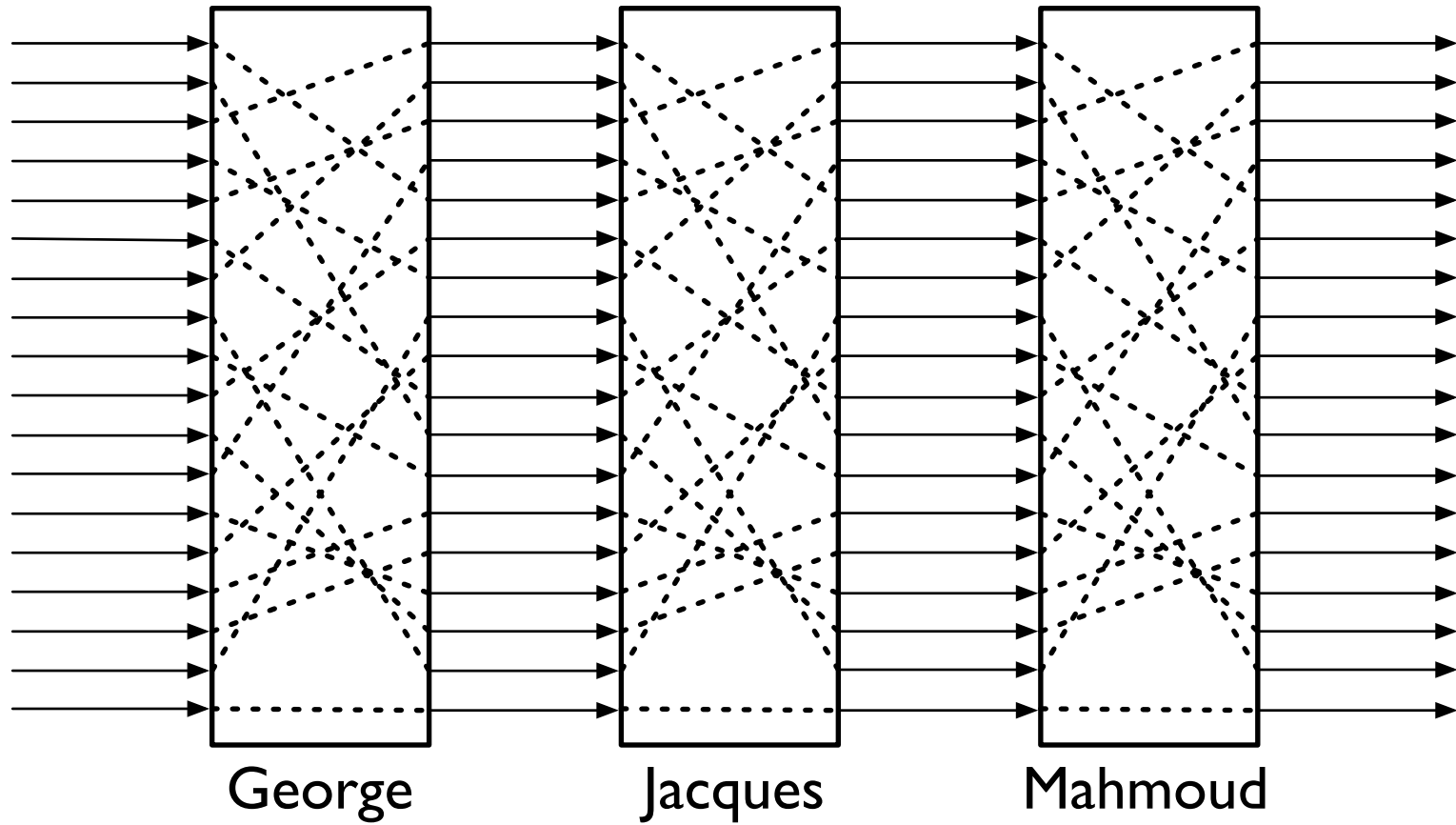
[Chaum81, PP90,  
PIK93, SK94,..]

# Mixnet

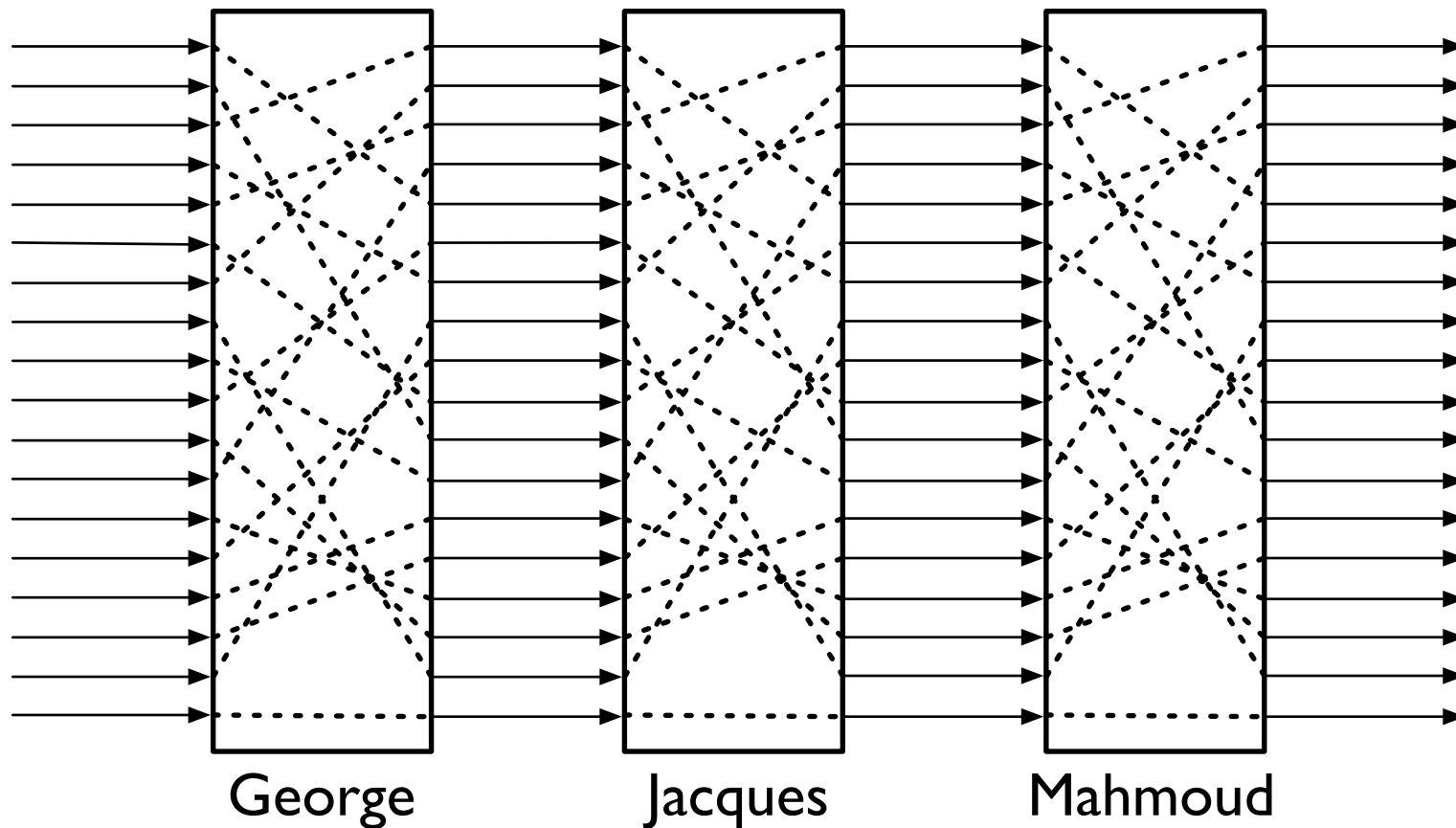


[Chaum81, PP90,  
PIK93, SK94,..]

# Mixnet

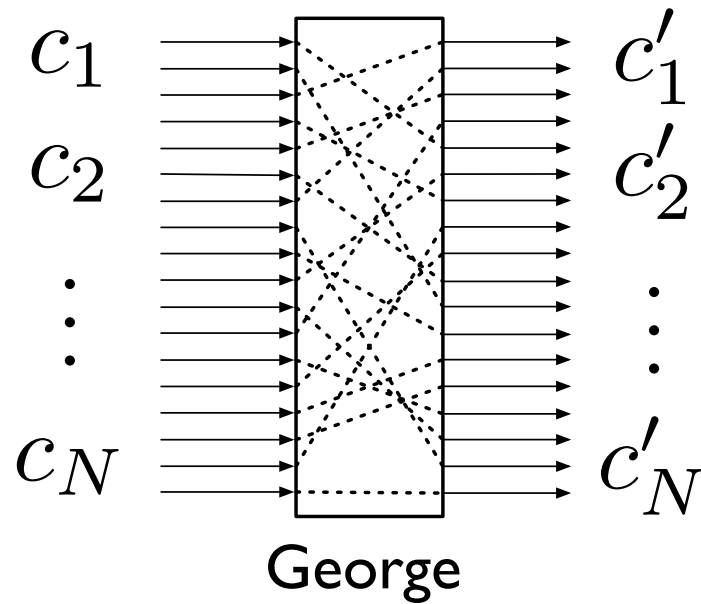


# Mixnet



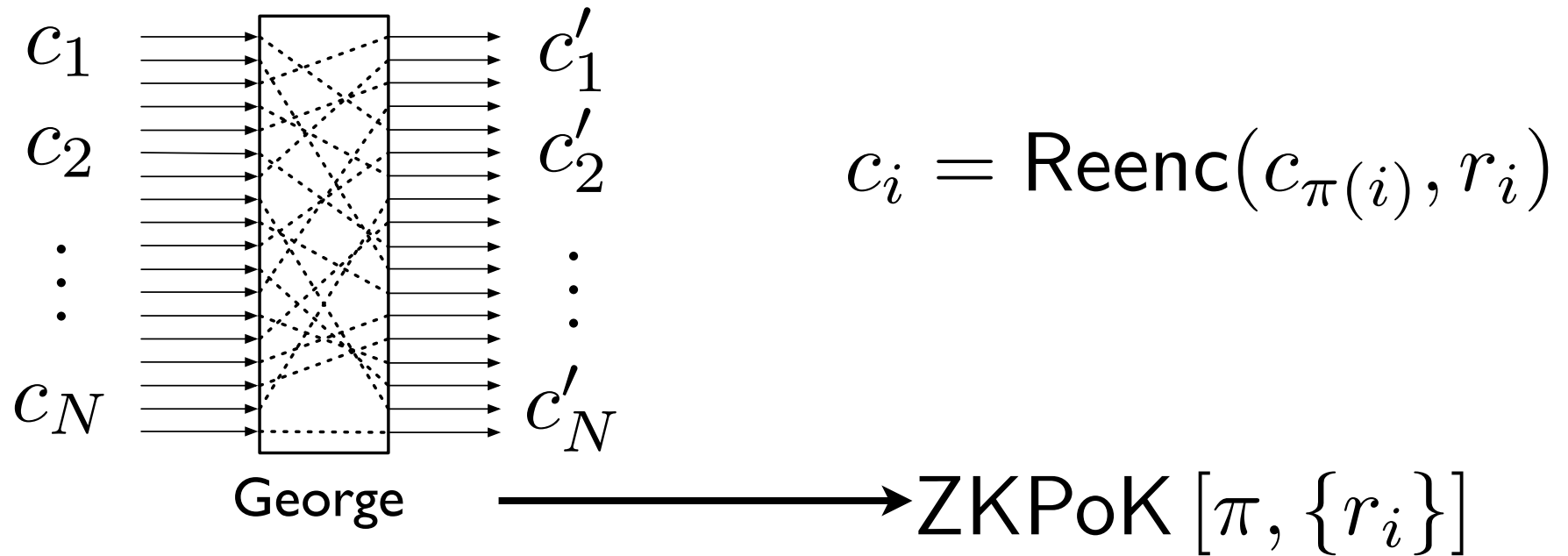
Each mix server shuffles and reencrypts  
(or partially decrypts) inputs.

# Proving the Mix

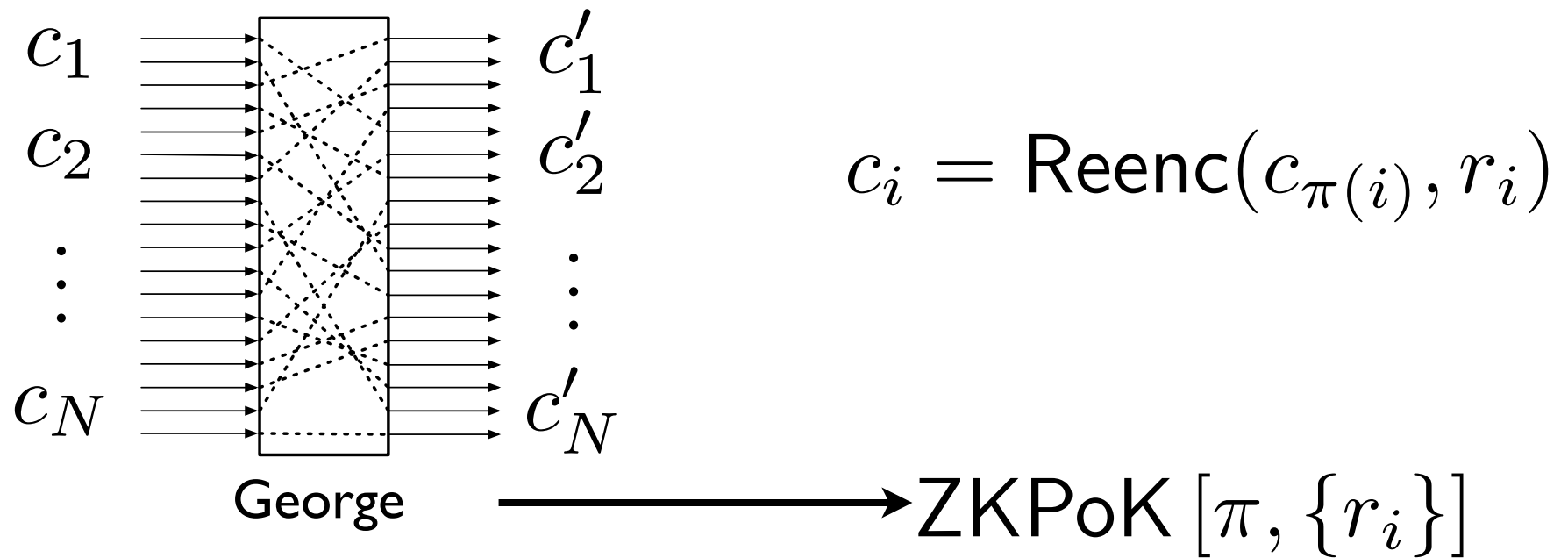


$$c_i = \text{Reenc}(c_{\pi(i)}, r_i)$$

# Proving the Mix

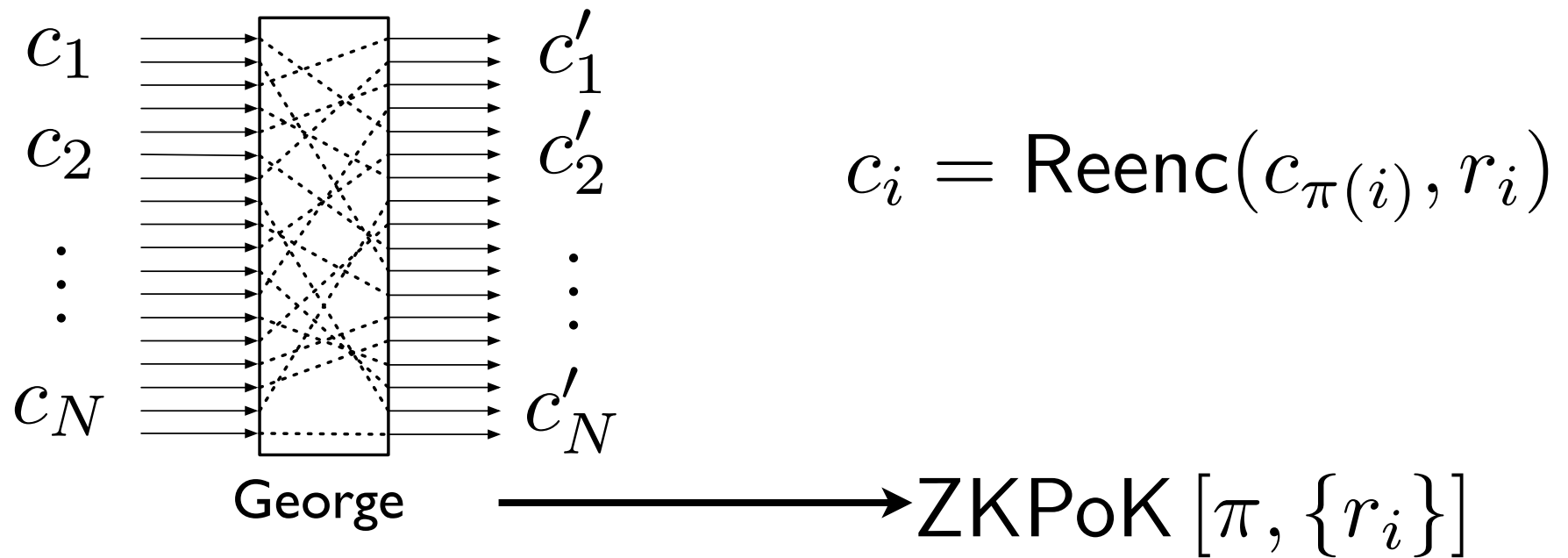


# Proving the Mix



**George  
can't cheat.**

# Proving the Mix



**George  
can't cheat.**

**$\pi$  and  $\{r_i\}$   
stay private.**

# Private vs. Public

**Private**

**Public**

$c_1$

$c_2$

$\vdots$

$c_N$

$c'_1$

$c'_2$

$\vdots$

$c'_N$

# Private vs. Public

**Private**

$\pi, \{r'_i\}$

**Public**

$c_1$

$c_2$

$\vdots$

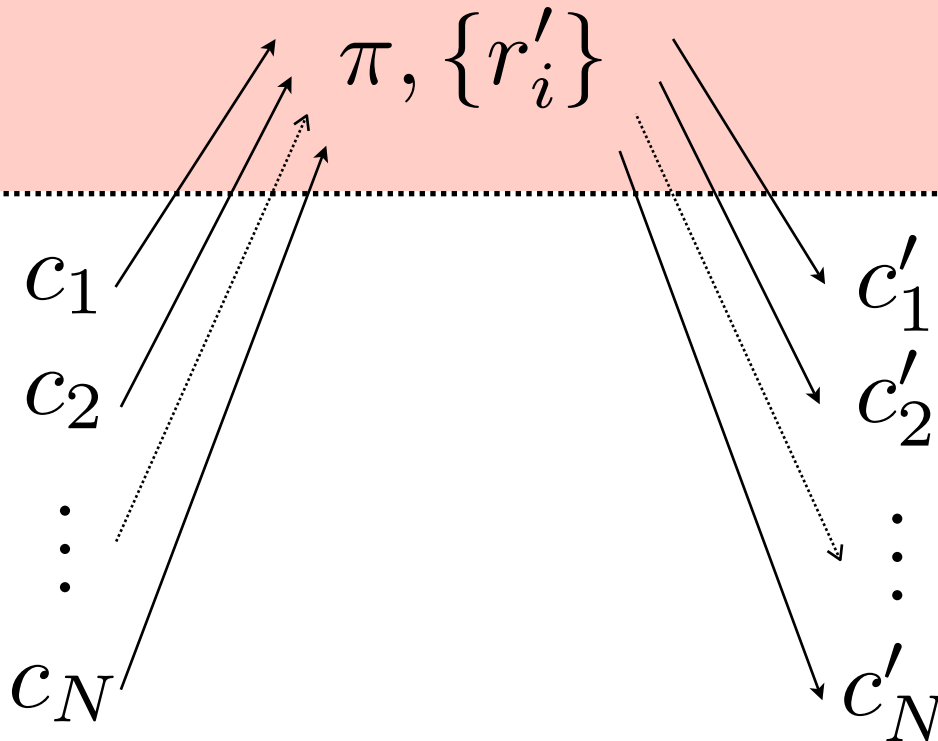
$c_N$

$c'_1$

$c'_2$

$\vdots$

$c'_N$



# Private vs. Public

**Private**

$\pi, \{r'_i\}$

**Public**

$c_1$

$c_2$

$\vdots$

$c_N$

$c'_1$

$c'_2$

$\vdots$

$c'_N$

what if we could  
replace the **private** mixnet  
with a **public** program?

# Private vs. Public

**Private**

$$\pi, \{r'_i\}$$

**Public**

$c_1$

$c'_1$

$c_2$

$c'_2$

$\vdots$

$\vdots$

$c_N$

$c'_N$

what if we could  
replace the **private** mixnet  
with a **public** program?

# Private vs. Public

**Private**

$\pi, \{r'_i\}$

**Public**

$c_1$

$c'_1$

$c_2$

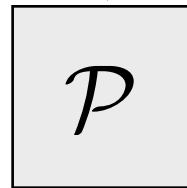
$c'_2$

$\vdots$

$\vdots$

$c_N$

$c'_N$



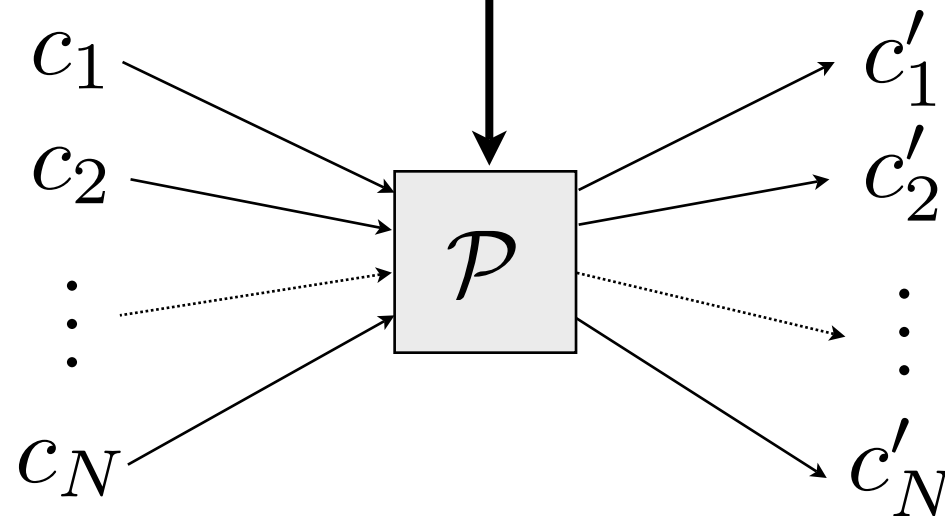
what if we could  
replace the **private** mixnet  
with a **public** program?

# Private vs. Public

**Private**

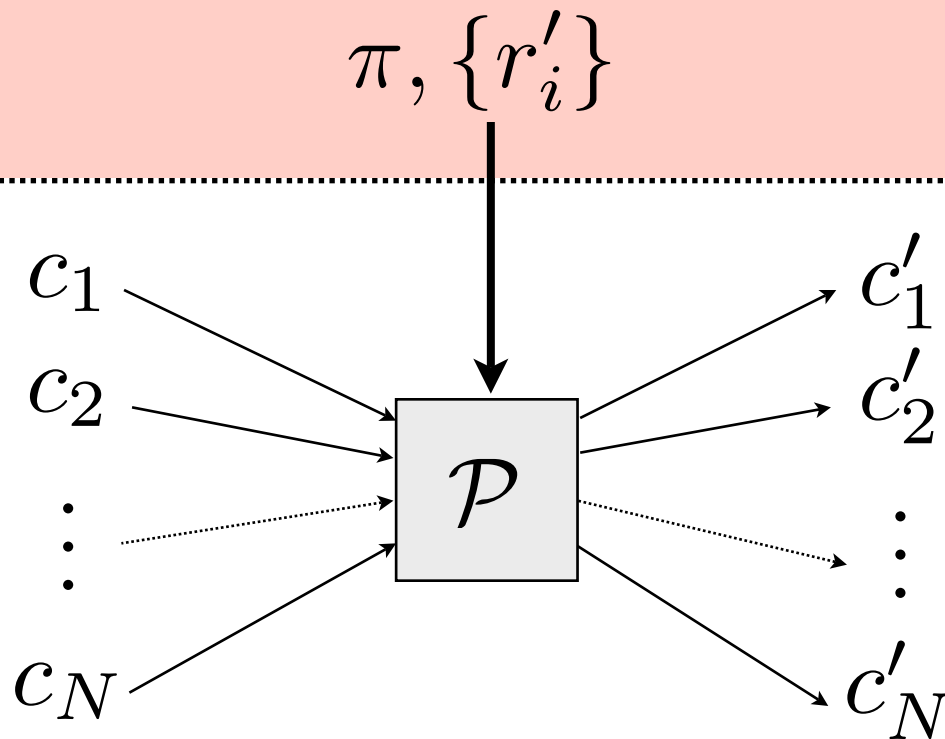
$\pi, \{r'_i\}$

**Public**



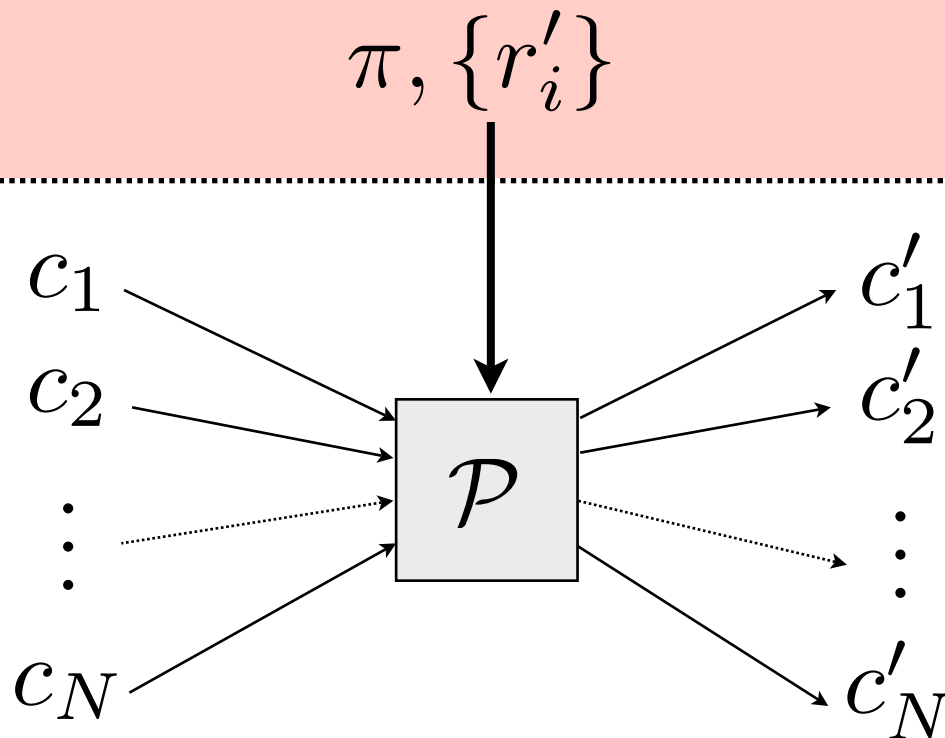
what if we could  
replace the **private** mixnet  
with a **public** program?

# So What?



- ➔ **public program**  
anyone can run it
- ➔ **pre-proven**  
all proofs before mixing
- ➔ **interesting!**  
to obfuscate such  
a functionality

# So What?



- **public program**  
anyone can run it
- **pre-proven**  
all proofs before mixing
- **interesting!**  
to obfuscate such  
a functionality

Can it really be done?  
[BGIRSVY2001, GT-K2005]

# Our Results

- \* horribly inefficient generic construction
- \* (somewhat) efficient public-shuffle constructions  
*using either BGN or Paillier cryptosystem*
- \* (somewhat) efficient distributed generation  
of a public shuffle program
- \* a new class of obfuscatable programs  
under [OS2005] or [BGIRSVY2001]

# A Generic Construction

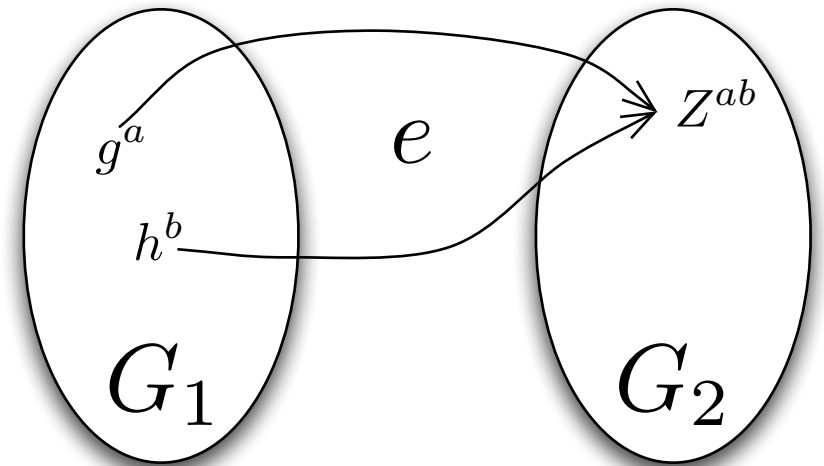
- \* Two homomorphic cryptosystems
- \* Message Space of first “contains”  
Ciphertext Space of second
- \* Inefficient and less interesting  
than specific constructions (esp. decryption)

# BGN Cryptosystem

$G_1, G_2$ , order  $n = p_1 p_2$

$e : G_1 \times G_1 \rightarrow G_2$

$e(g^a, h^b) = e(g, h)^{ab}$



# BGN Cryptosystem

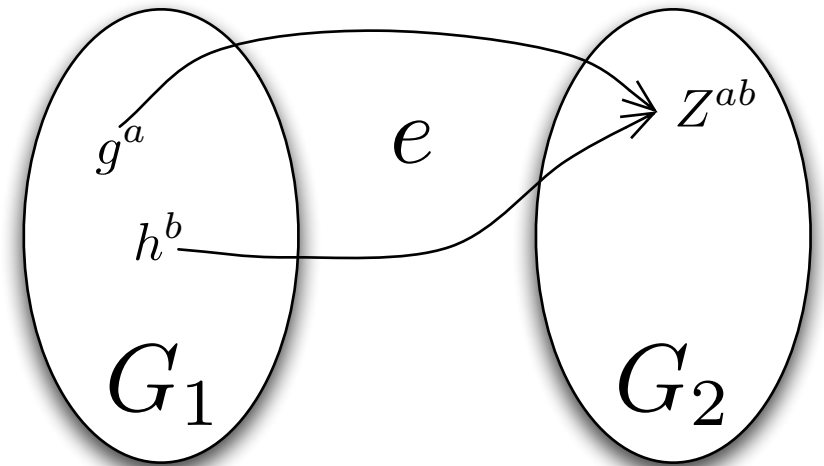
$G_1, G_2$ , order  $n = p_1 p_2$

$e : G_1 \times G_1 \rightarrow G_2$

$e(g^a, h^b) = e(g, h)^{ab}$

$pk = (n, g, h = u^{p_1})$

$sk = p_2$



# BGN Cryptosystem

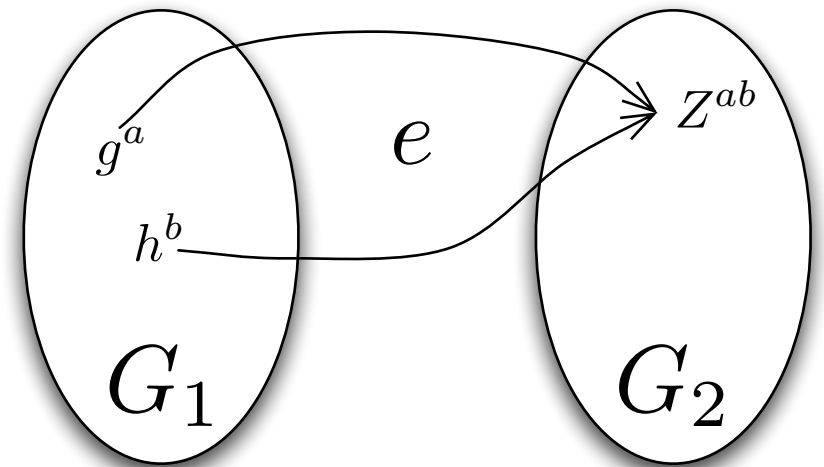
$G_1, G_2$ , order  $n = p_1 p_2$

$e : G_1 \times G_1 \rightarrow G_2$

$e(g^a, h^b) = e(g, h)^{ab}$

$pk = (n, g, h = u^{p_1})$

$Enc_{pk}(m) = g^m h^r$



$sk = p_2$

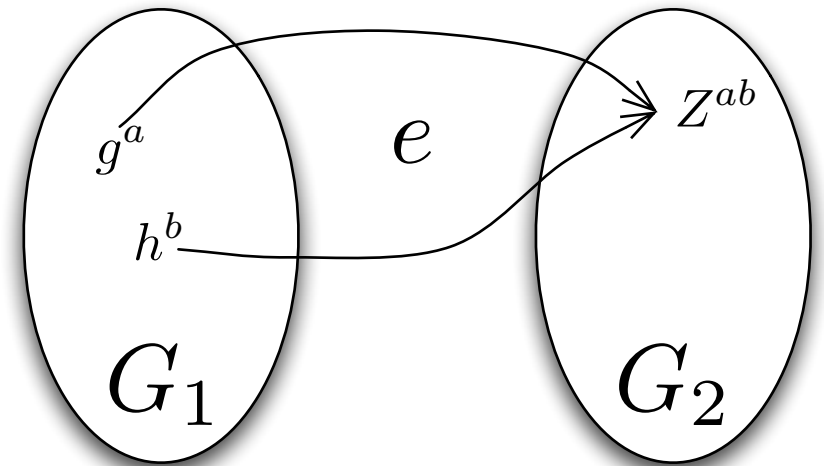
$Dec_{sk}(c) = \log_{g^{p_2}}(c^{p_2})$

# BGN Cryptosystem

$G_1, G_2$ , order  $n = p_1 p_2$

$e : G_1 \times G_1 \rightarrow G_2$

$e(g^a, h^b) = e(g, h)^{ab}$



$pk = (n, g, h = u^{p_1})$

$sk = p_2$

$\text{Enc}_{pk}(m) = g^m h^r$

$\text{Dec}_{sk}(c) = \log_{g^{p_2}}(c^{p_2})$

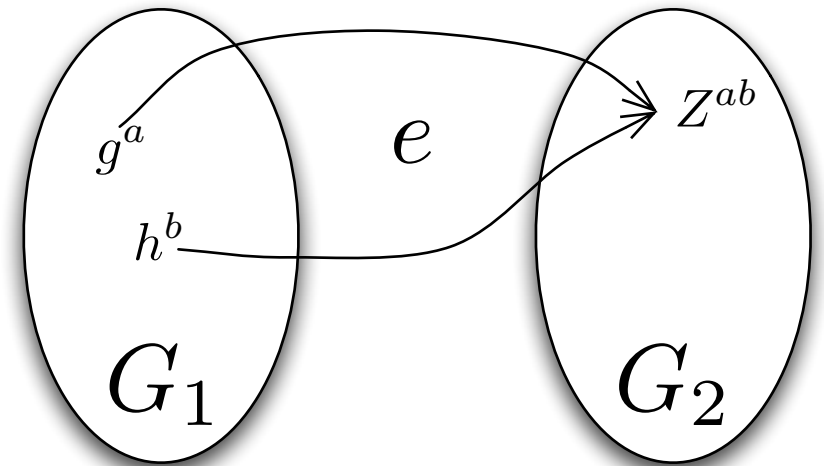
$\text{Enc}_{pk}(m_1) \cdot \text{Enc}_{pk}(m_2) = \text{Enc}_{pk}(m_1 + m_2)$

# BGN Cryptosystem

$G_1, G_2$ , order  $n = p_1 p_2$

$e : G_1 \times G_1 \rightarrow G_2$

$e(g^a, h^b) = e(g, h)^{ab}$



$pk = (n, g, h = u^{p_1})$

$sk = p_2$

$\text{Enc}_{pk}(m) = g^m h^r$

$\text{Dec}_{sk}(c) = \log_{g^{p_2}}(c^{p_2})$

$\text{Enc}_{pk}(m_1) \cdot \text{Enc}_{pk}(m_2) = \text{Enc}_{pk}(m_1 + m_2)$

$e(\text{Enc}_{pk}(m_1), \text{Enc}_{pk}(m_2)) = \text{Enc}_{pk}(m_1 \cdot m_2)$

# Oblivious Cancellation / Selection

$$\text{Enc}_{pk}(m) \otimes \text{Enc}_{pk}(0) = \text{Enc}_{pk}(0)$$

$$\text{Enc}_{pk}(m) \otimes \text{Enc}_{pk}(1) = \text{Enc}_{pk}(m)$$

$\text{Enc}_{pk}(0)$  and  $\text{Enc}_{pk}(1)$   
are indistinguishable

# Oblivious Cancellation / Selection

$$\text{Enc}_{pk}(m) \otimes \text{Enc}_{pk}(0) = \text{Enc}_{pk}(0)$$

$$\text{Enc}_{pk}(m) \otimes \text{Enc}_{pk}(1) = \text{Enc}_{pk}(m)$$

$\text{Enc}_{pk}(0)$  and  $\text{Enc}_{pk}(1)$   
are indistinguishable

Clearly Useful for PIR and OT [BGN2005].  
In fact, it's ***more*** powerful still.

# Matrix Multiplication

$$\begin{bmatrix} a_{11} & \dots & a_{1l} \\ a_{21} & \dots & a_{2l} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nl} \end{bmatrix} \times \begin{bmatrix} b_{11} & \dots & b_{1n} \\ b_{21} & \dots & b_{2n} \\ \vdots & \ddots & \vdots \\ b_{l1} & \dots & b_{ln} \end{bmatrix} = \begin{bmatrix} c_{11} & \dots & c_{1n} \\ c_{21} & \dots & c_{2n} \\ \vdots & \ddots & \vdots \\ c_{m1} & \dots & c_{mn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^l a_{ik} b_{kj}$$

Degree is exactly 2:  
only **one** multiplication!

# Homomorphic MM

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \end{bmatrix} = \begin{bmatrix} m_3 \\ m_1 \\ m_5 \\ m_2 \\ m_4 \end{bmatrix}$$

The diagram illustrates a homomorphic matrix multiplication. On the left, a 5x5 matrix with blue shading and a 5x1 column vector with blue shading are shown. The matrix has 1s at positions (1,3), (2,1), (3,5), (4,2), and (5,4). The column vector contains elements  $m_1$  through  $m_5$ . An  $\otimes$  symbol is between them. An equals sign follows, leading to a 5x1 column vector with red shading containing elements  $m_3$ ,  $m_1$ ,  $m_5$ ,  $m_2$ , and  $m_4$  in order from top to bottom.

# Homomorphic MM

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \end{bmatrix} = \begin{bmatrix} m_3 \\ m_1 \\ m_5 \\ m_2 \\ m_4 \end{bmatrix}$$

Homomorphic matrix multiplication  
by an encrypted permutation matrix

=

Shuffling in Public!

# Shuffling in Public

**Private**

$\pi$

**Public**

# Shuffling in Public

**Private**

$$\pi \longrightarrow \begin{bmatrix} 0 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 0 \end{bmatrix}$$

**Public**

# Shuffling in Public

**Private**

$$\pi \rightarrow \begin{bmatrix} 0 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 0 \end{bmatrix} \begin{matrix} \\ \\ \end{matrix} \{r_i\}$$

**Public**

$$\begin{bmatrix} \boxed{0} & \dots & \boxed{1} \\ \vdots & \ddots & \vdots \\ \boxed{1} & \dots & \boxed{0} \end{bmatrix}$$

# Shuffling in Public

**Private**

$$\pi \rightarrow \begin{bmatrix} 0 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 0 \end{bmatrix} \begin{matrix} \\ \\ \\ \end{matrix} \{r_i\}$$

**Public**

$$\begin{matrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{matrix} \begin{bmatrix} \boxed{0} & \dots & \boxed{1} \\ \vdots & \ddots & \vdots \\ \boxed{1} & \dots & \boxed{0} \end{bmatrix}$$

# Shuffling in Public

**Private**

$$\pi \longrightarrow \begin{bmatrix} 0 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 0 \end{bmatrix} \begin{matrix} \\ \\ \\ \end{matrix} \{r_i\}$$

**Public**

$$\begin{matrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{matrix} \otimes \begin{bmatrix} \boxed{0} & \dots & \boxed{1} \\ \vdots & \ddots & \vdots \\ \boxed{1} & \dots & \boxed{0} \end{bmatrix} = \begin{matrix} c'_1 \\ c'_2 \\ \vdots \\ c'_N \end{matrix}$$

# Shuffling in Public

**Private**

$$\pi \rightarrow \begin{bmatrix} 0 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 0 \end{bmatrix} \begin{matrix} \\ \\ \end{matrix} \{r_i\}$$

**Public**

$$\begin{matrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{matrix} \otimes \begin{bmatrix} 0 & \dots & 1 \\ \vdots & \mathcal{P} & \vdots \\ 1 & \dots & 0 \end{bmatrix} = \begin{matrix} c'_1 \\ c'_2 \\ \vdots \\ c'_N \end{matrix}$$

# Why Did We Succeed?

0	0	1	0	0
1	0	0	0	0
0	0	0	0	1
0	1	0	0	0
0	0	0	1	0

- [BGIRSVY2001, GT-K2005] tell us **generic** obfuscation is hard.
- Functionality defined on the plaintexts; we're only dealing with ciphertexts

“under the covers of encryption”

# Why Did We Succeed?

0	0	1	0	0
1	0	0	0	0
0	0	0	0	1
0	1	0	0	0
0	0	0	1	0

- [BGIRSVY2001, GT-K2005] tell us **generic** obfuscation is hard.
- Functionality defined on the plaintexts; we're only dealing with ciphertexts

“under the covers of encryption”

We don't know that this is really a permutation matrix!  
We must **prove** correct functionality.

# Proving the Matrix

is  $\begin{bmatrix} \square & \dots & \square \\ \vdots & \ddots & \vdots \\ \square & \dots & \square \end{bmatrix}$  an encrypted permutation matrix?

- Use Proof of Partial Knowledge [CDS94] to show that each element is either 0 or 1.
- Homomorphically compute the row and column sums and prove that they're all equal to 1.
- $N^2$  proofs. Not so great.

# Proving the Matrix (better)

$$\begin{bmatrix} \square & \dots & \square \\ \vdots & \ddots & \vdots \\ \square & \dots & \square \end{bmatrix}$$

# Proving the Matrix (better)

$$\begin{bmatrix} \square & \dots & \square \\ \vdots & \ddots & \vdots \\ \square & \dots & \square \end{bmatrix} \otimes \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix}$$

# Proving the Matrix (better)

$$\begin{bmatrix} \square & \dots & \square \\ \vdots & \ddots & \vdots \\ \square & \dots & \square \end{bmatrix} \otimes \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ c_N \end{bmatrix}$$

# Proving the Matrix (better)

$$\begin{bmatrix} \square & \dots & \square \\ \vdots & \ddots & \vdots \\ \square & \dots & \square \end{bmatrix} \otimes \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ c_N \end{bmatrix}$$

- Proof by Random Vector Challenge

# Proving the Matrix (better)

$$\begin{bmatrix} \square & \dots & \square \\ \vdots & \ddots & \vdots \\ \square & \dots & \square \end{bmatrix} \otimes \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ c_N \end{bmatrix}$$

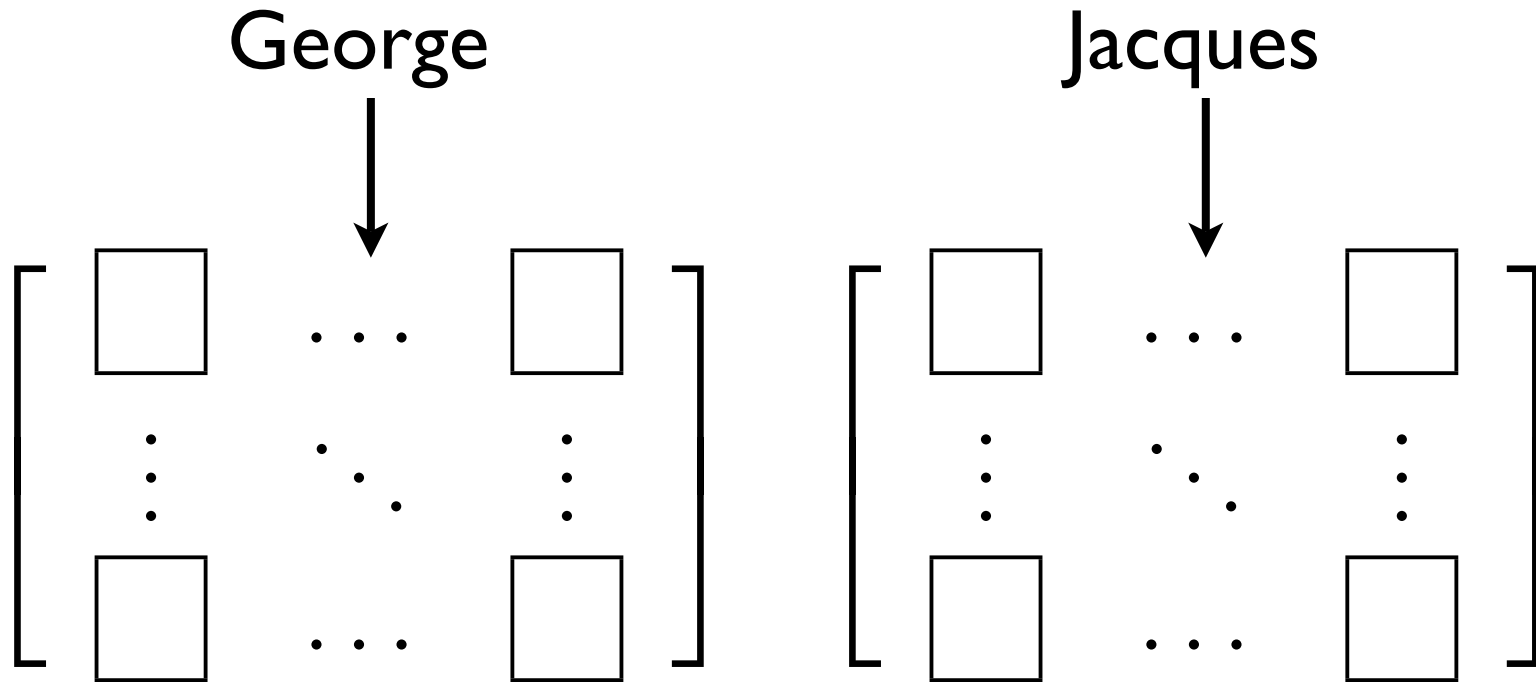
- Proof by Random Vector Challenge
- Neff:  $O(N)$  proof.

# Proving the Matrix (better)

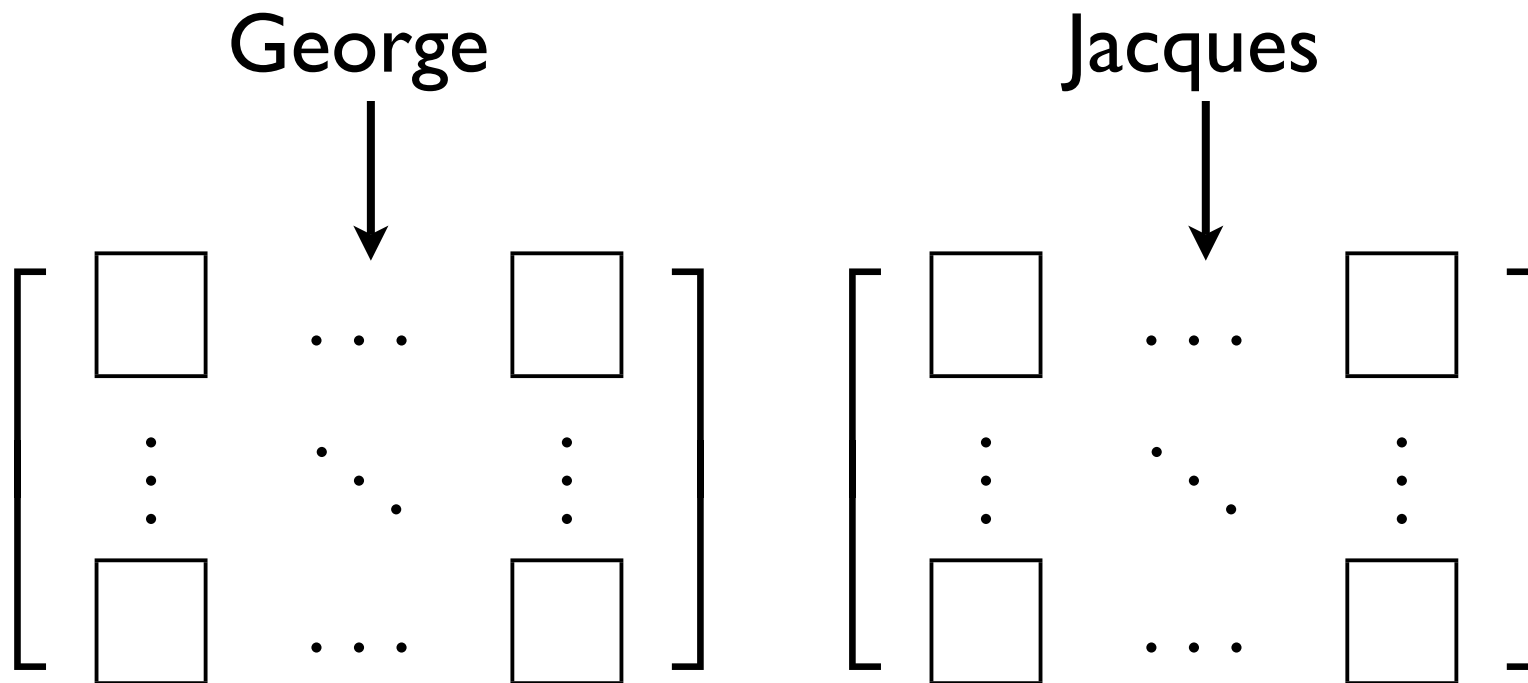
$$\begin{bmatrix} \square & \dots & \square \\ \vdots & \ddots & \vdots \\ \square & \dots & \square \end{bmatrix} \otimes \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ c_N \end{bmatrix}$$

- Proof by Random Vector Challenge
- Neff:  $O(N)$  proof.
- $N^2$  computation,  $N$  proofs.

# Mixing more than once?



# Mixing more than once?

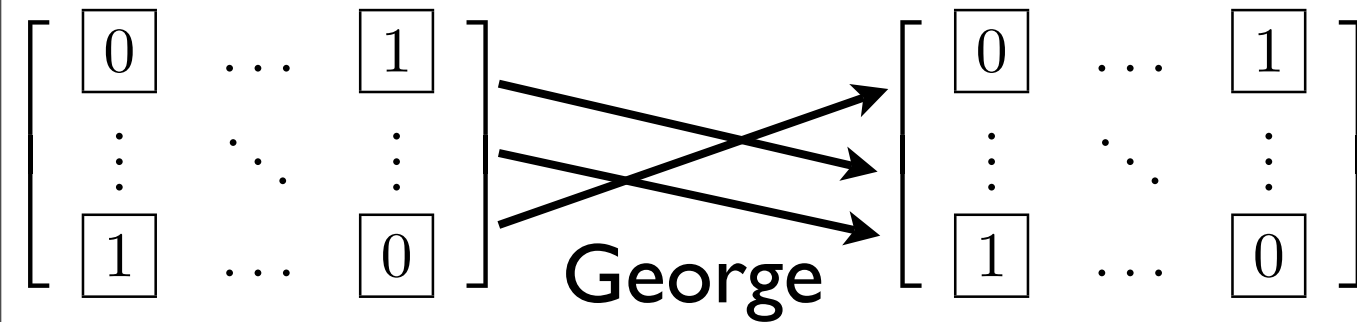


Not with BGN bilinear map...  
Only **one** multiplication.

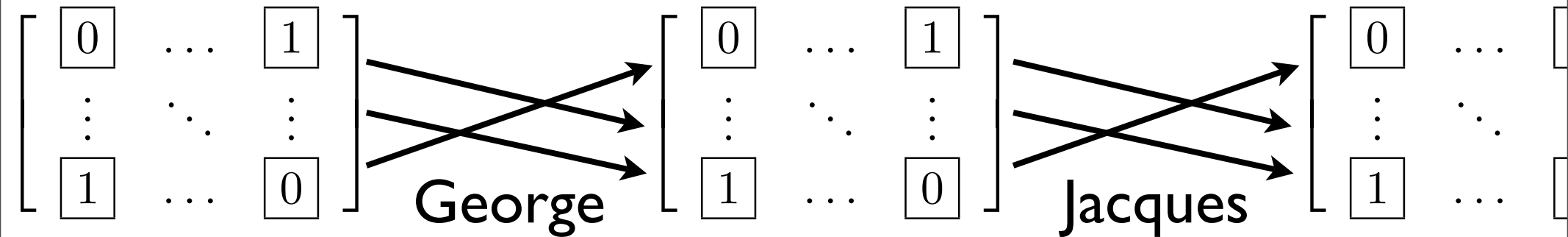
# Distributed Generation

$$\begin{bmatrix} \boxed{0} & \dots & \boxed{1} \\ \vdots & \ddots & \vdots \\ \boxed{1} & \dots & \boxed{0} \end{bmatrix}$$

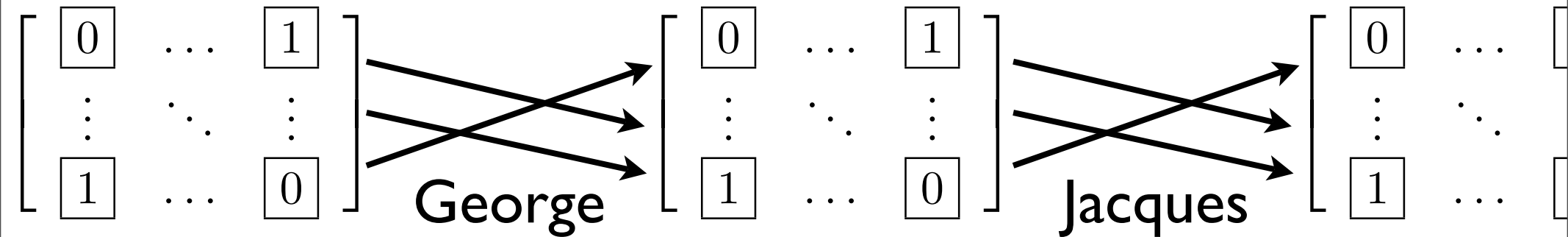
# Distributed Generation



# Distributed Generation

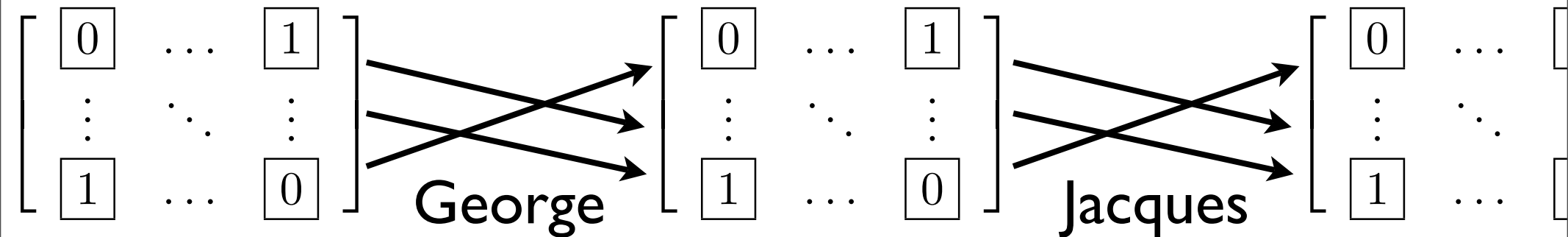


# Distributed Generation



- Use a **Mixnet** to shuffle the matrix rows

# Distributed Generation



- Use a **Mixnet** to shuffle the matrix rows
- Prove each one using Random Vector Test

# “Encapsulated” Mixing



- Capture the shuffle actions of the mix servers.
- Prove that everything went well.
- “Replay” them on the encrypted inputs when they’re available.

# Generalized Paillier

$$\text{Enc}_{pk}(m) = g^m h_2^r \bmod n^2$$

# Generalized Paillier

$$\text{Enc}_{pk}(m) = g^m h_2^r \bmod n^2$$

$$\text{Enc}_{pk,2}(m) = g^m h_3^r \bmod n^3$$

# Generalized Paillier

$$\text{Enc}_{pk}(m) = g^m h_2^r \bmod n^2$$

$$\text{Enc}_{pk,2}(m) = g^m h_3^r \bmod n^3$$

generator of  
 $n^2$  residues



# Generalized Paillier

$$\text{Enc}_{pk}(m) = g^m h_2^r \bmod n^2$$

$$\text{Enc}_{pk,2}(m) = g^m h_3^r \bmod n^3$$

generator of  
 $n^2$  residues



$$\text{Enc}_{pk,2}(\text{Enc}_{pk}(m)) = g^{g^m h_2^r} h_3^s \bmod n^3$$

# GP Homomorphisms

# GP Homomorphisms

$$\boxed{0}^{\boxed{m}} = \boxed{0 \cdot \boxed{m}} = \boxed{0}$$

# GP Homomorphisms

$$\boxed{0}^{\boxed{m}} = \boxed{0 \cdot m} = \boxed{0}$$

$$\boxed{0}^{\boxed{m}} = \boxed{0 \cdot m} = \boxed{0 + m} = \boxed{m}$$

# GP Homomorphisms

$$\boxed{0}^{\boxed{m}} = \boxed{0 \cdot m} = \boxed{0}$$

$$\boxed{0}^{\boxed{m}} = \boxed{0 \cdot m} = \boxed{0 + m} = \boxed{m}$$

$$\boxed{m} \cdot \boxed{0} = \boxed{0 + m} = \boxed{m}$$

# GP Public Shuffle

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \end{bmatrix} = \begin{bmatrix} m_3 \\ m_1 \\ m_5 \\ m_2 \\ m_4 \end{bmatrix}$$

The diagram illustrates the GP Public Shuffle operation. It shows a 5x5 matrix of zeros, where the zeros at (1,3), (2,1), (4,2), and (5,4) are highlighted with blue borders. This matrix is multiplied (indicated by the  $\otimes$  symbol) by a column vector of messages  $m_1, m_2, m_3, m_4, m_5$ , where  $m_1, m_2, m_3, m_4$  are highlighted with blue borders. The result is a column vector of messages  $m_3, m_1, m_5, m_2, m_4$ , where each message is enclosed in a dashed blue border, indicating a permutation of the original messages.

# GP Public Shuffle

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \end{bmatrix} = \begin{bmatrix} m_3 \\ m_1 \\ m_5 \\ m_2 \\ m_4 \end{bmatrix}$$

- Full-length plaintexts

# GP Public Shuffle

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \end{bmatrix} = \begin{bmatrix} m_3 \\ m_1 \\ m_5 \\ m_2 \\ m_4 \end{bmatrix}$$

- Full-length plaintexts
- Faster computation (modexp vs. BM)

# GP Public Shuffle

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \otimes \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \end{bmatrix} = \begin{bmatrix} m_3 \\ m_1 \\ m_5 \\ m_2 \\ m_4 \end{bmatrix}$$

- Full-length plaintexts
- Faster computation (modexp vs. BM)
- More complicated distributed generation

# GP Proof of Double Reenc

# GP Proof of Double Reenc

0

# GP Proof of Double Reenc

$$\boxed{0}^{\boxed{0}} \cdot \boxed{0}$$

# GP Proof of Double Reenc

$$\boxed{0}^{\boxed{0}} \cdot \boxed{0} = \boxed{0} \cdot \boxed{0} + 0 = \boxed{0}$$

# GP Proof of Double Reenc

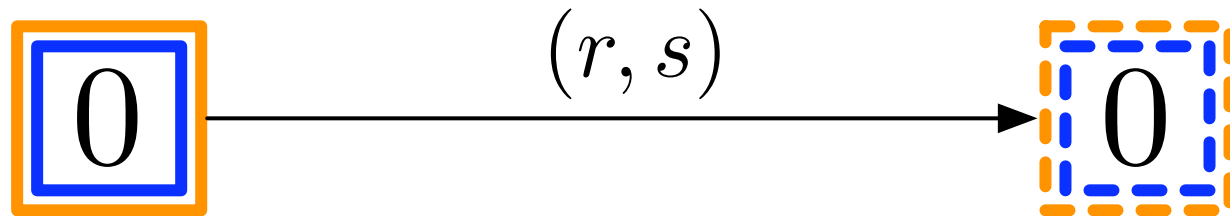
$$\boxed{0}^{\boxed{0}} \cdot \boxed{0} = \boxed{0} \cdot \boxed{0} + 0 = \boxed{0}$$

$$\text{DREEnc}(c, r, s) = c^{h_2^r} h_3^s \bmod n^3$$

# GP Proof of Double Reenc

$$\boxed{0}^{\boxed{0}} \cdot \boxed{0} = \boxed{0} \cdot \boxed{0} + 0 = \boxed{0}$$

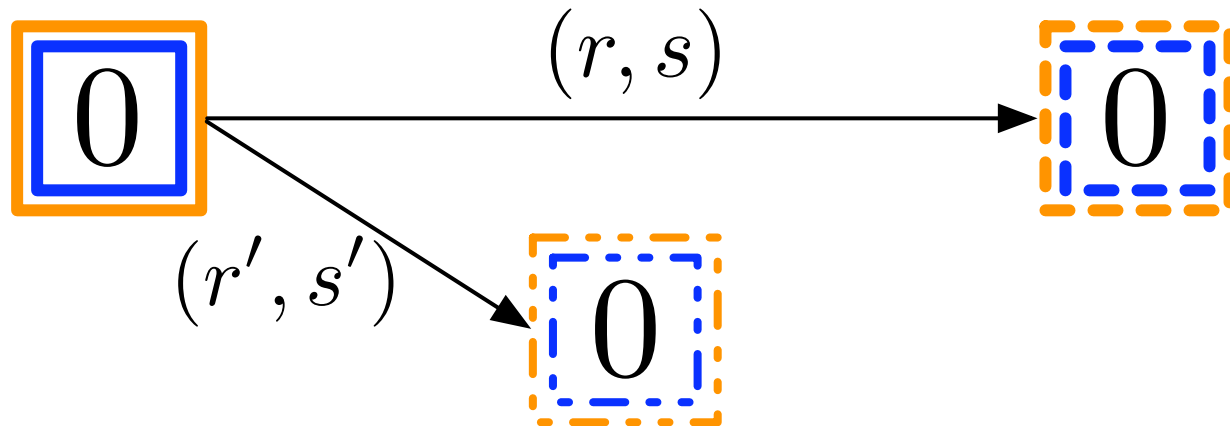
$$\text{DREEnc}(c, r, s) = c^{h_2^r} h_3^s \bmod n^3$$



# GP Proof of Double Reenc

$$\boxed{0}^{\boxed{0}} \cdot \boxed{0} = \boxed{0} \cdot \boxed{0} + 0 = \boxed{0}$$

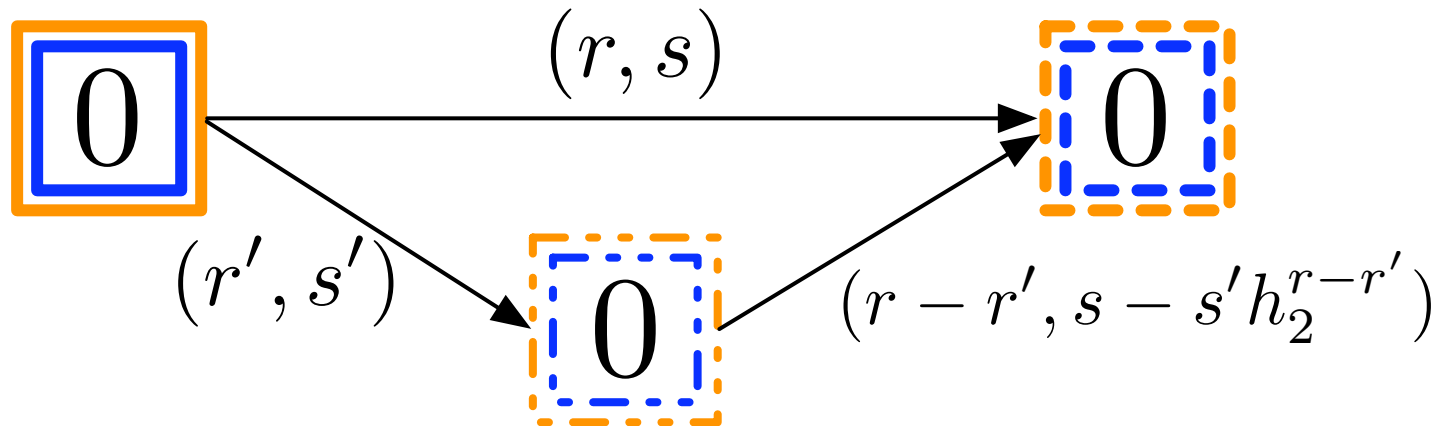
$$\text{DREEnc}(c, r, s) = c^{h_2^r} h_3^s \bmod n^3$$



# GP Proof of Double Reenc

$$\boxed{0}^{\boxed{0}} \cdot \boxed{0} = \boxed{0} \cdot \boxed{0} + 0 = \boxed{0}$$

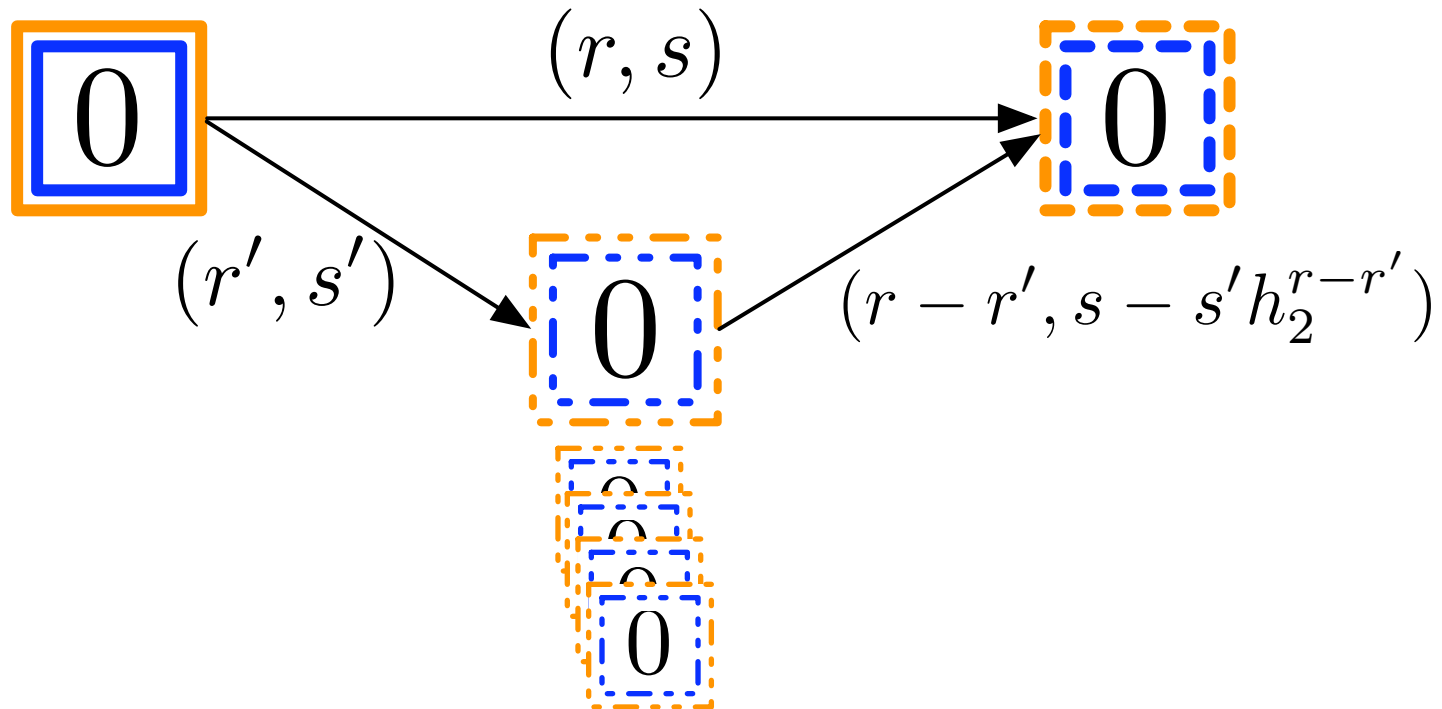
$$\text{DREEnc}(c, r, s) = c^{h_2^r} h_3^s \bmod n^3$$



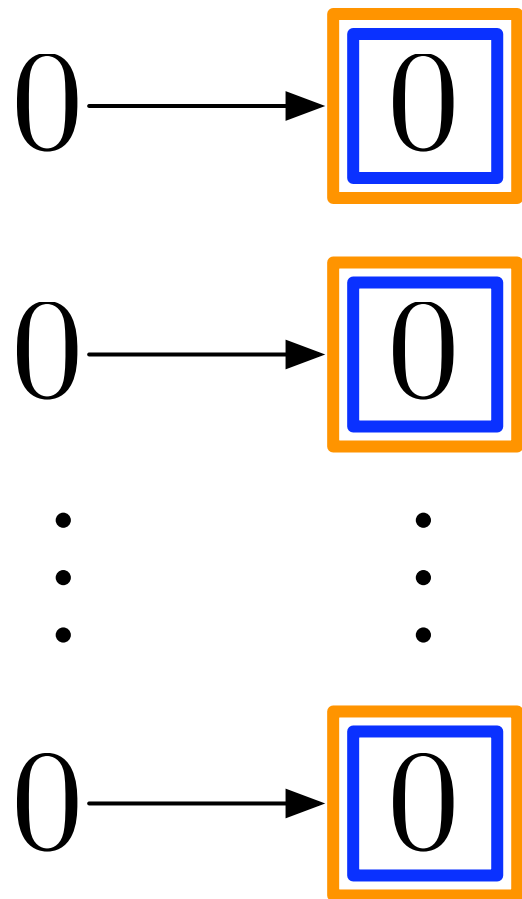
# GP Proof of Double Reenc

$$\boxed{0}^{\boxed{0}} \cdot \boxed{0} = \boxed{0} \cdot \boxed{0} + 0 = \boxed{0}$$

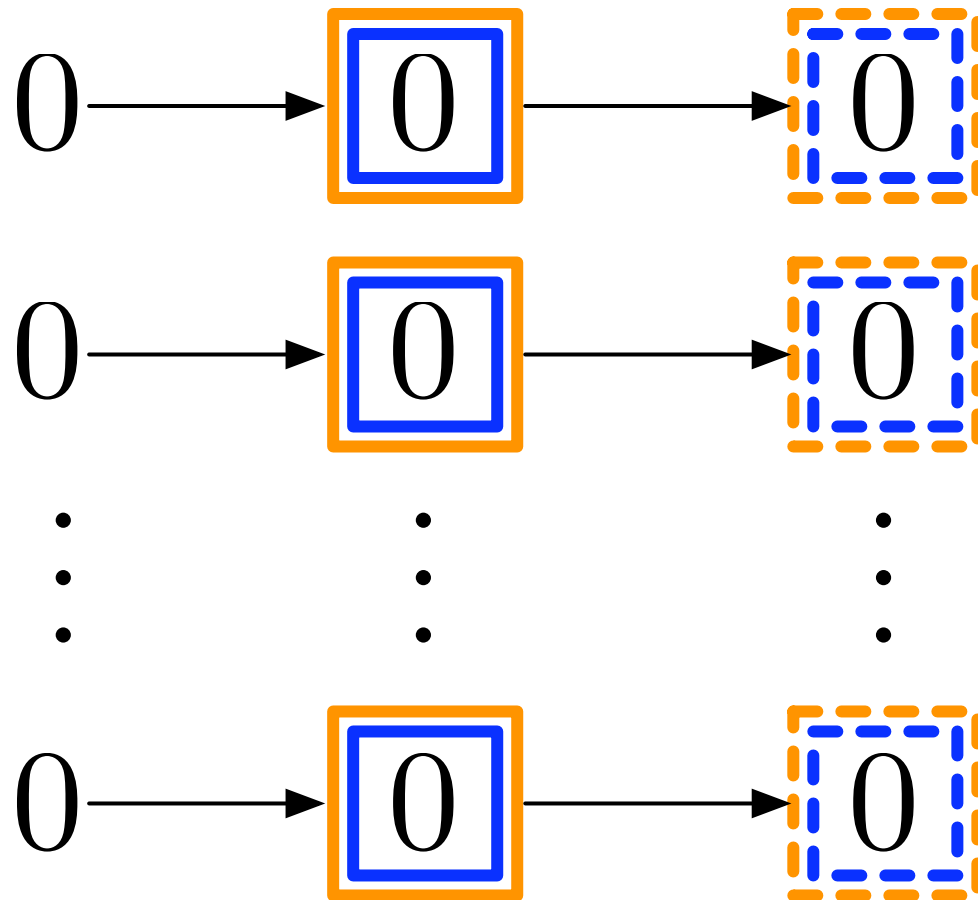
$$\text{DREEnc}(c, r, s) = c^{h_2^r} h_3^s \bmod n^3$$



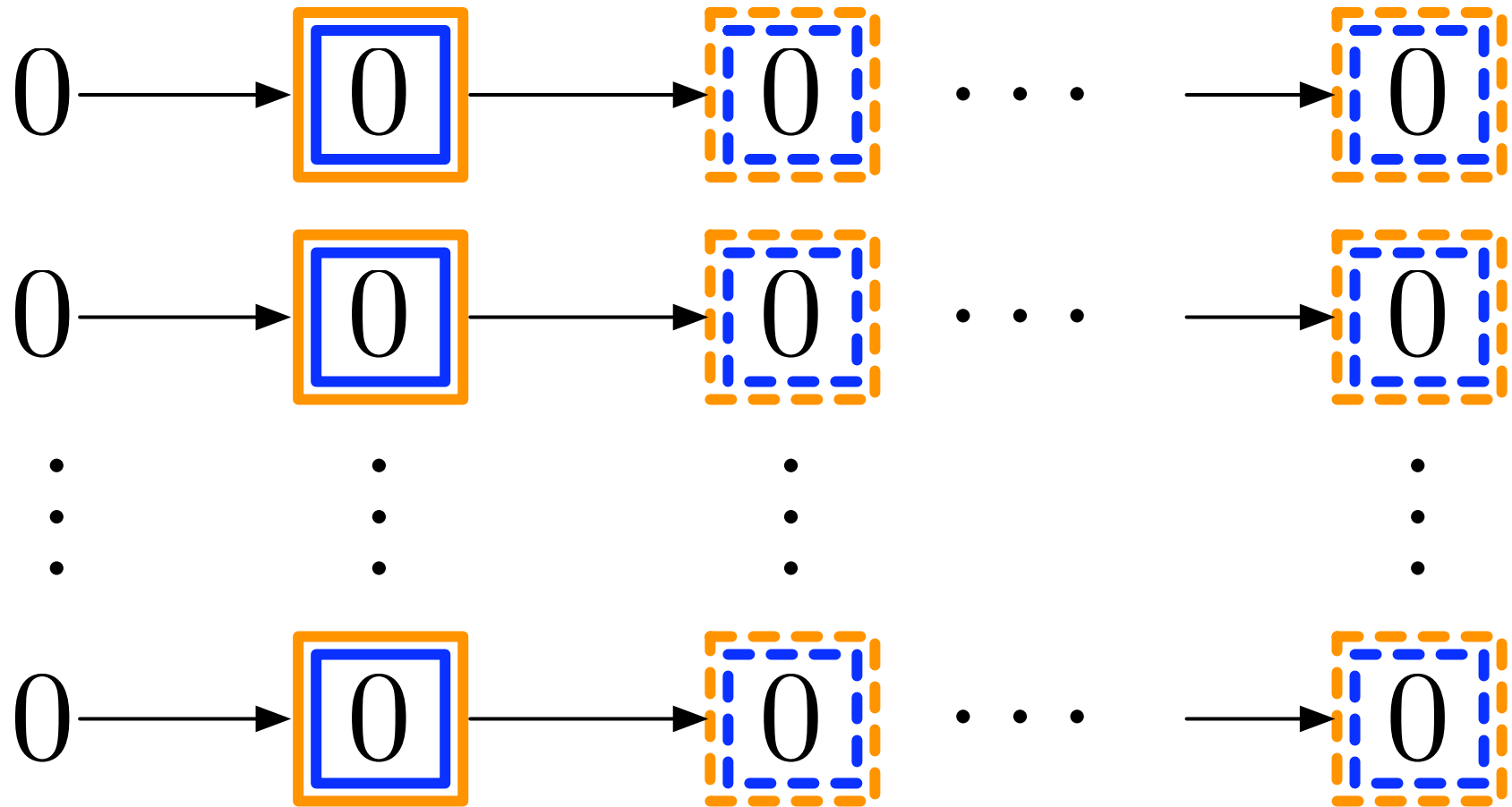
# Dist. Gen. of Diagonal



# Dist. Gen. of Diagonal



# Dist. Gen. of Diagonal



# Dist. Gen. of Matrix

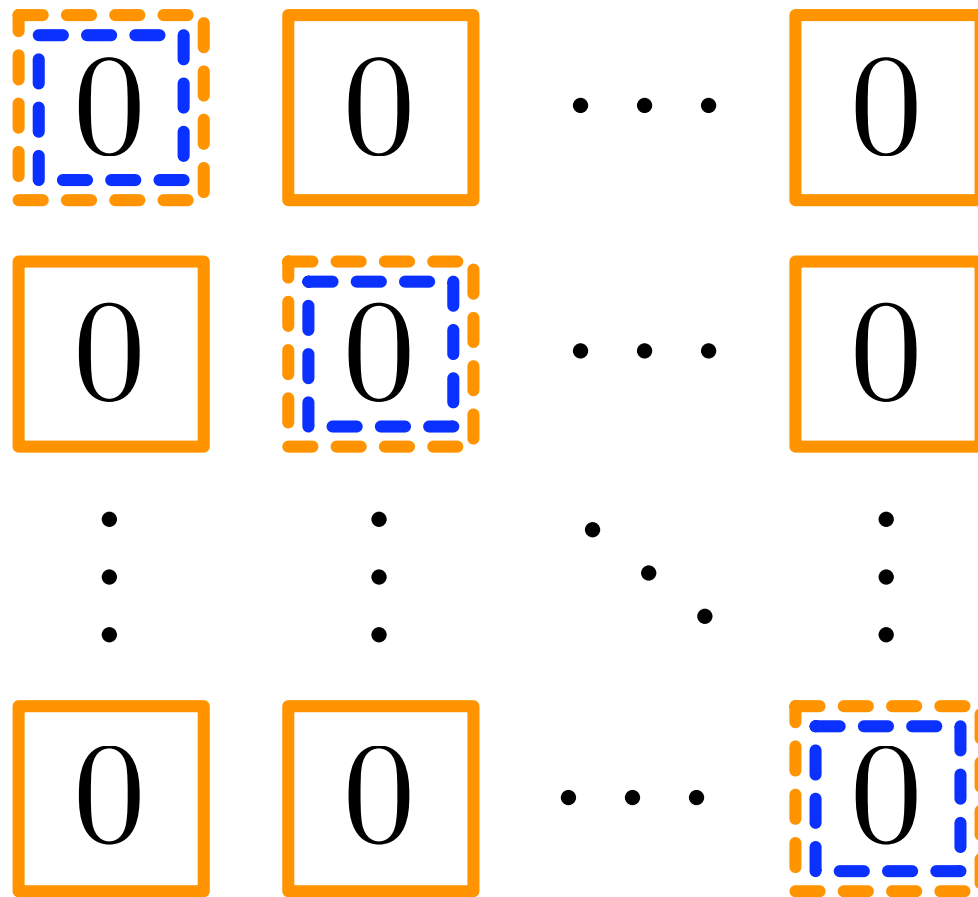
0

0

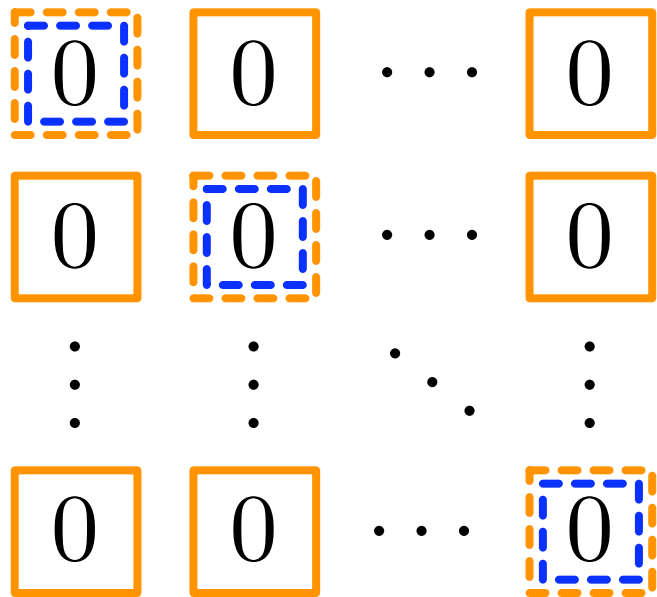
...

0

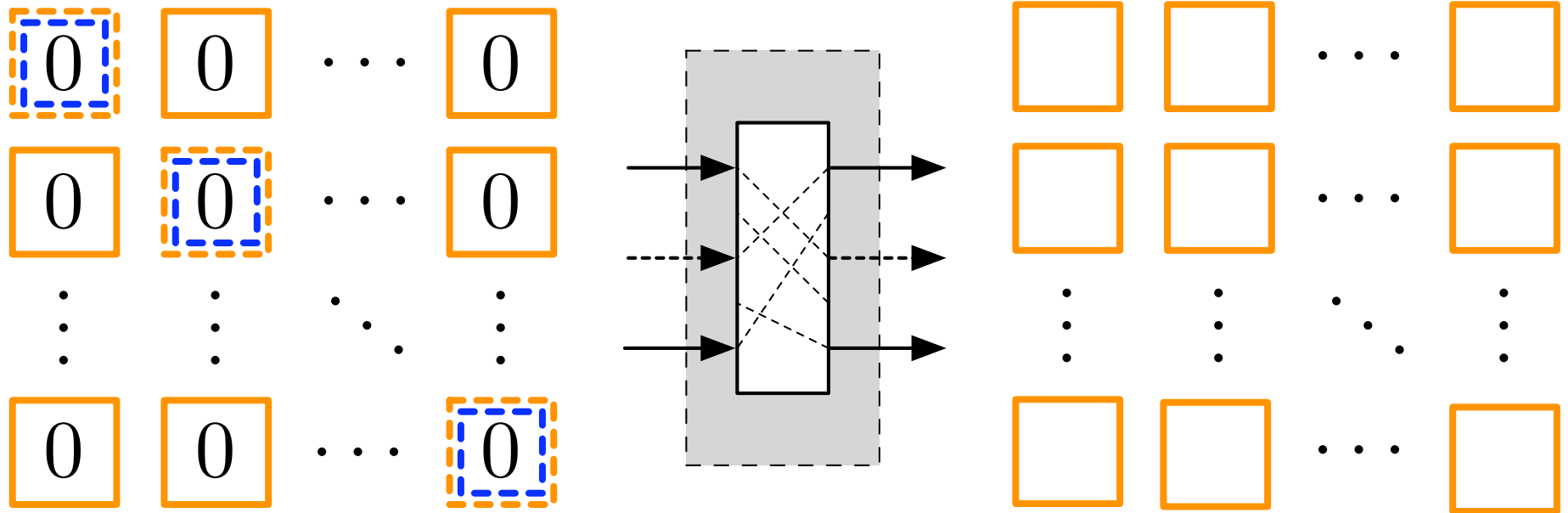
# Dist. Gen. of Matrix



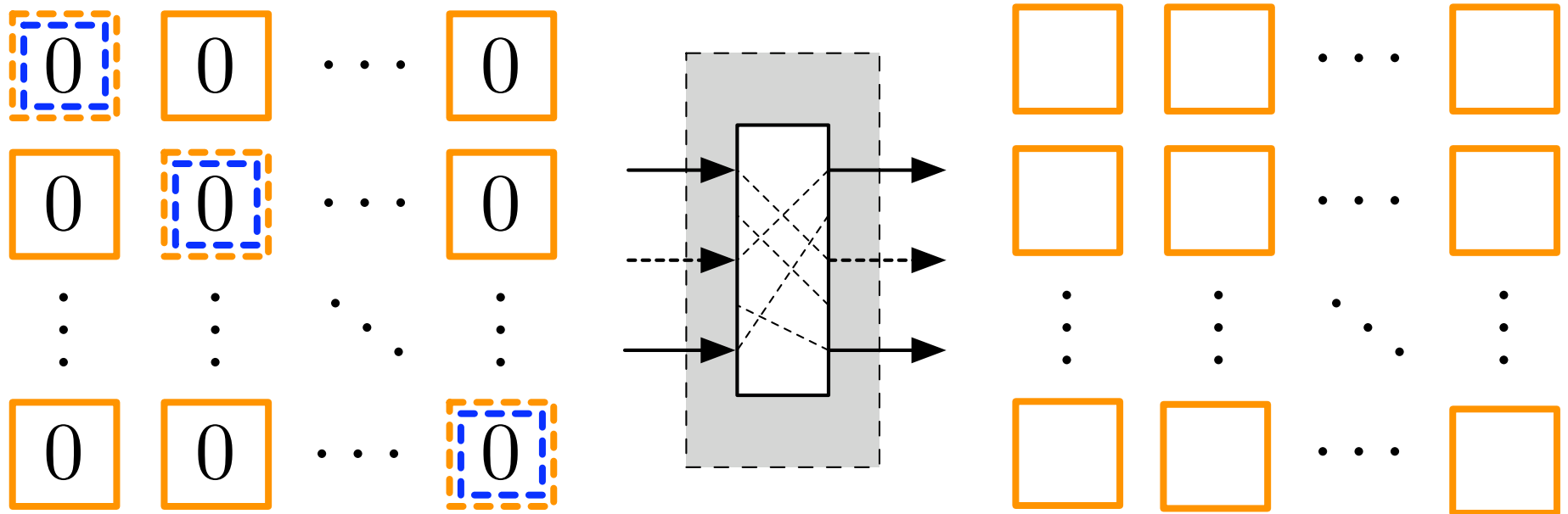
# Dist. Gen. of Matrix



# Dist. Gen. of Matrix

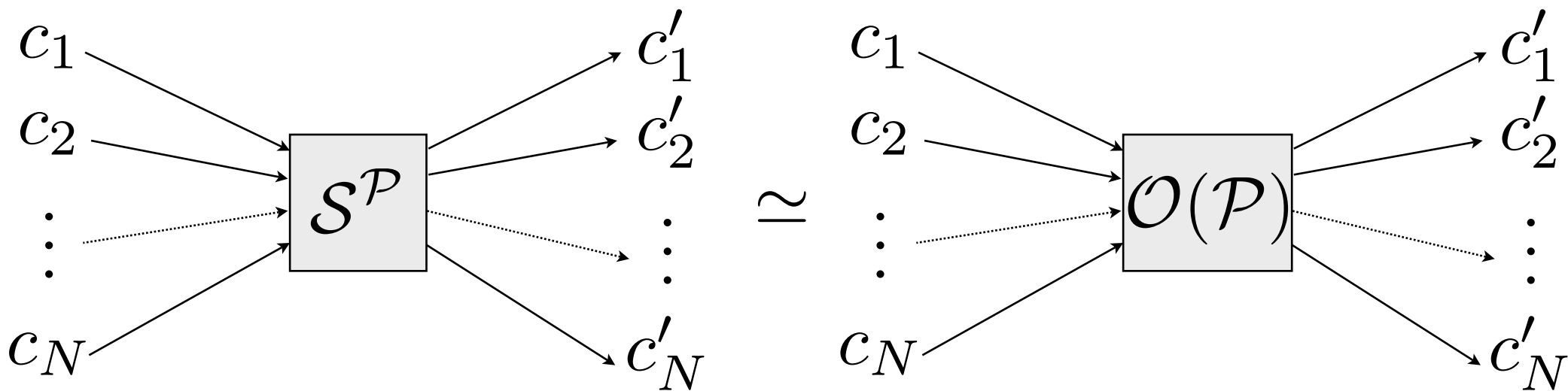


# Dist. Gen. of Matrix

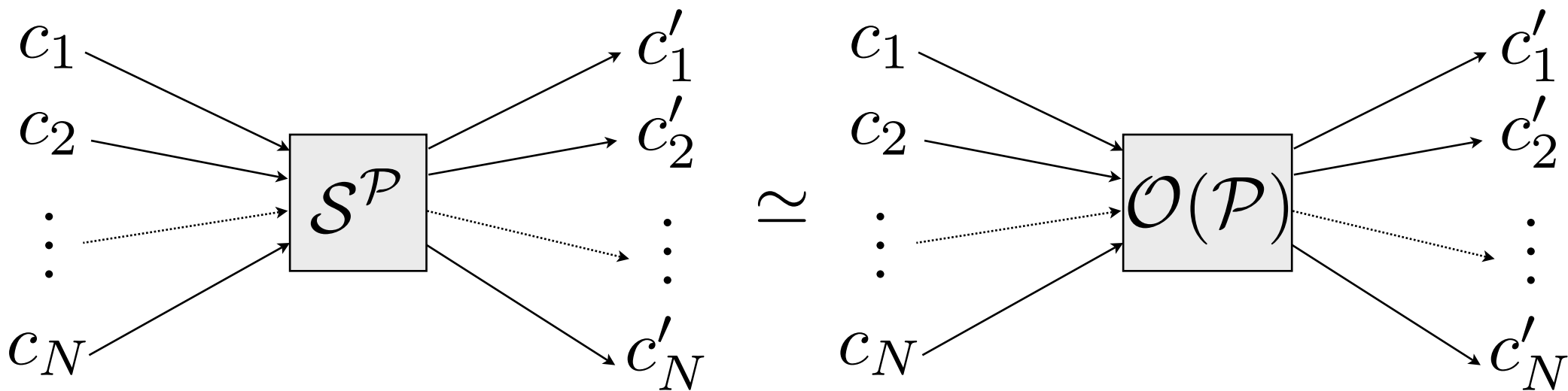


Proof with Random Vector Test.

# Obfuscation Model

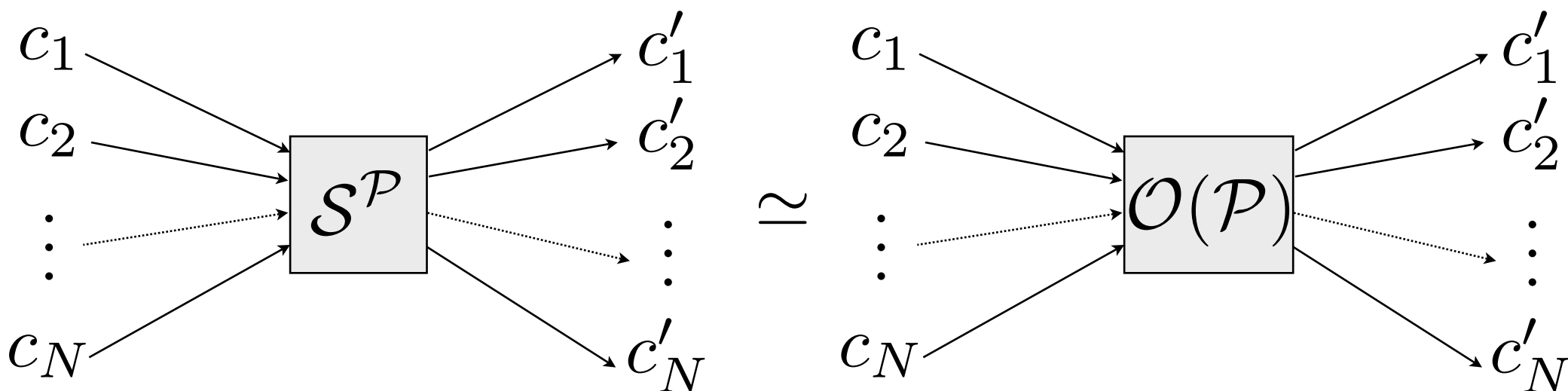


# Obfuscation Model



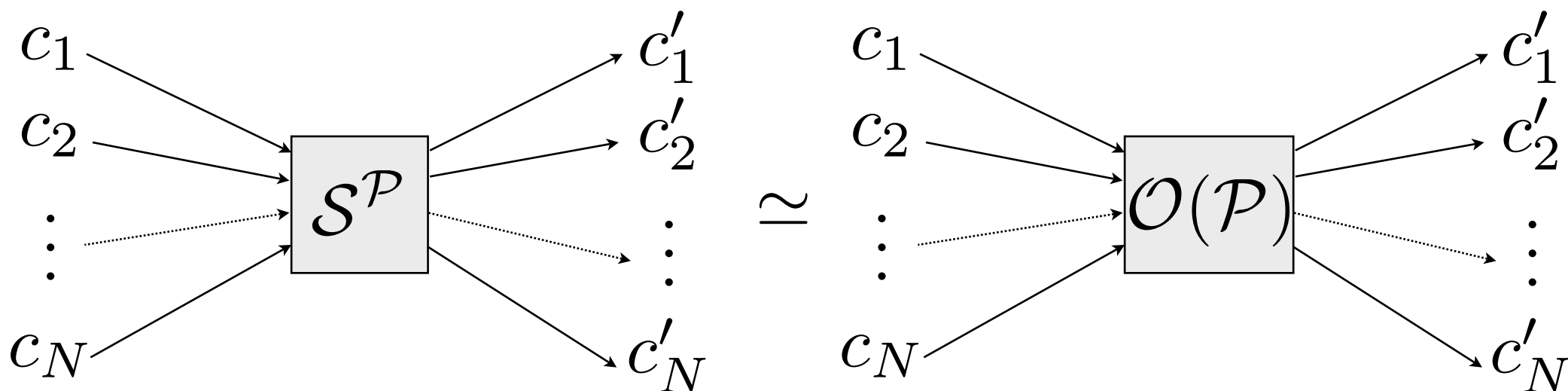
- [BGIRSVY2001] and [GT-K2005] Models

# Obfuscation Model



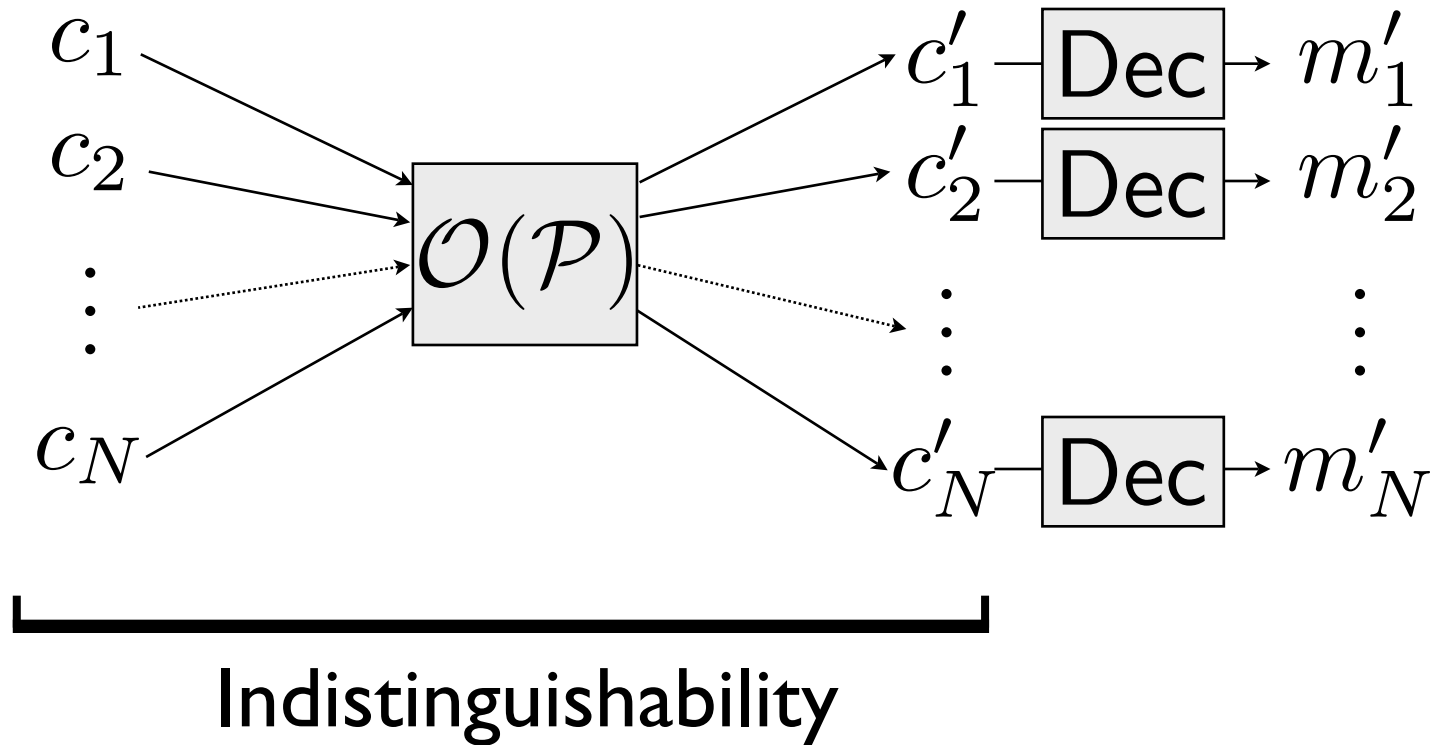
- [BGIRSVY2001] and [GT-K2005] Models
- simulation by generation of a new, random encrypted permutation matrix, based on **semantic security (IND-CPA)**

# Obfuscation Model

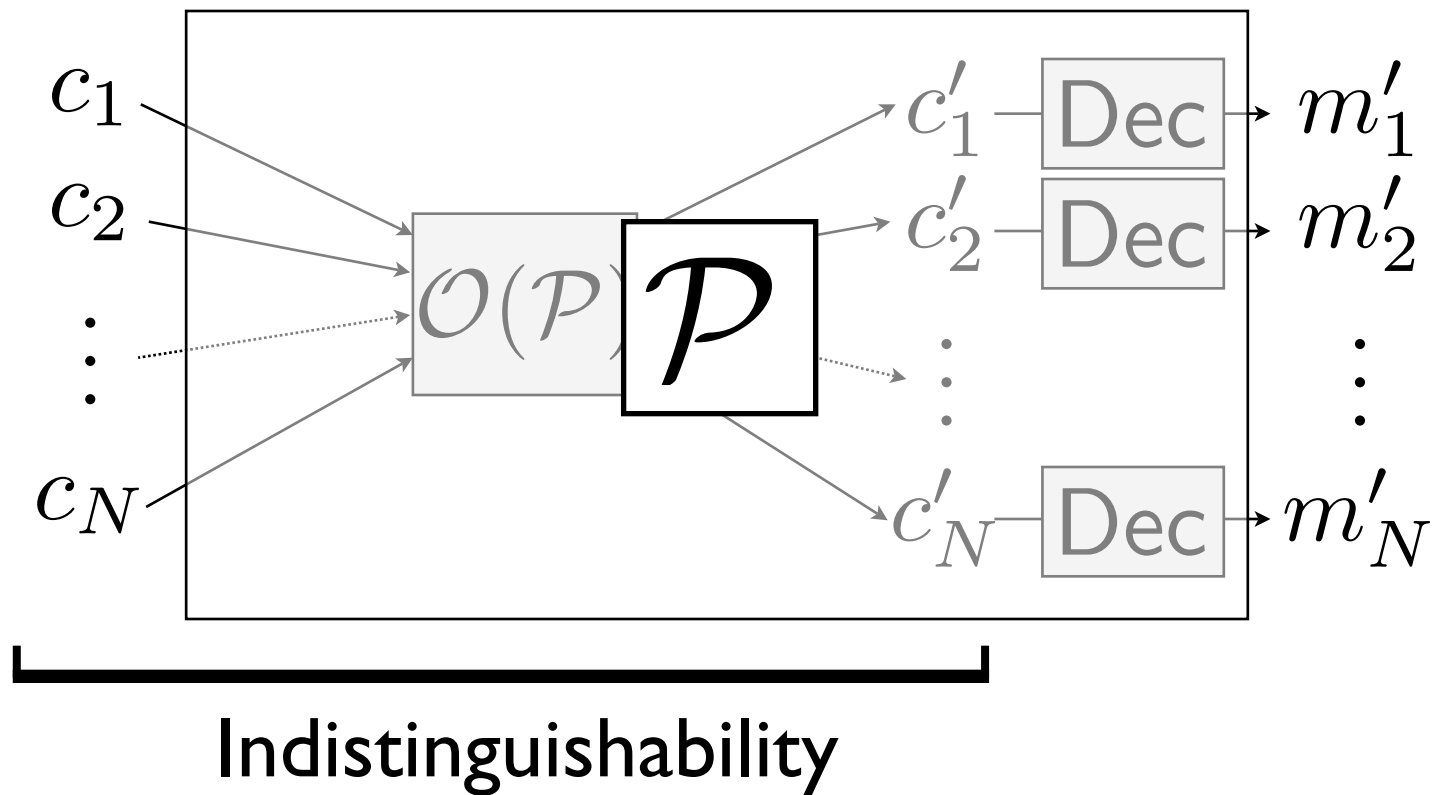


- [BGIRSVY2001] and [GT-K2005] Models
- simulation by generation of a new, random encrypted permutation matrix, based on **semantic security (IND-CPA)**
- still need to capture indistinguishability of two obfuscated shuffles.

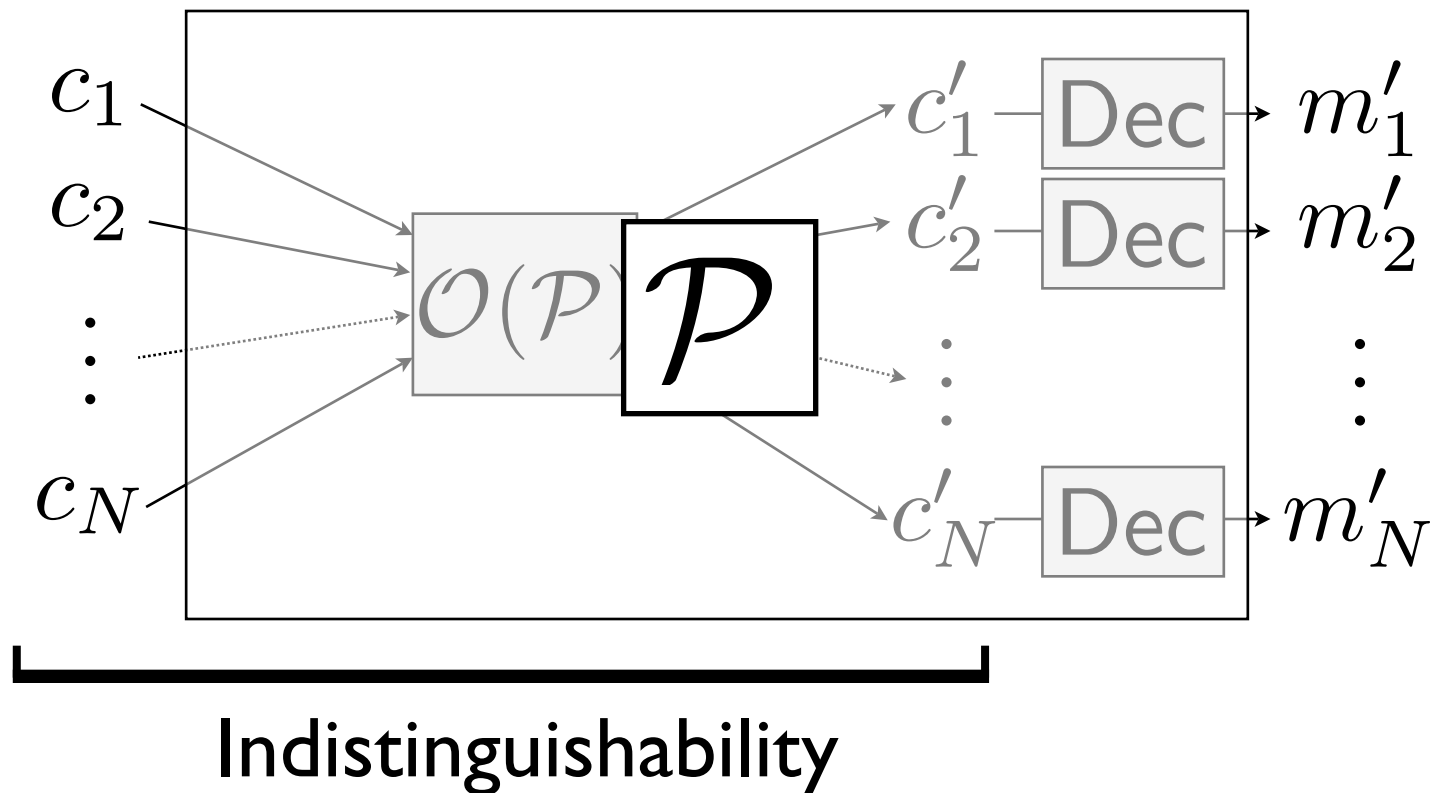
# Obfuscation Model (II): BGN



# Obfuscation Model (II): BGN

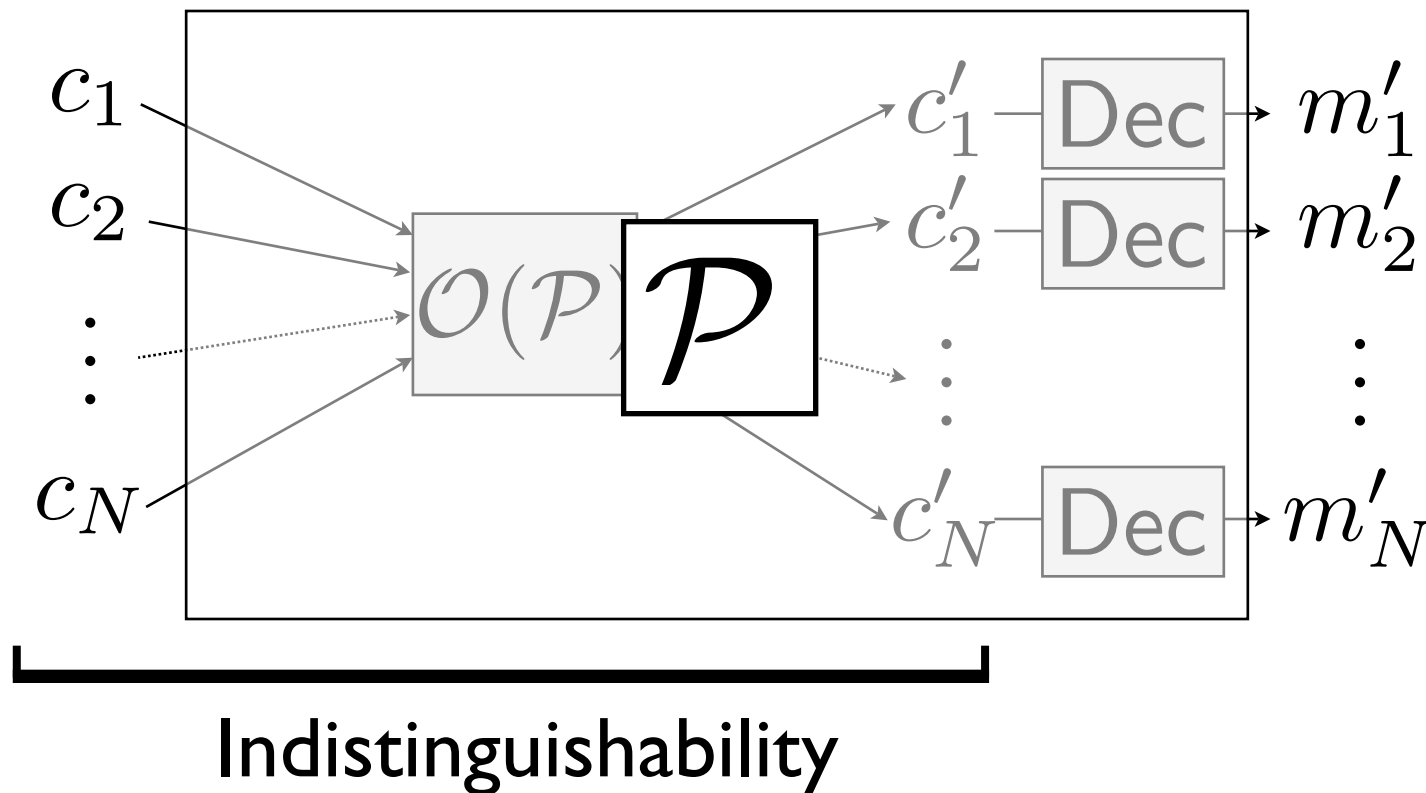


# Obfuscation Model (II): BGN



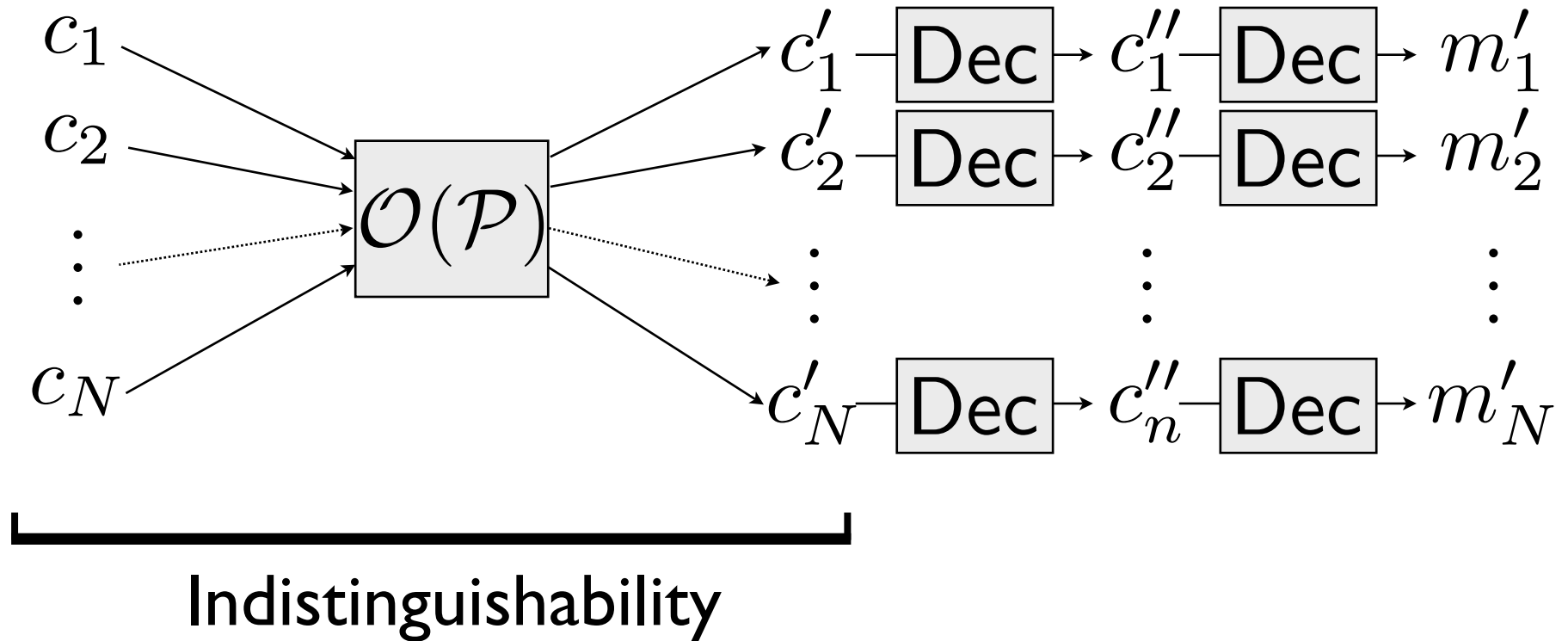
- [OS2005]: public-key obfuscation

# Obfuscation Model (II): BGN



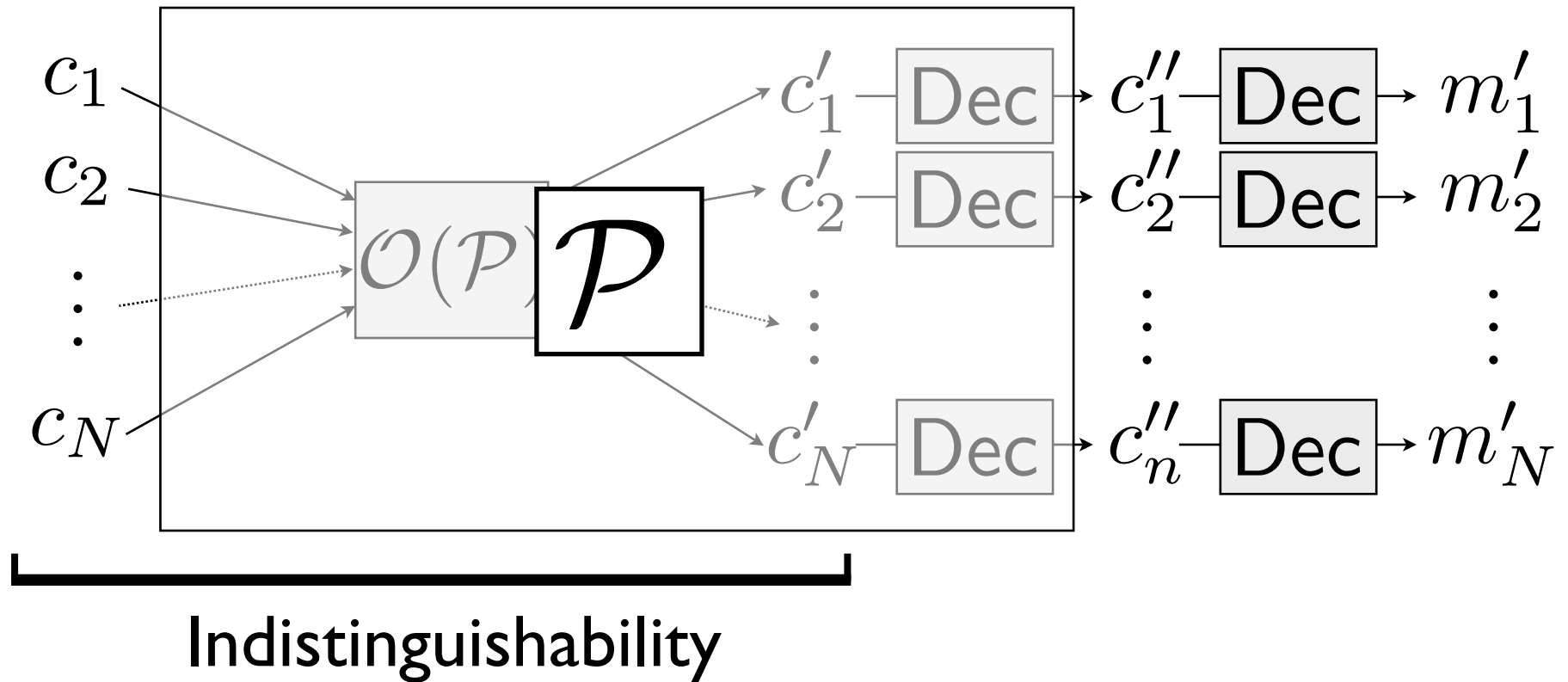
- [OS2005]: public-key obfuscation
- **with a twist**: the program can depend on the cryptosystem: in this case the program decrypts

# The Paillier Case



**Reencryption Shuffle**

# The Paillier Case



**Reencryption Shuffle**

# Proof Ideas

- \* IND-CPA  $\rightarrow$  IND of encrypted matrices  
*easy reduction using homomorphic properties*
- \* UC Proof of Ideal Mixnet realization
  - $\rightarrow$  fill in corrupted inputs by extraction from simulation of  $\mathcal{F}_{ZK}$
  - $\rightarrow$  fake decryption by using plaintexts returned by  $\mathcal{F}_{MN}$
  - $\rightarrow$  indistinguishability of fake encrypted honest inputs given IND-CPA of cryptosystem (hybrid argument).
  - $\rightarrow$  indistinguishability of fake decryption by extraction of shuffle permutation (without SK) and correction of permutation by one honest mix server, given IND-CPA of cryptosystem (simulation of  $\mathcal{F}_{CF}$  ).

# Shuffling in Public

- All proofs done prior to shuffling
- Shuffling becomes **entirely deterministic**
- Efficient enough for precinct-based elections

# Shuffling in Public

- All proofs done prior to shuffling
- Shuffling becomes **entirely deterministic**
- Efficient enough for precinct-based elections

## Future Directions

- Better than  $O(N^2)$ ?
- Other obfuscations using unexpected homomorphic properties?
- Plugging in latest NIZK techniques

