Secure Computation & Oblivious RAM

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Aakash Sharma** & **Gizem Guelesir**

under the guidance of **Michael Zohner**

Acknowledgement

Outline

TECHNISCHE UNIVERSITÄT DARMSTADT

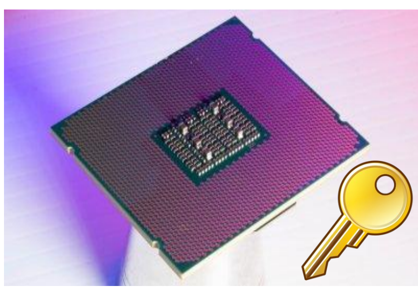
- Introduction
- What is Oblivious RAM?
- Path-ORAM
- SCORAM
- ObliVM Framework
  - Features
  - Experiments
- Conclusion

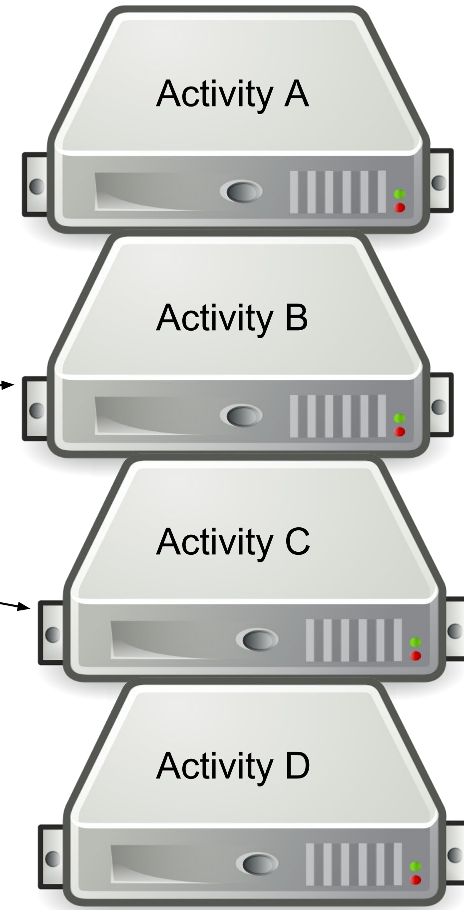
Introduction

- Key problem of *learning about a program from its execution*
- *Access pattern leakage*
- If a block stays at same memory location and it's accessed twice, *frequency and concurrency* information are leaked

Access pattern leakage scenario

Secure Processor

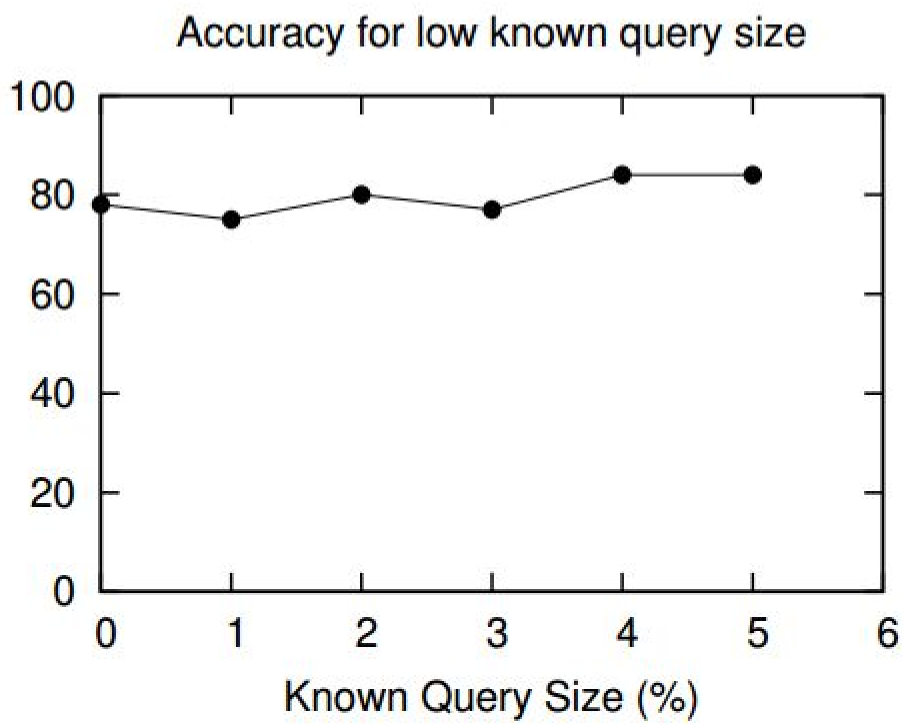




source: https://d.ibtimes.co.uk/en/full/1382894/intel-xeon-e7-processor.jpg?w=400
http://icons.iconarchive.com/icons/aha-soft/security/256/key-icon.png
http://theauroralighthouse.com/wp-content/uploads/2015/07/new-blog-pic4.jpg

Access patterns can leak more than you think

[Iqbal et. al. 2012]



What is Oblivious RAM?

- Initially proposed in theory by Goldreich in 1987 however practical work by Goldreich and Ostrovsky in 1996 [GO96]

“A technique to transform a memory access (with secret index i) into a sequence of memory accesses (which appear independent of the secret value of i)”

- Solution: After reading a block, block must relocate.

Path-ORAM

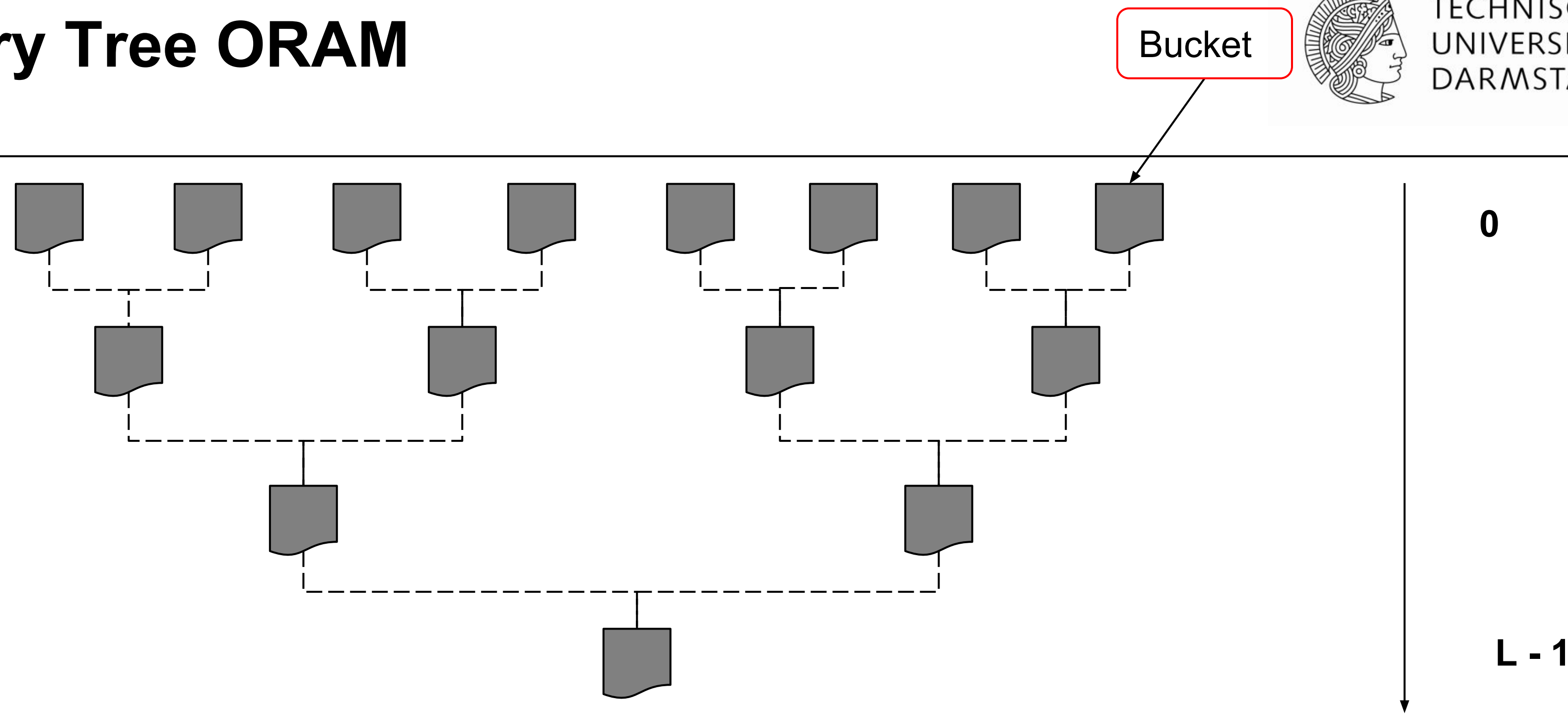
- Presented by Stefanov et. al. [SvDS+13 - CCS'13]

- Based on Tree-ORAM

3 simple steps (algorithm)

1. Look up for Path P := $pos[X]$
2. Read Path P into *stash*
3. Try to write *stash* back into path P, and pack as close as to leaf node as possible

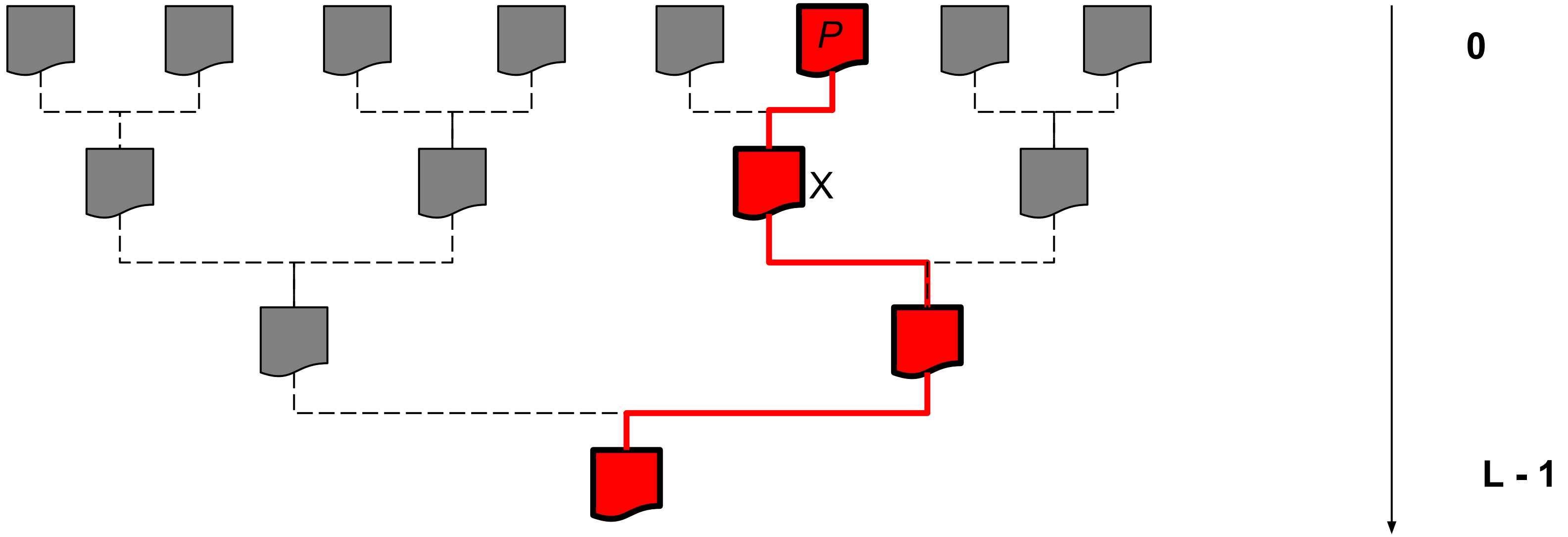
Binary Tree ORAM



Memory

CPU

How Path-ORAM works?



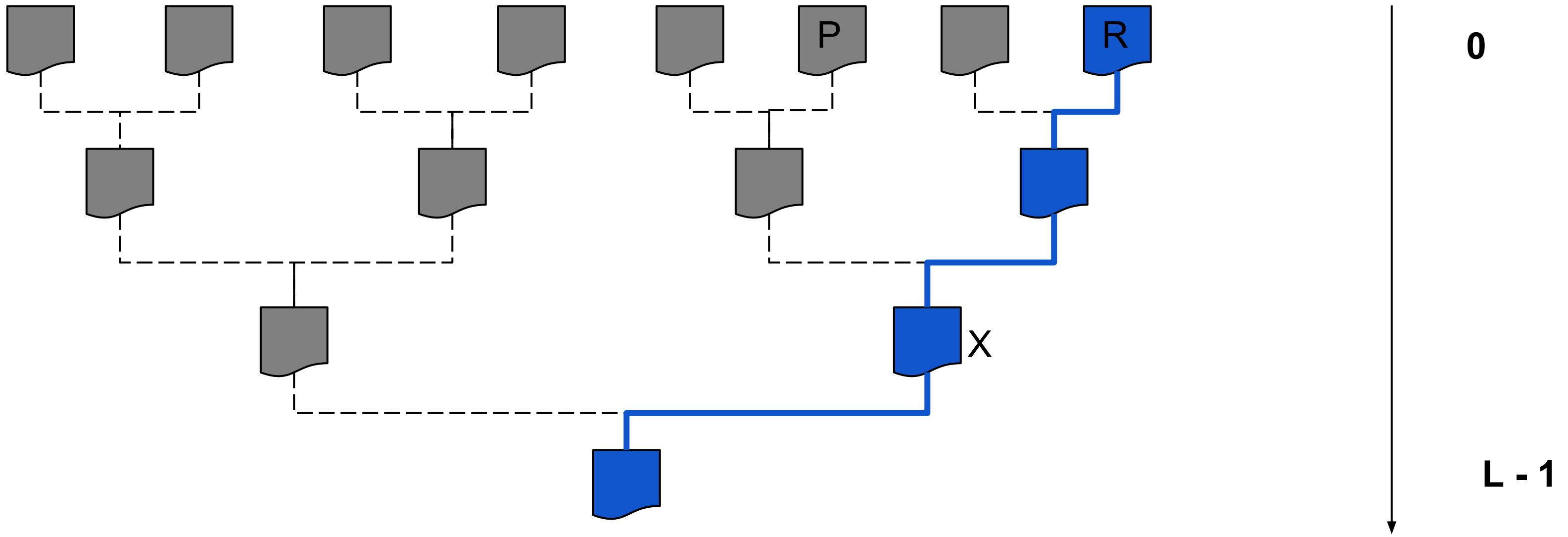
Memory

CPU

Position Map							*P*			

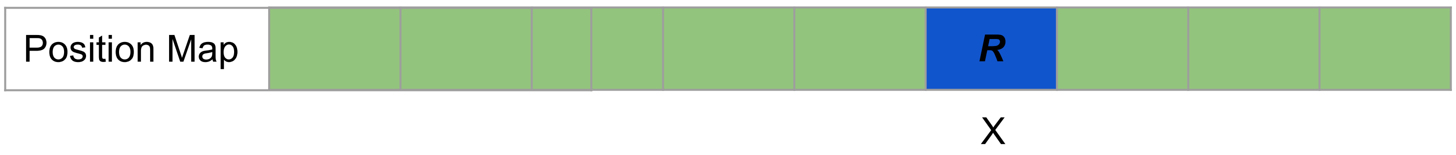
X

How Path-ORAM works?



Memory

CPU



Issues with Path-ORAM

TECHNISCHE UNIVERSITÄT DARMSTADT

- Performance (overhead with large datasets)
- Inefficient circuit size D ($\log^2 N$)

D number of bits in each block (payload)

N number of blocks in ORAM

SCORAM

- Proposed by Wang et. al. [WHC+14]
- Differs in eviction strategy
- Eviction is handled by performing the flush() operation α times, optimal value is 4
- After selecting a random path with every access, the deepest block along the path from root to leaf-1 level is evicted
- Some circuit level optimizations

SCORAM flush() algorithm

TECHNISCHE UNIVERSITÄT DARMSTADT

1. path := UniformRandom(0,. . ., N − 1)
2. bucket[0, . . ., L − 1] := array of buckets from leaf to root
3. B1 := the block in the stash with smallest LCA(path, B1 .label)
4. for i from 0 to L − 1 do (from leaf to root)

   if bucket[i] is not full and LCA(path, B.label) ≤ i and B1 has not been added already then

      Add B1 to bucket[i]

5. for i from L − 1 to 1 do (from root to leaf)

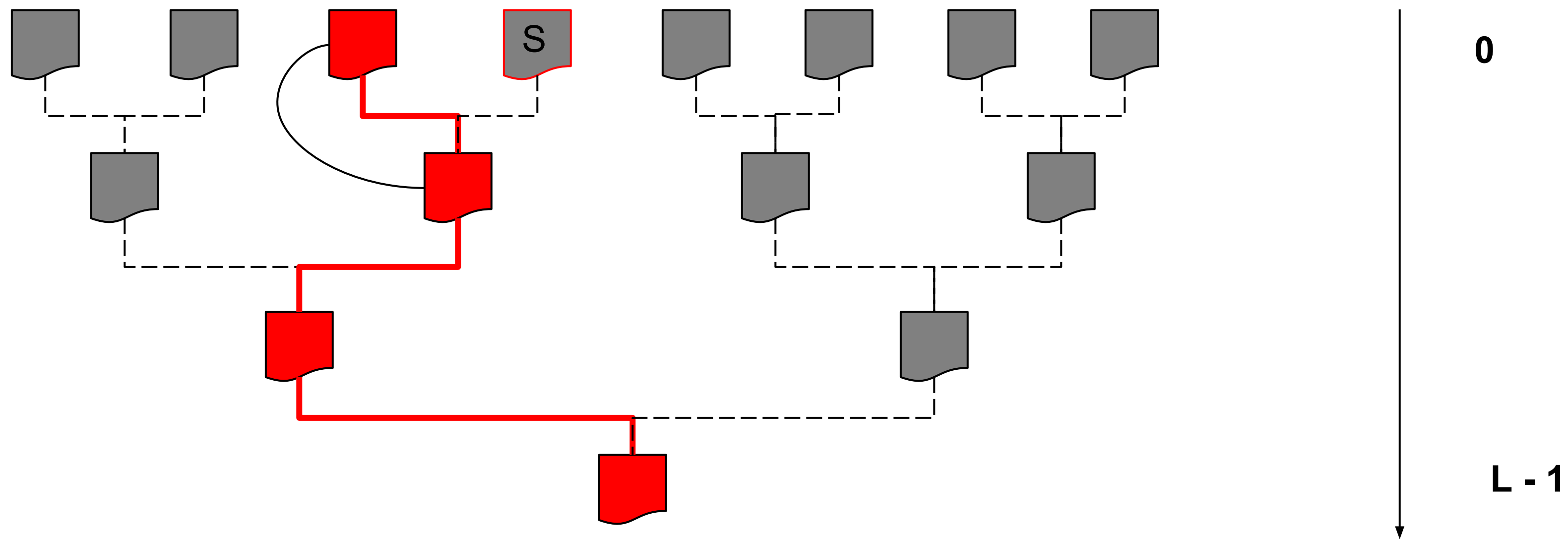
   B2 := the block in bucket[i] with smallest LCA(path, B2 .label)

   if bucket[i − 1] is not full and LCA(path, B2 .label) ≤ i then

      Move B2 from bucket[i] to bucket[i − 1]

How SCORAM works?

TECHNISCHE UNIVERSITÄT DARMSTADT

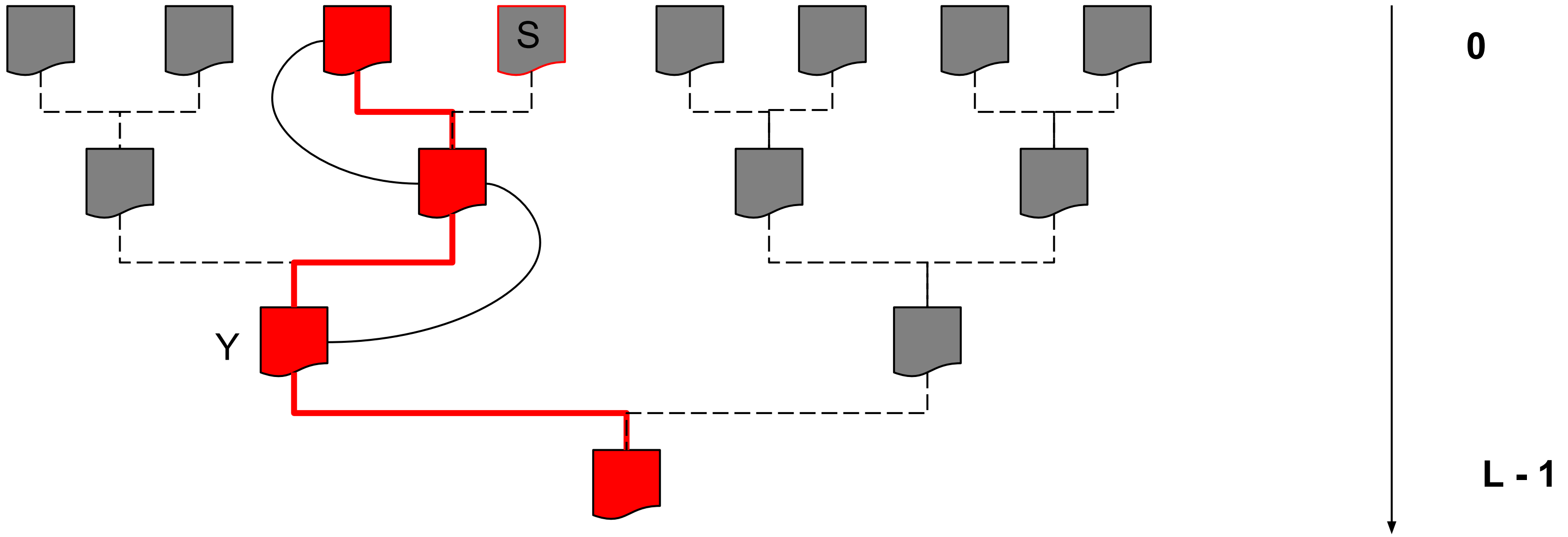


Memory

CPU

Position Map										

How SCORAM works?



Memory

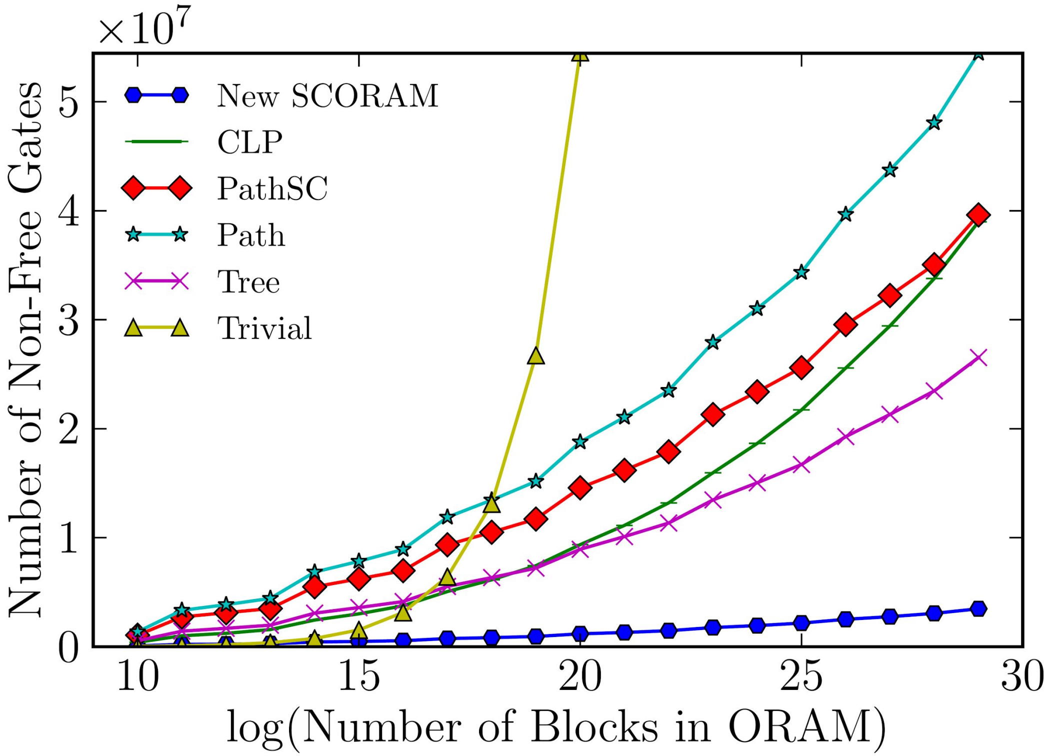
CPU

New metrics for evaluation

Wang et. al. [WHC+14] proposed metrics for evaluation performance of ORAM in secure computing scenarios

- **Cryptographic backend independent** metrics such as "AND gate count"
- **Cryptographic backend dependent** metrics such as "number of encryptions" or "bandwidth" and
- **Implementation and machine dependent** metrics such as "runtime".

Performance comparison



Need for programming framework

TECHNISCHE UNIVERSITÄT DARMSTADT

SCORAM scheme allows the design of more efficient ORAM schemes for secure computation,

- But implementing such a cryptographic protocol is still a tedious task best left to experts
- Non-experts to should be able to design cryptographic protocols based on ORAM and secure computation too
- Scalability becomes issue when data gets bigger, need for programming abstractions

ObliVM Framework

- Domain specific programming framework
  - extends SCVM **[LHS+14]**
  - expressive language ObliVM-lang
  - new features

- Two parties to do computation on their private inputs using a common function without revealing anything else except the output

- Provides support to the programmers to do the conversion from program to circuit representation (which secure computation relies on) in an efficient way

- ORAM the generic approach is deployed in cases where the customized efficient abstractions cannot be applied

How to achieve trace obliviousness

Trace obliviousness is the requirement to be transfer the oblivious program into circuit representation

Memory Trace Obliviousness: For each memory accesses

- Many random read/writes
- Shuffled mapping between the data in the program and physical location

Instruction Trace Obliviousness (program counter): secret conditional branch:

- Both branches executed
- Only one takes affect

Promise of ObliVM

TECHNISCHE UNIVERSITÄT DARMSTADT

Efficiency: ObliVM compiler provides maximum efficiency with large program into circuit conversion (no linear scan of memory, binary tree-based)

- magnitudes of generic ORAM
- 0.5% to 2% hand crafted designs

Intuitiveness: Non expert programmers can easily program secure computation protocols and use abstractions provided by expert programmers

Extensibility: Gives ability to expert programmer to extend the language with **higher level** protocols, rich libraries or implement **low level** libraries on the ObliVM without dealing with high complexity.

Programming Abstractions : MapReduce, loop coalescing, less programming effort, key to achieve scalability with big data

ObliVM-lang features

Security Labels:

- **public**, observable by both parties
- **secure** secretly shared, except random type
  - `secure int10`[`public` 1000] keys: only content secret, not placed on ORAMs

Standard Features: record types, like C type struct

Generic Constants: reusability, no need for hard coded constant

- Type `int@m` integer with m bits

Functions: signature of search function of a tree with generic constant m-bit integer key value pairs:

- T Tree`@m`<T>.`search`(public `int@m`)

ObliVM-lang features for security

Random numbers: always secretly shared

- `rnd32` 32 bit random integer, `RND` built in function
- `rnd@m RND`(public `int32` m), function which takes 32 bit integer m and returns m random bits.
- exception: only function which is dependent to the value m, dependent types brings complexity

Implicit de-classifications: assignment to a public variable

- random number at most once implicitly de-classified

Implicit de-classification example

- s is a secret variable

```
rnd32 r1 = RND(32), r2 = RND(32);
public int32 z;
if (s) z = r1; //implicit de-classification
else z = r2; //implicit de-classification
....
public int32 y = r2; //invalid assignment
```

- y and z can be correlated

Phantom functions

- Function call in secret if statement, real or phantom mode

```
phantom secure int32 prefixSum(public int32 n) {
    secure int32 ret = a[n];
    a[n] = 0;
    if (n != 0) ret = ret + prefixSum(n -1);
    return ret;
}
if (s) then x = prefixSum(n);  //will be executed independent of s, if s false elements are not assigned
```

- Code generated by compiler prefixSum(idx, indicator) // indicator = mode
- Generated traces will be same with any value of indicator

Loop coalescing illustration [LWN+15]

Technique reduces the cost of sparse graph algorithms

- code divided into bounded loop blocks
- total iterations = ∑ execution time of each block instead of π

```
bwhile(n) (; u<n;) {
    total = total + 1;
    i=s[u];
    bwhile (m) (i<s[u+1]) {
        // do something
        i=i+1;
    }
    u=u+1;
}
```

$n + m + n$ iterations in total

Block 1 $\times n$

Block 2 $\times m$

Block 3 $\times n$

```
state = (u<n) ? 1 : -1;
for (__itr=0; __itr<n+m+n; __itr++) {
    if (state==1) {   total=total+1; i=s[u];
        state = (i<s[u+1]) ? 2 : 3
    } else if (state==2) { // do something
        i=i+1; state = (i<s[u+1]) ? 2 : 3
    } else if (state==3) {
        u=u+1; state = (u<n) ? 1 : -1
    } // else execution is finished
}
```

Experiments

Algorithms		Complexity		
		Our Complexity	Generic ORAM	Best Known
Sparse Graph	Dijkstra's Algorithm	$O((E+V)\log^2 V)$	$O((E+V)\log^3 V)$	$O((E+V)\log^3 V)$ (Generic ORAM baseline [29])
	Prim's Algorithm	$O((E+V)\log^2 V)$	$O((E+V)\log^3 V)$	$O(E\frac{\log^3 V}{\log\log V})$ for $E = O(V\log^\gamma V), \gamma \geq 0$ [22] $O(E\frac{\log^3 V}{\log^\delta V})$ for $E = O(V2^{\log^\delta V}), \delta \in (0,1)$ [22] $O(E\log^2 V)$ for $E = \Omega(V^{1+\epsilon}), \epsilon \in (0,1]$ [22]
Dense Graph	Depth First Search	$O(V^2 \log V)$	$O(V^2 \log^2 V)$	$O(V^2 \log^2 V)$ [49]

Table 1: Complexity comparison of graph algorithms with various techniques **[LWN+15]**

- Costs are in terms of circuit size
- Oblivious Dijkstra's and Prim's algorithms implementations with loop coalescing technique

Future work

TECHNISCHE UNIVERSITÄT DARMSTADT

"4.1x to 5.1x better than SCORAM"

- Circuit ORAM [WCS15] in CCS'15

ObliVM

- Multiple parties
- More oblivious programming abstractions
- Public generic types

Conclusion

- Important step towards achieving secure two party computation protocol in practical scenarios which involve large datasets.

- Expressiveness and ease of use of the ObliVM programming framework would speedup adaptation of secure two party computation protocols.

- The ObliVM framework and many sample implementations are available at http://www.oblivm.com.

References

[**WHC⁺14**] Wang, Xiao Shaun and Huang, Yan and Chan, T-H. Hubert and Shelat, Abhi and Shi, Elaine. SCORAM: Oblivious RAM for Secure Computation *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*

[**GO96**] Goldreich, Oded and Ostrovsky, Rafail. Software protection and simulation on oblivious RAMs *in Journal of the ACM Volume 43 Issue 3, May 1996 Pages 431-473*

[**SVDS⁺13**] Stefanov, Emil and van Dijk, Marten and Shi, Elaine and Fletcher, Christopher and Ren, Ling and Yu, Xiangyao and Devadas, Srinivas. Path ORAM: An Extremely Simple Oblivious RAM Protocol *in Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*

[**IKK12**] Mohammad Saiful Islam, Mehmet Kuzu, Murat Kantarcioglu. Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation

[**WCS15**] Xiao Wang, Hubert Chan, Elaine Shi. Circuit ORAM: On Tightness of the Goldreich-Ostrovsky Lower Bound *in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security CCS '15*

References contd..

[LHS+14] Chang Liu, Yan Huang, Elaine Shi, Jonathan Katz, and Michael Hicks. Automating efficient ram-model secure computation. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP ’14, pages 623–638, Washington, DC, USA, 2014. IEEE Computer Society.

[LWN+15] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi. Oblivm: A programming framework for secure computation. In *Security and Privacy*, pages 359–376. IEEE, 2015.

Questions?

Source: http://www.thankyou-verymuch.com/TYVMI.png