Laconic Function Evaluation, Functional Encryption and Obfuscation for RAMs with Sublinear Computation

Fangqi Dong

IIIS, Tsinghua University

Zihan Hao

IIIS, Tsinghua University

Ethan Mook

Northeastern University

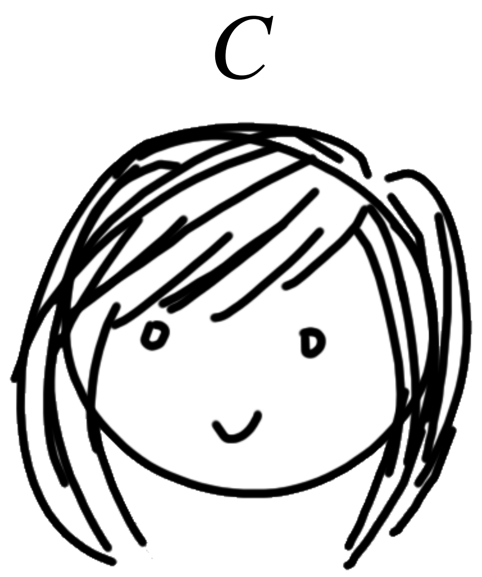
Daniel Wichs

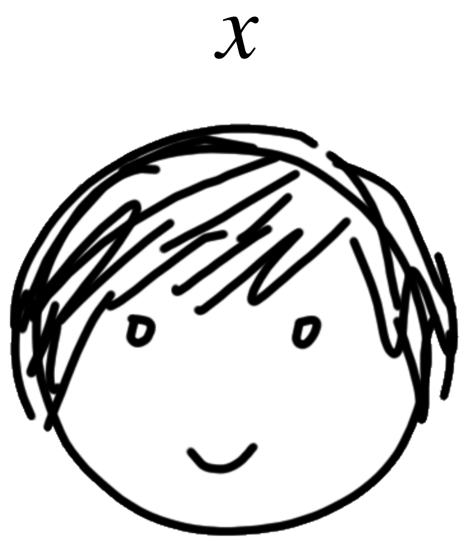
Northeastern University
&
NTT Research

Eurocrypt 2024

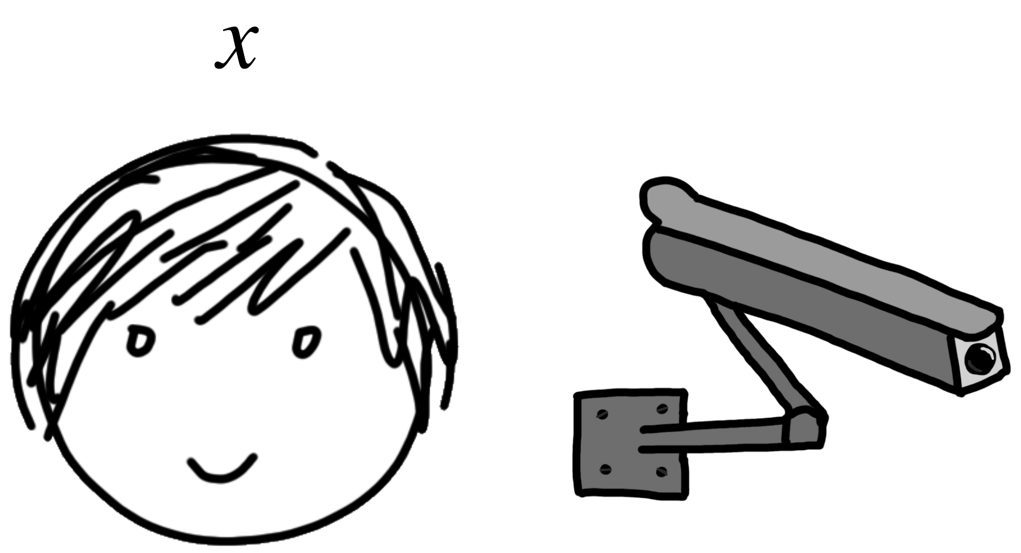
Laconic Function Evaluation (LFE)

Laconic Function Evaluation (LFE)



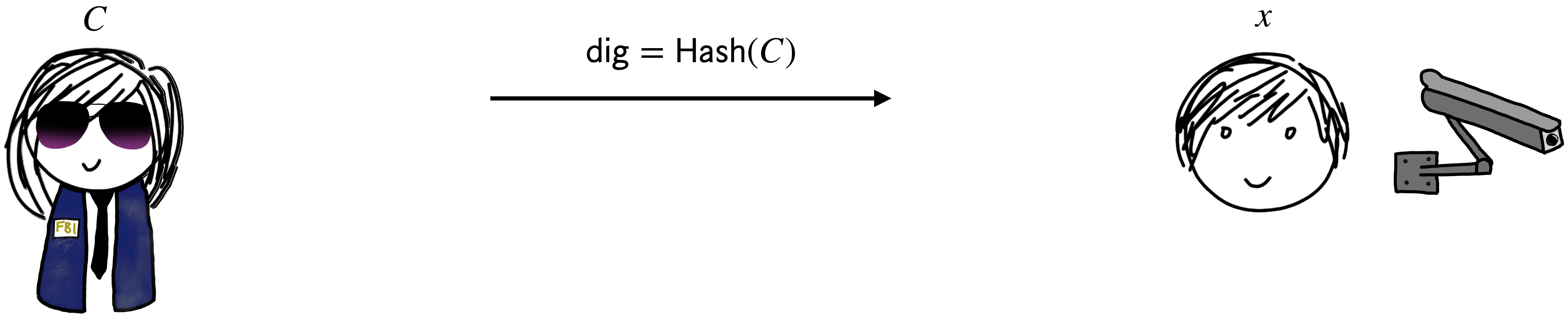


Laconic Function Evaluation (LFE)



Laconic Function Evaluation (LFE)

* in CRS model, CRS hidden



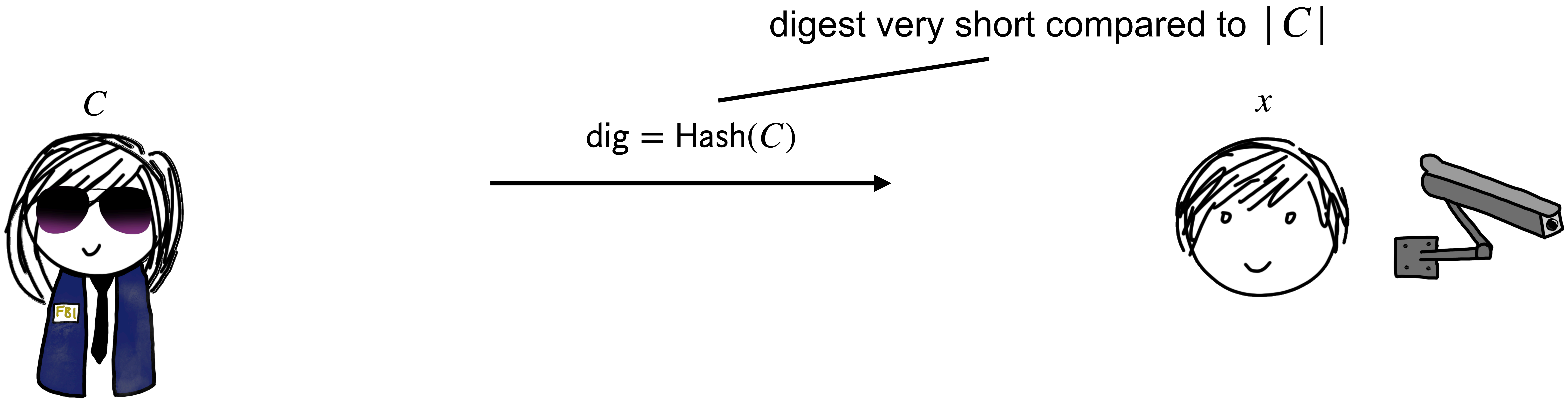
C

$\mathsf{dig} = \mathsf{Hash}(C)$

x

Laconic Function Evaluation (LFE)

* in CRS model, CRS hidden



digest very short compared to $|C|$

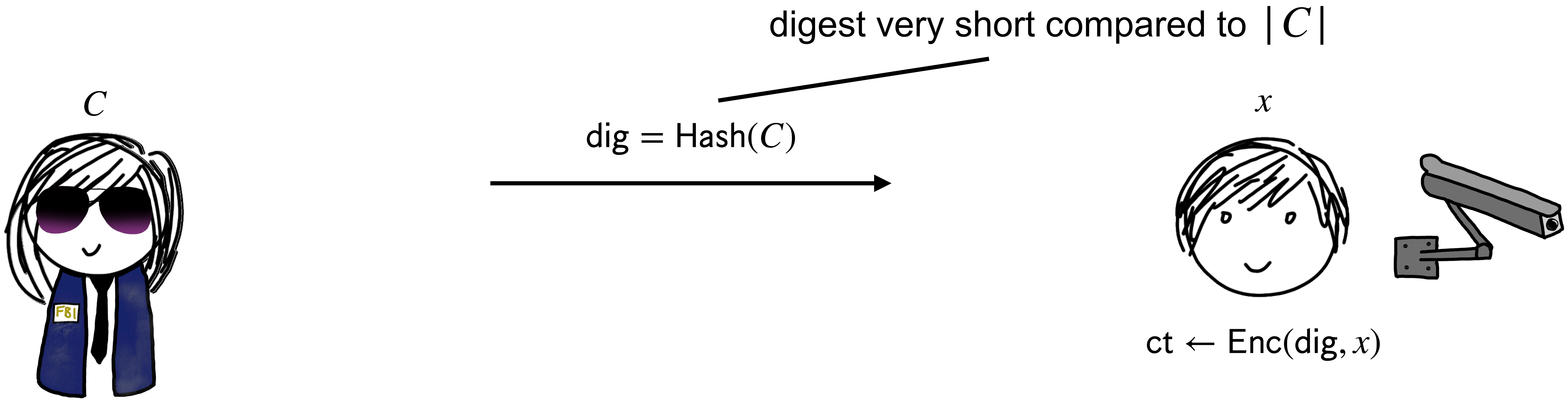
C

$\mathsf{dig} = \mathsf{Hash}(C)$

x

Laconic Function Evaluation (LFE)

* in CRS model, CRS hidden



digest very short compared to $|C|$

C

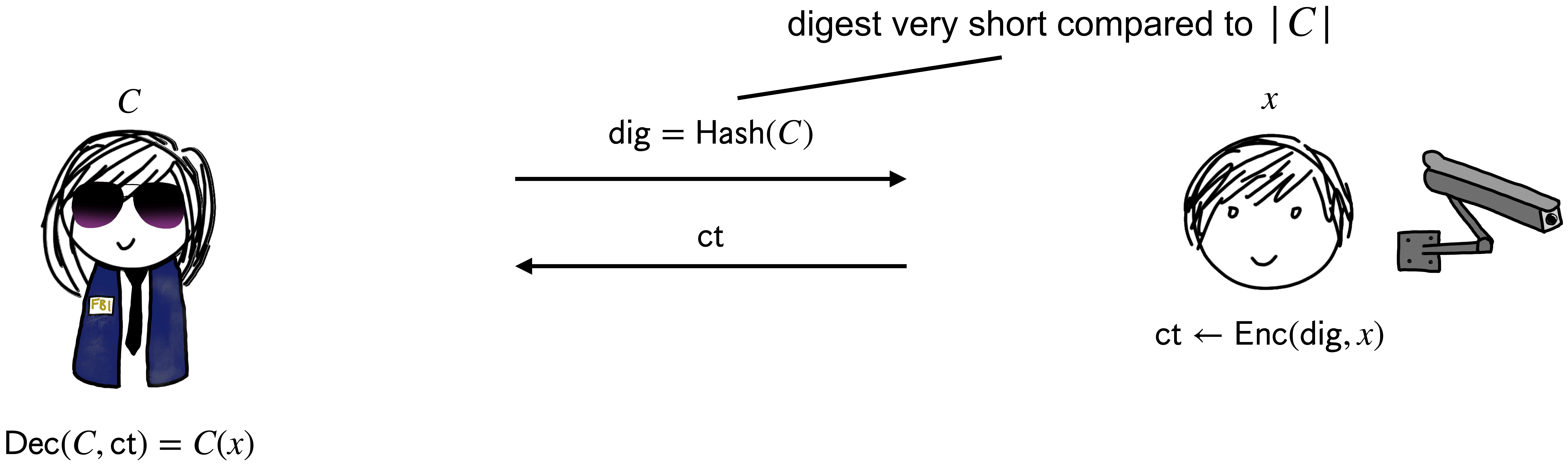
$\text{dig} = \text{Hash}(C)$

x

$\text{ct} \leftarrow \text{Enc}(\text{dig}, x)$

Laconic Function Evaluation (LFE)

* in CRS model, CRS hidden



digest very short compared to $|C|$

C

$\mathsf{dig} = \mathsf{Hash}(C)$

ct

x

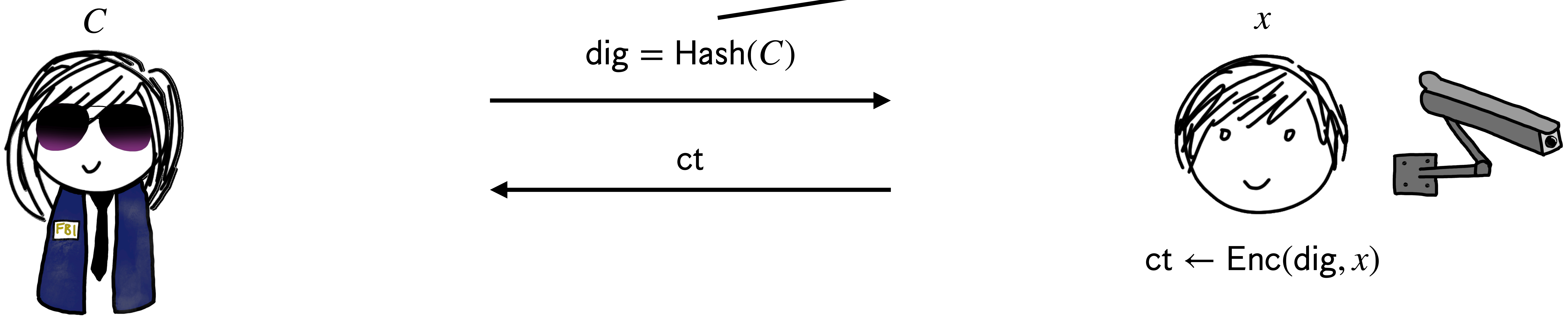
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, x)$

$\mathsf{Dec}(C, \mathsf{ct}) = C(x)$

Laconic Function Evaluation (LFE)

* in CRS model, CRS hidden

digest very short compared to $|C|$



C

$\mathsf{dig} = \mathsf{Hash}(C)$

ct

x

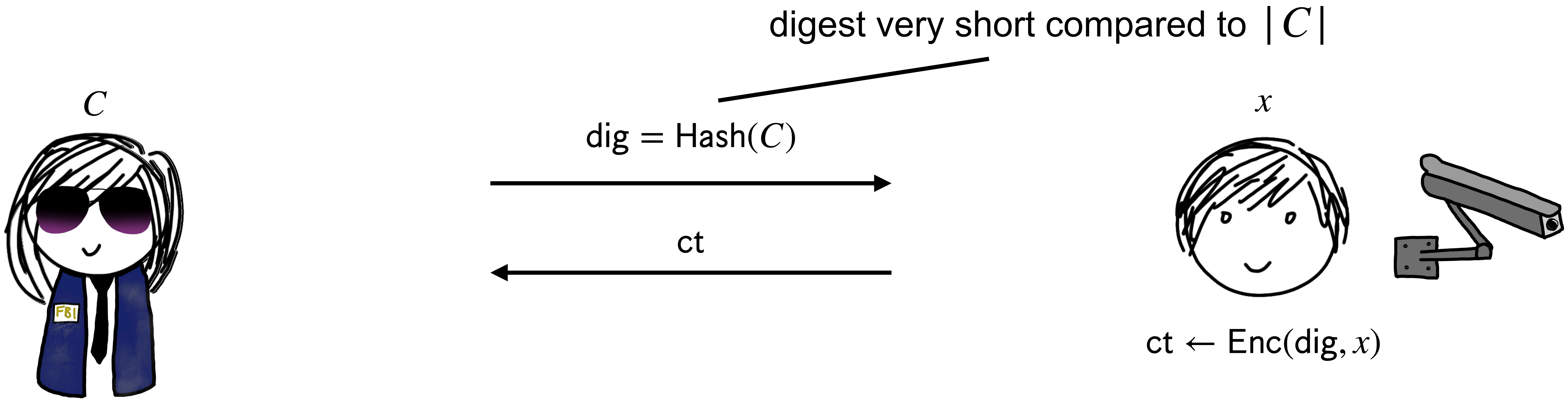
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, x)$

$\mathsf{Dec}(C, \mathsf{ct}) = C(x)$

Security: Server learns nothing more than $C(x)$

Laconic Function Evaluation (LFE)

* in CRS model, CRS hidden



digest very short compared to $|C|$

C

$\mathsf{dig} = \mathsf{Hash}(C)$

ct

x

$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, x)$

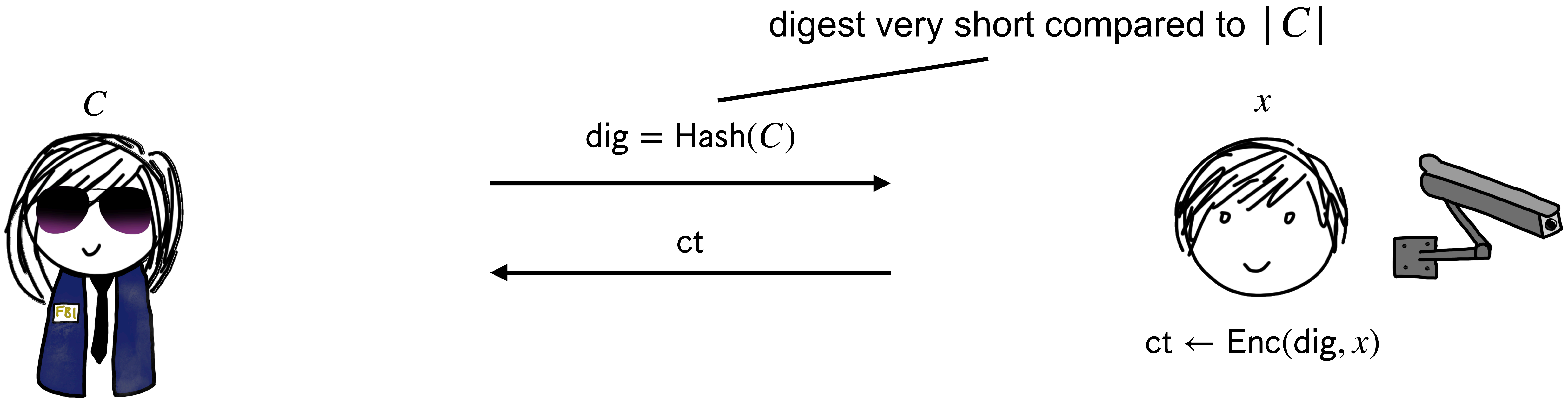
$\mathsf{Dec}(C, \mathsf{ct}) = C(x)$

Security: Server learns nothing more than $C(x)$

Like FHE: 2-round 2PC where Server does the computational work

Laconic Function Evaluation (LFE)

* in CRS model, CRS hidden



digest very short compared to $|C|$

C

$\mathsf{dig} = \mathsf{Hash}(C)$

ct

x

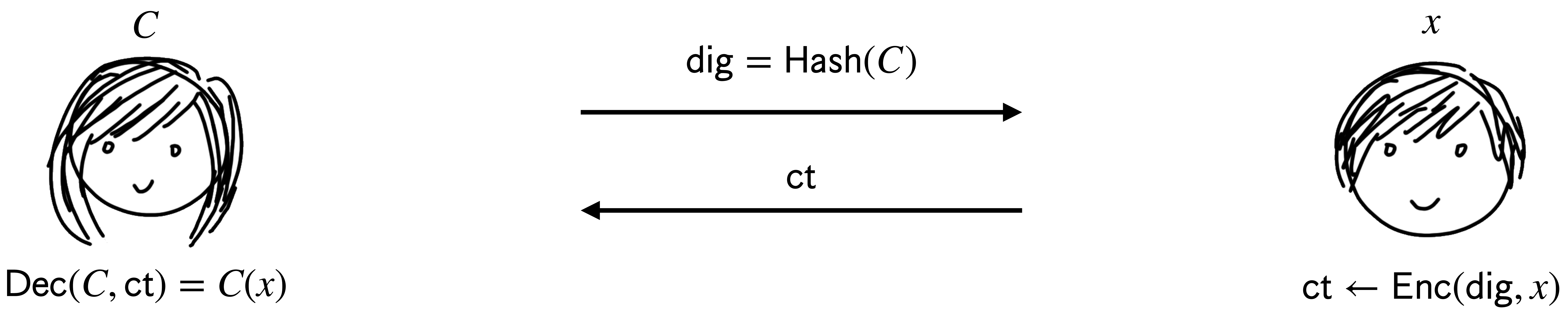
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, x)$

$\mathsf{Dec}(C, \mathsf{ct}) = C(x)$

Security: Server learns nothing more than $C(x)$

Like FHE: 2-round 2PC where Server does the computational work
But "flipped": Server learns the output (instead of Client)

Laconic Function Evaluation (LFE)



C

$\mathsf{dig} = \mathsf{Hash}(C)$

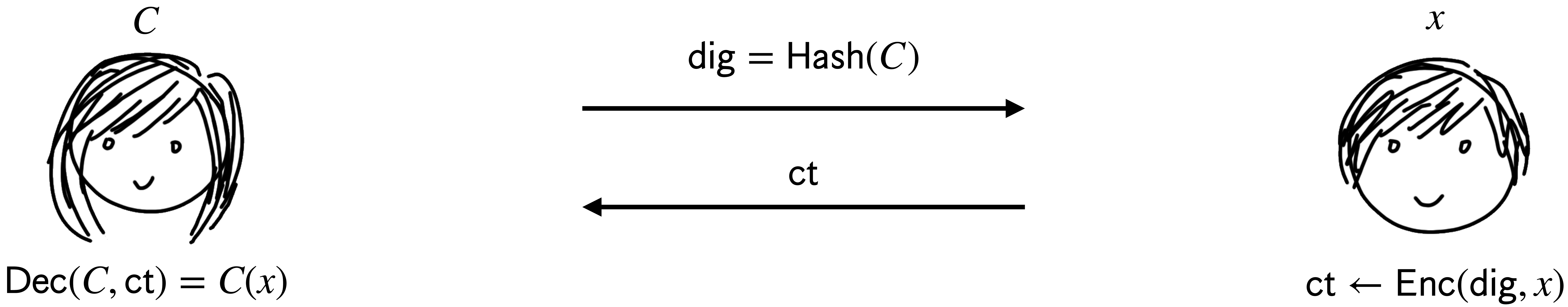
ct

x

$\mathsf{Dec}(C, \mathsf{ct}) = C(x)$

$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, x)$

Laconic Function Evaluation (LFE)



C

$\mathsf{dig} = \mathsf{Hash}(C)$

ct

x

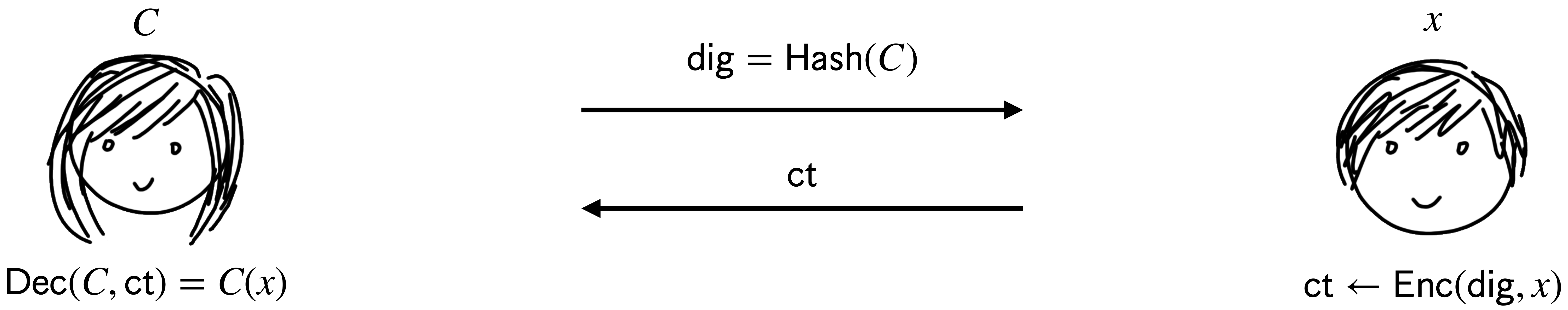
$\mathsf{Dec}(C, \mathsf{ct}) = C(x)$

$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, x)$

Prior work:

- [Quach-Wee-Wichs’17]: LFE for circuits from LWE

Laconic Function Evaluation (LFE)



C

$\mathsf{dig} = \mathsf{Hash}(C)$

ct

x

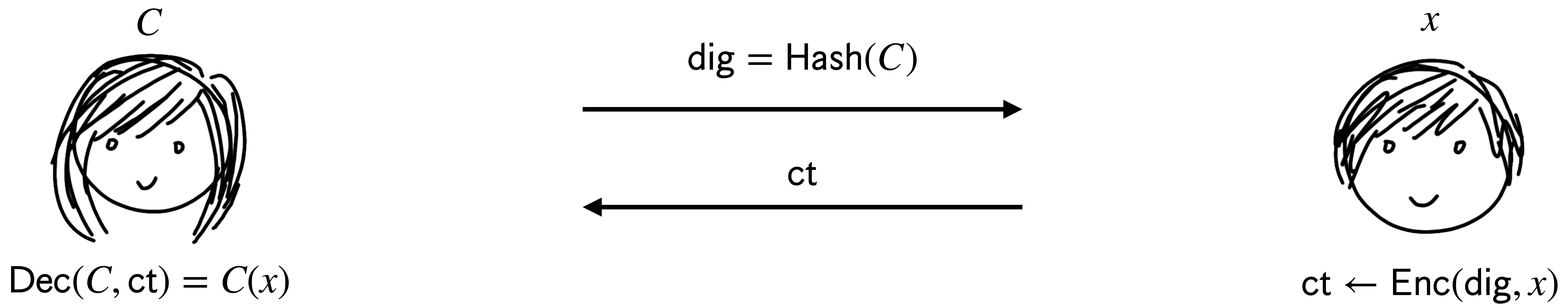
$\mathsf{Dec}(C, \mathsf{ct}) = C(x)$

$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, x)$

Prior work:

- [Quach-Wee-Wichs’17]: LFE for circuits from LWE
- [Döttling-Gajland-Malavolta’23]: LFE for TMs from iO + SSB

Laconic Function Evaluation (LFE)



C

$\mathsf{dig} = \mathsf{Hash}(C)$

ct

x

$\mathsf{Dec}(C, \mathsf{ct}) = C(x)$

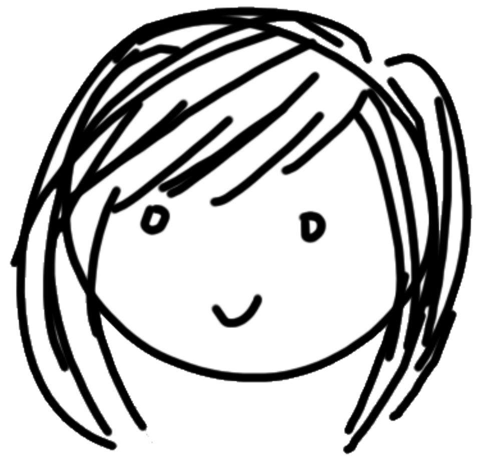
$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, x)$

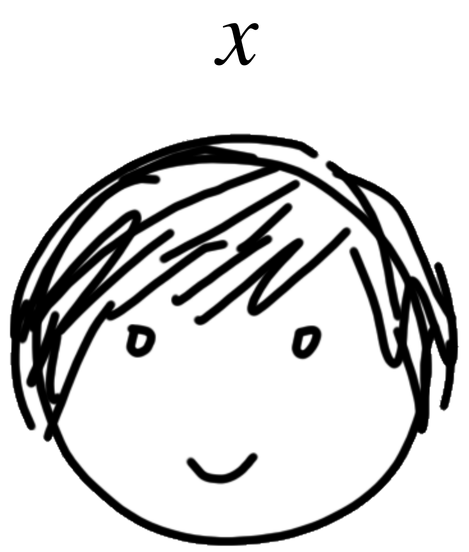
Prior work:

- [Quach-Wee-Wichs’17]: LFE for circuits from LWE
- [Döttling-Gajland-Malavolta’23]: LFE for TMs from iO + SSB

Problem: Server computation is at least linear in inputs!

LFE for RAMs

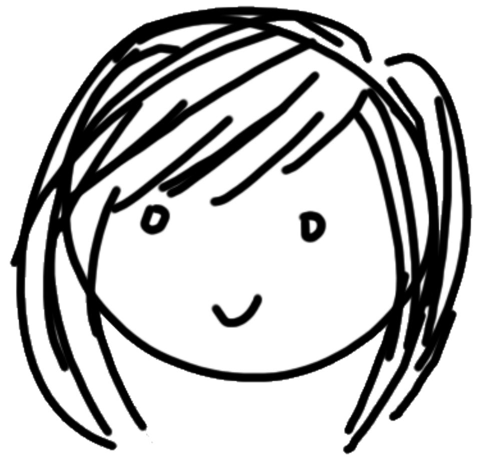


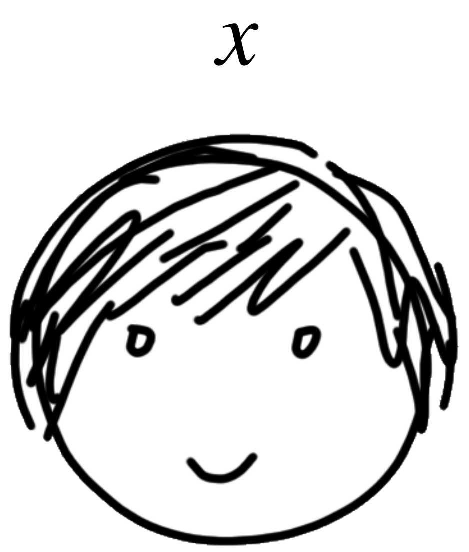


<u>Main Result</u>: We build LFE for RAMs assuming RingLWE

LFE for RAMs

Goal: output RAM computation $P(x, y)$



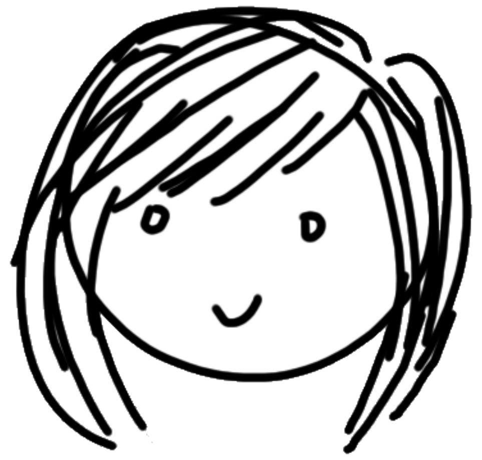


<u>Main Result</u>: We build LFE for RAMs assuming RingLWE

LFE for RAMs

Some fixed RAM program (e.g. universal)

Goal: output RAM computation $P(x, y)$



x

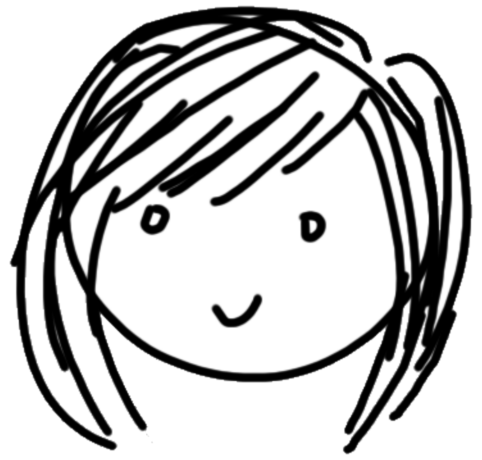
Main Result: We build LFE for RAMs assuming RingLWE

LFE for RAMs

Some fixed RAM program
(e.g. universal)

Goal: output RAM computation $P(x, y)$

$P(x, y)$ has RAM runtime T



x

<u>Main Result</u>: We build LFE for RAMs assuming RingLWE

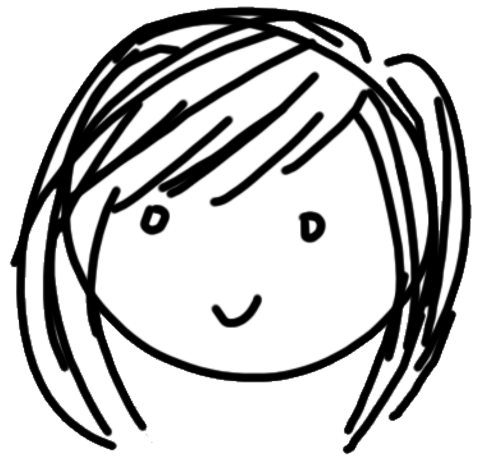
LFE for RAMs

Some fixed RAM program
(e.g. universal)

Goal: output RAM computation $P(x, y)$

$P(x, y)$ has RAM runtime T

$y \xrightarrow{\text{Prep}} \tilde{y}$



x

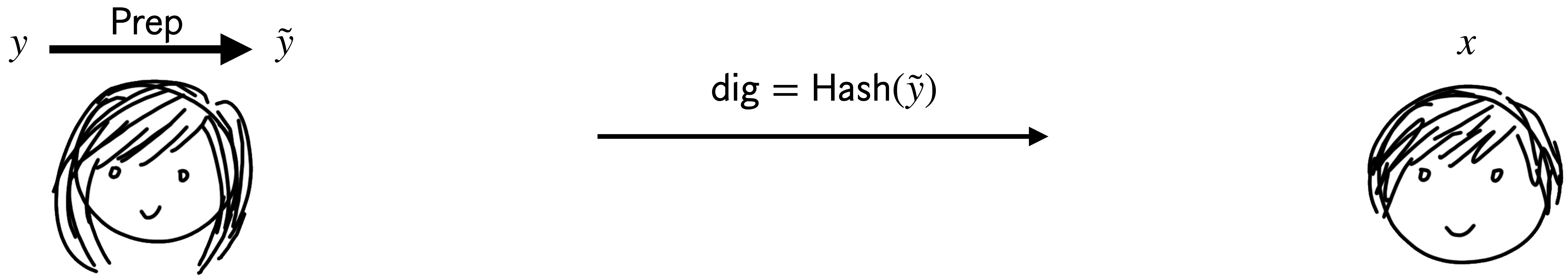
<u>Main Result</u>: We build LFE for RAMs assuming RingLWE

LFE for RAMs

Some fixed RAM program
(e.g. universal)

Goal: output RAM computation $P(x, y)$

$P(x, y)$ has RAM runtime T



$y \xrightarrow{\text{Prep}} \tilde{y}$

$\xrightarrow{\mathsf{dig} = \mathsf{Hash}(\tilde{y})}$

x

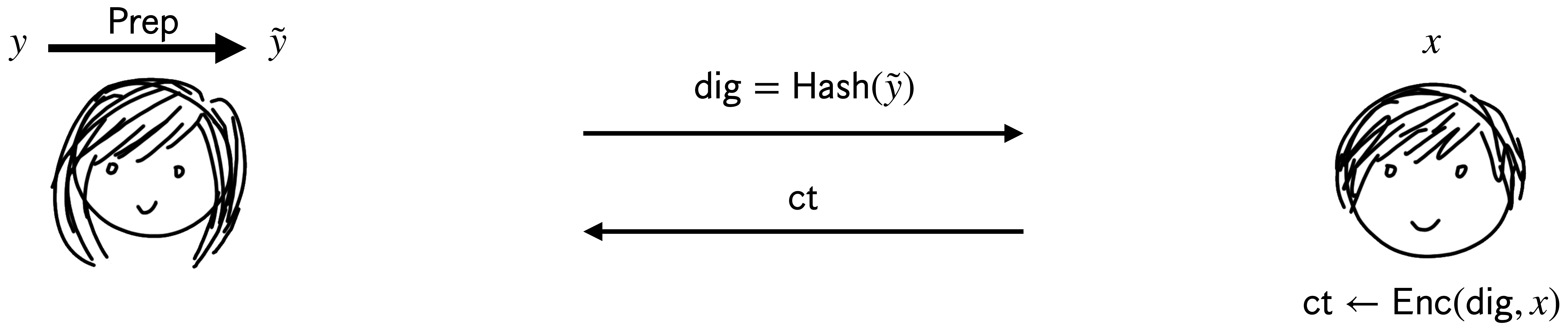
<u>Main Result</u>: We build LFE for RAMs assuming RingLWE

LFE for RAMs

Some fixed RAM program
(e.g. universal)

Goal: output RAM computation $P(x, y)$

$P(x, y)$ has RAM runtime T



$y \xrightarrow{\text{Prep}} \tilde{y}$

x

$\mathsf{dig} = \mathsf{Hash}(\tilde{y})$

ct

$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, x)$

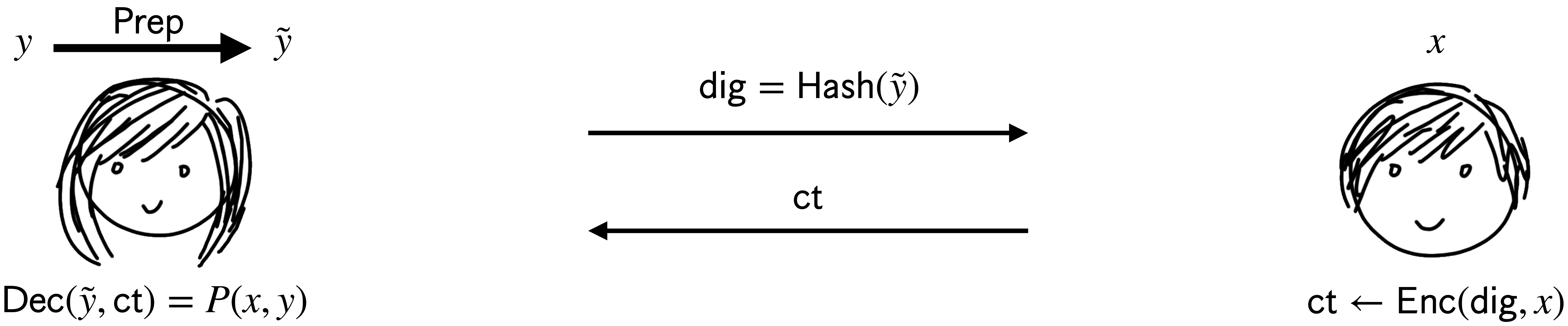
Main Result: We build LFE for RAMs assuming RingLWE

LFE for RAMs

Some fixed RAM program
(e.g. universal)

Goal: output RAM computation $P(x, y)$

$P(x, y)$ has RAM runtime T


$y \xrightarrow{\text{Prep}} \tilde{y}$
x
$\text{dig} = \text{Hash}(\tilde{y})$
ct
$\text{Dec}(\tilde{y}, \text{ct}) = P(x, y)$
$\text{ct} \leftarrow \text{Enc}(\text{dig}, x)$

Main Result: We build LFE for RAMs assuming RingLWE

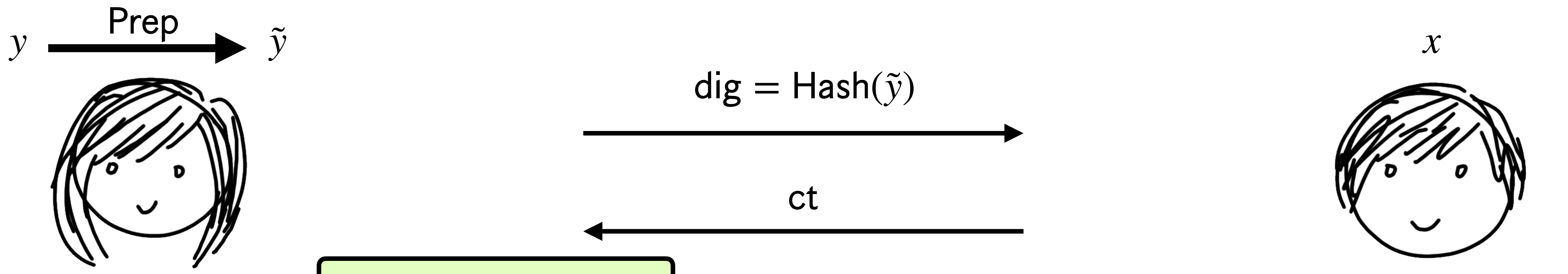
*suppressing poly(λ) and polylog factors

LFE for RAMs

Some fixed RAM program (e.g. universal)

Goal: output RAM computation $P(x, y)$

$P(x, y)$ has RAM runtime T



$y \xrightarrow{\text{Prep}} \tilde{y}$

x

$\text{dig} = \text{Hash}(\tilde{y})$

ct

$\text{Dec}(\tilde{y}, \text{ct}) = P(x, y)$

Dec run time: T

$\text{ct} \leftarrow \text{Enc}(\text{dig}, x)$

Main Result: We build LFE for RAMs assuming RingLWE

LFE for RAMs

*suppressing poly(λ) and polylog factors

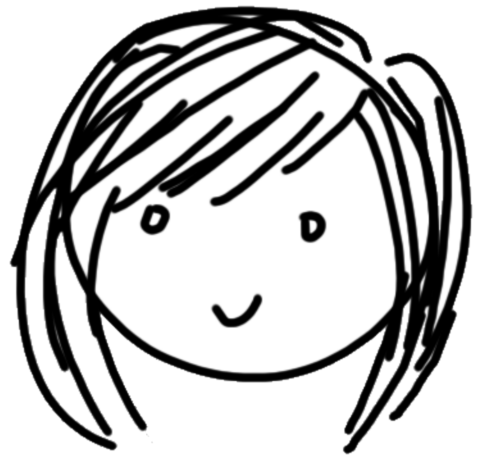
Some fixed RAM program (e.g. universal)

Goal: output RAM computation $P(x, y)$

$P(x, y)$ has RAM runtime T

Prep run time: $|y|^{1+\varepsilon}$

$y \xrightarrow{\text{Prep}} \tilde{y}$



x

$\mathsf{dig} = \mathsf{Hash}(\tilde{y})$ →

← ct

$\mathsf{Dec}(\tilde{y}, \mathsf{ct}) = P(x, y)$

Dec run time: T

$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, x)$

<u>Main Result</u>: We build LFE for RAMs assuming RingLWE

*suppressing poly(λ) and
polylog factors

LFE for RAMs

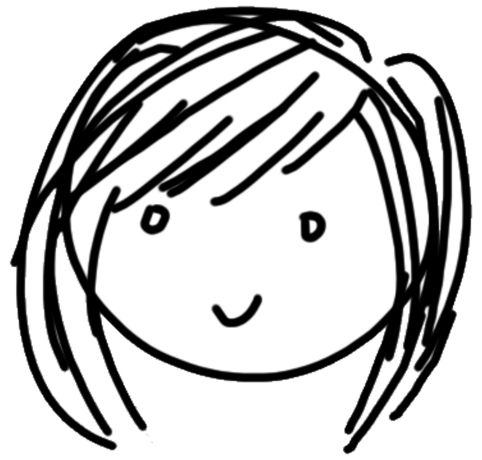
Some fixed RAM program
(e.g. universal)

Goal: output RAM computation $P(x, y)$

$P(x, y)$ has RAM runtime T

Prep run time: $|y|^{1+\varepsilon}$

$y \xrightarrow{\text{Prep}} \tilde{y}$



x

$\xrightarrow{\text{dig} = \text{Hash}(\tilde{y})}$

$\xleftarrow{\text{ct}}$

$\text{Dec}(\tilde{y}, \text{ct}) = P(x, y)$

Dec run time: T

Enc run time: $|x| + T$

$\text{ct} \leftarrow \text{Enc}(\text{dig}, x)$

<u>Main Result</u>: We build LFE for RAMs assuming RingLWE

LFE for RAMs

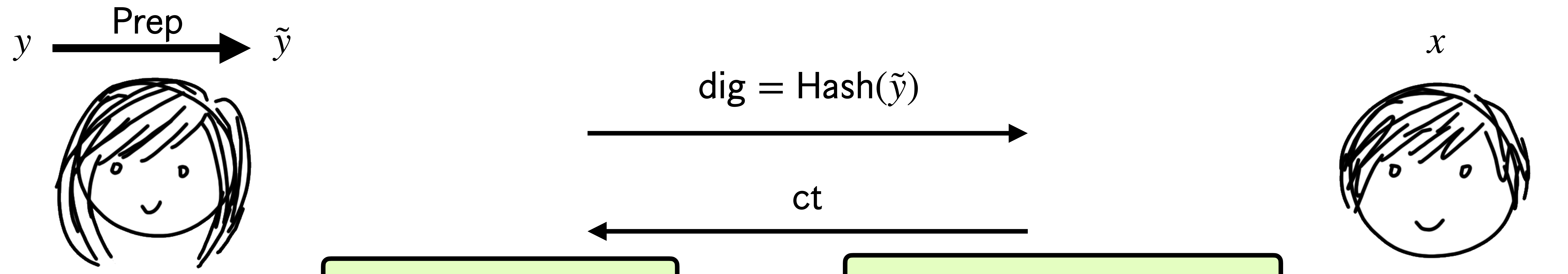
*suppressing poly(λ) and polylog factors

Goal: output RAM computation $P(x, y)$

Some fixed RAM program (e.g. universal)

$P(x, y)$ has RAM runtime T

Prep run time: $|y|^{1+\varepsilon}$


$y \xrightarrow{\text{Prep}} \tilde{y}$
x
dig $=$ Hash($\tilde{y}$)
ct

Dec($\tilde{y}$, ct) $= P(x, y)$

Dec run time: T

Enc run time: $|x| + \cancel{T}$

ct $\leftarrow$ Enc(dig, x)

<u>Main Result</u>: We build LFE for RAMs assuming RingLWE

Additionally assuming iO, get Enc run time just $|x|$

LFE for RAMs

*suppressing poly(λ) and polylog factors

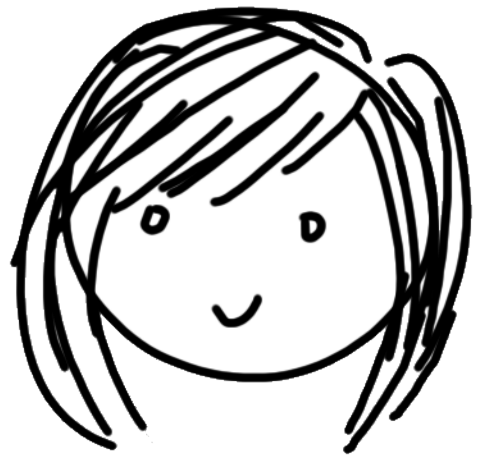
Goal: output RAM computation $P(x, y)$

Some fixed RAM program (e.g. universal)

$P(x, y)$ has RAM runtime T

Prep run time: $|y|^{1+\varepsilon}$

$y \xrightarrow{\text{Prep}} \tilde{y}$



x

dig $=$ Hash$(\tilde{y})$ →

← ct

Dec$(\tilde{y}, \text{ct}) = P(x, y)$

Dec run time: T

Enc run time: $|x| + \cancel{T}$

ct $\leftarrow$ Enc(dig, x)

<u>Main Result</u>: We build LFE for RAMs assuming RingLWE
Additionally assuming iO, get Enc run time just $|x|$

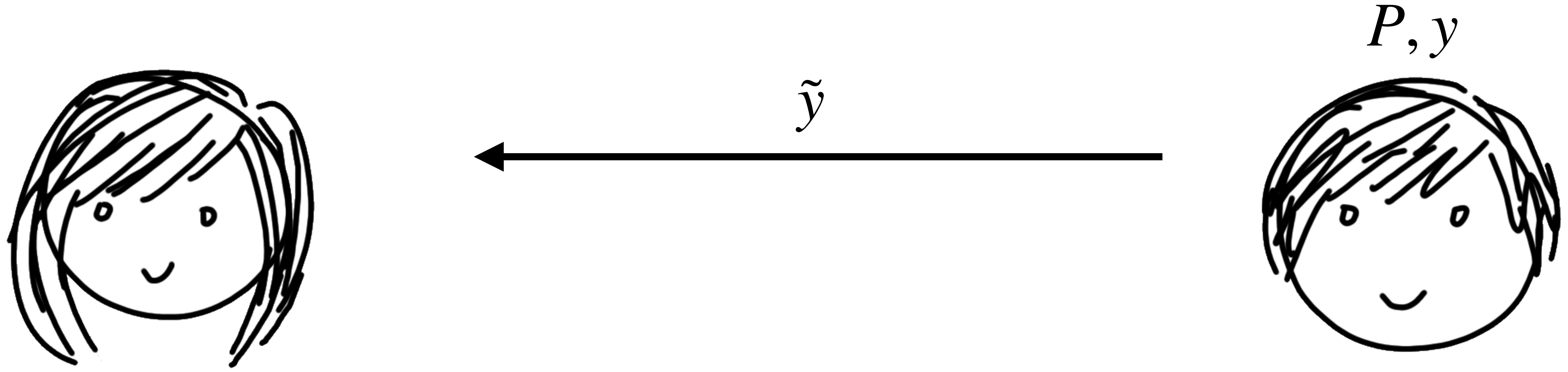
Main challenge: Privately accessing the public database y

RAM-LFE vs Garbled RAM

Garbled RAM

RAM-LFE

RAM-LFE vs Garbled RAM

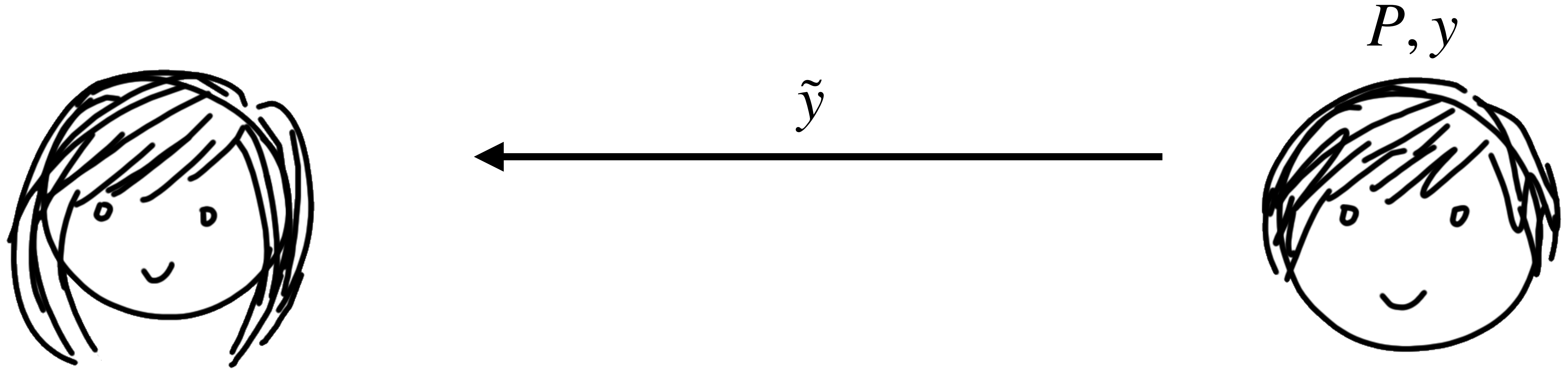


$\tilde{y}, \mathsf{sk} \leftarrow \mathsf{GarbleDB}(y)$

Garbled RAM

RAM-LFE

RAM-LFE vs Garbled RAM



P, y

$\tilde{y}$

$\tilde{y}, \mathsf{sk} \leftarrow \mathsf{GarbleDB}(y)$

y belongs to client and is garbled with respect to their secret key

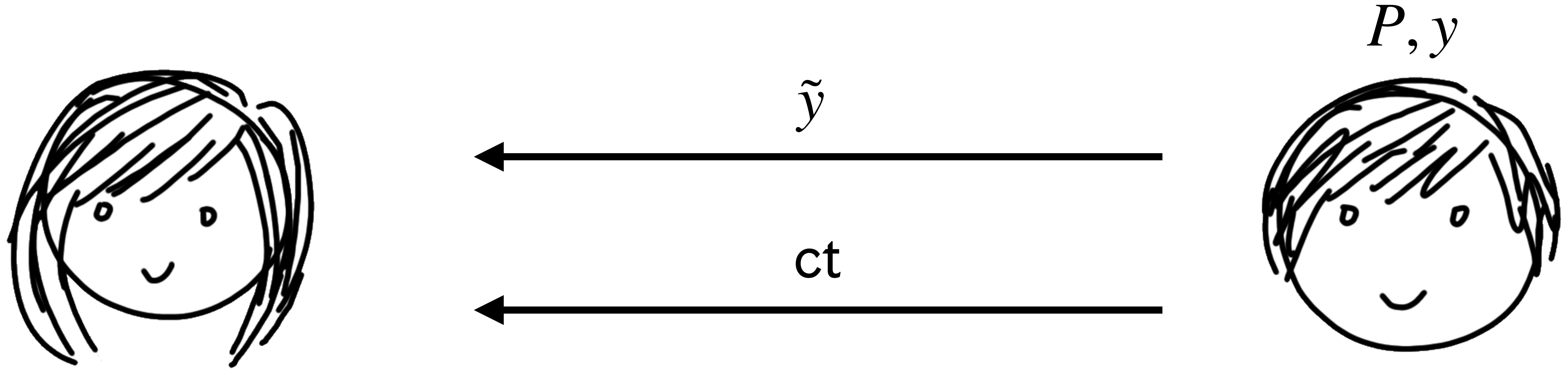
Garbled RAM

RAM-LFE

y

P, x

RAM-LFE vs Garbled RAM



$\tilde{y}, \mathsf{sk} \leftarrow \mathsf{GarbleDB}(y)$

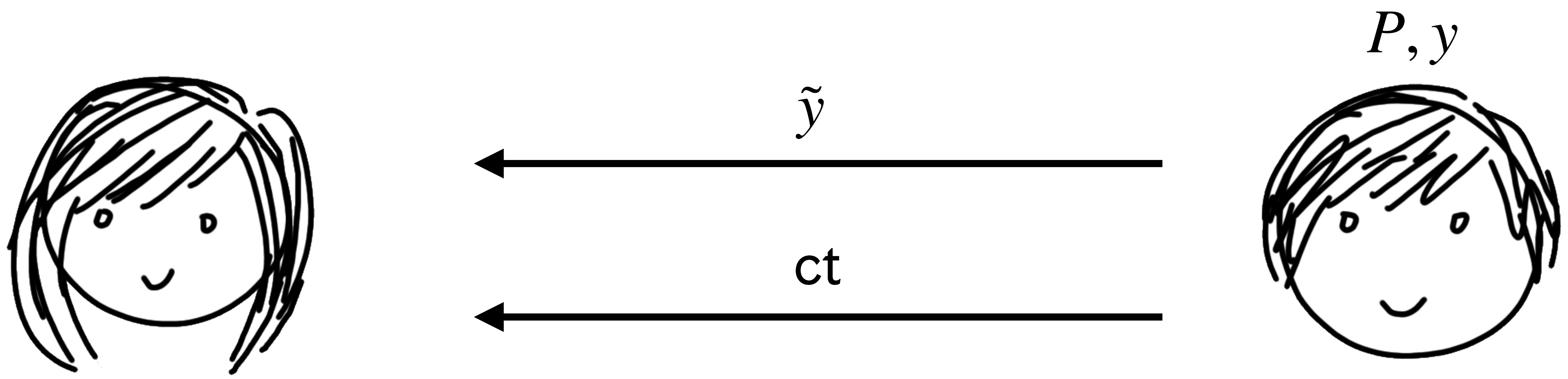
$\mathsf{ct} \leftarrow \mathsf{GarbleProg}(\mathsf{sk}, P)$

y belongs to client and is garbled with respect to their secret key

Garbled RAM

RAM-LFE

RAM-LFE vs Garbled RAM



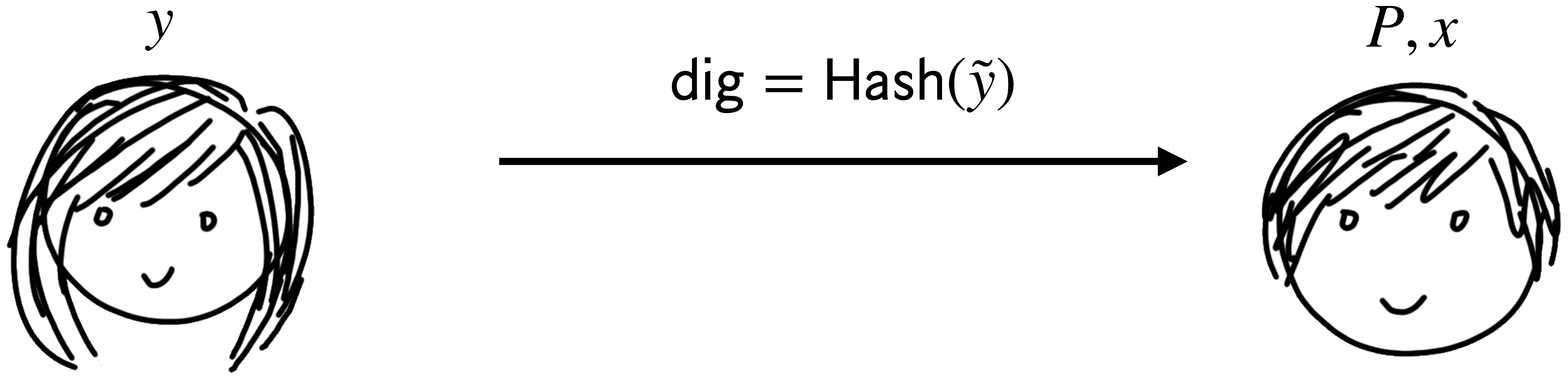
$\tilde{y}, \mathsf{sk} \leftarrow \mathsf{GarbleDB}(y)$

$\mathsf{ct} \leftarrow \mathsf{GarbleProg}(\mathsf{sk}, P)$

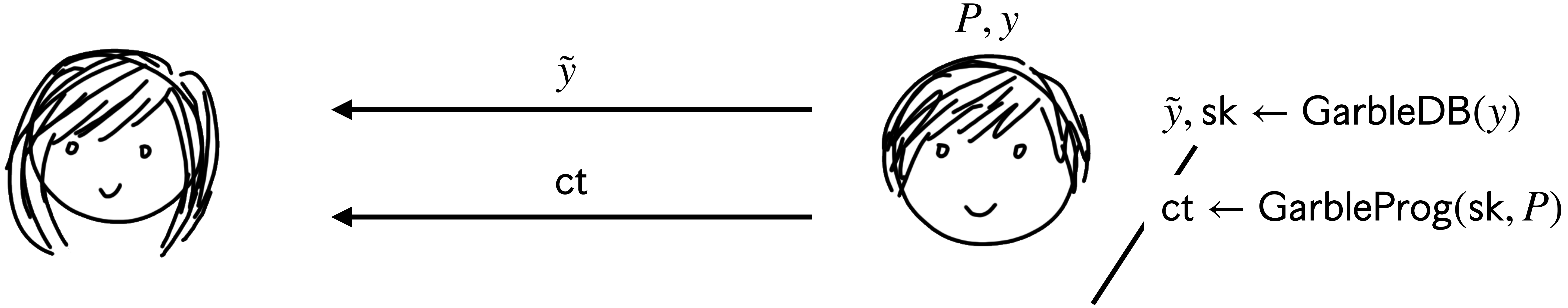
y belongs to client and is garbled with respect to their secret key

Garbled RAM

RAM-LFE



RAM-LFE vs Garbled RAM



P, y

$\tilde{y}$

ct

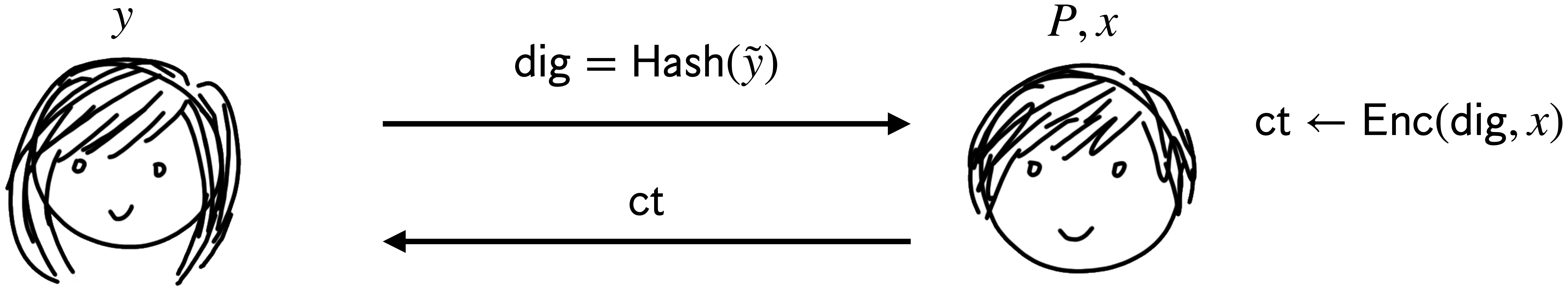
$\tilde{y}, \mathsf{sk} \leftarrow \mathsf{GarbleDB}(y)$

$\mathsf{ct} \leftarrow \mathsf{GarbleProg}(\mathsf{sk}, P)$

y belongs to client and is garbled with respect to their secret key

Garbled RAM

RAM-LFE



y

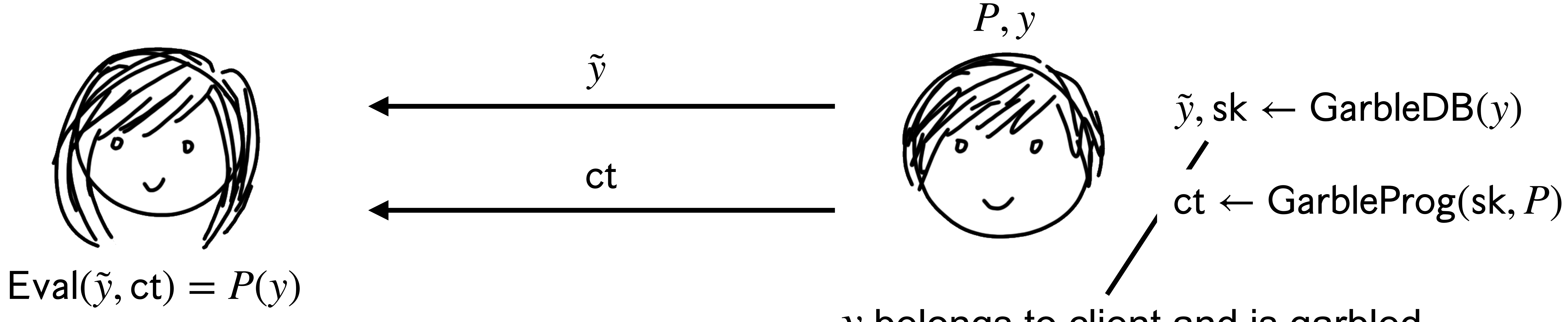
P, x

$\mathsf{dig} = \mathsf{Hash}(\tilde{y})$

ct

$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, x)$

RAM-LFE vs Garbled RAM



P, y

$\tilde{y}$

ct

$\tilde{y}, \mathsf{sk} \leftarrow \mathsf{GarbleDB}(y)$

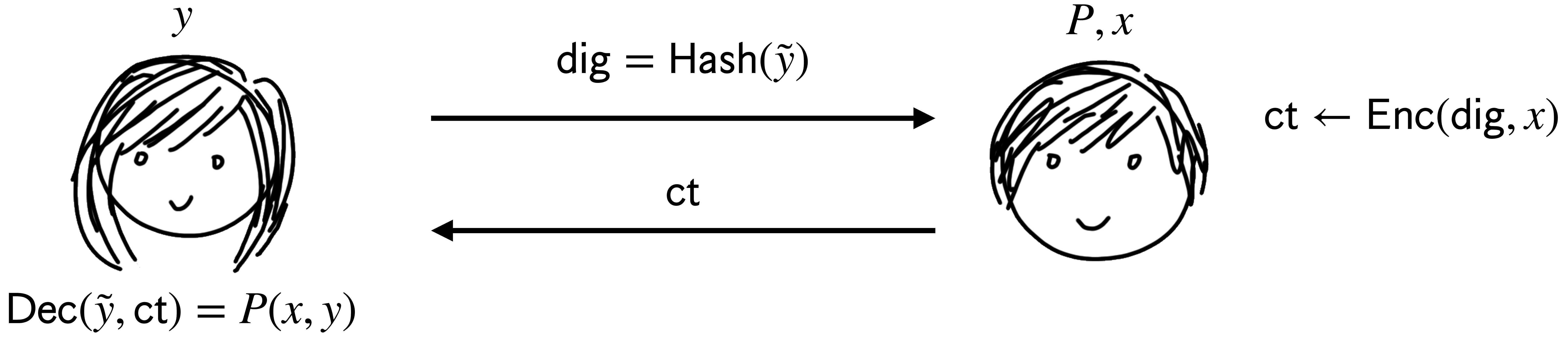
$\mathsf{ct} \leftarrow \mathsf{GarbleProg}(\mathsf{sk}, P)$

$\mathsf{Eval}(\tilde{y}, \mathsf{ct}) = P(y)$

y belongs to client and is garbled with respect to their secret key

Garbled RAM

RAM-LFE



y

P, x

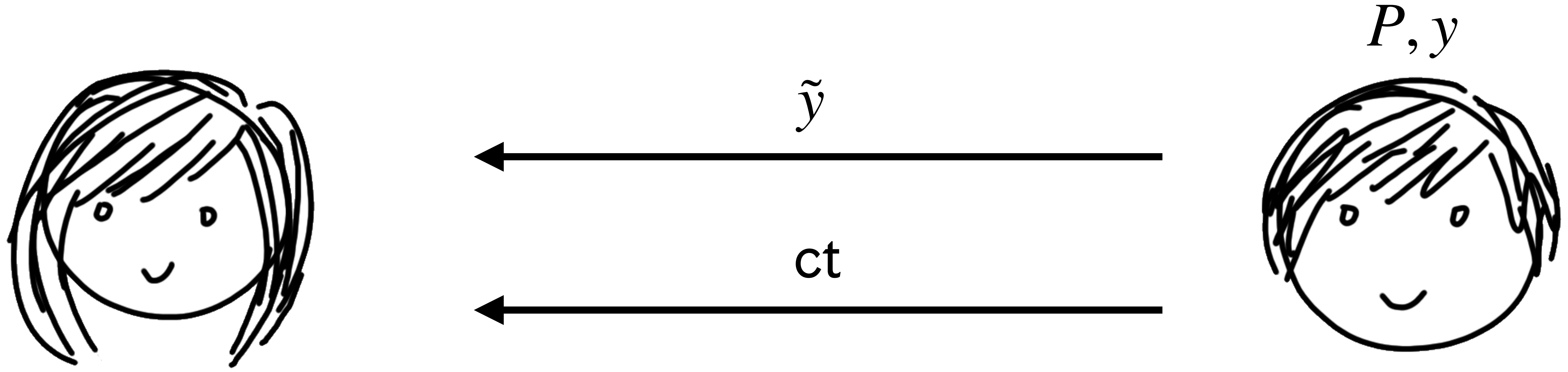
$\mathsf{dig} = \mathsf{Hash}(\tilde{y})$

ct

$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, x)$

$\mathsf{Dec}(\tilde{y}, \mathsf{ct}) = P(x, y)$

RAM-LFE vs Garbled RAM



P, y

$\tilde{y}$

ct

$\tilde{y}, \mathsf{sk} \leftarrow \mathsf{GarbleDB}(y)$

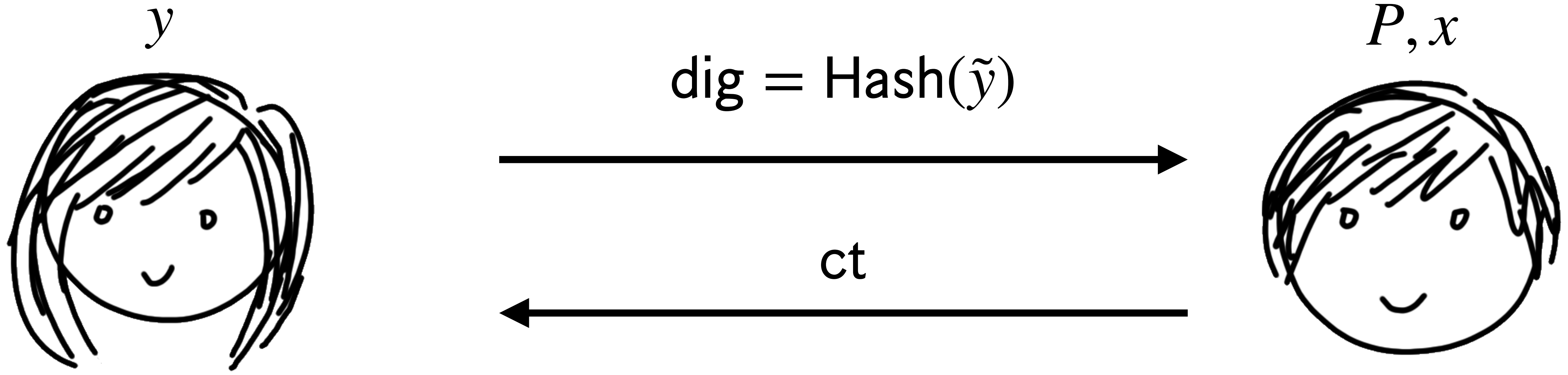
$\mathsf{ct} \leftarrow \mathsf{GarbleProg}(\mathsf{sk}, P)$

$\mathsf{Eval}(\tilde{y}, \mathsf{ct}) = P(y)$

y belongs to client and is garbled with respect to their secret key

Garbled RAM

RAM-LFE



y

P, x

$\mathsf{dig} = \mathsf{Hash}(\tilde{y})$

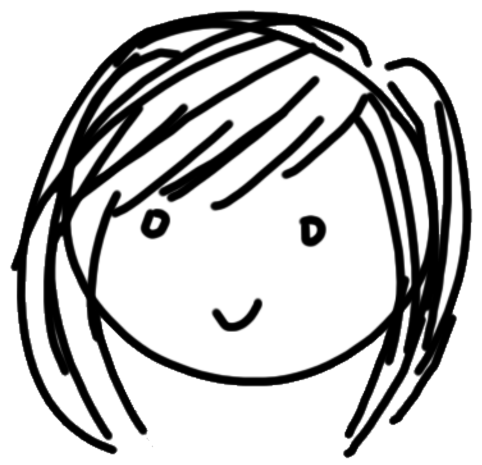
ct

$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{dig}, x)$

$\mathsf{Dec}(\tilde{y}, \mathsf{ct}) = P(x, y)$

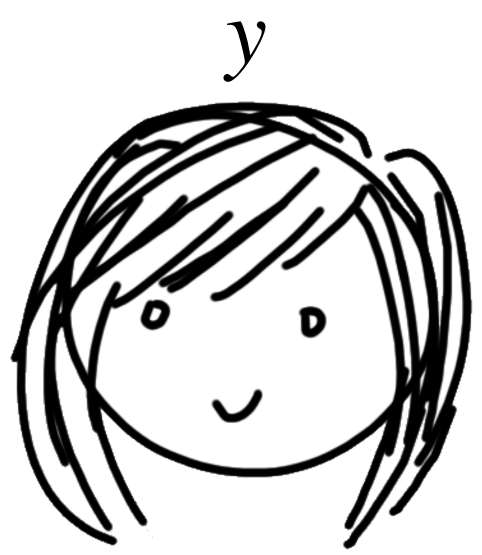
Need to evaluate over ***public*** database y

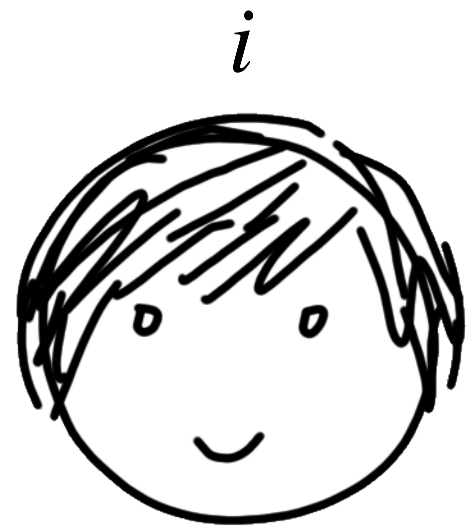
DEPIR vs ORAM



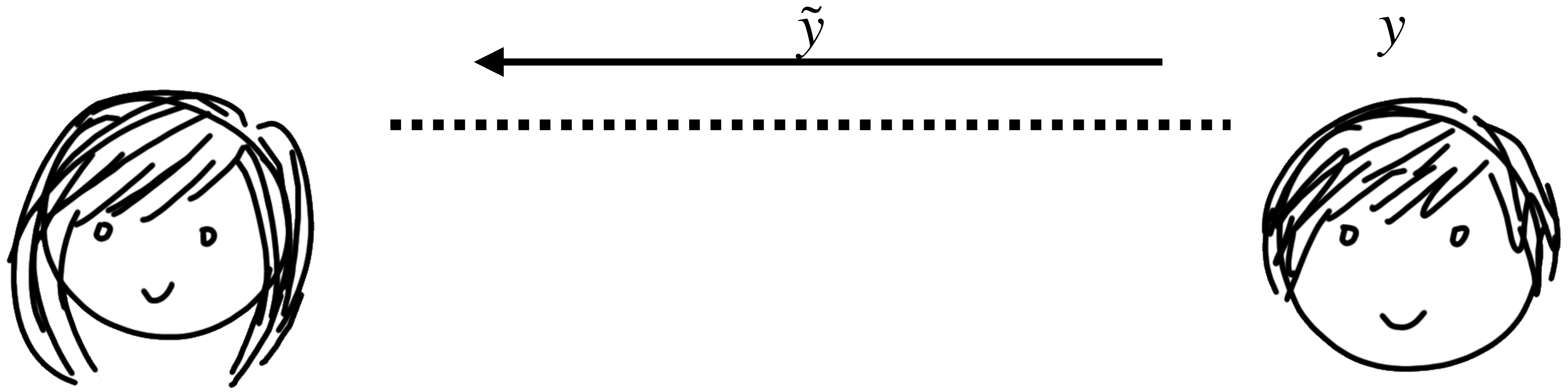
ORAM

DEPIR





DEPIR vs ORAM



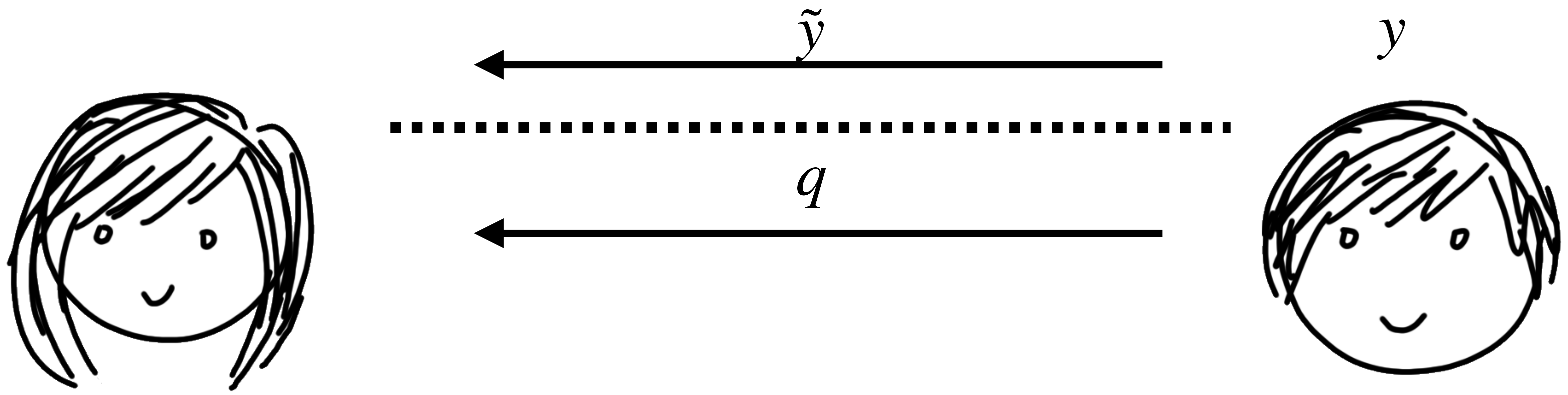
$\tilde{y}, \mathsf{sk} \leftarrow \mathsf{ORAM}.\mathsf{Init}(y)$

ORAM

DEPIR

$I, r \leftarrow \mathsf{DEPIR}.\mathsf{Query}(i)$

DEPIR vs ORAM

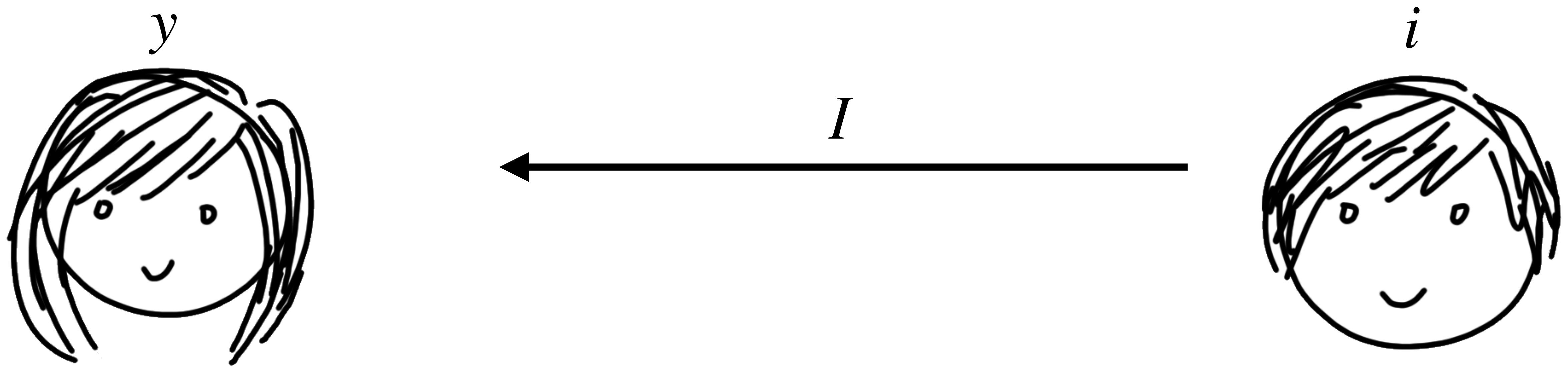


$\tilde{y}, \mathsf{sk} \leftarrow \mathsf{ORAM}.\mathsf{Init}(y)$

$q \leftarrow \mathsf{ORAM}.\mathsf{Read}(\mathsf{sk}, i)$

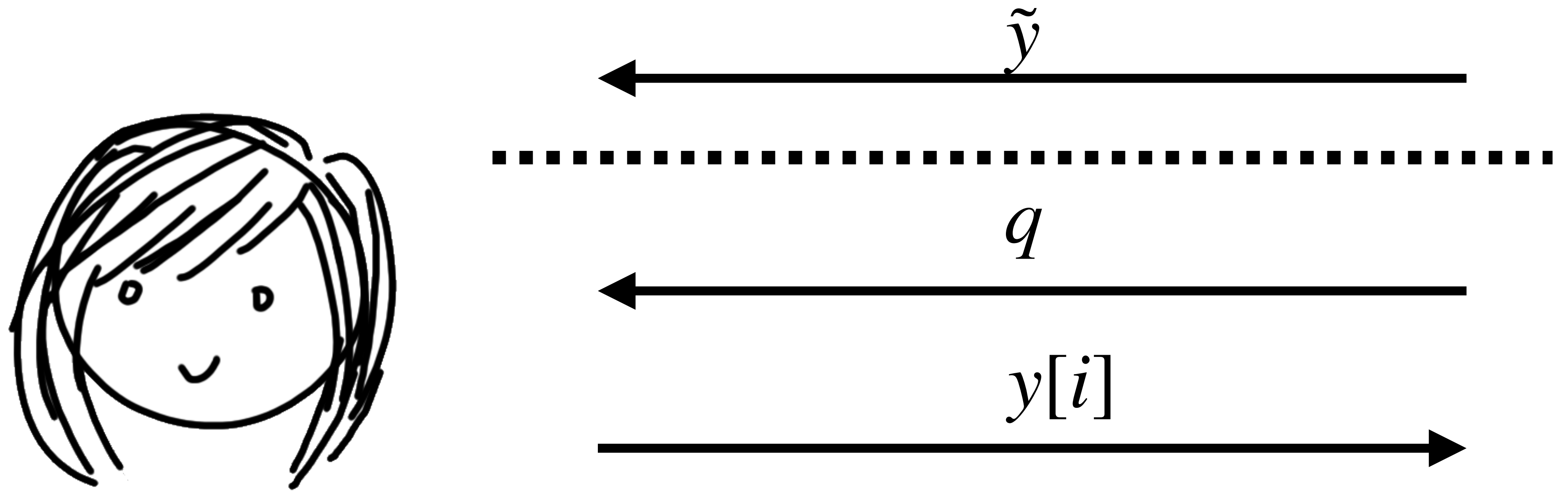
ORAM

DEPIR



$I, r \leftarrow \mathsf{DEPIR}.\mathsf{Query}(i)$

DEPIR vs ORAM



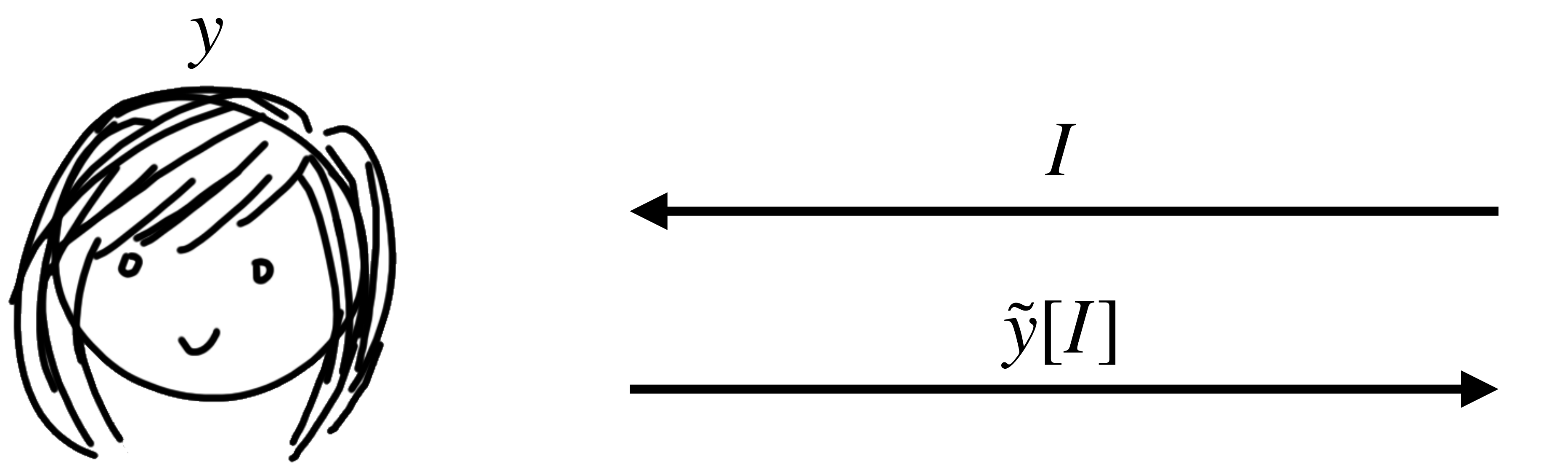
$\tilde{y}, \mathsf{sk} \leftarrow \mathsf{ORAM.Init}(y)$

$q \leftarrow \mathsf{ORAM.Read}(\mathsf{sk}, i)$

ORAM

DEPIR

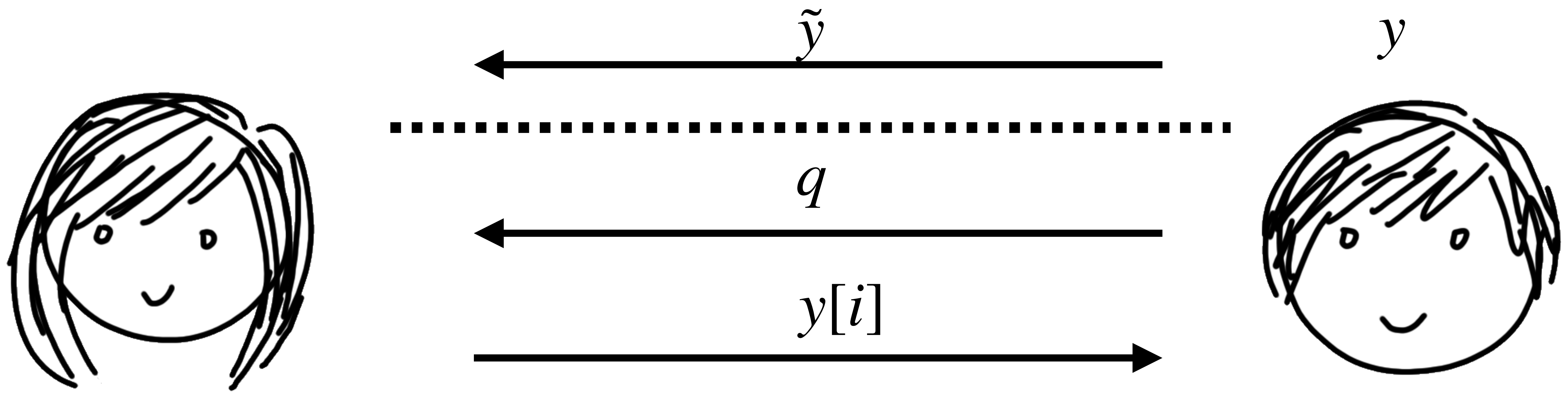
$\tilde{y} := \mathsf{DEPIR.Prep}(y)$



$I, r \leftarrow \mathsf{DEPIR.Query}(i)$

$y[i] = \mathsf{DEPIR.Dec}(\tilde{y}[I], r)$

DEPIR vs ORAM



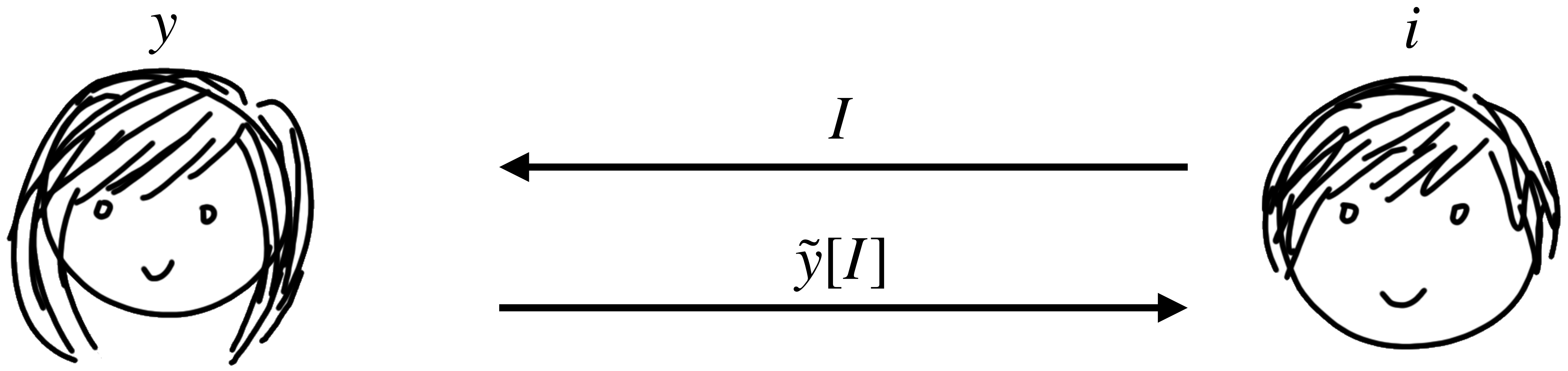
$\tilde{y}, \mathsf{sk} \leftarrow \mathsf{ORAM.Init}(y)$

$q \leftarrow \mathsf{ORAM.Read}(\mathsf{sk}, i)$

ORAM — Private database, requires client secret key

DEPIR

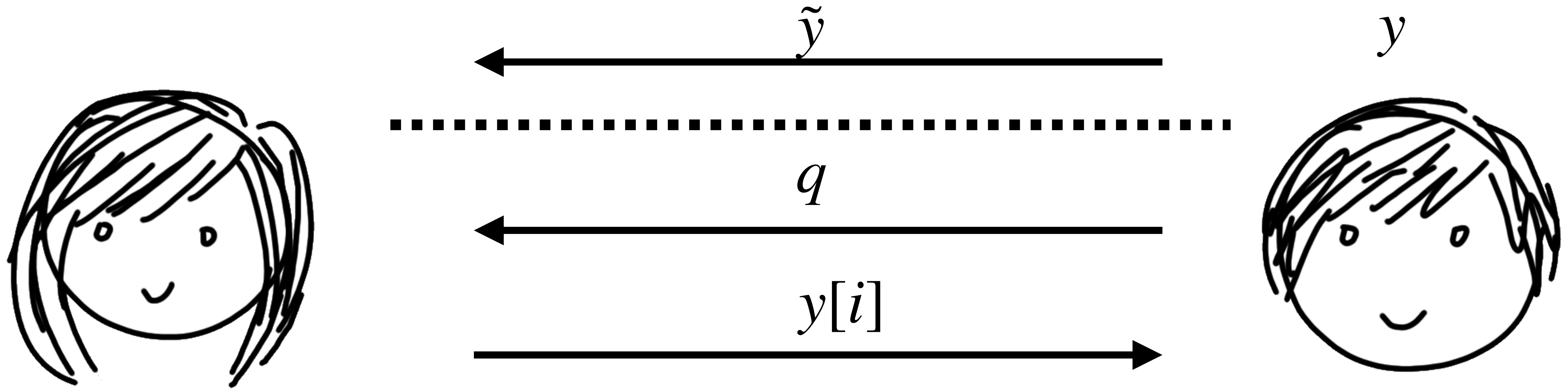
$\tilde{y} := \mathsf{DEPIR.Prep}(y)$



$I, r \leftarrow \mathsf{DEPIR.Query}(i)$

$y[i] = \mathsf{DEPIR.Dec}(\tilde{y}[I], r)$

DEPIR vs ORAM



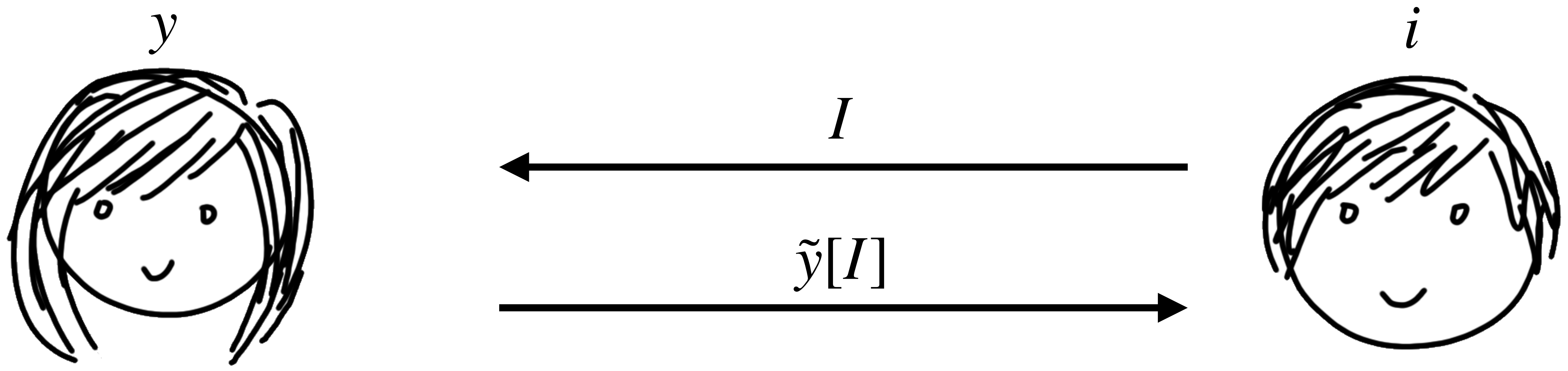
$\tilde{y}, \mathsf{sk} \leftarrow \mathsf{ORAM.Init}(y)$

$q \leftarrow \mathsf{ORAM.Read}(\mathsf{sk}, i)$

ORAM — Private database, requires client secret key

DEPIR — Public database, public deterministic preprocessing

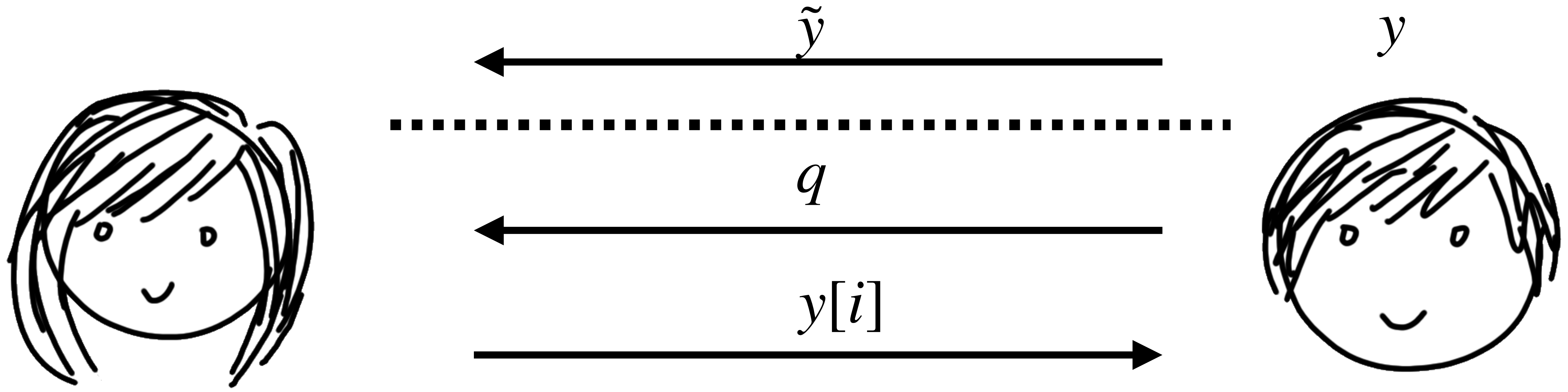
$\tilde{y} := \mathsf{DEPIR.Prep}(y)$



$I, r \leftarrow \mathsf{DEPIR.Query}(i)$

$y[i] = \mathsf{DEPIR.Dec}(\tilde{y}[I], r)$

DEPIR vs ORAM



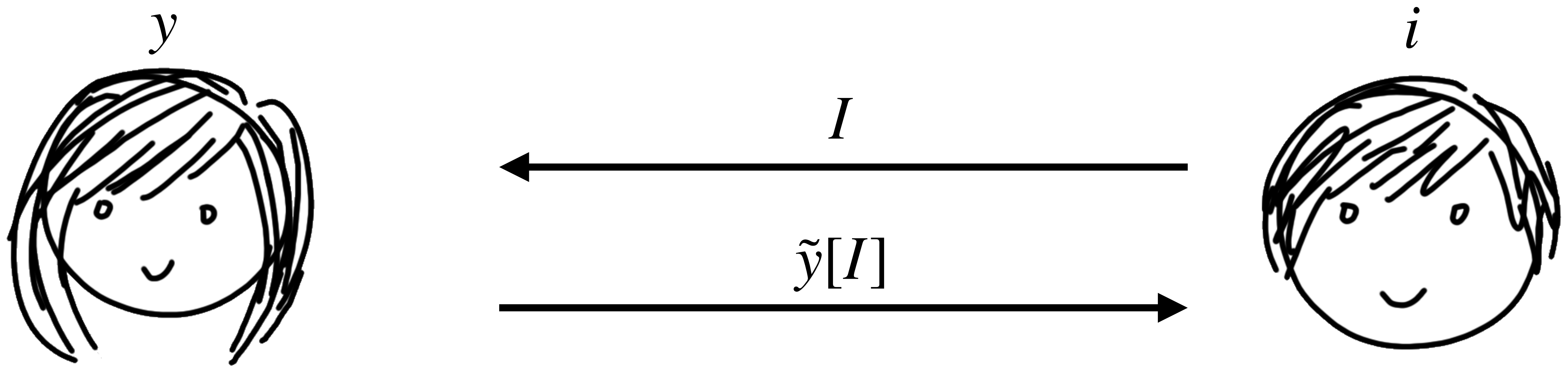
$\tilde{y}, \mathsf{sk} \leftarrow \mathsf{ORAM}.\mathsf{Init}(y)$

$q \leftarrow \mathsf{ORAM}.\mathsf{Read}(\mathsf{sk}, i)$

ORAM — Private database, requires client secret key

DEPIR — Public database, public deterministic preprocessing

$\tilde{y} := \mathsf{DEPIR}.\mathsf{Prep}(y)$



$I, r \leftarrow \mathsf{DEPIR}.\mathsf{Query}(i)$

$y[i] = \mathsf{DEPIR}.\mathsf{Dec}(\tilde{y}[I], r)$

Prior Work: [Lin-**M**-Wichs'23] build DEPIR from RingLWE

Construction template

We follow the general template for constructing Garbled RAM

Construction template

We follow the general template for constructing Garbled RAM

1. Construct “UMA” secure version

- Security only protects ***internal state*** not the ***memory access pattern***

Construction template

We follow the general template for constructing Garbled RAM

1. Construct "UMA" secure version

- Security only protects ***internal state*** not the ***memory access pattern***

For LFE: Crucially need UMA version to allow public database

Construction template

We follow the general template for constructing Garbled RAM

1. Construct "UMA" secure version

- Security only protects ***internal state*** not the ***memory access pattern***

For LFE: Crucially need UMA version to allow public database

2. Upgrade to full security

- Protect access pattern with ORAM **+ DEPIR**

Construction template

We follow the general template for constructing Garbled RAM

1. Construct "UMA" secure version
- Security only protects ***internal state*** not the ***memory access pattern***

For LFE: Crucially need UMA version to allow public database

2. Upgrade to full security
- Protect access pattern with ORAM **+ DEPIR**

3. For strong efficiency: Use iO to obfuscate the client's encryption procedure and offload to server

Construction template

We follow the general template for constructing Garbled RAM

1. Construct "UMA" secure version
- Security only protects ***internal state*** not the ***memory access pattern***

For LFE: Crucially need UMA version to allow public database

2. Upgrade to full security
- Protect access pattern with ORAM **+ DEPIR**

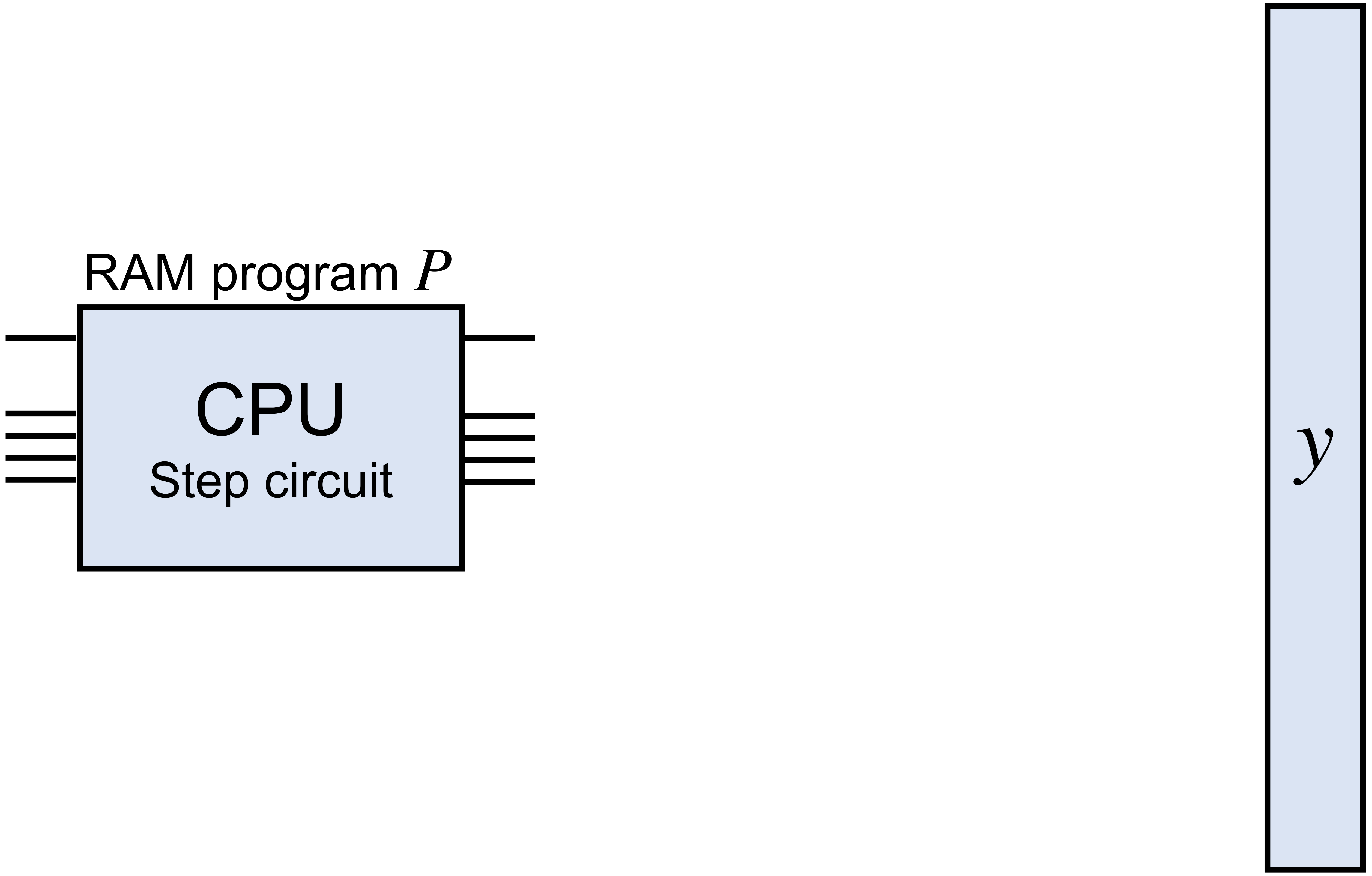
3. For strong efficiency: Use iO to obfuscate the client's encryption procedure and offload to server

Requires careful argument and special ORAM construction

UMA secure RAM-LFE

RAM-NISC from [Cho-Döttling-Garg-Gupta-Miao-Polychroniadou’17]

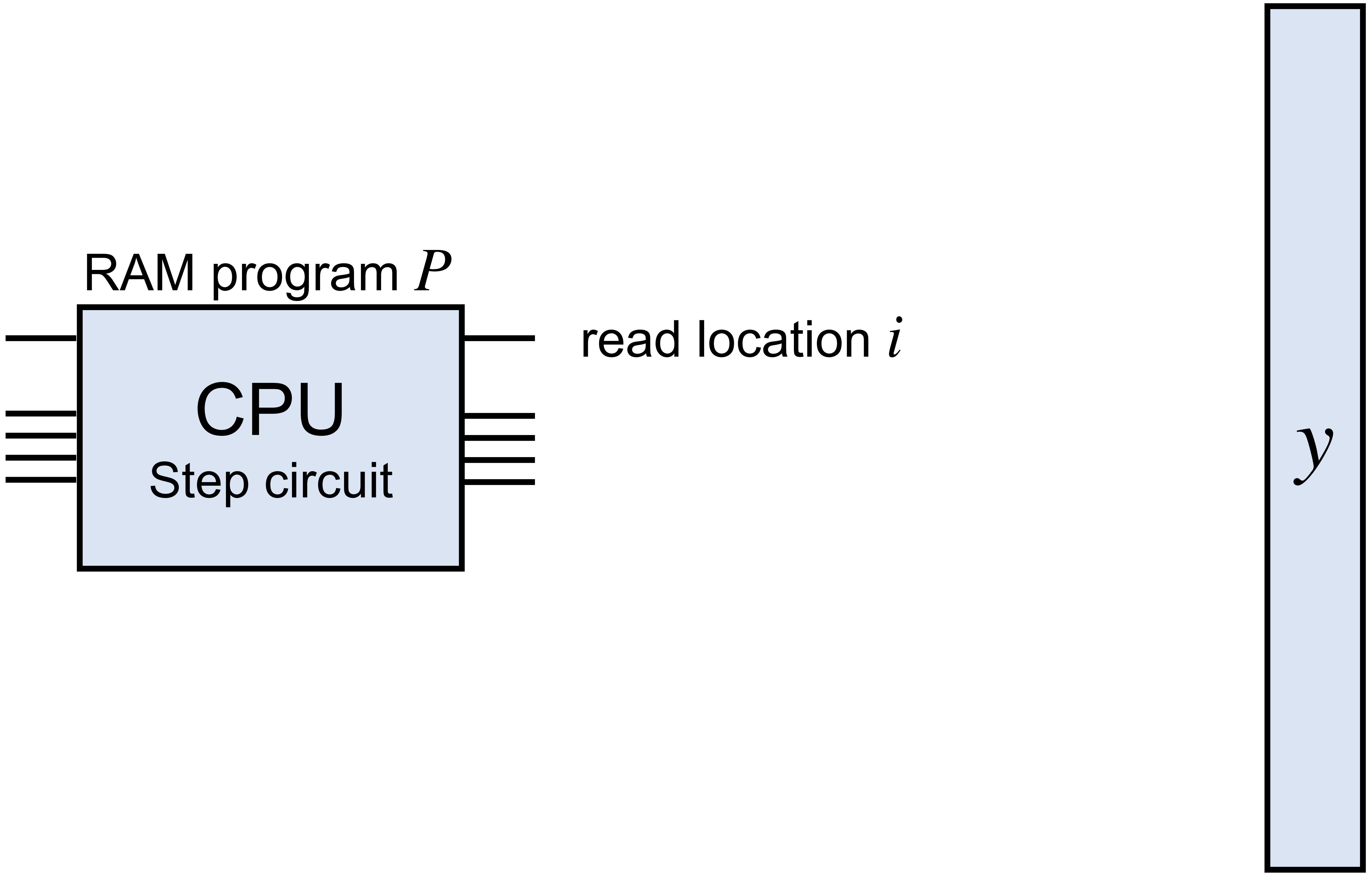
Building blocks: Laconic Oblivious Transfer + Garbled circuits



UMA secure RAM-LFE

RAM-NISC from [Cho-Döttling-Garg-Gupta-Miao-Polychroniadou'17]

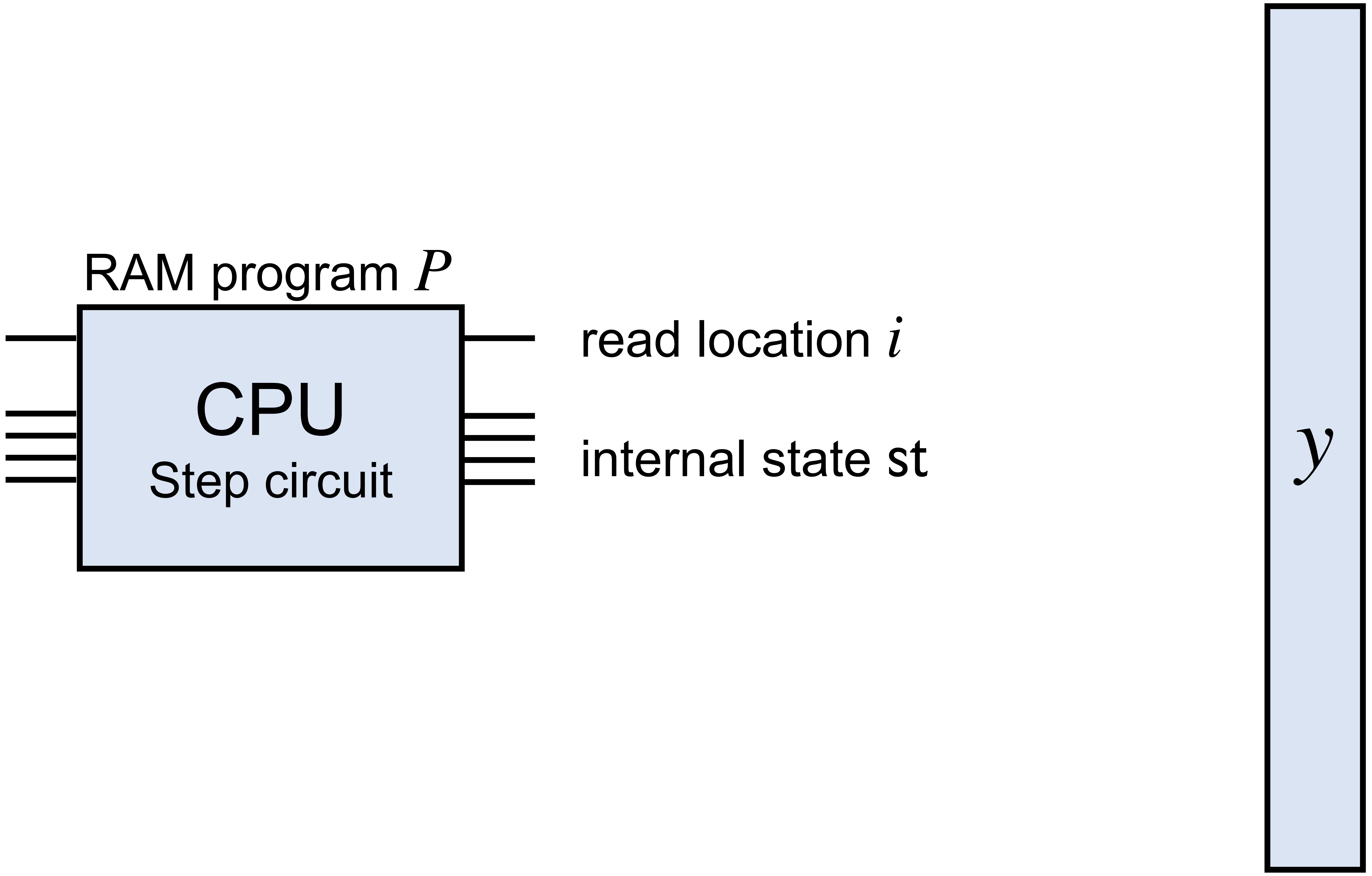
Building blocks: Laconic Oblivious Transfer + Garbled circuits



UMA secure RAM-LFE

RAM-NISC from [Cho-Döttling-Garg-Gupta-Miao-Polychroniadou'17]

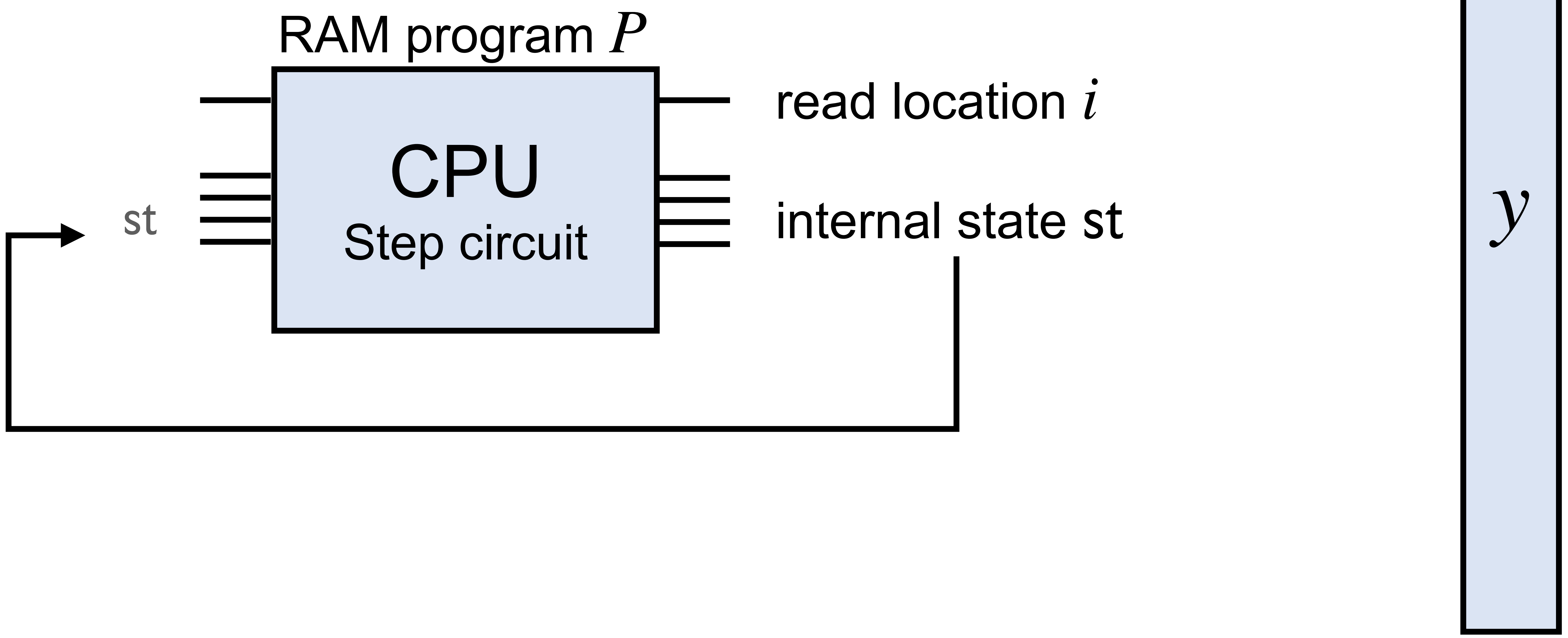
Building blocks: Laconic Oblivious Transfer + Garbled circuits



UMA secure RAM-LFE

RAM-NISC from [Cho-Döttling-Garg-Gupta-Miao-Polychroniadou’17]

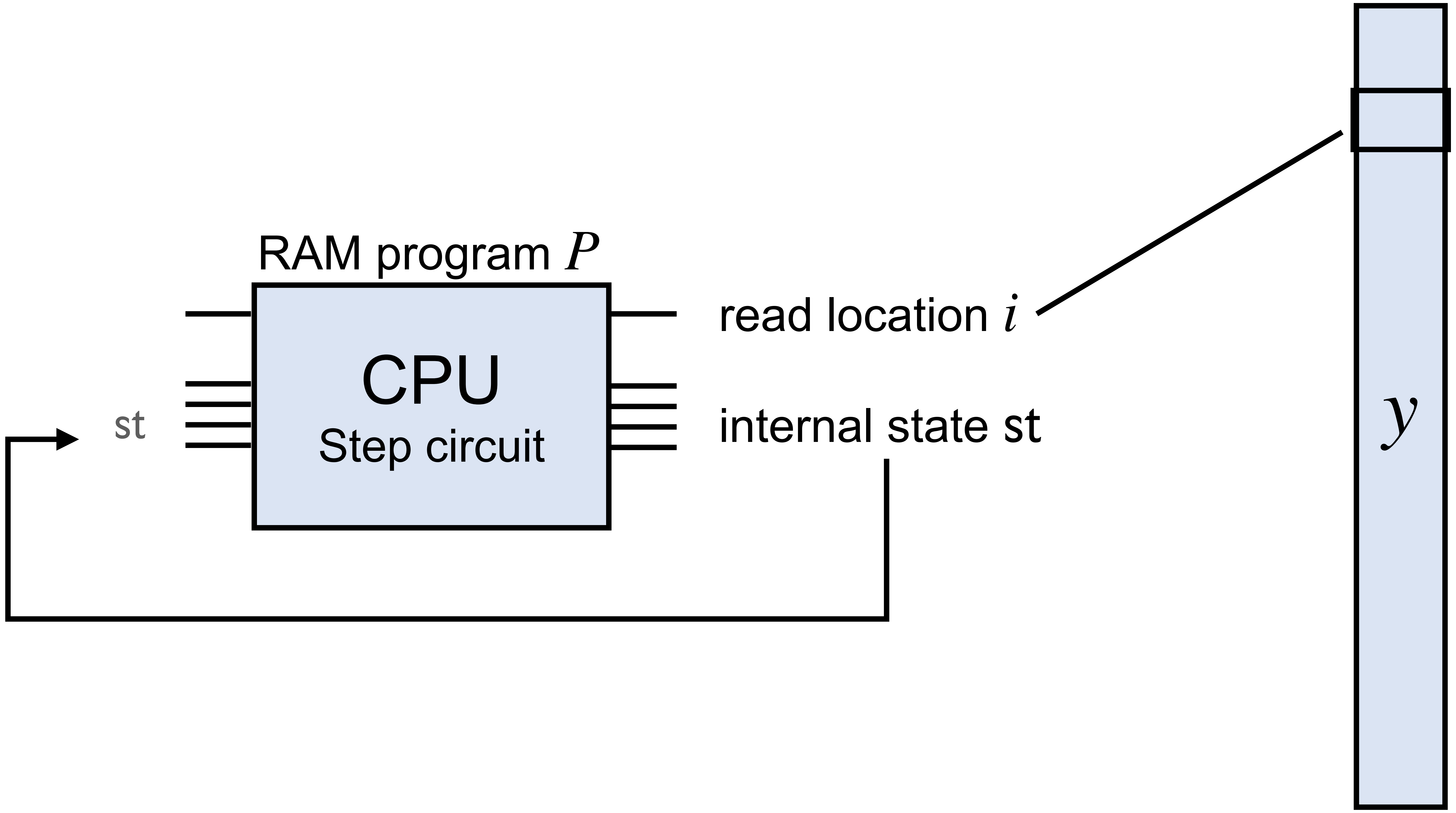
Building blocks: Laconic Oblivious Transfer + Garbled circuits



UMA secure RAM-LFE

RAM-NISC from [Cho-Döttling-Garg-Gupta-Miao-Polychroniadou’17]

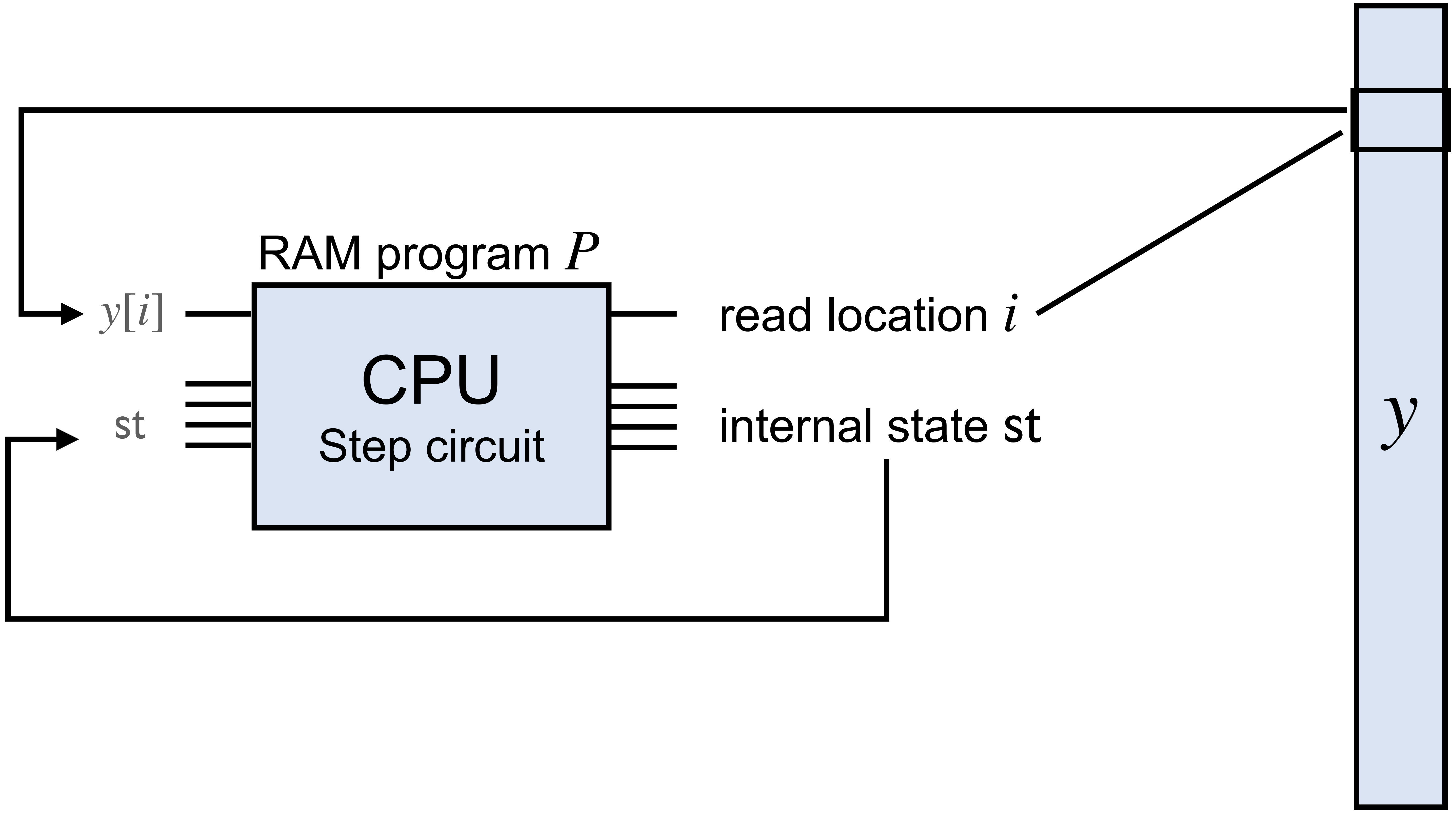
Building blocks: Laconic Oblivious Transfer + Garbled circuits



UMA secure RAM-LFE

RAM-NISC from [Cho-Döttling-Garg-Gupta-Miao-Polychroniadou’17]

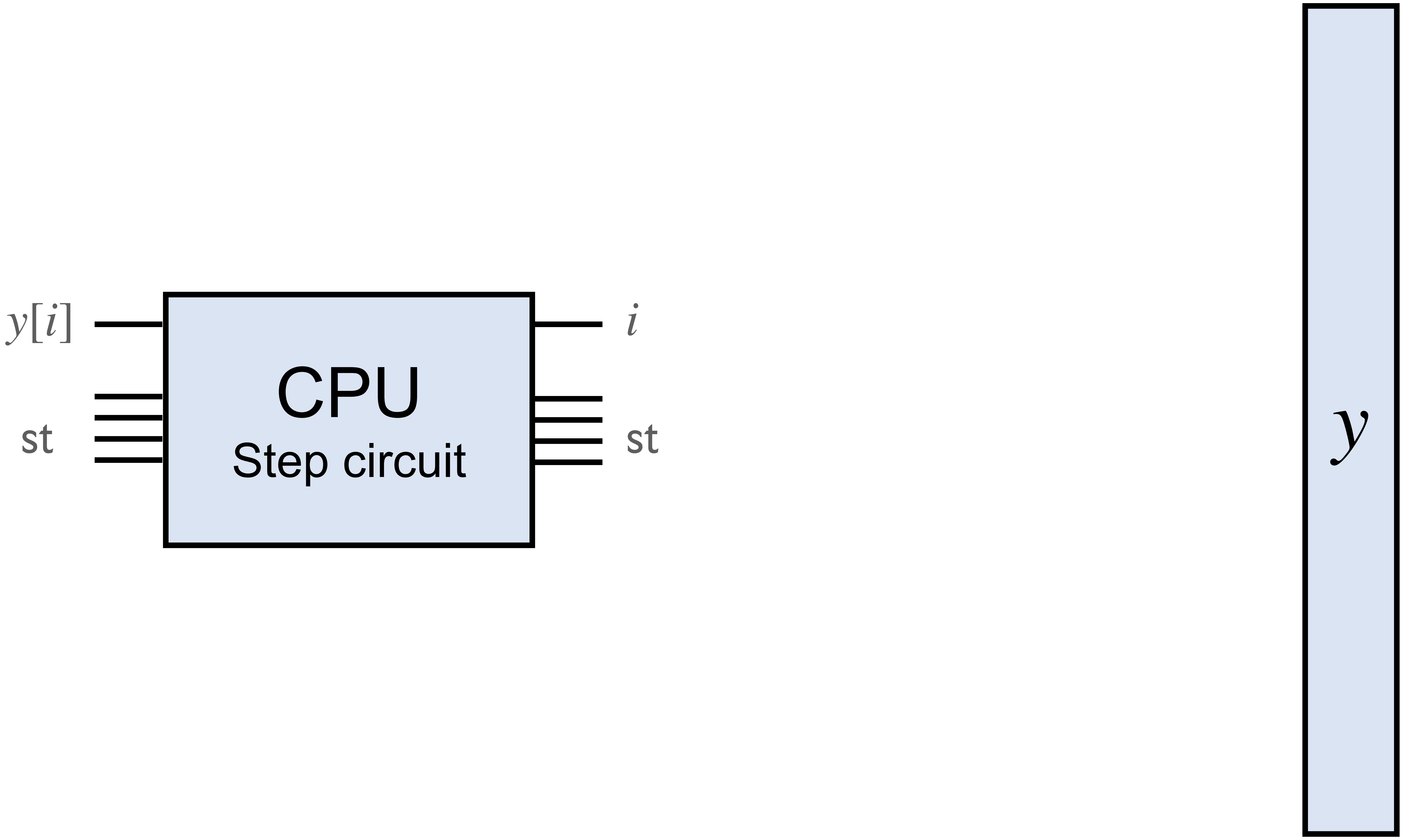
Building blocks: Laconic Oblivious Transfer + Garbled circuits



UMA secure RAM-LFE

RAM-NISC from [Cho-Döttling-Garg-Gupta-Miao-Polychroniadou'17]

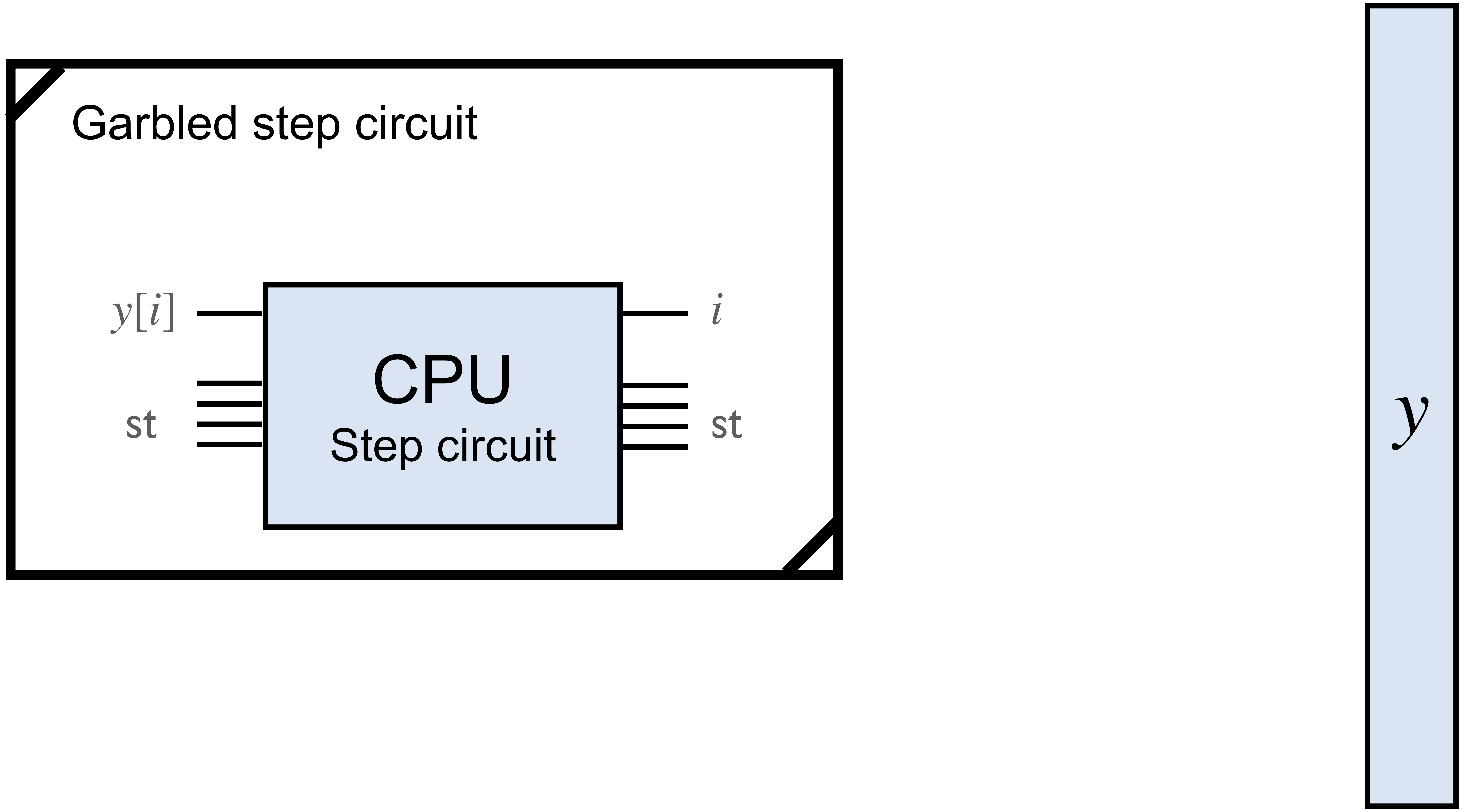
Building blocks: Laconic Oblivious Transfer + Garbled circuits



UMA secure RAM-LFE

RAM-NISC from [Cho-Döttling-Garg-Gupta-Miao-Polychroniadou’17]

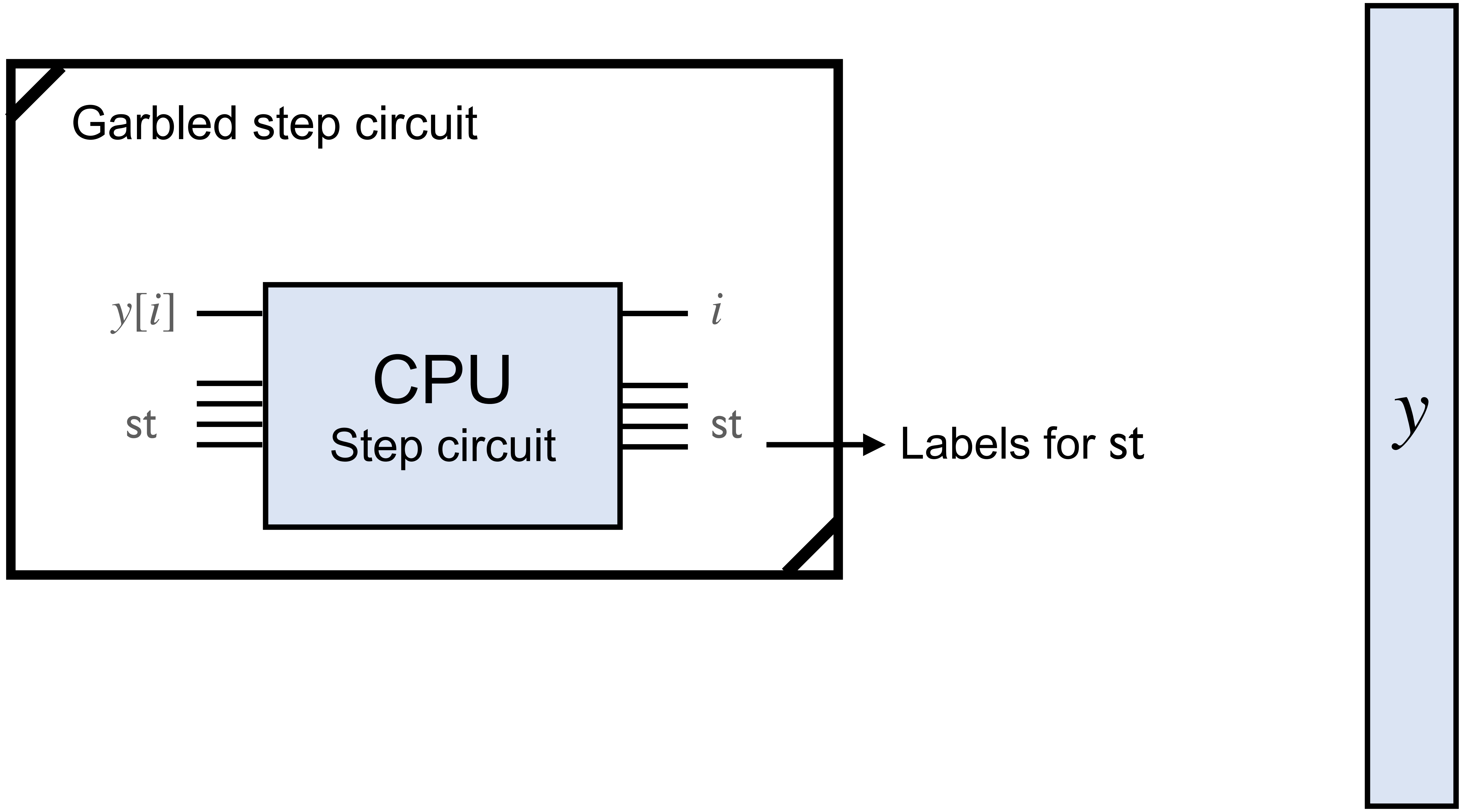
Building blocks: Laconic Oblivious Transfer + Garbled circuits



UMA secure RAM-LFE

RAM-NISC from [Cho-Döttling-Garg-Gupta-Miao-Polychroniadou'17]

Building blocks: Laconic Oblivious Transfer + Garbled circuits



Garbled step circuit

$y[i]$

st

CPU

Step circuit

i

st

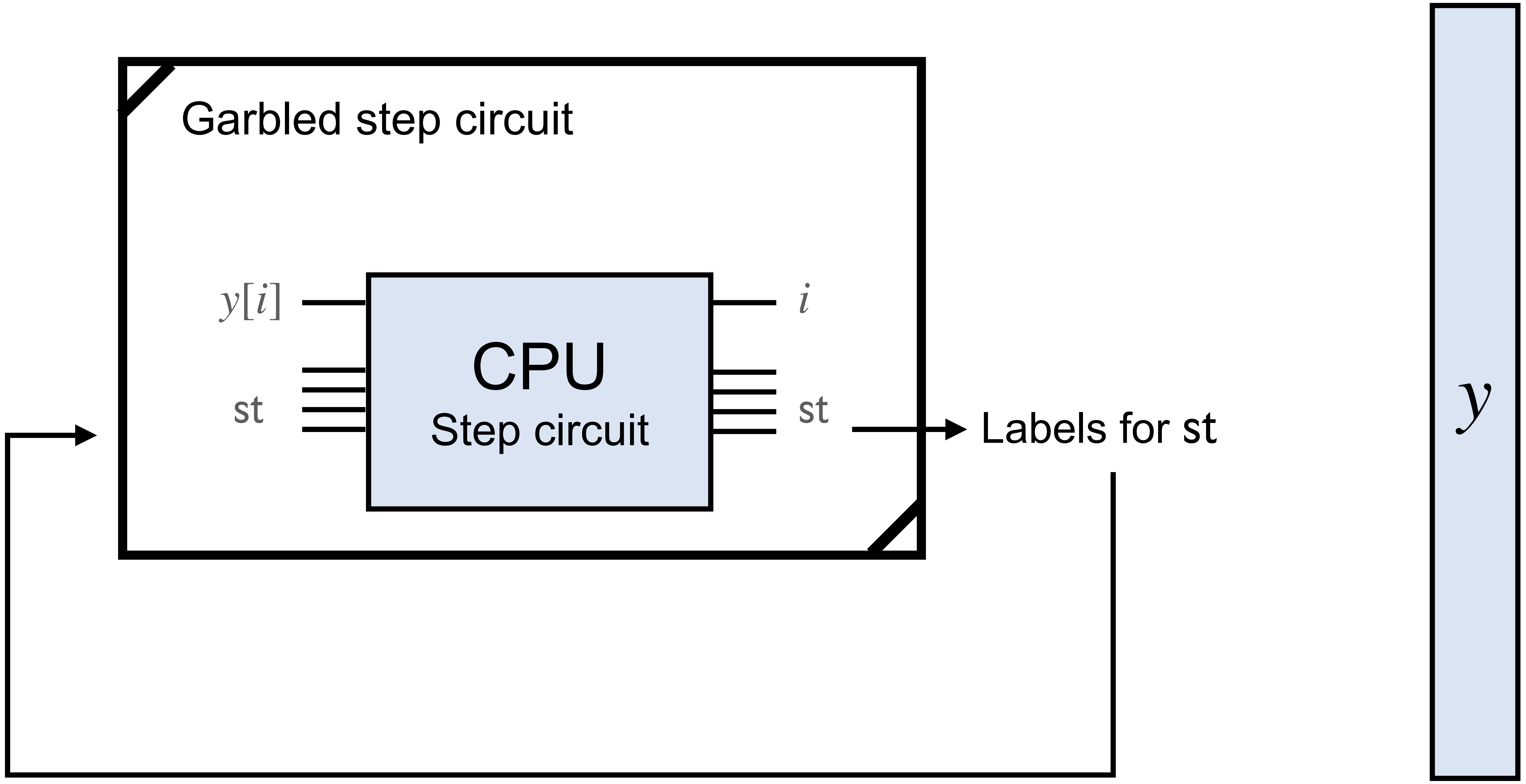
Labels for st

y

UMA secure RAM-LFE

RAM-NISC from [Cho-Döttling-Garg-Gupta-Miao-Polychroniadou’17]

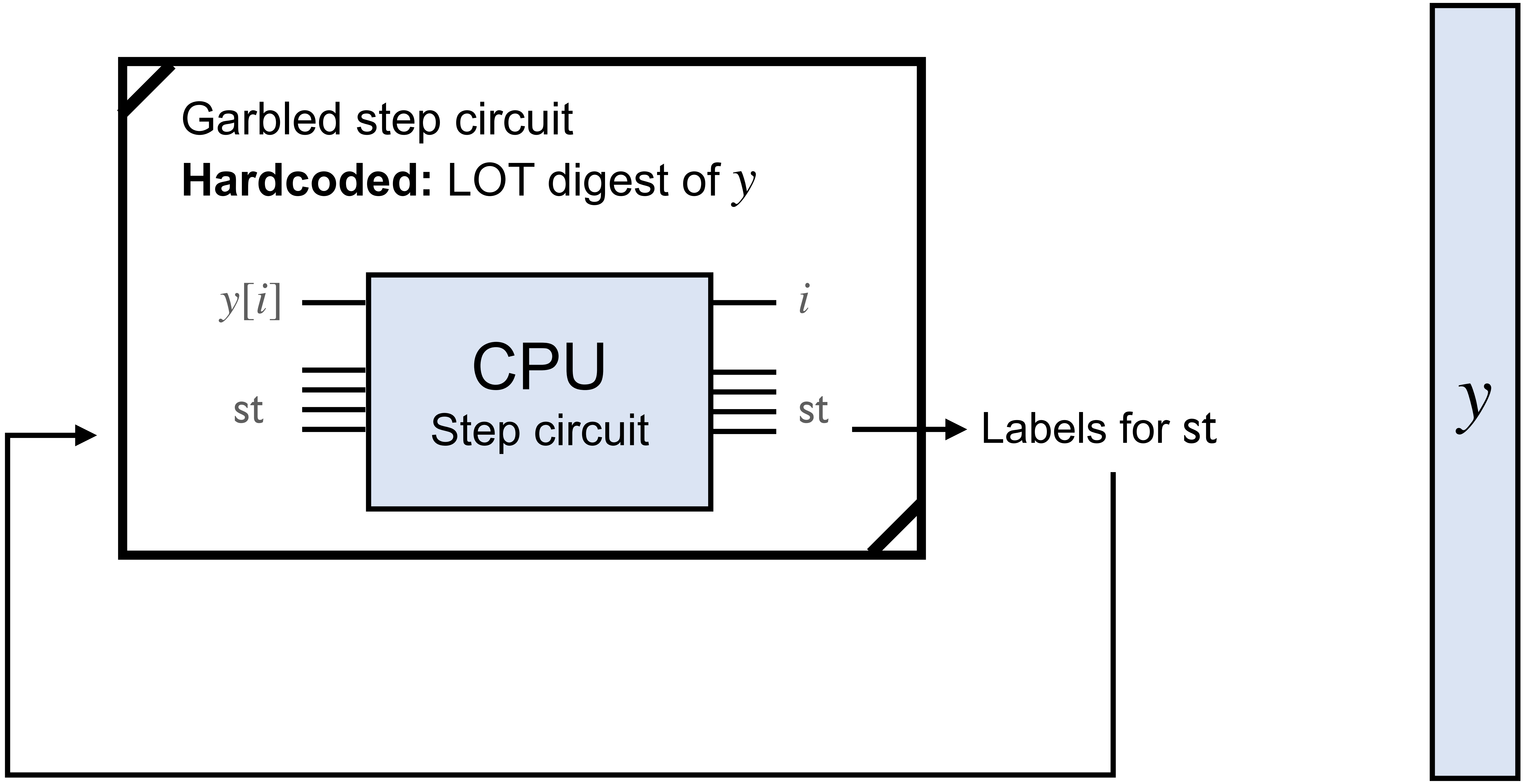
Building blocks: Laconic Oblivious Transfer + Garbled circuits



UMA secure RAM-LFE

RAM-NISC from [Cho-Döttling-Garg-Gupta-Miao-Polychroniadou'17]

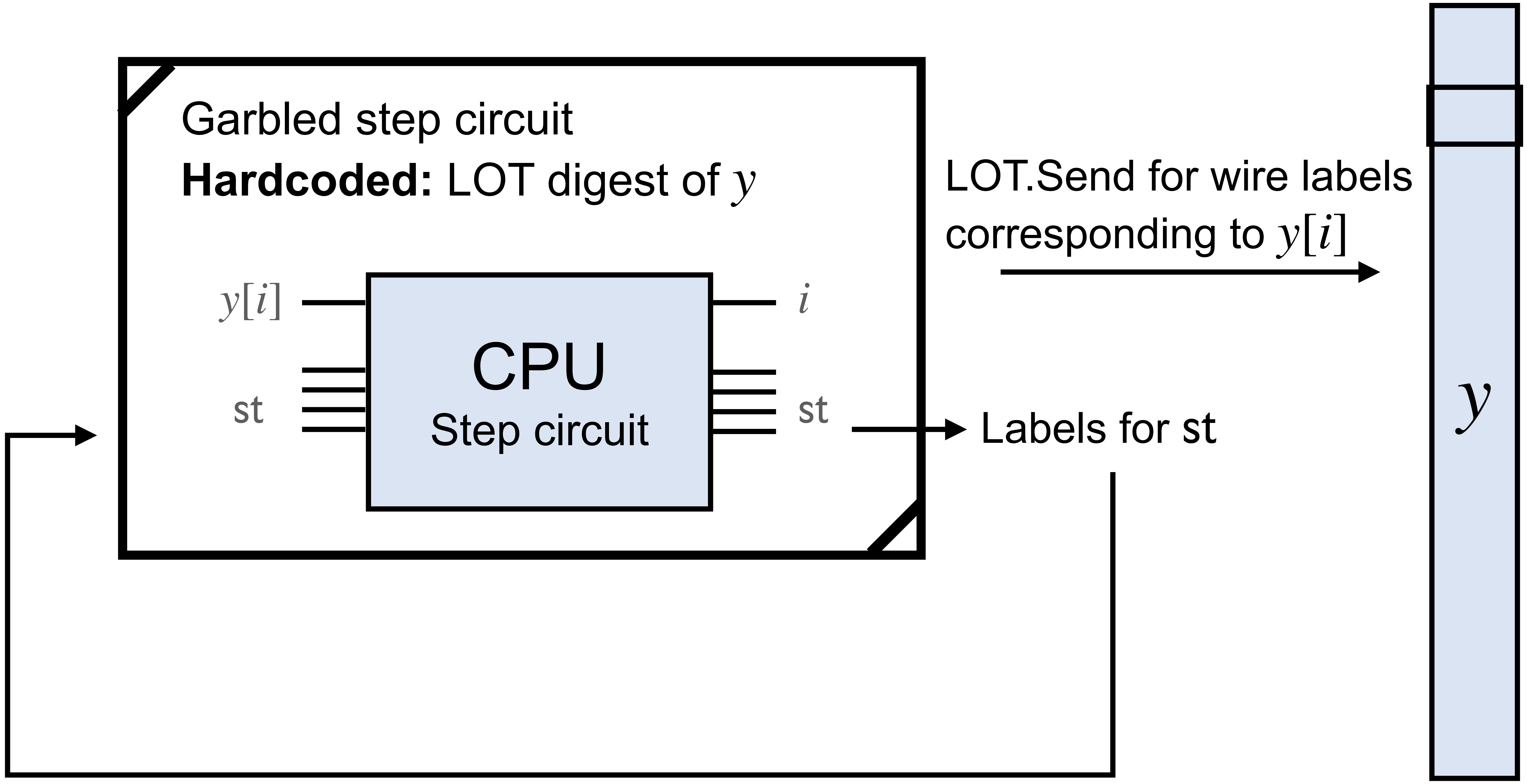
Building blocks: Laconic Oblivious Transfer + Garbled circuits



UMA secure RAM-LFE

RAM-NISC from [Cho-Döttling-Garg-Gupta-Miao-Polychroniadou'17]

Building blocks: Laconic Oblivious Transfer + Garbled circuits

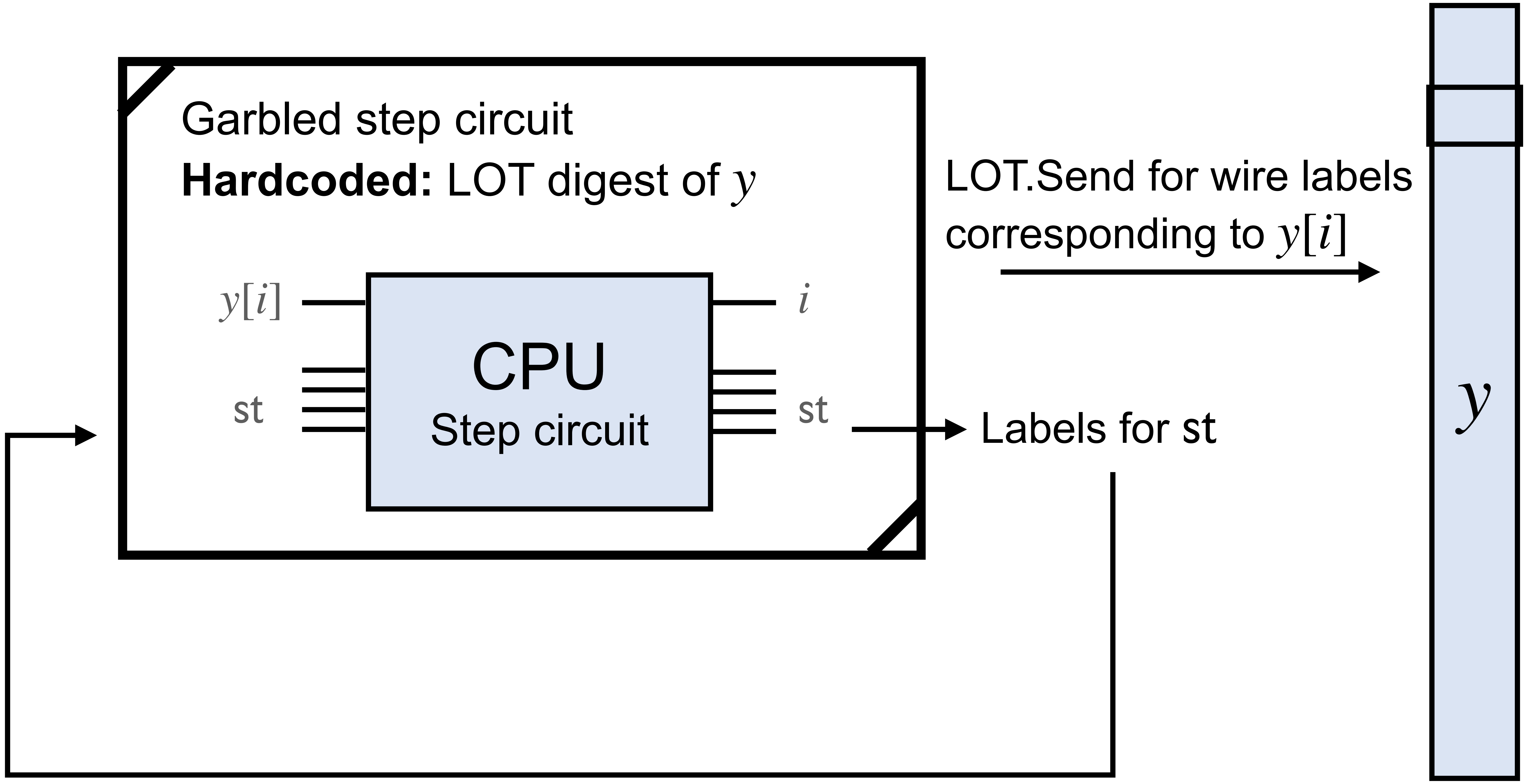


Garbled step circuit
Hardcoded: LOT digest of y
$y[i]$
st
CPU
Step circuit
i
st
LOT.Send for wire labels corresponding to $y[i]$
Labels for st
y

UMA secure RAM-LFE

RAM-NISC from [Cho-Döttling-Garg-Gupta-Miao-Polychroniadou’17]

Building blocks: Laconic Oblivious Transfer + Garbled circuits



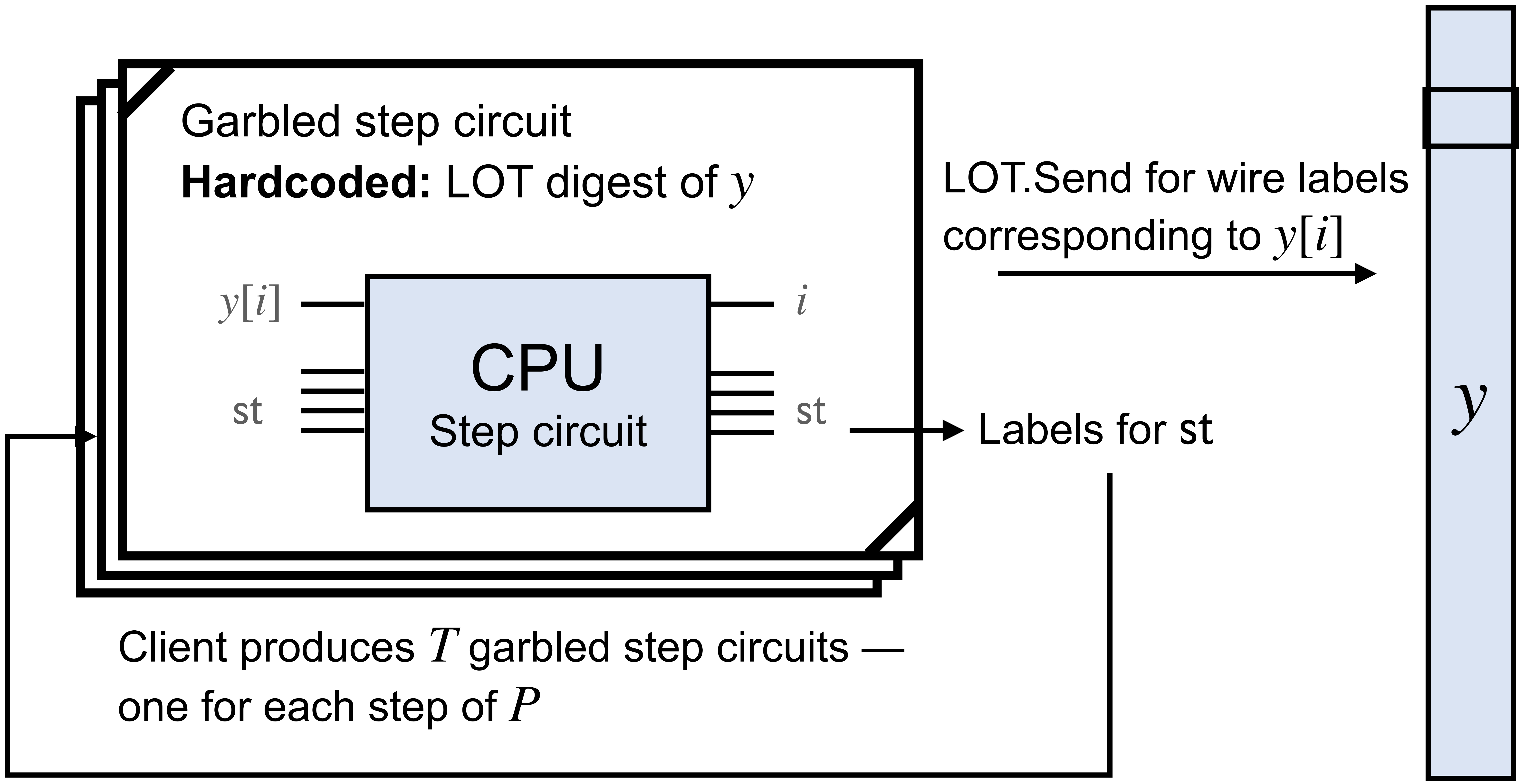
Garbled step circuit
Hardcoded: LOT digest of y
$y[i]$
st
CPU
Step circuit
i
st
LOT.Send for wire labels corresponding to $y[i]$
Labels for st
y

LOT ensures that only the label for $y[i]$ is revealed (and not $1 - y[i]$)

UMA secure RAM-LFE

RAM-NISC from [Cho-Döttling-Garg-Gupta-Miao-Polychroniadou’17]

Building blocks: Laconic Oblivious Transfer + Garbled circuits



Garbled step circuit
Hardcoded: LOT digest of y
$y[i]$
st
CPU
Step circuit
i
st
Labels for st
LOT.Send for wire labels corresponding to $y[i]$
y

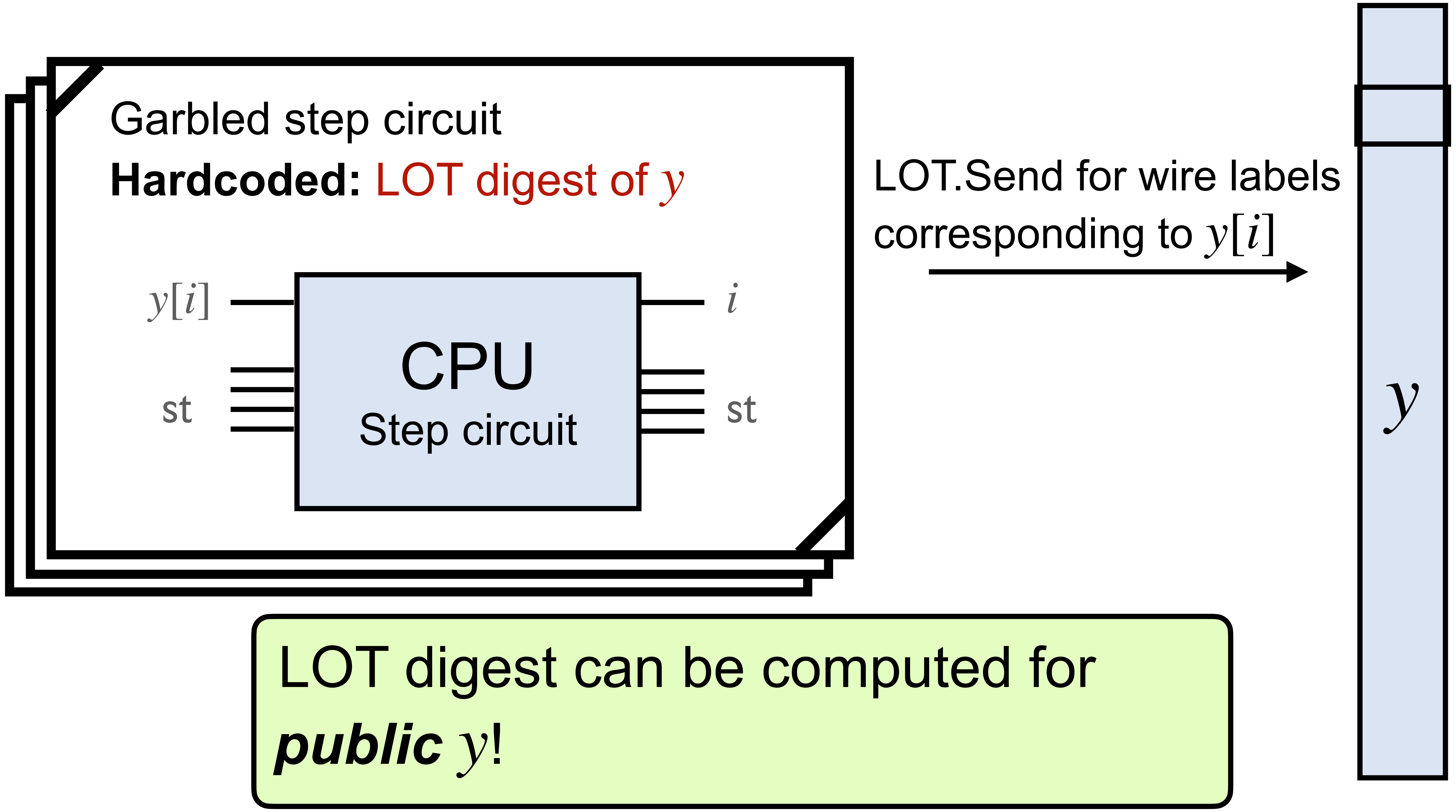
Client produces T garbled step circuits — one for each step of P

LOT ensures that only the label for $y[i]$ is revealed (and not $1 - y[i]$)

UMA secure RAM-LFE

RAM-NISC from [Cho-Döttling-Garg-Gupta-Miao-Polychroniadou'17]

Building blocks: Laconic Oblivious Transfer + Garbled circuits



Garbled step circuit

Hardcoded: LOT digest of y

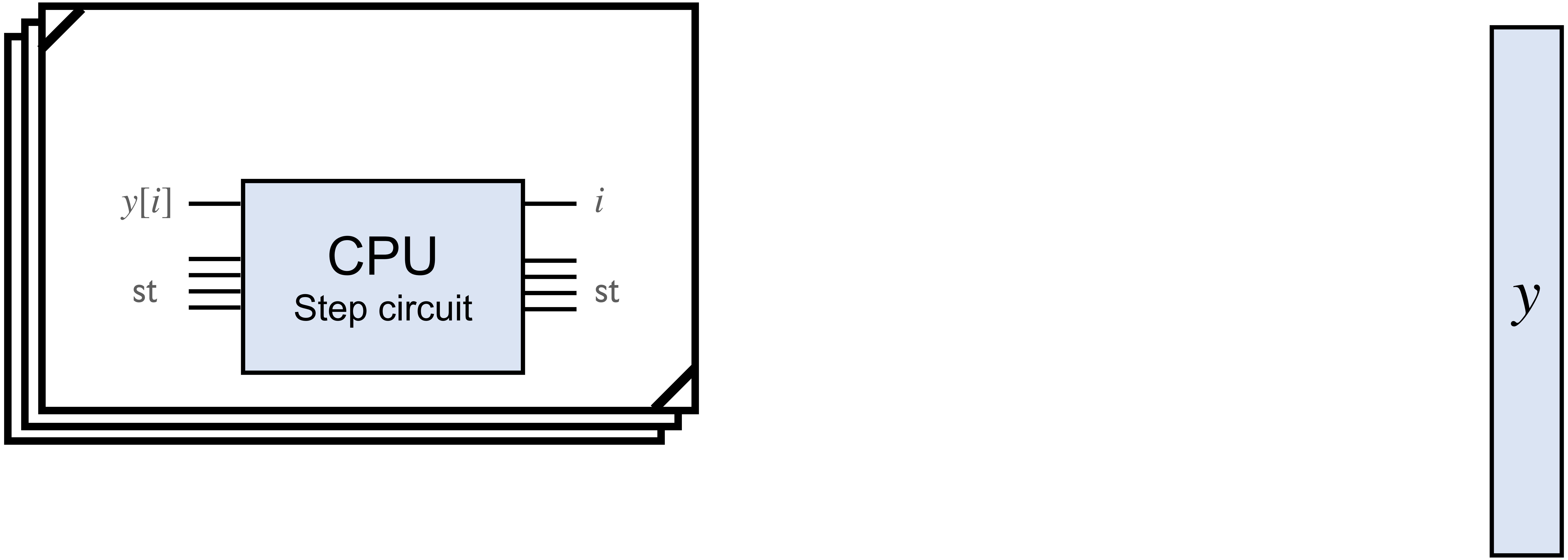
LOT.Send for wire labels corresponding to $y[i]$

LOT ensures that only the label for $y[i]$ is revealed (and not $1 - y[i]$)

LOT digest can be computed for ***public*** y!

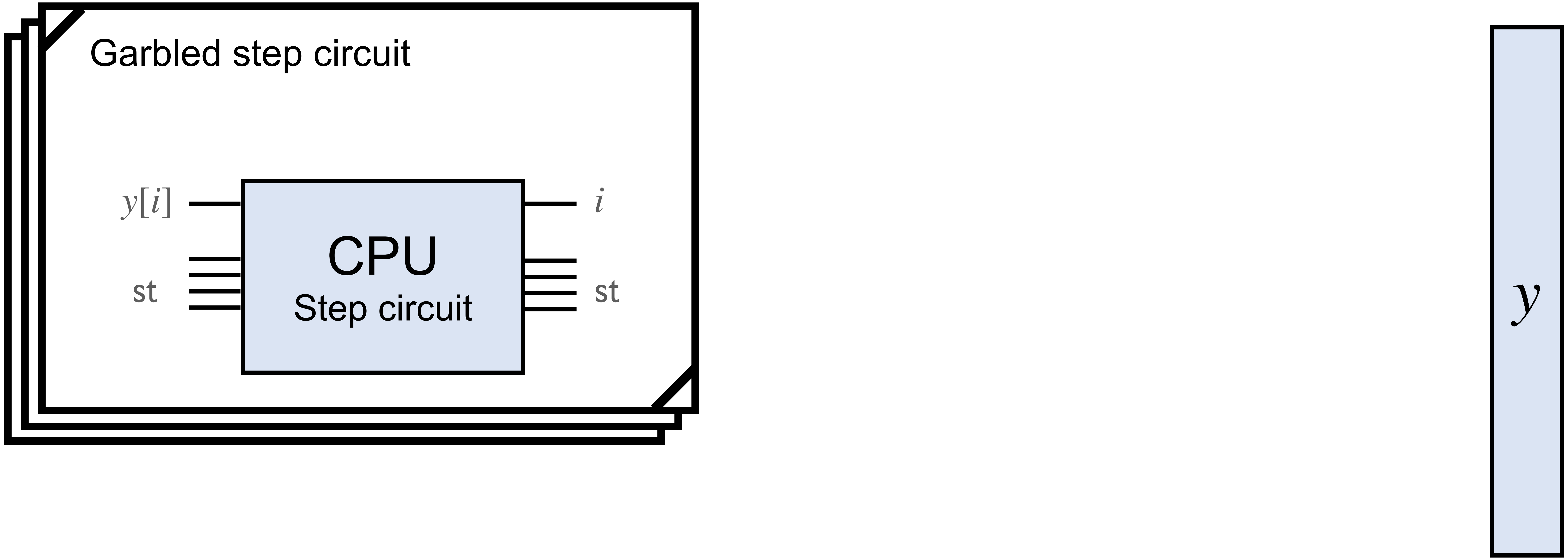
Full security with DEPIR

+ ORAM for the client’s database x



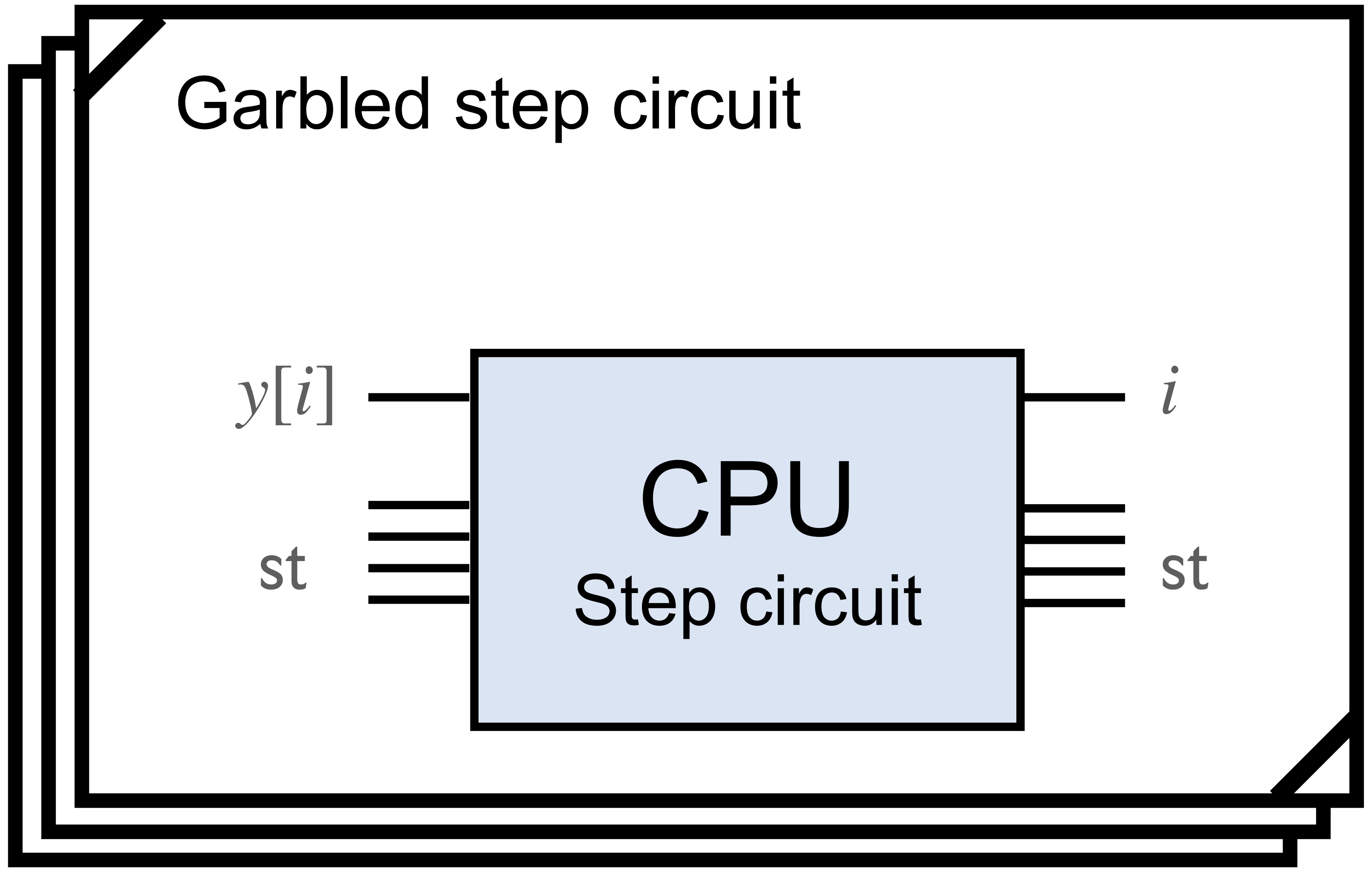
Full security with DEPIR

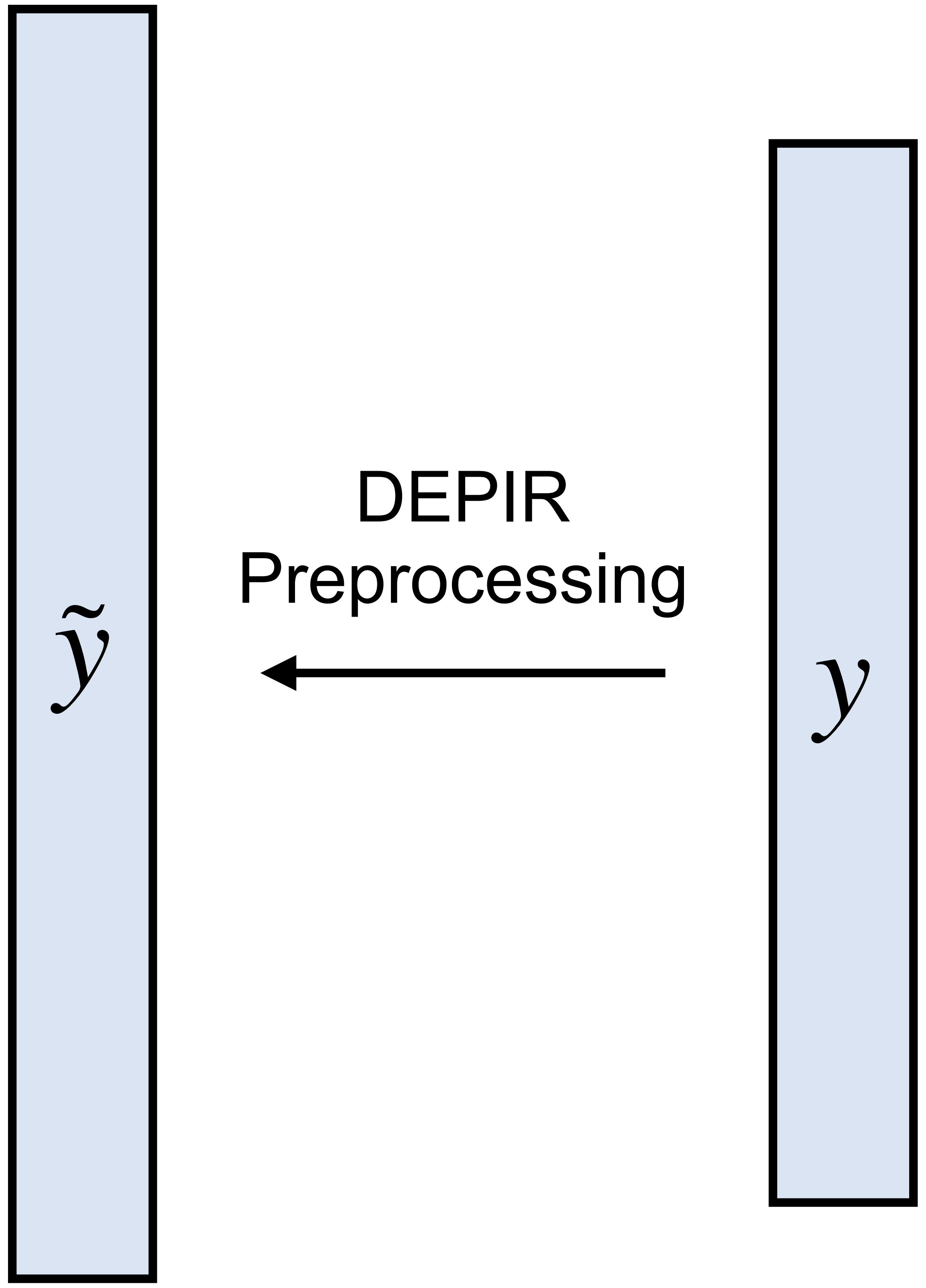
+ ORAM for the client’s database x



Full security with DEPIR

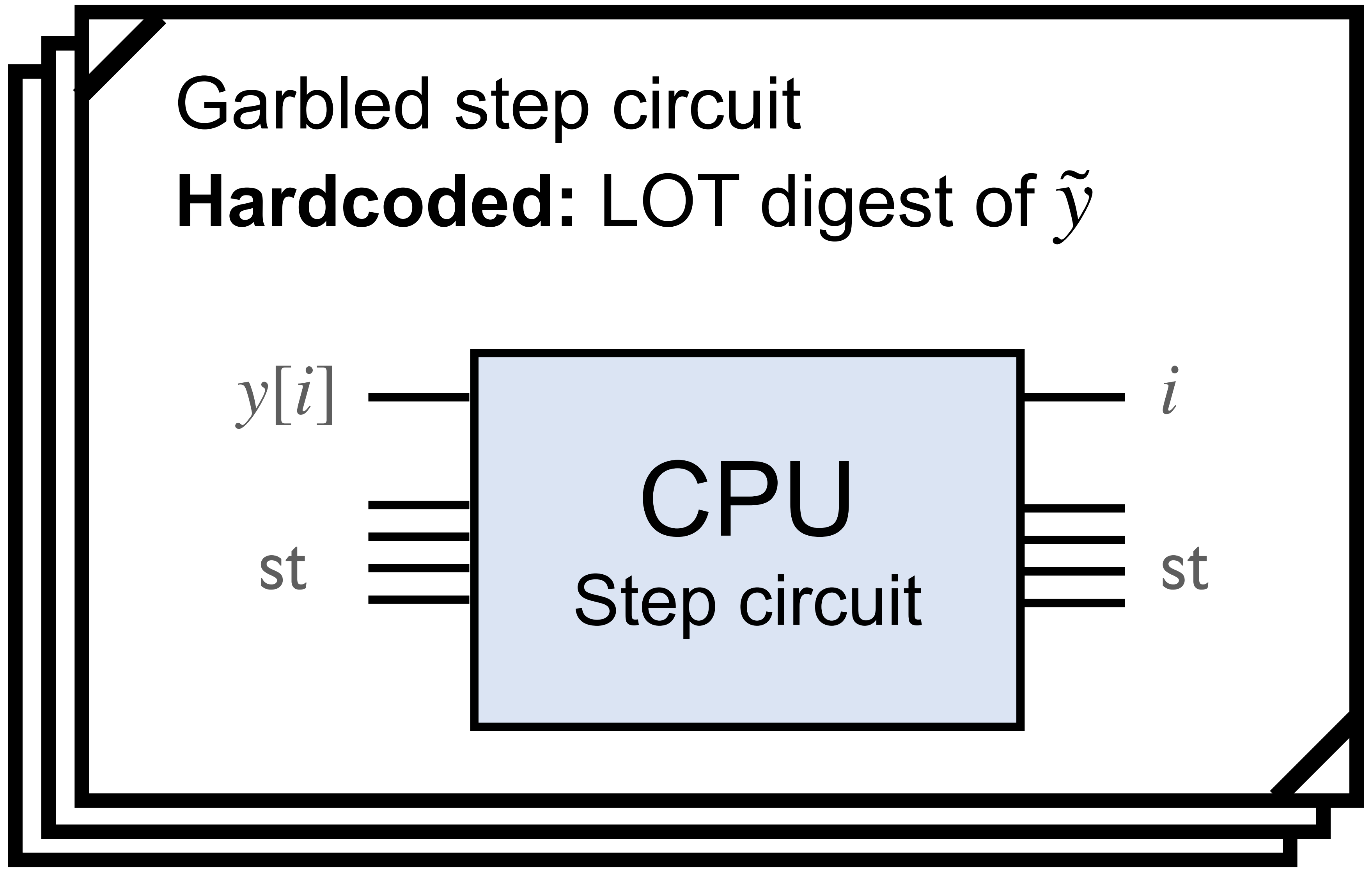
+ ORAM for the client’s database x

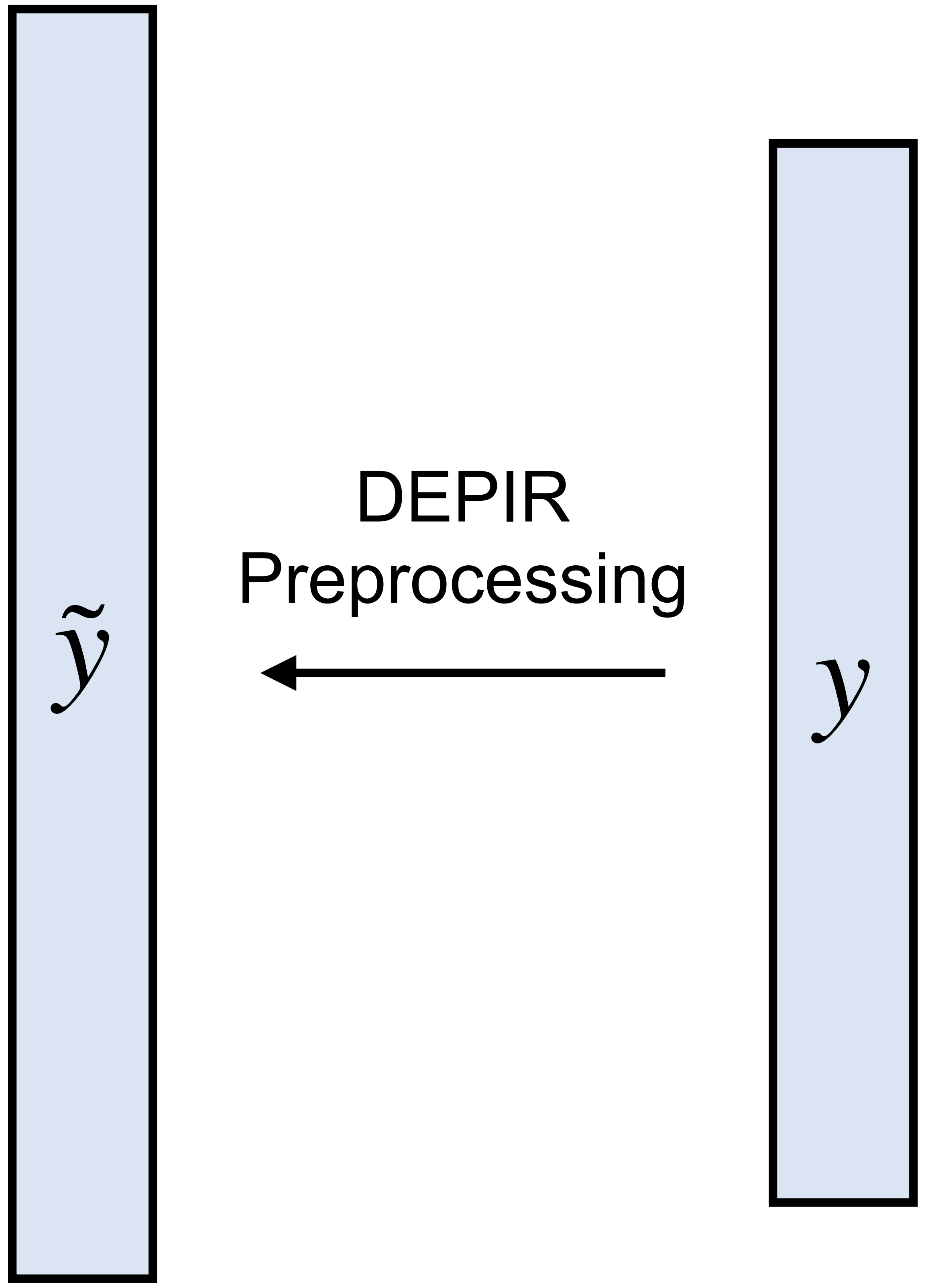




Full security with DEPIR

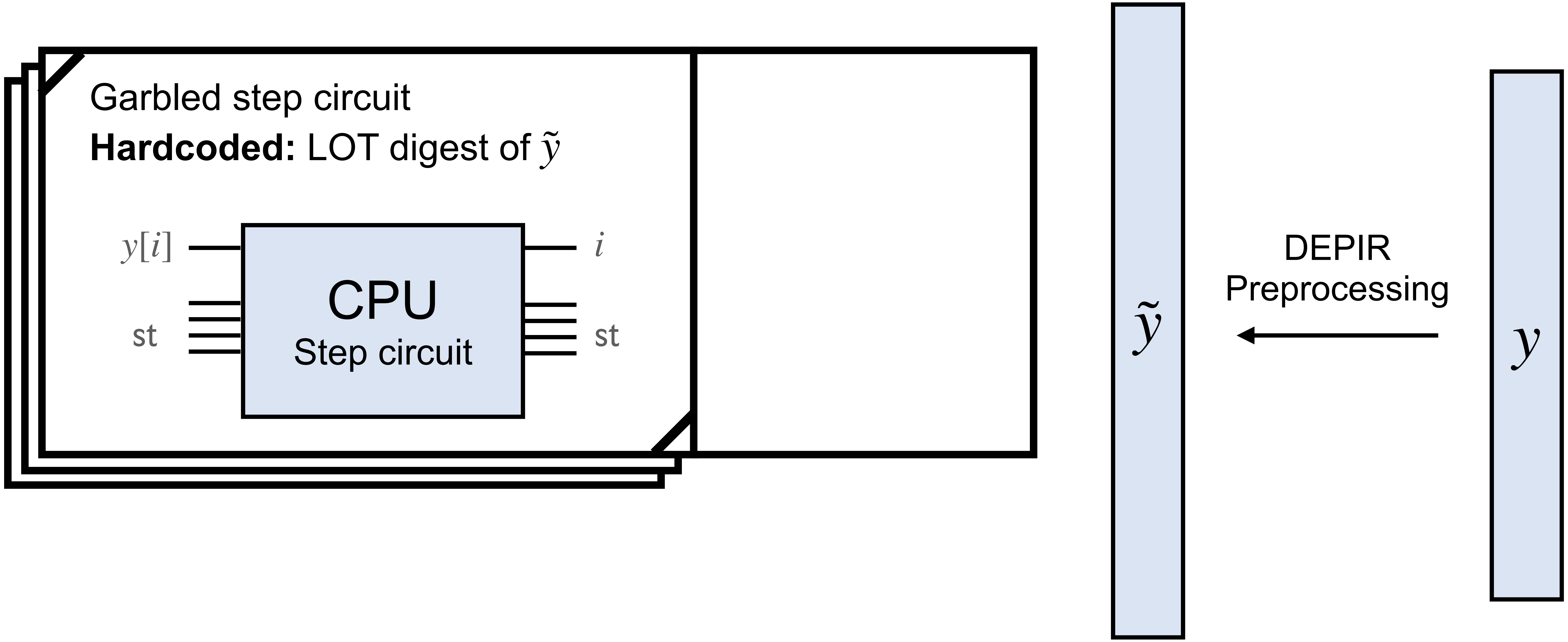
+ ORAM for the client's database x





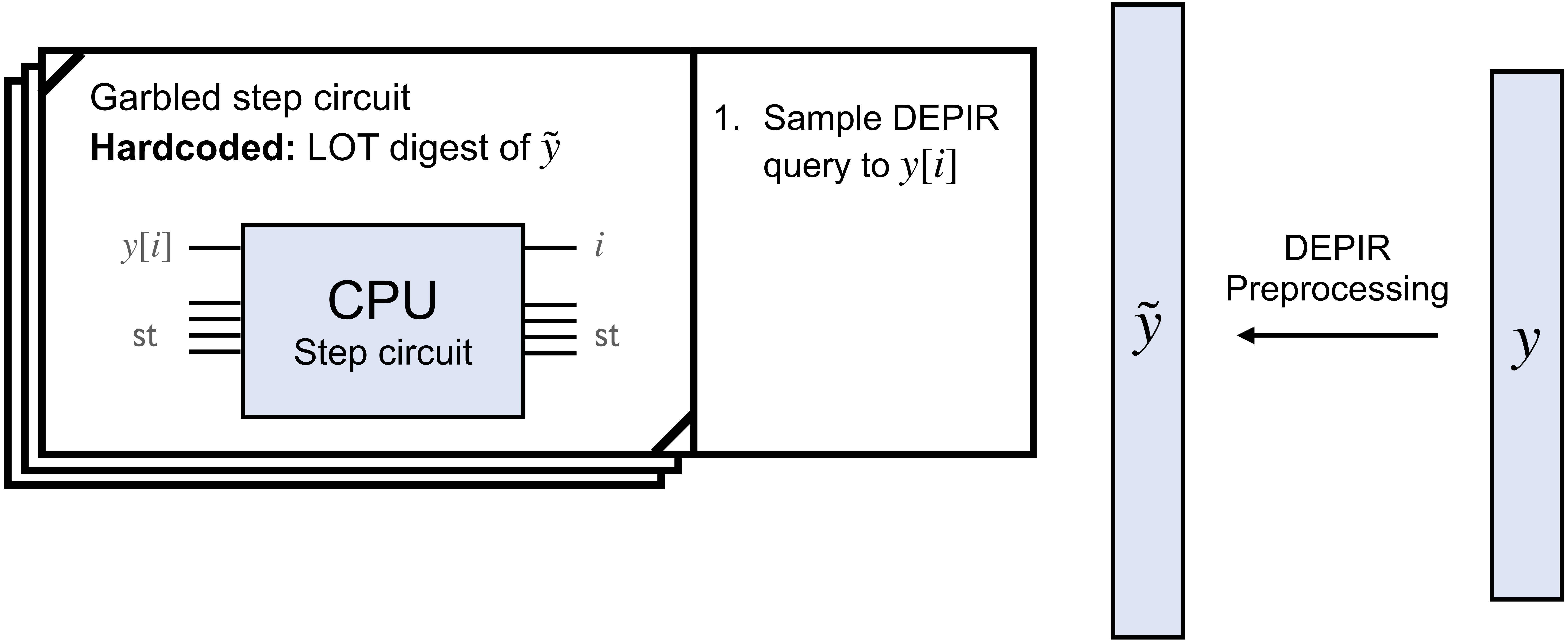
Full security with DEPIR

+ ORAM for the client's database x



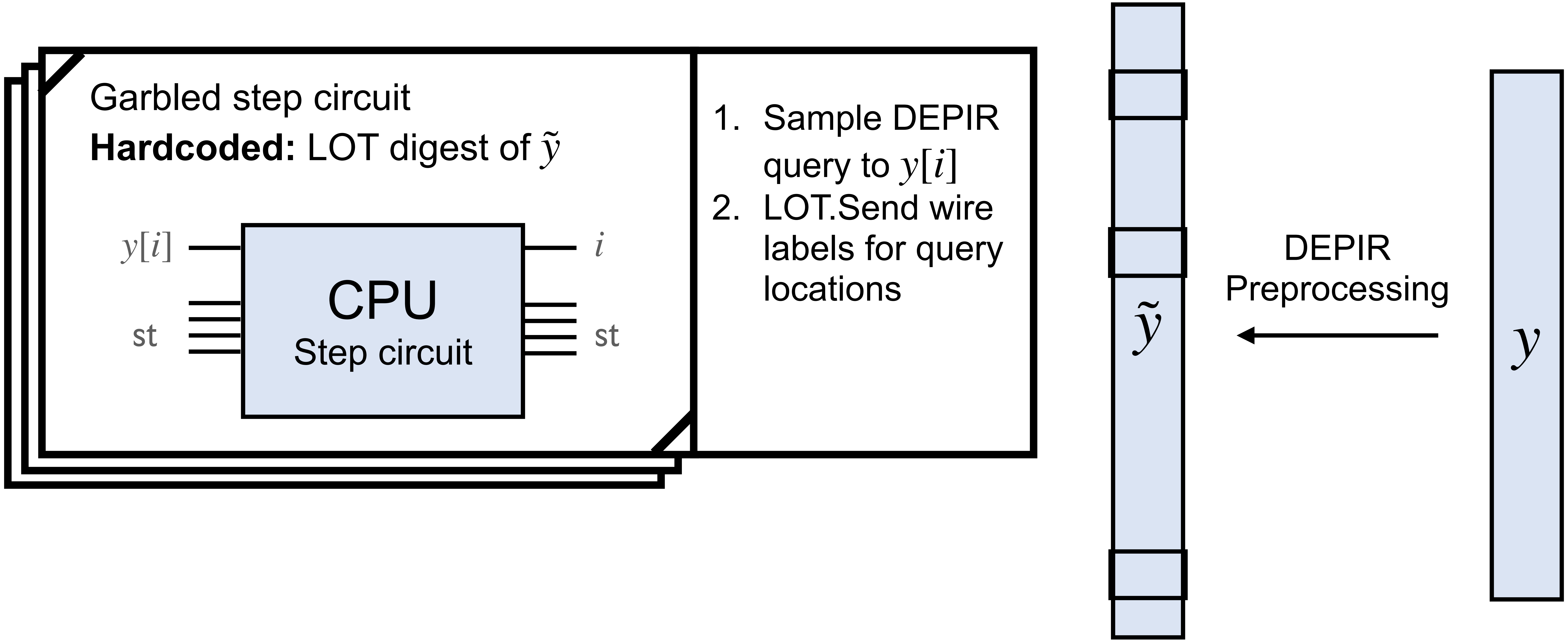
Full security with DEPIR

+ ORAM for the client's database x



Full security with DEPIR

+ ORAM for the client's database x



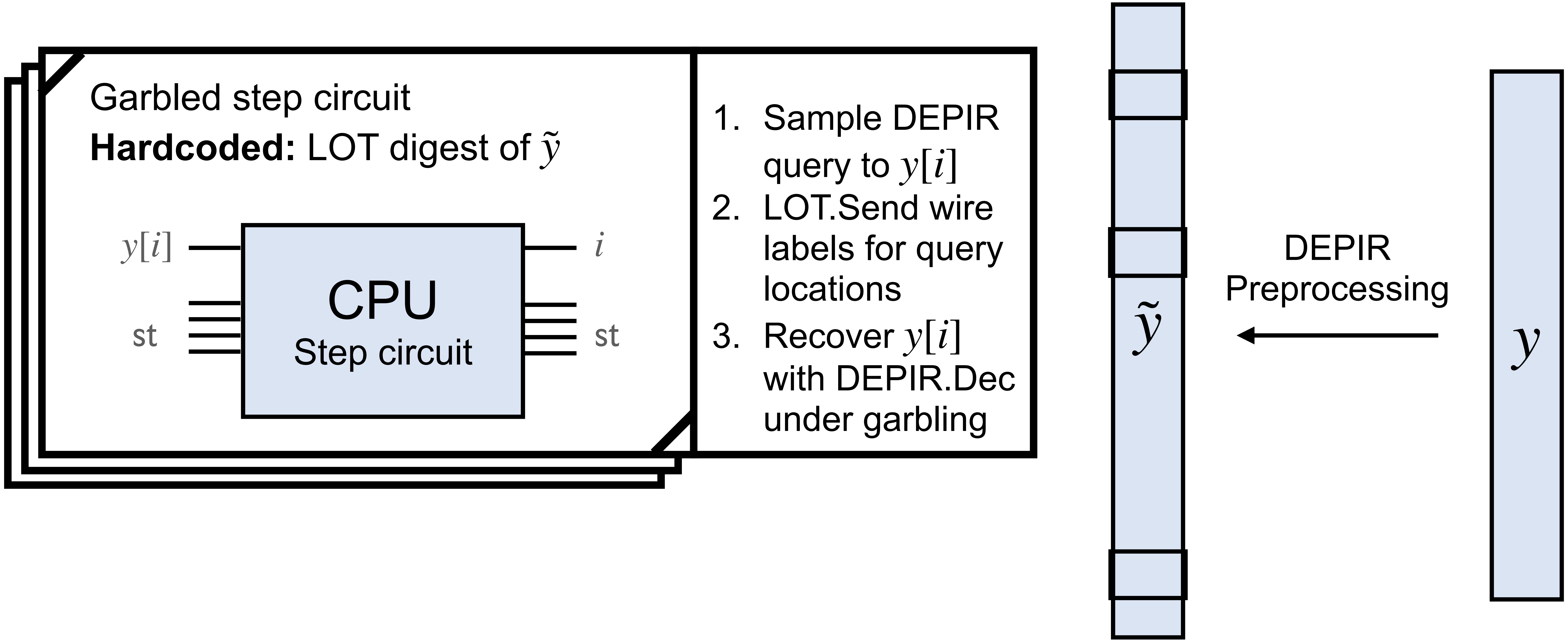
Garbled step circuit

Hardcoded: LOT digest of $\tilde{y}$

1. Sample DEPIR query to $y[i]$
2. LOT.Send wire labels for query locations

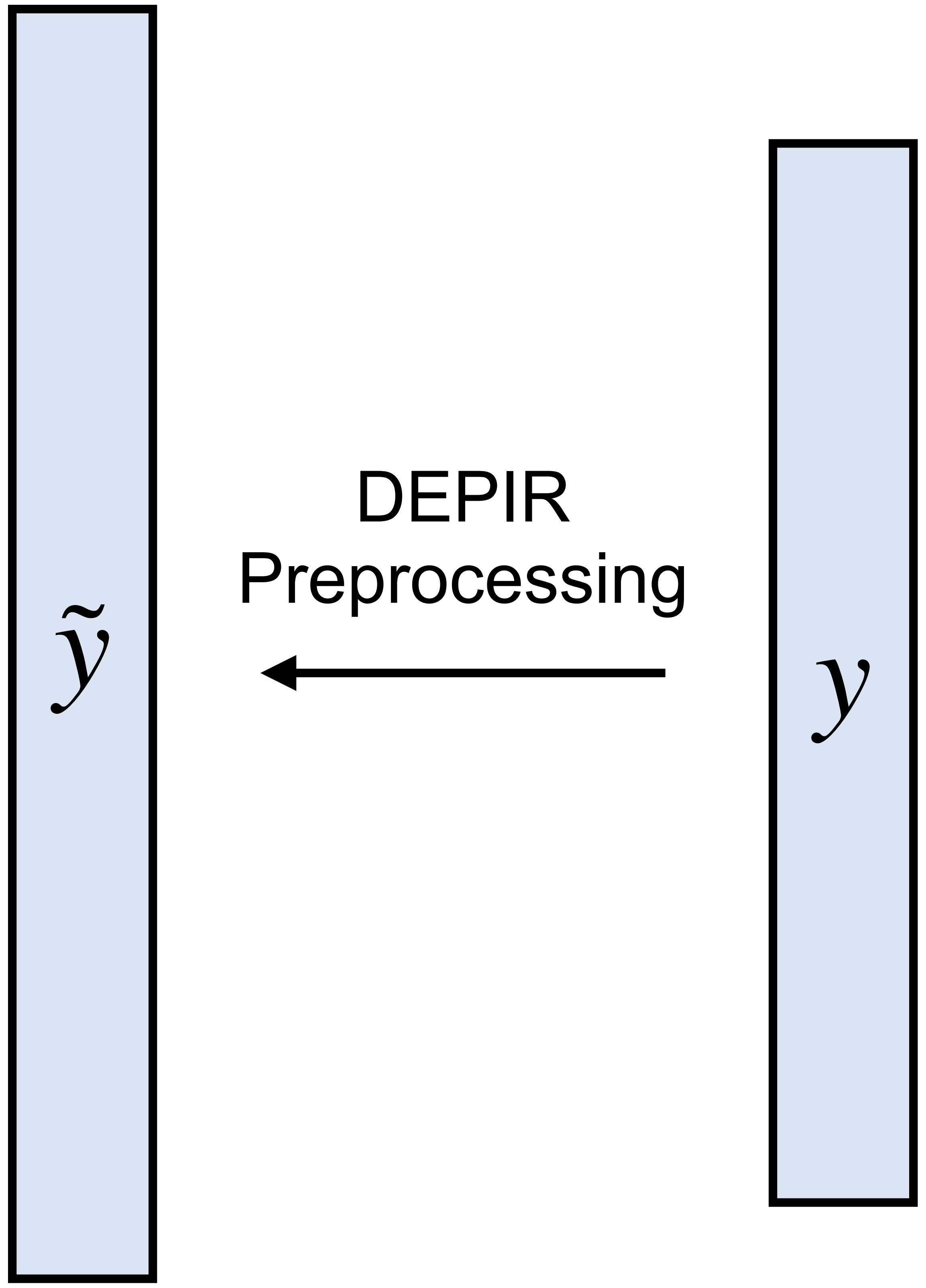
Full security with DEPIR

+ ORAM for the client's database x

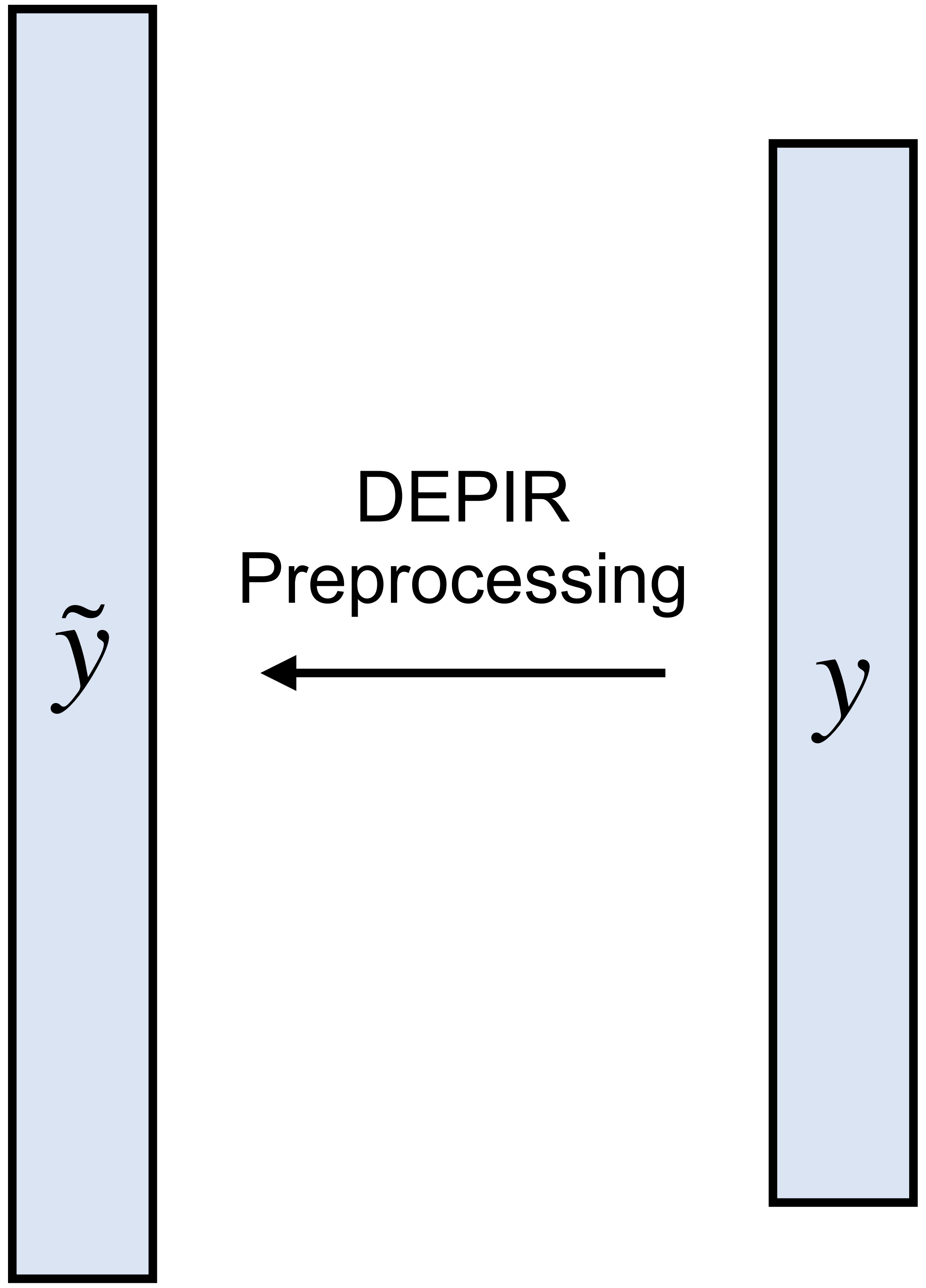


1. Sample DEPIR query to $y[i]$
2. LOT.Send wire labels for query locations
3. Recover $y[i]$ with DEPIR.Dec under garbling

Strong Efficiency with iO



Strong Efficiency with iO



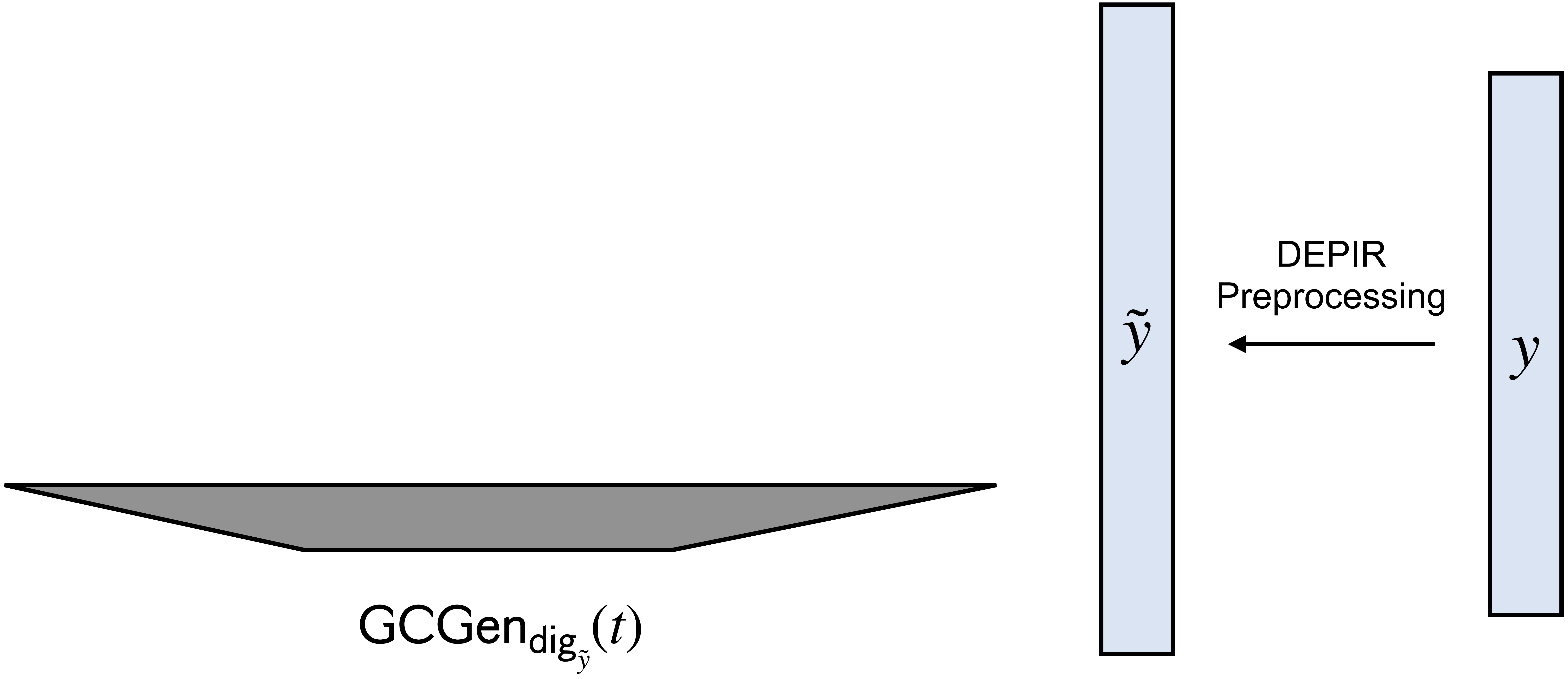
$\tilde{y}$

DEPIR
Preprocessing

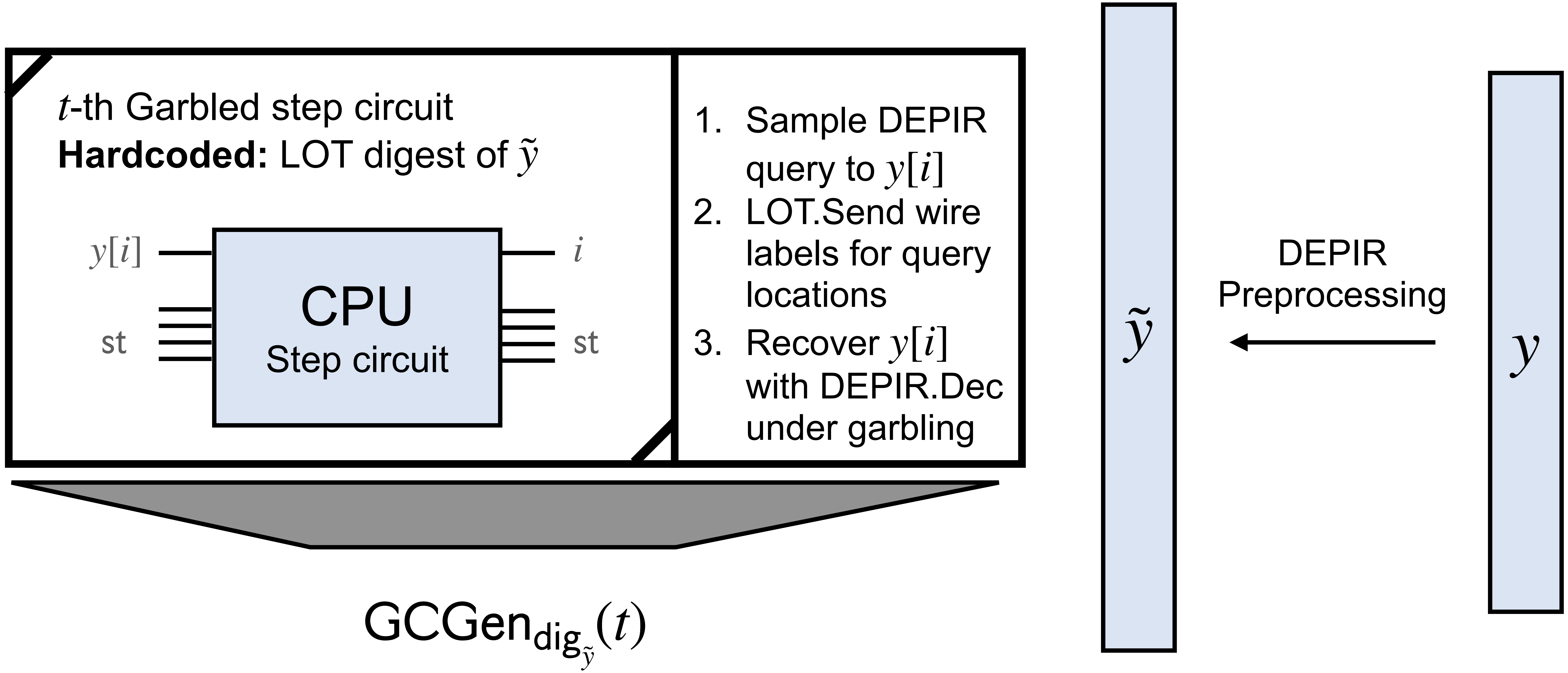
y

$\mathsf{GCGen}_{\mathsf{dig}_{\tilde{y}}}(t)$

Strong Efficiency with iO



Strong Efficiency with iO

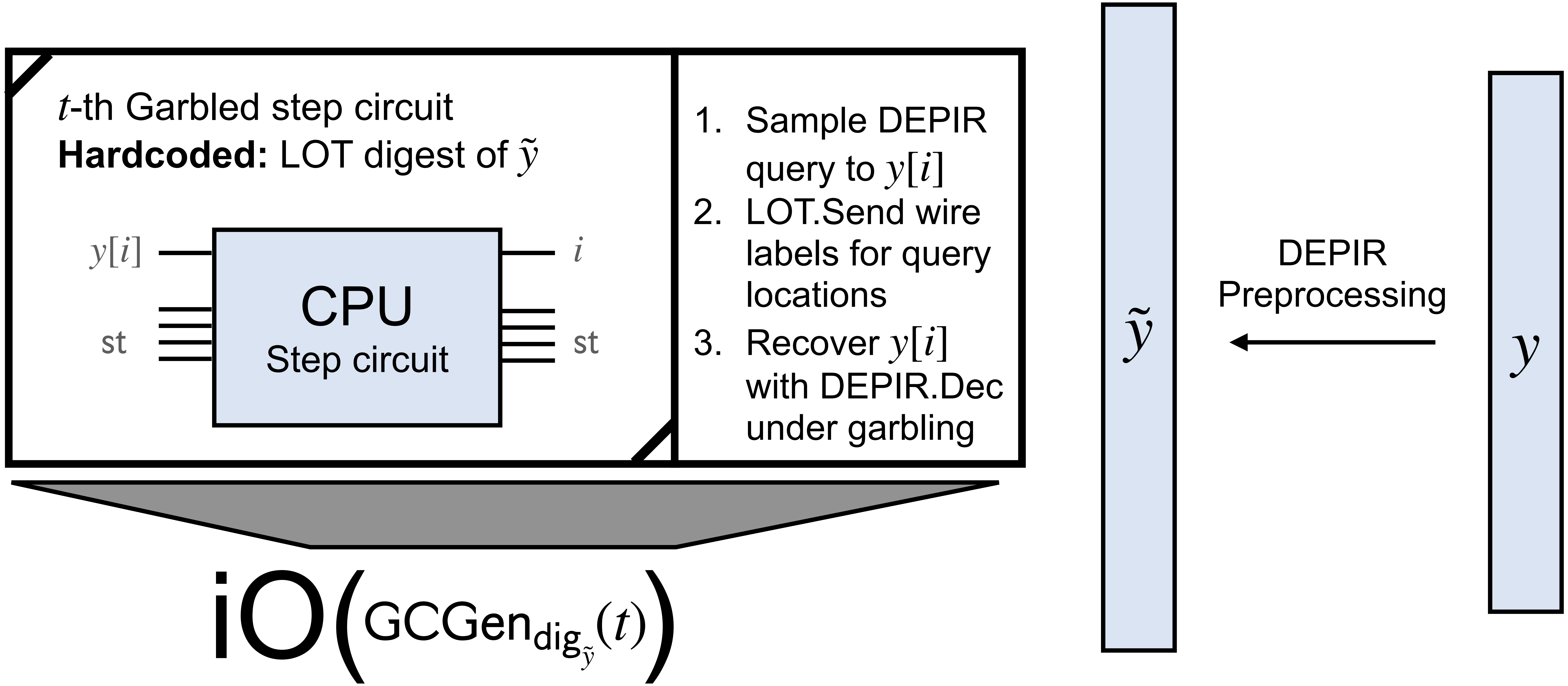


t-th Garbled step circuit

Hardcoded: LOT digest of $\tilde{y}$

1. Sample DEPIR query to $y[i]$
2. LOT.Send wire labels for query locations
3. Recover $y[i]$ with DEPIR.Dec under garbling

Strong Efficiency with iO



t-th Garbled step circuit

Hardcoded: LOT digest of $\tilde{y}$

1. Sample DEPIR query to $y[i]$
2. LOT.Send wire labels for query locations
3. Recover $y[i]$ with DEPIR.Dec under garbling

$\mathrm{iO}(\mathsf{GCGen}_{\mathsf{dig}_{\tilde{y}}}(t))$

Additional Results

Additional Results

<u>**Result**</u>**:** We build (multi-key) functional encryption for RAMs

Additional Results

<u>Result</u>: We build (multi-key) functional encryption for RAMs

- Each secret key associated to large database y

Additional Results

<u>Result</u>: We build (multi-key) functional encryption for RAMs

- Each secret key associated to large database y
- Decryption recovers $P(x, y)$ in sublinear time in $|x|, |y|$

Additional Results

Result: We build (multi-key) functional encryption for RAMs

- Each secret key associated to large database y
- Decryption recovers $P(x, y)$ in sublinear time in $|x|, |y|$

Assumptions: FE for circuits + RingLWE

Additional Results

<u>Result</u>: We build (multi-key) functional encryption for RAMs

- Each secret key associated to large database y
- Decryption recovers $P(x, y)$ in sublinear time in $|x|, |y|$

Assumptions: FE for circuits + RingLWE

Prior work: [ACFQ’22] only allows short secret keys

Additional Results

Result: We build (multi-key) functional encryption for RAMs

- Each secret key associated to large database y
- Decryption recovers $P(x, y)$ in sublinear time in $|x|, |y|$

Assumptions: FE for circuits + RingLWE

Prior work: [ACFQ'22] only allows short secret keys

Result: We build iO for RAMs

Additional Results

<u>Result</u>: We build (multi-key) functional encryption for RAMs

- Each secret key associated to large database y
- Decryption recovers $P(x, y)$ in sublinear time in $|x|, |y|$

Assumptions: FE for circuits + RingLWE

Prior work: [ACFQ’22] only allows short secret keys

<u>Result</u>: We build iO for RAMs

- Given (P, y), obfuscate the program $P(\cdot, y)$

Additional Results

Result: We build (multi-key) functional encryption for RAMs

- Each secret key associated to large database y
- Decryption recovers $P(x, y)$ in sublinear time in $|x|, |y|$

Assumptions: FE for circuits + RingLWE

Prior work: [ACFQ'22] only allows short secret keys

Result: We build iO for RAMs

- Given (P, y), obfuscate the program $P(\cdot, y)$
- Evaluation can be sublinear in $|y|$

Additional Results

Result: We build (multi-key) functional encryption for RAMs

- Each secret key associated to large database y
- Decryption recovers $P(x, y)$ in sublinear time in $|x|, |y|$

Assumptions: FE for circuits + RingLWE

Prior work: [ACFQ'22] only allows short secret keys

Result: We build iO for RAMs

- Given (P, y), obfuscate the program $P(\cdot, y)$
- Evaluation can be sublinear in $|y|$

Assumptions: iO for circuits + RingLWE

Additional Results

Result: We build (multi-key) functional encryption for RAMs

- Each secret key associated to large database y
- Decryption recovers $P(x, y)$ in sublinear time in $|x|, |y|$

Assumptions: FE for circuits + RingLWE

Prior work: [ACFQ’22] only allows short secret keys

Result: We build iO for RAMs

- Given (P, y), obfuscate the program $P(\cdot, y)$
- Evaluation can be sublinear in $|y|$

Assumptions: iO for circuits + RingLWE

Prior work: [BCGHJLPTV’18] doesn’t allow sublinear runtime

Thank you!

eprint: 2024/068