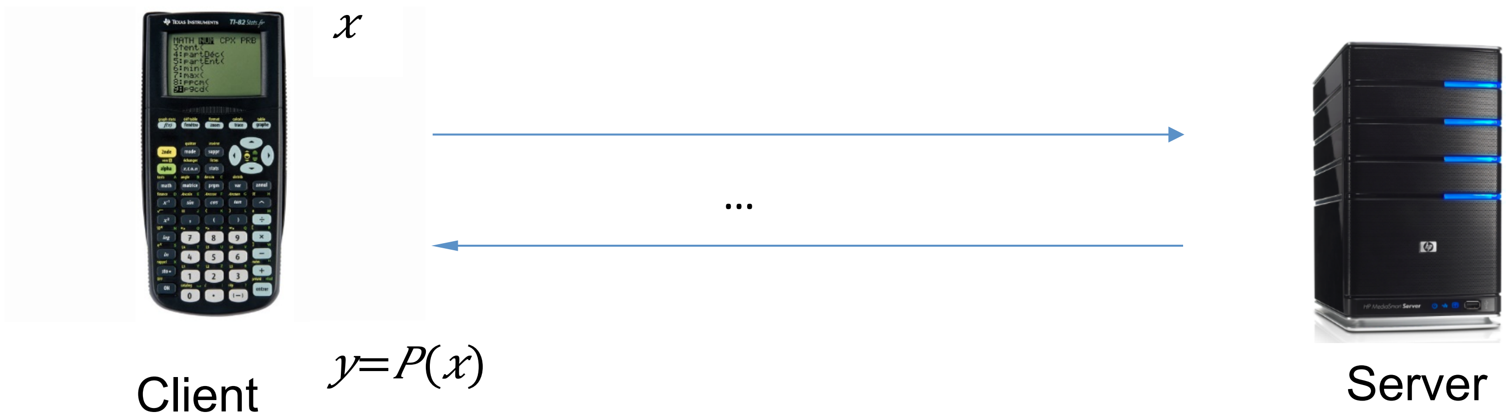
Outsourcing RAM Computation

Daniel Wichs
Northeastern University

Mainly based on joint works with:
Craig Gentry, Shai Halevi, Mariana Raykova

Problem Overview



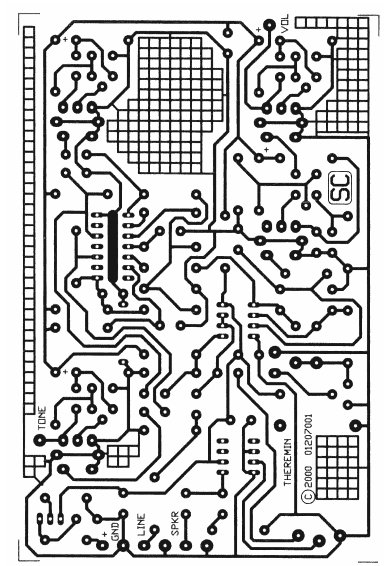
- Weak client wants to leverage resources of a powerful server to compute $P(x)$ without revealing x.

- Efficiency Requirements:
  – Client does much less work than computing $P(x)$
  – Server does about as much work as computing $P(x)$

Use FHE! Done?

- Private outsourcing is possible using Fully Homomorphic Encryption (FHE). [RAD78,Gen09,…]

- But FHE works over *circuits* rather than *RAM programs*.



Circuits vs. RAM

- Private outsourcing is possible using Fully Homomorphic Encryption (FHE). [RAD78,Gen09,...]

- But FHE works over *circuits* rather than *RAM programs*.
  - RAM complexity $T \Rightarrow$ circuit or TM complexity $T \uparrow 2$
  - For programs with initial "data in memory", efficiency gap can be exponential (e.g., Google search).

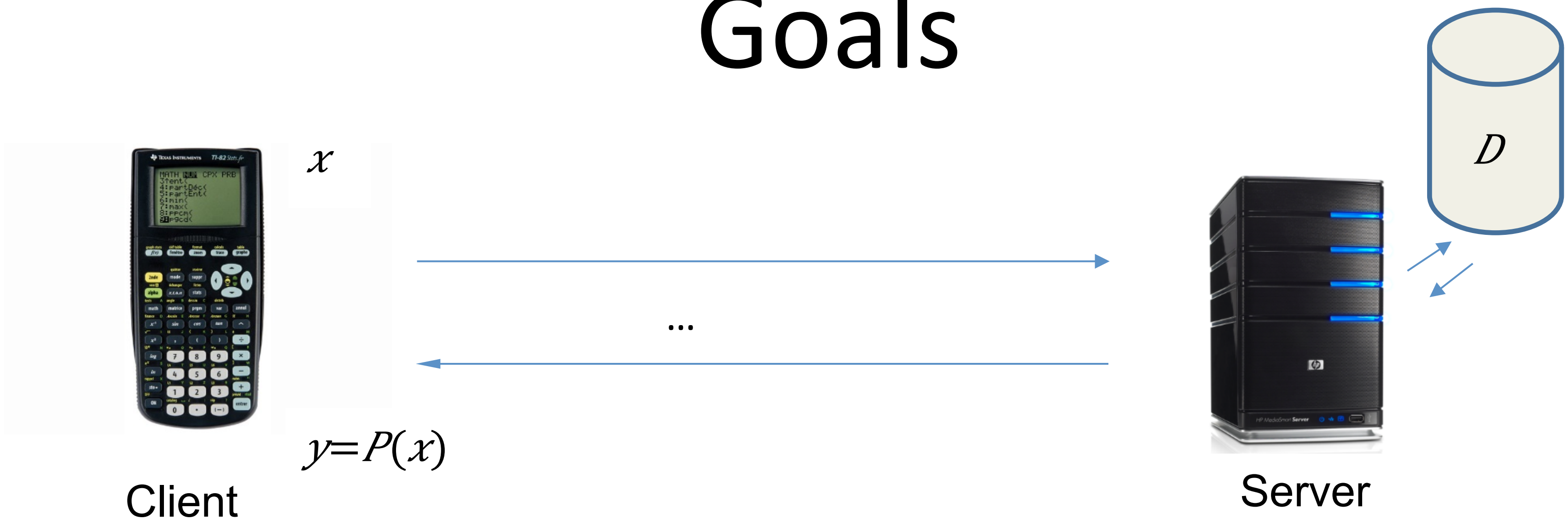
- Could use ORAM, but then client does all the work.

Goals

- Client's work: $O(|x|+|y|)$
- Server's work: O(RAM run-time of P).

- May allow client pre-processing of P.
  - Client does one-time computation in O(RAM run-time of P).
  - Later, outsource many executions of P. Amortized efficiency.

Goals



- Basic scenario: client wants to run independent executions of P on inputs x_1, x_2, x_3,...

- Persistent Memory Data:
  - Client initially outsources large private ‘memory data’ D.
  - Program executions $P^D(x_i)$ can read/write to D.

Goals

Client

Server

- Non-interactive solution: “reusable garbled RAM”.

Garbled Computation

Persistent Memory Data:

Garble Data: $D \rightarrow D$

Can execute many programs with read/write access to data.

Garbled RAM

[LO13, GHLORW14, GLOS15]

Garble RAM: $P \rightarrow P$

Garble input: $x \rightarrow x$

Size of P, run-time $P(x)$ is O(RAM run-time P).

Reusable Garbled RAM

[GHRW14, CHJV14,…]

Can garble many inputs per program.

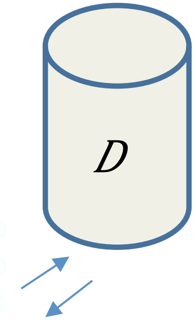
Efficiently outsource RAM comp.

Outsourcing via Reusable G-RAM

Client

x_i

P



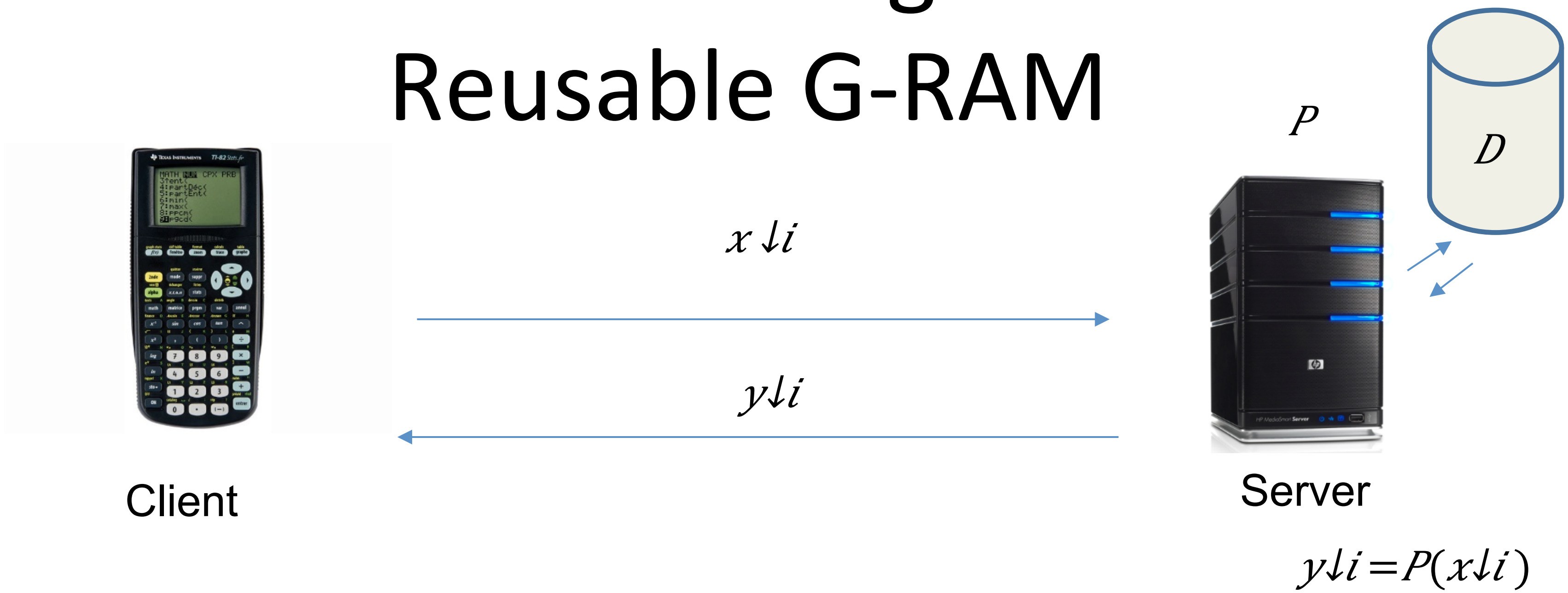
Server

$y_i = P(x_i)$

- Client garbles program $P \to P$ [data $D \to D$].
  - Pre-processing = O(run-time P)

- Client repeatedly garbles inputs $x_i \to x_i$.
- Server evaluates P on x_i to get y_i [using D]
  - Evaluation time = O(run-time P)

Outsourcing via Reusable G-RAM



- *Client learns* y_i. Server sends it back (+1 round = optimal).
- *Output privacy:* set y_i = encryption of real output. Server sends back y_i.
- *Verifiability:* y_i includes (one-time) MAC of real output.

Garbled RAM

Garbled RAM

PART I

- Overview of [LO13].
- Circularity issue, fixes.

Reusable Garbled RAM

PART II

Combine:

- Non-reusable garbled RAM.
- Obfuscation.

PART I

One-Time Garbled RAM

Garbled RAM Definition

without persistent data

Client

secret: k

GProg$(P,k) \rightarrow P$

GInput$(x, k) \rightarrow x$

Server

Eval$(P, x) \rightarrow y$

Garbled RAM Definition

with persistent data

GData $(D,k) \to D$

GProg $(P,k,i) \to P_i$ $\approx$ O(run-time)

GInput $(x_i, k,i) \to x_i$

Client

secret: k

Server

Eval $^{D} (P_i, x_i) \to y_i$

$\approx$ O(run-time)

- **Security:** server only learns $y_1, y_2, \ldots$ (even data access pattern is hidden!)

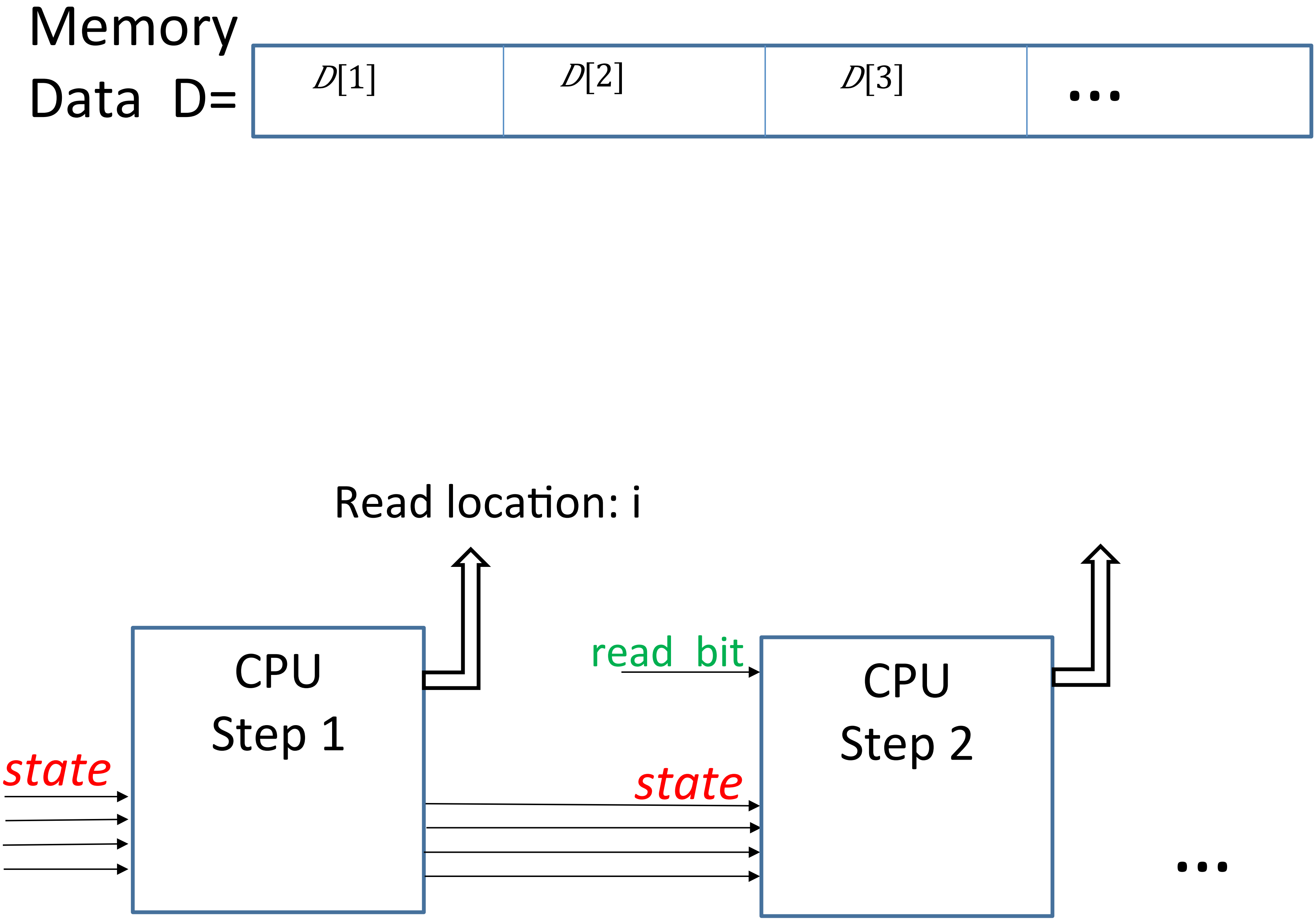
Weak vs. Full Security

- Weak security: May reveal data D, and *data-access pattern* of computations.
  - Locations of memory accessed in each step.
  - Values read and written to memory.

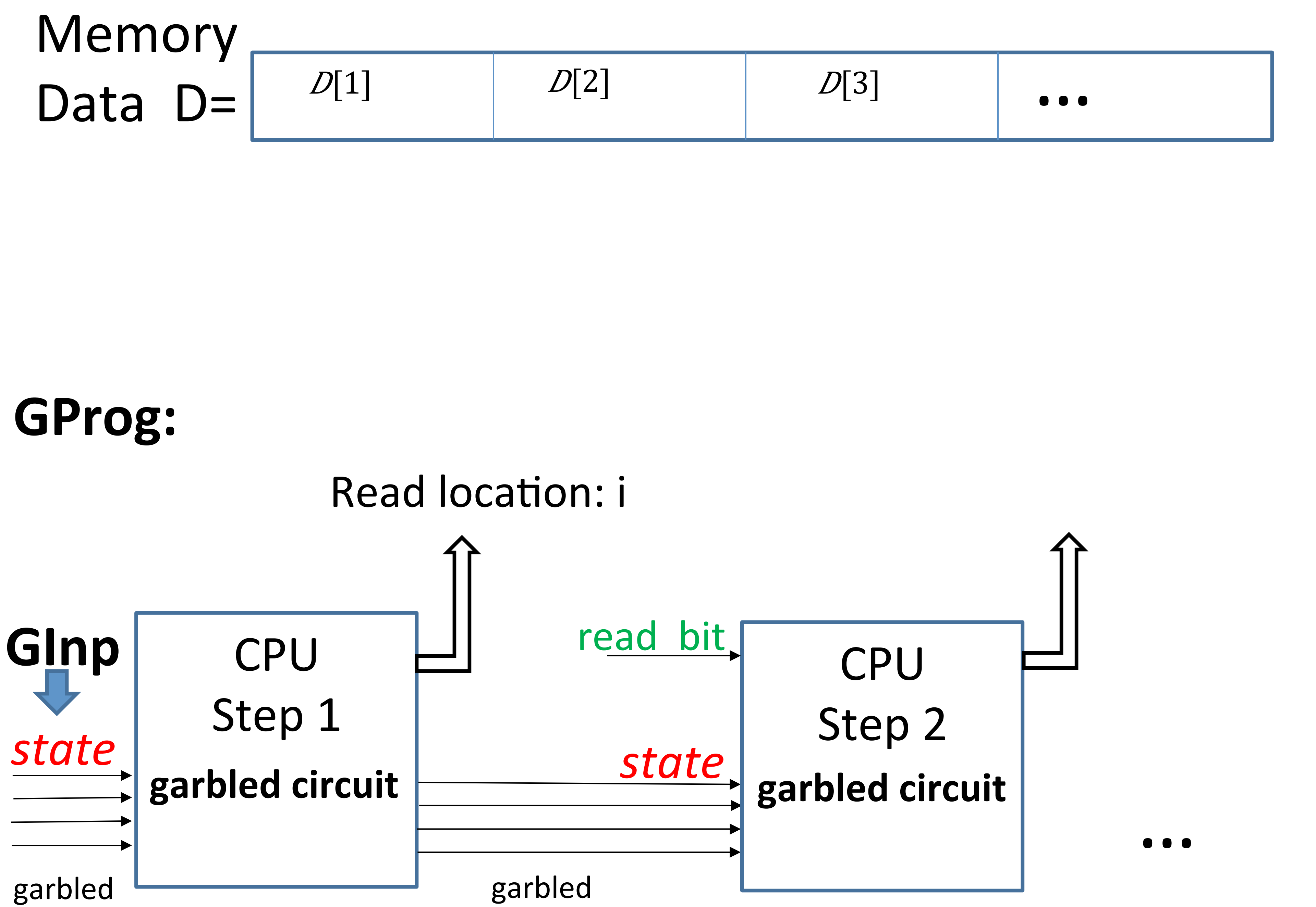
- Compiler: weak $\Rightarrow$ full security:
  - Use oblivious RAM [GO96,…] to encode/access memory.

Overview of [Lu-Ostrovsky 13]

For now, read-only computation.



Memory
Data D=
D[1]
D[2]
D[3]
...
Read location: i
CPU
Step 1
state
read bit
CPU
Step 2
state
...

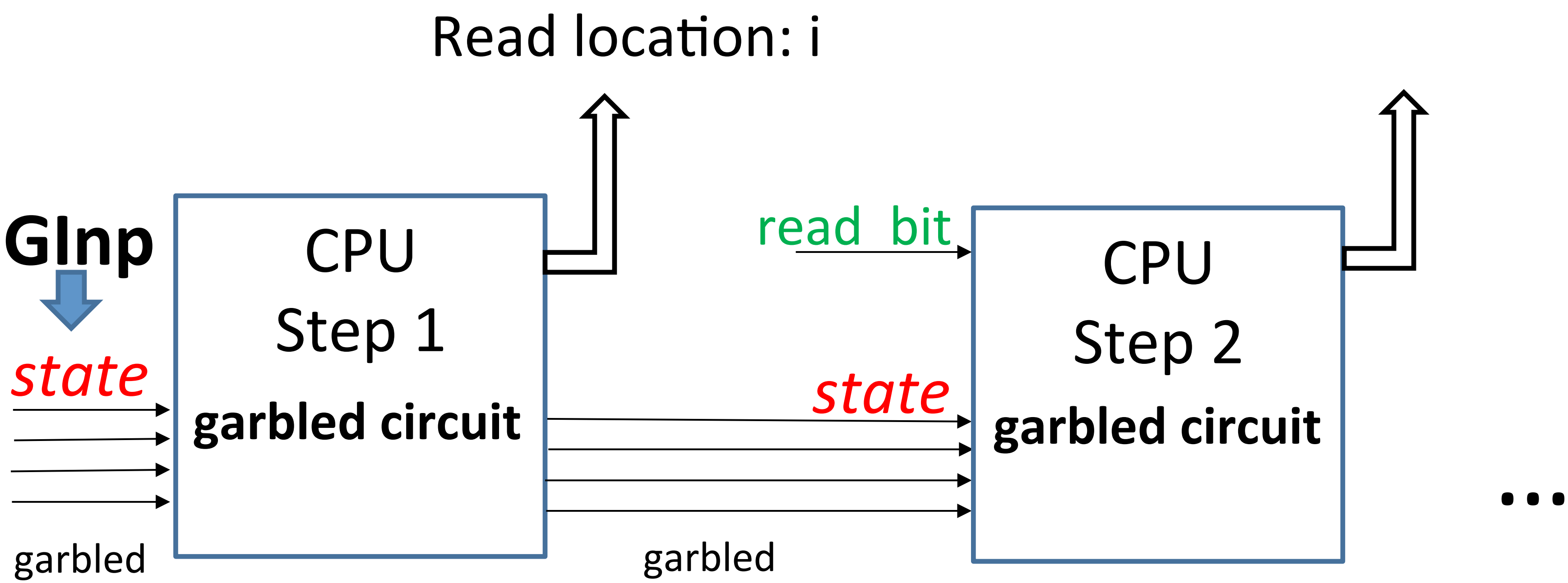

Memory
Data D=
$D[1]$
$D[2]$
$D[3]$
...
GProg:
Read location: i
GInp
CPU
Step 1
garbled circuit
state
garbled
read bit
state
CPU
Step 2
garbled circuit
garbled
...

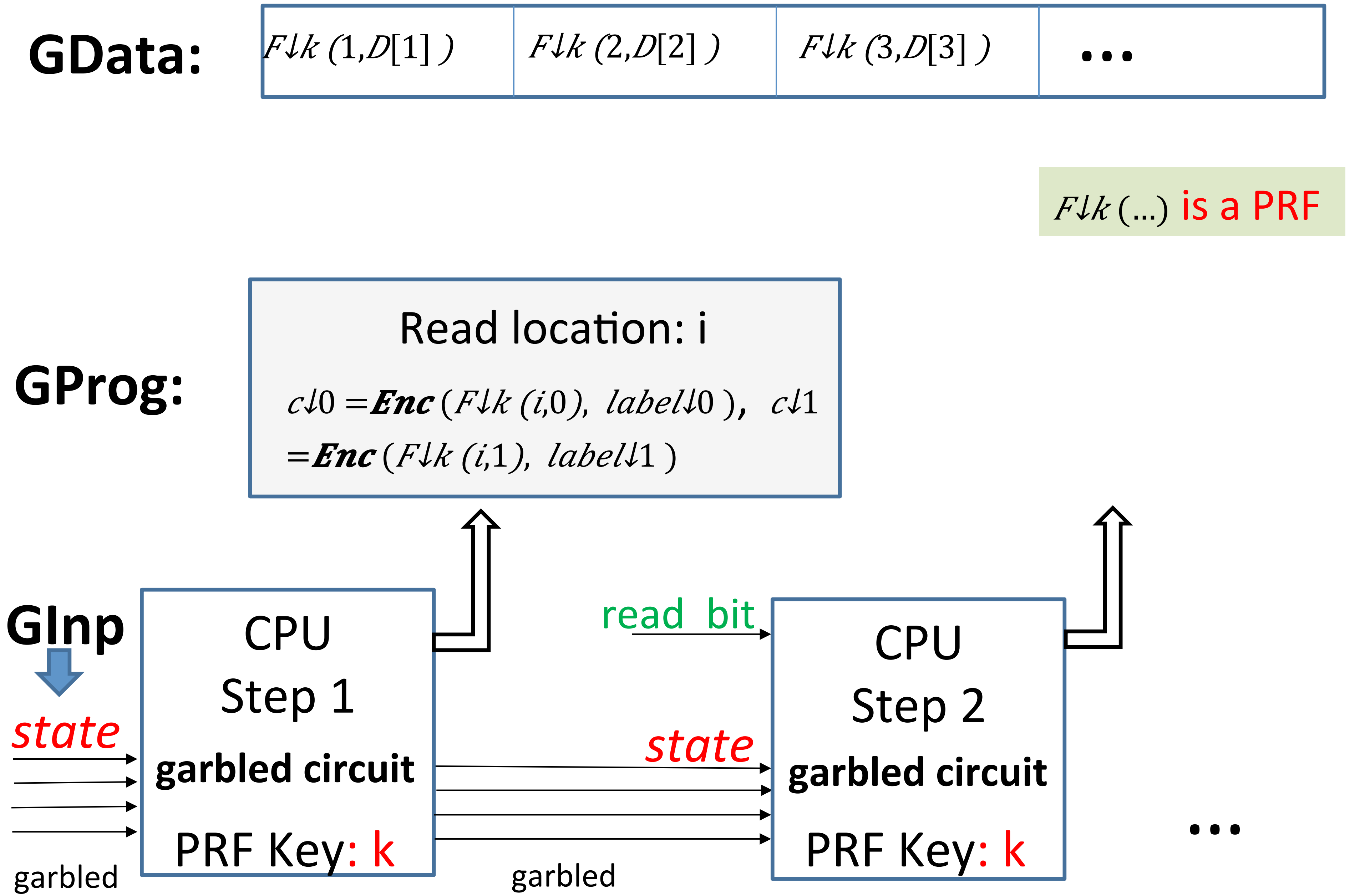
GData:

$F_k(1, D[1])$	$F_k(2, D[2])$	$F_k(3, D[3])$	…

$F_k(\ldots)$ is a PRF

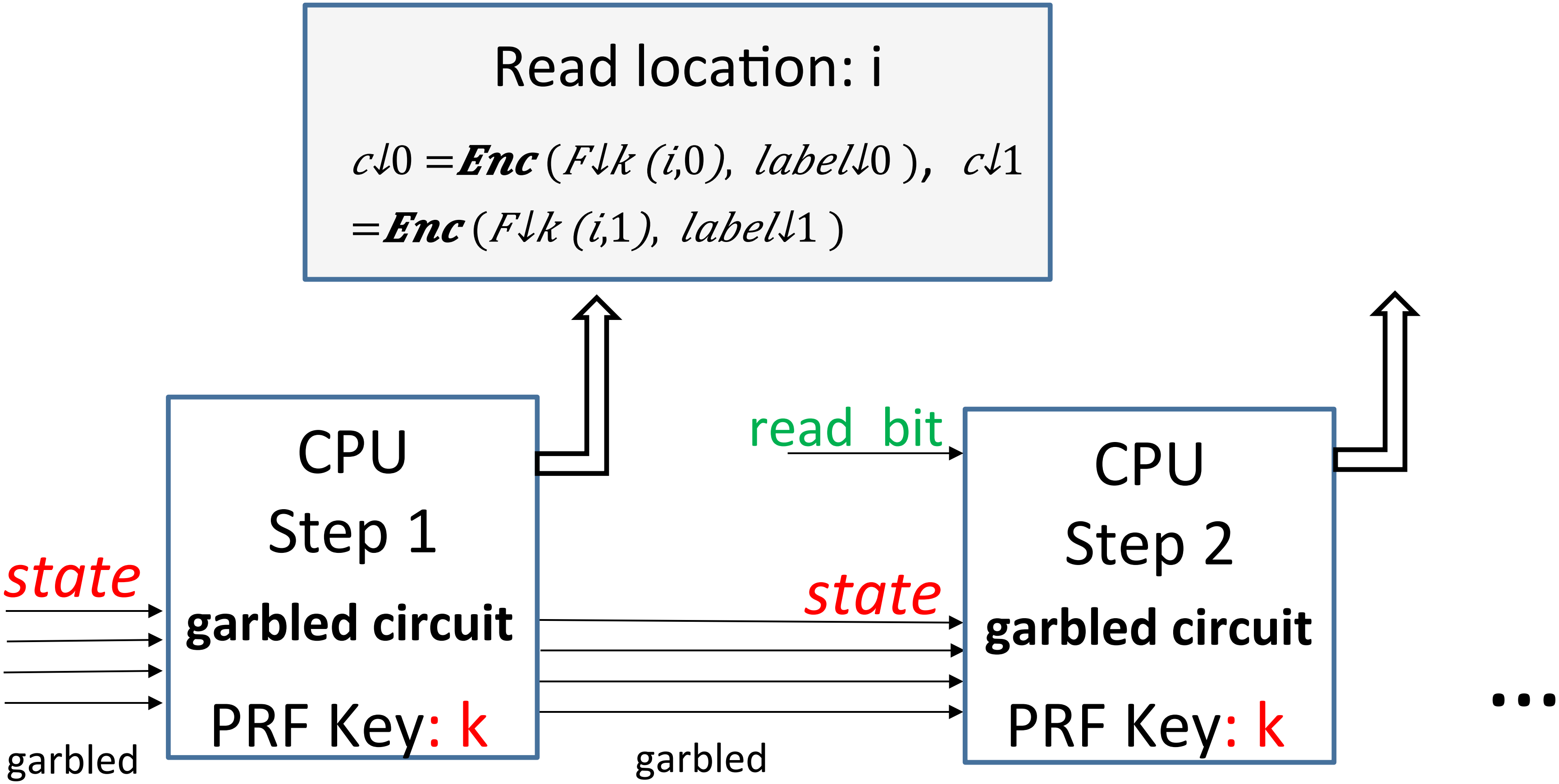
GProg:





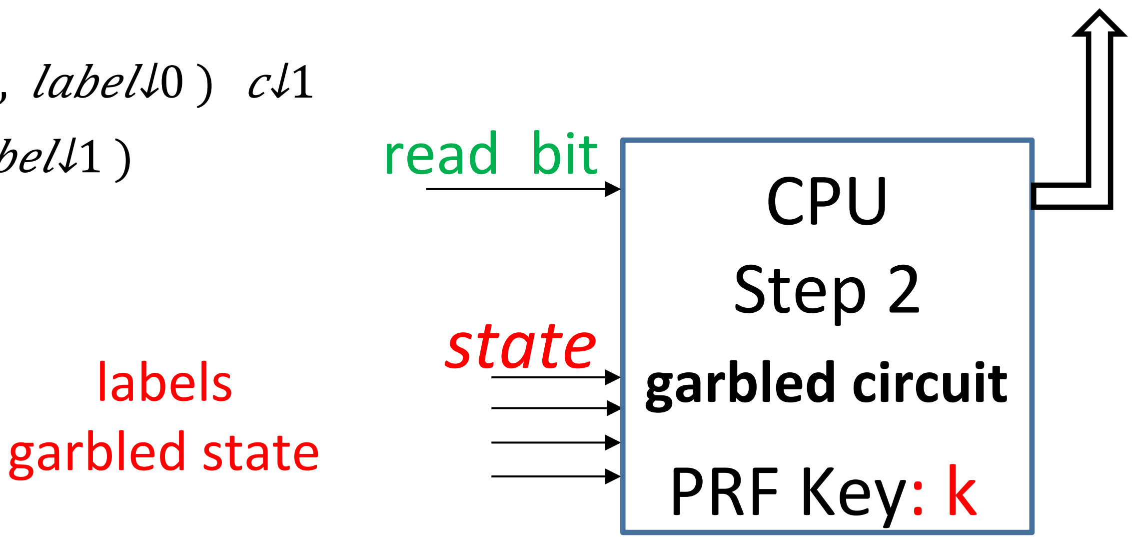
GData:
$F_k(1, D[1])$
$F_k(2, D[2])$
$F_k(3, D[3])$
...
$F_k(\dots)$ is a PRF
GProg:
Read location: i
$c_0 = \boldsymbol{Enc}(F_k(i,0),\ label_0),\ c_1 = \boldsymbol{Enc}(F_k(i,1),\ label_1)$
GInp
state
garbled
CPU
Step 1
garbled circuit
PRF Key: k
read bit
state
garbled
CPU
Step 2
garbled circuit
PRF Key: k
...

Let's try to prove security…



Use security of 1st garbled circuit only learn output

$c_0 = \boldsymbol{Enc}(F_k(i,0),\ label_0)$ $c_1 = \boldsymbol{Enc}(F_k(i,1),\ label_1)$

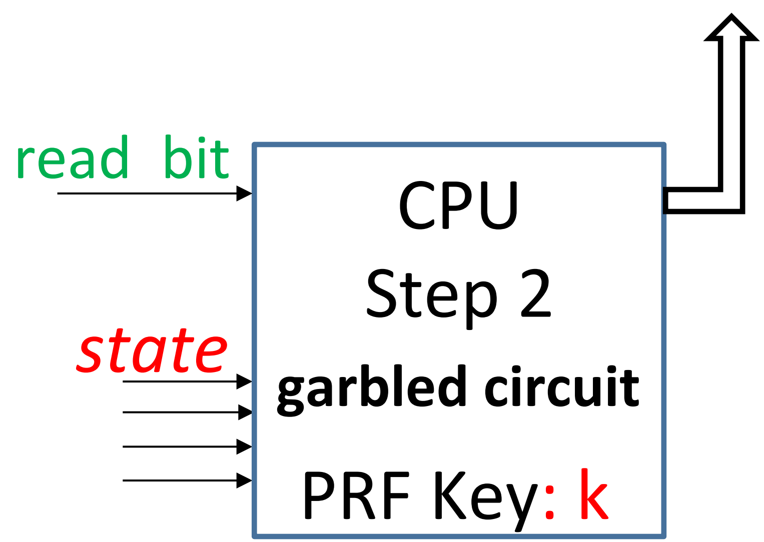


…

Use security of 1st garbled circuit only learn output (assume D[i]=1)

$c_0 = \boldsymbol{Enc}(F_k(i,0),\ label_0)$

$label_1$



labels
garbled state

…

Use security of 2nd garbled circuit

don't learn

$label_0$ for read bit

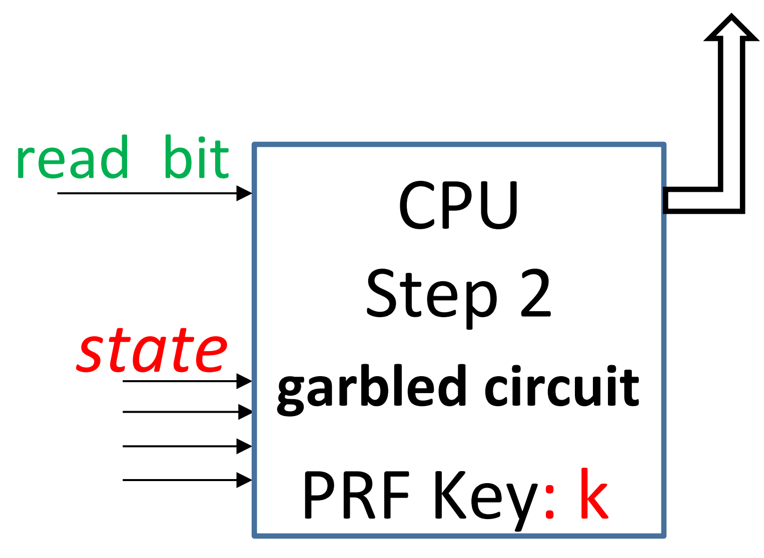
don't learn

PRF key k

Use security of Encryption/PRF

$c_0 = \boldsymbol{Enc}(F_k(i,0),\ label_0)$

$label_1$



labels

garbled state

…

Circularity* Problem!

* May appear rectangular

So is it secure?

- Perhaps, but…
  - No proof.
  - No “simple” circularity assumption on one primitive.

Can we fix it? Yes!

- Fix 1: Using identity-based encryption (IBE).
  [GHLORW14]

- Fix 2: Evolving key, "key revocation" (OWF).
  [GHLORW14], [GLOS15]

PART II

Reusable Garbled RAM

Main Results

1-time Garbled RAM

+

Reusable Garbled Circuits
(obfuscation)

→

Reusable Garbled RAM

Reusable Garbled RAM Definition

without persistent data

View can be simulated given y_1, y_2 …

GProg$(P,k) \to P$

{ GInput$(x_i, k) \to x_i$ **: i=1,…}**

Client

secret: k

Server

Eval$(P, x_i) \to y_i$

Reusable Garbled RAM Definition

with persistent data

View can be simulated given $y_1, y_2, \ldots$

GData $(D,k) \to D$

GProg $(P,k) \to P$

GInput $(x_i, k, i) \to x_i$

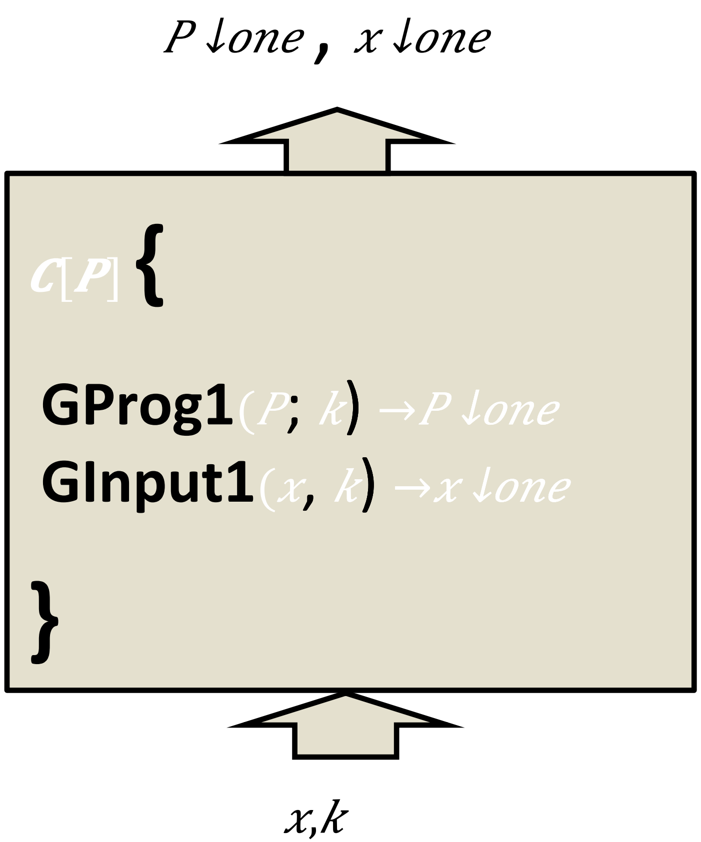
Client

secret: k

Server

Eval$^{D}(P, x_i) \to y_i$

- **Construct** reusable garbled RAM by combining:
  - one-time garbled RAM (GProg1, GInput1, GEval1)
  - reusable garbled circuits



Reusable Gprog $P \rightarrow P\downarrow reuse$

reusable circuit-garbling of $C[P]$

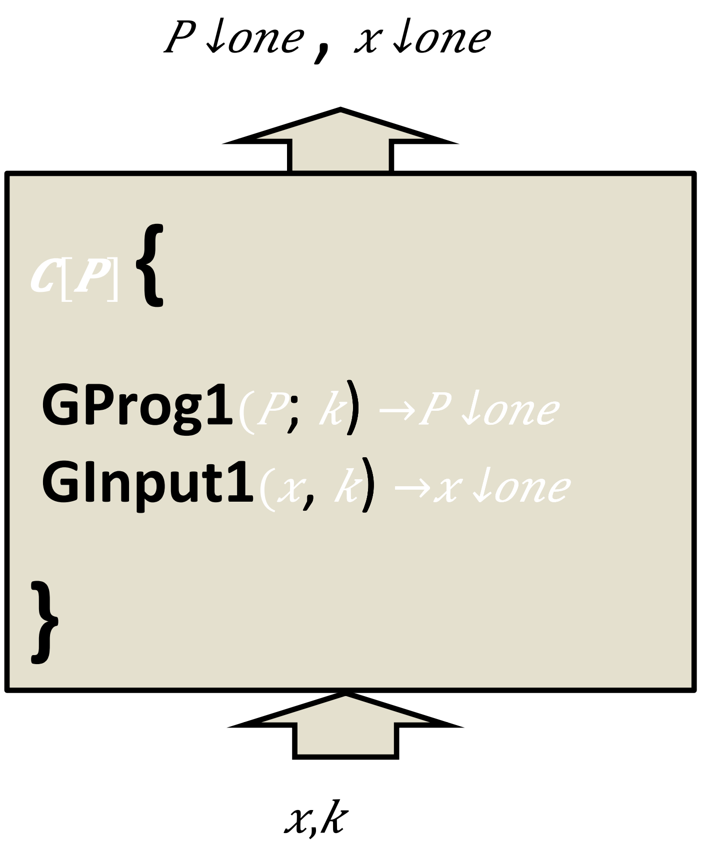
Reusable Ginput $x \rightarrow x\downarrow$

Choose fresh one-time key k

garble input (x,k) for $C[P]$

- Size of $C[P]$ = (RAM run-time of P)
- |input| = O(|x|)
- |output| = (RAM run-time of P)

- **Construct** reusable garbled RAM by combining:
  - one-time garbled RAM (GProg1, GInput1, GEval1)
  - reusable garbled circuits



Problem: In reusable garbled circuits of [GKPVZ13], size of garbled input always exceeds size of circuit output.

Unfortunately: This is inherent. Cannot do better if want simulation security.

- Size of $C[P]$ = (RAM run-time of P)
- |input| = O(|x|)
- |output| = (RAM run-time of P)

- **Construct** reusable garbled RAM by combining:
  - one-time garbled RAM (GProg1, GInput1, GEval1)
  - reusable garbled circuits

- **Solution:**
  - Show that we do not need simulation-security for reusable garbled-circuits. A weaker notion suffices.
  - Construct reusable garbled-circuits with weaker security notion but better efficiency needed in construction. (using indistinguishability obfuscation)

- **Theorem:** Get reusable garbled RAM where:
  - Garble, evaluate program: O(RAM run-time P).
  - Garble input = O(input + output size).

  assuming "ind. obfuscation" + stat. sound NIZK.

- **Theorem:** Get reusable garbled RAM with persistent memory where:
  - garble data = O(data size)
  - garble program = O(description size P)
  - garble input = O(input + output size)
  - evaluate = O(RAM run-time P)

  assuming "strong differing-inputs obfuscation" (heuristic).
- **New**: can be done from ind. obf. [CHJV14] ([BGT14,KLW14,LP14])

Thank You!

Don’t turn me into a circuit!