Distributed Oblivious RAM for Secure Two-Party Computation

Steve Lu
Stealth

Rafail Ostrovsky
UCLA
Stealth

Tokyo, Japan
March 5, 2013

Supported in part by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center (DoI/NBC) contract number D11PC20199. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation therein. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsement, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

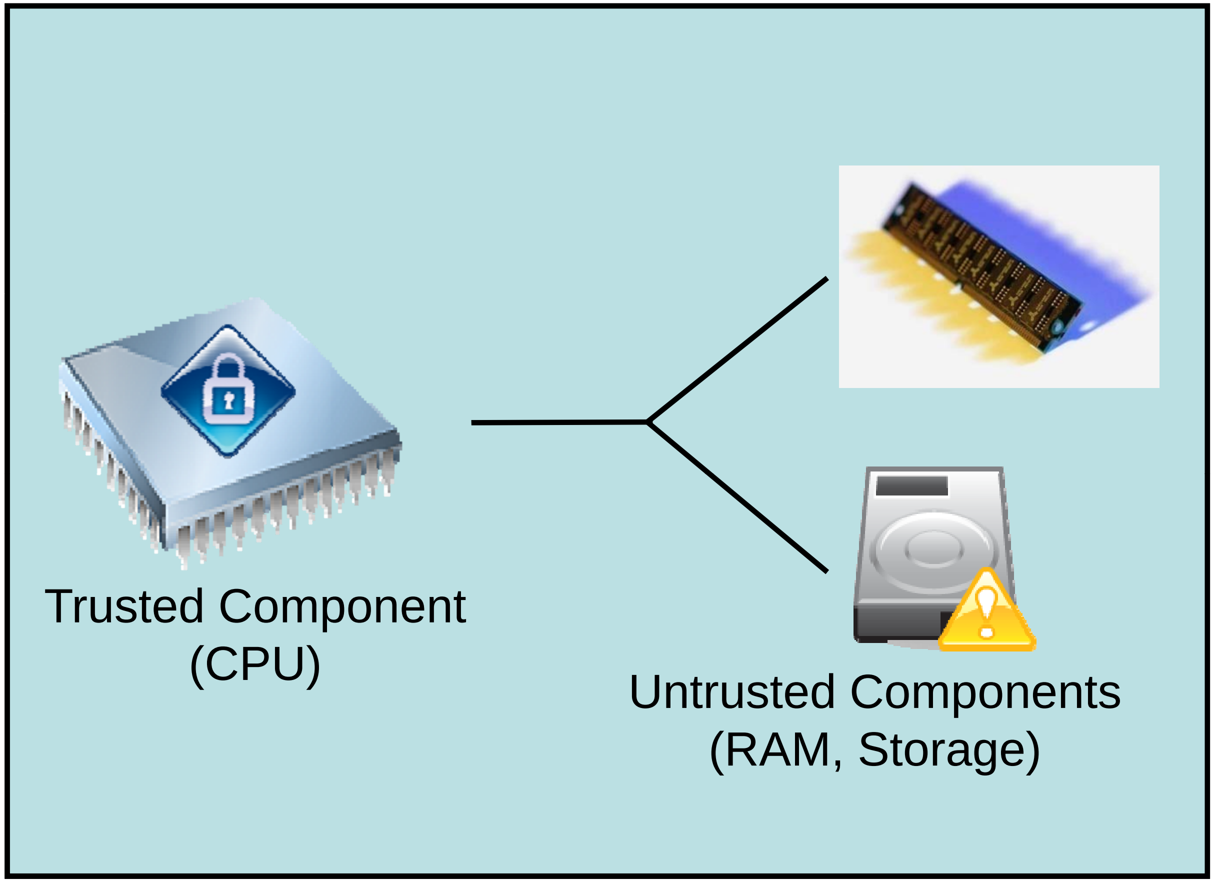
Overview

- Motivation
- Problem Statement
- Review
- New Results
- Conclusion

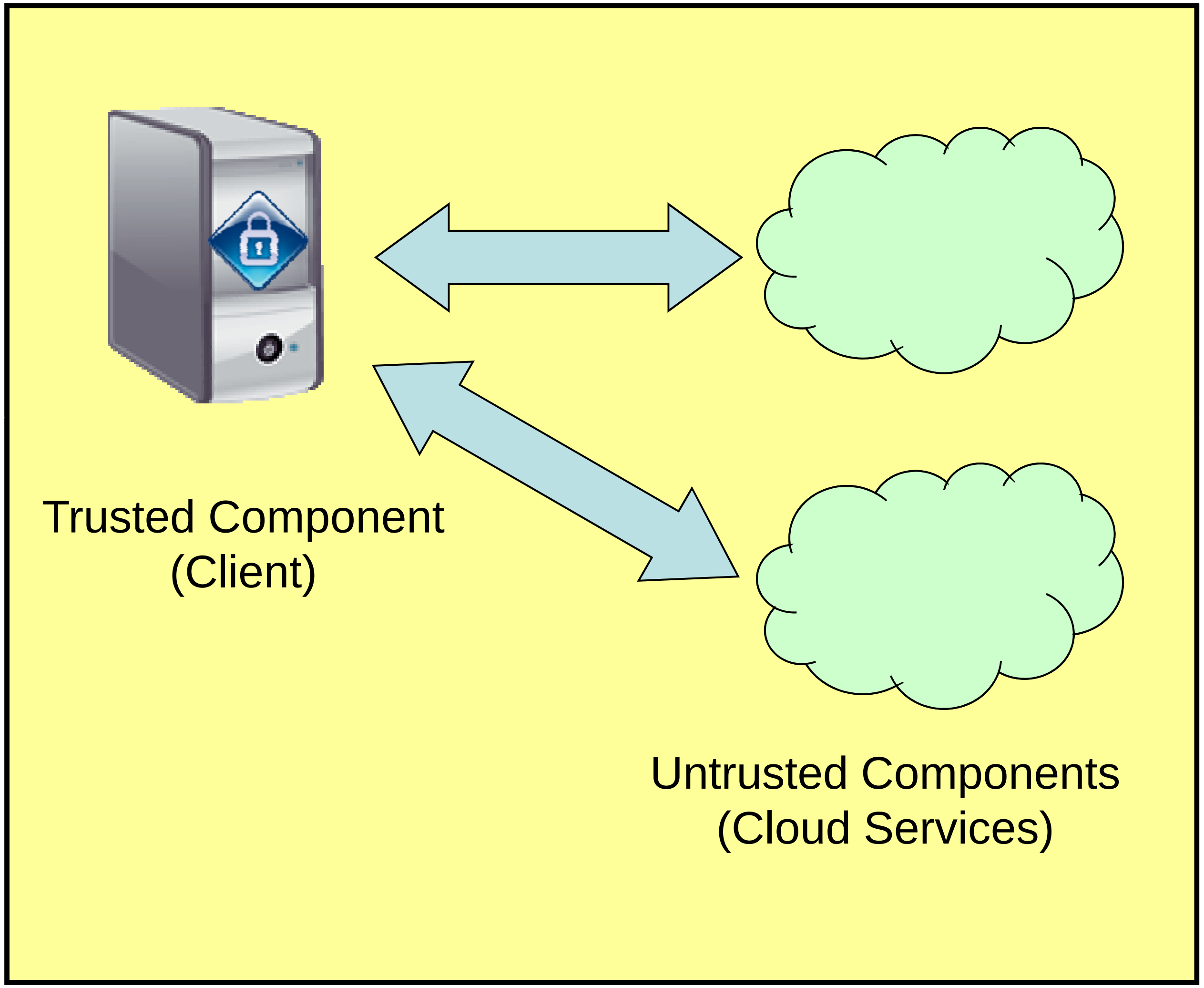
Background – Oblivious RAM (Goldreich'87)

- RAM Model
  - Small trusted component (CPU, client)
  - Large untrusted component (RAM, server)
- Obliviousness
  - Hide the contents and so-called "access-pattern"
    - A program Π is oblivious if one can simulate the (randomized) sequence of accesses to RAM given only the number of accesses

Original Motivation of ORAM: Bootstrapping Secure Hardware



Private Cloud Services



Overview

- Motivation
- Problem Statement
- Review
- New Results
- Conclusion

Oblivious RAM Solutions

- Goal: Given a T-time S-space program Π, compile it into a T'-time S'-space oblivious program Π'

- "Square-root" solution (Goldreich [G87,GO96])
  - $O(n^{1/2} \log n)$ (amortized) Client time overhead
- "Hierarchical" solution (Ostrovsky [O90,GO96])
  - $O(\log^3 n)$ (amortized) Client time overhead
- Constant (in security param.)Client space in both

Many Subsequent Works

- Constant Client Space
  - Pinkas-Reinman [PR10], Goodrich-Mitzenmacher [GM11], Kushilevitz-L-Ostrovsky [KLO12],…
- Larger Client Space
  - Williams-Sion [WS08], Williams-Sion-Carbunar [WSC08], Goodrich-Mitzenmacher [GM11], Boneh-Mazieres-Popa [BMP11], Goodrich-Mitzenmacher-Ohrimenko-Tamassia [GMOT12], Stefanov-Shi-Song [SSS11],…
- Information-Theoretic
  - Ajtai [A10], Damgård-Meldgaard-Nielsen [DMN11],…
- Worst-Case Client Time per query
  - Ostrovsky-Shoup [OS97], Stefanov-Shi-Song [SSS11], Goodrich-Mitzenmacher-Ohrimenko-Tamassia [GMOT11], Shi-Chan-Stefanov-Li [SCSL11],…
- ...

Motivating Problem

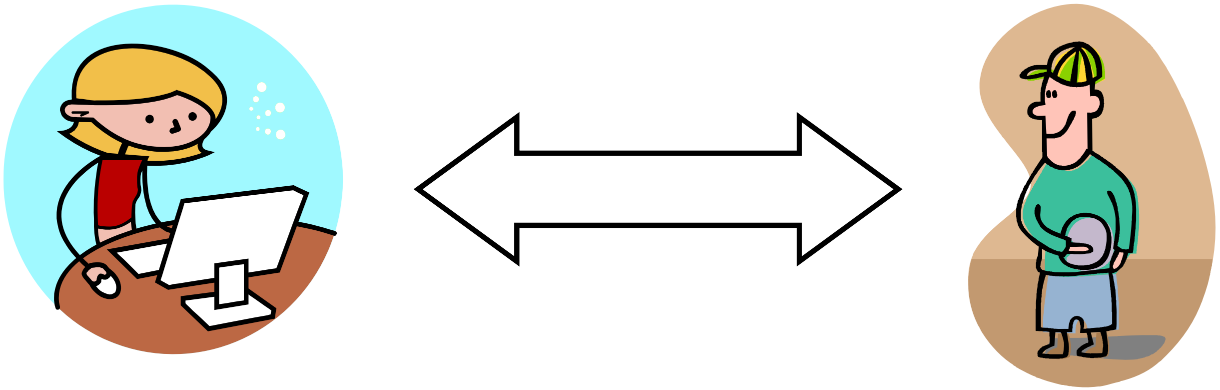
- For solutions with constant client memory
  - Lowest overhead $O(\log^2 n/\log\log n)$ Kushilevitz-L-Ostrovsky [KLO12]

- <u>Problem #1:</u> Can we improve the overhead?

More Motivation

- Most existing secure computation protocols operate on circuits
  - Circuit needs to be as large as the longest execution path
  - Circuit needs to be as large as the inputs
  - Most algorithms are not considered in terms of circuits
- Modular approach
  - Build efficient secure computation for a small class of circuits
  - Extend to arbitrary programs
- Problem #2: Can we come up with efficient candidates for secure RAM computation?

Secure Computation of RAM Programs



Input A

Input B

Wish to securely compute some
program Π (A,B)

Can we bootstrap existing secure circuit computation solutions?
(Rather than converting the programs into circuits)

Our Contribution

- We show how to get ORAM client overhead down to O(logn)
  - In a modified model
  - Constant client memory
  - From OWF

- There are alternative approaches that achieve this by increasing client memory [GM11,SSS11,…]
  - These are efficient stand-alone solutions for ORAM, but doesn't mesh well with our next step...

Our Contribution (Cont.)

- We show how this leads to an efficient 2-party protocol for secure computation *of RAM programs*

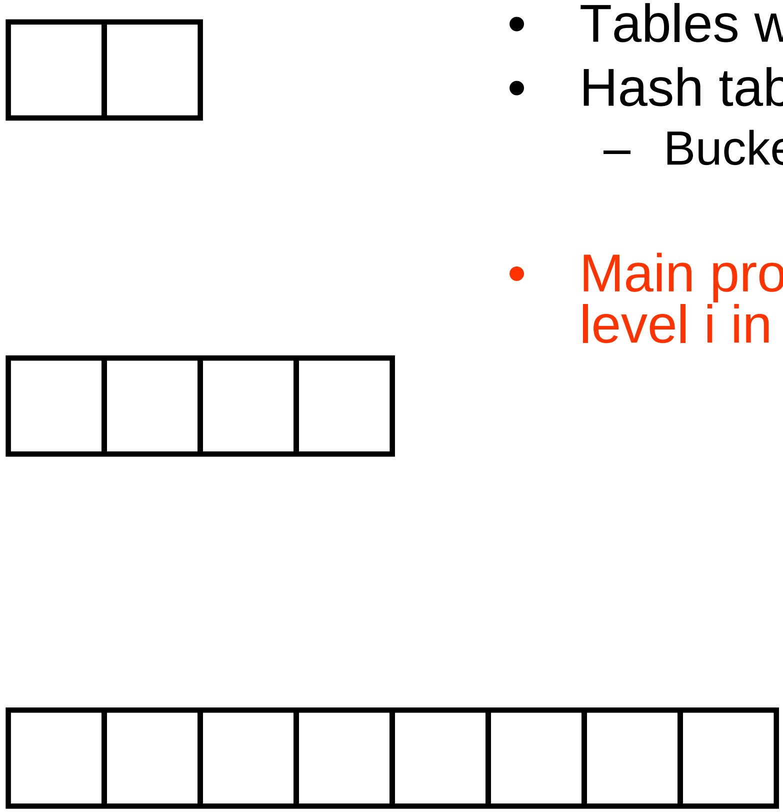
Ostrovsky-Shoup compiler [OS97]

Secure circuit computation with constant overhead (e.g. Ishai et al. [IKOS08, IPS08])

Overview

- Motivation
- Problem Statement
- Review
- New Results
- Conclusion

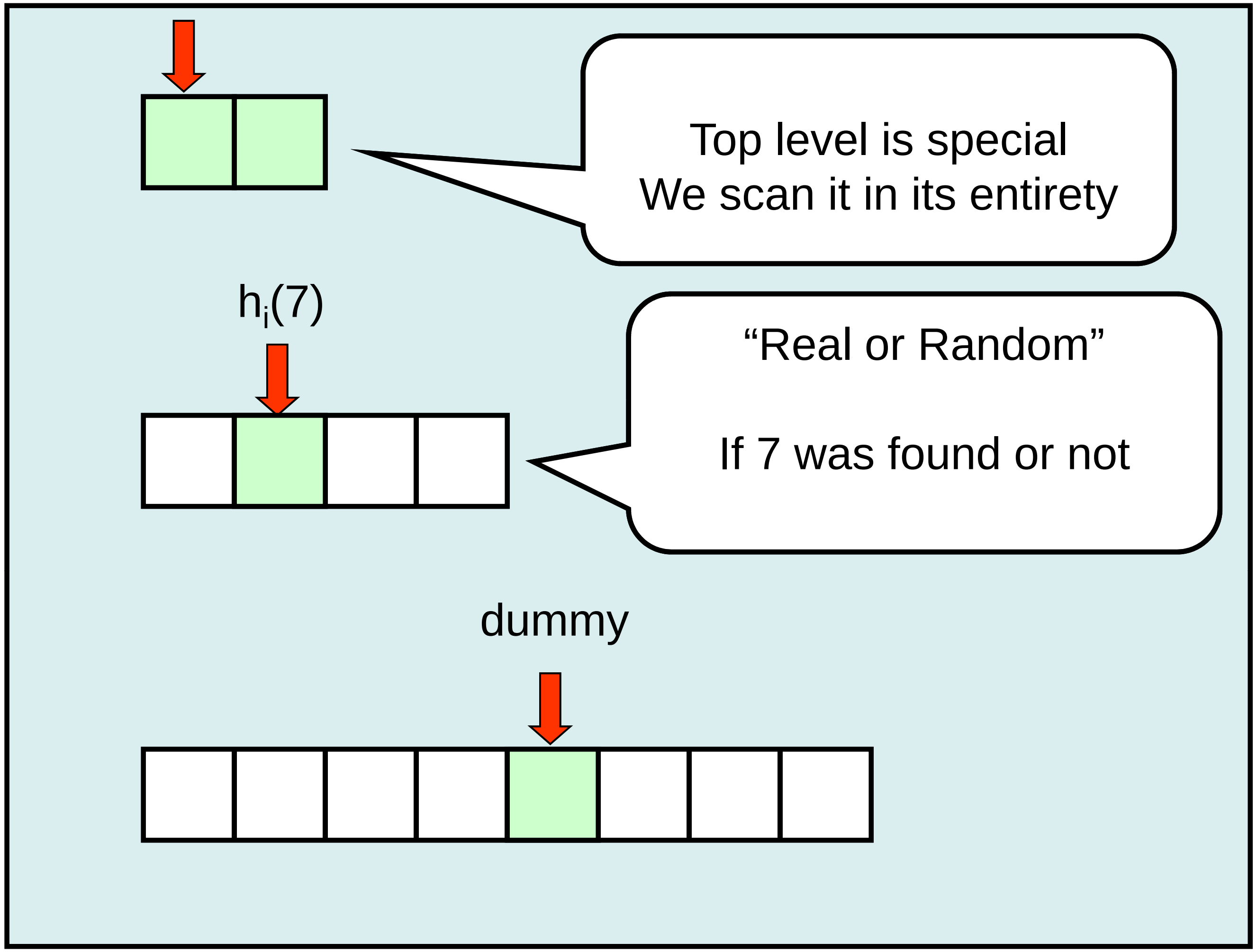
Review: Hierarchical Solution [O90,GO96]



- Set up the Server/RAM in a hierarchy of tables
- Tables with sizes in geometric progression
- Hash tables
  - Bucketed hash tables with log sized buckets

- Main property: (v,x) appears encrypted in a level i in table position $hash_i(v)$

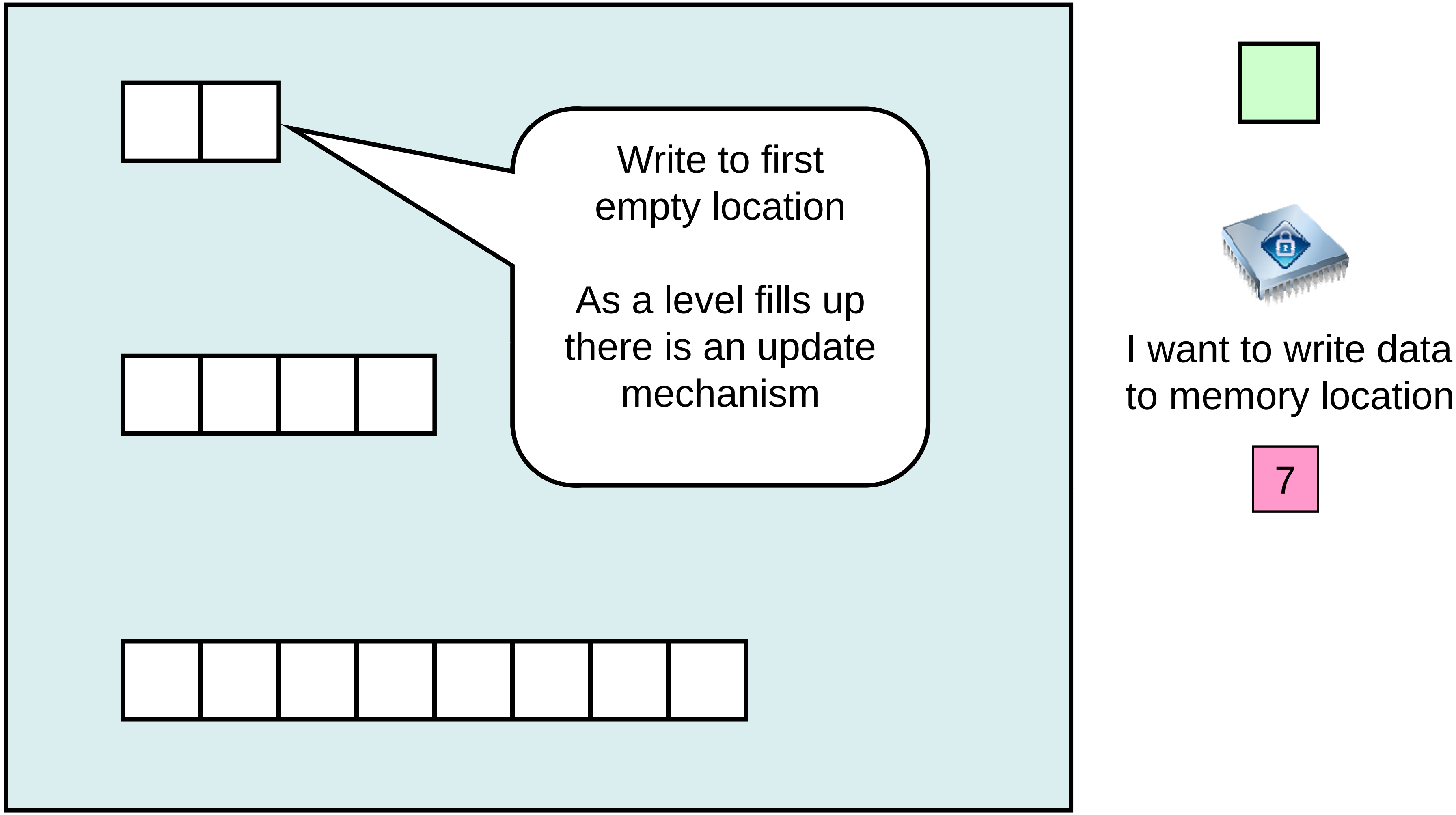
Review: Reading an element



I want to read memory location

7

Review: Writing an element

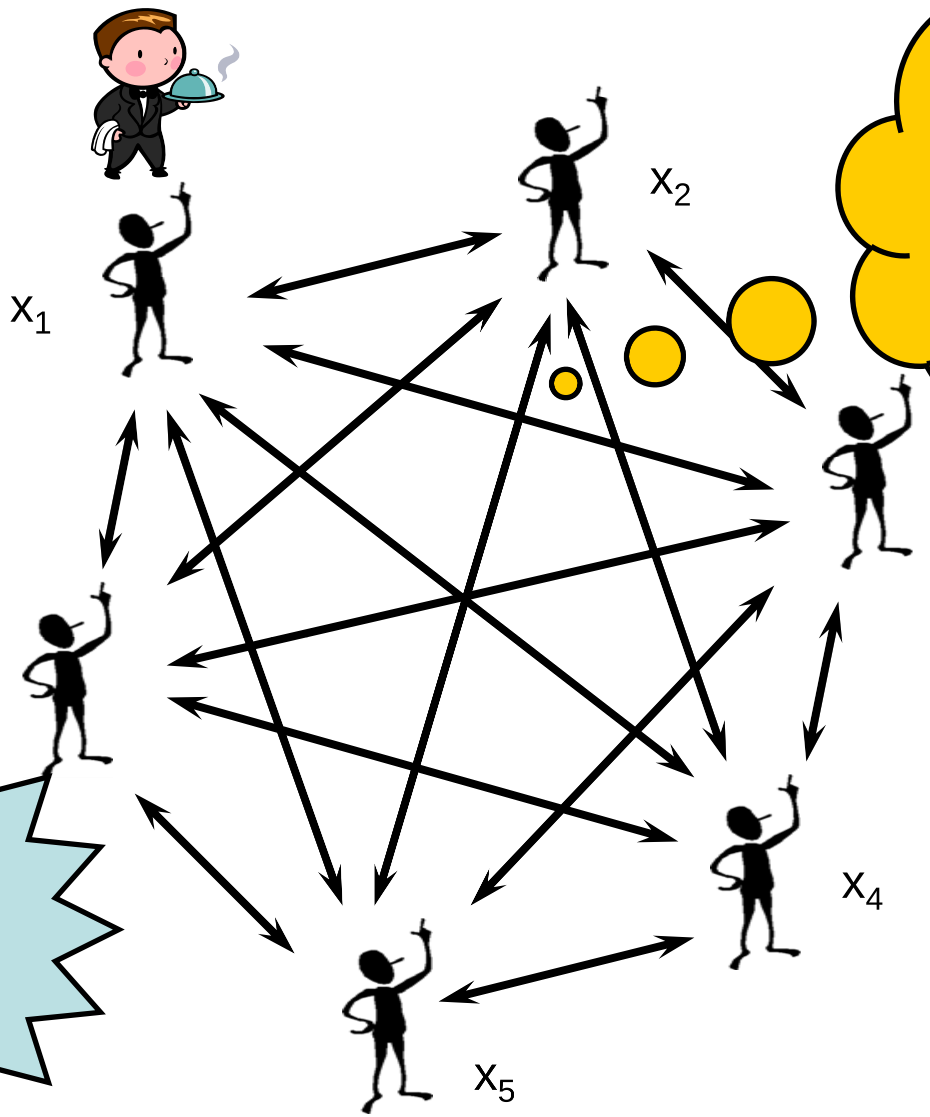


Review [GM11]: ORAM with Cuckoo Hashing

- Cuckoo hash tables [PR01]
  - O(1) worst-case lookup, O(n) space
- Given a *log-sized stash* and sufficiently large table, negl. overflow
- Use cuckoo hash for larger levels
- Oblivious shuffle into cuckoo hash table
  - Our solution bypasses this

Review: Application to Secure RAM Computation (introduced in [OS97])

We want to Compute $F(x_1,\ldots,x_6)$



Main Idea:
Designate Servers,
Then jointly simulate
ORAM CPU using
secure *circuit*
computation

This means
we are able to get
secure program
computation with
overhead proportional
to that of ORAM (and
underlying secure
circuit computation)

Gordon et al.
[G+12] for
sublinear
DB queries

Overview

- Motivation
- Problem Statement
- Review
- New Results
- Conclusion

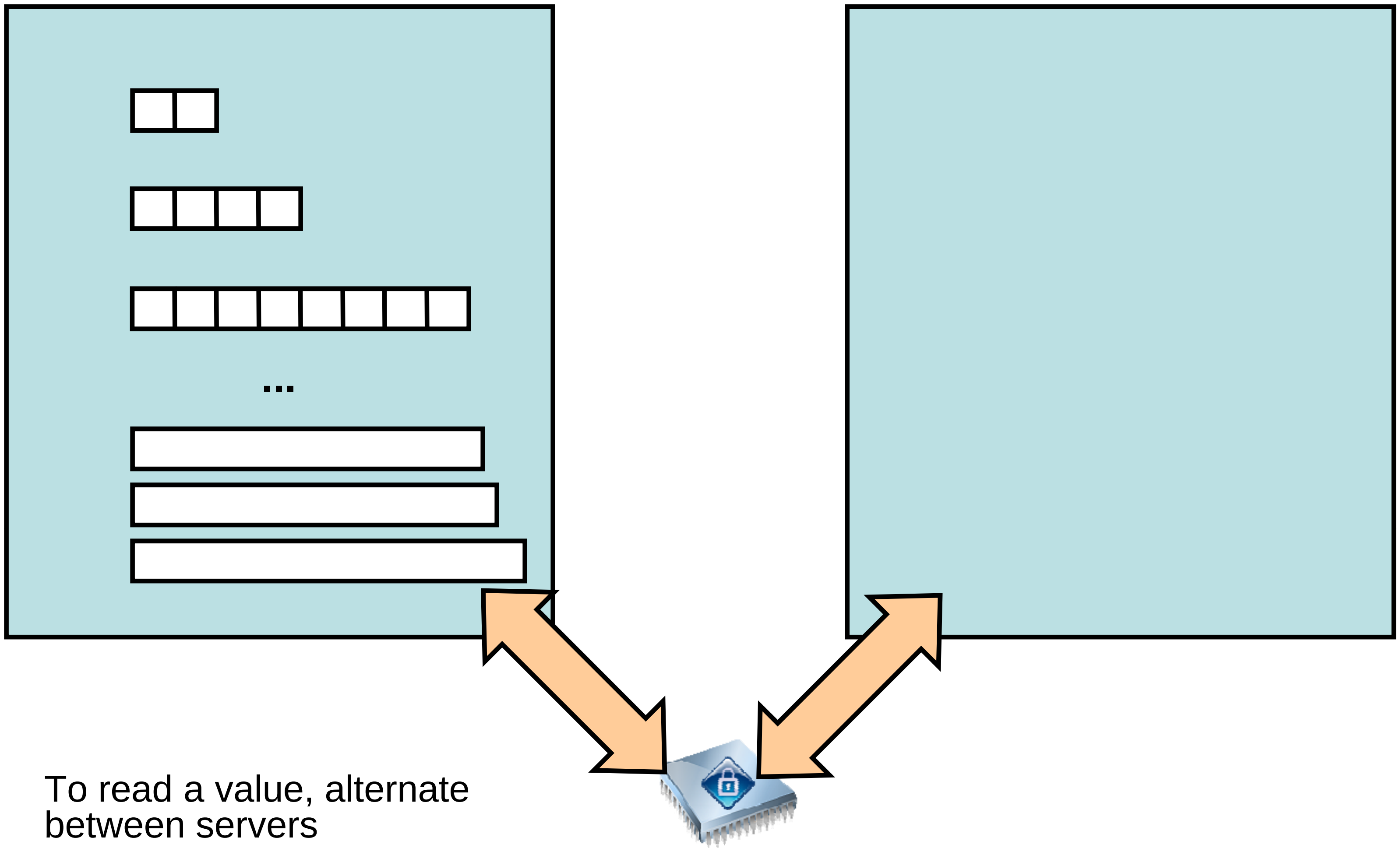
Our Results

- We make two changes to the model:
  - Multiple non-colluding servers
    - Useful theoretical tool
      - Interactive Proofs → multiple provers
      - Private Information Retrieval → multiple servers
      - …
    - e.g. two different cloud services
  - Server can now perform simple computations

Our Results

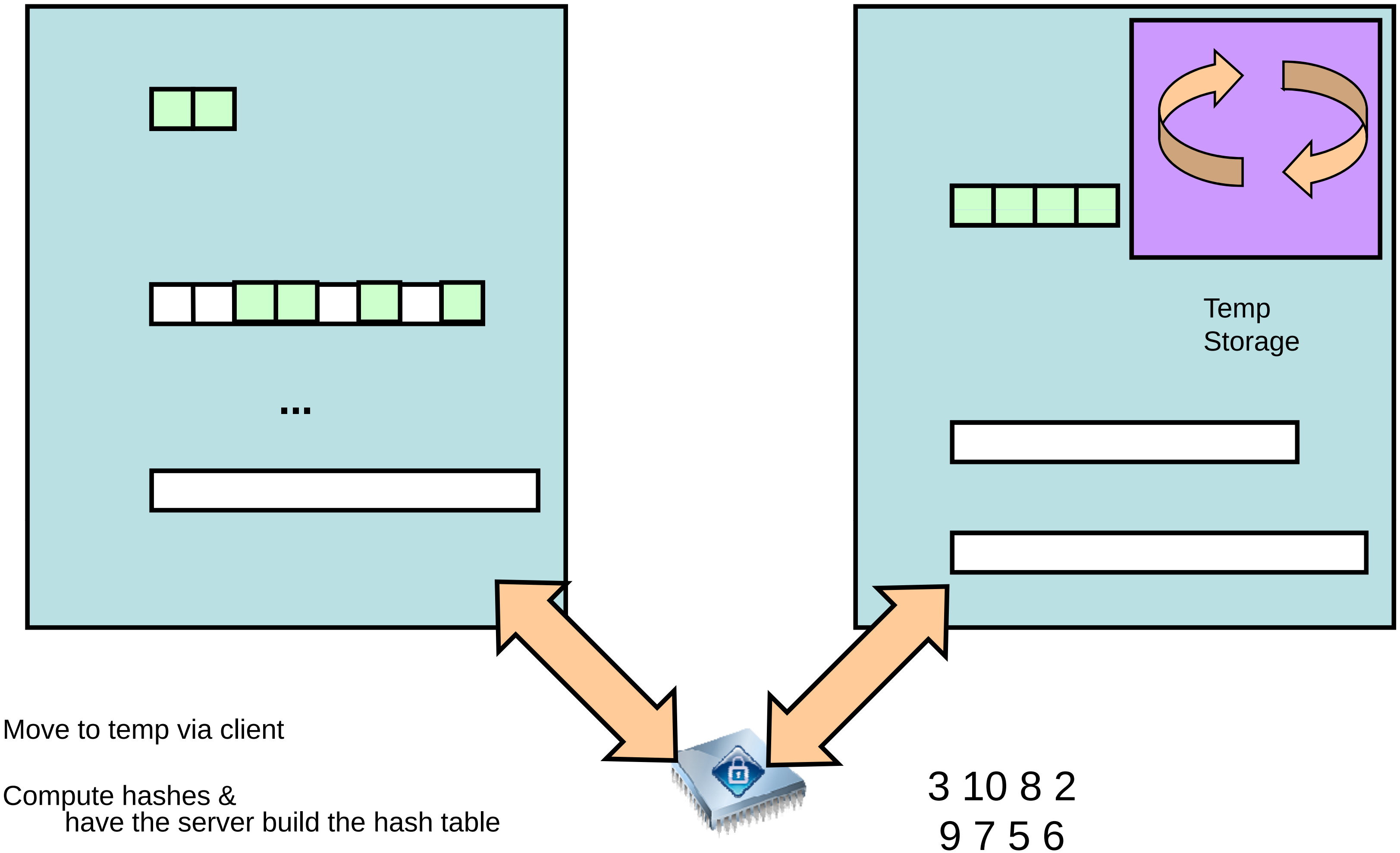
- In this model:
  - O(log n) access overhead with constant client memory
    - Matches lower bound in the *original* setting [GO96]
  - Bypass the expensive “oblivious sort” during updates

Distributed Oblivious RAM



- To read a value, alternate between servers
- Let’s see how update works

Distributed Oblivious RAM
Updating the levels – without sorting!



Move to temp via client

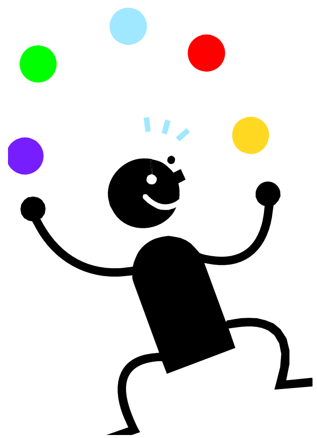
Compute hashes &
have the server build the hash table

Move back to other server via client

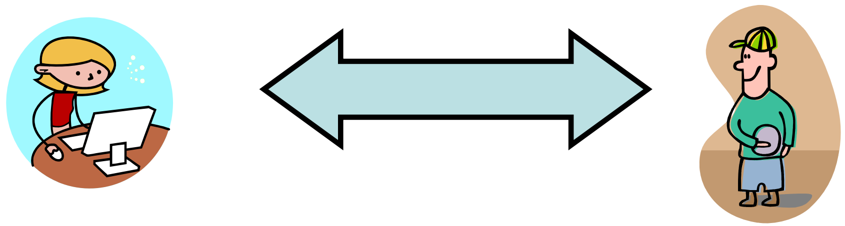
3 10 8 2
9 7 5 6
4 1

Choosing The Parameters

- Top level size
  - O(log n)
- Bucket size
  - O(log n/loglog n)
  - Stash of O(log n)
- Cuckoo Stash size
  - O(log n)
- “Cache the stash”



Application: Secure Computation on RAM Programs



Input A　　　　　　Input B

Wish to securely compute some program Π (A,B)

Exploring the idea of [OS97]:

- Alice plays the role of Server 1
- Bob plays the role of Server 2
- Design a circuit for ORAM CPU
- Use secure (constant overhead) *circuit* computation to run the CPU step
- Result of computation tells each server where to look

Overview

- Motivation
- Problem Statement
- Review
- New Results
- Conclusion

Conclusion

- Described new result for $O(\log n)$ overhead ORAM in the multi-server model
- Application to secure RAM computation

Open Problems

- Improve the overhead (or show a new lower bound in this model)

- What else can we do with this model?

- Can we get non-interactive ORAM for an entire program (multiple read/write)?

  (Yes! --Come to the rump session ☺)

Thank You

Choosing The Hash Tables

- Smallest buffer
  - Just an array
  - O(log n) size turns out to be the right answer
- Standard Bucket Hashing for smaller levels:
  - Tension between bucket size and overflow probability
    - Bucket too big → Too much overhead
      - How big is too big? O(loglog n) levels, goal is O(log n) overhead, so at most O(log n/loglog n)
    - Bucket too small → Overflow probability becomes 1/poly (leads to security problems, see [KLO12])
      - O(log n/loglog n) not large enough!

Choosing The Hash Tables (cont.)

- How do we get around this?
  - Add a log n sized stash
  - Isn't this worse?
    - Additional log n elements we need to scan per level
    - Larger than a bucket!
- Observation (cf [GM11,KLO12]):
  - Only one active stash (lowest updated level)
  - This stash can be re-inserted into the hierarchy

Choosing The Hash Tables (cont.)

- Cuckoo Hash Tables
  - Larger levels all use cuckoo hash tables with stash
  - log n sized stash
    - Can be re-inserted as well

Client Overhead

- Read/Write
  - Read the entire smallest buffer O(log n)
  - Read one bucket for each bucketed hash level
    - ~7loglog n levels
    - Stash implicitly read
    - Bucket of size O(log n/loglog n)
    - Total: O(log n)
  - Read two locations for each cuckoo hash table
    - ~log n levels
    - Stash implicitly read
    - 2 locations each
    - Total: O(log n)
- Update
  - For each level, if that level is of size k, then every k steps the Client moves O(k) elements between the servers
  - O(log n) levels
  - Total: O(log n)