

Homomorphic Encryption

Lecture 15

And some applications

Homomorphic Encryption

Homomorphic Encryption

- Group Homomorphism: Two groups G and G' are homomorphic if there exists a function (homomorphism) $f:G \rightarrow G'$ such that for all $x,y \in G$, $f(x +_G y) = f(x) +_{G'} f(y)$

Homomorphic Encryption

- Group Homomorphism: Two groups G and G' are homomorphic if there exists a function (homomorphism) $f:G \rightarrow G'$ such that for all $x,y \in G$, $f(x +_G y) = f(x) +_{G'} f(y)$
- Homomorphic Encryption: A CPA secure encryption s.t. $\text{Dec}(C +_C D) = \text{Dec}(C) +_M \text{Dec}(D)$ for ciphertexts C, D

Homomorphic Encryption

- Group Homomorphism: Two groups G and G' are homomorphic if there exists a function (homomorphism) $f:G \rightarrow G'$ such that for all $x,y \in G$, $f(x +_G y) = f(x) +_{G'} f(y)$
- Homomorphic Encryption: A CPA secure encryption s.t. $\text{Dec}(C +_C D) = \text{Dec}(C) +_M \text{Dec}(D)$ for ciphertexts C, D
 - i.e. $\text{Enc}(x) +_C \text{Enc}(y)$ is like $\text{Enc}(x +_M y)$

Rerandomization

Rerandomization

- Often (but not always) another property is required of a homomorphic encryption scheme

Rerandomization

- Often (but not always) another property is required of a homomorphic encryption scheme
- Unlinkability

Rerandomization

- Often (but not always) another property is required of a homomorphic encryption scheme
- Unlinkability
 - For any two ciphertexts $c_x = \text{Enc}(x)$ and $c_y = \text{Enc}(y)$, $\text{Add}(c_x, c_y)$ should be identically distributed as $\text{Enc}(x +_M y)$. Add is a randomized operation

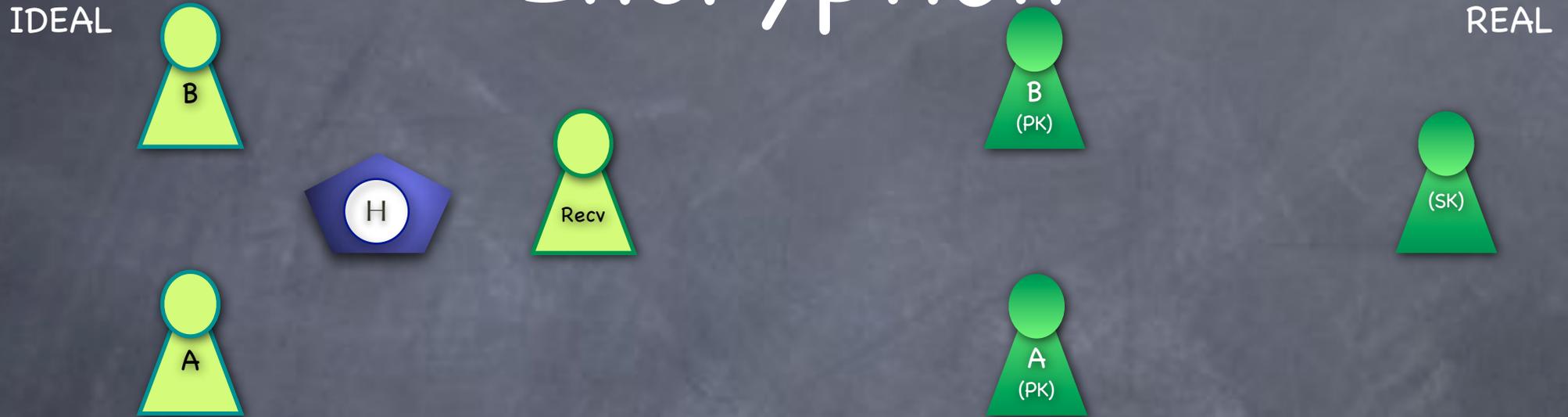
Rerandomization

- Often (but not always) another property is required of a homomorphic encryption scheme
- Unlinkability
 - For any two ciphertexts $c_x = \text{Enc}(x)$ and $c_y = \text{Enc}(y)$, $\text{Add}(c_x, c_y)$ should be identically distributed as $\text{Enc}(x +_M y)$. Add is a randomized operation
 - Alternately, a deterministic operation $+_c$ s.t. for all c_x, c_y , $c_x +_c \text{Enc}(y)$ and $\text{Enc}(x) +_c c_y$ are identically distributed as $\text{Enc}(x +_M y)$

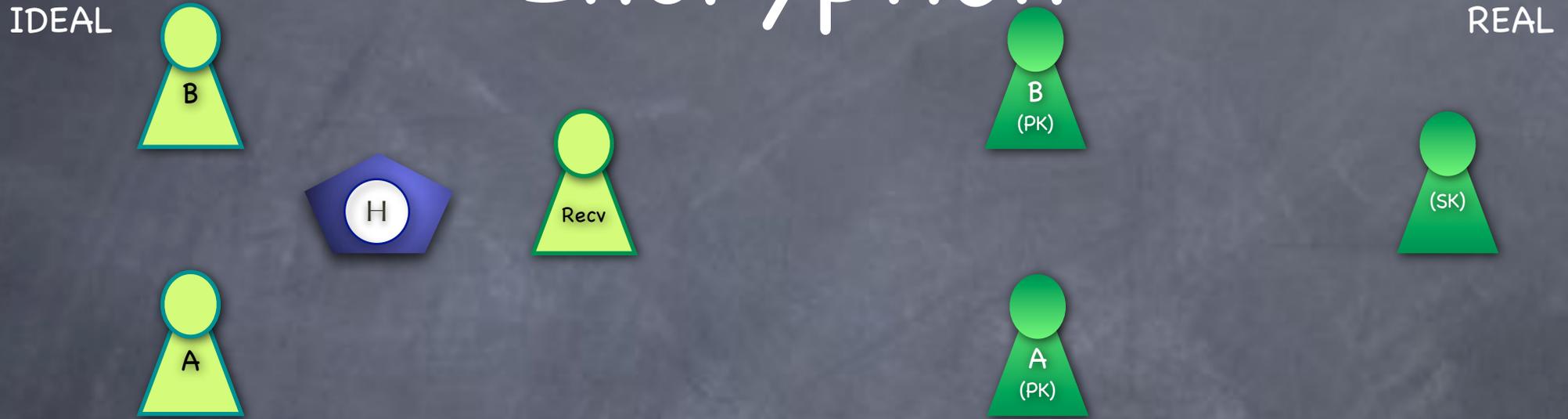
Rerandomization

- Often (but not always) another property is required of a homomorphic encryption scheme
- Unlinkability
 - For any two ciphertexts $c_x = \text{Enc}(x)$ and $c_y = \text{Enc}(y)$, $\text{Add}(c_x, c_y)$ should be identically distributed as $\text{Enc}(x +_M y)$. Add is a randomized operation
 - Alternately, a deterministic operation $+_c$ s.t. for all c_x, c_y , $c_x +_c \text{Enc}(y)$ and $\text{Enc}(x) +_c c_y$ are identically distributed as $\text{Enc}(x +_M y)$
 - Then, we can let $\text{Add}(c_x, c_y) = \text{ReRand}(c_x +_c c_y)$ where $\text{ReRand}(c) = c +_c \text{Enc}(0)$

Unlinkable Homomorphic Encryption

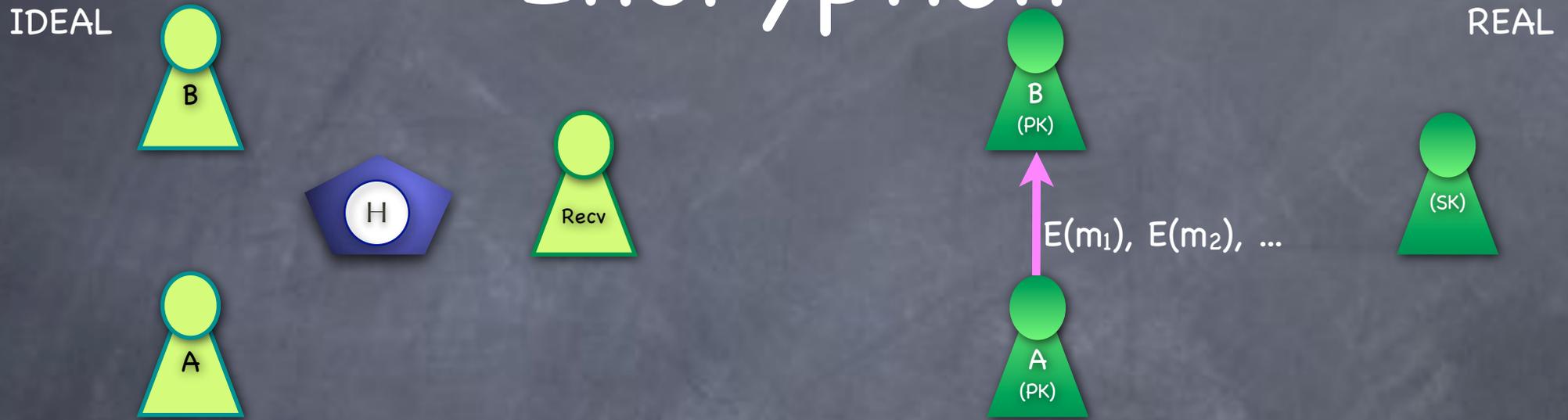


Unlinkable Homomorphic Encryption



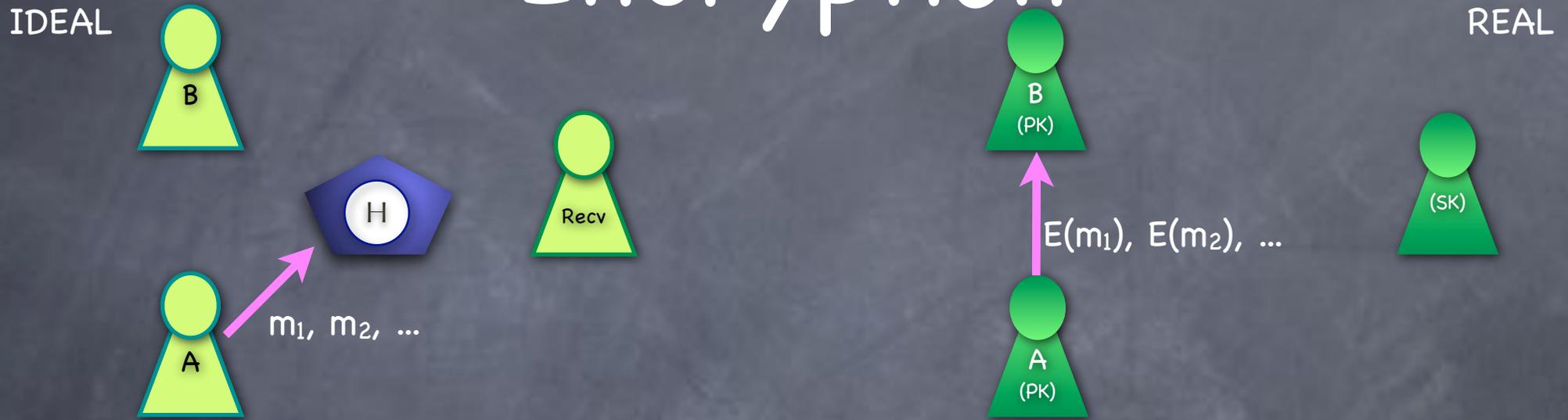
- Considers only passive corruption

Unlinkable Homomorphic Encryption



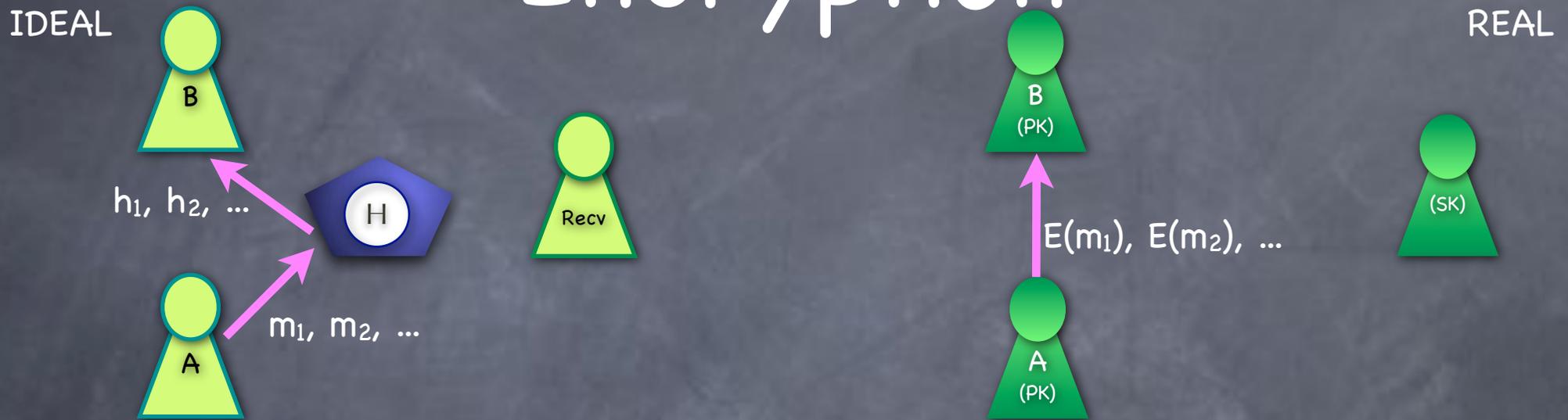
- Considers only passive corruption

Unlinkable Homomorphic Encryption



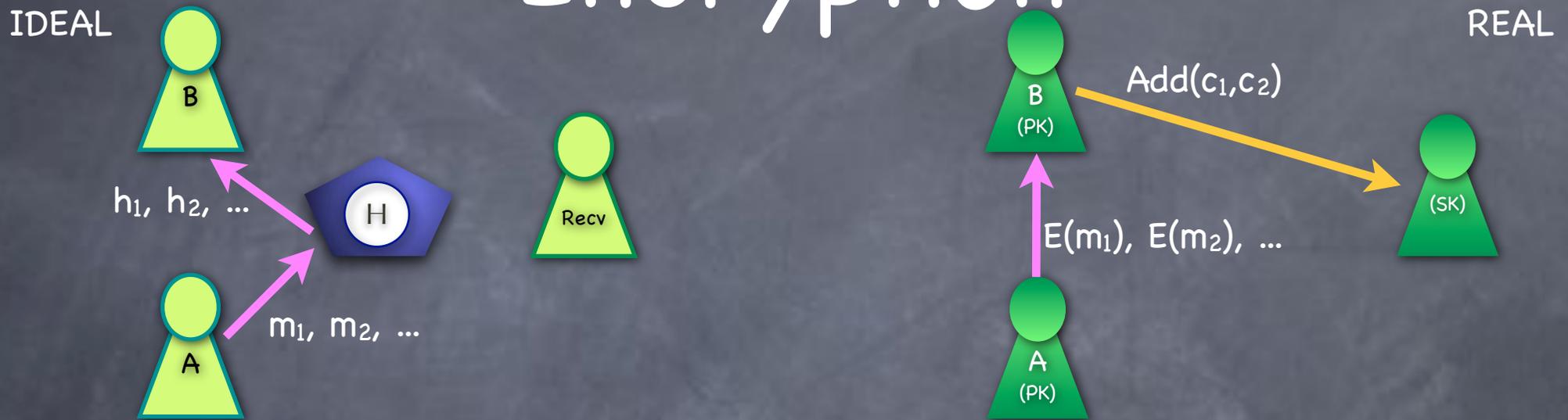
- Considers only passive corruption

Unlinkable Homomorphic Encryption



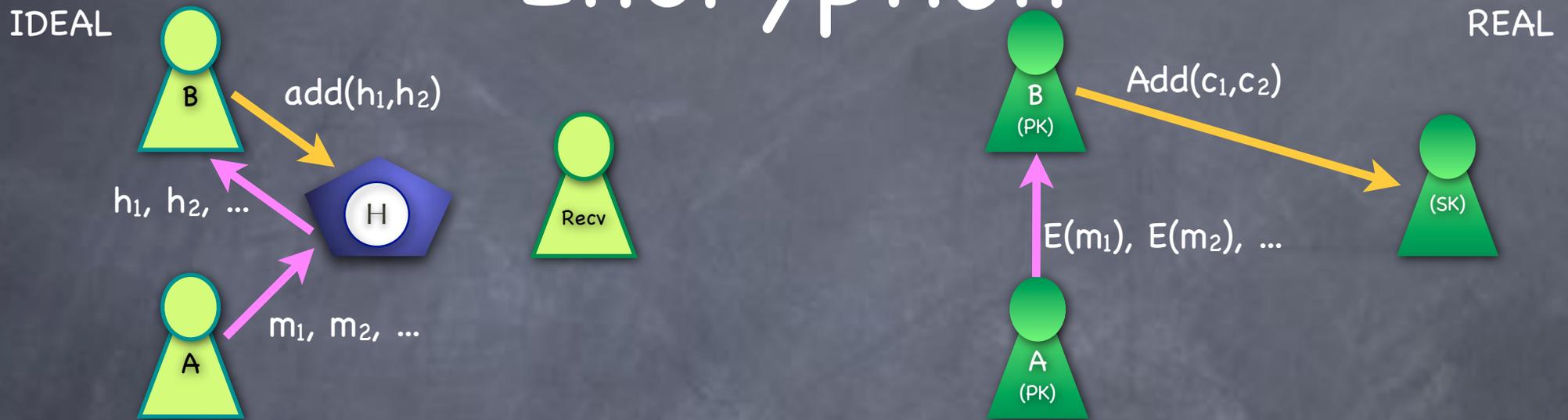
- Considers only passive corruption

Unlinkable Homomorphic Encryption



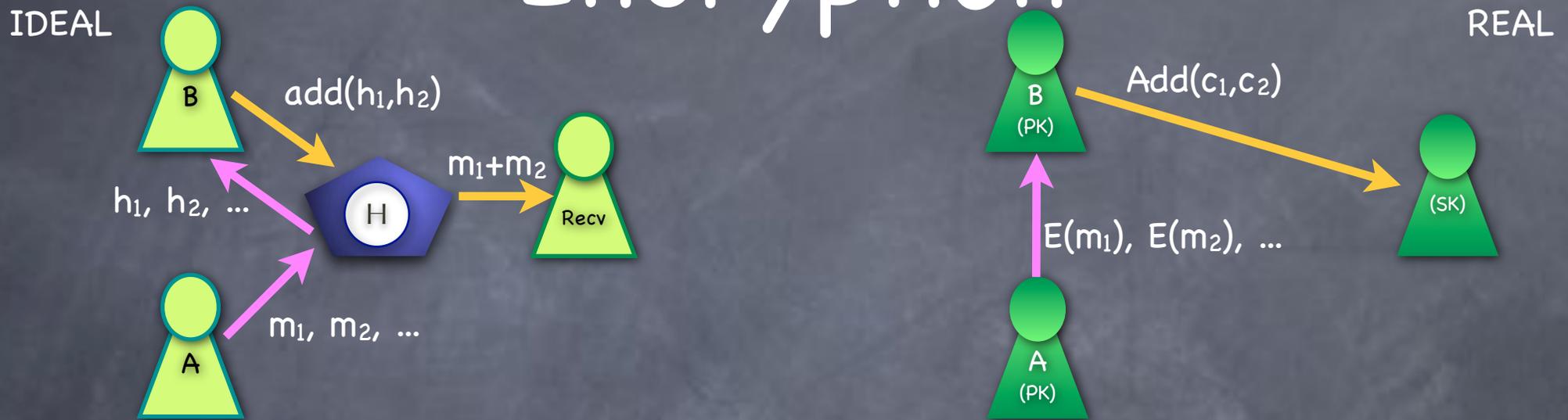
- Considers only passive corruption

Unlinkable Homomorphic Encryption



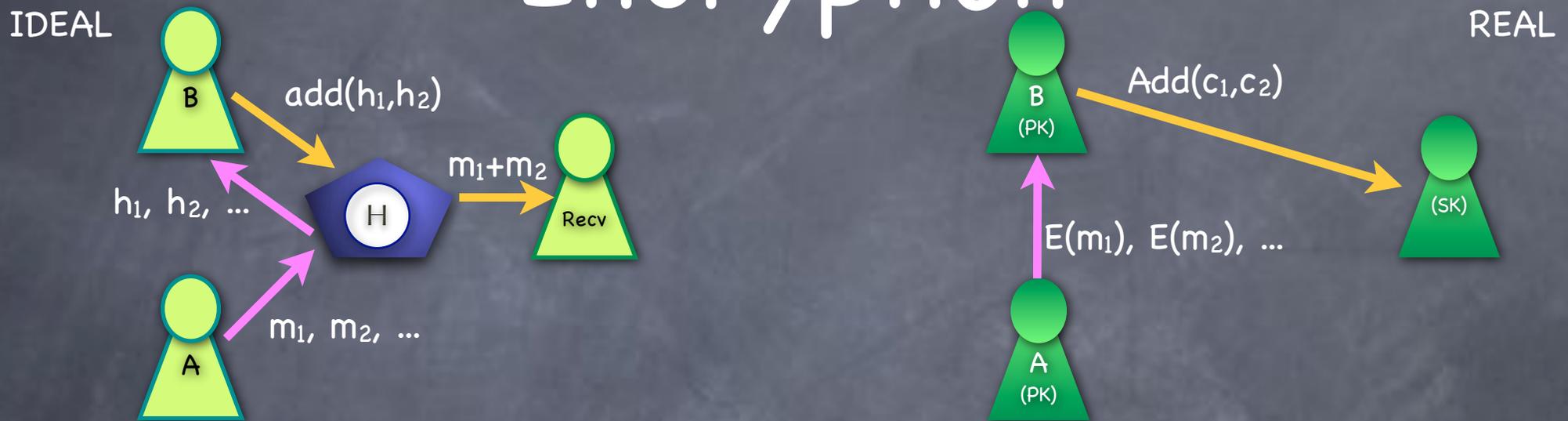
- Considers only passive corruption

Unlinkable Homomorphic Encryption



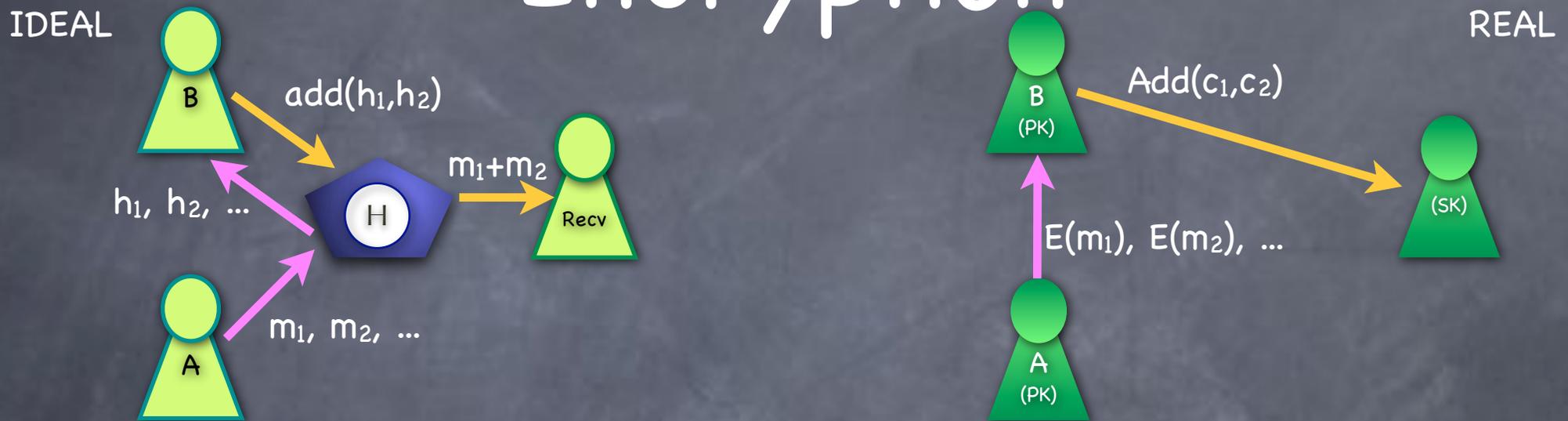
- Considers only passive corruption

Unlinkable Homomorphic Encryption



- Considers only passive corruption
- Functionality gives “handles” to messages posted; accepts requests for posting fresh messages, or derived messages

Unlinkable Homomorphic Encryption



- Considers only passive corruption
- Functionality gives “handles” to messages posted; accepts requests for posting fresh messages, or derived messages
- Unlinkability: Above, when receiver is (passively) corrupt, in IDEAL gets only the message $m_1 + m_2$; is not told if it is a fresh message or derived from other messages

An OT Protocol

(passive receiver corruption)

An OT Protocol

(passive receiver corruption)

- Using an (unlinkable) rerandomizable encryption scheme

An OT Protocol

(passive receiver corruption)

- Using an (unlinkable) rerandomizable encryption scheme



An OT Protocol (passive receiver corruption)

- Using an (unlinkable) rerandomizable encryption scheme



An OT Protocol (passive receiver corruption)

- Using an (unlinkable) rerandomizable encryption scheme



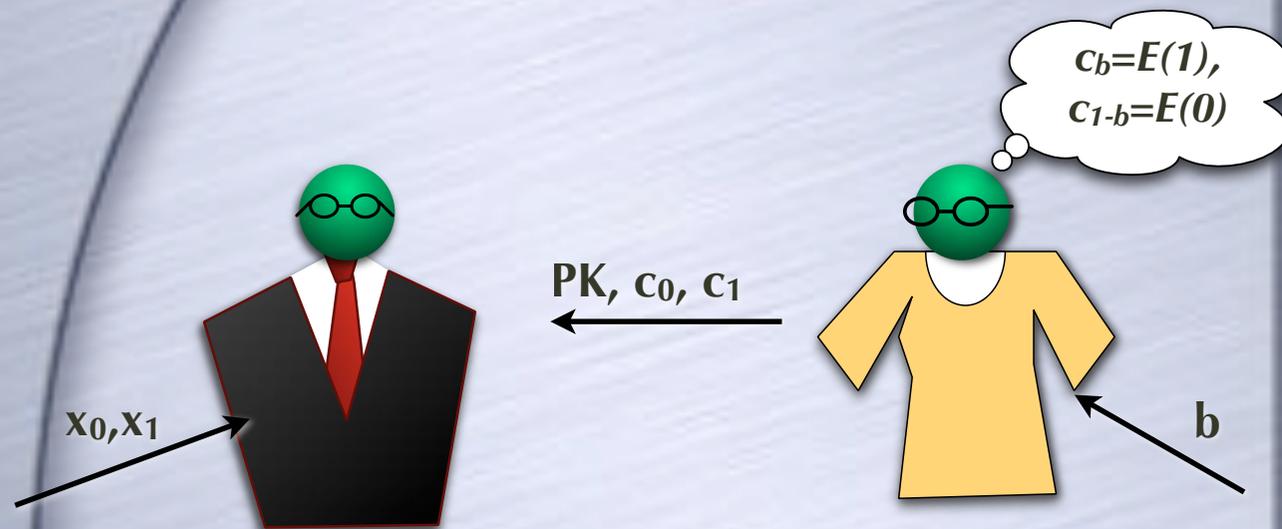
An OT Protocol (passive receiver corruption)

- Using an (unlinkable) rerandomizable encryption scheme



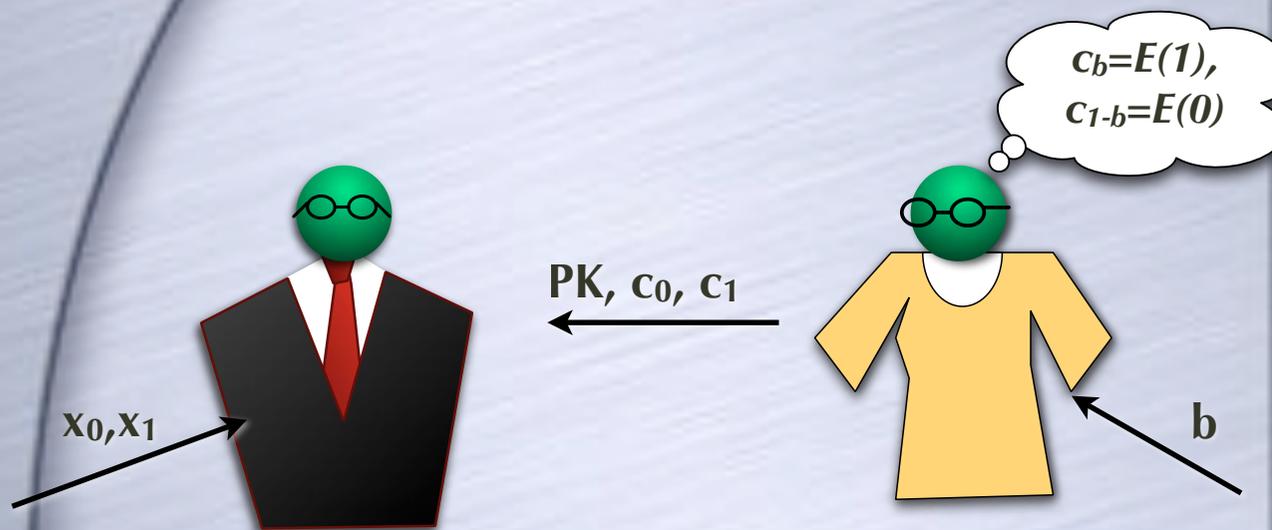
An OT Protocol (passive receiver corruption)

- Using an (unlinkable) rerandomizable encryption scheme



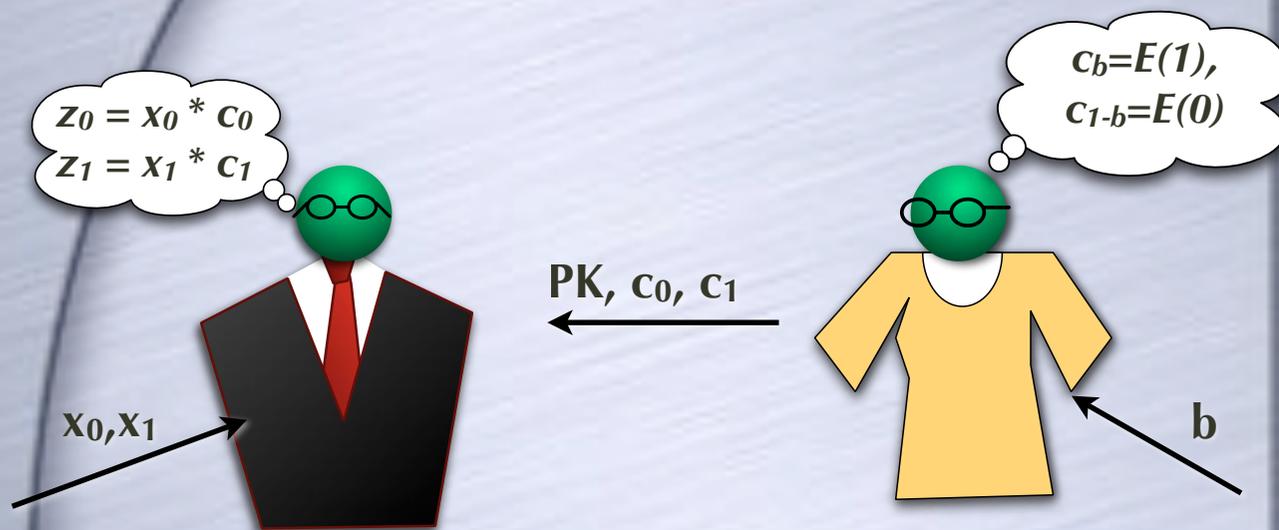
An OT Protocol (passive receiver corruption)

- Using an (unlinkable) rerandomizable encryption scheme
- Receiver picks (PK, SK) . Sends PK and $E(0), E(1)$ in suitable order



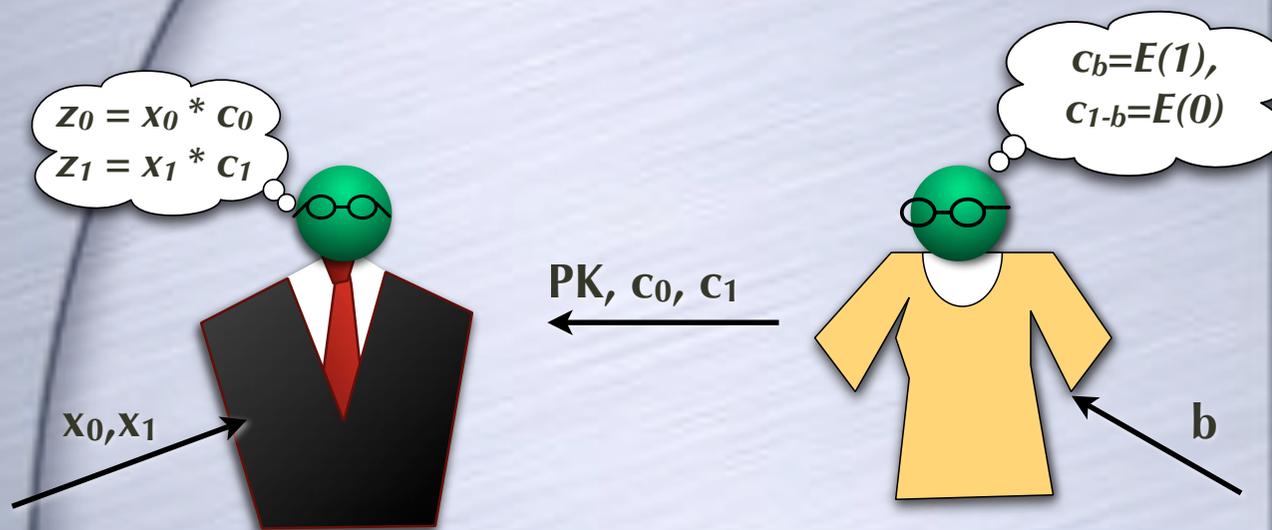
An OT Protocol (passive receiver corruption)

- Using an (unlinkable) rerandomizable encryption scheme
- Receiver picks (PK, SK) . Sends PK and $E(0), E(1)$ in suitable order



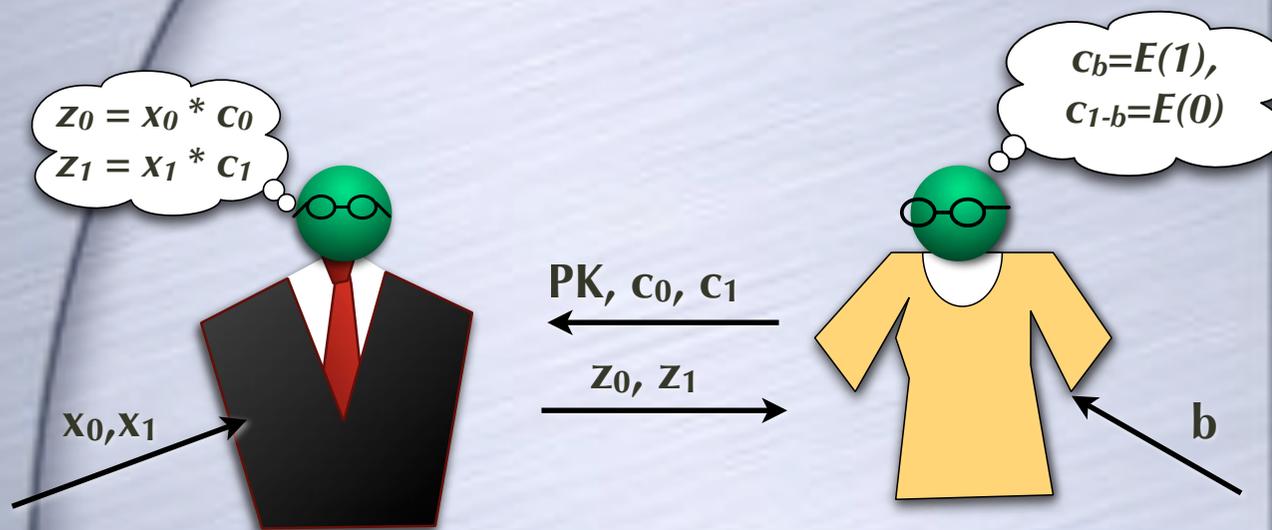
An OT Protocol (passive receiver corruption)

- Using an (unlinkable) rerandomizable encryption scheme
 - Receiver picks (PK, SK) . Sends PK and $E(0), E(1)$ in suitable order
 - Sender “multiplies” c_i with x_i :
 $1 * c := \text{ReRand}(c), 0 * c := E(0)$



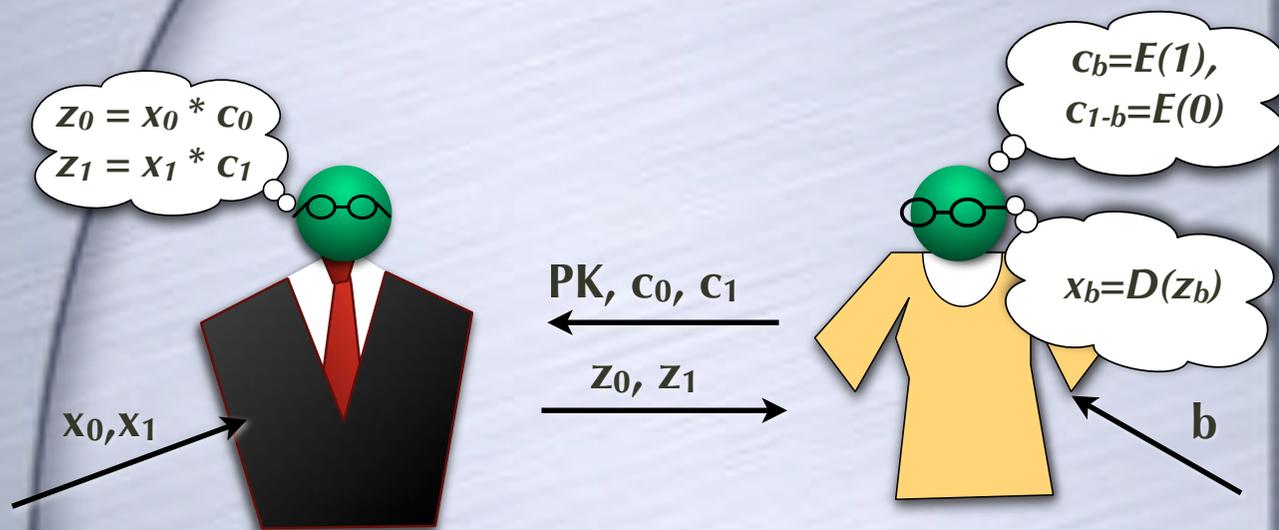
An OT Protocol (passive receiver corruption)

- Using an (unlinkable) rerandomizable encryption scheme
 - Receiver picks (PK, SK) . Sends PK and $E(0), E(1)$ in suitable order
 - Sender “multiplies” c_i with x_i :
 $1 * c := \text{ReRand}(c), 0 * c := E(0)$



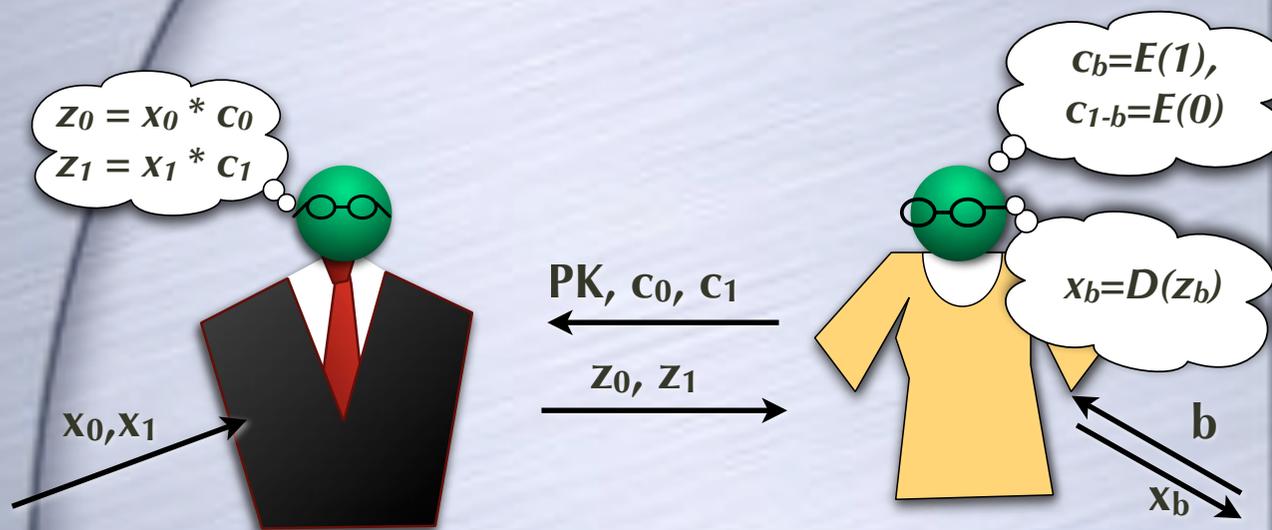
An OT Protocol (passive receiver corruption)

- Using an (unlinkable) rerandomizable encryption scheme
 - Receiver picks (PK, SK) . Sends PK and $E(0), E(1)$ in suitable order
 - Sender “multiplies” c_i with x_i :
 $1 * c := \text{ReRand}(c), 0 * c := E(0)$



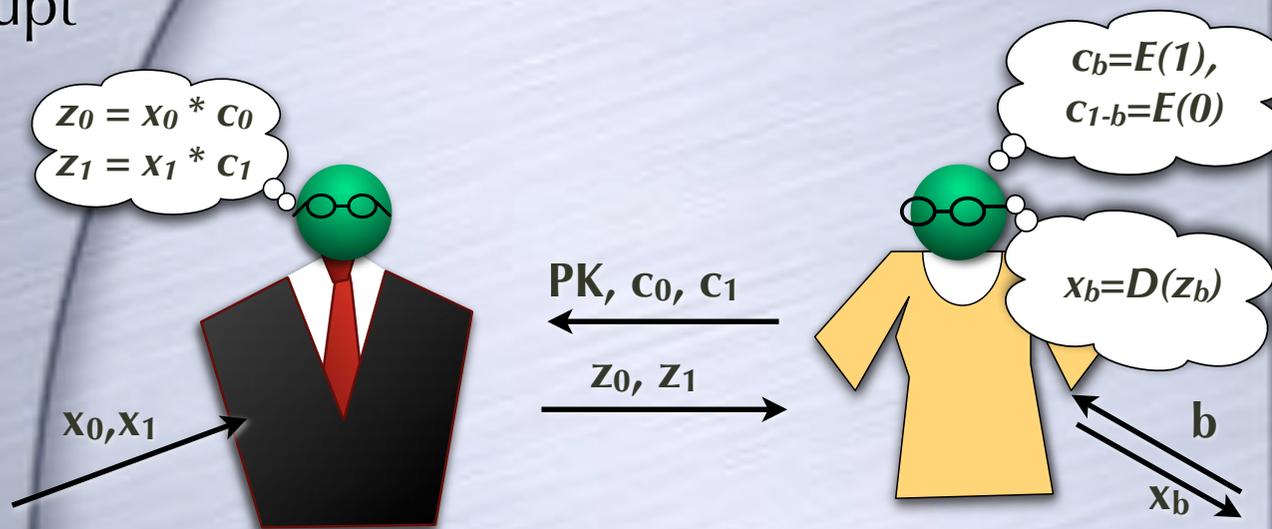
An OT Protocol (passive receiver corruption)

- Using an (unlinkable) rerandomizable encryption scheme
 - Receiver picks (PK, SK) . Sends PK and $E(0), E(1)$ in suitable order
 - Sender “multiplies” c_i with x_i :
 $1 * c := \text{ReRand}(c), 0 * c := E(0)$



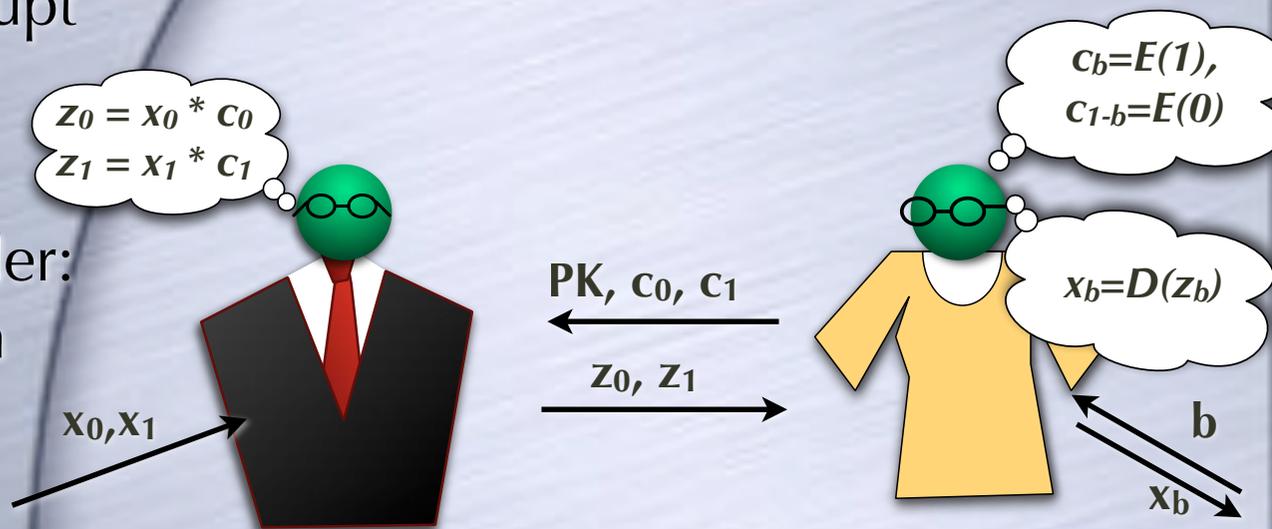
An OT Protocol (passive receiver corruption)

- Using an (unlinkable) rerandomizable encryption scheme
 - Receiver picks (PK, SK) . Sends PK and $E(0), E(1)$ in suitable order
 - Sender “multiplies” c_i with x_i :
 $1 * c := \text{ReRand}(c), 0 * c := E(0)$
- Simulation for passive-corrupt receiver: set $z_b = E(x_b)$ and $z_{1-b} = E(0)$



An OT Protocol (passive receiver corruption)

- Using an (unlinkable) rerandomizable encryption scheme
 - Receiver picks (PK, SK) . Sends PK and $E(0), E(1)$ in suitable order
 - Sender “multiplies” c_i with x_i :
 $1 * c := \text{ReRand}(c), 0 * c := E(0)$
- Simulation for passive-corrupt receiver: set $z_b = E(x_b)$ and $z_{1-b} = E(0)$
- Simulation for corrupt sender: Extract x_0, x_1 by setting both c_0 and c_1 to $E(1)$



Private Information Retrieval

Private Information Retrieval

- Setting: A server holds a large vector of values (“database”). Client wants to retrieve the value at a particular index i

Private Information Retrieval

- Setting: A server holds a large vector of values (“database”). Client wants to retrieve the value at a particular index i
 - Client wants privacy against an honest-but-curious server

Private Information Retrieval

- Setting: A server holds a large vector of values (“database”). Client wants to retrieve the value at a particular index i
 - Client wants privacy against an honest-but-curious server
 - Server has no security requirements

Private Information Retrieval

- Setting: A server holds a large vector of values (“database”). Client wants to retrieve the value at a particular index i
 - Client wants privacy against an honest-but-curious server
 - Server has no security requirements
- Trivial solution: Server sends the entire vector to the client

Private Information Retrieval

- Setting: A server holds a large vector of values (“database”). Client wants to retrieve the value at a particular index i
 - Client wants privacy against an honest-but-curious server
 - Server has no security requirements
- Trivial solution: Server sends the entire vector to the client
- PIR: to do it with significantly less communication

Private Information Retrieval

- Setting: A server holds a large vector of values (“database”). Client wants to retrieve the value at a particular index i
 - Client wants privacy against an honest-but-curious server
 - Server has no security requirements
- Trivial solution: Server sends the entire vector to the client
- PIR: to do it with significantly less communication
 - Variant (we don't look at): multiple-server PIR, with non-colluding servers

Private Information Retrieval

- Setting: A server holds a large vector of values (“database”). Client wants to retrieve the value at a particular index i
 - Client wants privacy against an honest-but-curious server
 - Server has no security requirements
- Trivial solution: Server sends the entire vector to the client
- PIR: to do it with significantly less communication
 - Variant (we don't look at): multiple-server PIR, with non-colluding servers
- Tool: Additively homomorphic encryption

Paillier's Scheme

Paillier's Scheme

- Over $\mathbb{Z}_{n^2}^* \simeq \mathbb{Z}_n \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within $2x$ of each other

Paillier's Scheme

- Over $\mathbb{Z}_{n^2}^* \simeq \mathbb{Z}_n \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within $2x$ of each other
To ensure $\gcd(n, \phi(n))=1$

Paillier's Scheme

- Over $\mathbb{Z}_{n^2}^* \simeq \mathbb{Z}_n \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within $2x$ of each other
To ensure $\gcd(n, \phi(n))=1$
- Isomorphism: $\psi(a,b) = g^a b^n$ where $g=(1+n)$

Paillier's Scheme

- Over $\mathbb{Z}_{n^2}^* \simeq \mathbb{Z}_n \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within 2x of each other
To ensure $\gcd(n, \phi(n))=1$
- Isomorphism: $\psi(a,b) = g^a b^n$ where $g=(1+n)$
- $\text{Enc}(m) = \psi(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*

Paillier's Scheme

- Over $\mathbb{Z}_{n^2}^* \simeq \mathbb{Z}_n \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within 2x of each other
 - Isomorphism: $\psi(a,b) = g^a b^n$ where $g=(1+n)$
To ensure $\gcd(n,\phi(n))=1$
- $\text{Enc}(m) = \psi(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*
- ψ can be inverted if p,q known

Paillier's Scheme

- Over $\mathbb{Z}_{n^2}^* \simeq \mathbb{Z}_n \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within 2x of each other
 - Isomorphism: $\psi(a,b) = g^a b^n$ where $g=(1+n)$
To ensure $\gcd(n,\phi(n))=1$
- $\text{Enc}(m) = \psi(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*
- ψ can be inverted if p,q known
- (Additive) Homomorphism: $\text{Enc}(m).\text{Enc}(m')$ is $\text{Enc}(m+m')$

Paillier's Scheme

- Over $\mathbb{Z}_{n^2}^* \simeq \mathbb{Z}_n \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within 2x of each other
To ensure $\gcd(n, \phi(n))=1$
- Isomorphism: $\psi(a,b) = g^a b^n$ where $g=(1+n)$
- $\text{Enc}(m) = \psi(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*
- ψ can be inverted if p,q known
- (Additive) Homomorphism: $\text{Enc}(m).\text{Enc}(m')$ is $\text{Enc}(m+m')$
 - $\psi(m,r).\psi(m',r') = \psi(m+m',r.r')$

Paillier's Scheme

- Over $\mathbb{Z}_{n^2}^* \simeq \mathbb{Z}_n \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within 2x of each other
To ensure $\gcd(n, \phi(n))=1$
- Isomorphism: $\psi(a,b) = g^a b^n$ where $g=(1+n)$
- $\text{Enc}(m) = \psi(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*
- ψ can be inverted if p,q known
- (Additive) Homomorphism: $\text{Enc}(m) \cdot \text{Enc}(m')$ is $\text{Enc}(m+m')$
 - $\psi(m,r) \cdot \psi(m',r') = \psi(m+m', r \cdot r')$ in \mathbb{Z}_n

Paillier's Scheme

- Over $\mathbb{Z}_{n^2}^* \simeq \mathbb{Z}_n \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within $2x$ of each other
 To ensure $\gcd(n, \phi(n))=1$
 - Isomorphism: $\psi(a,b) = g^a b^n$ where $g=(1+n)$
- $\text{Enc}(m) = \psi(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*
- ψ can be inverted if p,q known
- (Additive) Homomorphism: $\text{Enc}(m) \cdot \text{Enc}(m')$ is $\text{Enc}(m+m')$

$$\text{in } \mathbb{Z}_{n^2}^* \quad \psi(m,r) \cdot \psi(m',r') = \psi(m+m', r \cdot r') \quad \text{in } \mathbb{Z}_n$$

Paillier's Scheme

- Over $\mathbb{Z}_{n^2}^* \simeq \mathbb{Z}_n \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within $2x$ of each other
 To ensure $\gcd(n, \phi(n))=1$
 - Isomorphism: $\psi(a,b) = g^a b^n$ where $g=(1+n)$
- $\text{Enc}(m) = \psi(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*
- ψ can be inverted if p,q known
- (Additive) Homomorphism: $\text{Enc}(m).\text{Enc}(m')$ is $\text{Enc}(m+m')$
- $\psi(m,r).\psi(m',r') = \psi(m+m',r.r')$
 - in $\mathbb{Z}_{n^2}^*$ (left side)
 - in \mathbb{Z}_n (right side)
- Security: Rerandomization can be made perfect. IND-CPA secure under "Decisional Composite Residuosity" assumption

Paillier's Scheme

- Over $\mathbb{Z}_{n^2}^* \simeq \mathbb{Z}_n \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within $2x$ of each other
 To ensure $\gcd(n, \phi(n))=1$
 - Isomorphism: $\psi(a,b) = g^a b^n$ where $g=(1+n)$
- $\text{Enc}(m) = \psi(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*
- ψ can be inverted if p,q known
- (Additive) Homomorphism: $\text{Enc}(m).\text{Enc}(m')$ is $\text{Enc}(m+m')$
- $\psi(m,r).\psi(m',r') = \psi(m+m',r.r')$
 - in $\mathbb{Z}_{n^2}^*$ (left side)
 - in \mathbb{Z}_n (right side)
- Security: Rerandomization can be made perfect. IND-CPA secure under "Decisional Composite Residuosity" assumption
 - Given $n=pq$ (but not p,q), $\psi(0,\text{rand})$ looks random (i.e. like $\psi(\text{rand},\text{rand})$)

Private Information Retrieval

Private Information Retrieval

- Using additive homomorphic encryption (need not be unlinkable)

Private Information Retrieval

- Using additive homomorphic encryption (need not be unlinkable)
 - Client sends some encrypted representation of the index (need CPA security here)

Private Information Retrieval

- Using additive homomorphic encryption (need not be unlinkable)
 - Client sends some encrypted representation of the index (need CPA security here)
 - Server operates on this representation (homomorphically) using the entire database, so that the message in the resulting encrypted data has the relevant answer (and maybe more). It sends this (short) encrypted data to client, who decrypts to get answer (depends on correctness here)

Private Information Retrieval

- Using additive homomorphic encryption (need not be unlinkable)
 - Client sends some encrypted representation of the index (need CPA security here)
 - Server operates on this representation (homomorphically) using the entire database, so that the message in the resulting encrypted data has the relevant answer (and maybe more). It sends this (short) encrypted data to client, who decrypts to get answer (depends on correctness here)
 - In the following: database values are integers in $[0, m)$; homom. enc. over a group with an element 1 s.t. $\text{ord}(1) \geq m$. For integer x and ciphertext \underline{c} , define $x^* \underline{c}$ using "repeated doubling": $0^* \underline{c} = E(0)$; $1^* \underline{c} = \underline{c}$; $(a+b)^* \underline{c} = \text{Add}(a^* \underline{c}, b^* \underline{c})$.

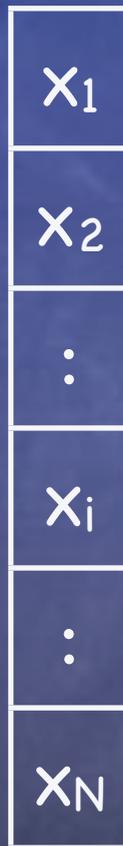
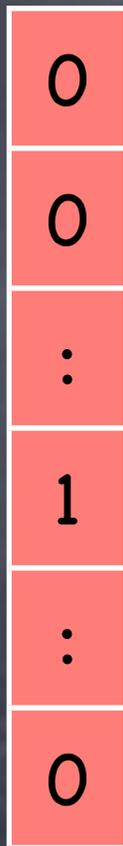
Private Information Retrieval



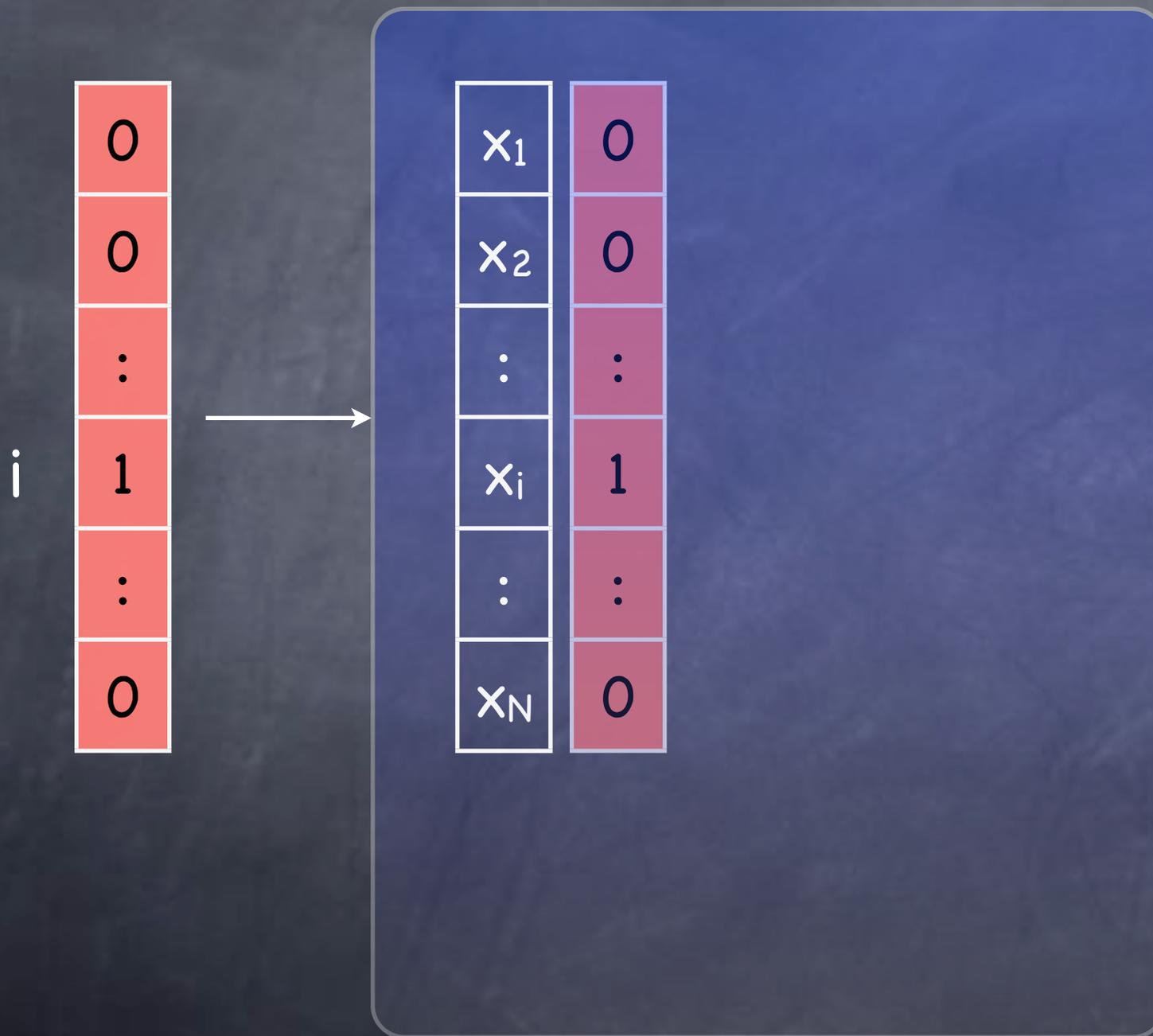
i

Private Information Retrieval

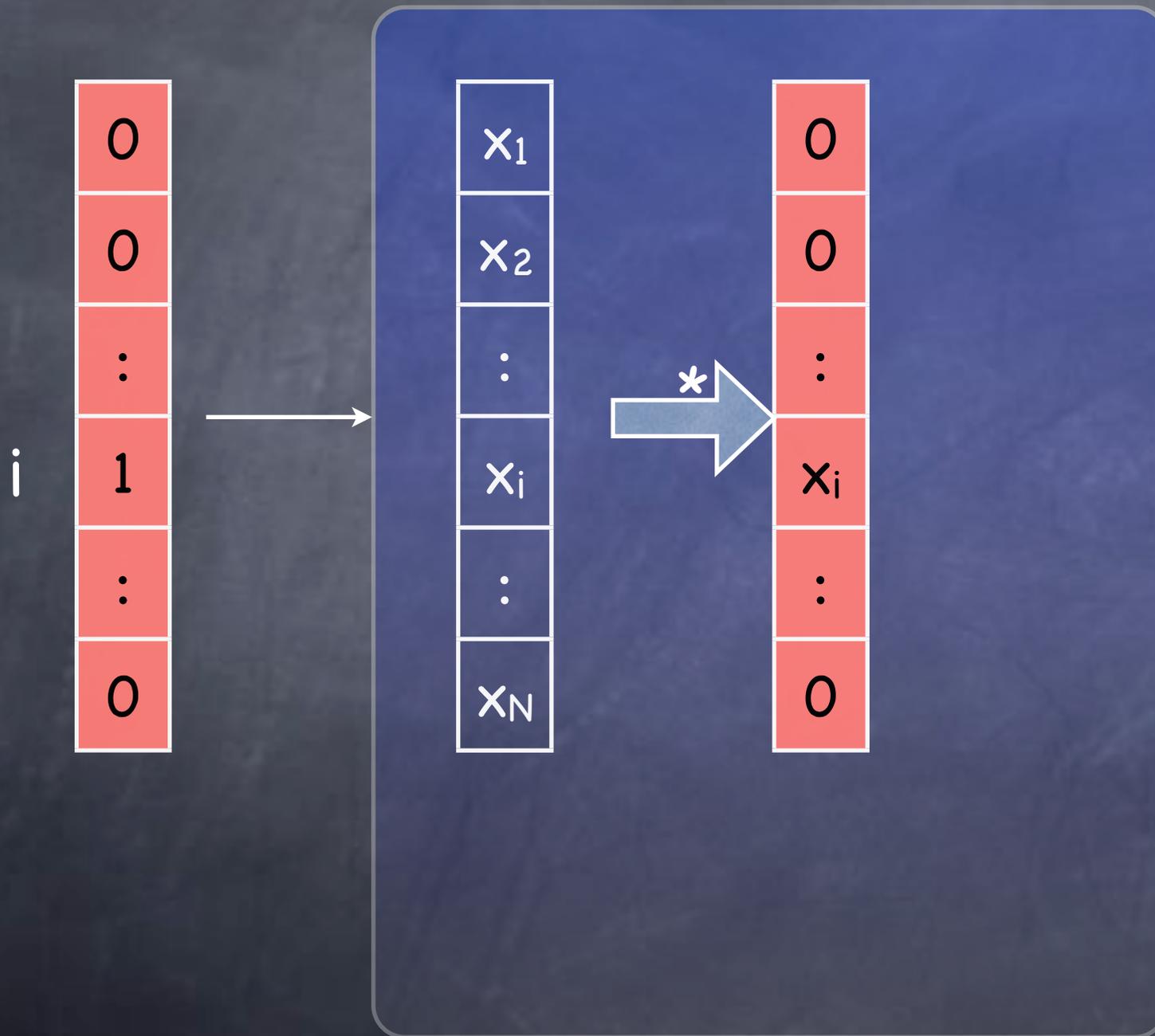
i



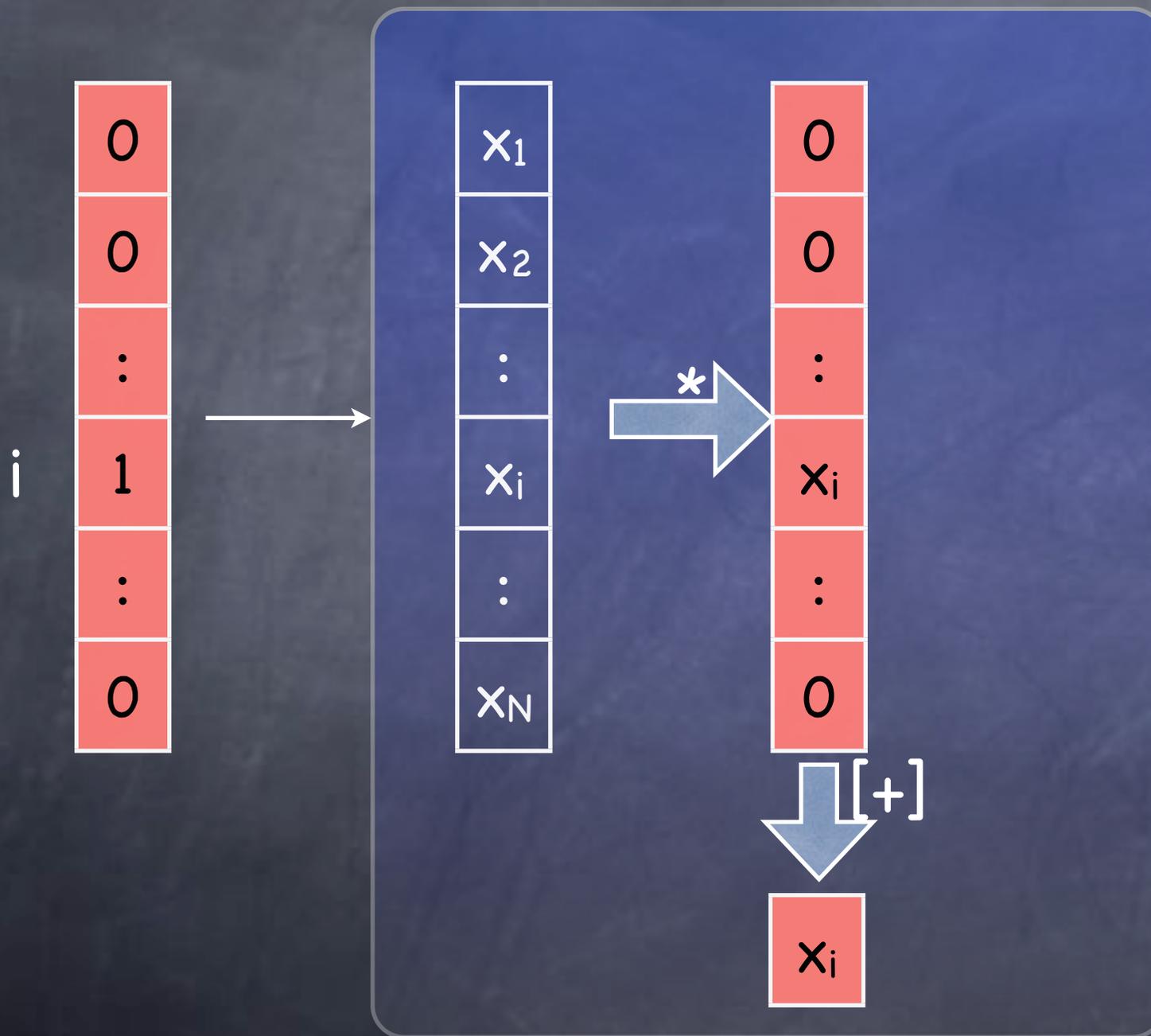
Private Information Retrieval



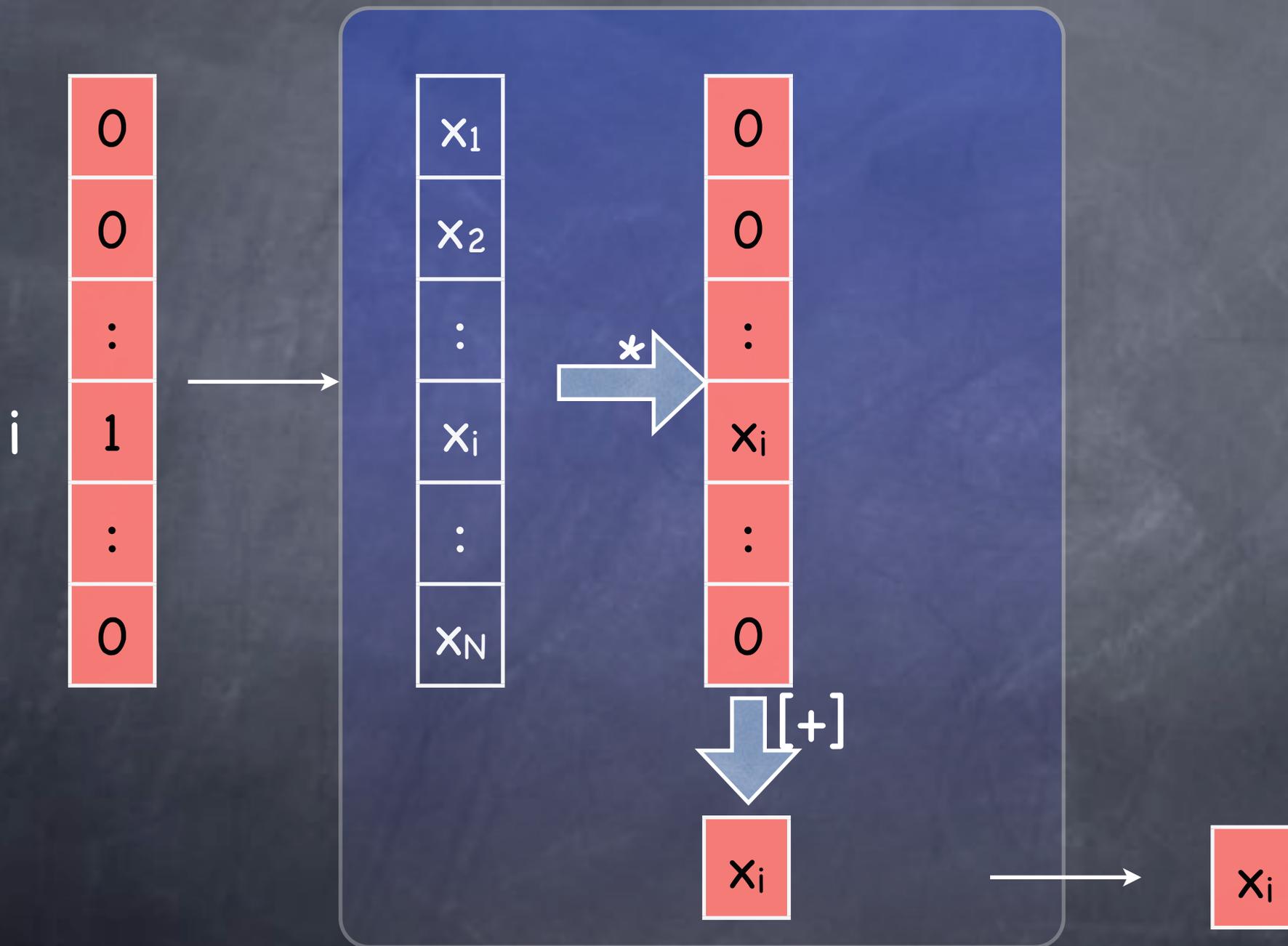
Private Information Retrieval



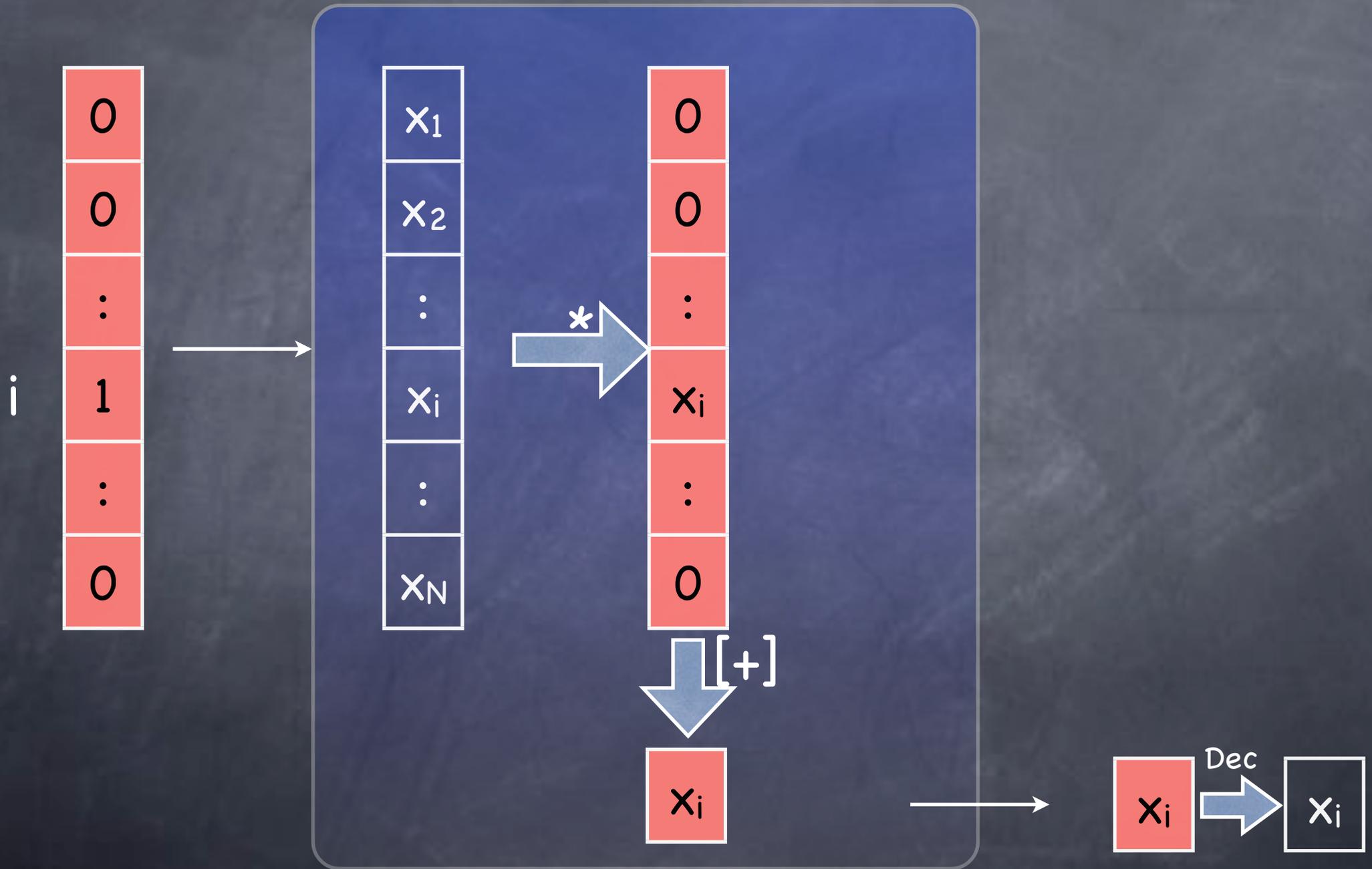
Private Information Retrieval



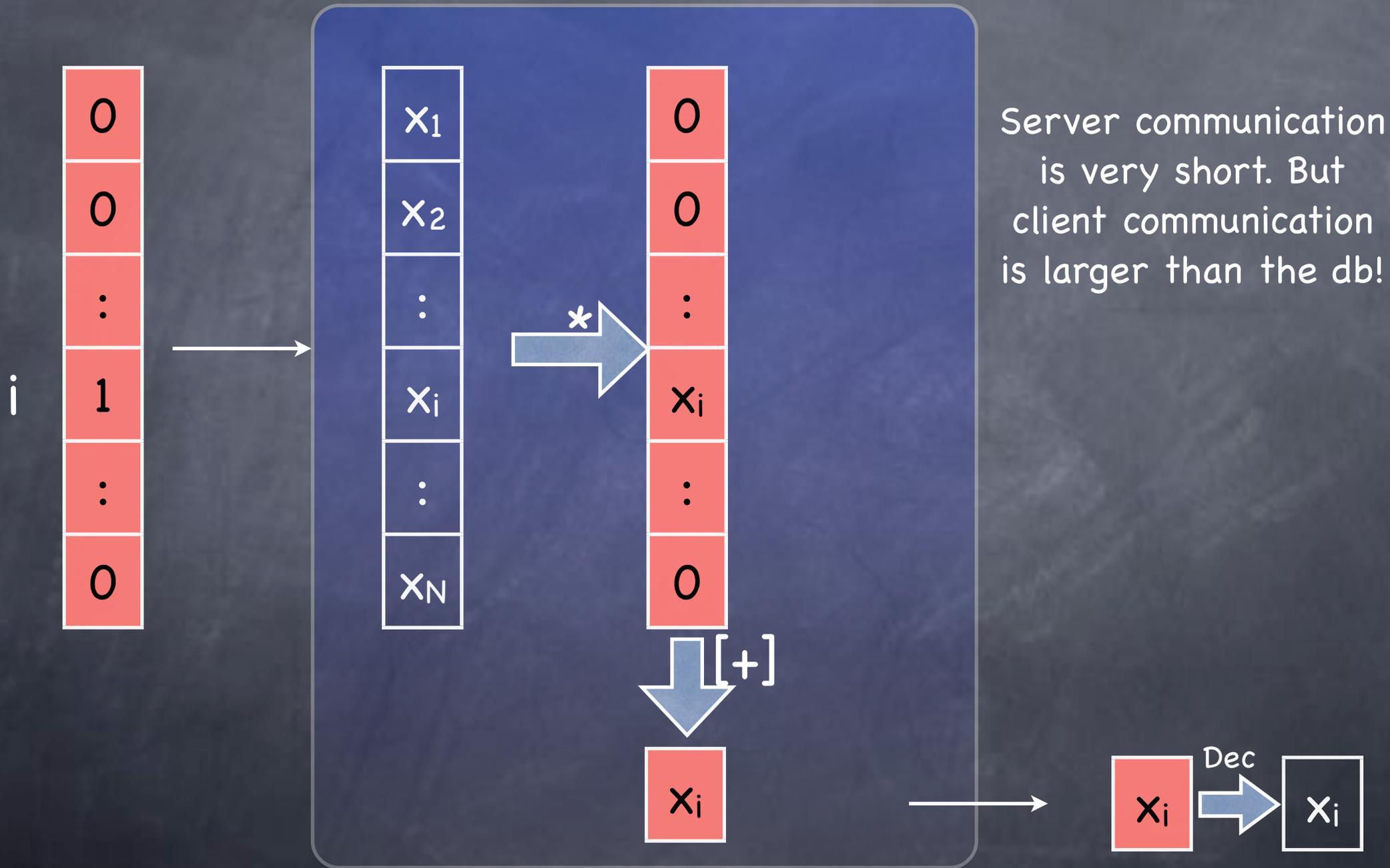
Private Information Retrieval



Private Information Retrieval



Private Information Retrieval



Private Information Retrieval

x_{11}					x_{1N}
x_{21}					x_{2N}
:					:
x_{i1}			x_{ij}		x_{iN}
:					:
x_N					x_{NN}

Private Information Retrieval

0
0
:
1
:
0

x_{11}					x_{1N}
x_{21}					x_{2N}
:					:
x_{i1}			x_{ij}		x_{iN}
:					:
x_N					x_{NN}

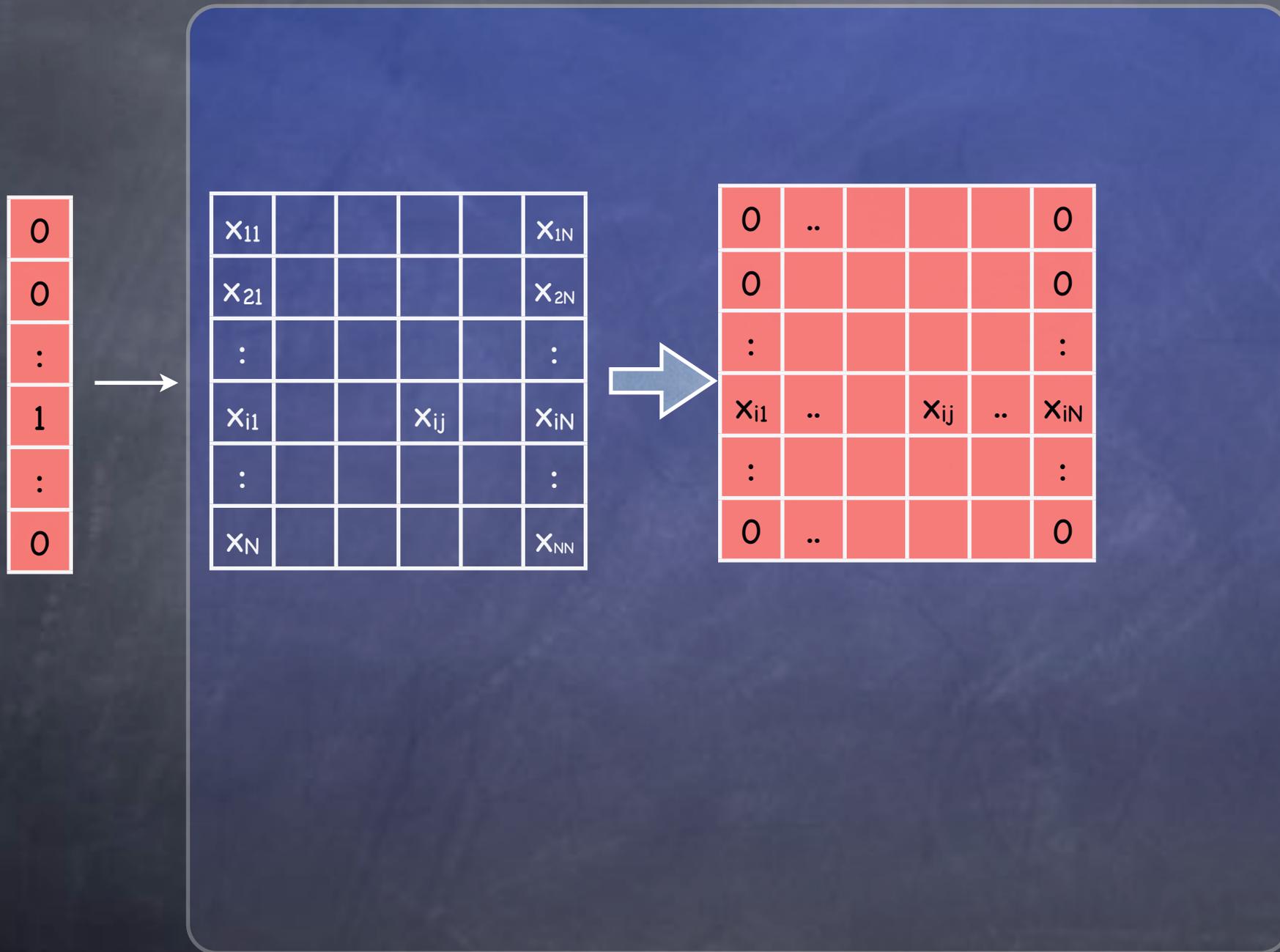
Private Information Retrieval

0
0
:
1
:
0

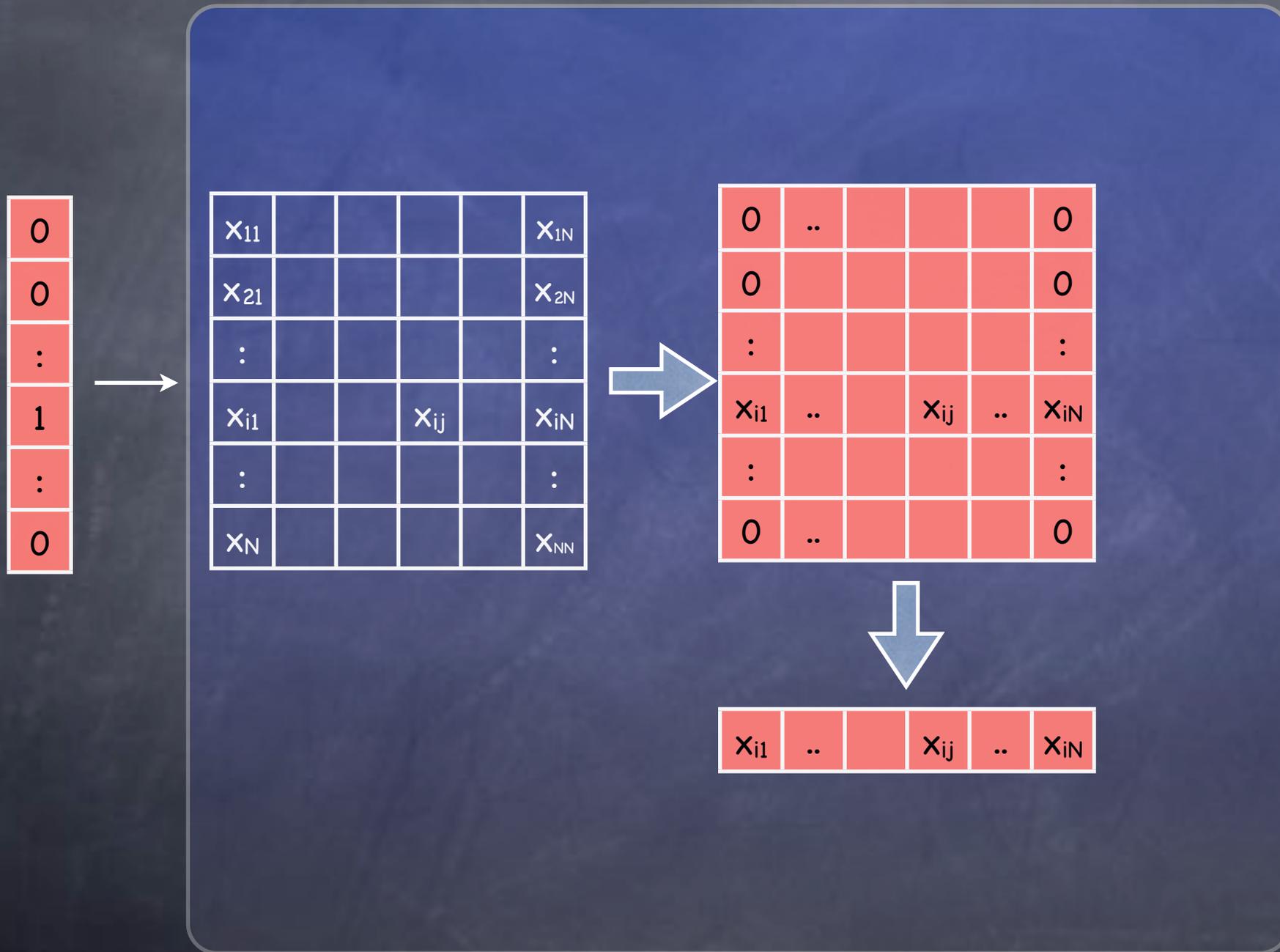


x_{11}					x_{1N}
x_{21}					x_{2N}
:					:
x_{i1}			x_{ij}		x_{iN}
:					:
x_N					x_{NN}

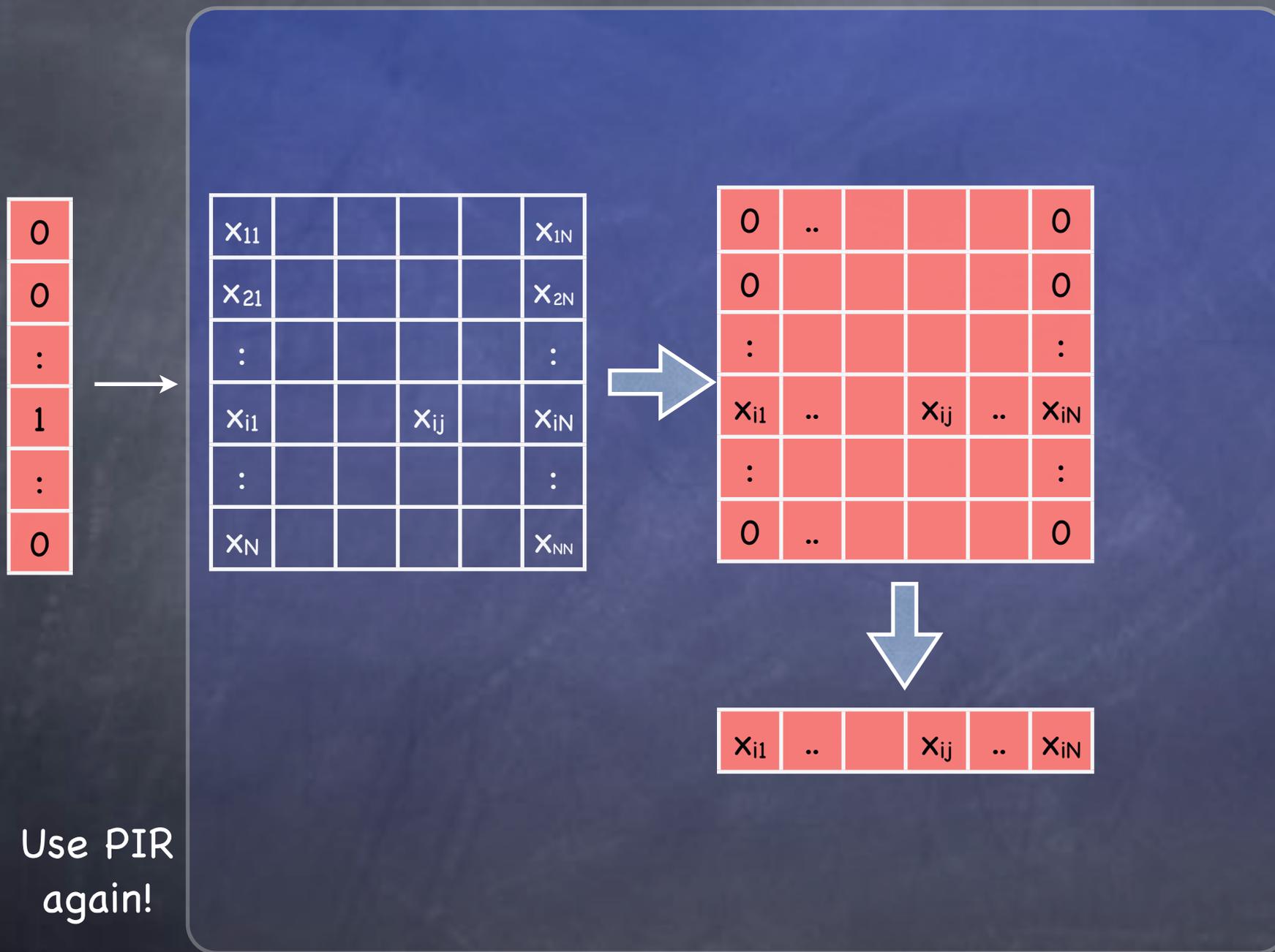
Private Information Retrieval



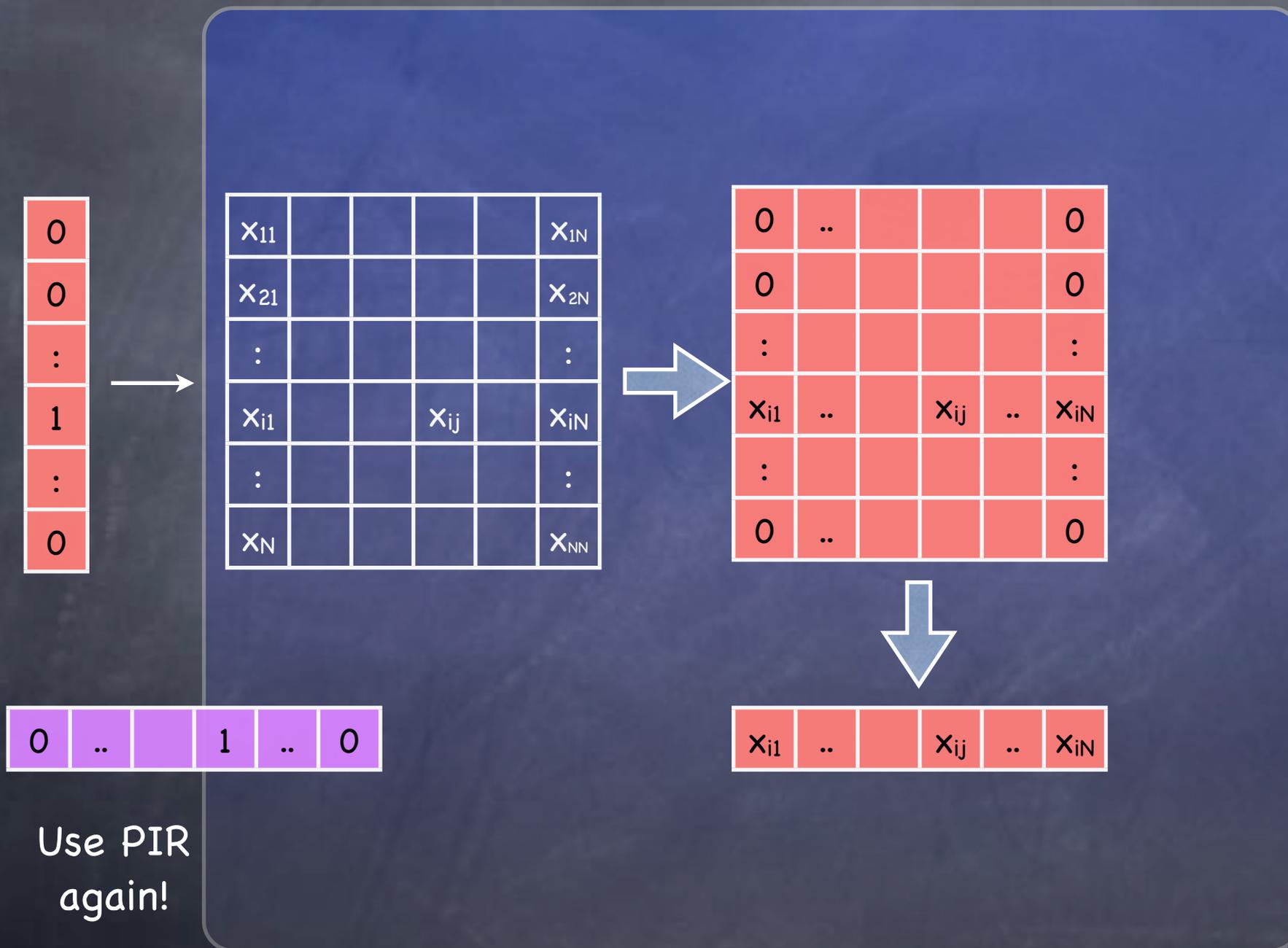
Private Information Retrieval



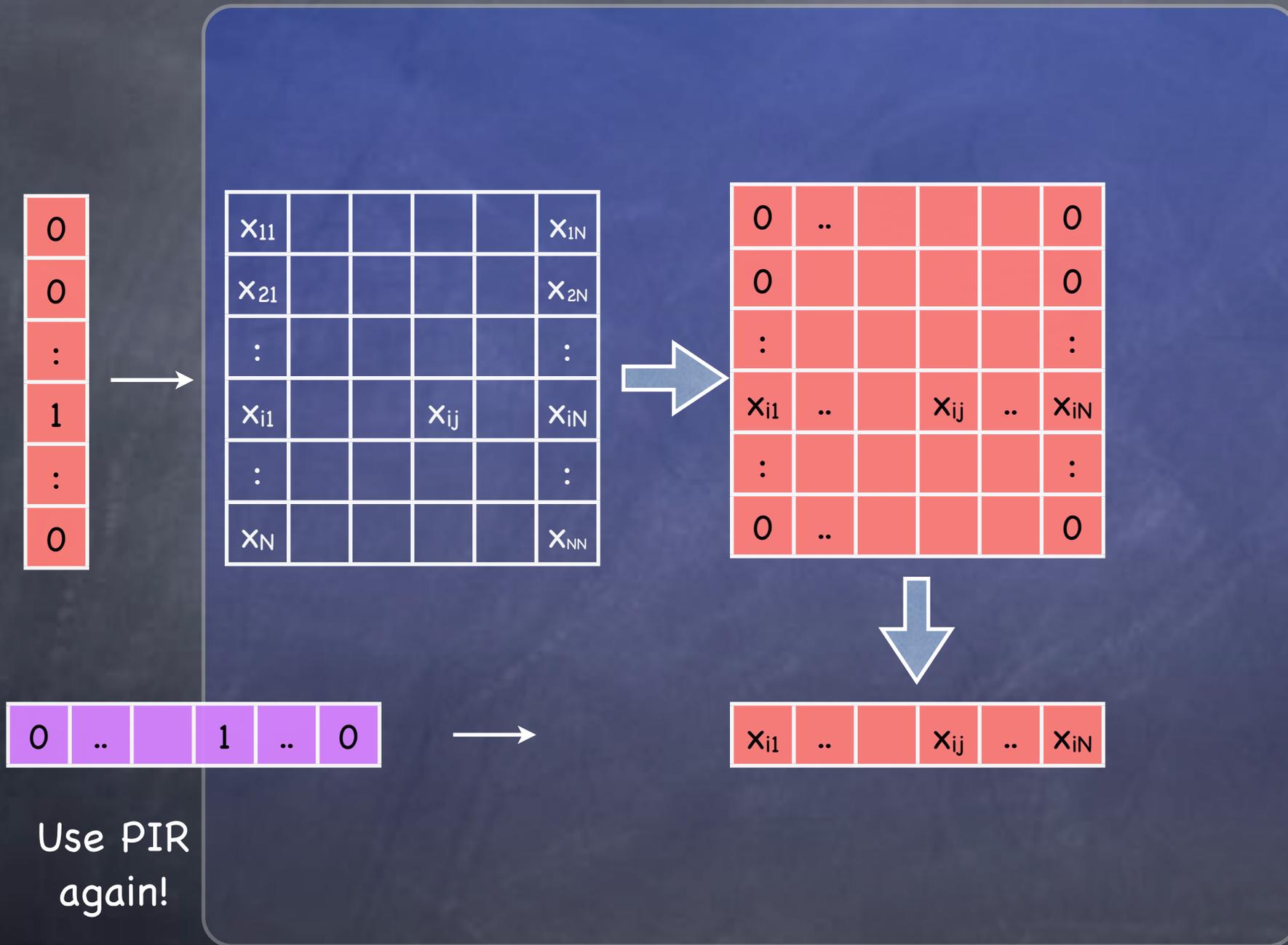
Private Information Retrieval



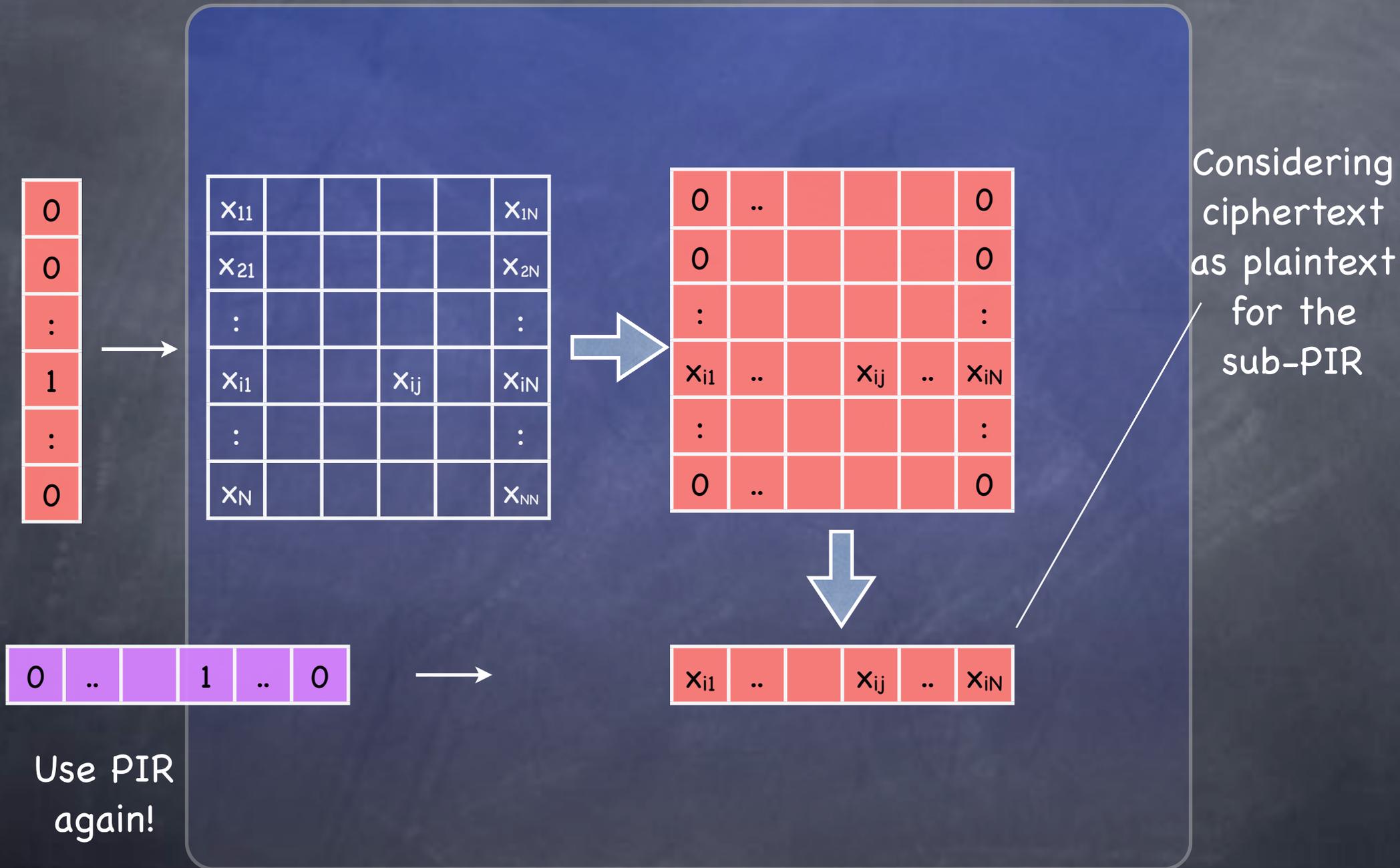
Private Information Retrieval



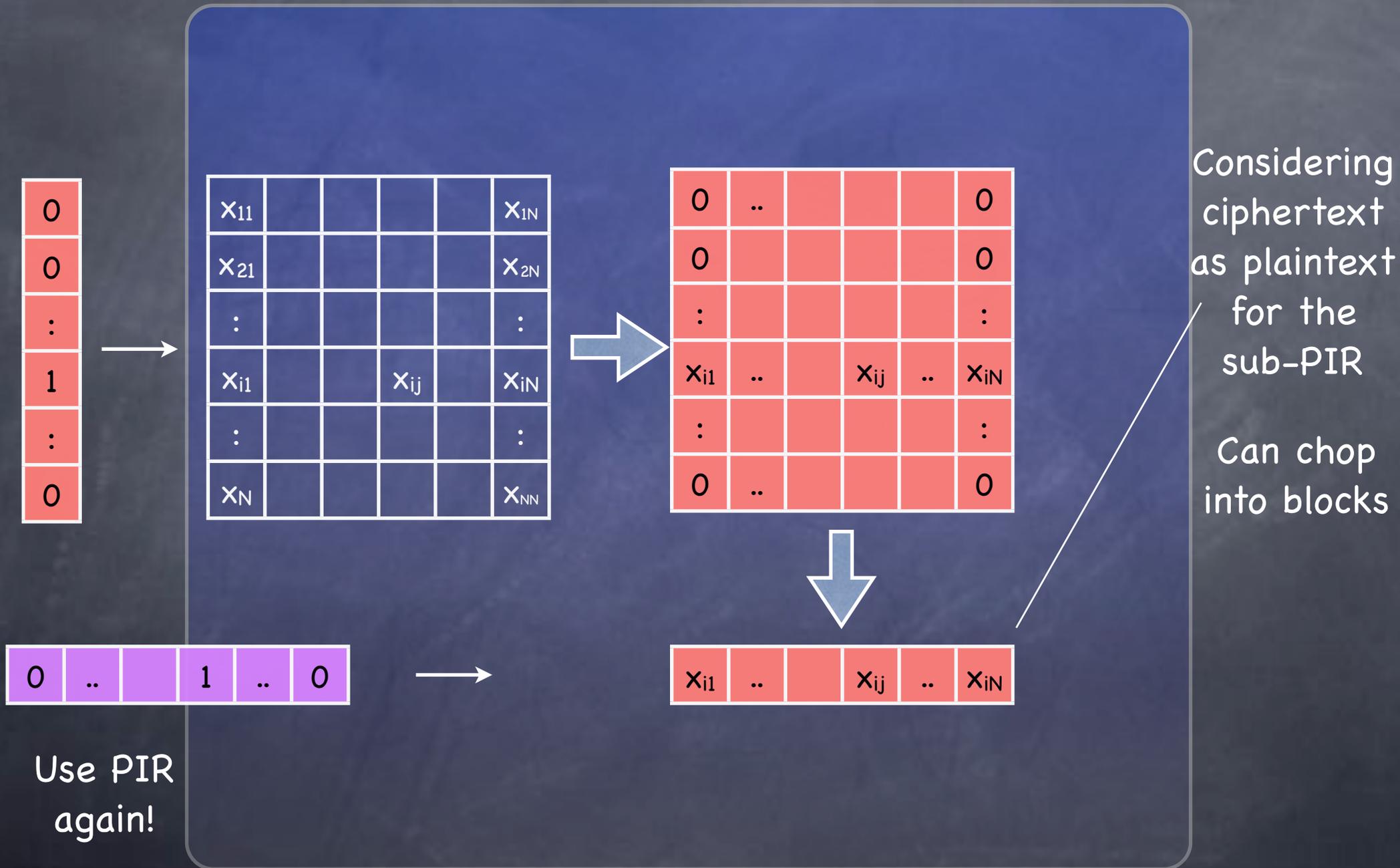
Private Information Retrieval



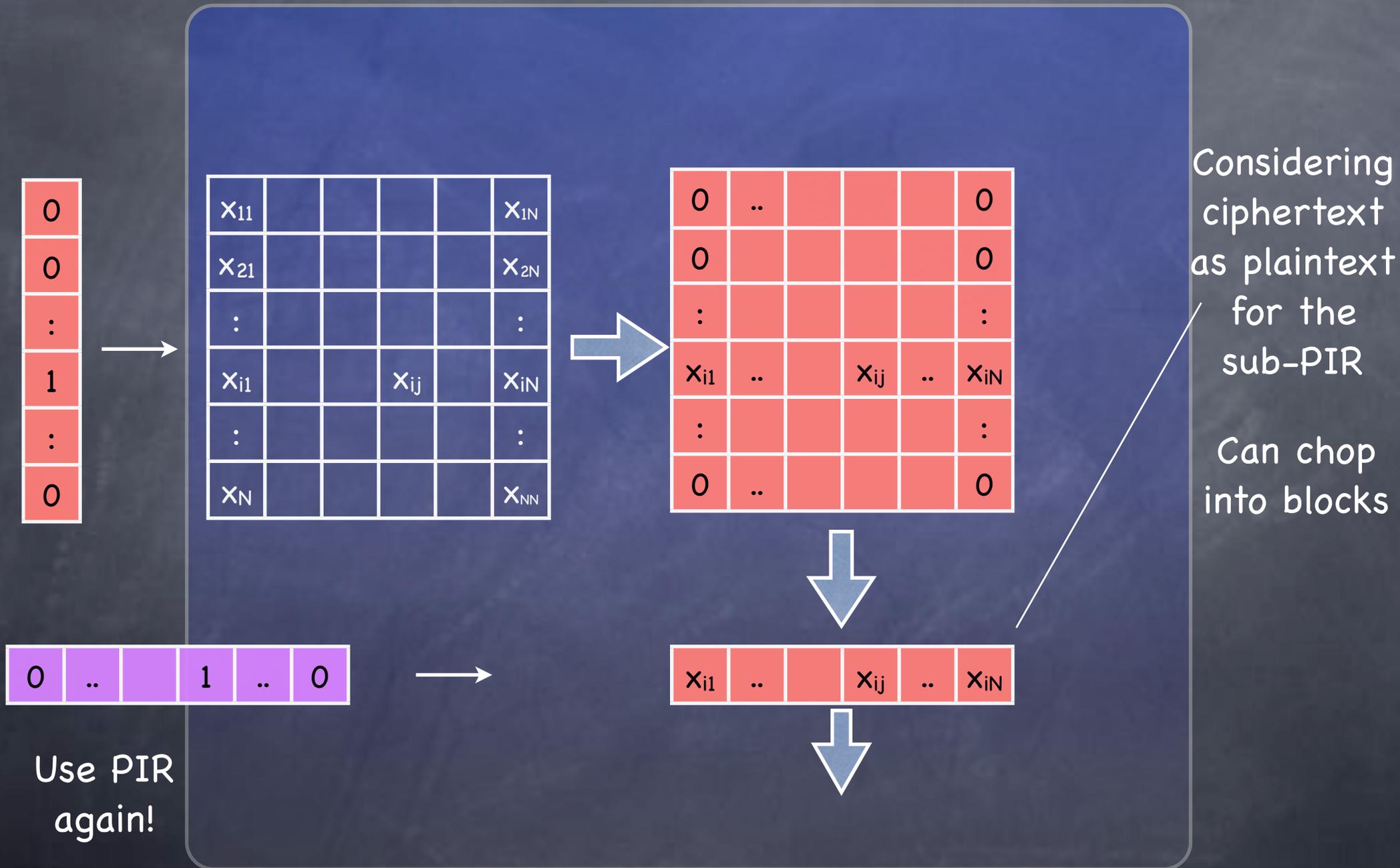
Private Information Retrieval



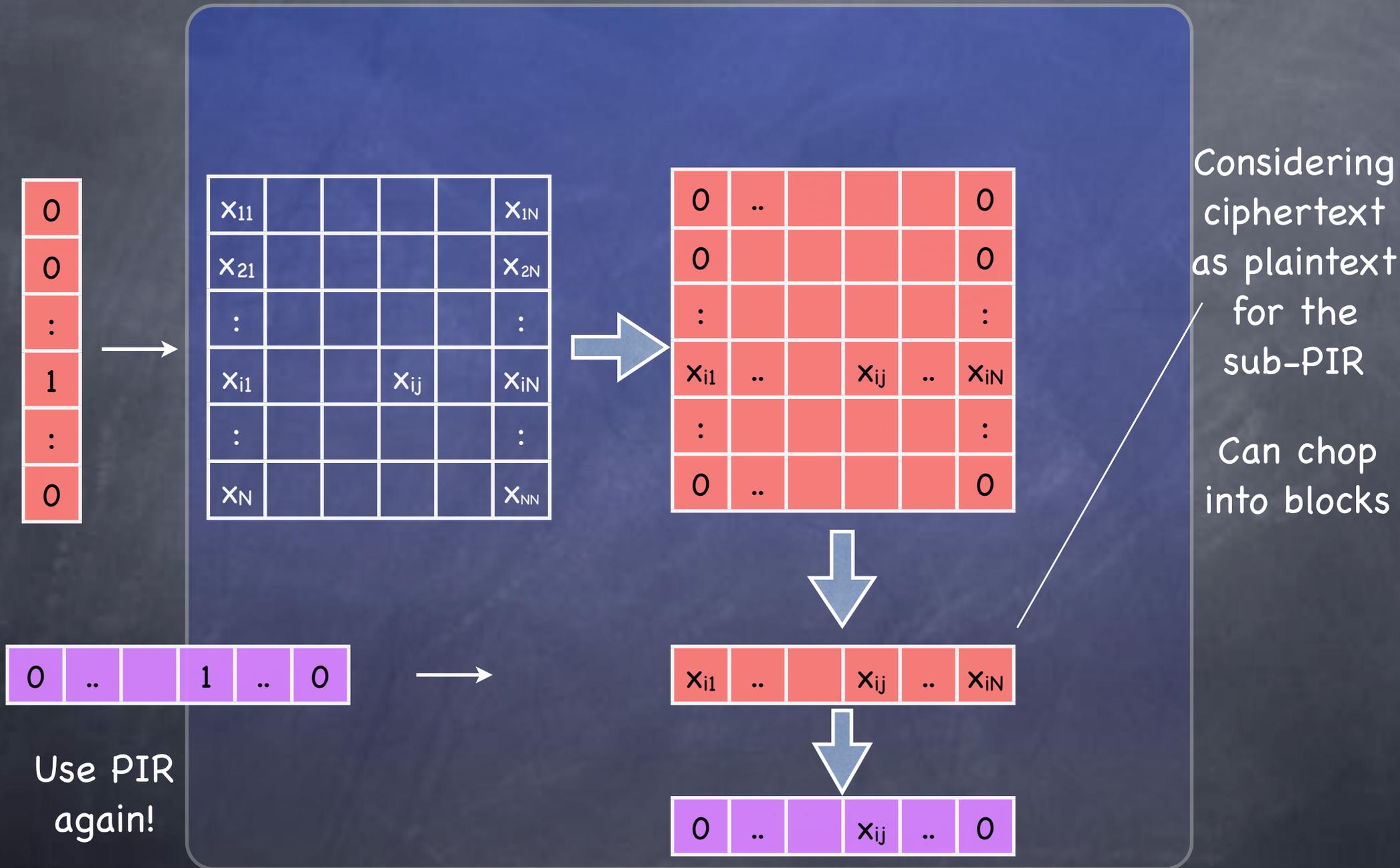
Private Information Retrieval



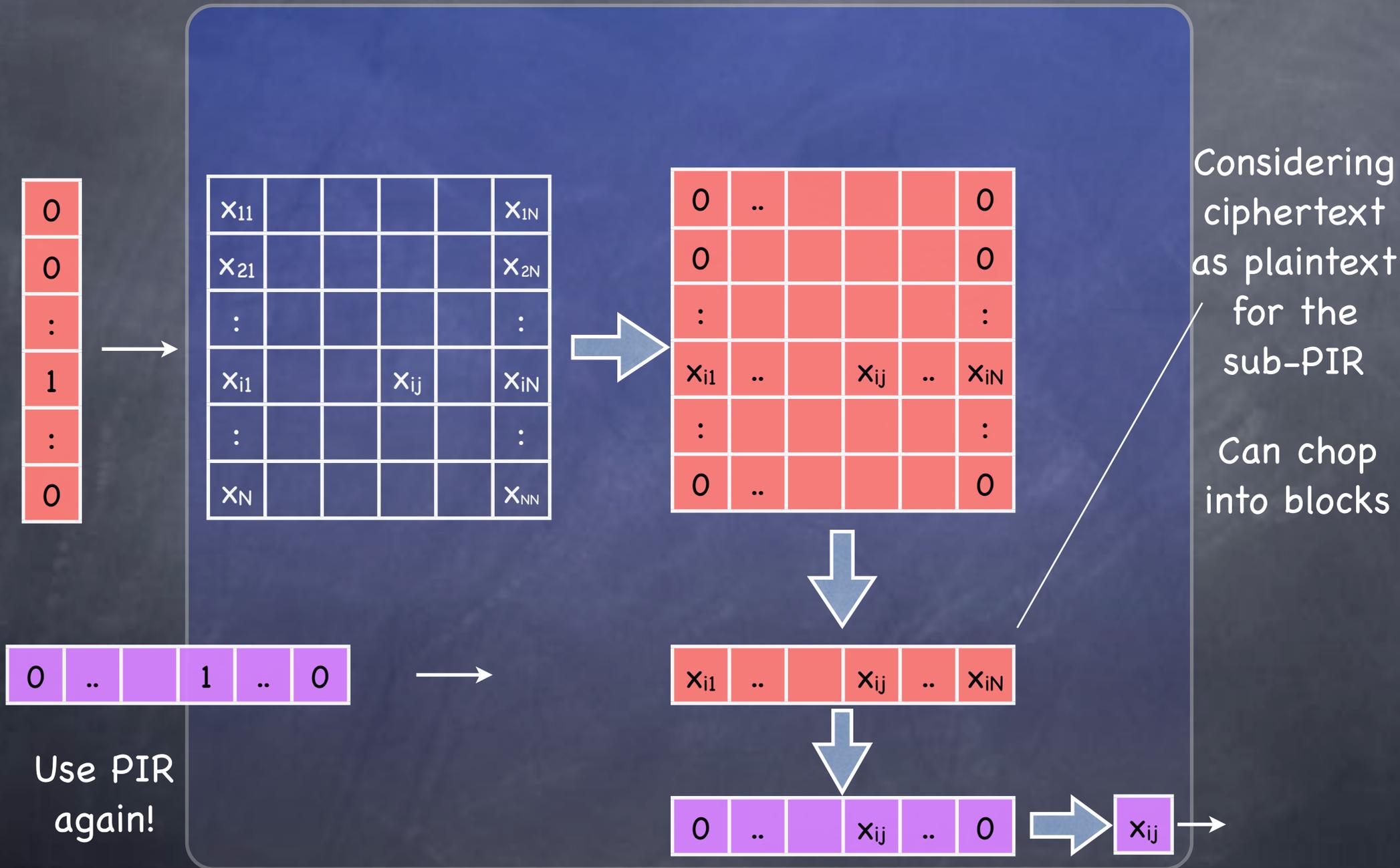
Private Information Retrieval



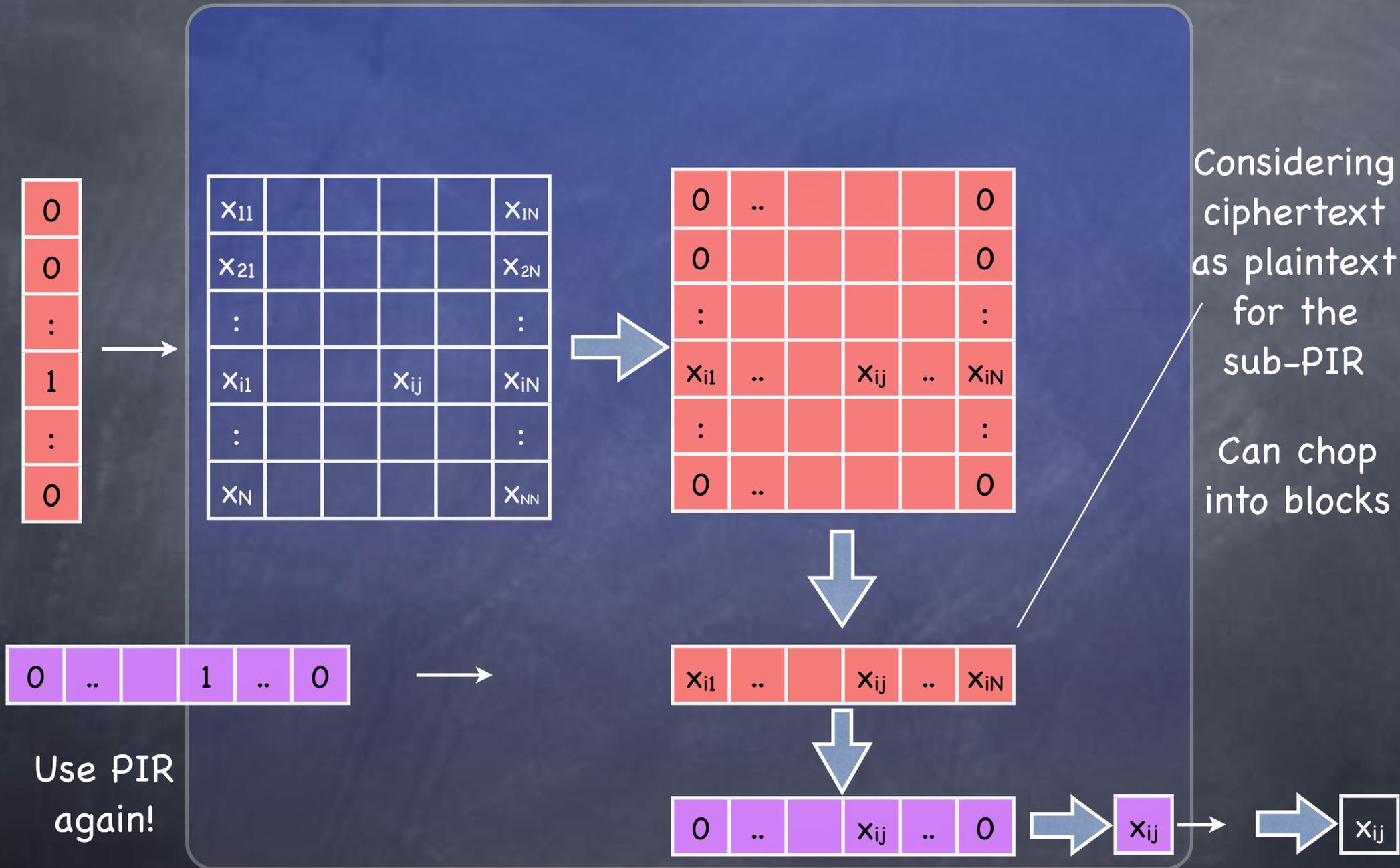
Private Information Retrieval



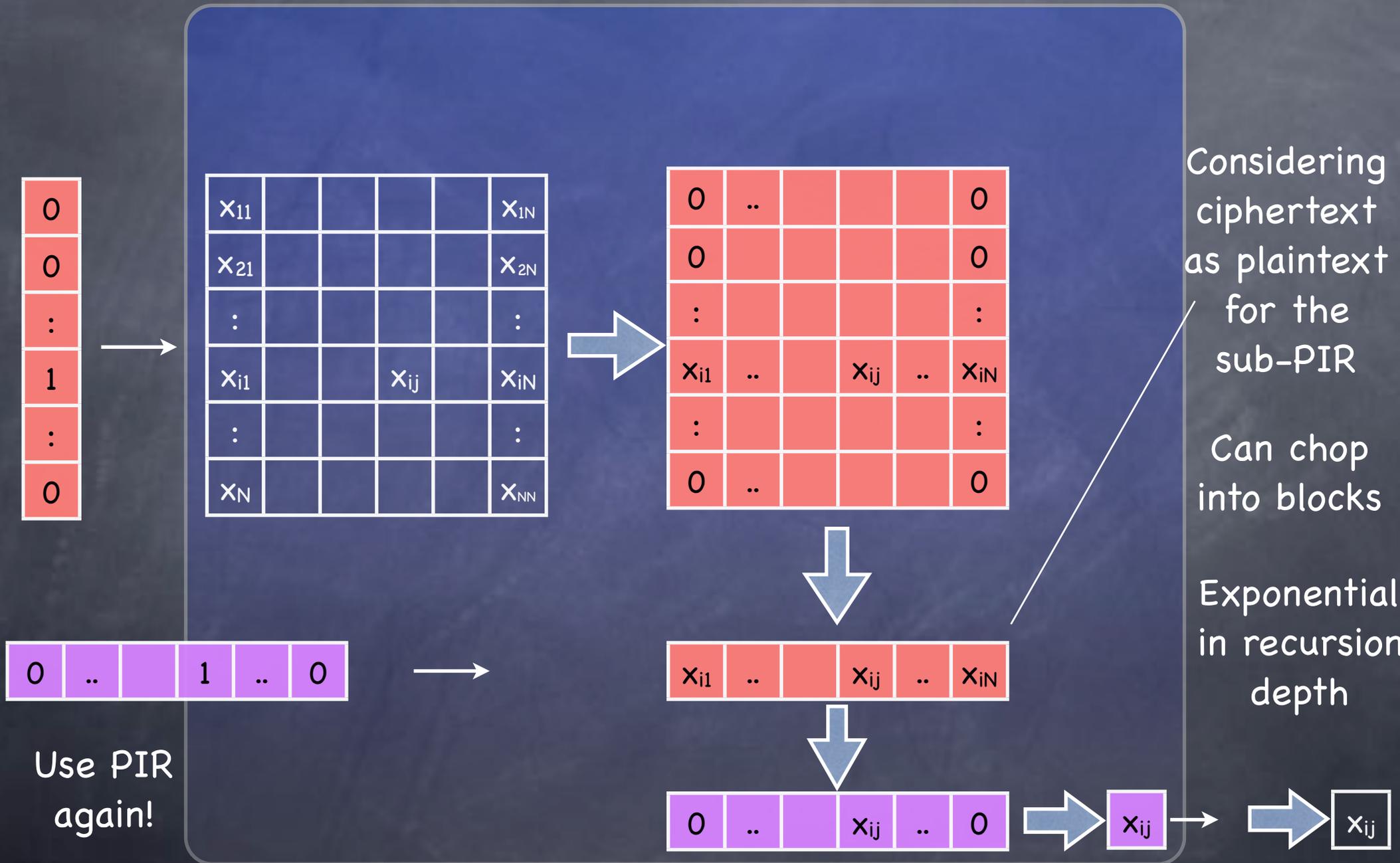
Private Information Retrieval



Private Information Retrieval



Private Information Retrieval



Private Information Retrieval

Private Information Retrieval

- Can dramatically improve efficiency if given an efficient “recursive” homomorphic encryption scheme

Private Information Retrieval

- Can dramatically improve efficiency if given an efficient “recursive” homomorphic encryption scheme
 - Ciphertext in one level can be used as plaintext in the next level

Private Information Retrieval

- Can dramatically improve efficiency if given an efficient “recursive” homomorphic encryption scheme
 - Ciphertext in one level can be used as plaintext in the next level
 - Ciphertext size increases only “additively” from level to level

Private Information Retrieval

- Can dramatically improve efficiency if given an efficient “recursive” homomorphic encryption scheme
 - Ciphertext in one level can be used as plaintext in the next level
 - Ciphertext size increases only “additively” from level to level
- Does such a family of encryption schemes exist?

Flexibility and Efficiency Issues

Flexibility and Efficiency Issues

- Public-Key (i.e., n) fixes the group for homomorphic operation (i.e., \mathbb{Z}_n)

Flexibility and Efficiency Issues

- Public-Key (i.e., n) fixes the group for homomorphic operation (i.e., \mathbb{Z}_n)
 - Sender cannot choose the group

Flexibility and Efficiency Issues

- Public-Key (i.e., n) fixes the group for homomorphic operation (i.e., \mathbb{Z}_n)
 - Sender cannot choose the group
- Size of ciphertext about double that of the plaintext

Flexibility and Efficiency Issues

- Public-Key (i.e., n) fixes the group for homomorphic operation (i.e., \mathbb{Z}_n)
 - Sender cannot choose the group
- Size of ciphertext about double that of the plaintext
 - Note: can't use "hybrid encryption" if homomorphic property is to be preserved

Damgård-Jurik Scheme

Damgård–Jurik Scheme

- Over $\mathbb{Z}_{n^{s+1}}^*$ $\cong \mathbb{Z}_n^s \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within $2x$ of each other

Damgård-Jurik Scheme

- Over $\mathbb{Z}_{n^{s+1}}^* \simeq \mathbb{Z}_{ns} \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within $2x$ of each other
 - Isomorphism: $\psi_s(a,b) = g^a b^{n^s}$ where $g=(1+n)$

Damgård-Jurik Scheme

- Over $\mathbb{Z}_{n(s+1)}^* \simeq \mathbb{Z}_{ns} \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within 2x of each other
 - Isomorphism: $\psi_s(a,b) = g^a b^{n^s}$ where $g=(1+n)$
- $\text{Enc}(m) = \psi_s(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*

Damgård-Jurik Scheme

- Over $\mathbb{Z}_{n(s+1)}^* \simeq \mathbb{Z}_{ns} \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within 2x of each other
 - Isomorphism: $\psi_s(a,b) = g^a b^{n^s}$ where $g=(1+n)$
- $\text{Enc}(m) = \psi_s(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*
- ψ_s can still be inverted if p,q known (but more involved)

Damgård-Jurik Scheme

- Over $\mathbb{Z}_{n(s+1)}^* \approx \mathbb{Z}_{ns} \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within 2x of each other
 - Isomorphism: $\psi_s(a,b) = g^a b^{n^s}$ where $g=(1+n)$
- $\text{Enc}(m) = \psi_s(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*
- ψ_s can still be inverted if p,q known (but more involved)
- Homomorphism: $\text{Enc}(m).\text{Enc}(m')$ is $\text{Enc}(m+m')$

Damgård-Jurik Scheme

- Over $\mathbb{Z}_{n(s+1)}^* \approx \mathbb{Z}_{ns} \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within 2x of each other
 - Isomorphism: $\psi_s(a,b) = g^a b^{n^s}$ where $g=(1+n)$
- $\text{Enc}(m) = \psi_s(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*
- ψ_s can still be inverted if p,q known (but more involved)
- Homomorphism: $\text{Enc}(m).\text{Enc}(m')$ is $\text{Enc}(m+m')$
 - $\psi_s(m,r).\psi_s(m',r') = \psi_s(m+m',r.r')$

Damgård-Jurik Scheme

- Over $\mathbb{Z}_{n^{(s+1)}}^* \simeq \mathbb{Z}_{ns} \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within 2x of each other
 - Isomorphism: $\psi_s(a,b) = g^a b^{n^s}$ where $g=(1+n)$
 - $\text{Enc}(m) = \psi_s(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*
 - ψ_s can still be inverted if p,q known (but more involved)
 - Homomorphism: $\text{Enc}(m) \cdot \text{Enc}(m')$ is $\text{Enc}(m+m')$

in $\mathbb{Z}_{n^{(s+1)}}^*$

- $\psi_s(m,r) \cdot \psi_s(m',r') = \psi_s(m+m',r \cdot r')$

Damgård-Jurik Scheme

- Over $\mathbb{Z}_{n^{(s+1)}}^* \simeq \mathbb{Z}_{ns} \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within 2x of each other
 - Isomorphism: $\psi_s(a,b) = g^{ab^{n^s}}$ where $g=(1+n)$
 - $\text{Enc}(m) = \psi_s(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*
 - ψ_s can still be inverted if p,q known (but more involved)
 - Homomorphism: $\text{Enc}(m).\text{Enc}(m')$ is $\text{Enc}(m+m')$
 - $\psi_s(m,r).\psi_s(m',r') = \psi_s(m+m',r.r')$
 - in $\mathbb{Z}_{n^{(s+1)}}^*$
 - in \mathbb{Z}_{ns}

Damgård-Jurik Scheme

- Over $\mathbb{Z}_{n^{(s+1)}}^* \approx \mathbb{Z}_{n^s} \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within 2x of each other

- Isomorphism: $\psi_s(a,b) = g^{abn^s}$ where $g=(1+n)$

- Enc(m) = $\psi_s(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*

- ψ_s can still be inverted if p,q known (but more involved)

- Homomorphism: Enc(m).Enc(m') is Enc(m+m')

- in $\mathbb{Z}_{n^{(s+1)}}^*$ $\psi_s(m,r) \cdot \psi_s(m',r') = \psi_s(m+m',r \cdot r')$ in \mathbb{Z}_{n^s}

- Recursive encryption: Ciphertext for $\psi_s(\mathbb{Z}_{n^{(s+1)}}^*)$ is a plaintext for $\psi_{s+1}(\mathbb{Z}_{n^{(s+1)}})$ for the same public-key n . Note: $s \log n$ bits encrypted to $(s+1)\log n$ bits.

Damgård–Jurik Scheme

- Over $\mathbb{Z}_{n^{(s+1)}}^* \approx \mathbb{Z}_{n^s} \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within 2x of each other

- Isomorphism: $\psi_s(a,b) = g^{ab^{n^s}}$ where $g=(1+n)$

- Enc(m) = $\psi_s(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*

- ψ_s can still be inverted if p,q known (but more involved)

- Homomorphism: Enc(m).Enc(m') is Enc(m+m')

- in $\mathbb{Z}_{n^{(s+1)}}^*$ $\psi_s(m,r) \cdot \psi_s(m',r') = \psi_s(m+m',r \cdot r')$ in \mathbb{Z}_{n^s}

- Recursive encryption: Ciphertext for $\psi_s(\mathbb{Z}_{n^{(s+1)}}^*)$ is a plaintext for $\psi_{s+1}(\mathbb{Z}_{n^{(s+1)}})$ for the same public-key n . Note: $s \log n$ bits encrypted to $(s+1)\log n$ bits.

- Security: Rerandomization can be made perfect. IND-CPA secure under "Decisional Composite Residuosity" assumption

Damgård–Jurik Scheme

- Over $\mathbb{Z}_{n^{(s+1)}}^* \simeq \mathbb{Z}_{n^s} \times \mathbb{Z}_n^*$, $n=pq$, p,q primes within 2x of each other

- Isomorphism: $\psi_s(a,b) = g^{ab^{n^s}}$ where $g=(1+n)$

- Enc(m) = $\psi_s(m,r)$ for m in \mathbb{Z}_n and a random r in \mathbb{Z}_n^*

- ψ_s can still be inverted if p,q known (but more involved)

- Homomorphism: Enc(m).Enc(m') is Enc(m+m')

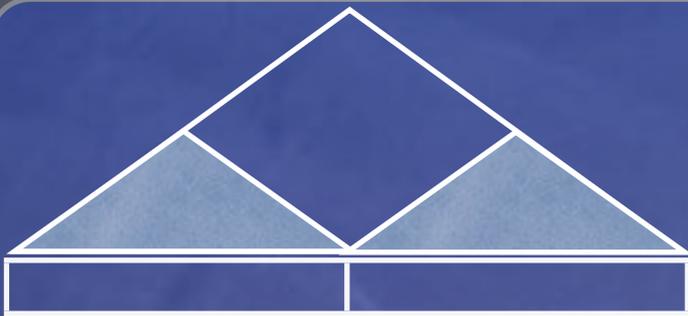
in $\mathbb{Z}_{n^{(s+1)}}^*$ $\psi_s(m,r) \cdot \psi_s(m',r') = \psi_s(m+m',r \cdot r')$ in \mathbb{Z}_{n^s}

- Recursive encryption: Ciphertext for $\psi_s(\mathbb{Z}_{n^{(s+1)}}^*)$ is a plaintext for $\psi_{s+1}(\mathbb{Z}_{n^{(s+1)}})$ for the same public-key n . Note: $s \log n$ bits encrypted to $(s+1)\log n$ bits.

- Security: Rerandomization can be made perfect. IND-CPA secure under "Decisional Composite Residuosity" assumption

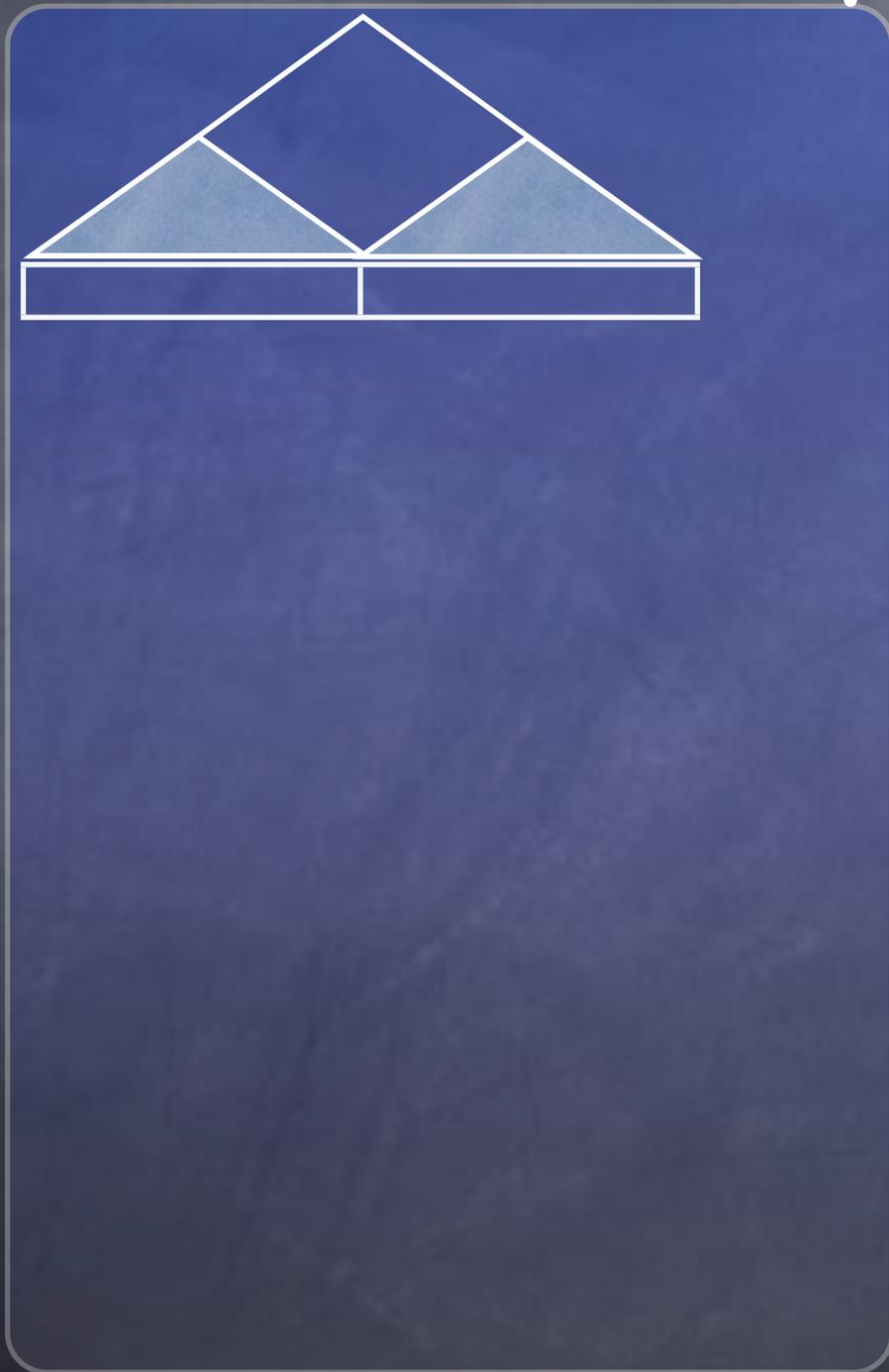
- Given $n=pq$ (but not p,q), $\psi_1(0,\text{rand})$ looks random (same as Paillier)

Final PIR protocol



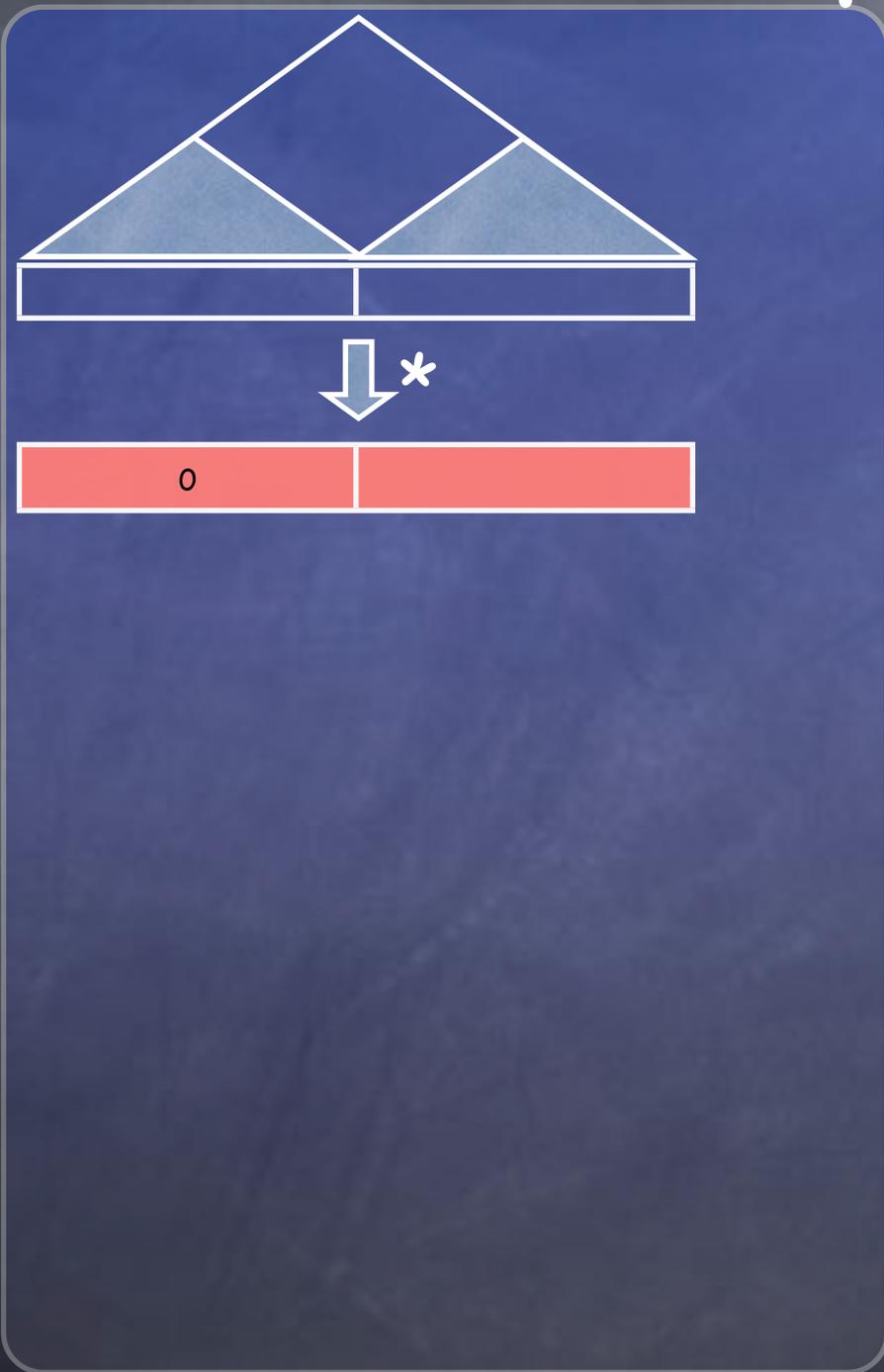
⋮

Final PIR protocol



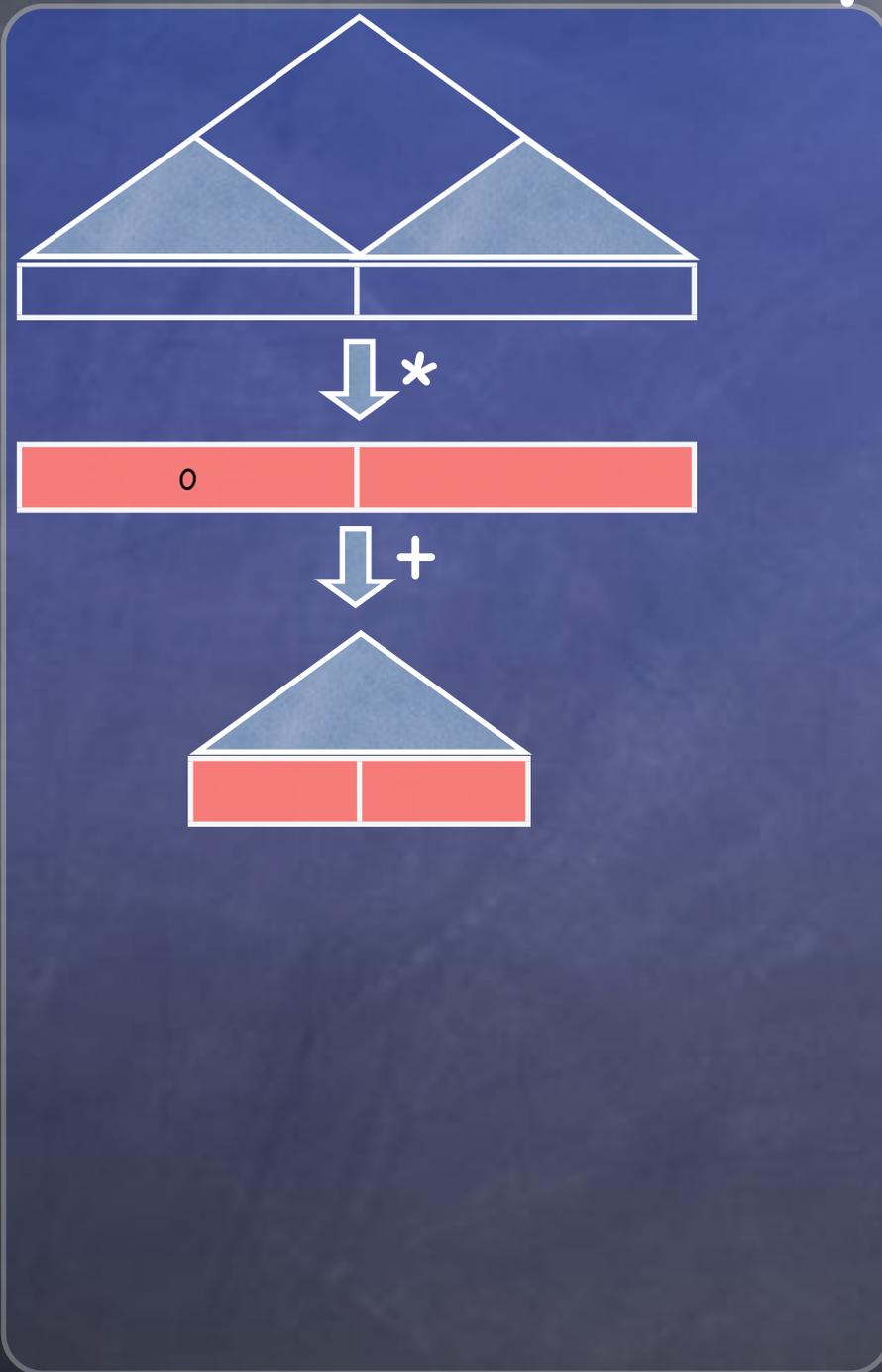
Final PIR protocol

0 1



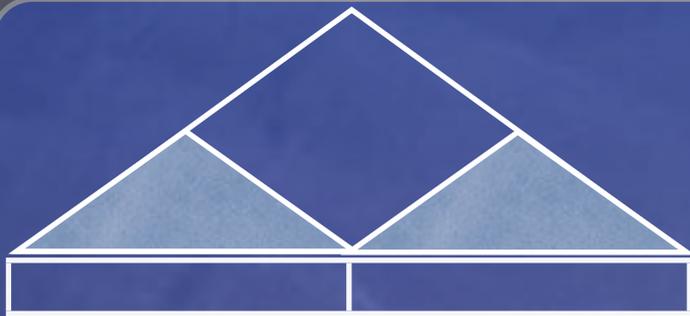
Final PIR protocol

0 1

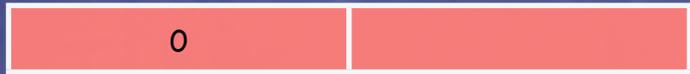


Final PIR protocol

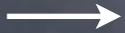
0 1



*



+



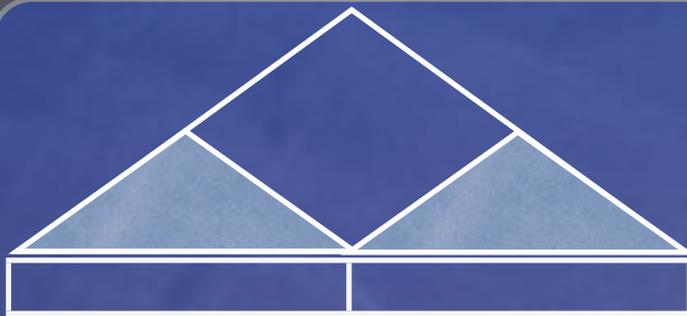
0 1



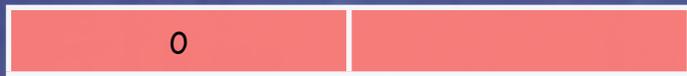
⋮

Final PIR protocol

0 1



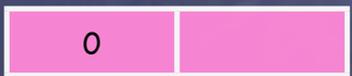
$*$



$+$



$+$

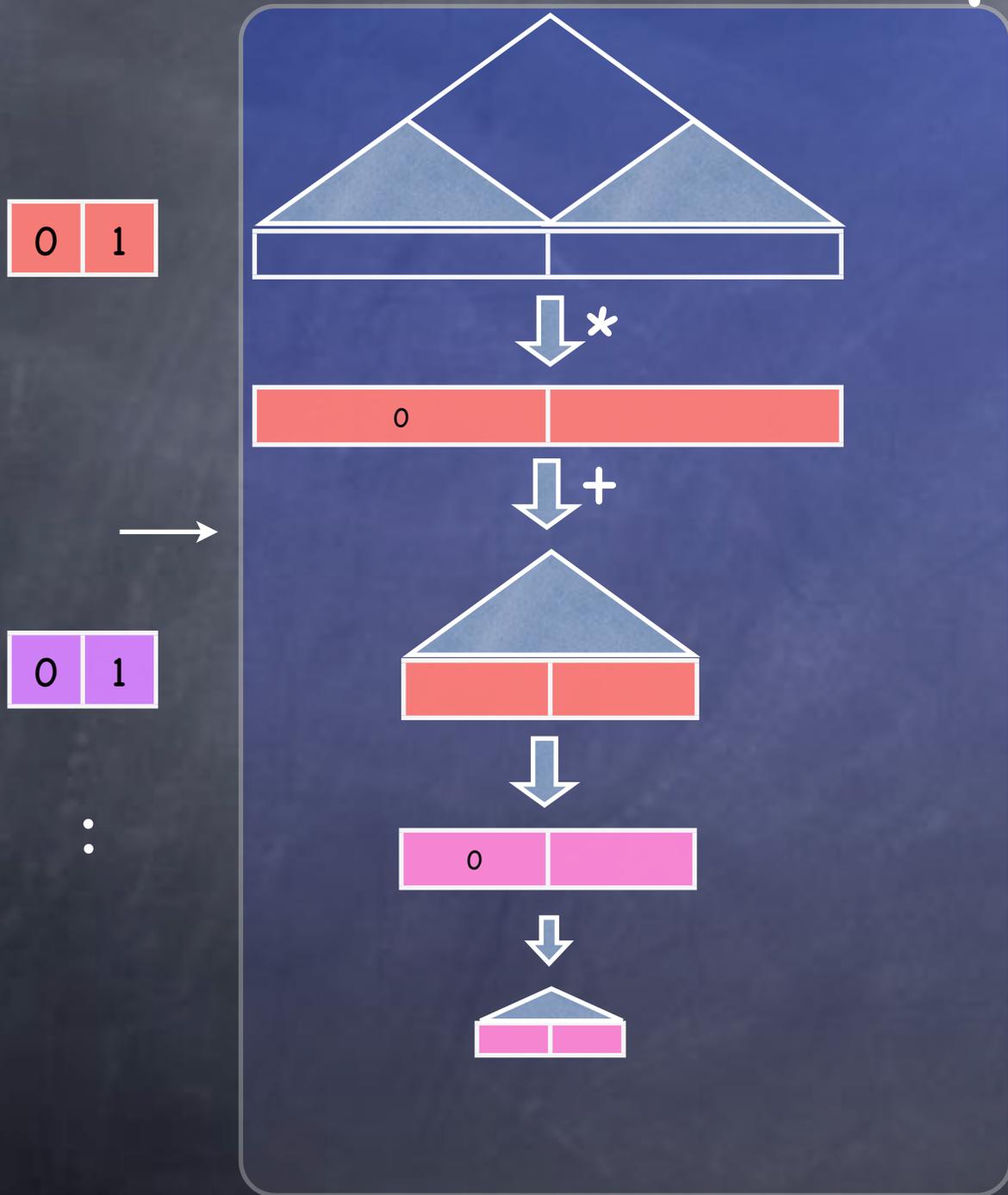


→

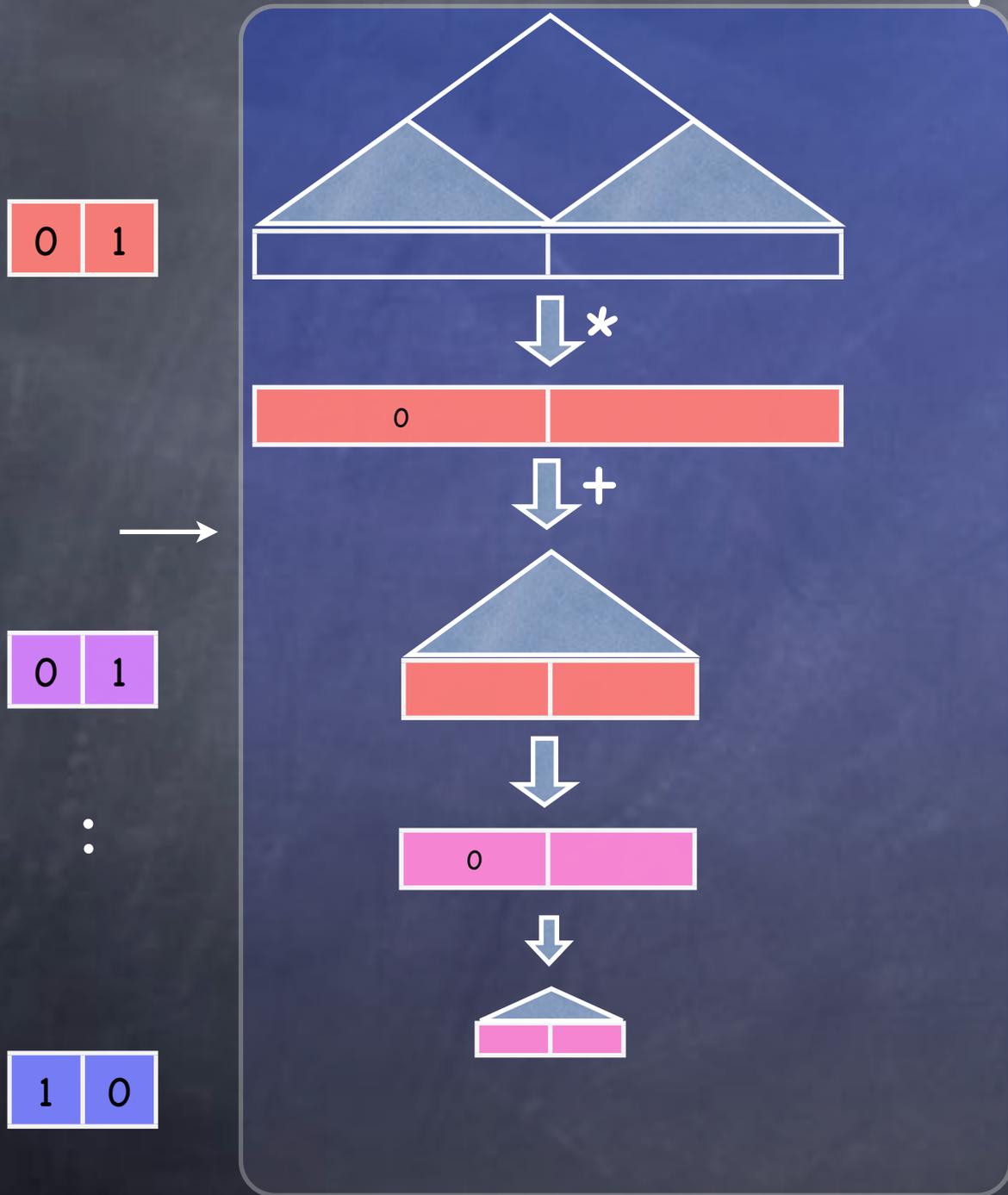
0 1

⋮

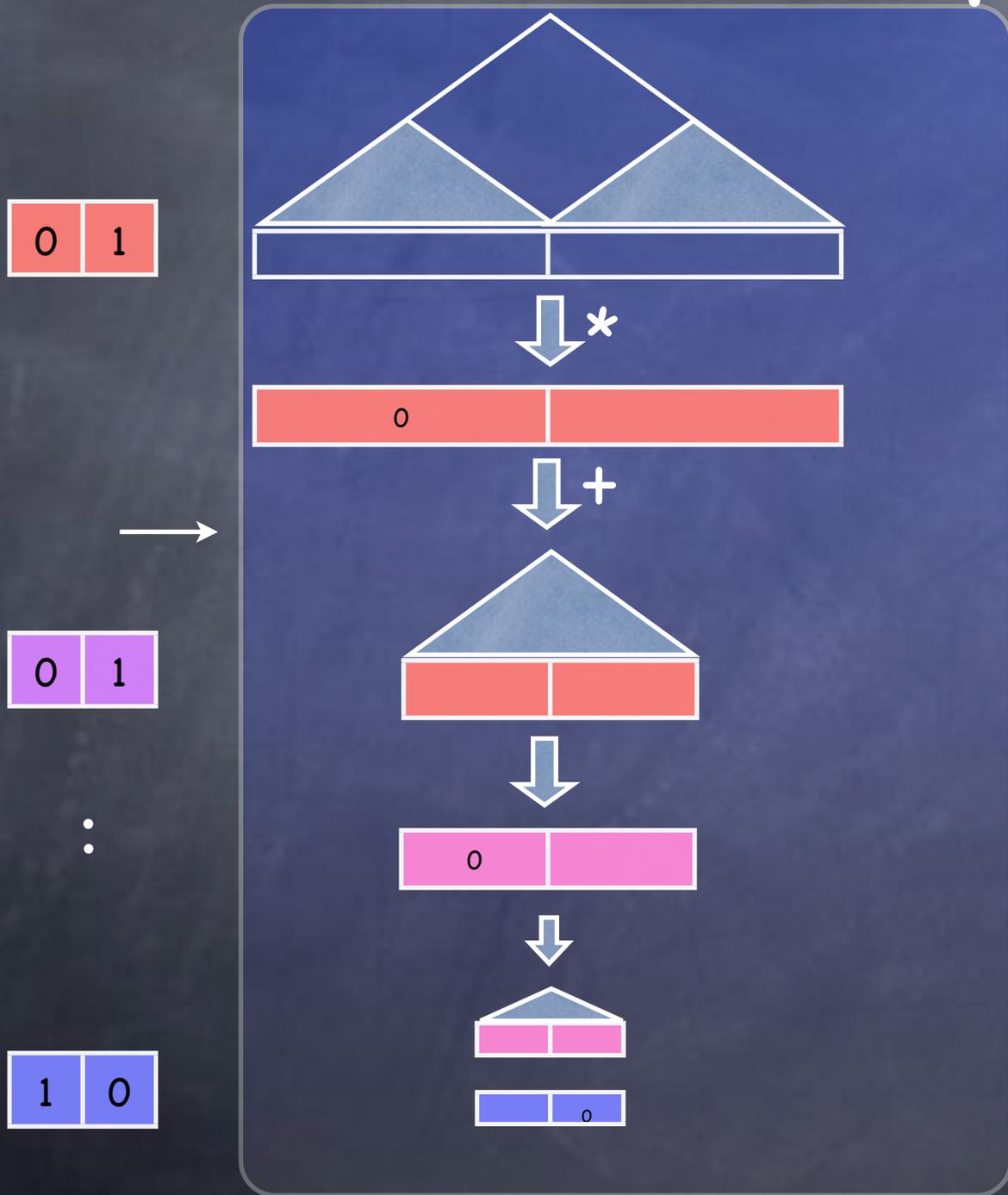
Final PIR protocol



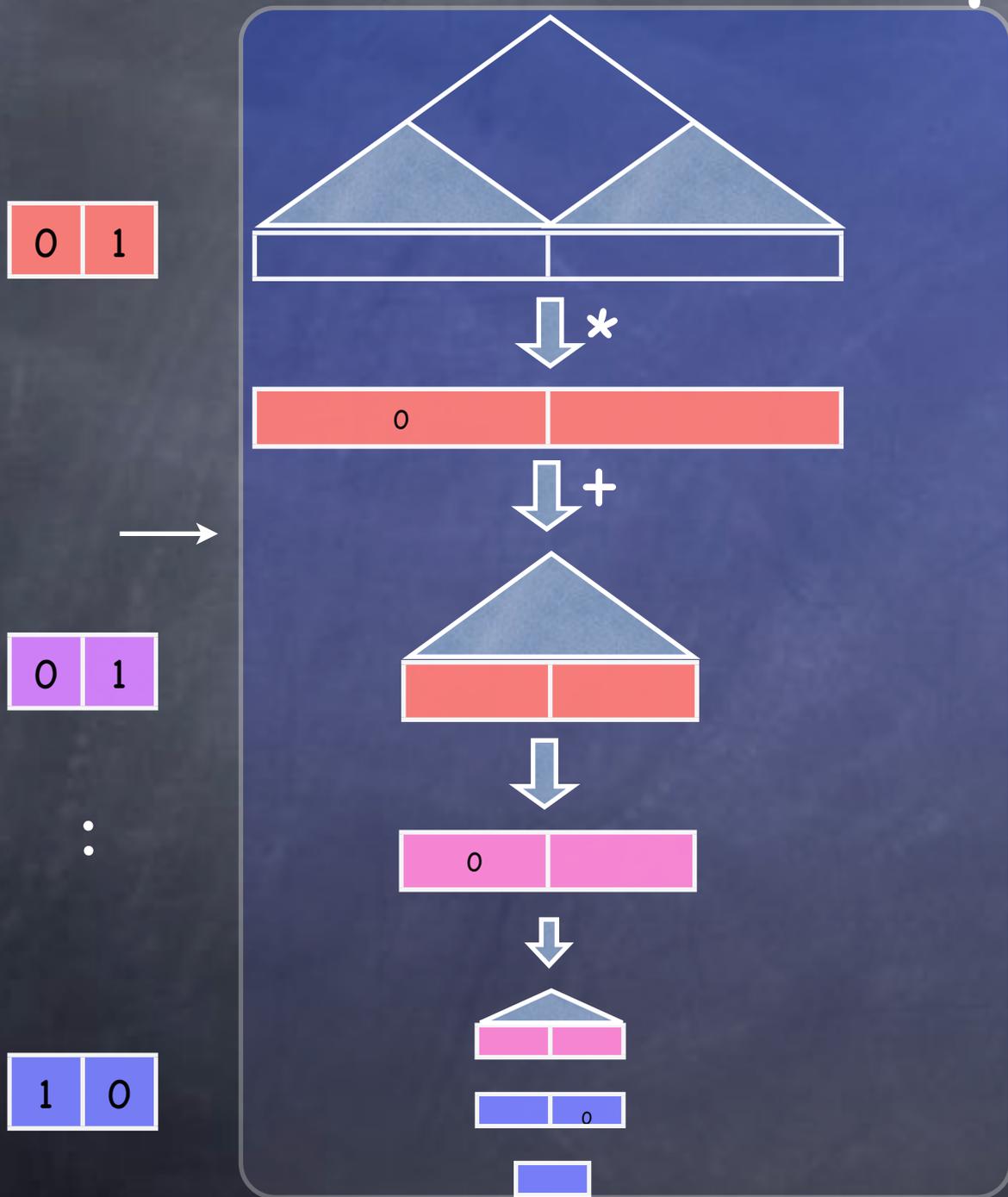
Final PIR protocol



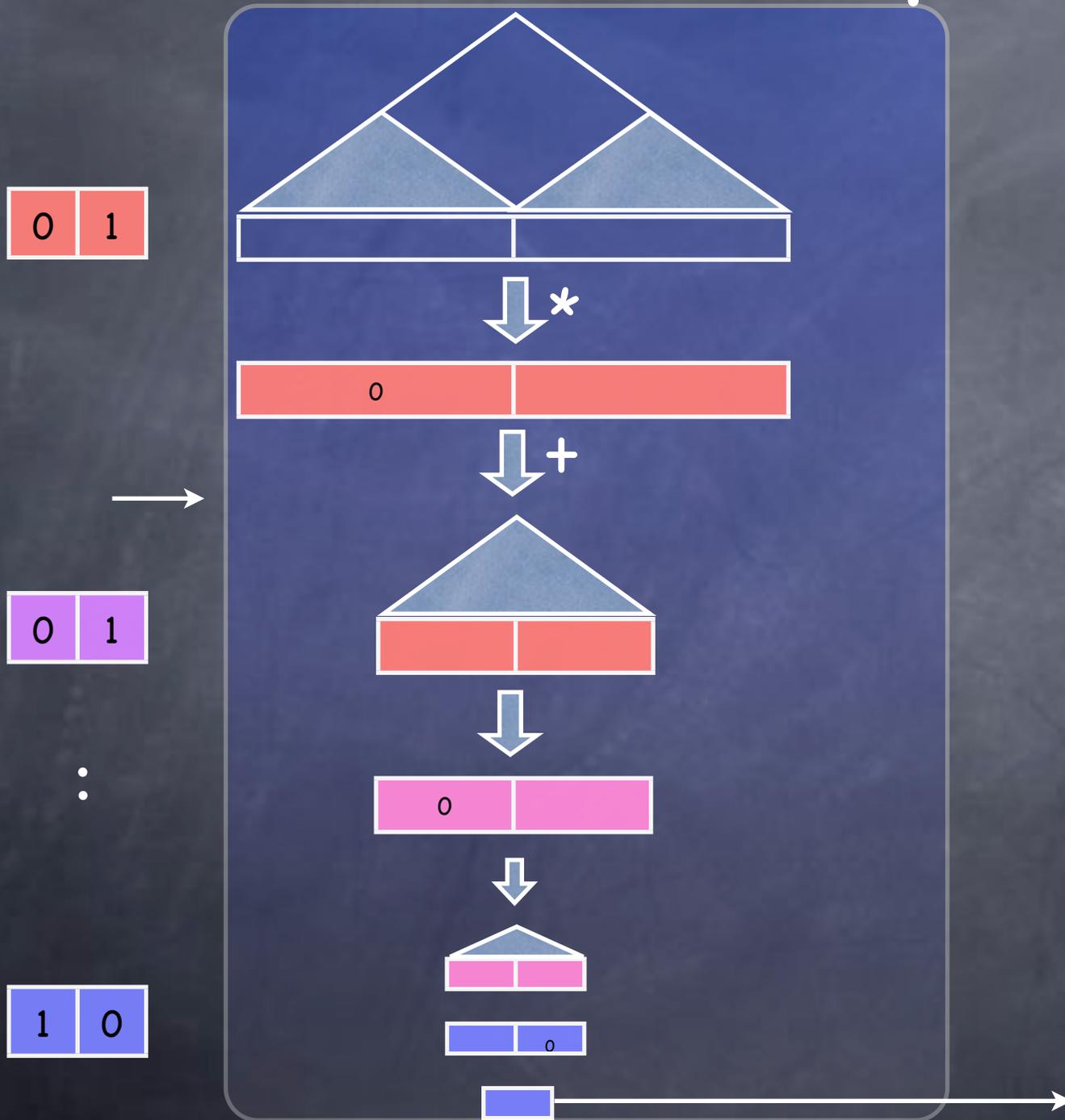
Final PIR protocol



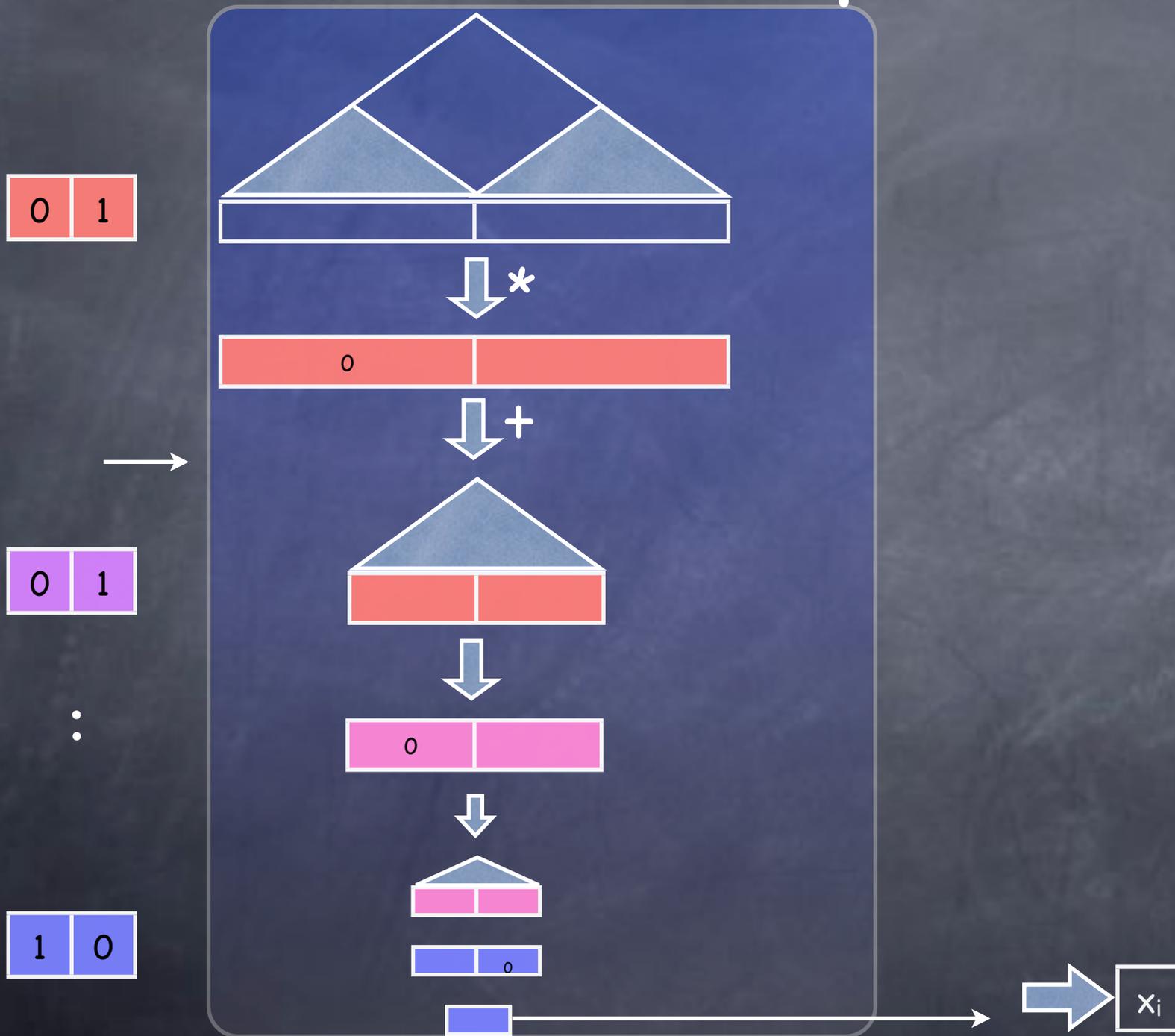
Final PIR protocol



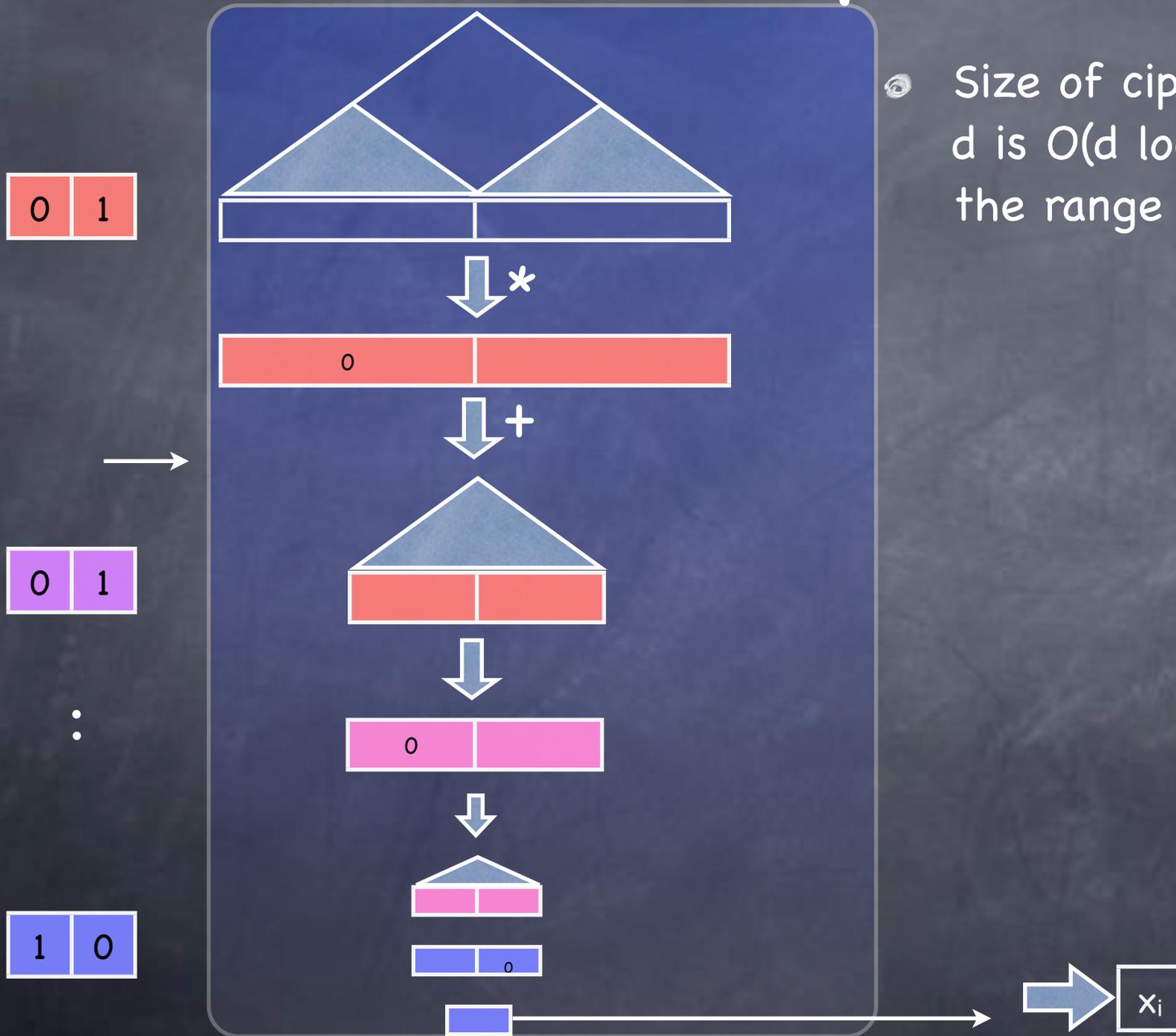
Final PIR protocol



Final PIR protocol

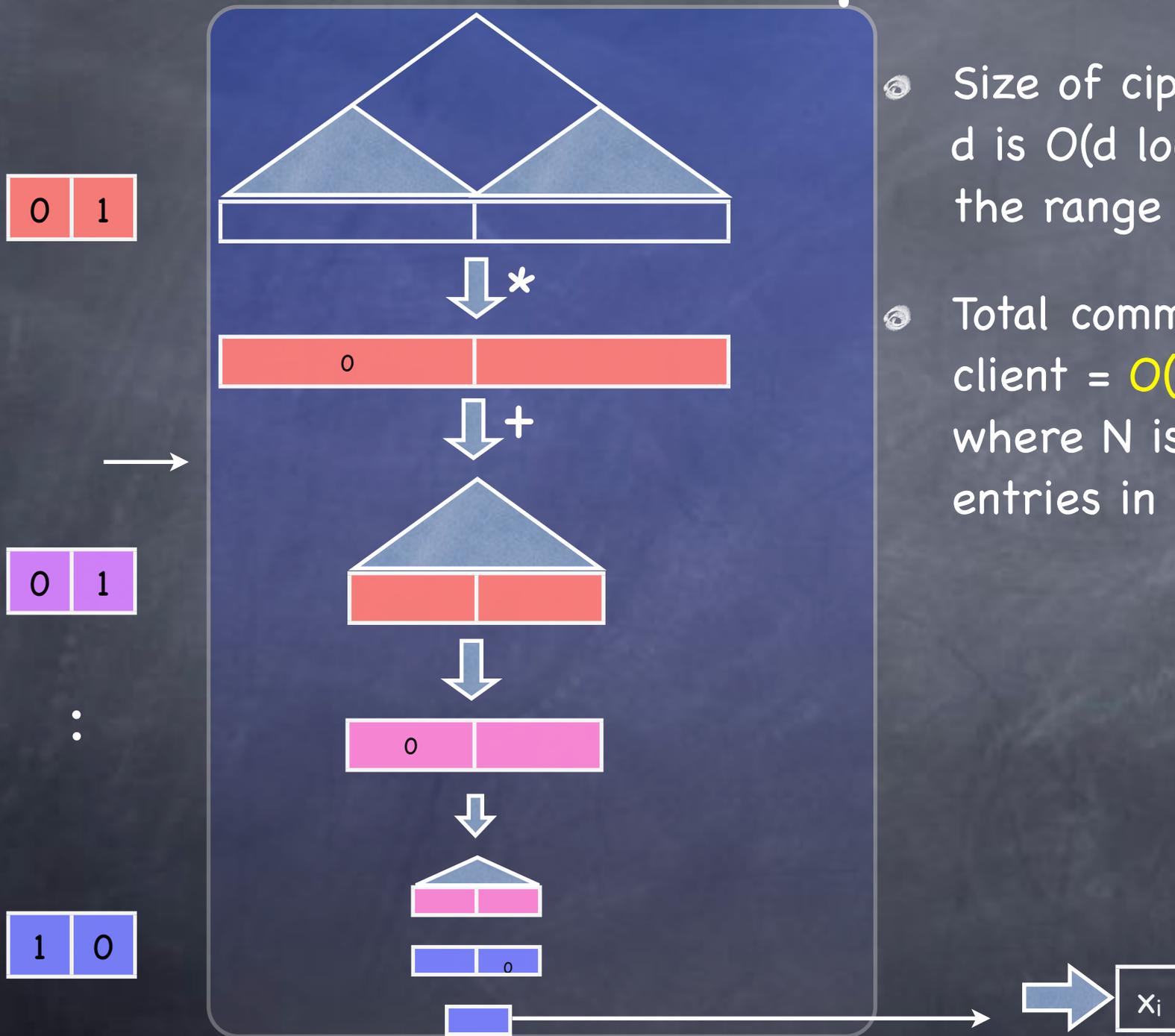


Final PIR protocol



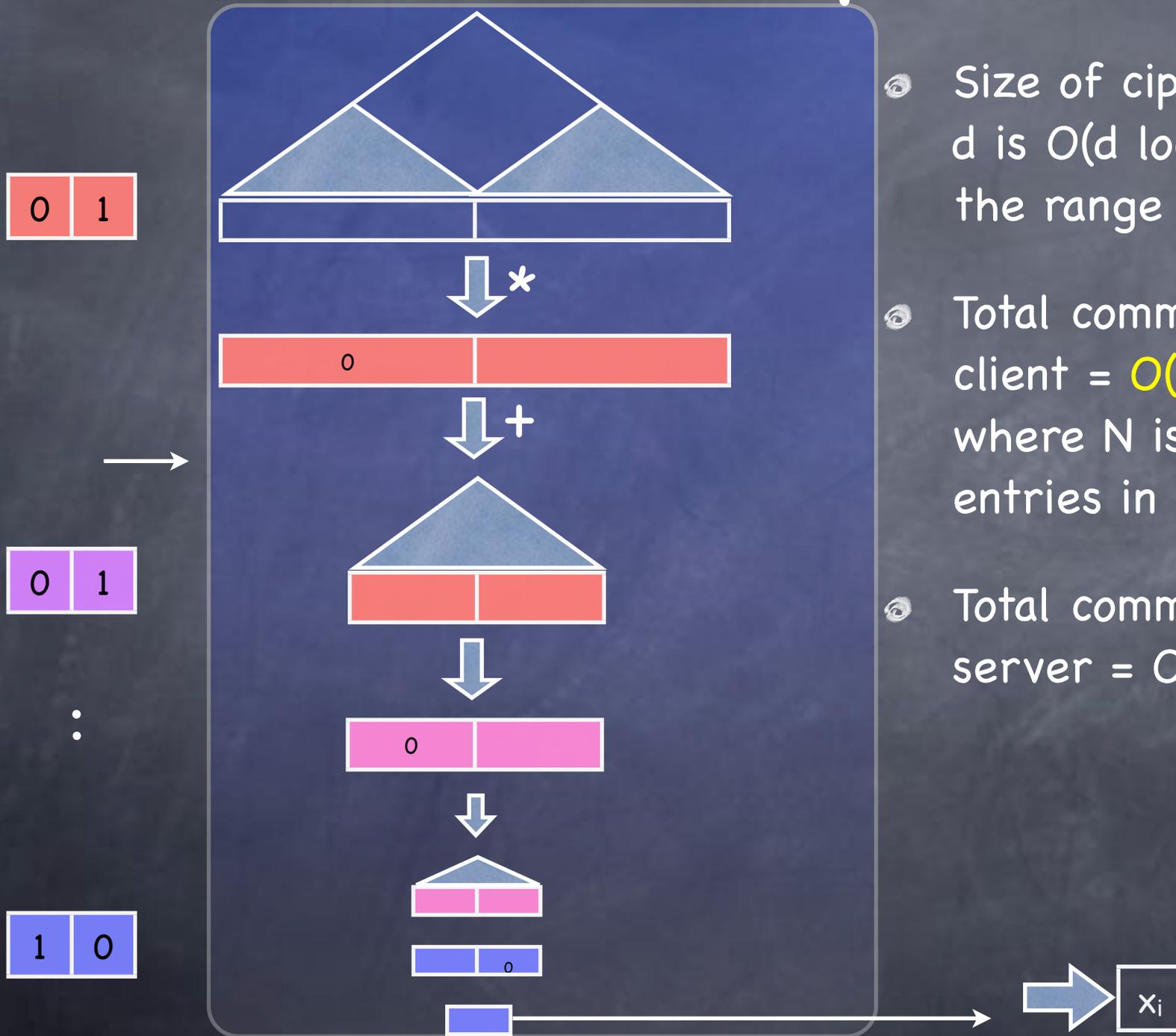
- Size of ciphertext at depth d is $O(d \log m)$ where m is the range of values in db

Final PIR protocol



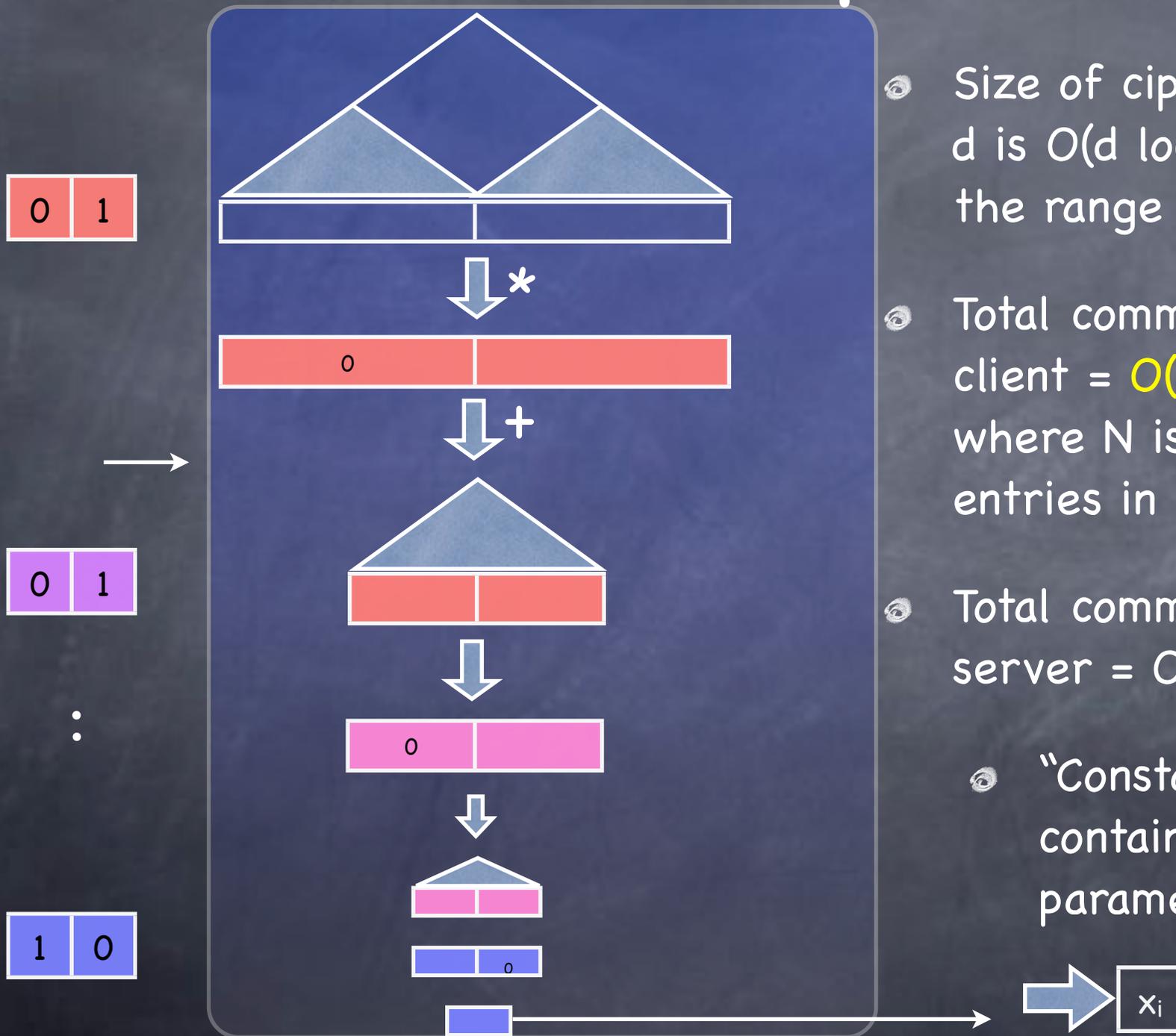
- Size of ciphertext at depth d is $O(d \log m)$ where m is the range of values in db
- Total communication from client = $O(\log^2 N \log m)$, where N is the number of entries in the db

Final PIR protocol



- Size of ciphertext at depth d is $O(d \log m)$ where m is the range of values in db
- Total communication from client = $O(\log^2 N \log m)$, where N is the number of entries in the db
- Total communication from server = $O(\log N \log m)$

Final PIR protocol



- Size of ciphertext at depth d is $O(d \log m)$ where m is the range of values in db
- Total communication from client = $O(\log^2 N \log m)$, where N is the number of entries in the db
- Total communication from server = $O(\log N \log m)$
- "Constant" in $O(\cdot)$ contains security parameter

Homomorphic Encryption for MPC

Homomorphic Encryption for MPC

- Recall GMW (passive-secure): each wire value was kept shared among the parties

Homomorphic Encryption for MPC

- Recall GMW (passive-secure): each wire value was kept shared among the parties
- Alternate approach: each wire value is kept encrypted, publicly

Homomorphic Encryption for MPC

- Recall GMW (passive-secure): each wire value was kept shared among the parties
- Alternate approach: each wire value is kept encrypted, publicly
 - Evaluate each wire: additive homomorphism (unlinkable)

Homomorphic Encryption for MPC

- Recall GMW (passive-secure): each wire value was kept shared among the parties
- Alternate approach: each wire value is kept encrypted, publicly
 - Evaluate each wire: additive homomorphism (unlinkable)
 - Notation: $[x] [+] [y] = [x+y]$, and $a^*[x] = [ax]$

Homomorphic Encryption for MPC

- Recall GMW (passive-secure): each wire value was kept shared among the parties
- Alternate approach: each wire value is kept encrypted, publicly
 - Evaluate each wire: additive homomorphism (unlinkable)
 - Notation: $[x] [+] [y] = [x+y]$, and $a^*[x] = [ax]$
 - And decrypt the output wire value: threshold decryption

Homomorphic Encryption for MPC

- Recall GMW (passive-secure): each wire value was kept shared among the parties
- Alternate approach: each wire value is kept encrypted, publicly
 - Evaluate each wire: additive homomorphism (unlinkable)
 - Notation: $[x] [+] [y] = [x+y]$, and $a^*[x] = [ax]$
 - And decrypt the output wire value: threshold decryption
 - Threshold decryption: KeyGen protocol so that PK is public and SK shared; Decryption protocol that lets the parties decrypt a ciphertext keeping their SK shares private

Homomorphic Encryption for MPC

- Recall GMW (passive-secure): each wire value was kept shared among the parties
- Alternate approach: each wire value is kept encrypted, publicly
 - Evaluate each wire: additive homomorphism (unlinkable)
 - Notation: $[x] [+] [y] = [x+y]$, and $a^*[x] = [ax]$
 - And decrypt the output wire value: threshold decryption
 - Threshold decryption: KeyGen protocol so that PK is public and SK shared; Decryption protocol that lets the parties decrypt a ciphertext keeping their SK shares private
 - (For active-security, also ZK proofs/proofs of knowledge)

Homomorphic Encryption for MPC

Homomorphic Encryption for MPC

- Run KeyGen and obtain PK and private shares for SK

Homomorphic Encryption for MPC

- Run KeyGen and obtain PK and private shares for SK
- Each party encrypts its input and publishes

Homomorphic Encryption for MPC

- Run KeyGen and obtain PK and private shares for SK
- Each party encrypts its input and publishes

For active-security, include ZK proofs of correctness/knowledge of plaintext, when publishing

Homomorphic Encryption for MPC

- Run KeyGen and obtain PK and private shares for SK
- Each party encrypts its input and publishes
- At an addition gate, carry out homomorphic addition: $[z]=[x][+][y]$

For active-security, include ZK proofs of correctness/knowledge of plaintext, when publishing

Homomorphic Encryption for MPC

- Run KeyGen and obtain PK and private shares for SK
- Each party encrypts its input and publishes
- At an addition gate, carry out homomorphic addition: $[z]=[x][+][y]$
- At a multiplication gate, given $[x]$ and $[y]$, to compute $[xy]$:

For active-security, include ZK proofs of correctness/knowledge of plaintext, when publishing

Homomorphic Encryption for MPC

- Run KeyGen and obtain PK and private shares for SK
- Each party encrypts its input and publishes
- At an addition gate, carry out homomorphic addition: $[z]=[x][+][y]$
- At a multiplication gate, given $[x]$ and $[y]$, to compute $[xy]$:
 - Share x : All parties except P_1 , choose their shares s_i ; to help P_1 compute s_1 , they publish $[-s_i]$, P_1 publishes $[r]$; they threshold decrypt $t=[r + x + \sum_{i=2:m} (-s_i)]$. P_1 sets $s_1 = t-r$

For active-security, include ZK proofs of correctness/knowledge of plaintext, when publishing

Homomorphic Encryption for MPC

- Run KeyGen and obtain PK and private shares for SK
- Each party encrypts its input and publishes
- At an addition gate, carry out homomorphic addition: $[z]=[x][+][y]$
- At a multiplication gate, given $[x]$ and $[y]$, to compute $[xy]$:
 - Share x : All parties except P_1 , choose their shares s_i ; to help P_1 compute s_1 , they publish $[-s_i]$, P_1 publishes $[r]$; they threshold decrypt $t=[r + x + \sum_{i=2:m} (-s_i)]$. P_1 sets $s_1 = t-r$
 - Each party publishes $s_i^*[y] = [s_i y]$; they compute $[\sum s_i y]=[xy]$

For active-security, include ZK proofs of correctness/knowledge of plaintext, when publishing

Homomorphic Encryption for MPC

- Run KeyGen and obtain PK and private shares for SK
- Each party encrypts its input and publishes
- At an addition gate, carry out homomorphic addition: $[z]=[x][+][y]$
- At a multiplication gate, given $[x]$ and $[y]$, to compute $[xy]$:
 - Share x : All parties except P_1 , choose their shares s_i ; to help P_1 compute s_1 , they publish $[-s_i]$, P_1 publishes $[r]$; they threshold decrypt $t=[r + x + \sum_{i=2:m} (-s_i)]$. P_1 sets $s_1 = t-r$
 - Each party publishes $s_i * [y] = [s_i y]$; they compute $[\sum s_i y]=[xy]$
- Threshold decrypt the output

For active-security, include ZK proofs of correctness/knowledge of plaintext, when publishing

The plaintext domain

The plaintext domain

- In some encryption schemes the plaintext domain is fixed as a system parameter

The plaintext domain

- In some encryption schemes the plaintext domain is fixed as a system parameter
 - e.g. El Gamal, when the DDH group is fixed

The plaintext domain

- In some encryption schemes the plaintext domain is fixed as a system parameter
 - e.g. El Gamal, when the DDH group is fixed
- But sometimes the plaintext domain is chosen as part of the public-key

The plaintext domain

- In some encryption schemes the plaintext domain is fixed as a system parameter
 - e.g. El Gamal, when the DDH group is fixed
- But sometimes the plaintext domain is chosen as part of the public-key
 - e.g. Paillier, when the modulus $n = pq$ is chosen

The plaintext domain

- In some encryption schemes the plaintext domain is fixed as a system parameter
 - e.g. El Gamal, when the DDH group is fixed
- But sometimes the plaintext domain is chosen as part of the public-key
 - e.g. Paillier, when the modulus $n = pq$ is chosen
- For non-homomorphic encryption, not critical: can use a scheme with a larger domain into which the required domain can be embedded

The plaintext domain

- In some encryption schemes the plaintext domain is fixed as a system parameter
 - e.g. El Gamal, when the DDH group is fixed
- But sometimes the plaintext domain is chosen as part of the public-key
 - e.g. Paillier, when the modulus $n = pq$ is chosen
- For non-homomorphic encryption, not critical: can use a scheme with a larger domain into which the required domain can be embedded
- But not good for homomorphic encryption: say, an application needs to use addition modulo 10; can we use Paillier?

The plaintext domain

The plaintext domain

- Say, an application needs to use addition modulo 10; can we use Paillier?

The plaintext domain

- Say, an application needs to use addition modulo 10; can we use Paillier?
 - Suppose there is a bound on how many times the homomorphic operation will be carried out

The plaintext domain

- Say, an application needs to use addition modulo 10; can we use Paillier?
 - Suppose there is a bound on how many times the homomorphic operation will be carried out
 - Then, work with a suitably large modulus, so that no overflow occurs

The plaintext domain

- Say, an application needs to use addition modulo 10; can we use Paillier?
 - Suppose there is a bound on how many times the homomorphic operation will be carried out
 - Then, work with a suitably large modulus, so that no overflow occurs
 - Not unlinkable: $9+3$ and 2 look different

The plaintext domain

- Say, an application needs to use addition modulo 10; can we use Paillier?
 - Suppose there is a bound on how many times the homomorphic operation will be carried out
 - Then, work with a suitably large modulus, so that no overflow occurs
 - Not unlinkable: $9+3$ and 2 look different
 - Suppose also OK to reveal how many operations were done

The plaintext domain

- Say, an application needs to use addition modulo 10; can we use Paillier?
 - Suppose there is a bound on how many times the homomorphic operation will be carried out
 - Then, work with a suitably large modulus, so that no overflow occurs
 - Not unlinkable: $9+3$ and 2 look different
 - Suppose also OK to reveal how many operations were done
 - Each time add a large random multiple of 10 (but not large enough to cause overflow): $9+3+10r$ and $2+10r$ are statistically close if r drawn from a large range

Today

Today

- Homomorphic Encryption: El Gamal, Paillier, Damgård-Jurik

Today

- Homomorphic Encryption: El Gamal, Paillier, Damgård-Jurik
- Applications of Homomorphic Encryption

Today

- Homomorphic Encryption: El Gamal, Paillier, Damgård-Jurik
- Applications of Homomorphic Encryption
 - A simple (passive-secure) OT protocol using rerandomizable encryption

Today

- Homomorphic Encryption: El Gamal, Paillier, Damgård-Jurik
- Applications of Homomorphic Encryption
 - A simple (passive-secure) OT protocol using rerandomizable encryption
 - PIR (using Damgård-Jurik encryption scheme)

Today

- Homomorphic Encryption: El Gamal, Paillier, Damgård-Jurik
- Applications of Homomorphic Encryption
 - A simple (passive-secure) OT protocol using rerandomizable encryption
 - PIR (using Damgård-Jurik encryption scheme)
 - MPC

Today

- Homomorphic Encryption: El Gamal, Paillier, Damgård-Jurik
- Applications of Homomorphic Encryption
 - A simple (passive-secure) OT protocol using rerandomizable encryption
 - PIR (using Damgård-Jurik encryption scheme)
 - MPC
- Not covered: “Fully Homomorphic Encryption”, security against active corruption (ZK proofs, non-malleable homomorphic encryption)

Today

- Homomorphic Encryption: El Gamal, Paillier, Damgård-Jurik
- Applications of Homomorphic Encryption
 - A simple (passive-secure) OT protocol using rerandomizable encryption
 - PIR (using Damgård-Jurik encryption scheme)
 - MPC
- Not covered: “Fully Homomorphic Encryption”, security against active corruption (ZK proofs, non-malleable homomorphic encryption)
- Coming up: more applications – in voting