Homomorphic Encryption

Lecture 18

And some applications

Homomorphic Encryption

Homomorphic Encryption

- Group Homomorphism: Two groups G and G' are homomorphic if there exists a function (homomorphism) $f:G \to G'$ such that

  for all $x,y \in G$, $f(x) +_{G'} f(y) = f(x +_G y)$

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 $x,y \in G$, $f(x) +_{G'} f(y) = f(x +_G y)$

- Homomorphic Encryption: A CPA secure (public-key) encryption s.t. $\mathrm{Dec}(C) +_M \mathrm{Dec}(D) = \mathrm{Dec}(C +_C D)$ for ciphertexts C, D

Homomorphic Encryption

- Group Homomorphism: Two groups G and G′ are homomorphic if there exists a function (homomorphism) $f:G \to G'$ such that

  for all $x,y \in G$, $f(x) +_{G'} f(y) = f(x +_{G} y)$

- Homomorphic Encryption: A CPA secure (public-key) encryption s.t. $\mathrm{Dec}(C) +_M \mathrm{Dec}(D) = \mathrm{Dec}(C +_C D)$ for ciphertexts C, D
  - i.e. $\mathrm{Enc}(x) +_C \mathrm{Enc}(y)$ is like $\mathrm{Enc}(x +_M y)$

Homomorphic Encryption

- Group Homomorphism: Two groups G and G′ are homomorphic if there exists a function (homomorphism) $f: G \to G'$ such that

  for all $x, y \in G$, $f(x) +_{G'} f(y) = f(x +_G y)$

- Homomorphic Encryption: A CPA secure (public-key) encryption s.t. $\mathrm{Dec}(C) +_M \mathrm{Dec}(D) = \mathrm{Dec}(C +_C D)$ for ciphertexts C, D
  - i.e. $\mathrm{Enc}(x) +_C \mathrm{Enc}(y)$ is like $\mathrm{Enc}(x +_M y)$
  - Interesting when $+_C$ doesn't require the decryption key

Homomorphic Encryption

- Group Homomorphism: Two groups G and G′ are homomorphic if there exists a function (homomorphism) $f:G \to G'$ such that

  for all $x,y \in G$, $f(x) +_{G'} f(y) = f(x +_G y)$

- Homomorphic Encryption: A CPA secure (public-key) encryption s.t. $\mathrm{Dec}(C) +_M \mathrm{Dec}(D) = \mathrm{Dec}(C +_C D)$ for ciphertexts C, D
  - i.e. $\mathrm{Enc}(x) +_C \mathrm{Enc}(y)$ is like $\mathrm{Enc}(x +_M y)$
  - Interesting when $+_C$ doesn't require the decryption key
- e.g. El Gamal: $(g^{x1}, m_1Y^{x1}) * (g^{x2}, m_2Y^{x2}) = (g^{x3}, m_1m_2Y^{x3})$

Homomorphic Encryption

- Group Homomorphism: Two groups G and G′ are homomorphic if there exists a function (homomorphism) $f:G\to G'$ such that

  for all $x,y \in G$, $f(x) +_{G'} f(y) = f(x +_G y)$

- Homomorphic Encryption: A CPA secure (public-key) encryption s.t. $\mathrm{Dec}(C) +_M \mathrm{Dec}(D) = \mathrm{Dec}(C +_C D)$ for ciphertexts C, D
  - i.e. $\mathrm{Enc}(x) +_C \mathrm{Enc}(y)$ is like $\mathrm{Enc}(x +_M y)$
  - Interesting when $+_C$ doesn't require the decryption key
- e.g. El Gamal: $(g^{x1},m_1Y^{x1}) * (g^{x2},m_2Y^{x2}) = (g^{x3},m_1m_2Y^{x3})$
- Not covered today: Fully Homomorphic Encryption, which supports ring homomorphism (addition <u>and</u> multiplication of messages)

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=Enc(x) and c_y=Enc(y), Add(c_x,c_y) should be identically distributed as 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=Enc(x) and c_y=Enc(y), Add(c_x,c_y) should be identically distributed as Enc(x $+_M$ y). Add is a randomized operation
  - Alternately, a ReRand operation s.t. for all valid ciphertexts c_x, ReRand(c_x) is identically distributed as Enc(x)

Rerandomization

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

Rerandomization

- Often (but not always) another property is required of a homomorphic encryption scheme
- Unlinkability
  - For any two ciphertexts c_x=Enc(x) and c_y=Enc(y), Add(c_x,c_y) should be <u>identically distributed</u> as Enc(x $+_M$ y). Add is a randomized operation
  - Alternately, a ReRand operation s.t. for all valid ciphertexts c_x, ReRand(c_x) is identically distributed as Enc(x)
    - Then, we can let Add(c_x,c_y) = ReRand(c_x $+_c$ c_y) where $+_c$ may be deterministic
  - Rerandomization useful even without homomorphism

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, receiver gets only the message m_1+m_2 in IDEAL; is not told if it is a fresh message or derived from other messages

An OT Protocol
(for passive corruption)

An OT Protocol (for passive corruption)

- Using an (unlinkable) rerandomizable encryption scheme

An OT Protocol
(for passive corruption)

- Using an (unlinkable) rerandomizable encryption scheme

An OT Protocol
(for passive corruption)

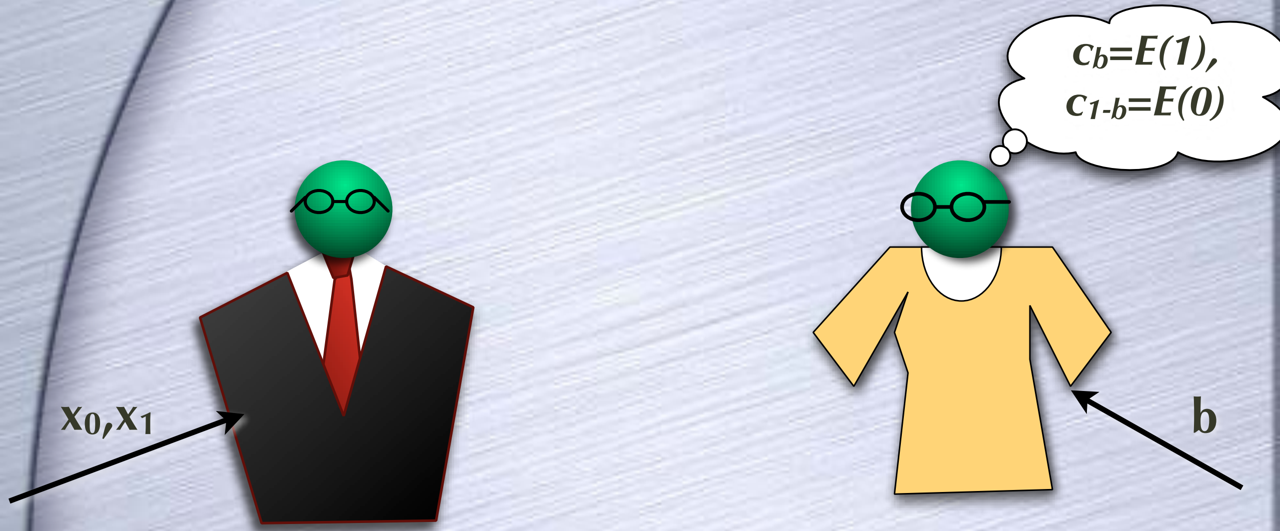
- Using an (unlinkable) rerandomizable encryption scheme

An OT Protocol
(for passive corruption)

- Using an (unlinkable) rerandomizable encryption scheme

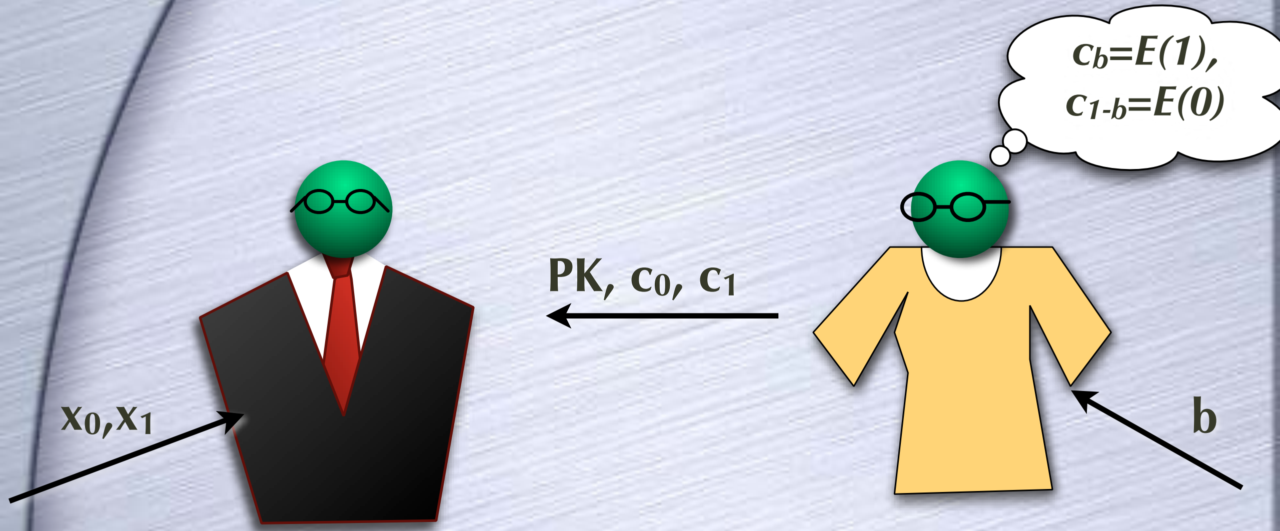
An OT Protocol
(for passive corruption)

- Using an (unlinkable) rerandomizable encryption scheme



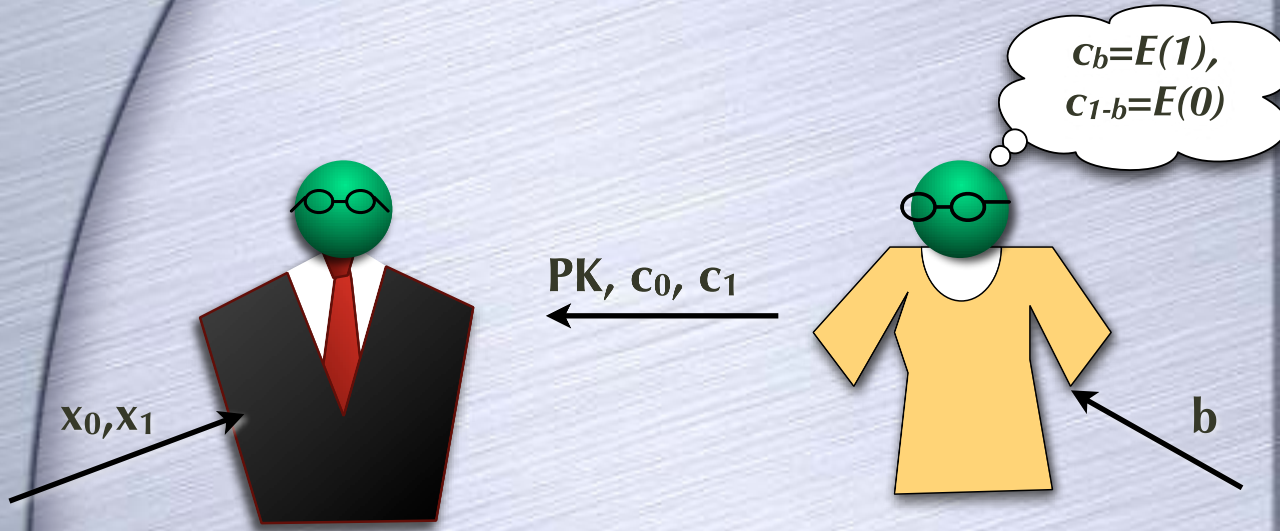
An OT Protocol
(for passive corruption)

- Using an (unlinkable) rerandomizable encryption scheme



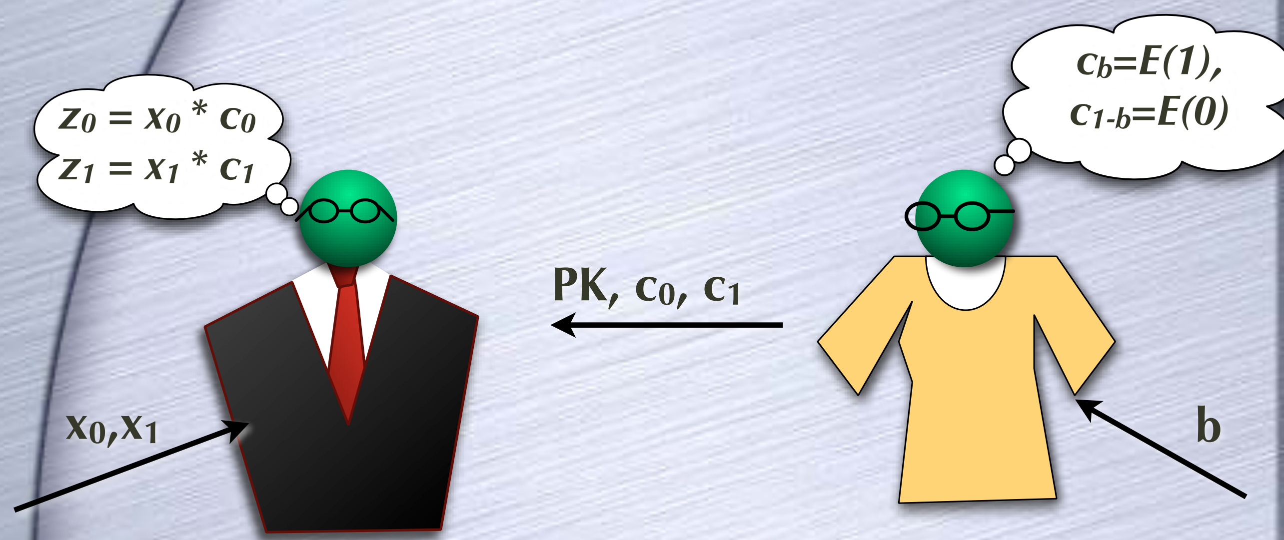
An OT Protocol
(for passive corruption)

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



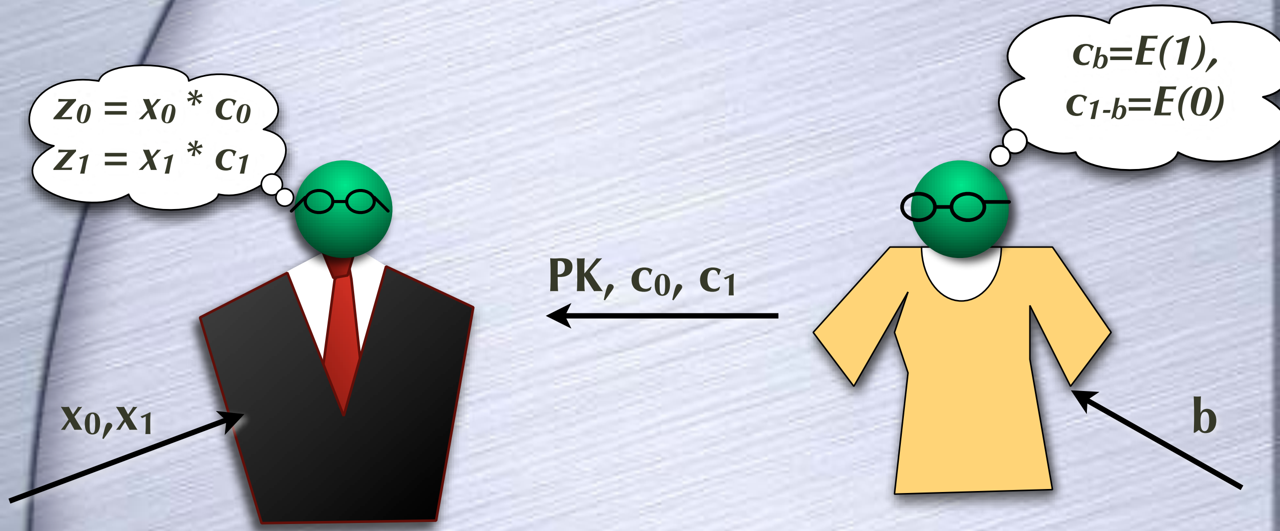
An OT Protocol
(for passive corruption)

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



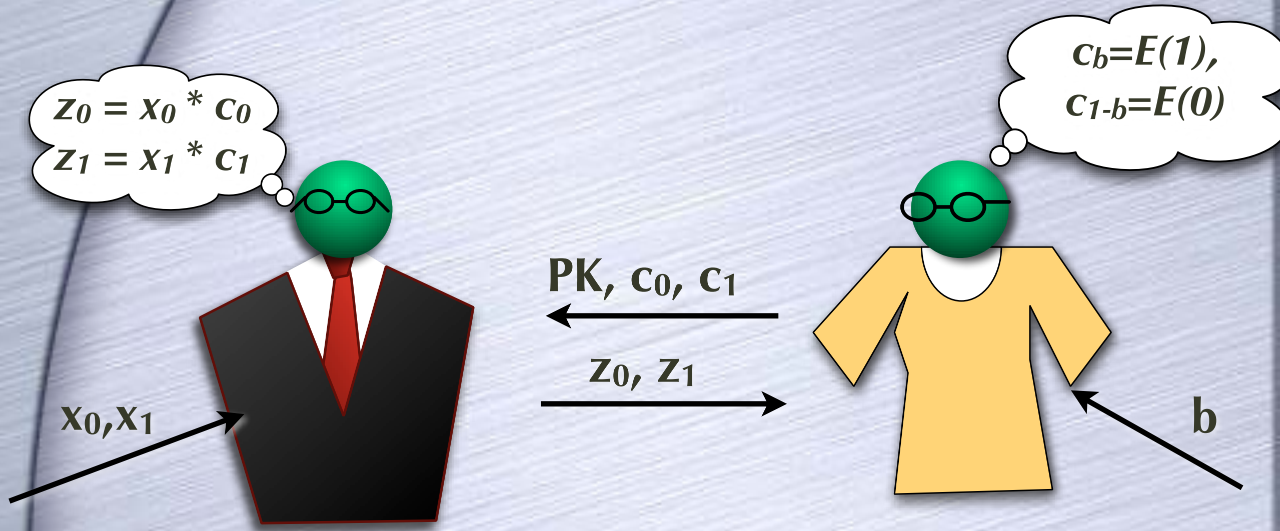
An OT Protocol
(for passive corruption)

- 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:=ReRand(c), 0*c:=E(0)



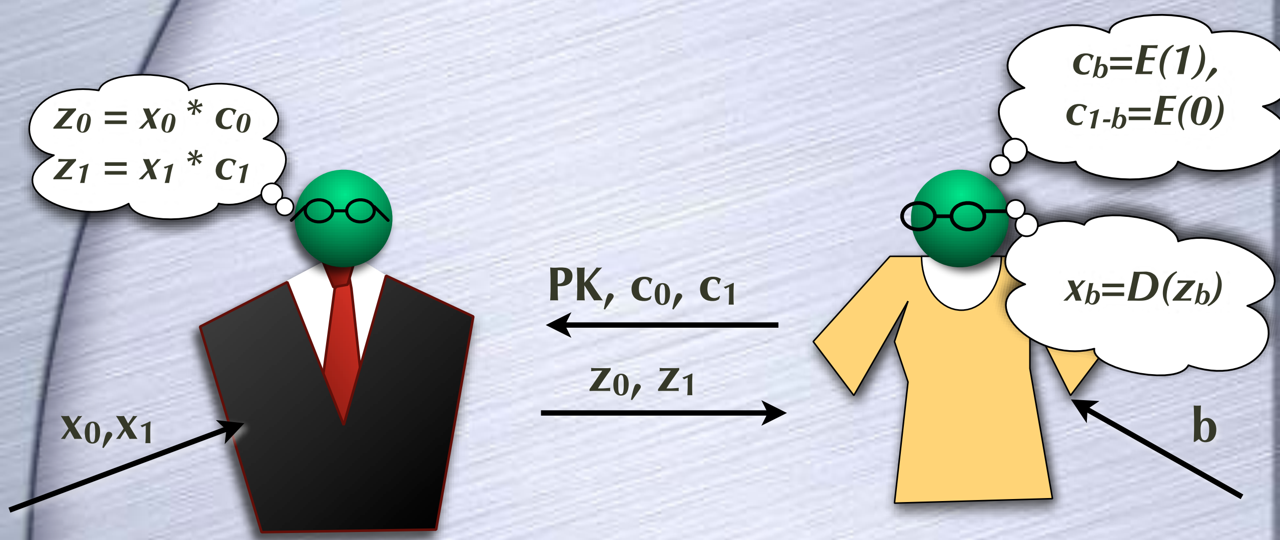
An OT Protocol
(for passive corruption)

- 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:=ReRand(c), 0*c:=E(0)



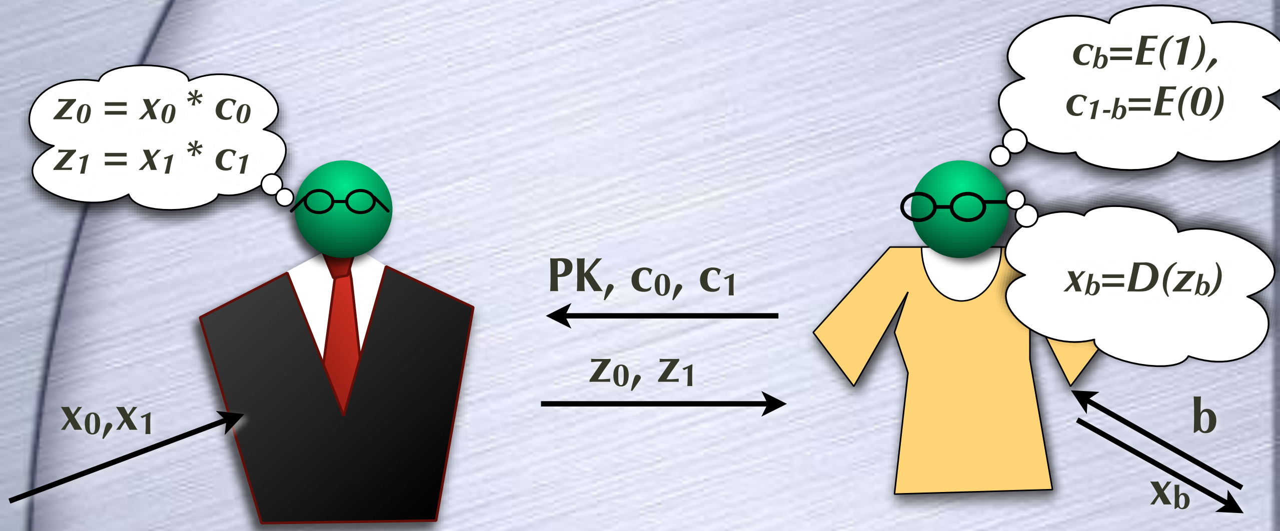
An OT Protocol
(for passive corruption)

- 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:=ReRand(c), 0*c:=E(0)



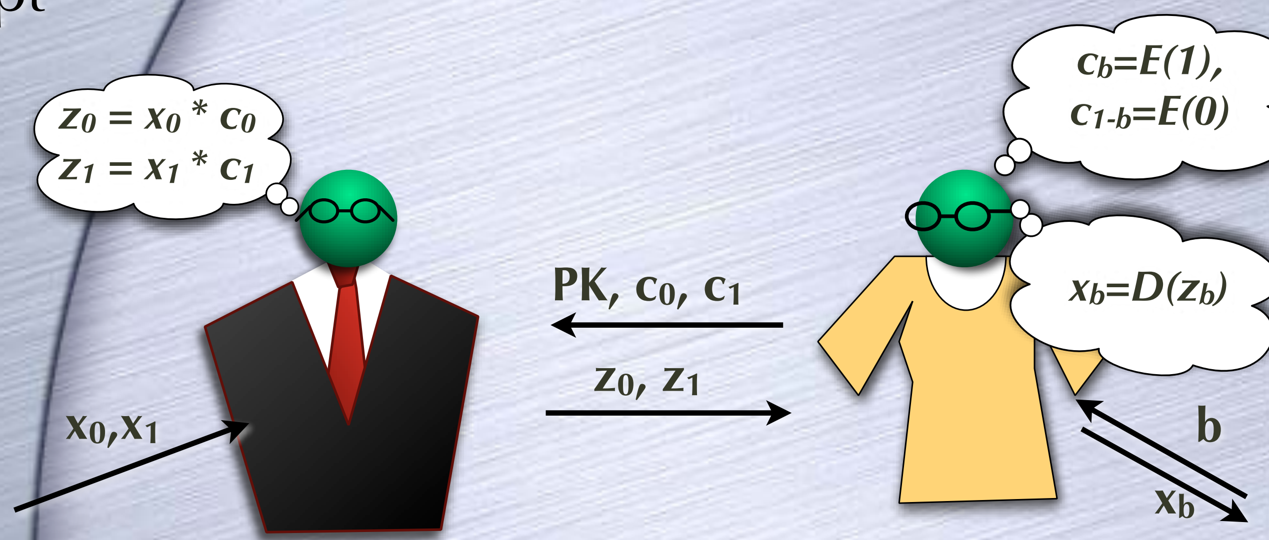
An OT Protocol
(for passive corruption)

- 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:=ReRand(c), 0*c:=E(0)



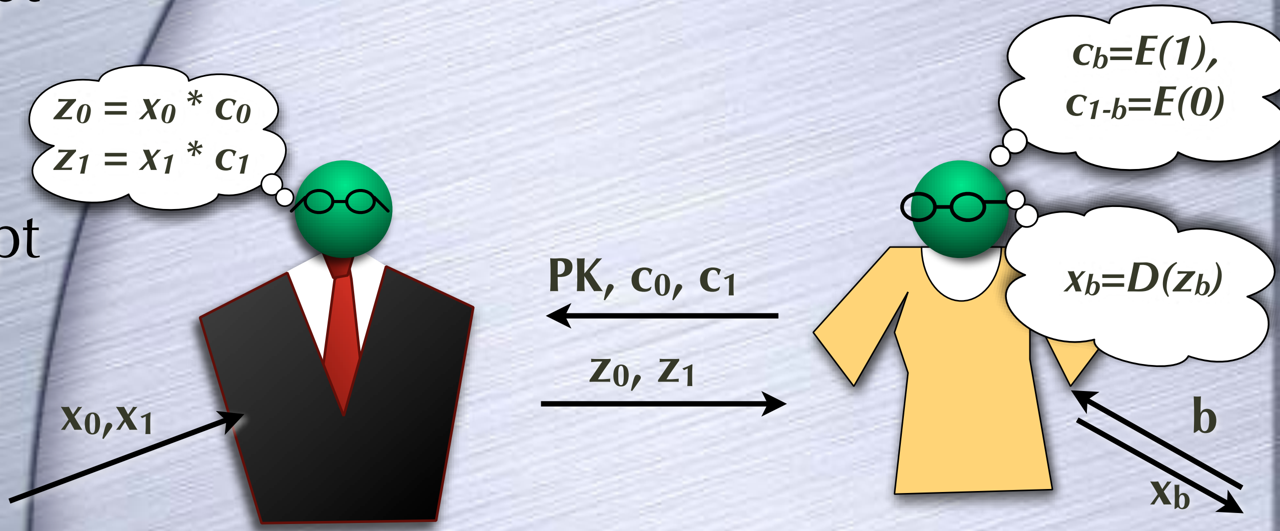
An OT Protocol
(for passive corruption)

- 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: $1*c := \mathrm{ReRand}(c)$, $0*c := E(0)$
- Simulation for passive-corrupt receiver: set $z_b = E(x_b)$ and $z_{1-b} = E(0)$



An OT Protocol (for passive 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:=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 passive-corrupt sender: Extract x_0,x_1 from input; set c_0,c_1 to be say 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: Homomorphic encryption over the message space

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: Homomorphic encryption over the message space
  - When message space is $\mathbb{Z}_n$: additively homomorphic encryption

Paillier's Scheme

Paillier's Scheme

- Uses $\mathbb{Z}_{n^2}^* \simeq \mathbb{Z}_n \times \mathbb{Z}_n^*$, n=pq, p,q primes

Paillier's Scheme

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

  To ensure $\gcd(n, \varphi(n))=1$

Paillier's Scheme

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

  To ensure $\gcd(n, \varphi(n))=1$

  - Isomorphism: $\psi(a,b) = g^a b^n \pmod{n^2}$ where $g=(1+n)$

Paillier's Scheme

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

  To ensure $\gcd(n,\varphi(n))=1$

  - Isomorphism: $\psi(a,b) = g^a b^n \pmod{n^2}$ where $g=(1+n)$
- $Enc(m) = \psi(m,r)$ for m in $\mathbb{Z}_n$ and a random r in $\mathbb{Z}_n^*$

Paillier's Scheme

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

  To ensure $\gcd(n, \varphi(n))=1$

  - Isomorphism: $\psi(a,b) = g^a b^n \pmod{n^2}$ where $g=(1+n)$
- $\mathrm{Enc}(m) = \psi(m,r)$ for m in $\mathbb{Z}_n$ and a random r in $\mathbb{Z}_n^*$
- ψ can be efficiently inverted if p,q known

Paillier's Scheme

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

  To ensure gcd(n, φ(n))=1

  - Isomorphism: $\psi(a,b) = g^a b^n \pmod{n^2}$ where g=(1+n)
- Enc(m) = ψ(m,r) for m in $\mathbb{Z}_n$ and a random r in $\mathbb{Z}_n{}^*$
- ψ can be efficiently inverted if p,q known
- (Additive) Homomorphism: Enc(m).Enc(m') is Enc(m+m')

Paillier's Scheme

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

  To ensure $\gcd(n, \varphi(n))=1$

  - Isomorphism: $\psi(a,b) = g^a b^n \pmod{n^2}$ where $g=(1+n)$
- Enc(m) = $\psi(m,r)$ for m in $\mathbb{Z}_n$ and a random r in $\mathbb{Z}_n^*$
- ψ can be efficiently inverted if p,q known
- (Additive) Homomorphism: Enc(m).Enc(m') is Enc(m+m')
  - $\psi(m,r).\psi(m',r') = \psi(m+m',r.r')$

Paillier's Scheme

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

  To ensure $\gcd(n,\varphi(n))=1$

  - Isomorphism: $\psi(a,b) = g^a b^n \pmod{n^2}$ where $g=(1+n)$
- $\mathrm{Enc}(m) = \psi(m,r)$ for m in $\mathbb{Z}_n$ and a random r in $\mathbb{Z}_n^*$
- ψ can be efficiently inverted if p,q known
- (Additive) Homomorphism: $\mathrm{Enc}(m).\mathrm{Enc}(m')$ is $\mathrm{Enc}(m+m')$
  - $\psi(m,r).\psi(m',r') = \psi(m+m',r.r')$

in $\mathbb{Z}_n$

Paillier's Scheme

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

  To ensure $\gcd(n,\varphi(n))=1$

  - Isomorphism: $\psi(a,b) = g^a b^n \pmod{n^2}$ where $g=(1+n)$
- Enc(m) = $\psi(m,r)$ for m in $\mathbb{Z}_n$ and a random r in $\mathbb{Z}_n^*$
- ψ can be efficiently inverted if p,q known
- (Additive) Homomorphism: Enc(m).Enc(m') is Enc(m+m')
  - $\psi(m,r).\psi(m',r') = \psi(m+m',r.r')$

in $\mathbb{Z}_{n^2}^*$

in $\mathbb{Z}_n$

Paillier's Scheme

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

  To ensure $\gcd(n,\varphi(n))=1$

  - Isomorphism: $\psi(a,b) = g^a b^n \pmod{n^2}$ where $g=(1+n)$
- Enc(m) = ψ(m,r) for m in $\mathbb{Z}_n$ and a random r in $\mathbb{Z}_n^*$
- ψ can be efficiently inverted if p,q known
- (Additive) Homomorphism: Enc(m).Enc(m') is Enc(m+m')
  - $\psi(m,r).\psi(m',r') = \psi(m+m',r.r')$

  in $\mathbb{Z}_{n^2}^*$ in $\mathbb{Z}_n$
- IND-CPA secure under "Decisional Composite Residuosity" assumption: Given n=pq (but not p,q), ψ(0,rand) looks random (i.e. like ψ(rand,rand))

Paillier's Scheme

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

  To ensure $\gcd(n,\varphi(n))=1$

  - Isomorphism: $\psi(a,b) = g^a b^n \pmod{n^2}$ where $g=(1+n)$
- Enc(m) = $\psi(m,r)$ for m in $\mathbb{Z}_n$ and a random r in $\mathbb{Z}_n^*$
- ψ can be efficiently inverted if p,q known
- (Additive) Homomorphism: Enc(m).Enc(m') is Enc(m+m')
  - $\psi(m,r).\psi(m',r') = \psi(m+m',r.r')$

in $\mathbb{Z}_{n^2}^*$

in $\mathbb{Z}_n$

- IND-CPA secure under "Decisional Composite Residuosity" assumption: Given n=pq (but not p,q), $\psi(0,\text{rand})$ looks random (i.e. like $\psi(\text{rand},\text{rand})$)
- Unlinkability: ReRand(c) = c.Enc(0)

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 the entire database using this encryption (homomorphically), 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 the entire database using this encryption (homomorphically), 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. ord(1) $\geq$ m. For integer x and ciphertext $\underline{c}$, define x*c using "repeated doubling": $0*\underline{c} = E(0)$; $1*\underline{c} = \underline{c}$; $(a+b)*\underline{c} = \mathrm{Add}(\ a*\underline{c},\ b*\underline{c}\ )$.

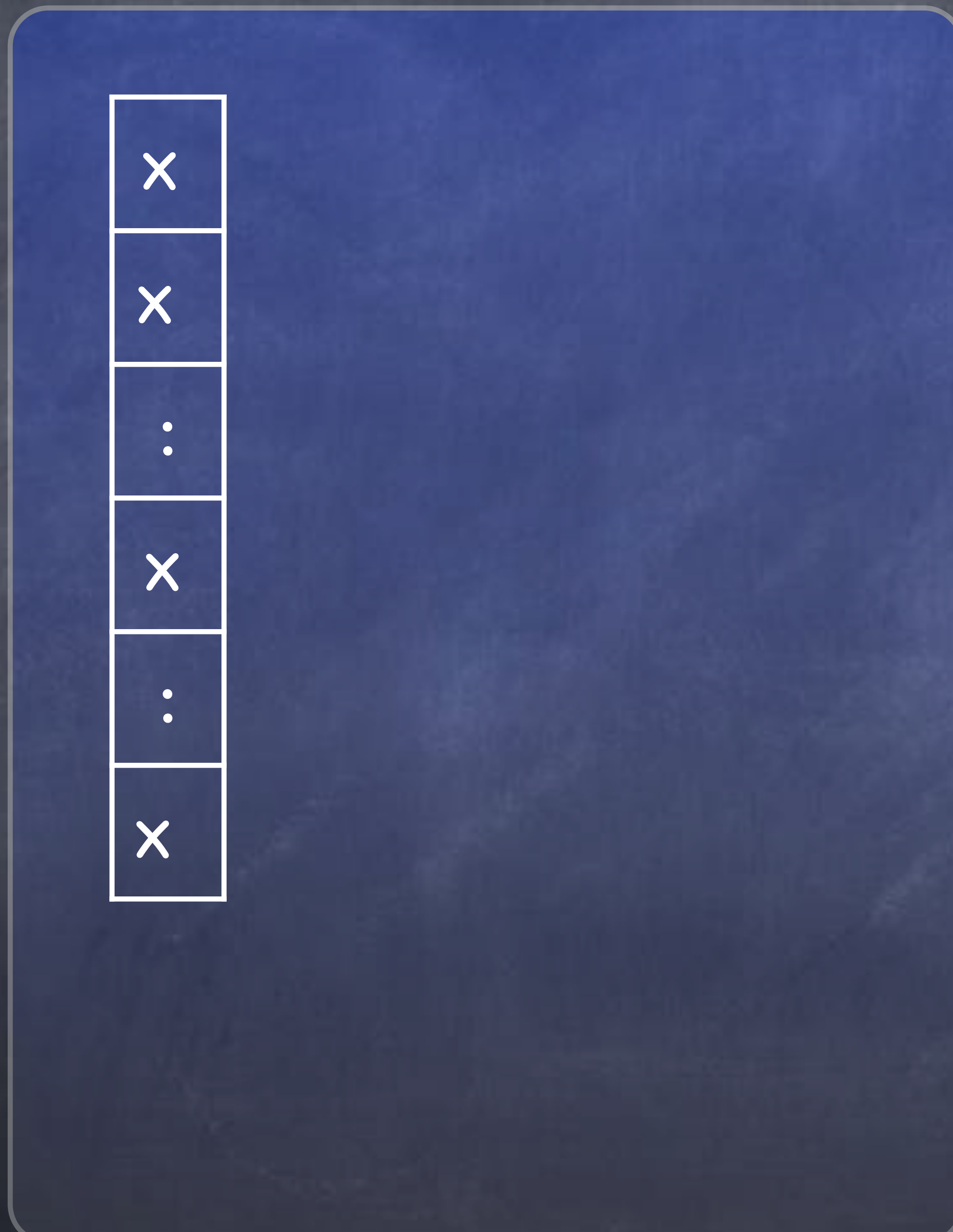
Private Information Retrieval

- Using additive homomorphic encryption (need not be unlinkable)
  - Client sends some encrypted representation of the index (need CPA security here)
  - Server operates on the entire database using this encryption (homomorphically), 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. ord(1) ≥ m. For integer x and ciphertext $\underline{c}$, define $x*\underline{c}$ using "repeated doubling": $0*\underline{c} = E(0)$; $1*\underline{c} = \underline{c}$; $(a+b)*\underline{c} = \mathrm{Add}(a*\underline{c}, b*\underline{c})$.

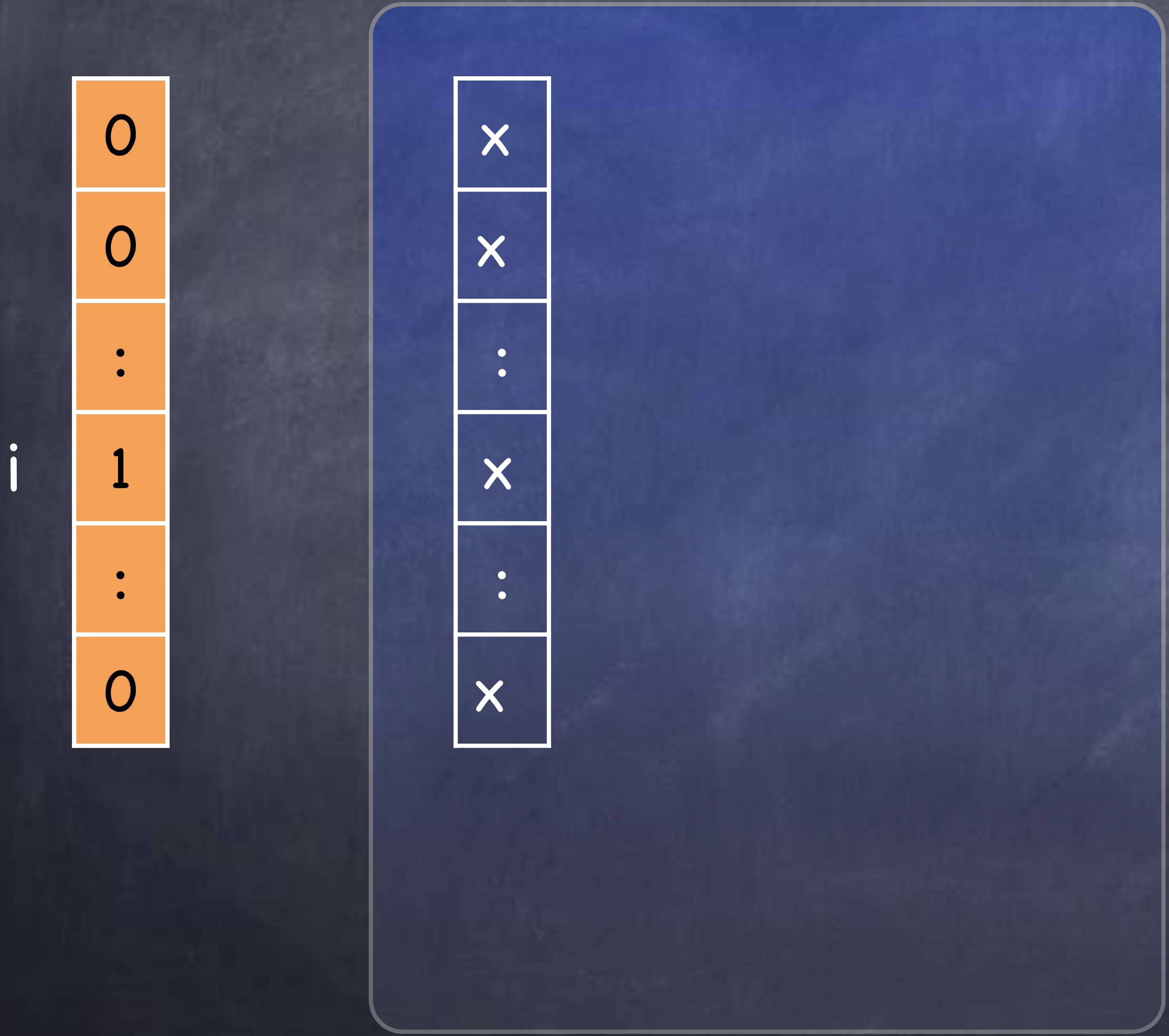
For Paillier, can use exponentiation

Private Information Retrieval

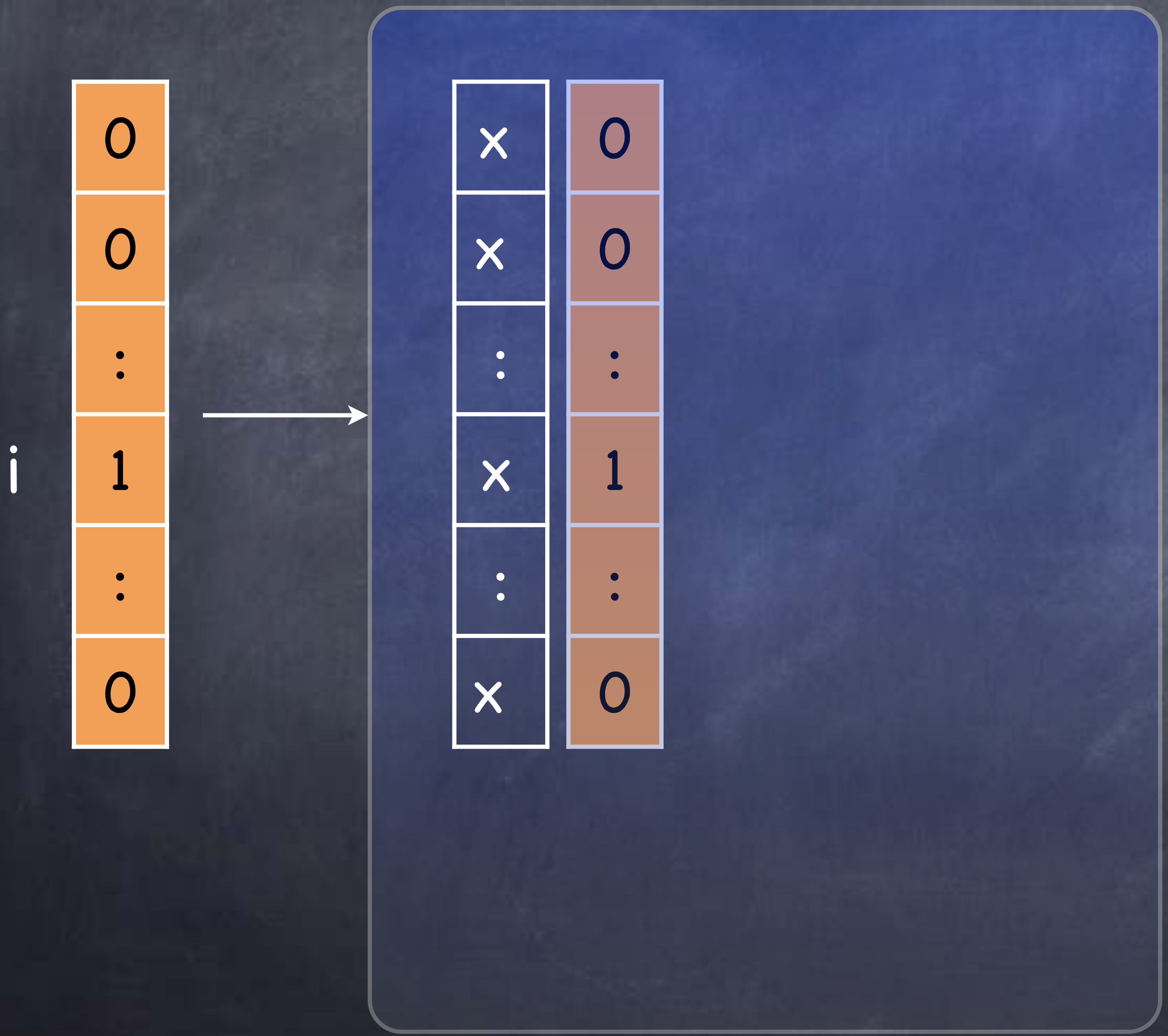
i



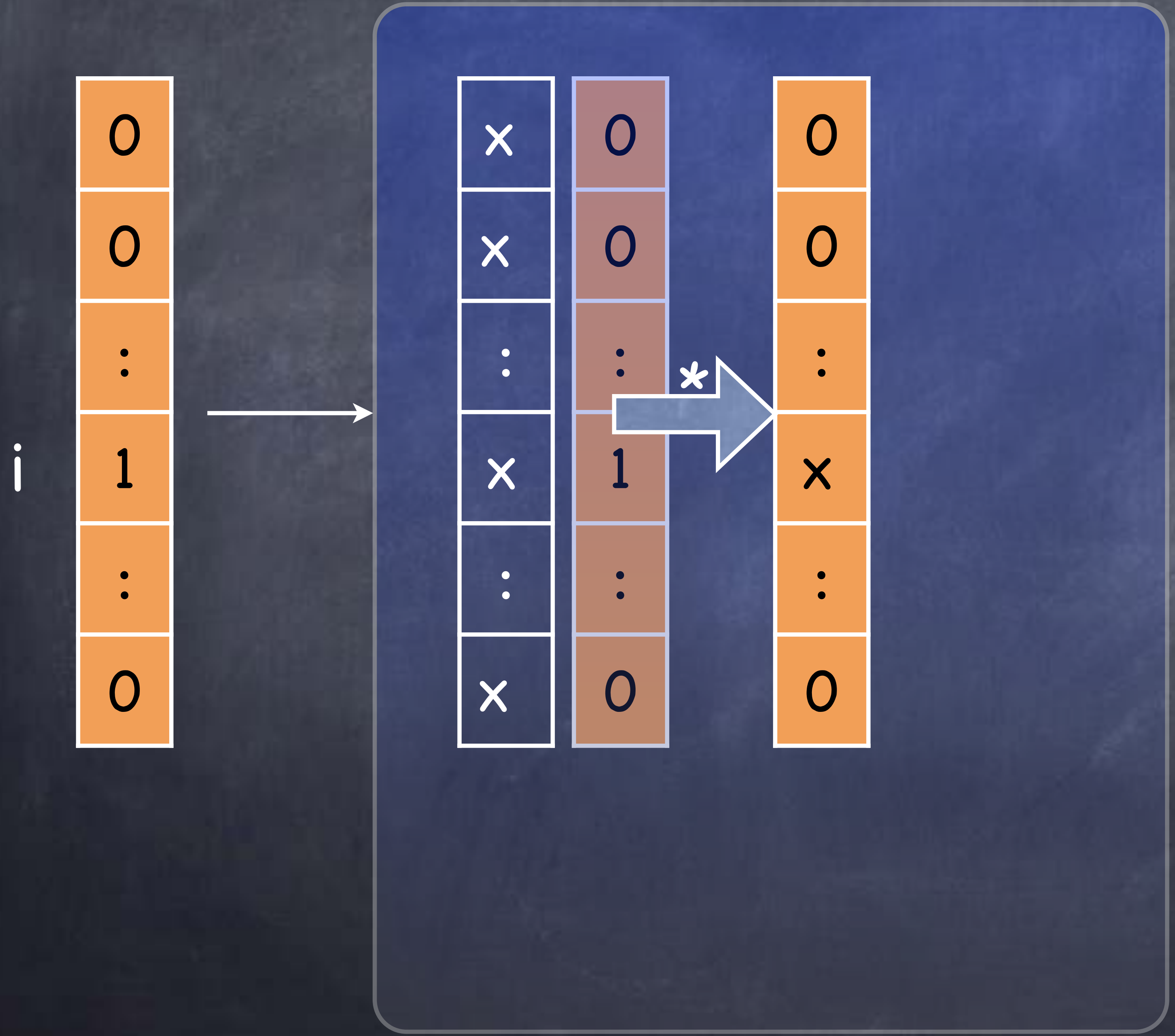
Private Information Retrieval



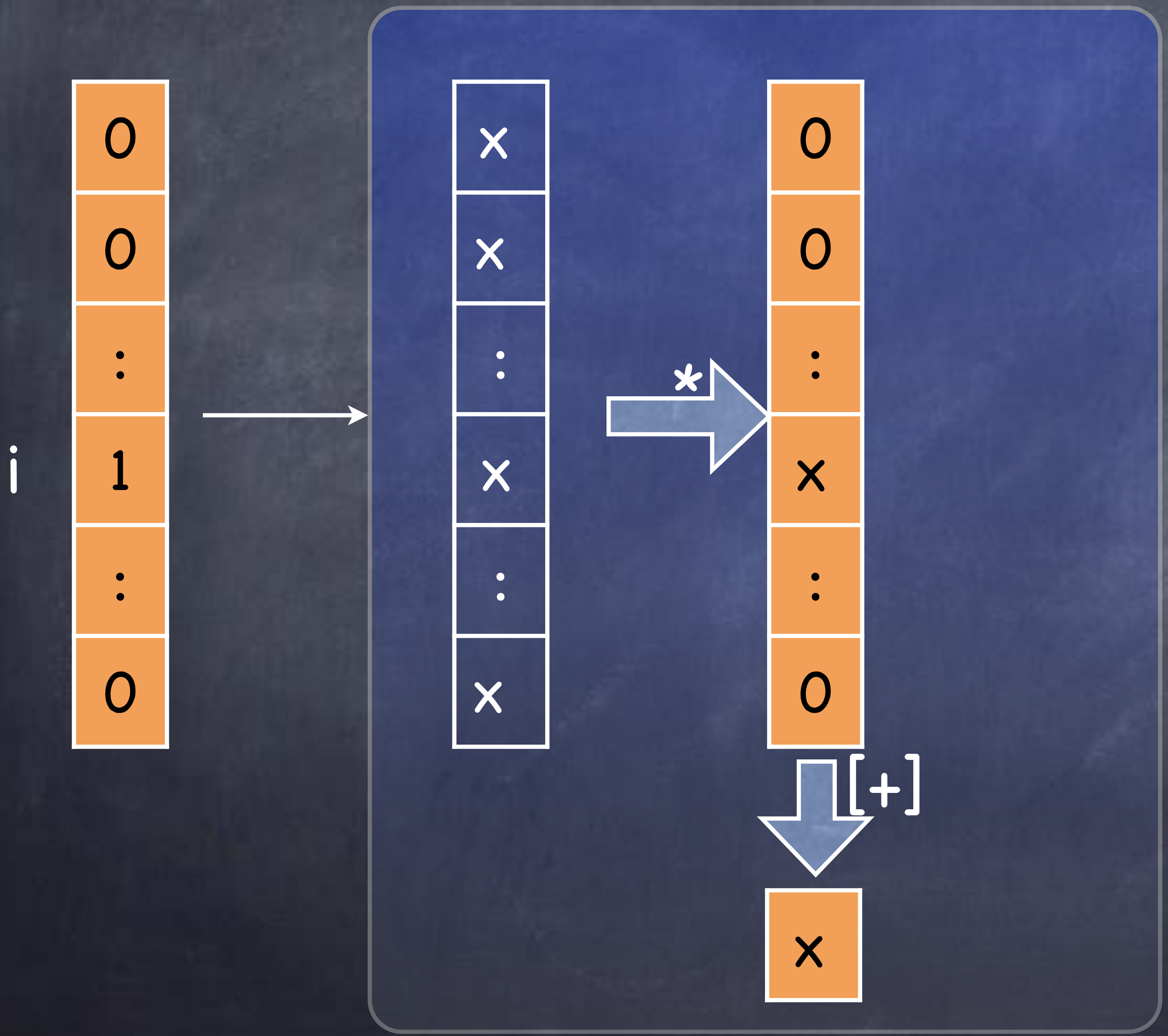
Private Information Retrieval



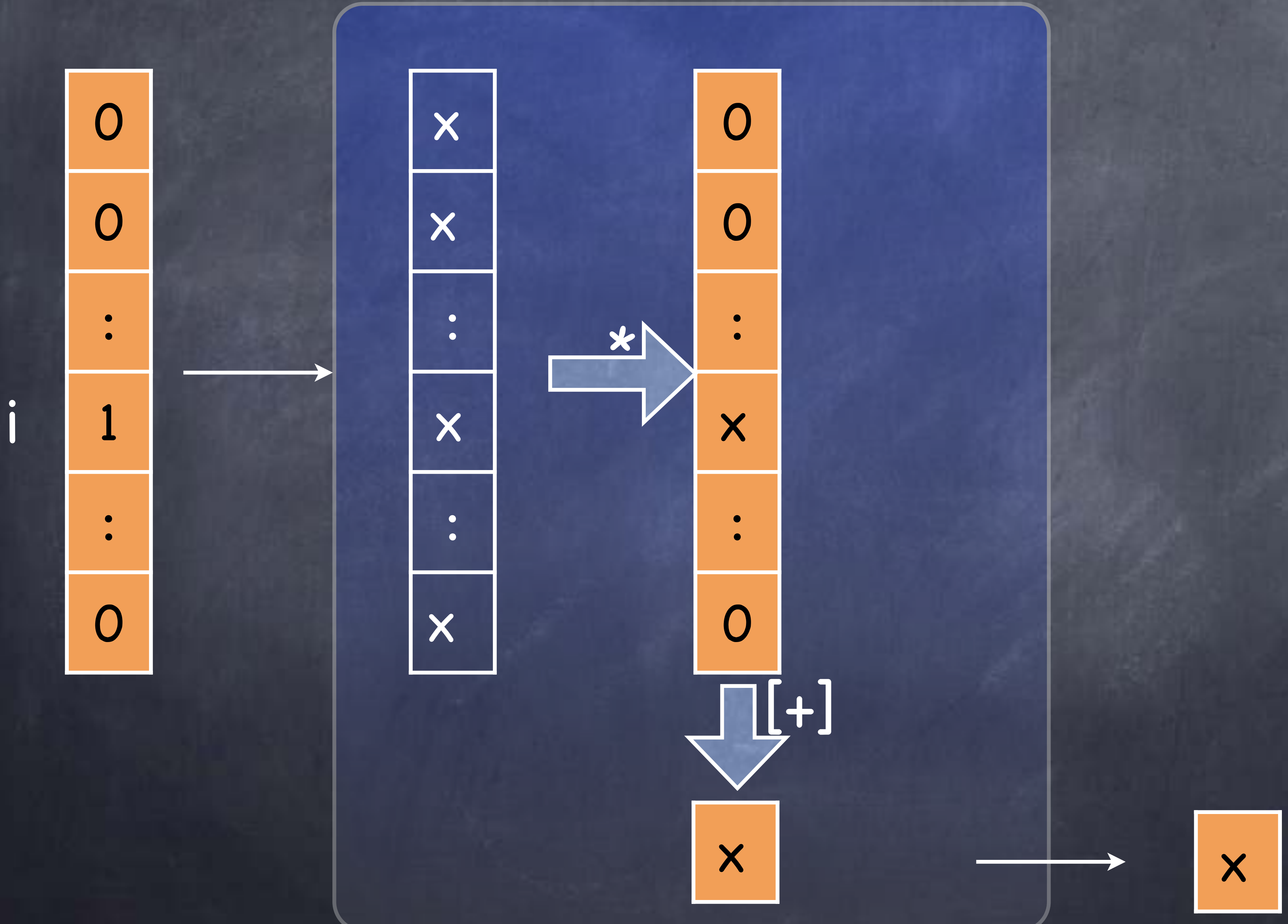
Private Information Retrieval



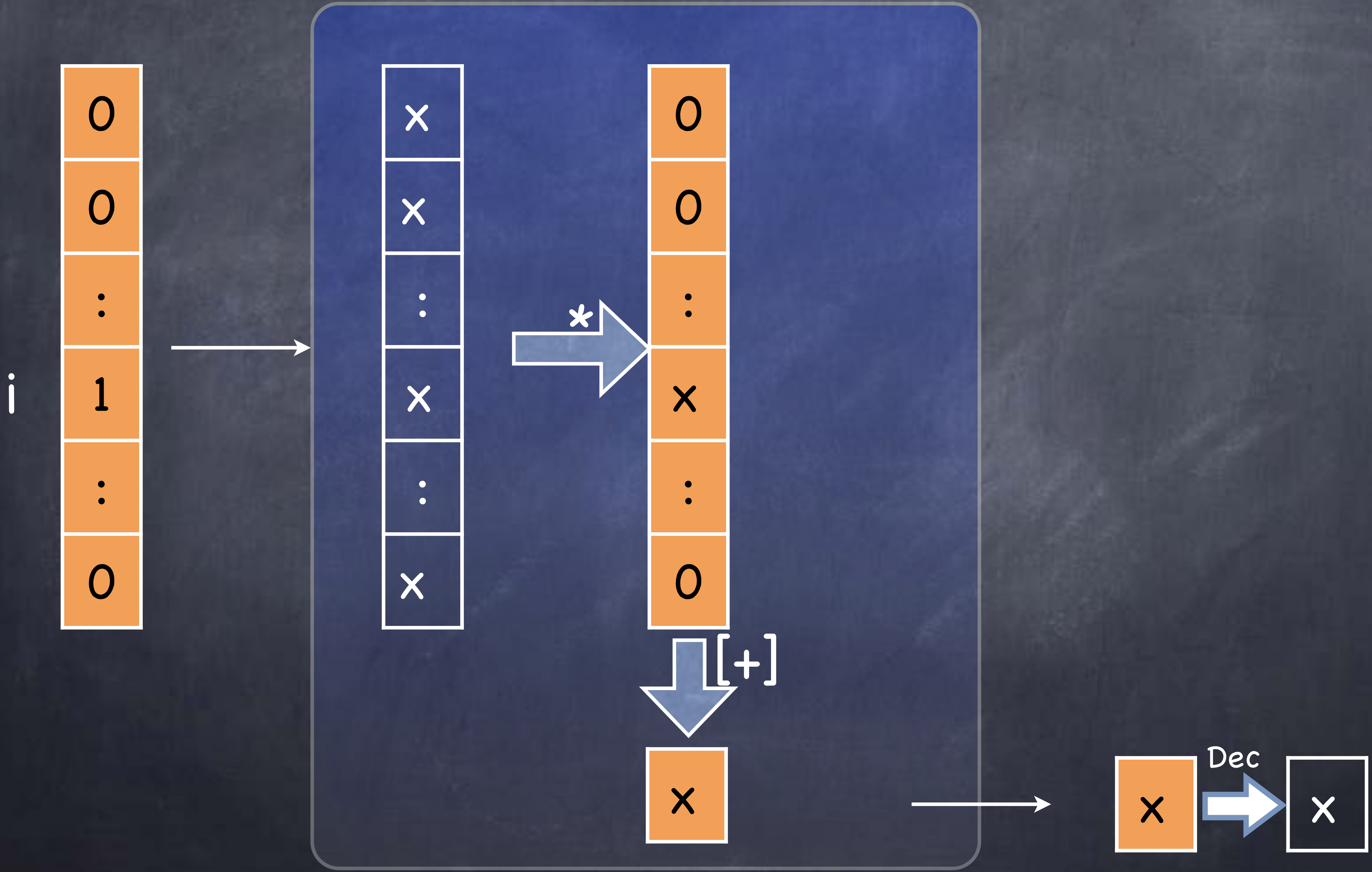
Private Information Retrieval



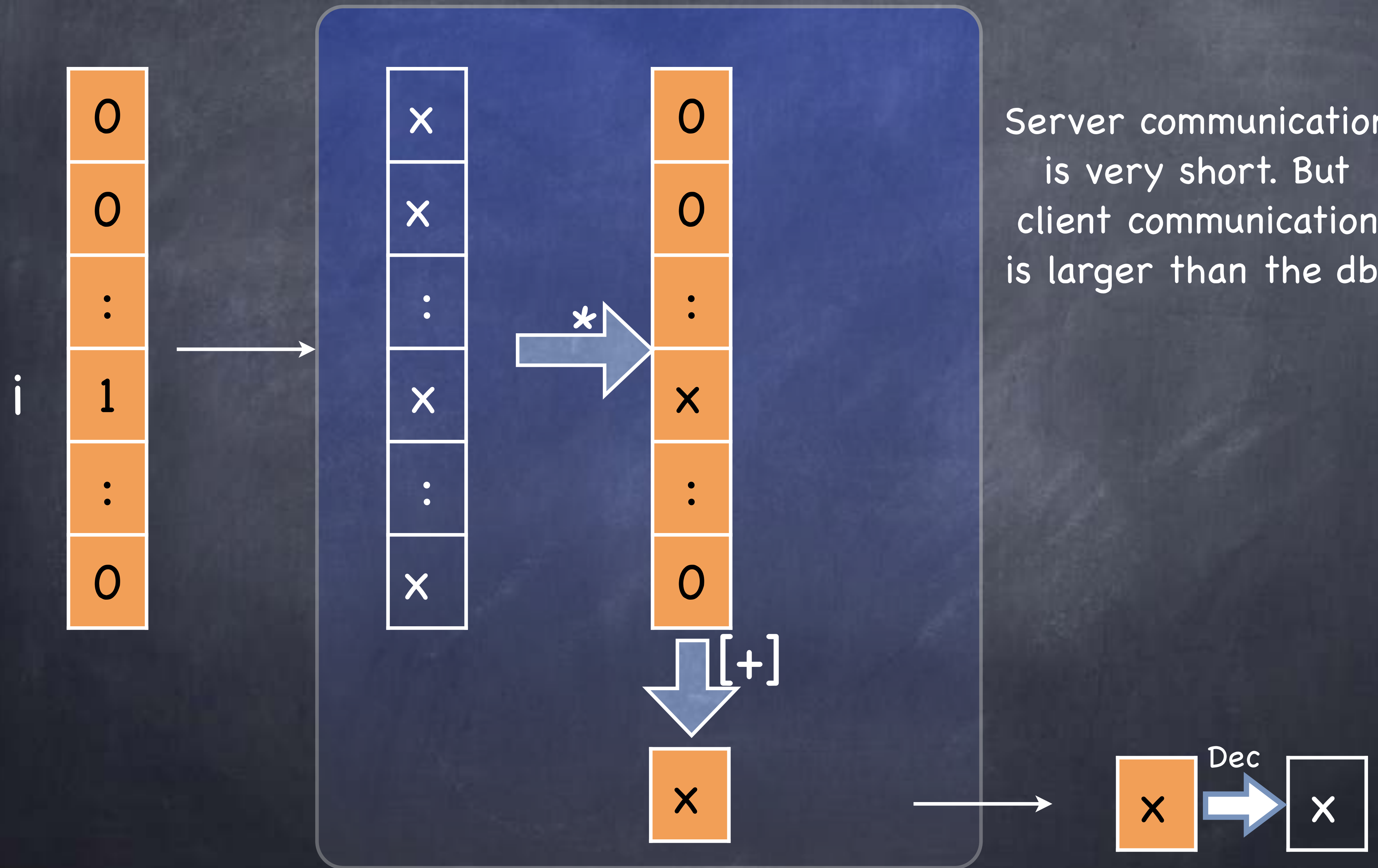
Private Information Retrieval



Private Information Retrieval



Private Information Retrieval

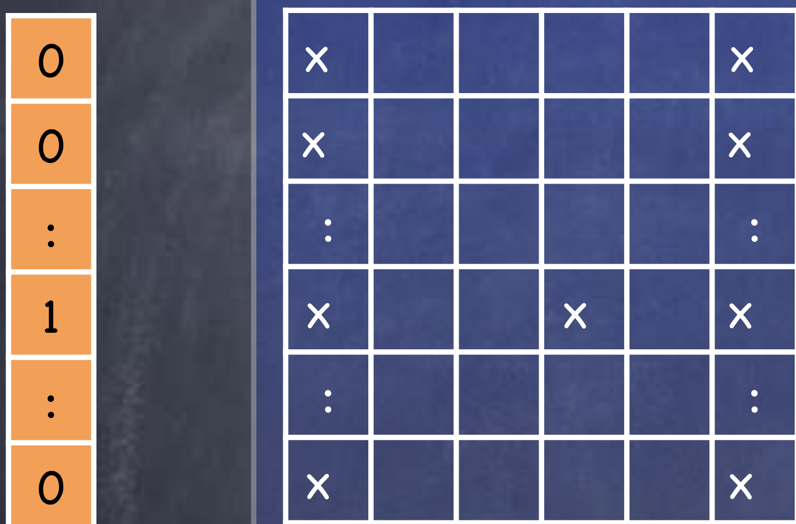


Server communication is very short. But client communication is larger than the db!

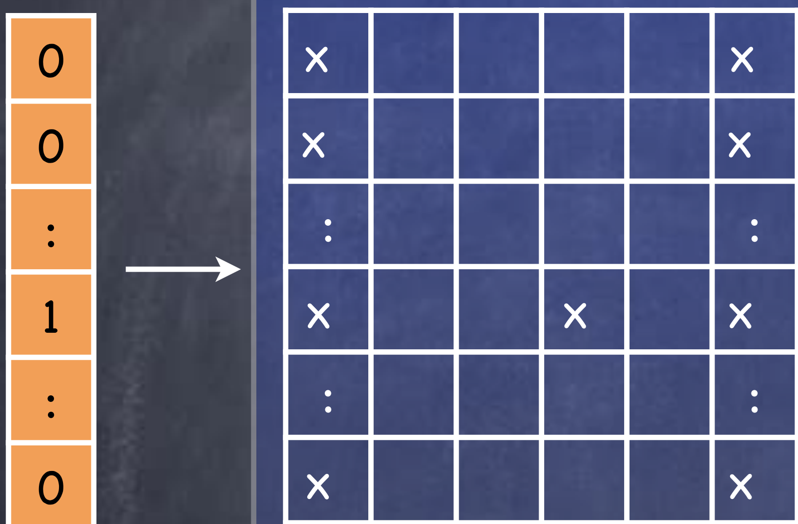
Private Information Retrieval

x					x
x					x
:					:
x			x		x
:					:
x					x

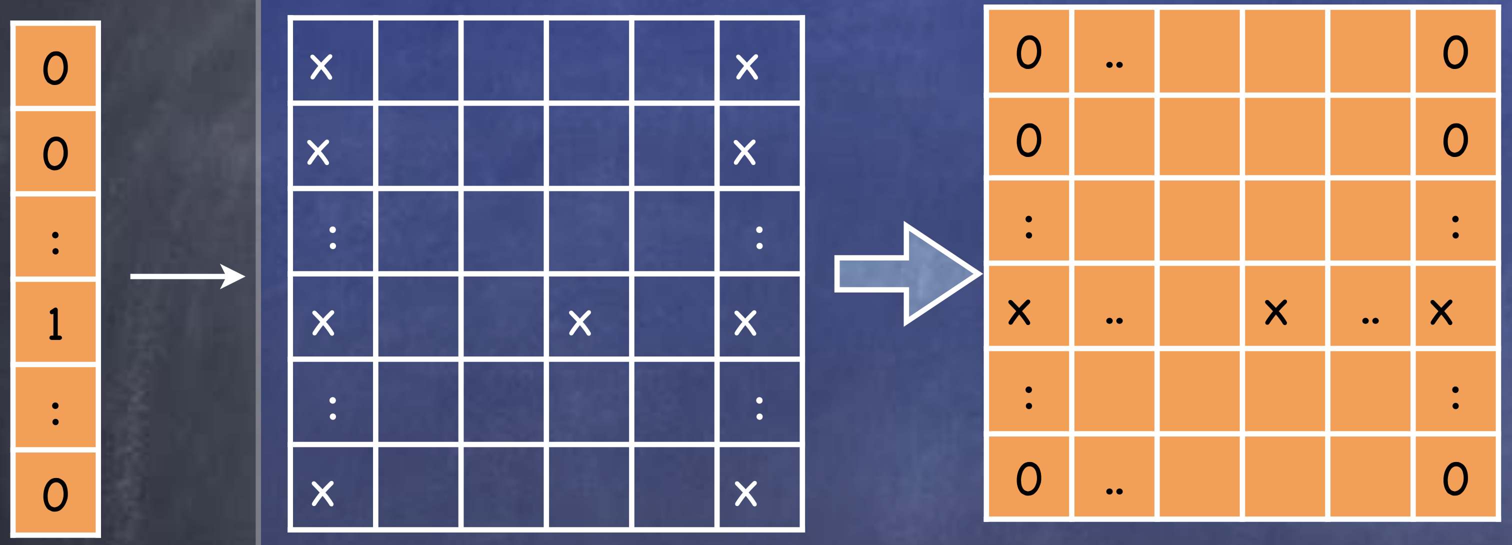
Private Information Retrieval



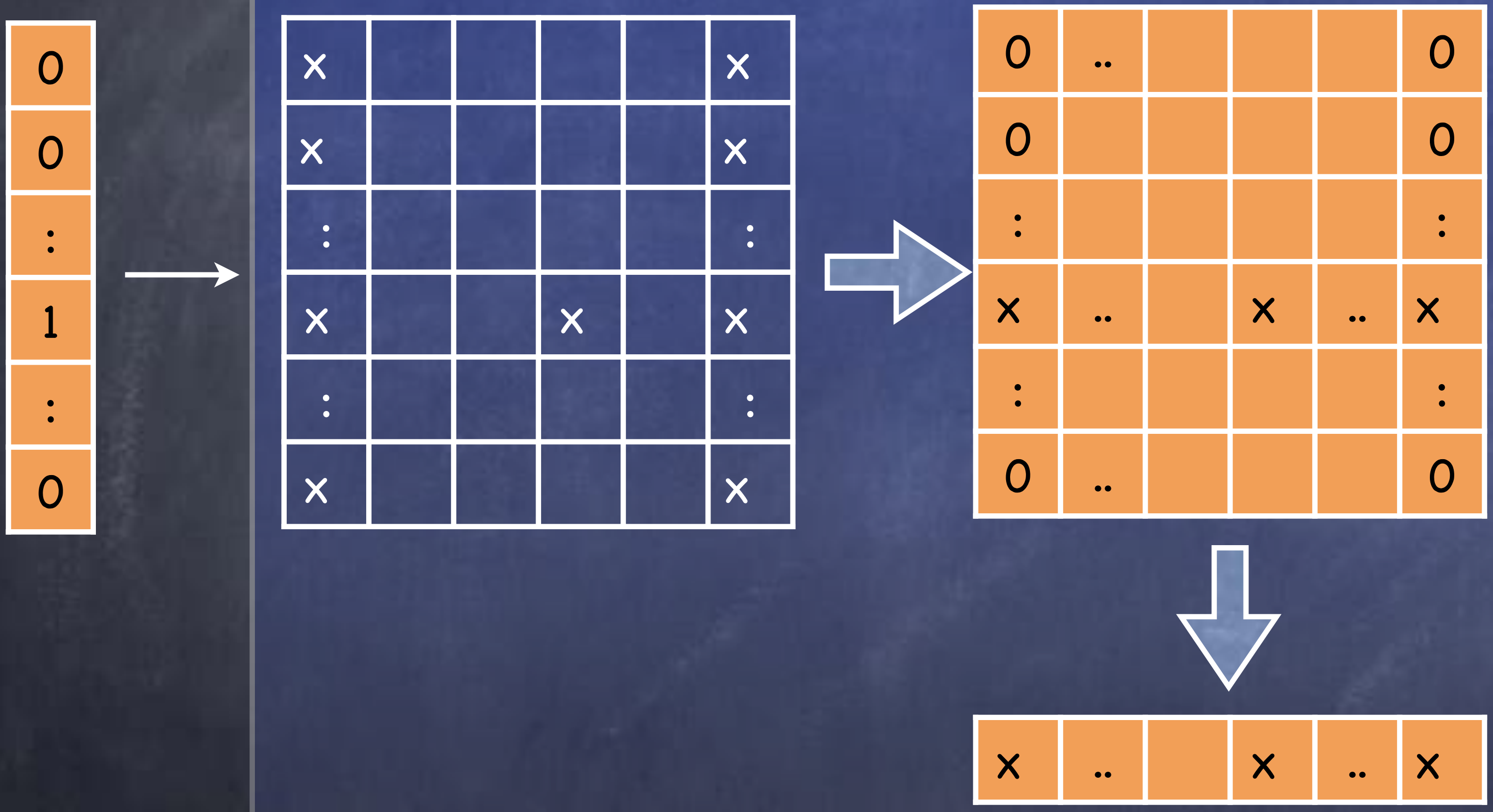
Private Information Retrieval



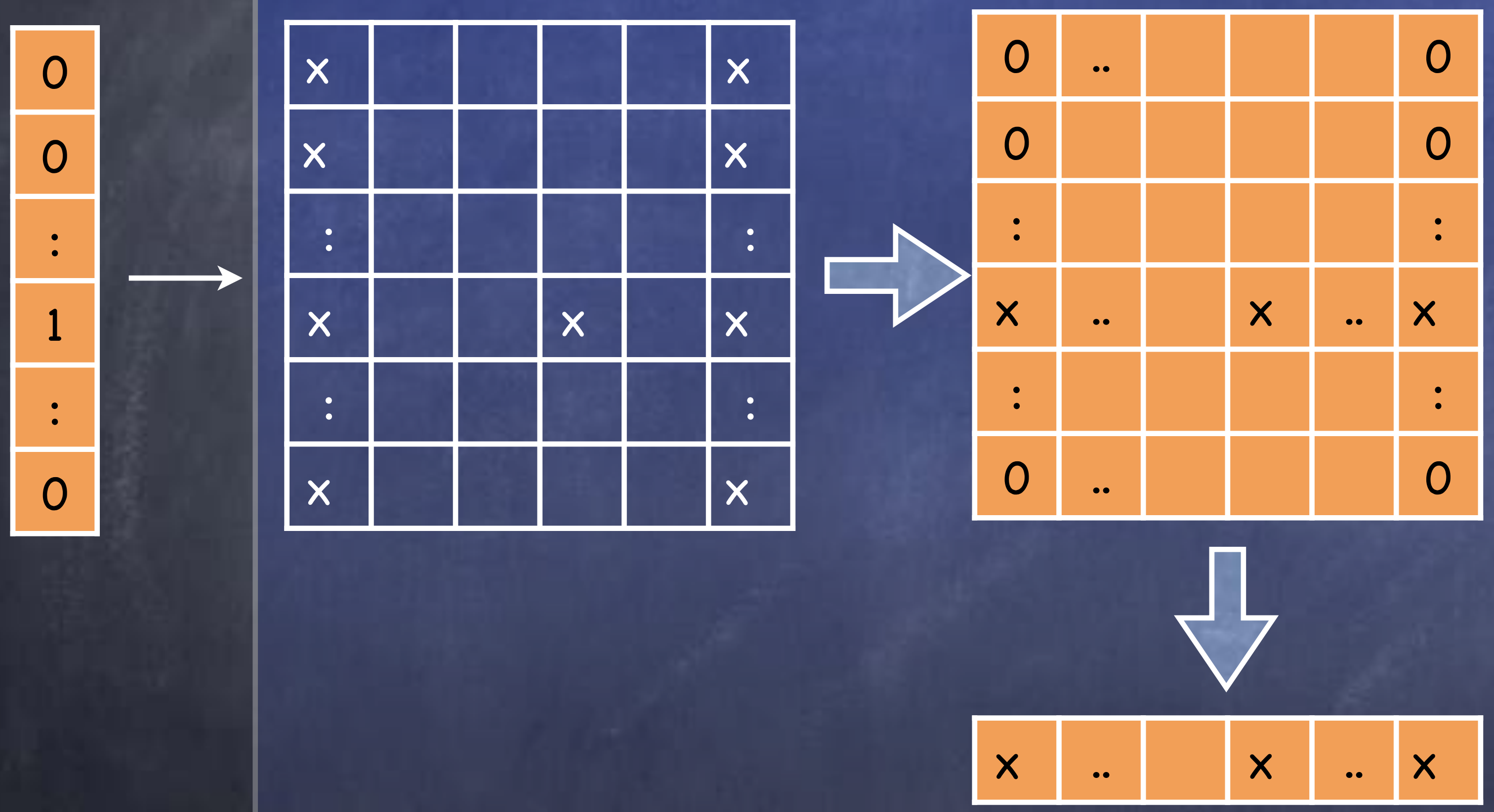
Private Information Retrieval



Private Information Retrieval

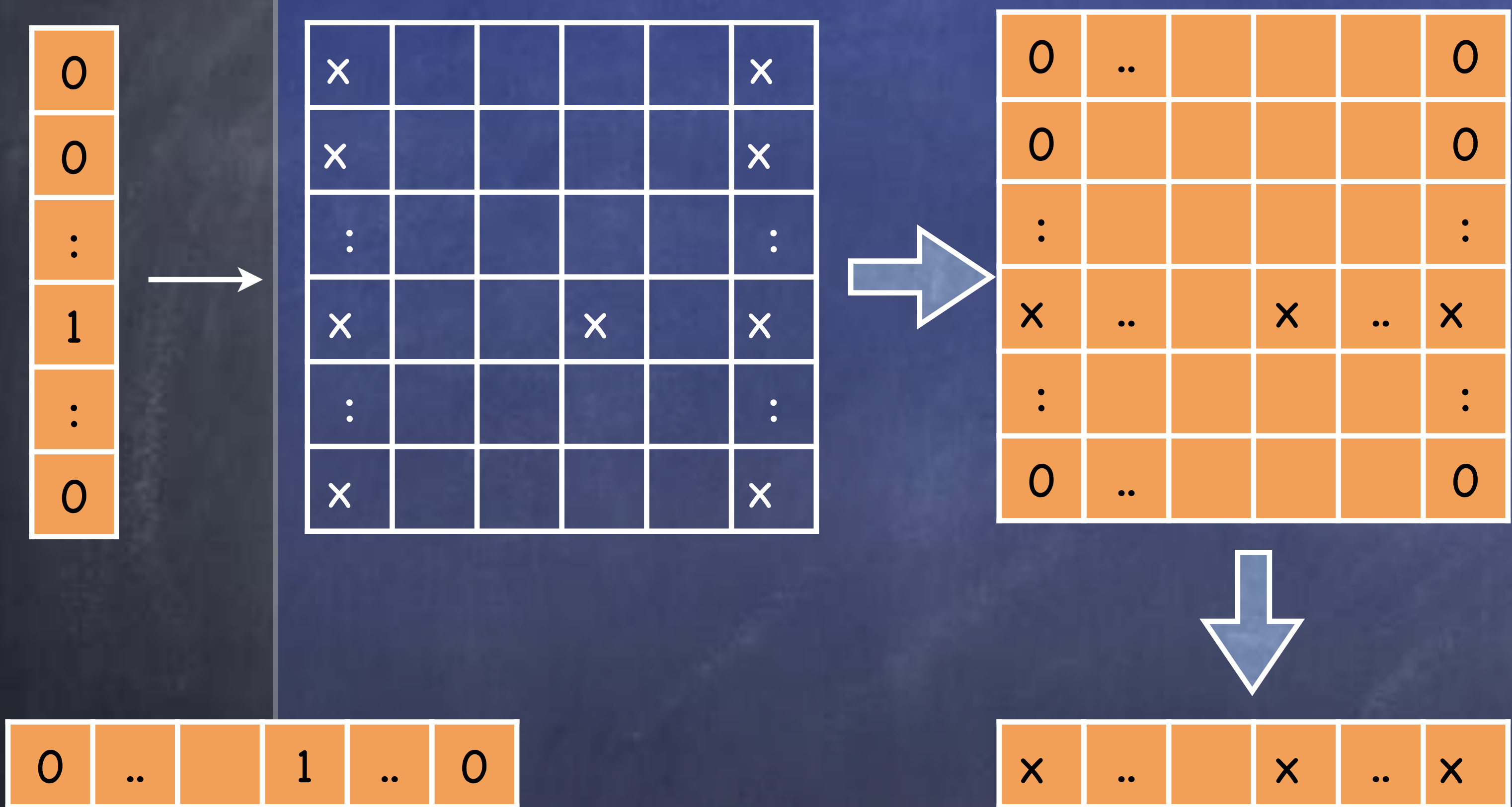


Private Information Retrieval



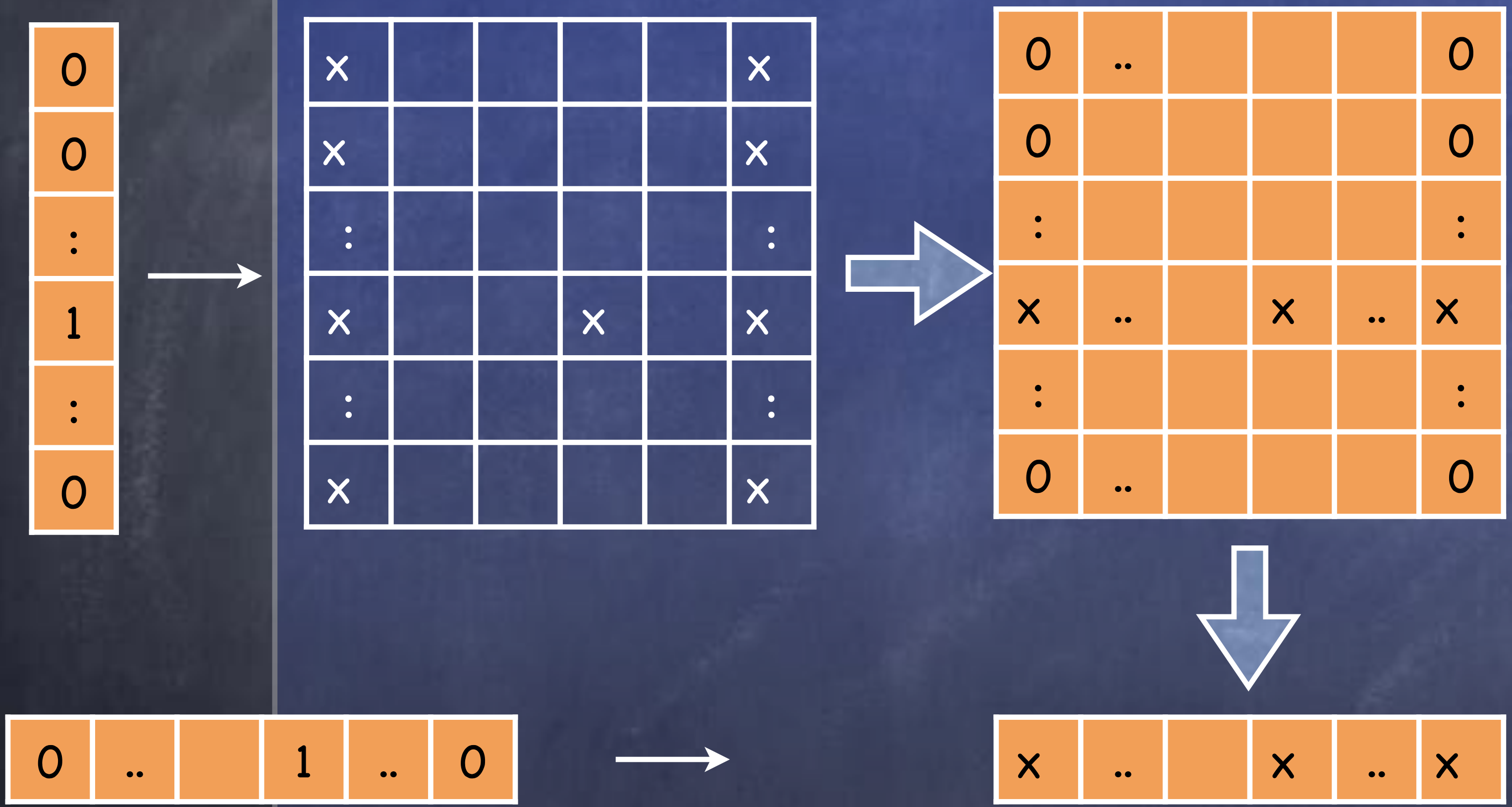
Use PIR again!

Private Information Retrieval



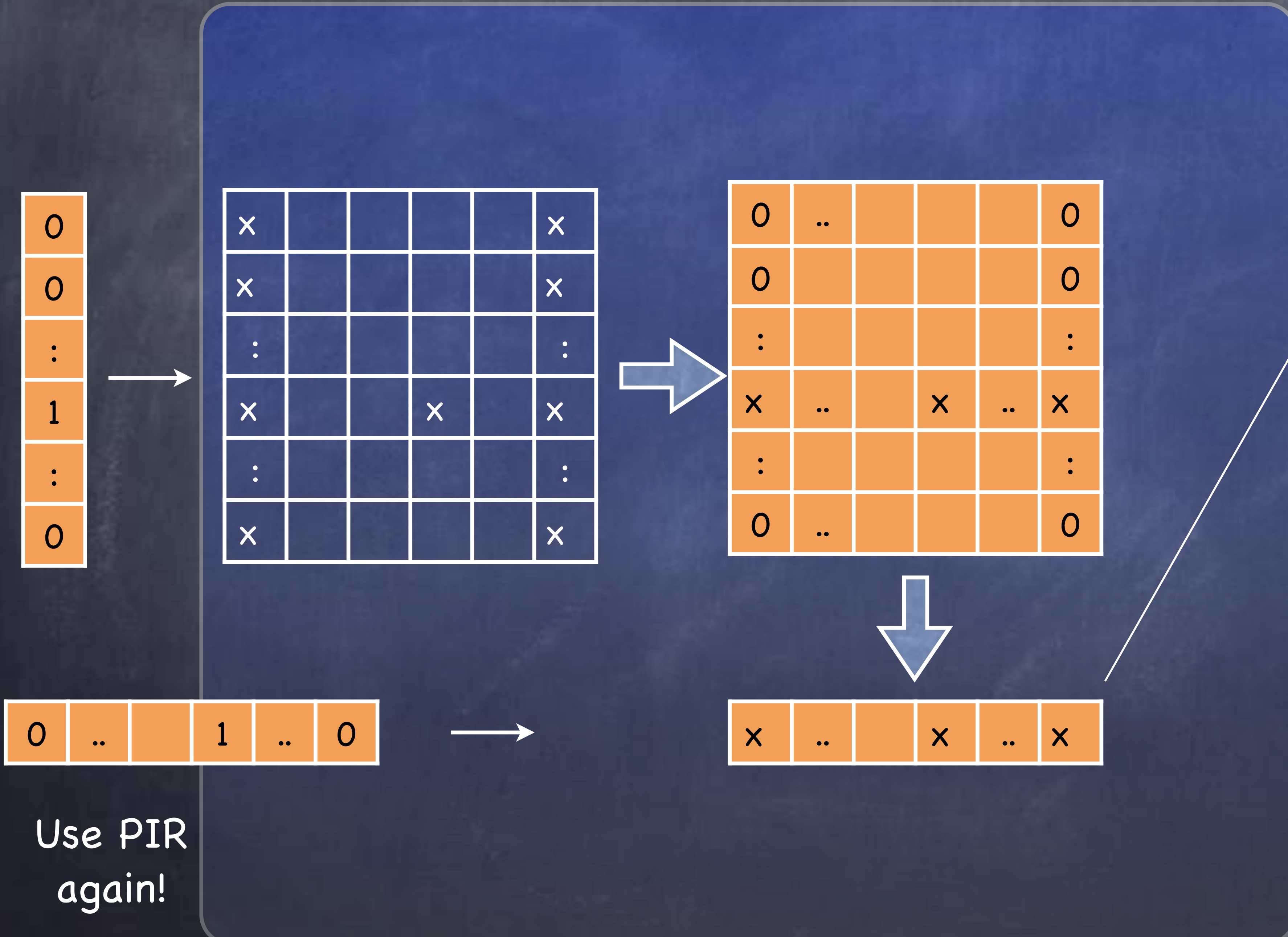
Use PIR again!

Private Information Retrieval



Use PIR again!

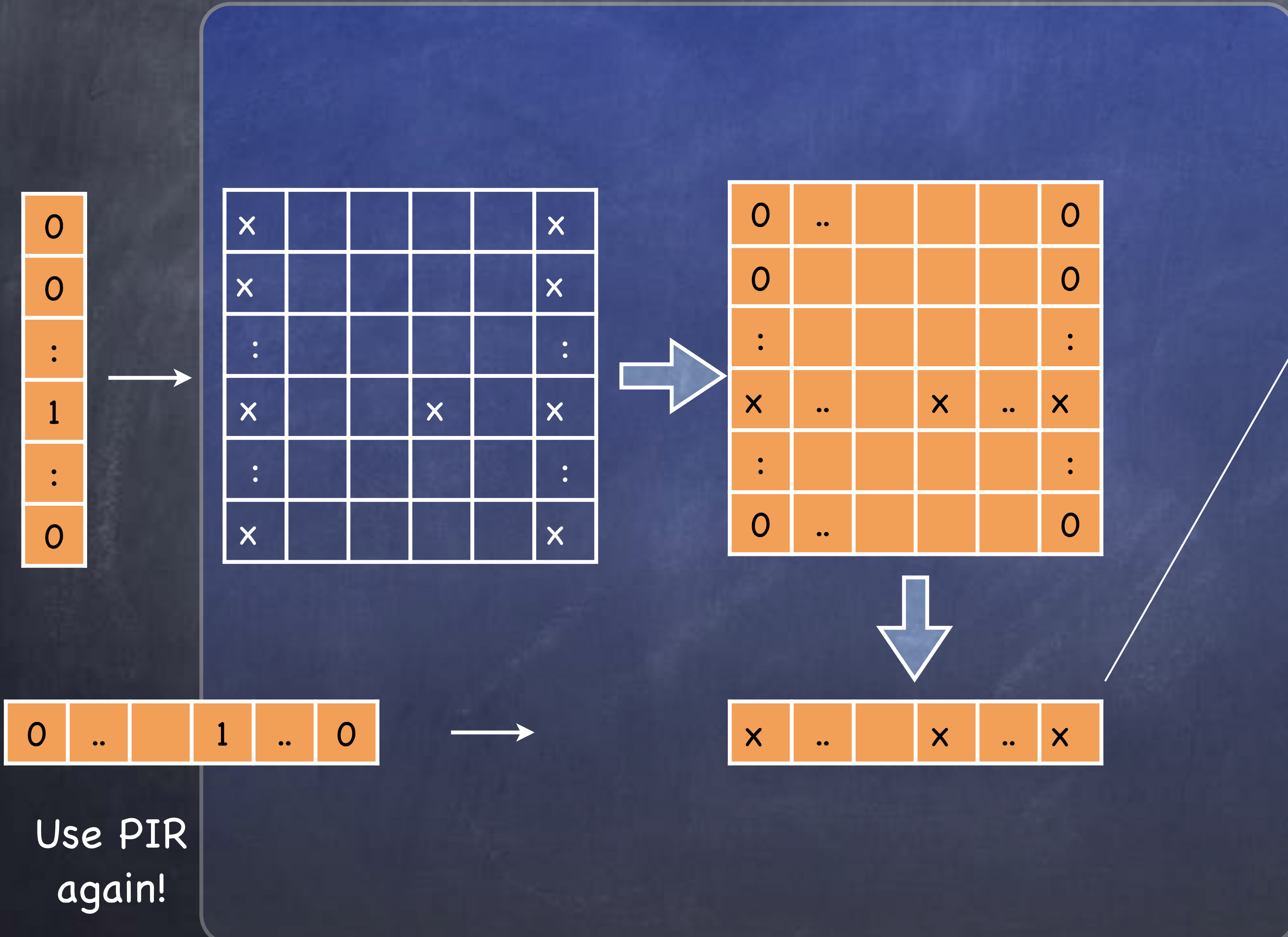
Private Information Retrieval



Considering ciphertext as plaintext for the sub-PIR

Use PIR again!

Private Information Retrieval

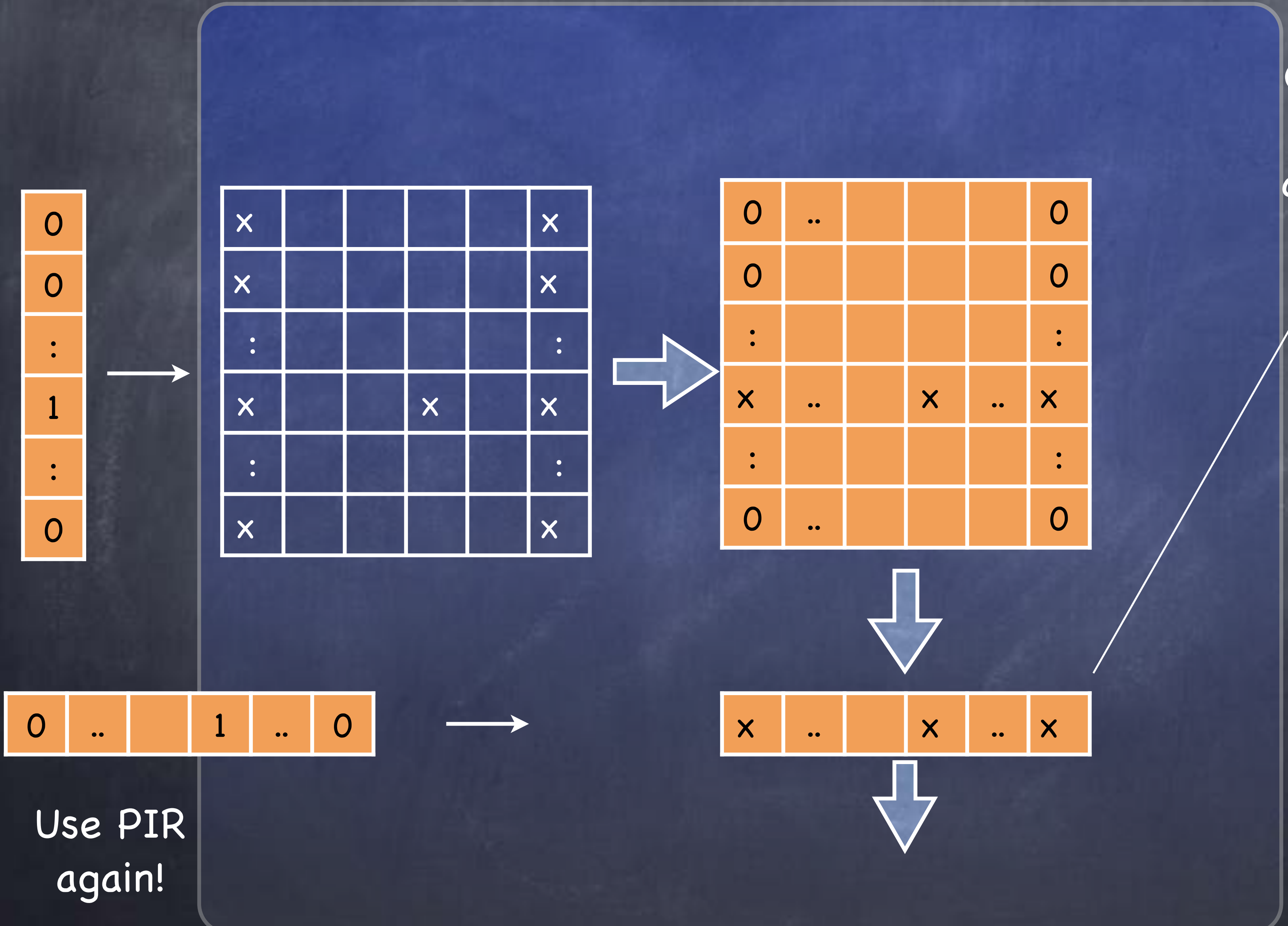


Considering ciphertext as plaintext for the sub-PIR

Can chop ciphertexts into smaller blocks

Use PIR again!

Private Information Retrieval

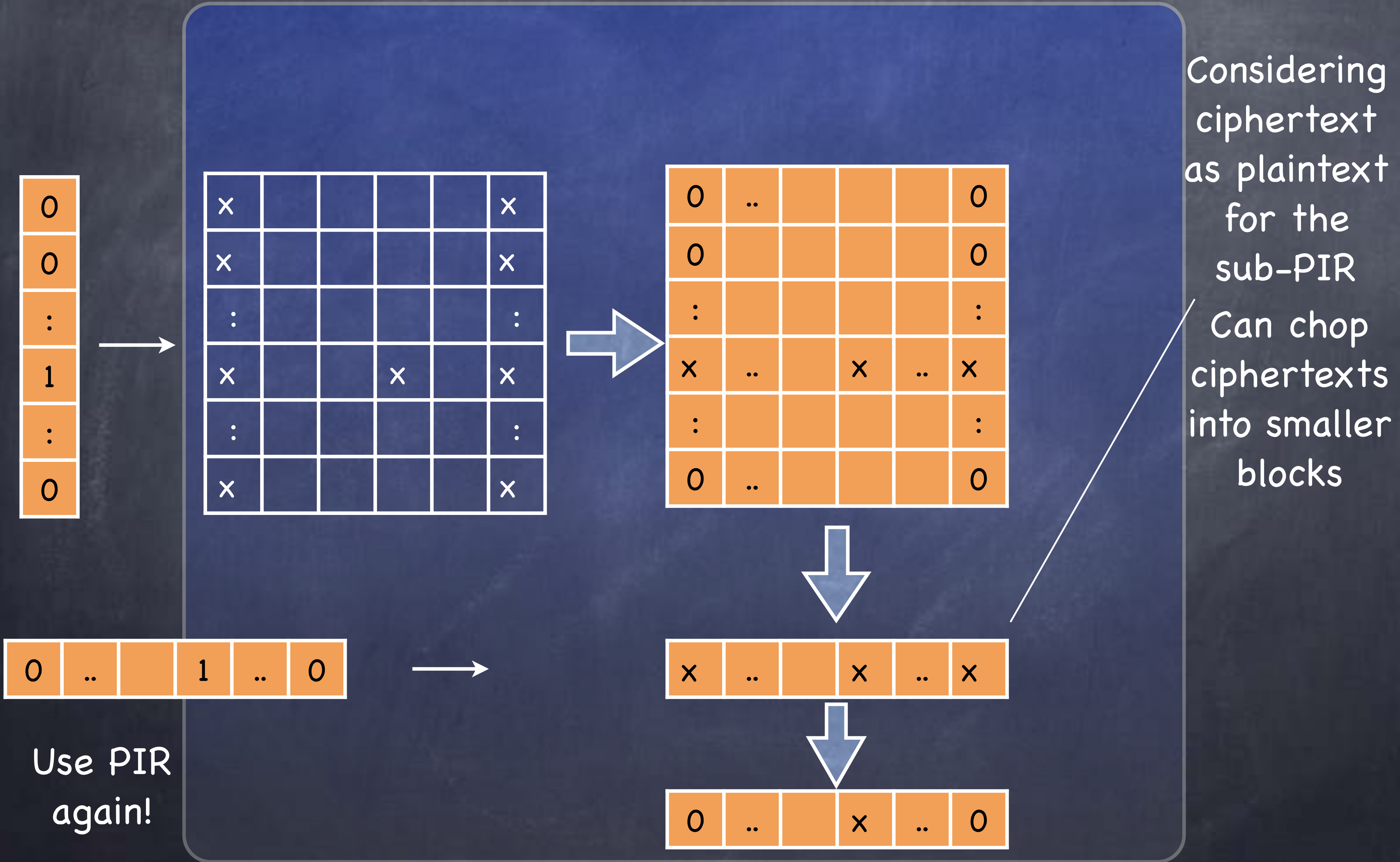


Considering ciphertext as plaintext for the sub-PIR

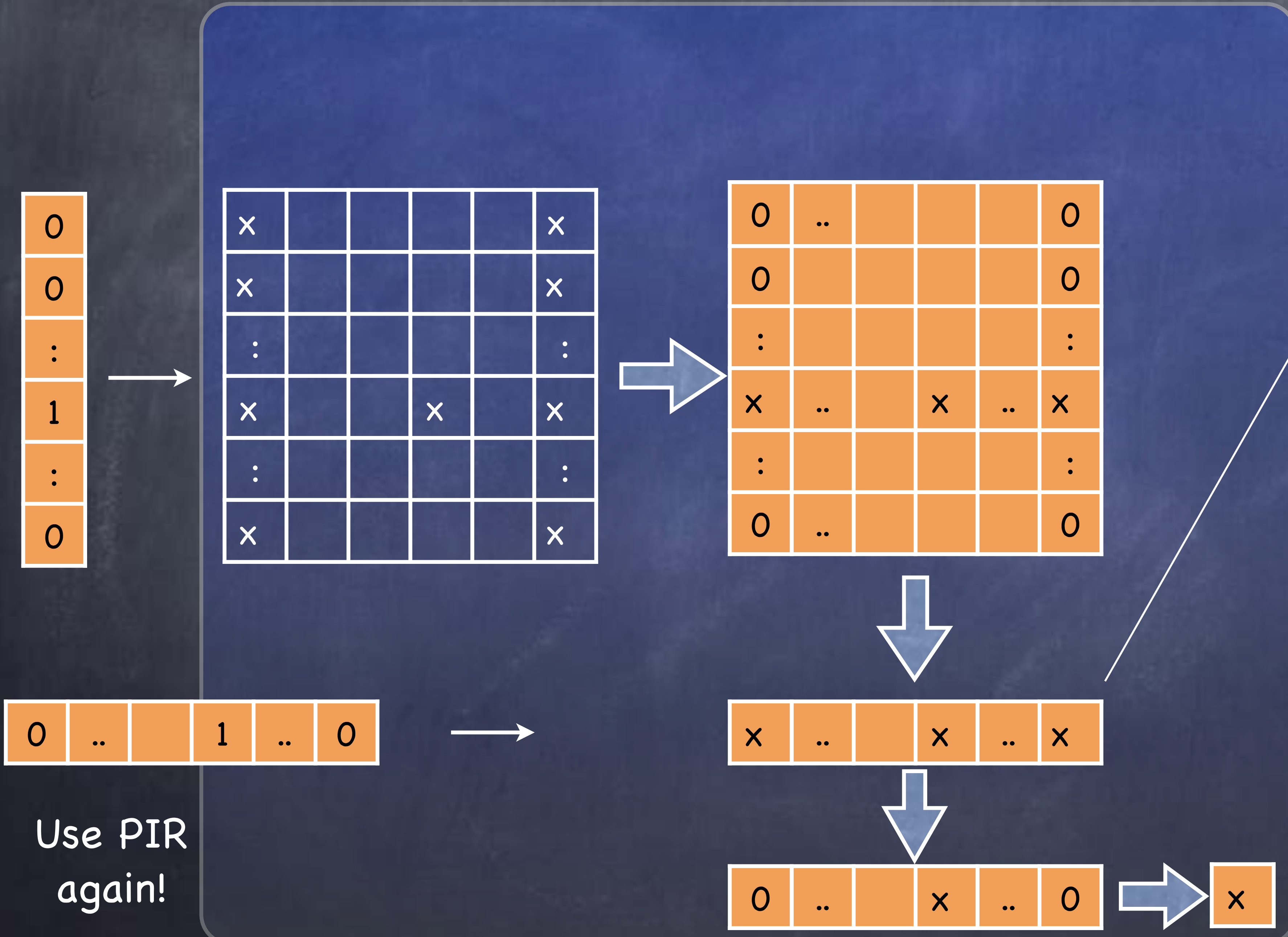
Can chop ciphertexts into smaller blocks

Use PIR again!

Private Information Retrieval



Private Information Retrieval

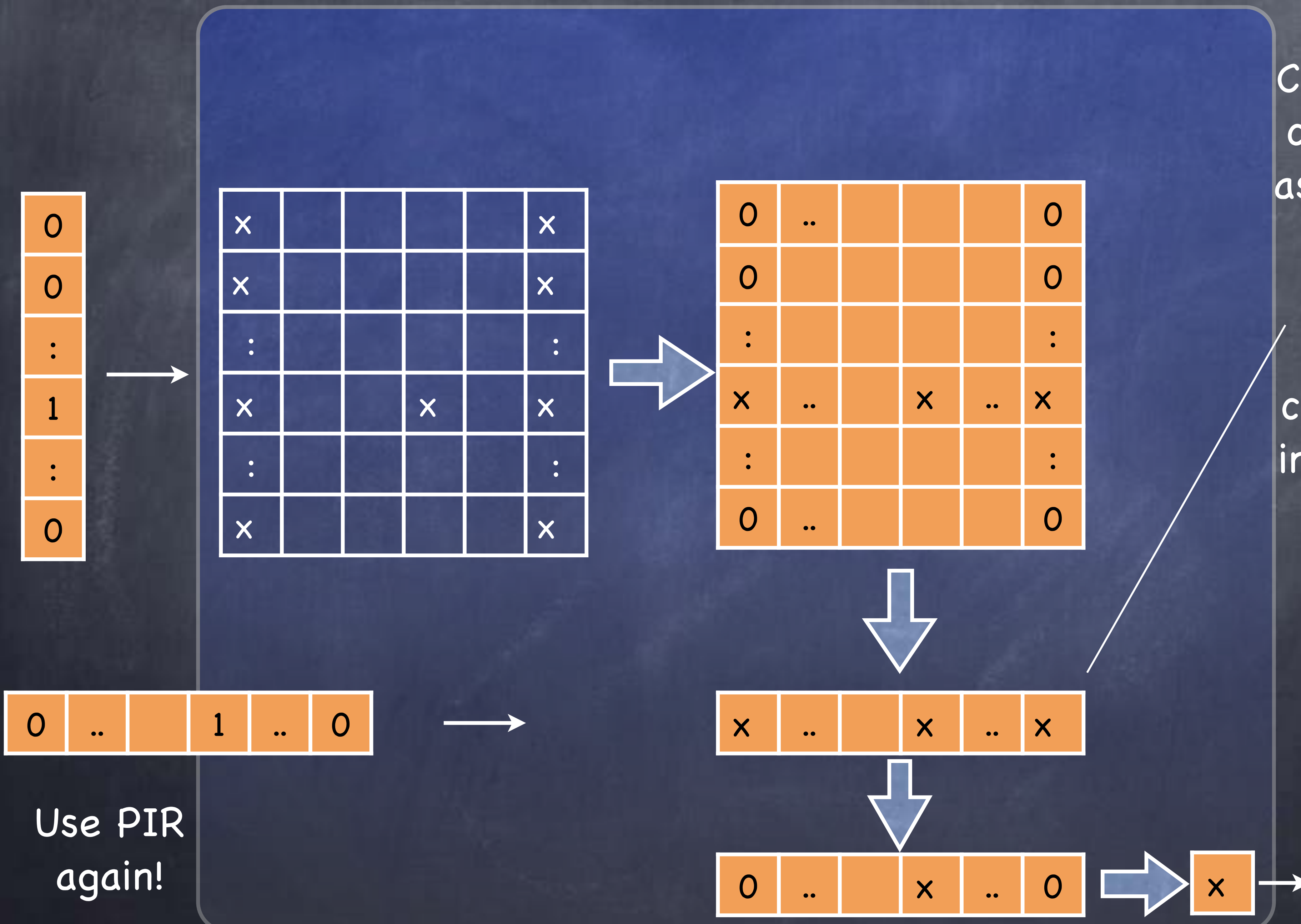


Considering ciphertext as plaintext for the sub-PIR

Can chop ciphertexts into smaller blocks

Use PIR again!

Private Information Retrieval

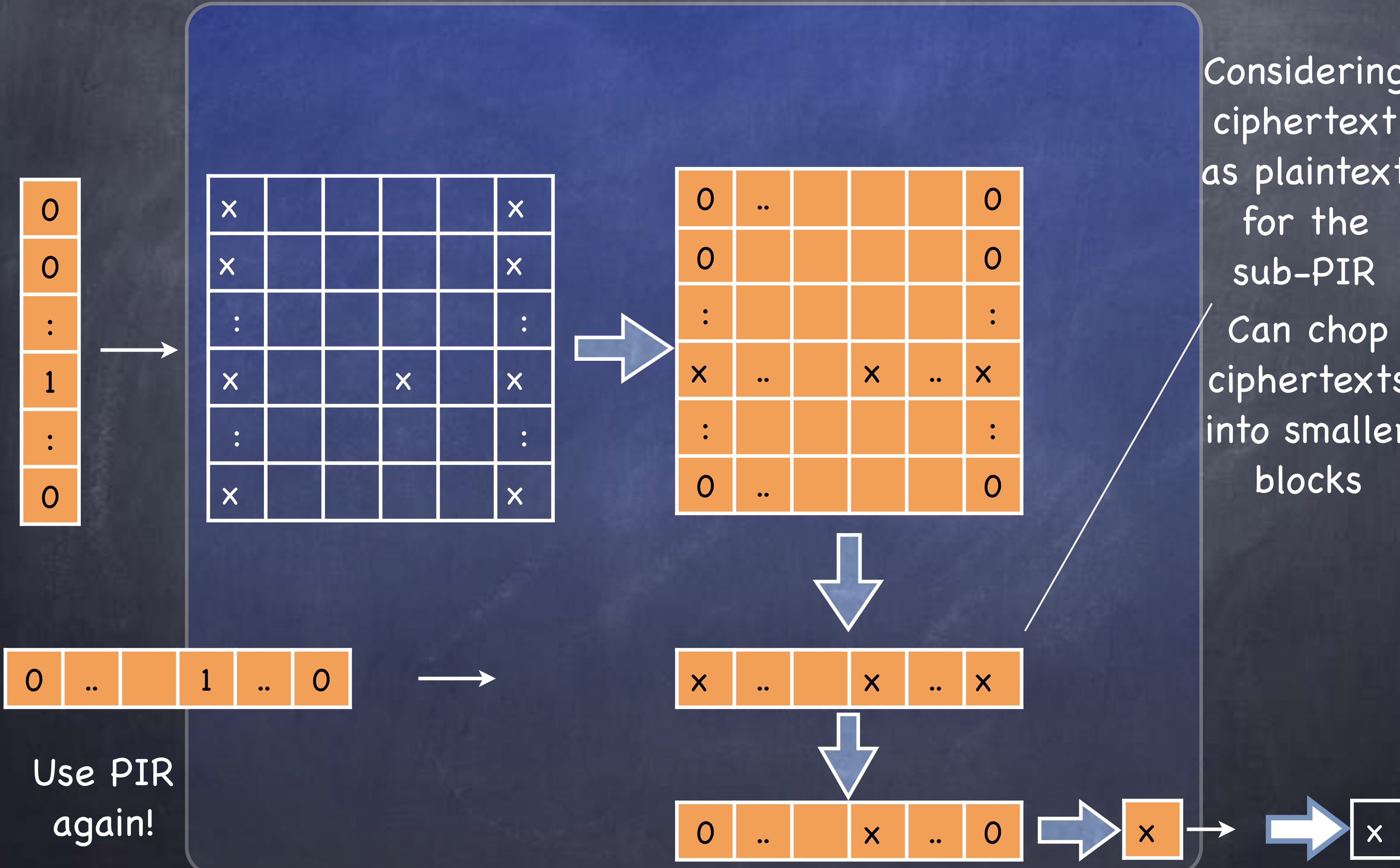


Considering ciphertext as plaintext for the sub-PIR

Can chop ciphertexts into smaller blocks

Use PIR again!

Private Information Retrieval

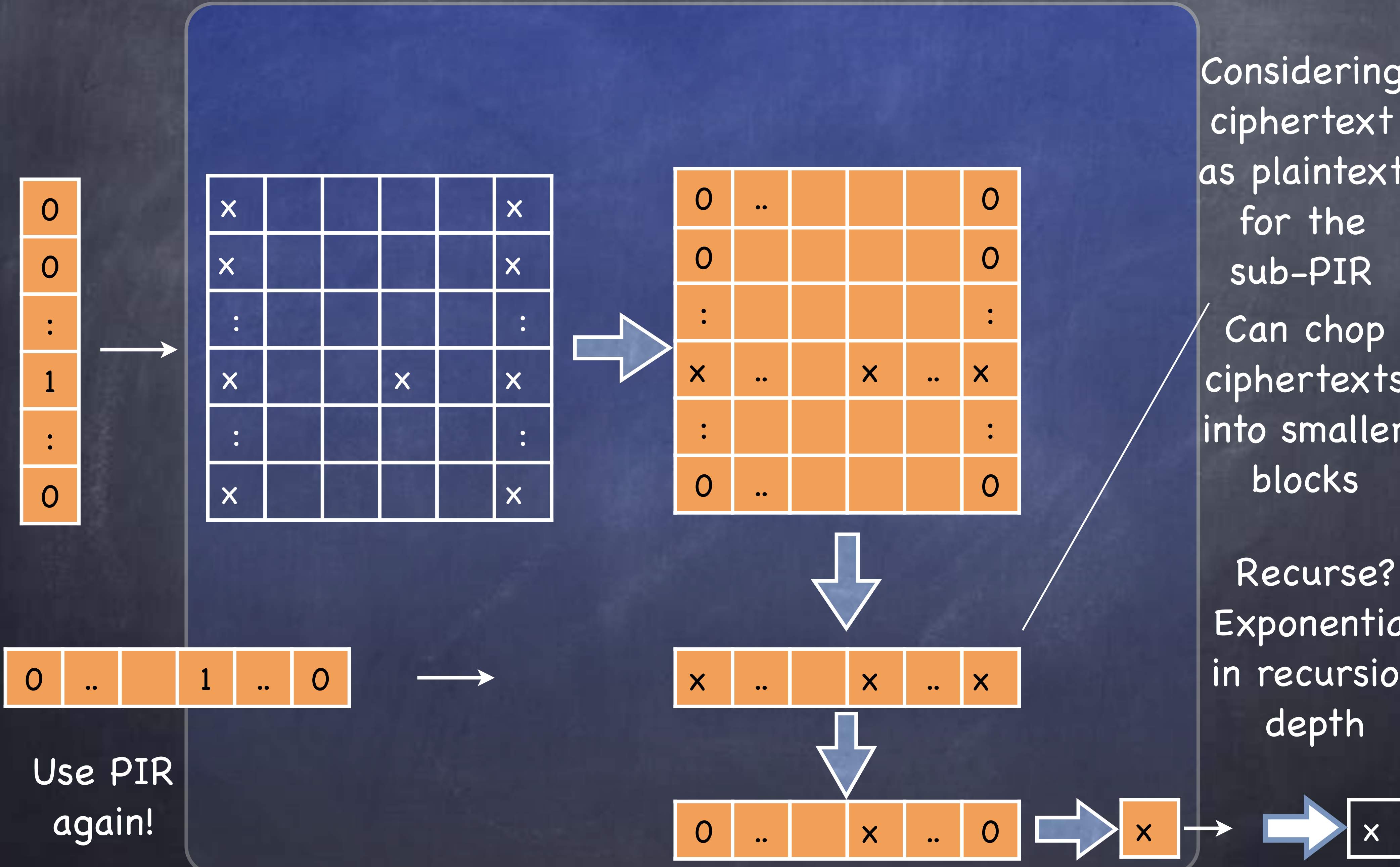


Considering ciphertext as plaintext for the sub-PIR

Can chop ciphertexts into smaller blocks

Use PIR again!

Private Information Retrieval



Considering ciphertext as plaintext for the sub-PIR

Can chop ciphertexts into smaller blocks

Recurse? Exponential in recursion depth

Use PIR again!

Private Information Retrieval

Private Information Retrieval

- Can dramatically improve efficiency if we have an efficient "recursive" homomorphic encryption scheme

Private Information Retrieval

- Can dramatically improve efficiency if we have an efficient "recursive" homomorphic encryption scheme
  - Ciphertext in one level is plaintext in the next level

Private Information Retrieval

- Can dramatically improve efficiency if we have an efficient "recursive" homomorphic encryption scheme
    - Ciphertext in one level is plaintext in the next level
        - In Paillier, public-key (i.e., n) fixes the group for homomorphic operation (i.e., $\mathbb{Z}_n$)

Private Information Retrieval

- Can dramatically improve efficiency if we have an efficient "recursive" homomorphic encryption scheme
  - Ciphertext in one level is plaintext in the next level
    - In Paillier, public-key (i.e., n) fixes the group for homomorphic operation (i.e., $\mathbb{Z}_n$)
  - Ciphertext size increases only "additively" from level to level

Private Information Retrieval

- Can dramatically improve efficiency if we have an efficient "recursive" homomorphic encryption scheme
  - Ciphertext in one level is plaintext in the next level
    - In Paillier, public-key (i.e., n) fixes the group for homomorphic operation (i.e., $\mathbb{Z}_n$)
  - Ciphertext size increases only "additively" from level to level
    - In Paillier, size of ciphertext about double that of the plaintext. (Note: can't use "hybrid encryption" if homomorphic property is to be preserved.)

Private Information Retrieval

- Can dramatically improve efficiency if we have an efficient "recursive" homomorphic encryption scheme
  - Ciphertext in one level is plaintext in the next level
    - In Paillier, public-key (i.e., n) fixes the group for homomorphic operation (i.e., $\mathbb{Z}_n$)
  - Ciphertext size increases only "additively" from level to level
    - In Paillier, size of ciphertext about double that of the plaintext. (Note: can't use "hybrid encryption" if homomorphic property is to be preserved.)
- Does such a family of encryption schemes exist?

Damgård–Jurik Scheme

Damgård-Jurik Scheme

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

Damgård-Jurik Scheme

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

Damgård-Jurik Scheme

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

Damgård-Jurik Scheme

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

Damgård-Jurik Scheme

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

Damgård-Jurik Scheme

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

Damgård-Jurik Scheme

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

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

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

Damgård-Jurik Scheme

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

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

in $\mathbb{Z}_{n^s}$

Damgård-Jurik Scheme

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

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

in $\mathbb{Z}_{n^s}$

- Recursive encryption: Output (ciphertext) of ψ_s ($\mathbb{Z}_{n^{(s+1)}}^*$) is an input (plaintext) for ψ_{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

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

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

  - $\psi_s(m,r).\psi_s(m',r') = \psi_s(m+m',r.r')$ in $\mathbb{Z}_{n^s}$
- Recursive encryption: Output (ciphertext) of ψ_s ($\mathbb{Z}_{n^{(s+1)}}^*$) is an input (plaintext) for ψ_{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.
- IND-CPA secure under "Decisional Composite Residuosity" assumption: Given n=pq (but not p,q), ψ_1(0,rand) looks random (same as Paillier)

Damgård-Jurik Scheme

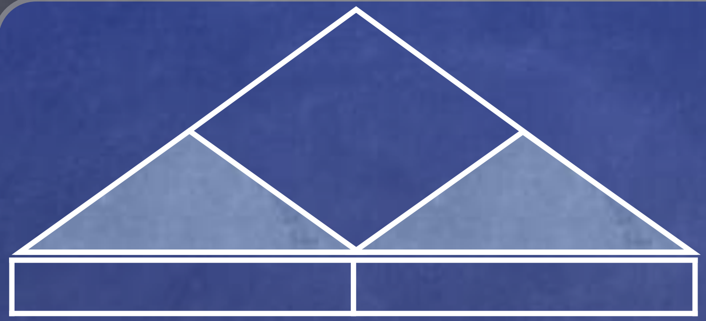
- Uses $\mathbb{Z}_{n^{(s+1)}}^* \simeq \mathbb{Z}_{n^s} \times \mathbb{Z}_n^*$, n=pq, p,q primes within 2x of each other
  - Isomorphism: $\psi_s(a,b) = g^a b^{n\hat{}s}$ where g=(1+n)
- Enc(m) = $\psi_s(m,r)$ for m in $\mathbb{Z}_{n^s}$ and a random r in $\mathbb{Z}_n^*$
- ψ_s can still be efficiently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

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

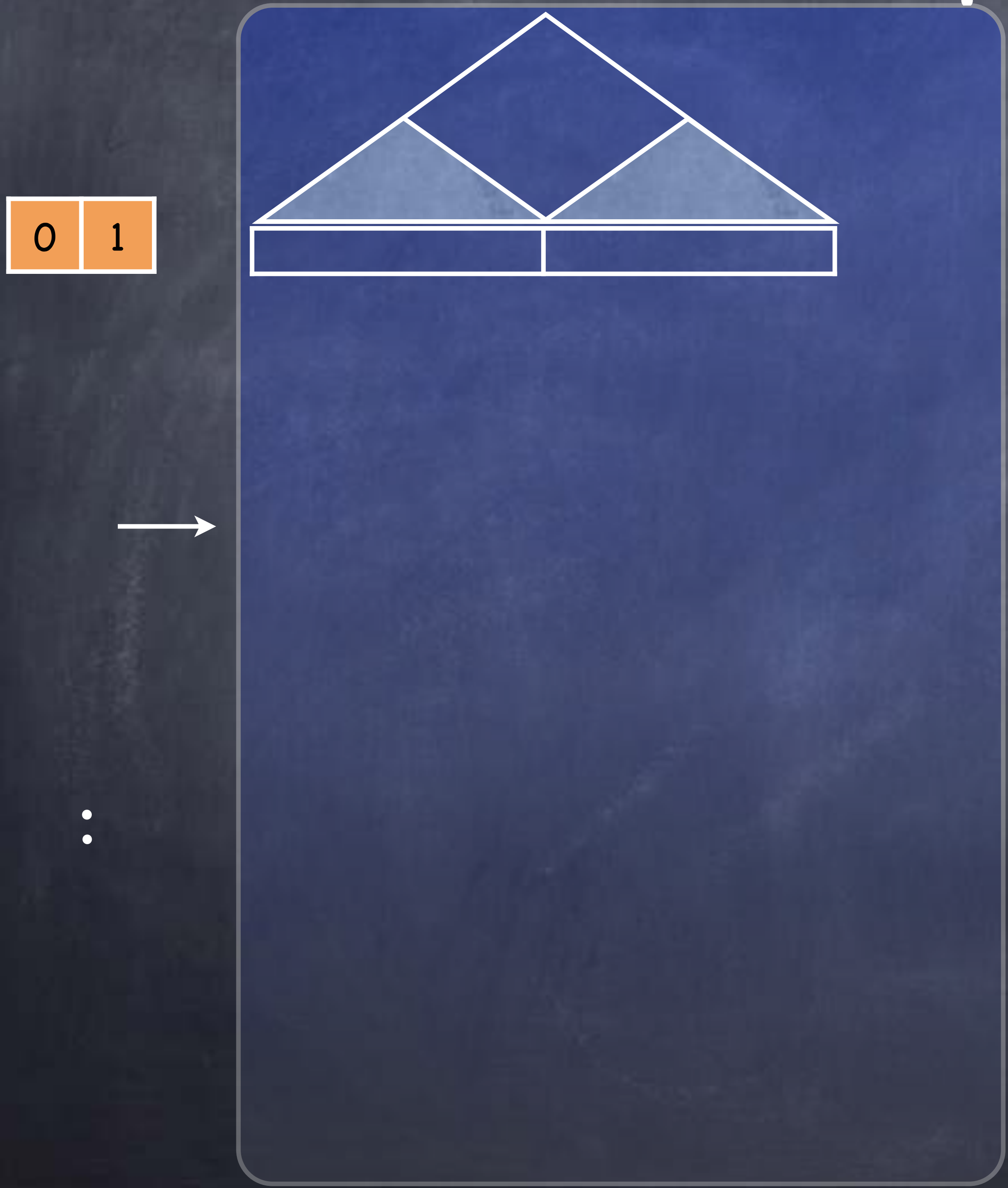
- Recursive encryption: Output (ciphertext) of ψ_s ($\mathbb{Z}_{n^{(s+1)}}^*$) is an input (plaintext) for ψ_{s+1} ($\mathbb{Z}_{n^{(s+1)}}$) for the same public-key n.

  Note: s log n bits encrypted to (s+1)log n bits.
- 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)
- Unlinkability: ReRand(c) = c.Enc(0) (using same s in Enc as for c)

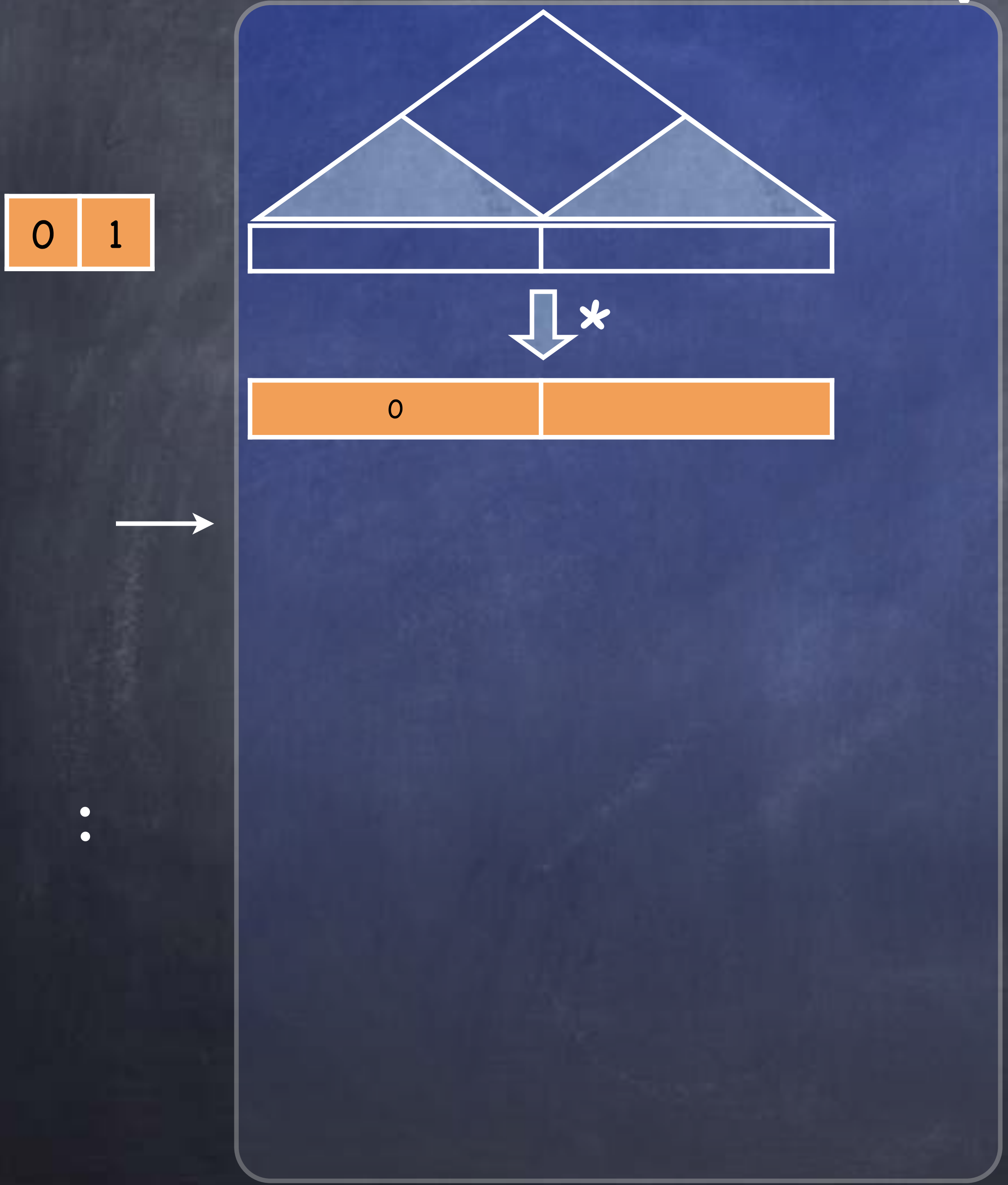
Final PIR protocol



Final PIR protocol

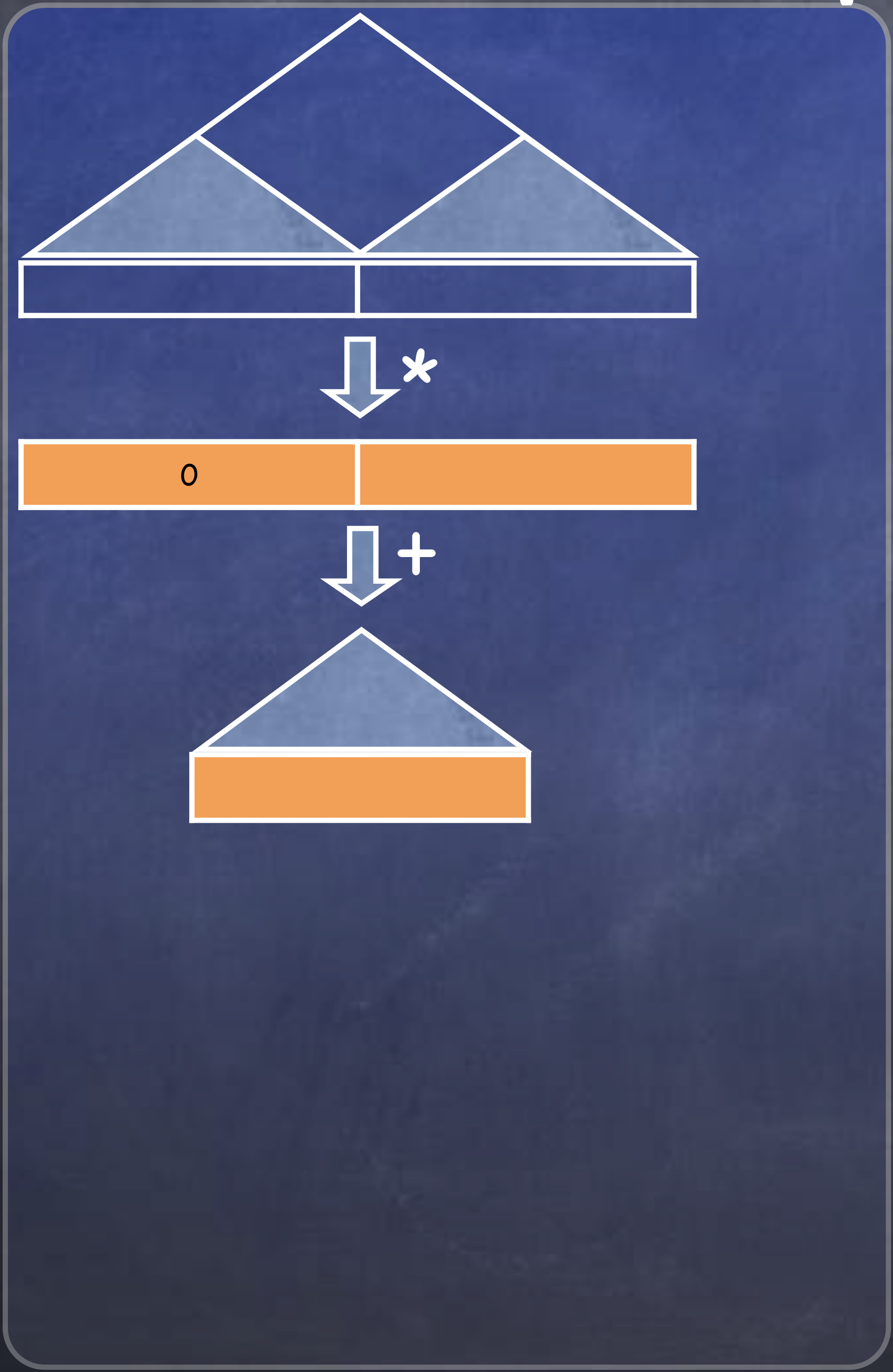


Final PIR protocol



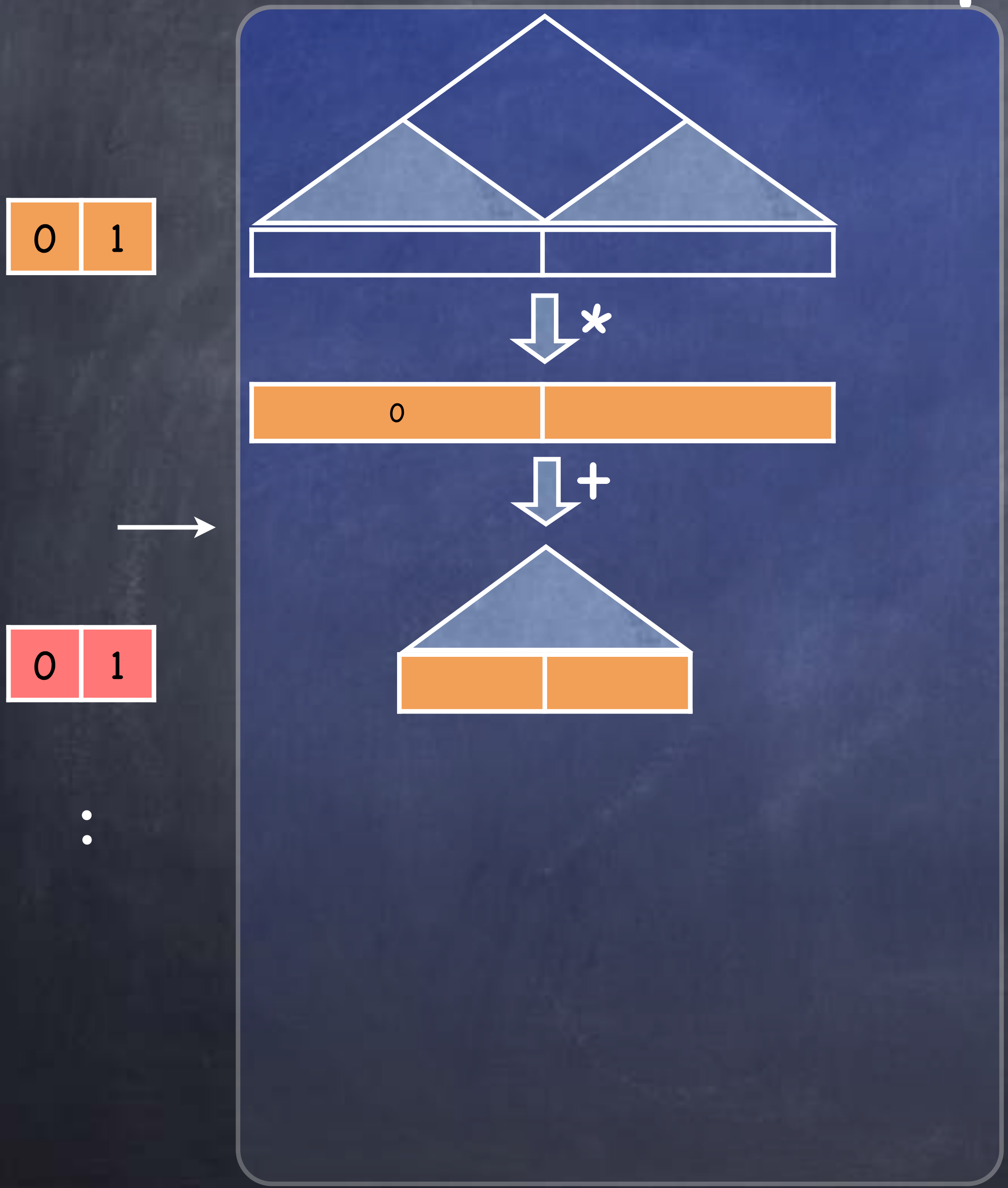
Final PIR protocol

0	1

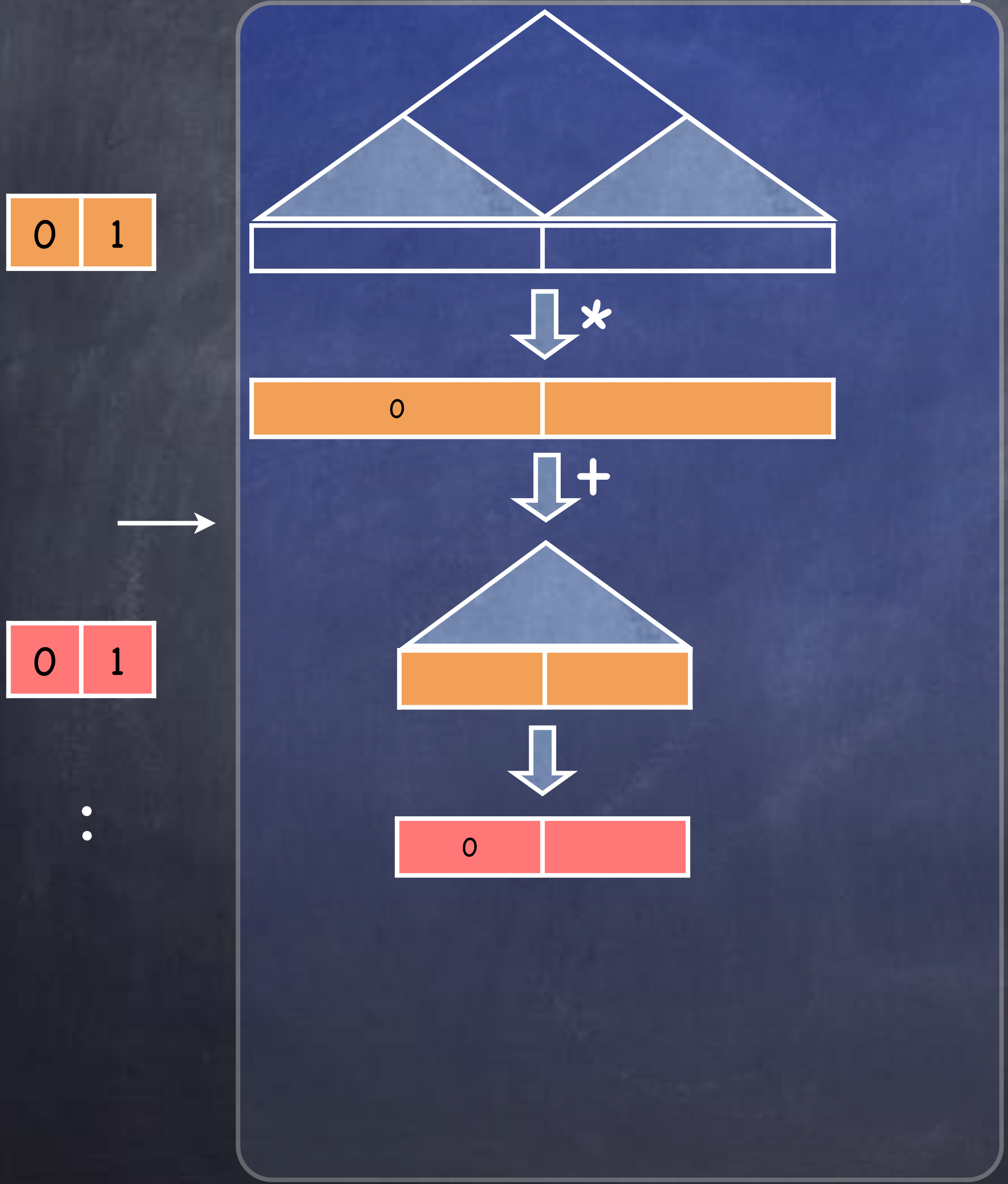


:

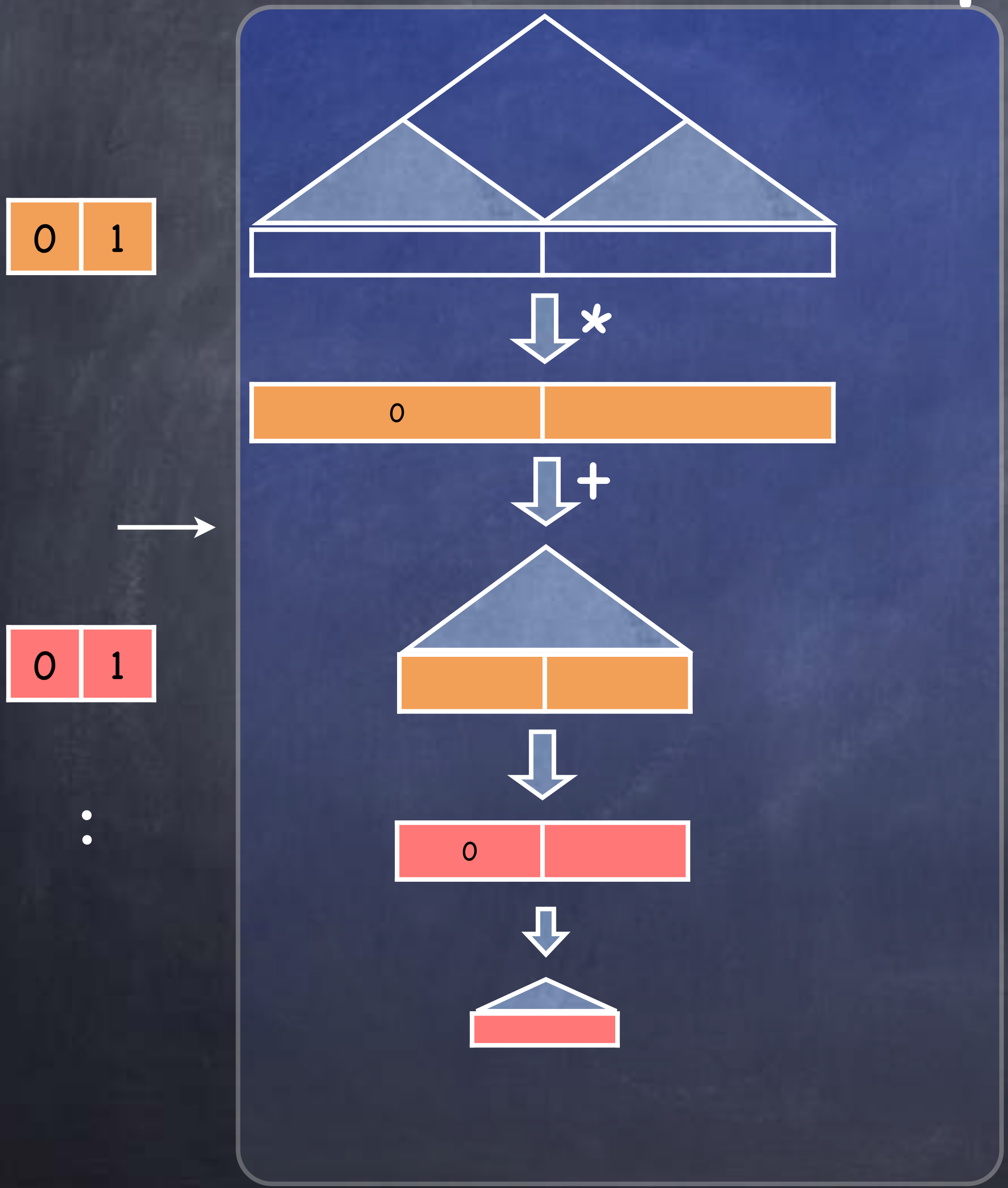
Final PIR protocol



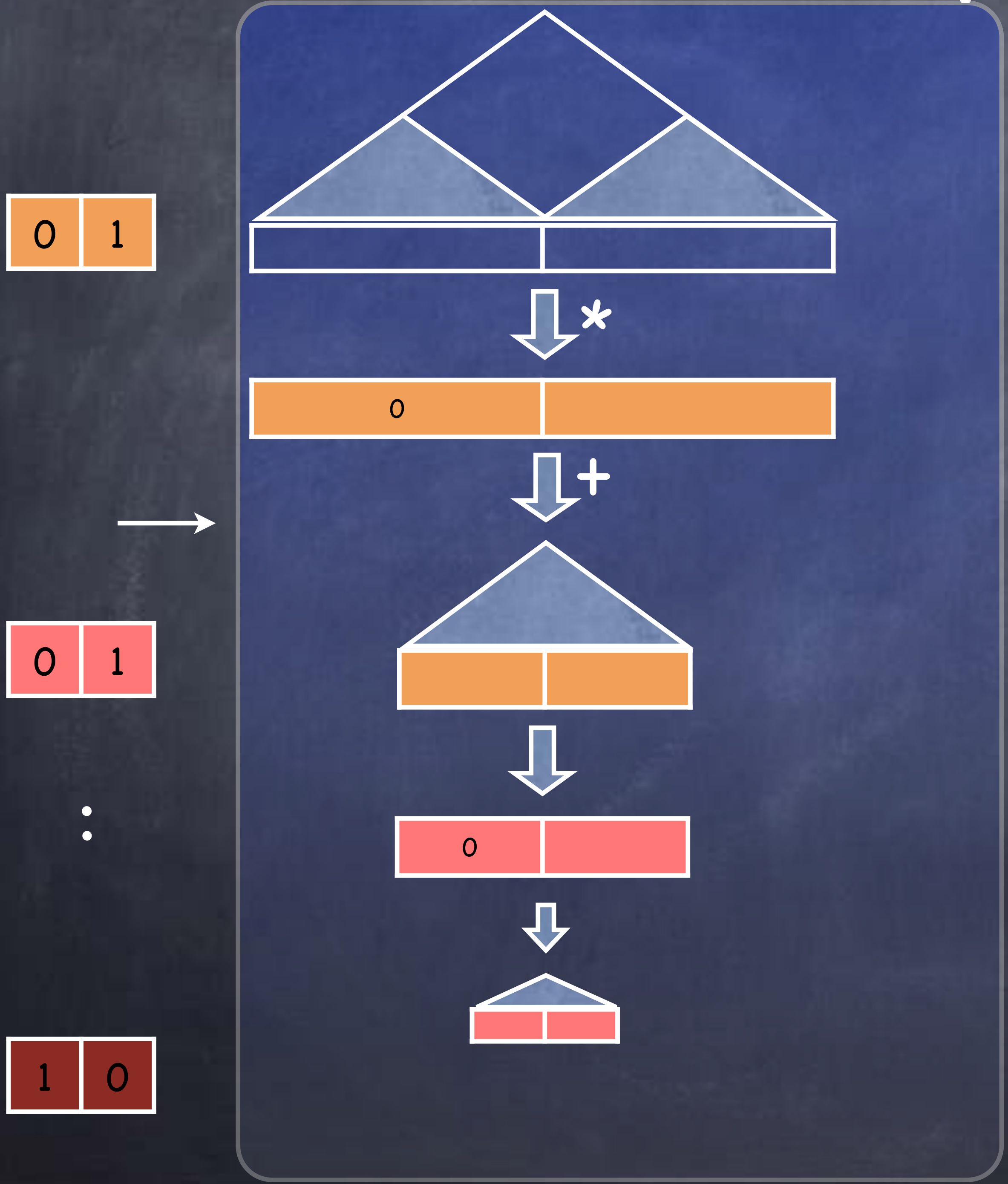
Final PIR protocol



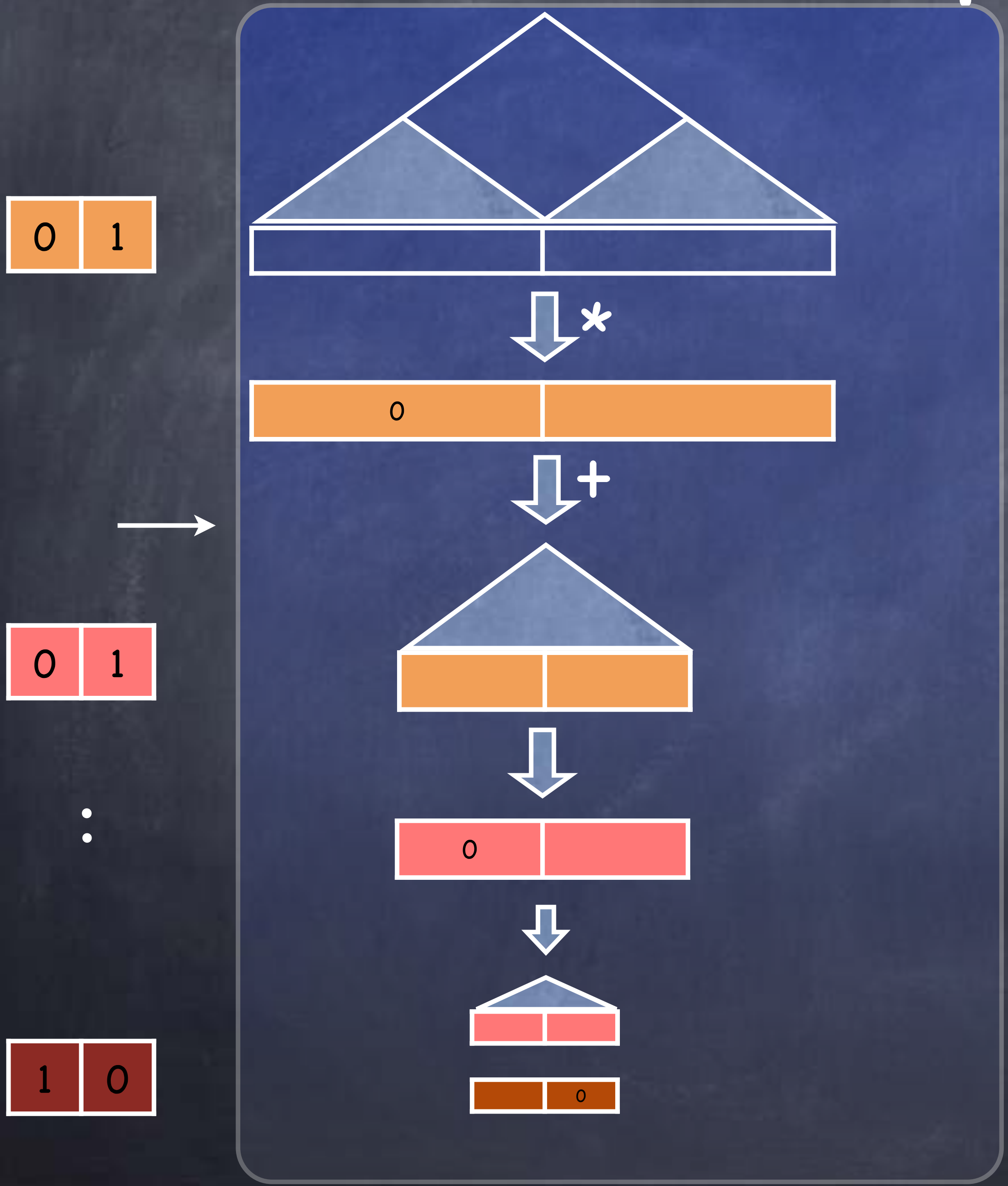
Final PIR protocol



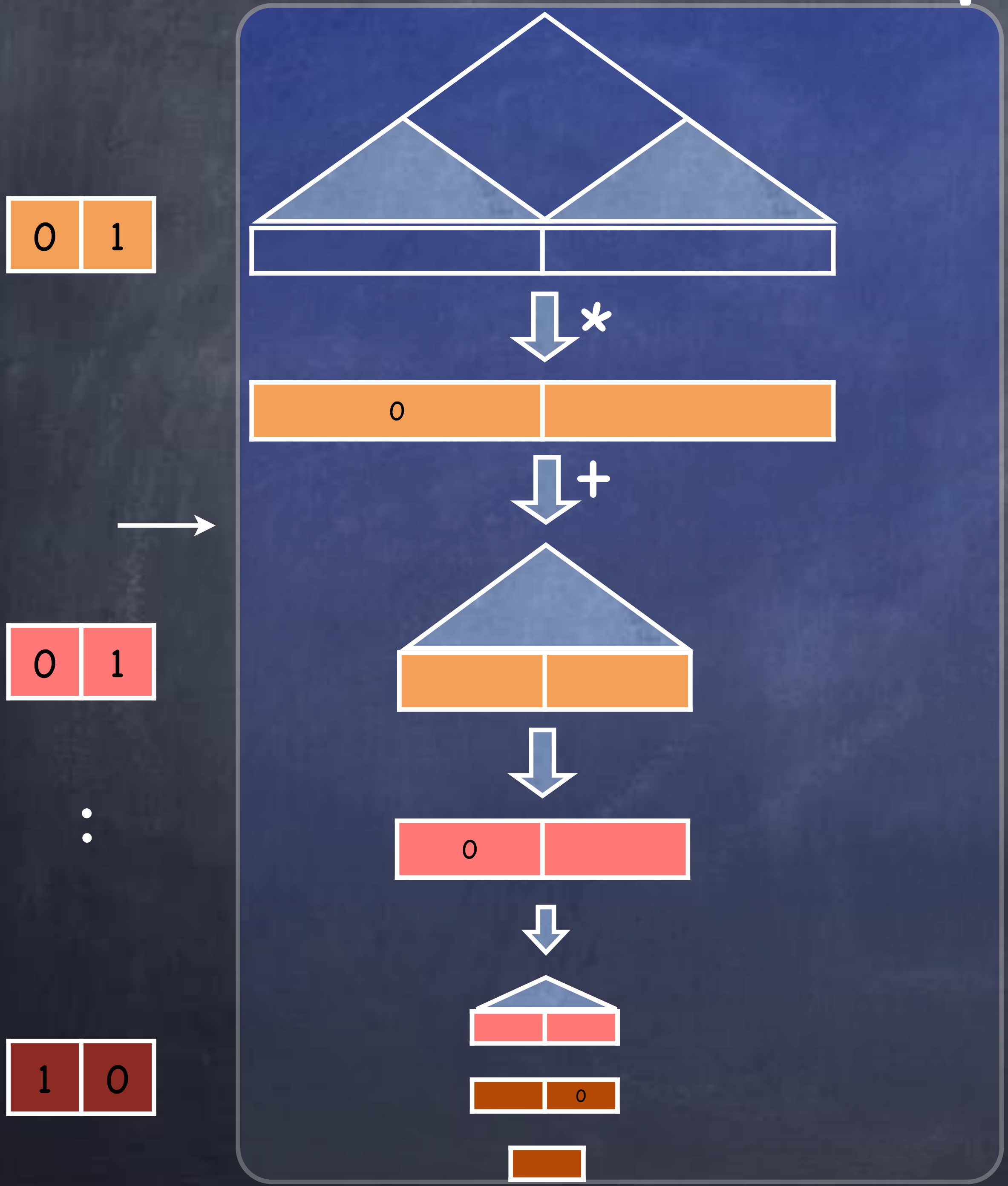
Final PIR protocol



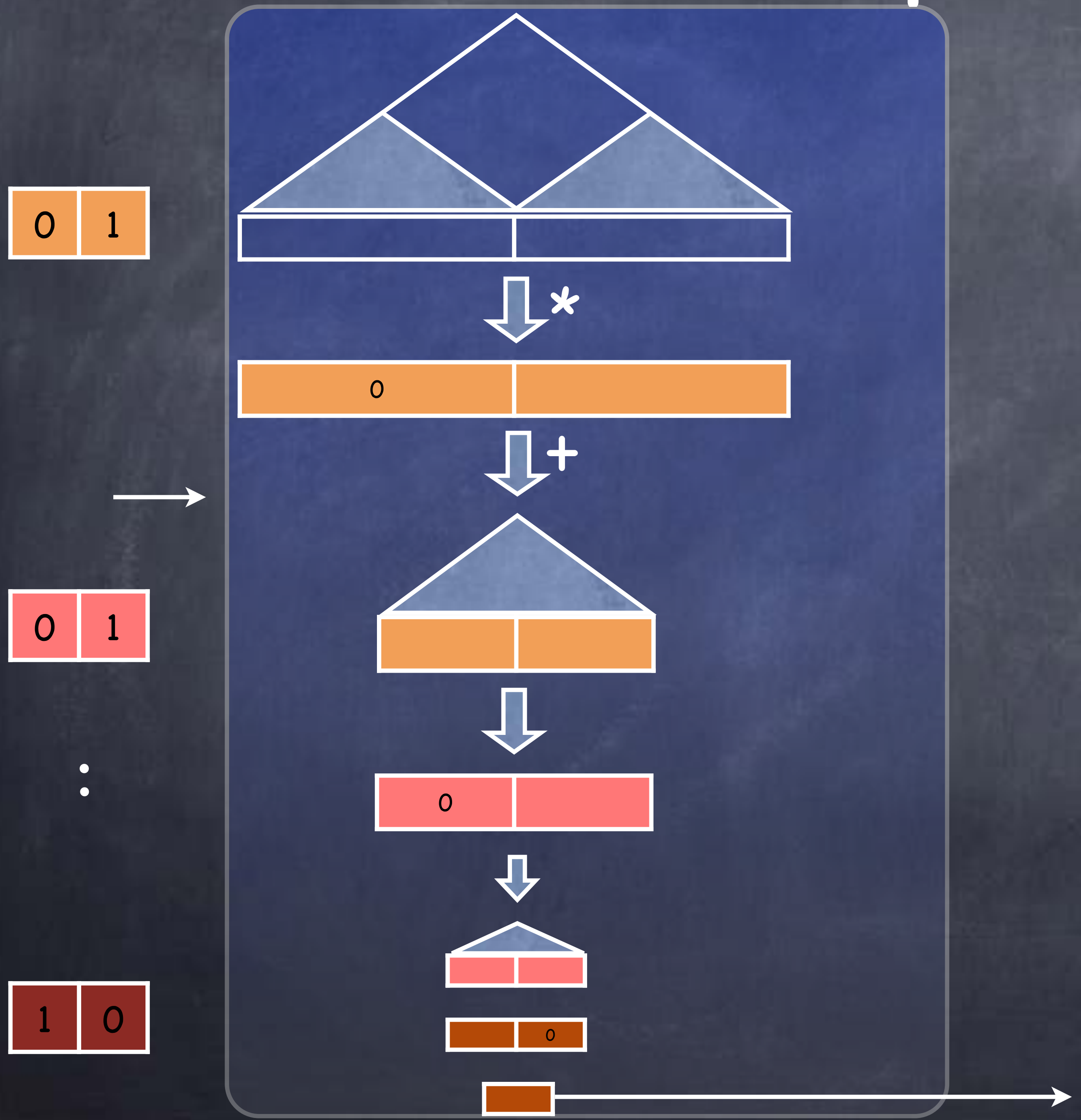
Final PIR protocol



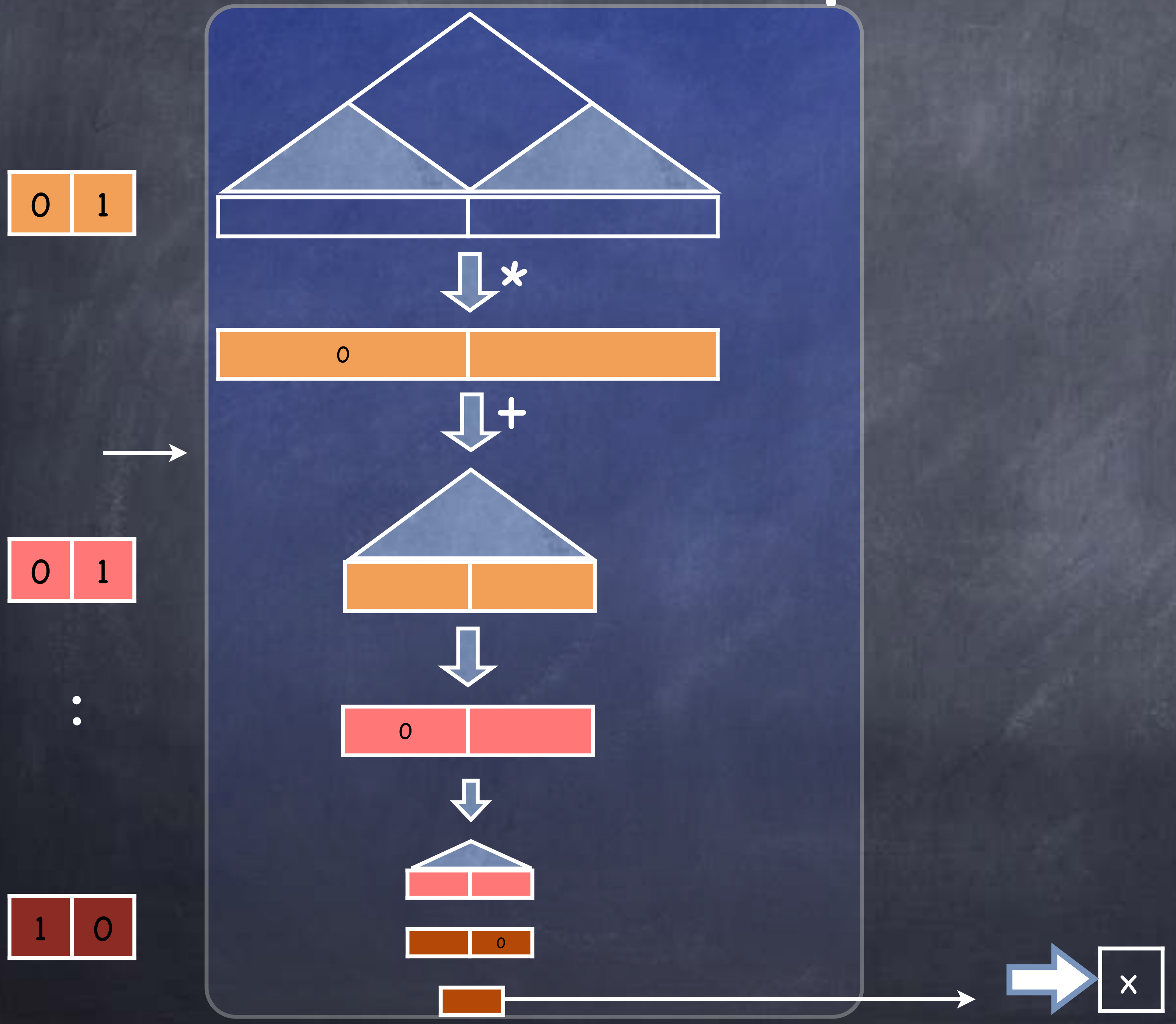
Final PIR protocol



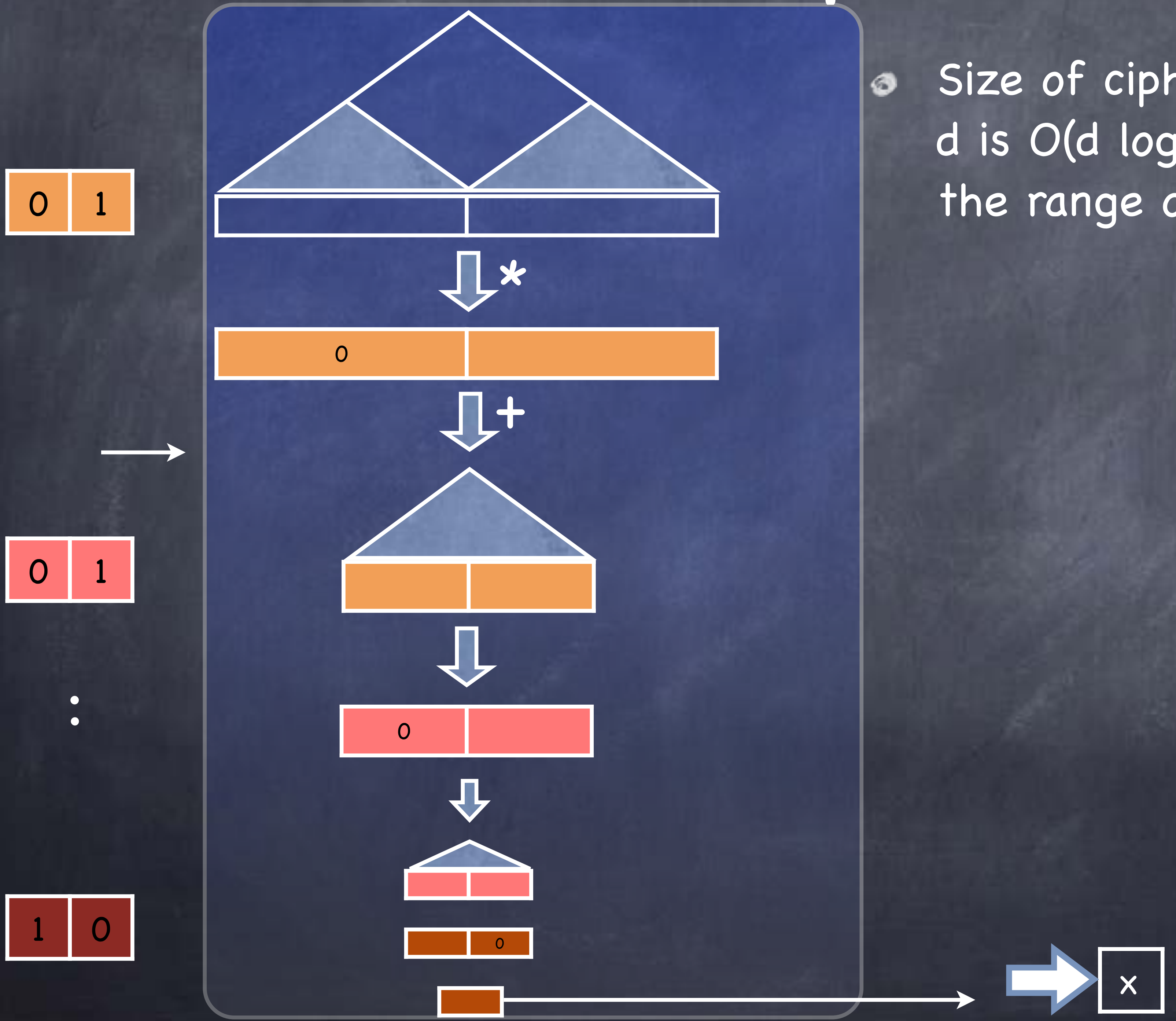
Final PIR protocol



Final PIR protocol

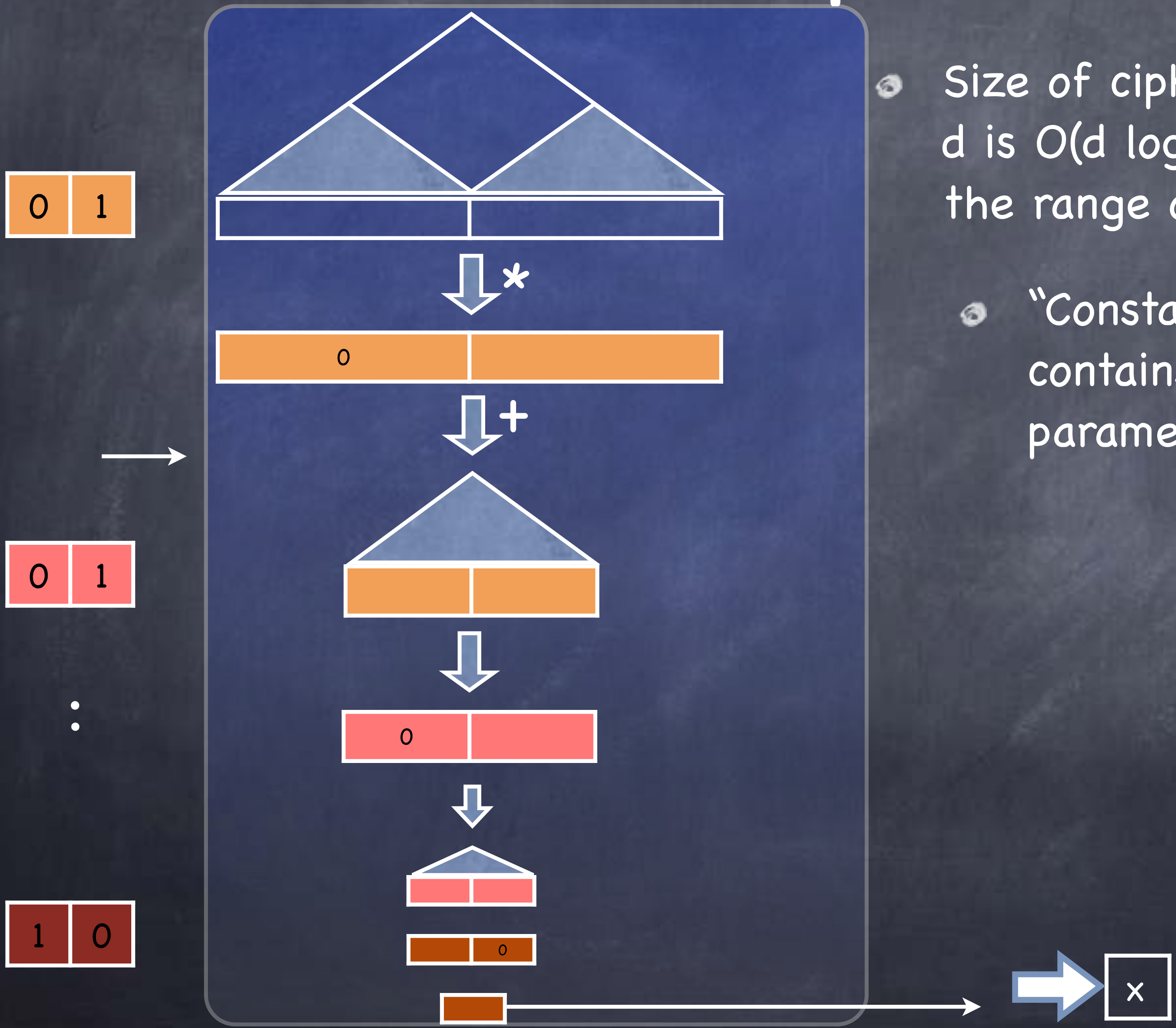


Final PIR protocol



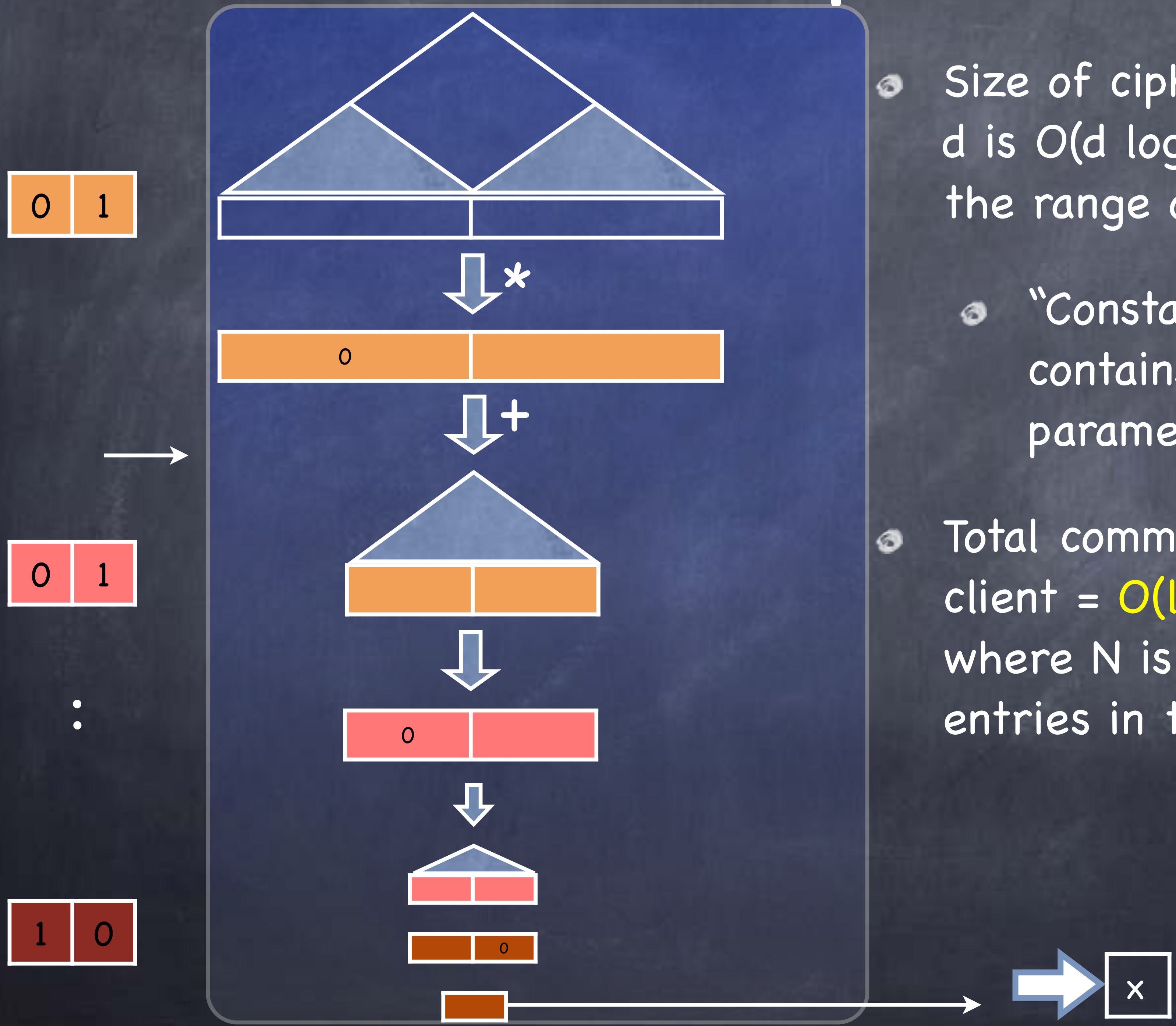
- Size of ciphertext at depth d is O(d log m) where m is the range of values in db

Final PIR protocol



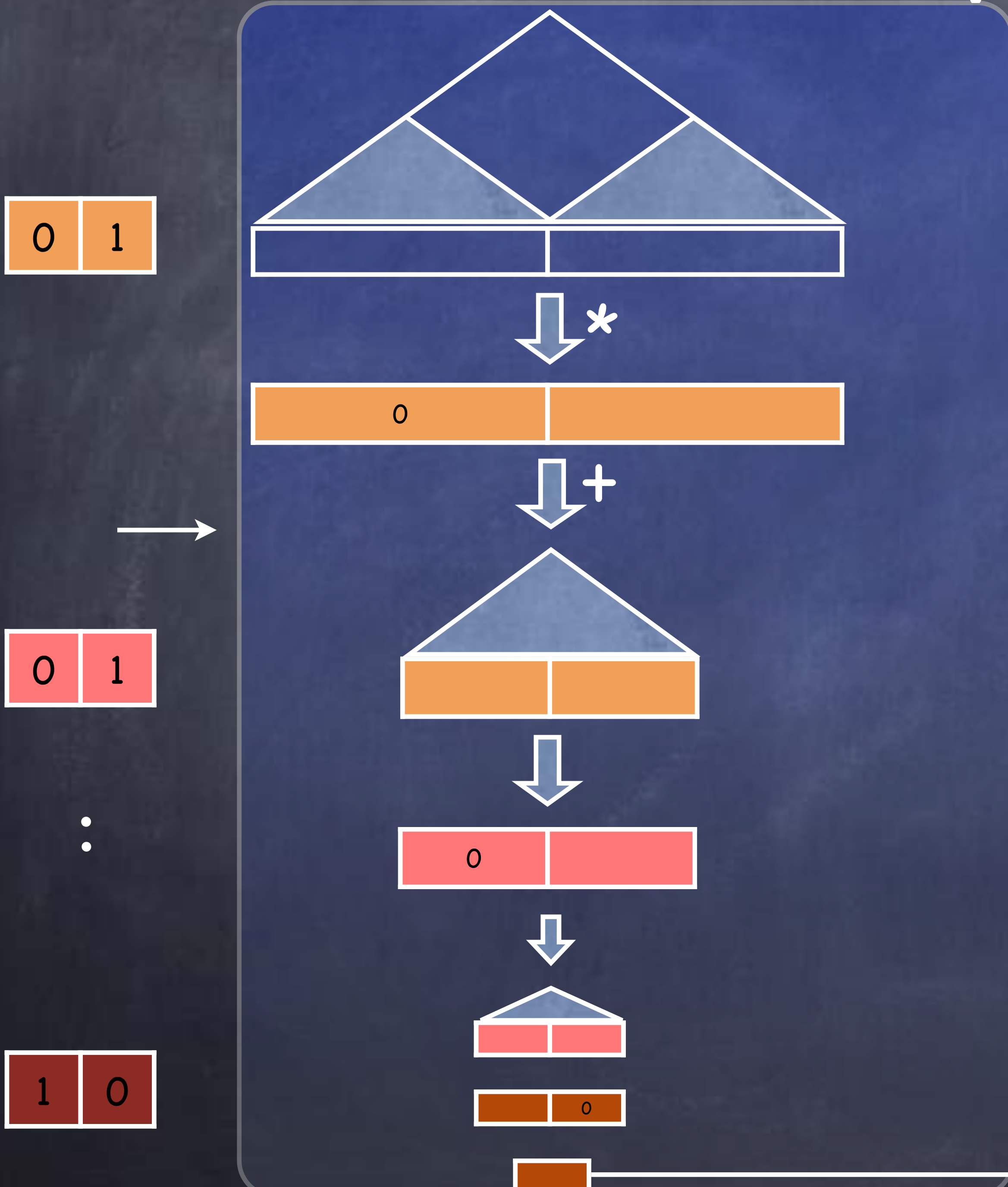
- Size of ciphertext at depth d is O(d log m) where m is the range of values in db
  - "Constant" in O(.) contains security parameter

Final PIR protocol



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

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, and the key is kept shared

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, and the key is kept shared
  - Will evaluate each wire using 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, and the key is kept shared
    - Will evaluate each wire using 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, and the key is kept shared
    - Will evaluate each wire using 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, and the key is kept shared
  - Will evaluate each wire using 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, and the key is kept shared
  - Will evaluate each wire using 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

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

- 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 $[t]=[r + x + \Sigma_{i=2:m} (-s_i)]$. P_1 sets $s_1 = t-r$

Homomorphic Encryption for MPC

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

- 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 $[t]=[r + x + \Sigma_{i=2:m} (-s_i)]$. P_1 sets $s_1 = t-r$
  - Each party publishes $s_i*[y] = [s_i\ y]$; they compute $[\Sigma s_i y]=[xy]$

Homomorphic Encryption for MPC

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

- 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 $[t]=[r + x + \Sigma_{i=2:m} (-s_i)]$. P_1 sets $s_1 = t-r$
  - Each party publishes $s_i*[y] = [s_i\ y]$; they compute $[\Sigma s_i y]=[xy]$
- Threshold decrypt the output

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
    - But 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
    - But not unlinkable: 9+3 and 2 look different
  - Also suppose 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
    - But not unlinkable: 9+3 and 2 look different
  - Also suppose 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