Secure Computation ORAM
The Case of 3-Party Computation

Stanislaw Jarecki, UC Irvine

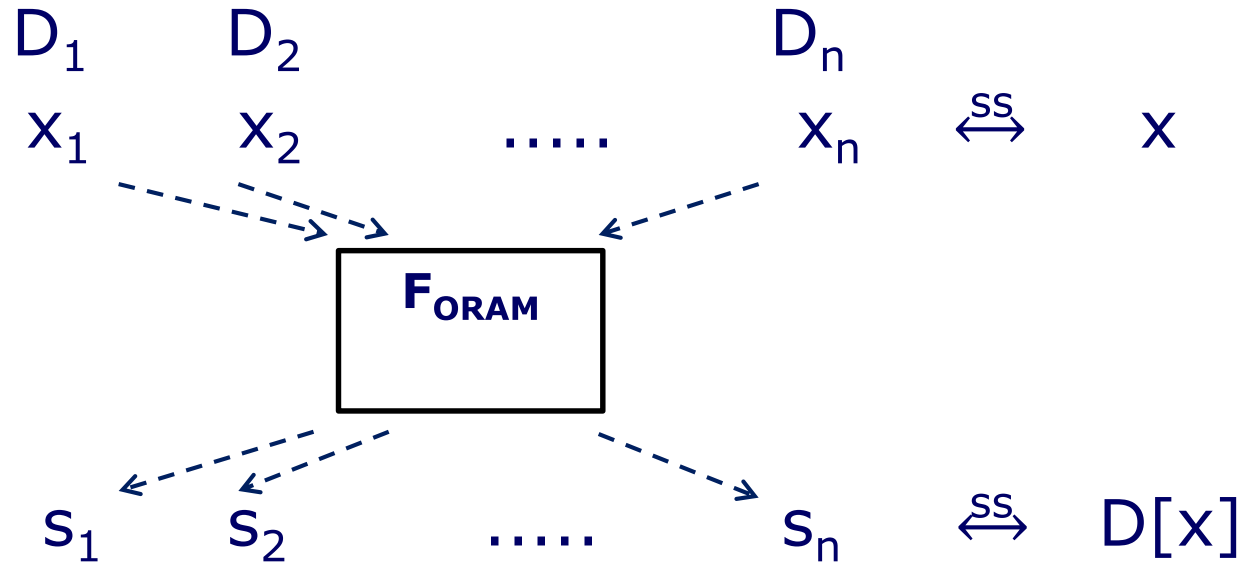
Cryptography in the RAM Model Workshop,
Cambridge, MA,
June 2016

AC'15: Sky Faber, **S.J.**, Sotirios Kentros, Boyang Wei

New Work: **S.J.**, Boyang Wei

Secure Computation of (O)RAM Access (SC-ORAM)

SC-ORAM = Sec.Comp of F_{ORAM}: sharing of D,x ⟶ sharing of D[x]

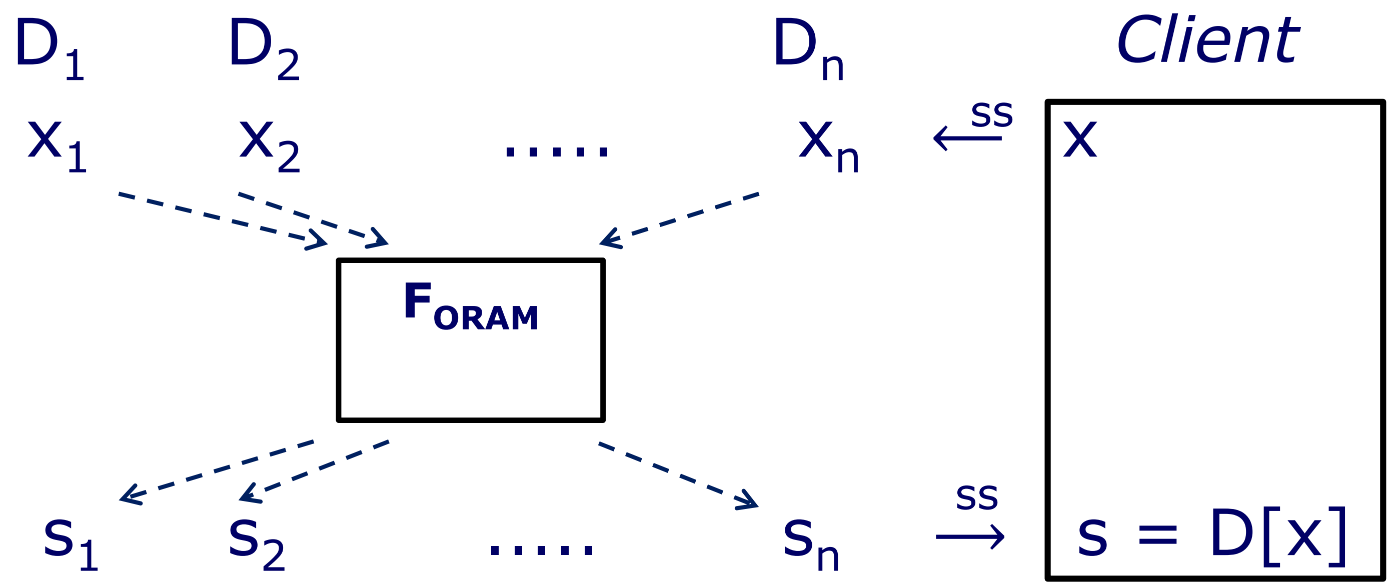


(for <write>: additional shared input v and D ⟶ D' s.t. D'[x]=v)

Secure Computation of (O)RAM Access (SC-ORAM)

Application: n-Server Private Database (≈ n-Server SPIR)

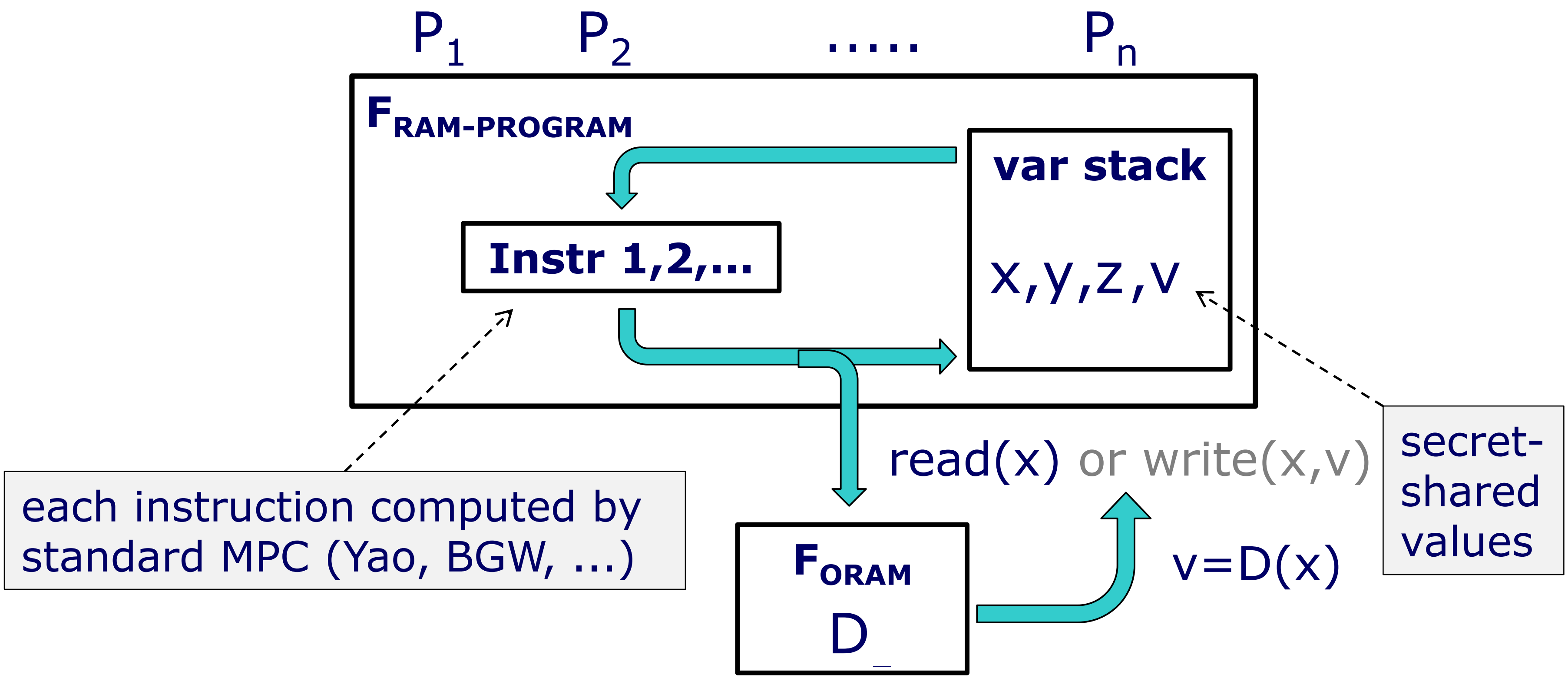
SC-ORAM = Sec.Comp of F_{ORAM}: sharing of D,x ⟶ sharing of D[x]



Secure Computation of (O)RAM Access (SC-ORAM)

Application: Sec. Comp. of RAM Program [OS'97,DMN'11,GKKKMRV'12]

SC-ORAM = Sec.Comp of F_{ORAM}: sharing of D,x ⟶ sharing of D[x]

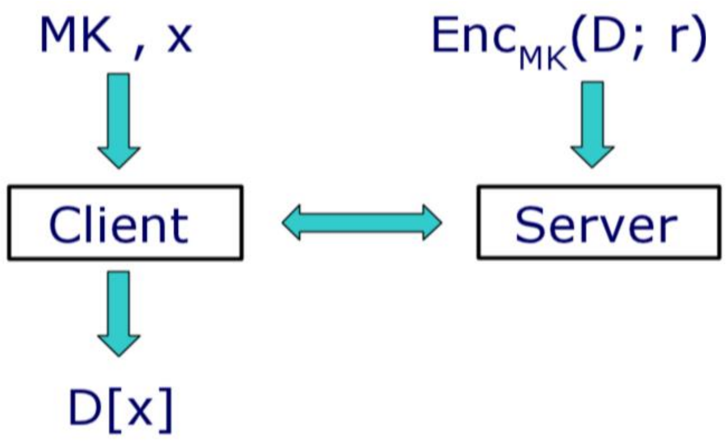


Sec. Comp. of RAM programs with polylog(|D|) overhead

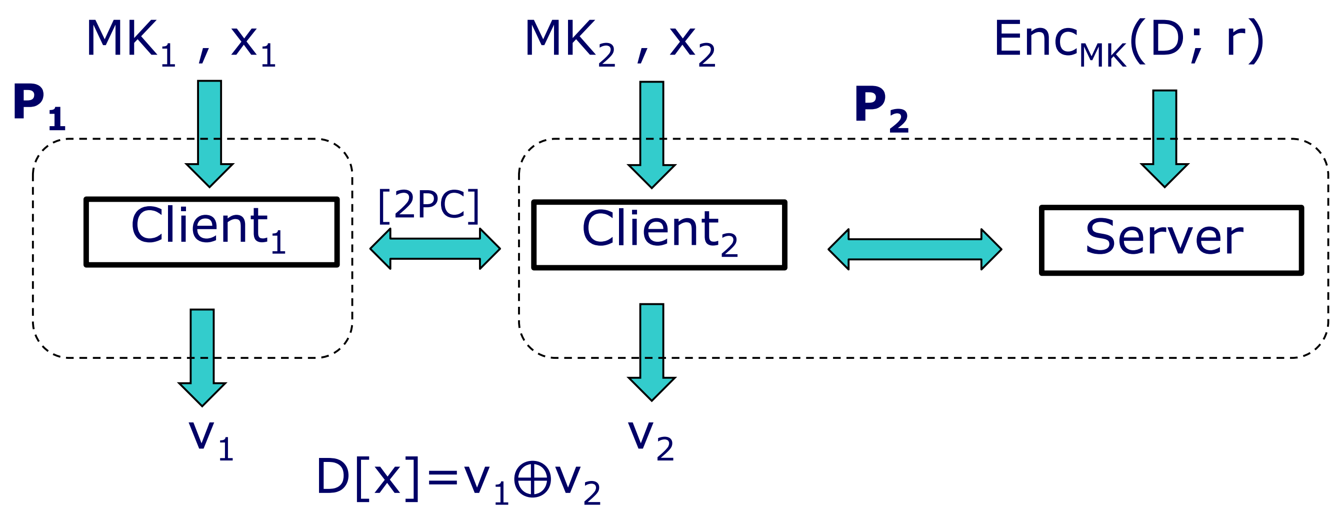
Generic SC-ORAM Construction [OS'97,GKKKMRV'12]

ORAM Scheme + Secure Comp. of Client's Code ⟶ SC-ORAM

ORAM:

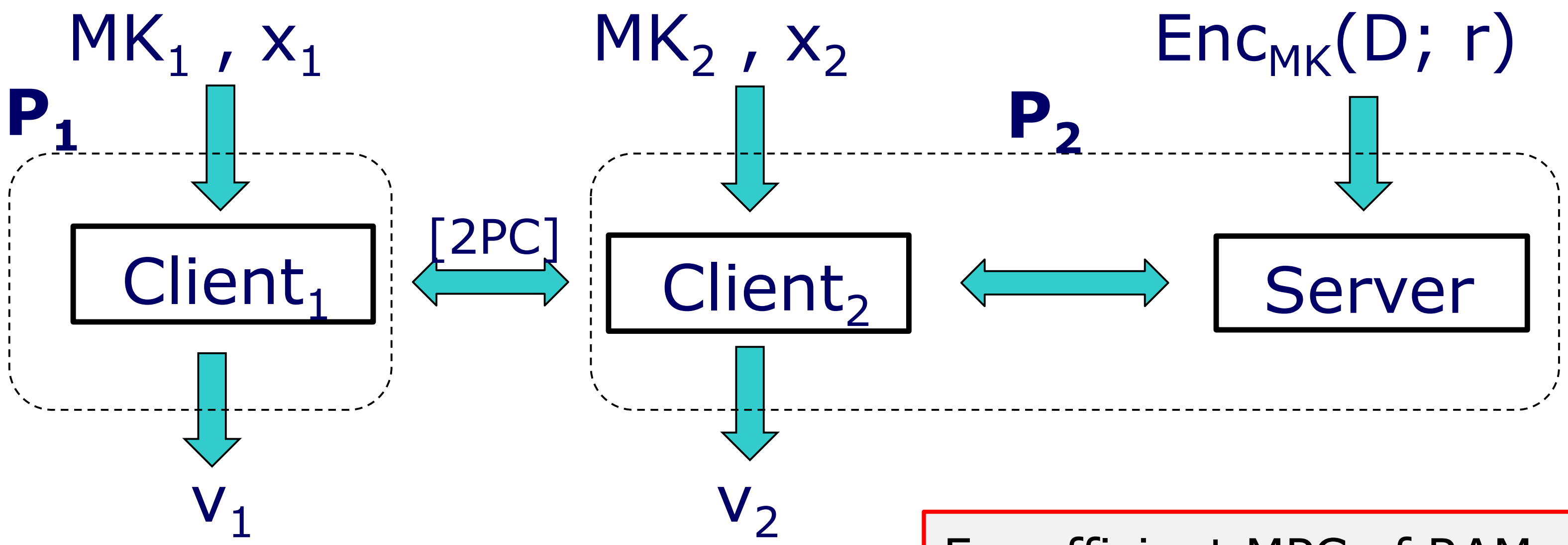


SC-ORAM:



Generic SC-ORAM Construction [OS'97,GKKKMRV'12]

ORAM Scheme + Secure Comp. of Client's Code



$D[x] = v_1 \oplus v_2$

For efficient MPC of RAM programs we need ORAM whose Client is "Secure-Computation Friendly"

[GKKKMRV'12a]: GO'96 ORAM + Yao + PK-based SS-OPRF gadget

[GKKKMRV'12b]: Path-ORAM [Shi+'11] + Yao

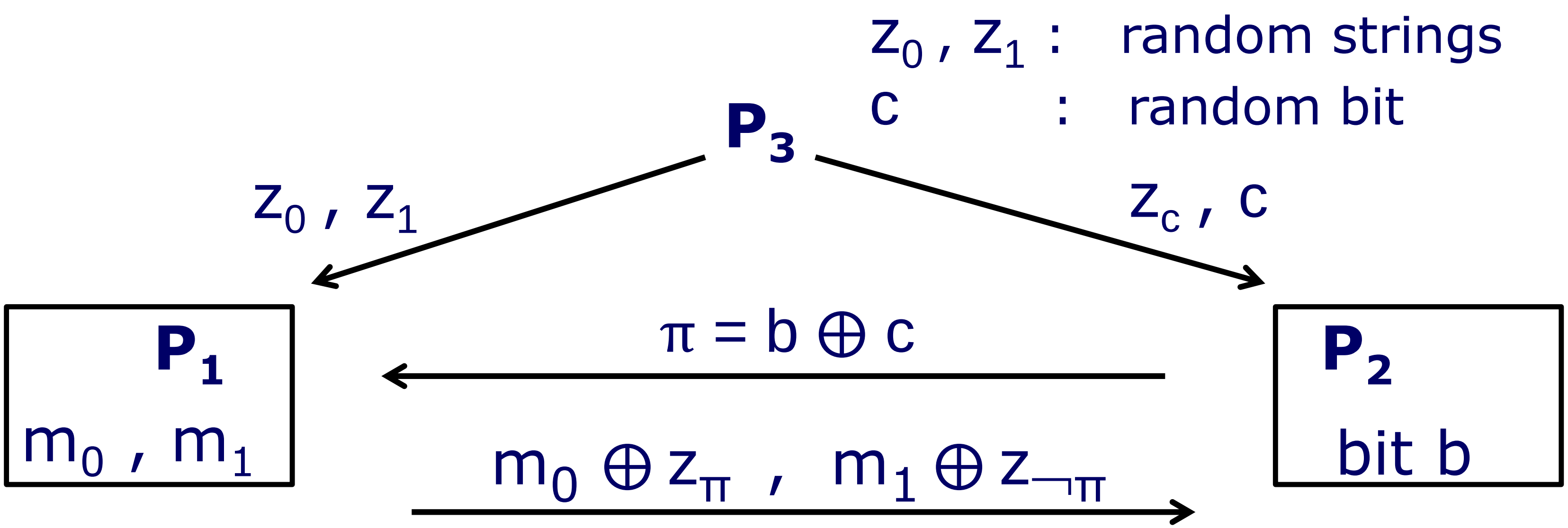
[WHCSS'14]: Path-ORAM modified + Yao ⇒ smaller circuits

Our Question:

Could SC-ORAM be faster given 3 players with honest majority?

3 Parties = 2 Parties with <u>correlated randomness</u>

Example: Oblivious Transfer with Precomputation [Bea'95]



2 Parties: OT needs PK crypto ops [IR'89]

3 Parties: OT costs 4 xor's

SC for Path-ORAM [Shi+'11]

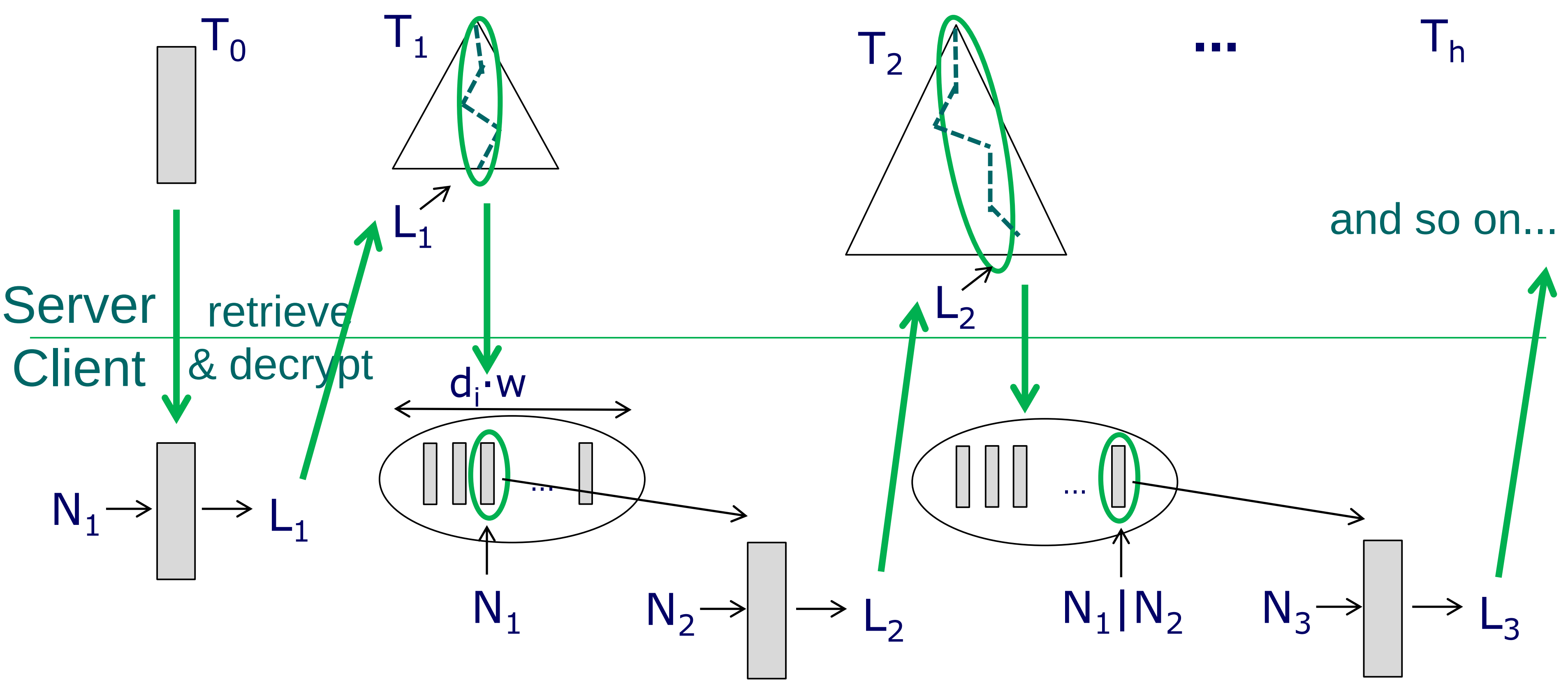
Path-ORAM Access: Recursive Tree+Array Lookup

Split address space of m bits, h chunks of $\tau = m/h$ bits

$N = [N_1 \mid N_2 \mid \ldots \mid N_h]$

T_i is a binary tree of depth $d_i = i \cdot \tau$, tree nodes are buckets of size w

$ORAM = (T_0, T_1, T_2, \ldots, T_h)$



SC for Path-ORAM [Shi+'11]

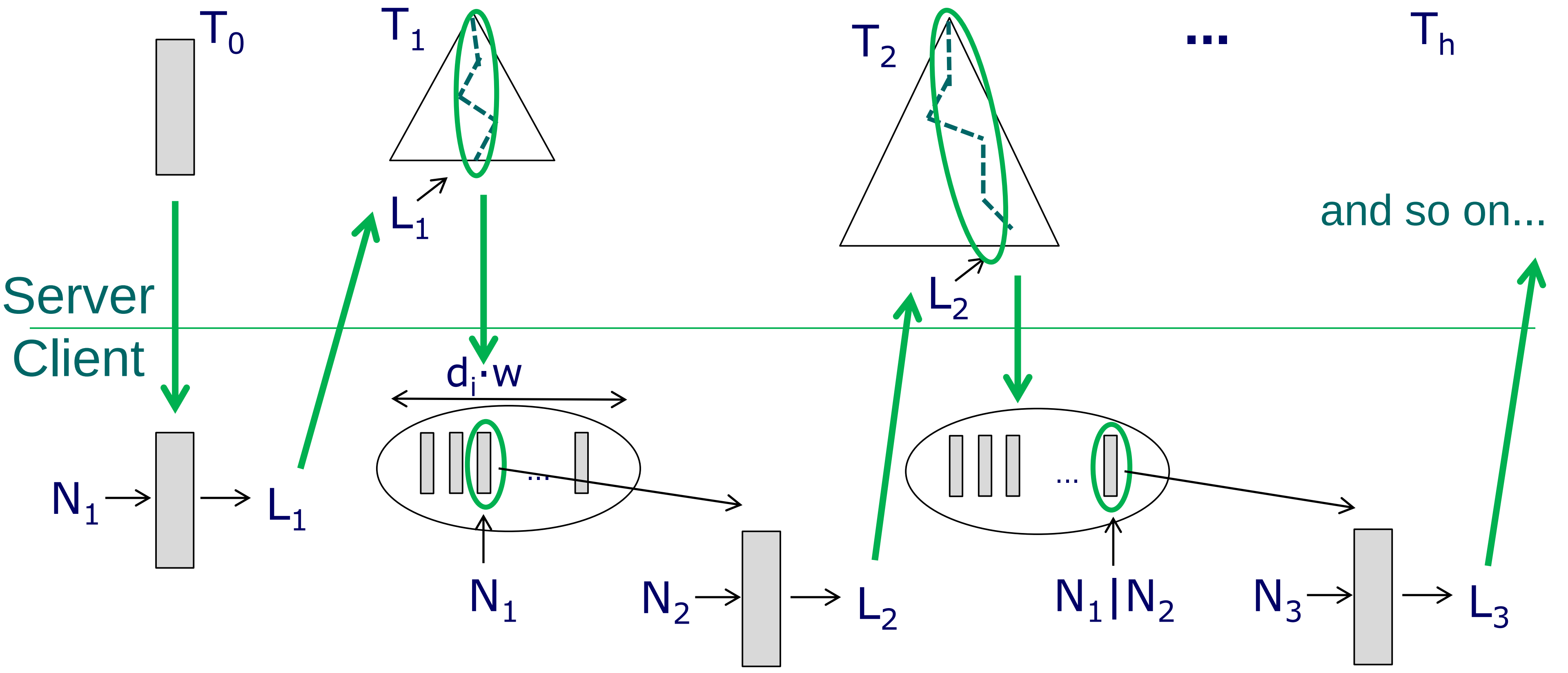
Path-ORAM Access: Recursive Tree+Array Lookup

Split address space of m bits, h chunks of $\tau = m/h$ bits

$N = [N_1 \mid N_2 \mid \ldots \mid N_h]$

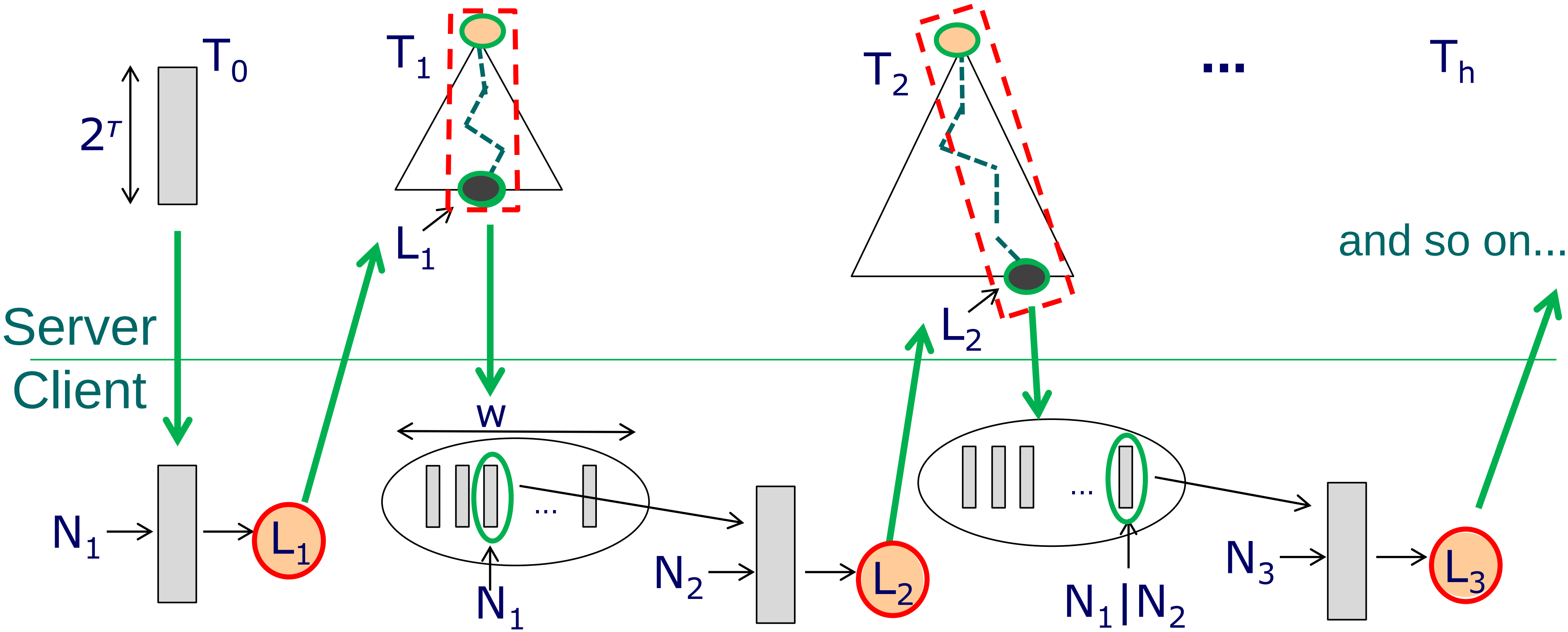
T_i is a binary tree of depth $d_i = i \cdot \tau$, tree nodes are buckets of size w

Client's code is a sequence of *array* or *dictionary* list look-ups...



SC for Path-ORAM [Shi+'11]

The other half: Path-ORAM *Eviction*



Eviction: 1) put the (modified) retrieved entry on top
2) move all* entries down towards their targets labels

SC-ORAM: To reduce circuit size, use constrained eviction strategy

SC for Tree-ORAM
Three Steps

Access: Retrieve data assoc. with searched-for address N

SS[X , N] $\rightarrow$ d s.t. (N,d) $\in$ X

Eviction.1: Compute *movement logic*, T : [n] $\rightarrow$ [n]

SS[$X_{|N}$] $\rightarrow$ SS[T]

Eviction.2: Permute path X according to T

SS[X , T] $\rightarrow$ SS[T(X)] s.t. T(X) = $X_{T(1)},\ldots,X_{T(n)}$

2PC-ORAM costs: online, passive adv (last tree, w/o small constants)

bndw: $|X|\cdot k$

comp: $|C_A|+|C_T|+|C_M|$ ciphers (+ k OT's)

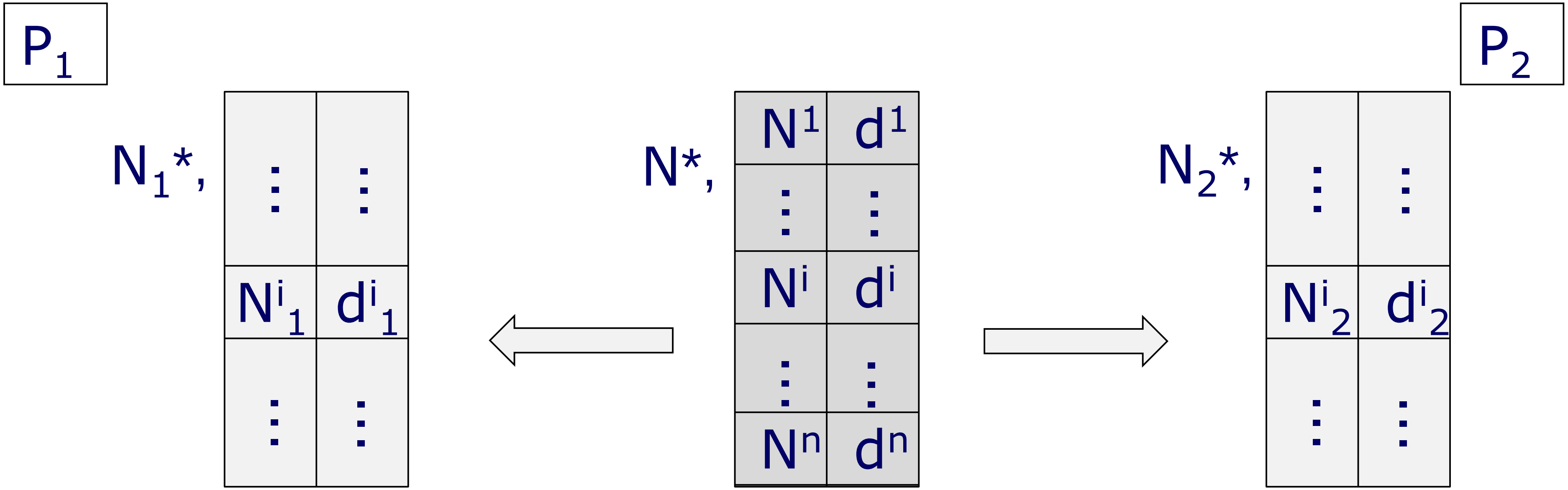
X = $(X_1,\ldots,X_n)$: tree path
k: sec.par.

3PC for Tree-ORAM

Access Step: SS[X , N] $\rightarrow$ d s.t. (N,d) $\in$ X

Client's code is a sequence of array look-ups...

3PC idea:
- secret-share all data (T_i's and N) between P_1 & P_2
- send matching entry to P_3 via *Conditional SS-OT*



P_3 $\rightarrow$ $d^i_1 \oplus d^i_2$ for i s.t. $N^i_1 \oplus N_1^* = N^i_2 \oplus N_2^*$

3PC for Tree-ORAM

Access Step: SS[X , N] $\rightarrow$ d s.t. (N,d) $\in$ X

k: sec.par.
n: # tuples in path
D: record size
m: address size

= String Equality Problem:

2PC, Yao's GC: knD bndw (+k exp's)
2PC, arith.circ.: bndw--, rounds++
2PC, DH-KE: n exp's

3PC: *Conditional Disclosure of Secrets*
[GIKM00], IT: 4nD bndw
[AC'15], crypt: 2n(m+D) bndw
$\approx$ 2x *plain* Client-Server ORAM

3PC, 2-PIR: +1 round, 2nm+$\sqrt{n}$D bnds

P_1

N_1^*,

⋮	⋮
N^i_1 / a^i_1	d^i_1
⋮	⋮

P_2

⋮	⋮
N^i_2 / b^i_2	d^i_2
⋮	⋮

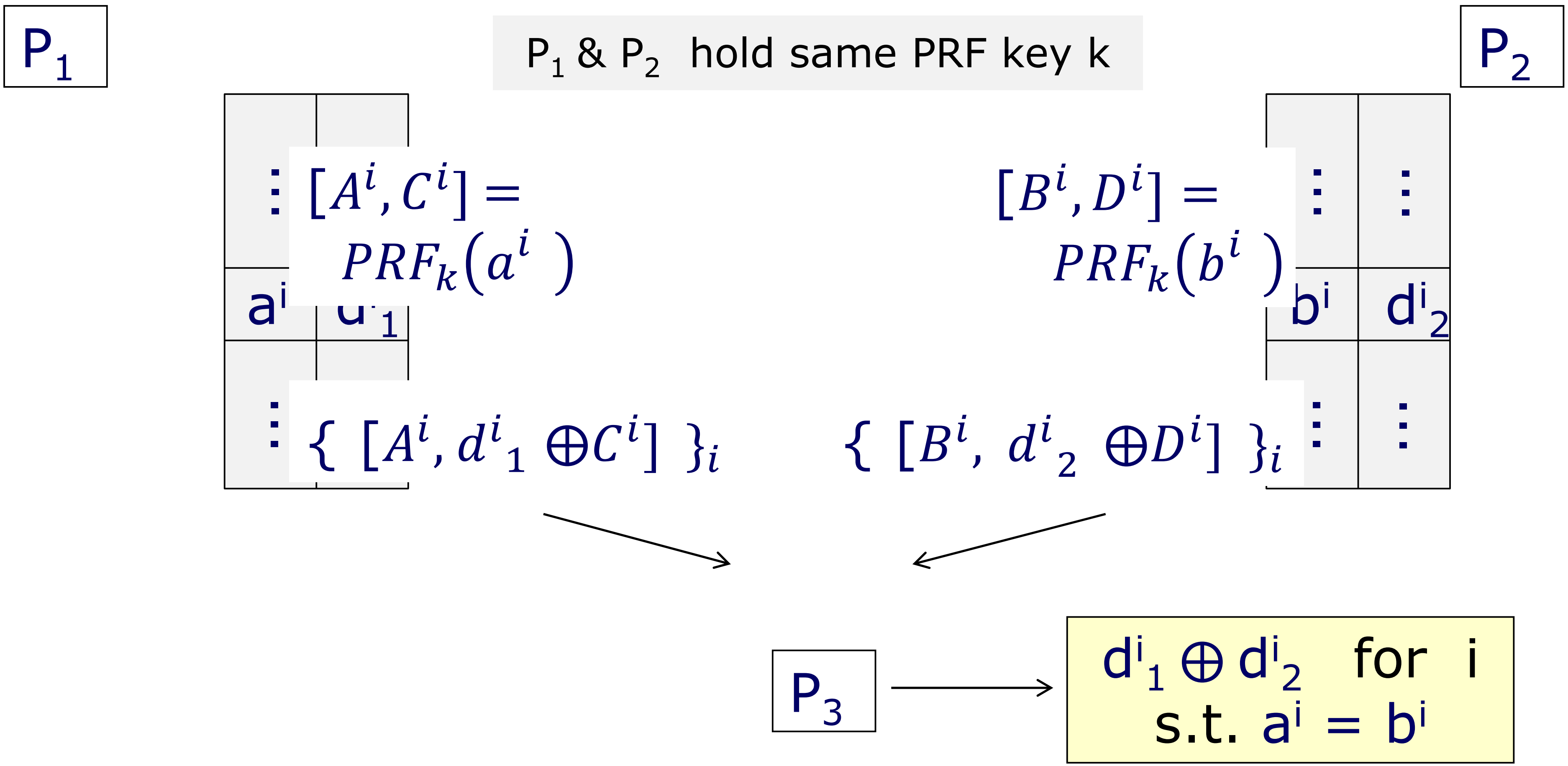
P_3 $\rightarrow$

$d^i_1 \oplus d^i_2$ for i
s.t. $a^i = b^i$

3PC for Tree-ORAM

Problem: P_3 learns position i where the $a^i = b^i$ match occurs…

3PC sol.: P_1 & P_2 shift their input lists by (the same) random offset
P_3 can learn a pointer into the shifted list (= random in [n])



Path-ORAM: from 2PC to 3PC
Access Step via CDS a.k.a. SS-COT

Access (C_A):	SS[X , N]	$\rightarrow$	d s.t. (N,d) $\in$ X
Ev.1 (C_T):	SS[$X_{\|N}$]	$\rightarrow$	SS[T]
Ev.2 (C_M):	SS[X , T]	$\rightarrow$	SS[T(X)] s.t. T(X) = $X_{T(1)},\ldots,X_{T(n)}$

	2PC-ORAM	3PC-ORAM
Acc:	*bndw:* $\|X\|\cdot k$ + *ciph:* $\|C_A\|$ + OT's	*bndw:* $\|X\|$
Ev.1:	*ciph:* $\|C_T\|$	?
Ev.2:	*ciph:* $\|C_M\|$	?

X = $(X_1,\ldots,X_n)$: tree path
k : sec.par.

- 100x cheaper access
- Benefits:
    - response time (eviction in background)
    - access inherently sequential
    - batch access with postponed eviction

3PC for Tree-ORAM
Eviction Steps

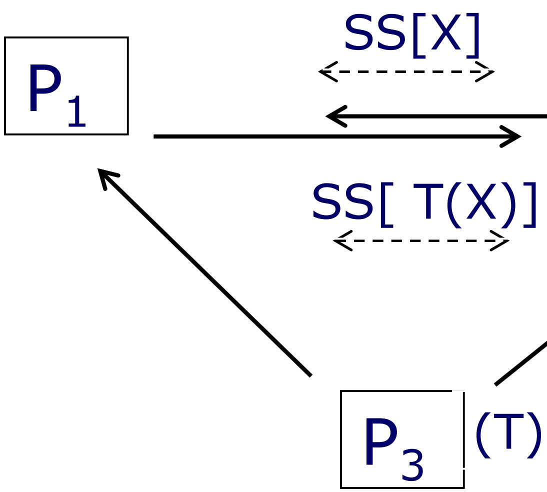
Ev.1 (C_T): SS[$X_{|N}$] → SS[T]

Ev.2 (C_M): SS[X , T] → SS[T(X)] s.t. $T(X) = X_{T(1)},\ldots,X_{T(n)}$

3PC idea: Use Yao for C_T, but make transition table T "uniform" s.t.:

Ev.1: If P_1 and P_2 locally permute secret-shared list X then P_3 can learn T *in the clear*

Ev.2: SS[X, T] → SS[T(x)] is a simple variant of OT



- 1 message flow
- bndw: 4|X|
- no crypto ops

Uniform Transition Table T [AC'15]:

- T moves 2 items from node i to i+1
- P_1&P_2 shuffle each bucket
- P_3 learns 2 first movable/empty items after 2 random shifts

Path-ORAM: from

Access (SS-COT), Ev.1 (

k=128, m=32

2PC: $\alpha k=512$ for $\alpha=4$

$\Rightarrow$ *D = 16B*

$\Rightarrow$ *τ = 2* $\Rightarrow$ *16 rounds*

3PC: $\alpha+k=384$ for $\alpha=256$

$\Rightarrow$ *D = 1KB*

$\Rightarrow$ *τ = 8* $\Rightarrow$ *4 rounds*

Access (C_A): SS[X , N] $\rightarrow$ d s

Ev.1 (C_T): SS[$X_{|N}$] $\rightarrow$ SS

Ev.2 (C_M): SS[X , T] $\rightarrow$ SS

2PC-ORAM

Acc: *bndw:* $|X| \cdot k$ + *ciph:* $|C_A|$ +OT's

Ev.1: *ciph:* $|C_T|$

Ev.2: *ciph:* $|C_M|$

bndw: $|X|k = n(m+|d|)k \approx m^2 w \cdot \alpha k$

ciph: $m^2 w \cdot (\alpha + \alpha_{CT} + \alpha \cdot \alpha_{CM})$ +OT's

3PC-ORAM

bndw: $|X|$

ciph: $|C_T|$ + *bndw: nm·k*

bndw: $|X|$

$m^2 w \cdot (\alpha + k)$

$m^2 w \cdot \alpha_{CT}$

$X = (X_1,\ldots,X_n)$: tree path	X_i = (addr.,data)	k : sec.par.
$n = m \cdot w$	m : address size	w : bucket width
$\alpha = \max(2^\tau, D/m)$	τ : addr. chunk size	D : record size
$|d| \approx m \cdot \alpha$	α_{CT}, α_{CM}: circ.comp. of C_T,C_M (=circuit size / input length)	

Path-ORAM: from 2PC to 3PC

Access (SS-COT), Ev.1 (Yao), Ev.2 (SS-OT)

	2PC-ORAM	3PC-ORAM
Acc:	*bndw:* $\lvert X\rvert \cdot k$ + *ciph:* $\lvert C_A\rvert$ +OT's	*bndw:* $\lvert X\rvert$
Ev.1:	*ciph:* $\lvert C_T\rvert$	*ciph*: $\lvert C_T\rvert$ + *bndw: nm·k*
Ev.2:	*ciph:* $\lvert C_M\rvert$	*bndw:* $\lvert X\rvert$
	bndw: $\lvert X\rvert k = n(m+\lvert d\rvert)k \approx m^2 w \cdot \alpha k$	$m^2 w \cdot (\alpha+k)$
	ciph: $m^2 w \cdot (\alpha+\alpha_{CT}+\alpha \cdot \alpha_{CM})$	$m^2 w \cdot \alpha_{CT}$

AC'15: 3PC with simplistic eviction: very low α_{CT}, $w=O(m+k) \approx 100$

WCS'15: "Circuit-ORAM": 2PC, greedy eviction, higher α_{CT}, $w=3$, $\alpha_{CM}=2$

New work: 2PC with same eviction as in Circuit-ORAM, slightly higher α_{CT}

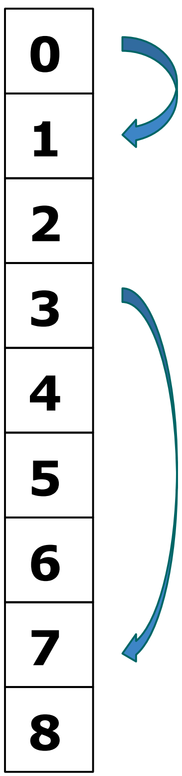
$X = (X_1,\ldots,X_n)$: tree path	X_i = (addr.,data)	k : sec.par.
$n = m \cdot w$	m : address size	w : bucket width
$\alpha = \max(2^\tau, D/m)$	τ : addr. chunk size	D : record size
$\lvert d\rvert \approx m \cdot \alpha$	α_{CT}, α_{CM}: circ.comp. of C_T,C_M (=circuit size / input length)	

Circuit-ORAM Eviction [WCS'15]

From 2PC to 3PC : Making Transition Table T *Uniform*

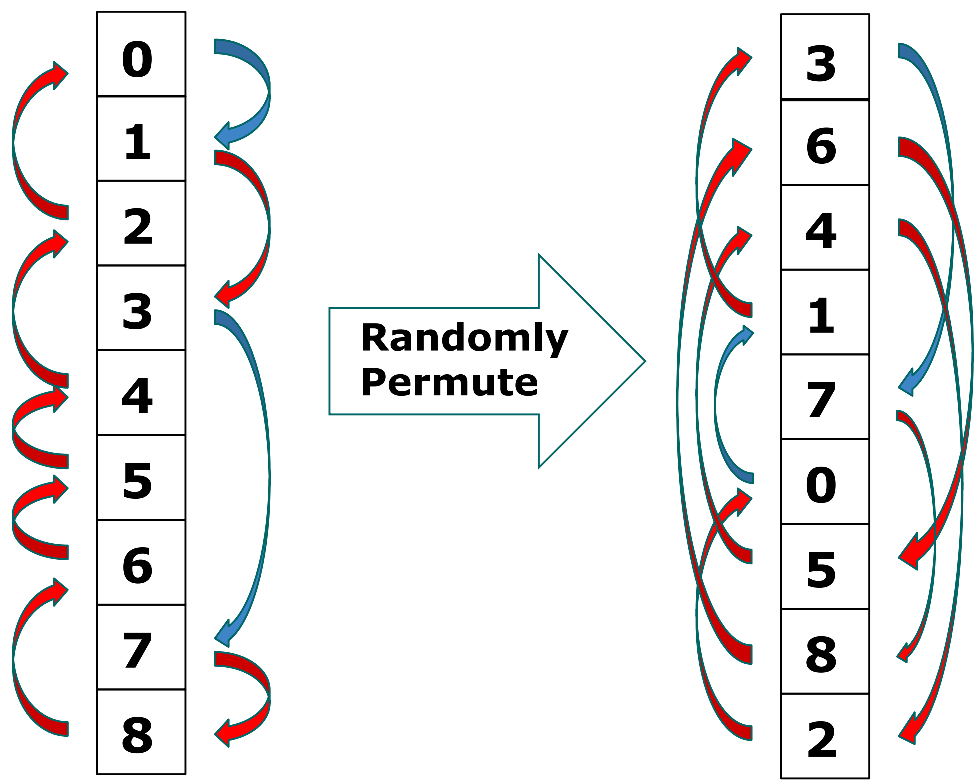
Circuit ORAM Eviction:

greedy: "deepest goes first"



Making it Uniform:

1. Fill-in jumps so T is a cycle
2. Reveal $(\Pi\circ T)(i)$ instead of $T(i)$ for \$ Π
   - permute outside Garb.Circ.
   - +2 rounds for (de-)mask/permute

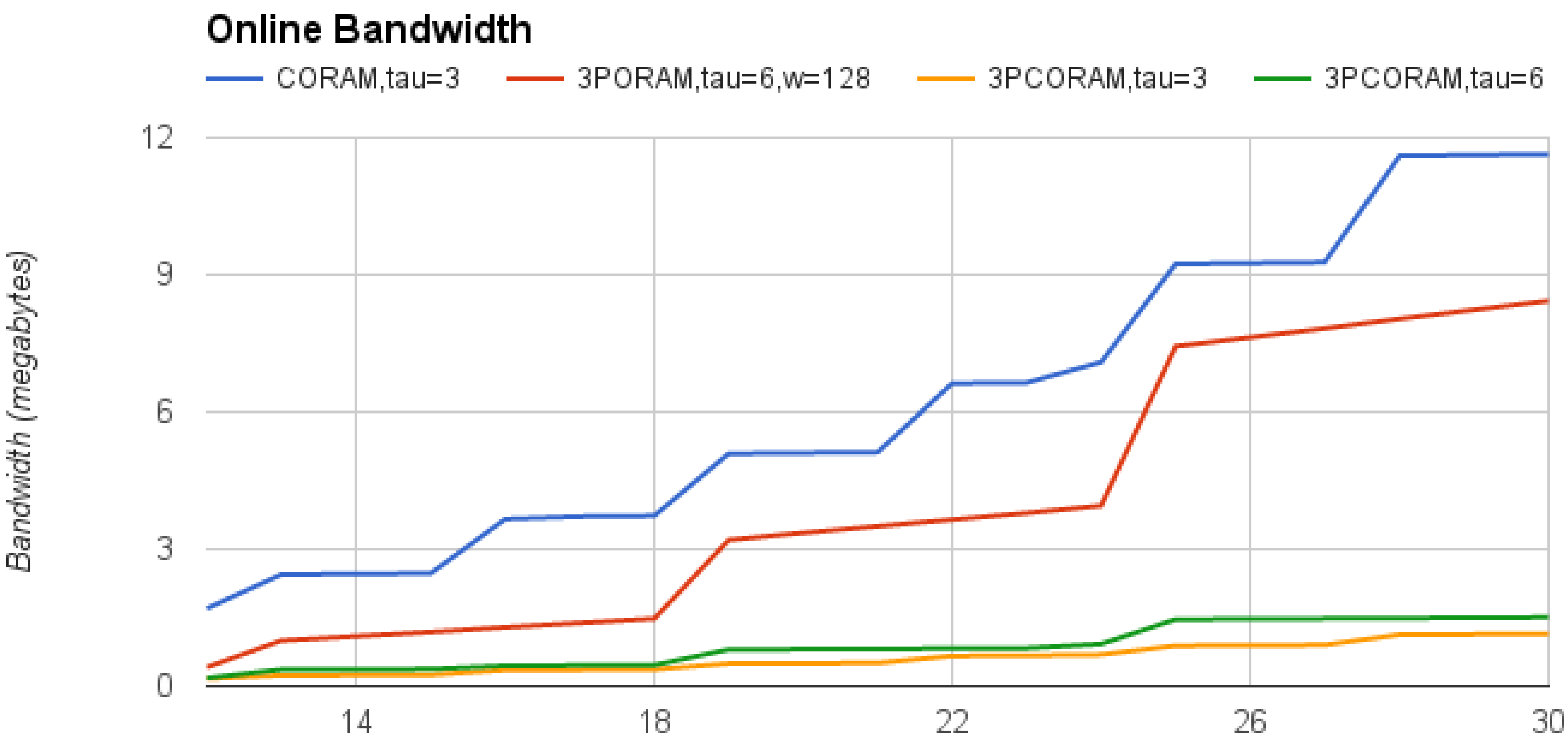


Path-ORAM: 2PC vs. 3PC

2PC-ORAM: $bndw = m^2w \cdot \alpha k$
$|circ| = m^2w \cdot (\alpha + \alpha_{CT} + \alpha \cdot \alpha_{CM})$

3PC-ORAM: $bndw = m^2w \cdot (\alpha + k)$
$|circ| = m^2w \cdot \alpha_{CT}$

k : sec.par (=128). m : address size w : bucket width (=3)
α = max(2^T,D/m)=2^T T : addr. chunk size D : record size (=4B)
α_{CT} (= ?) α_{CM} (=2) : circ.comp. of C_T,C_M (=circuit size / input length)



3PC: ~10 x

m: address size

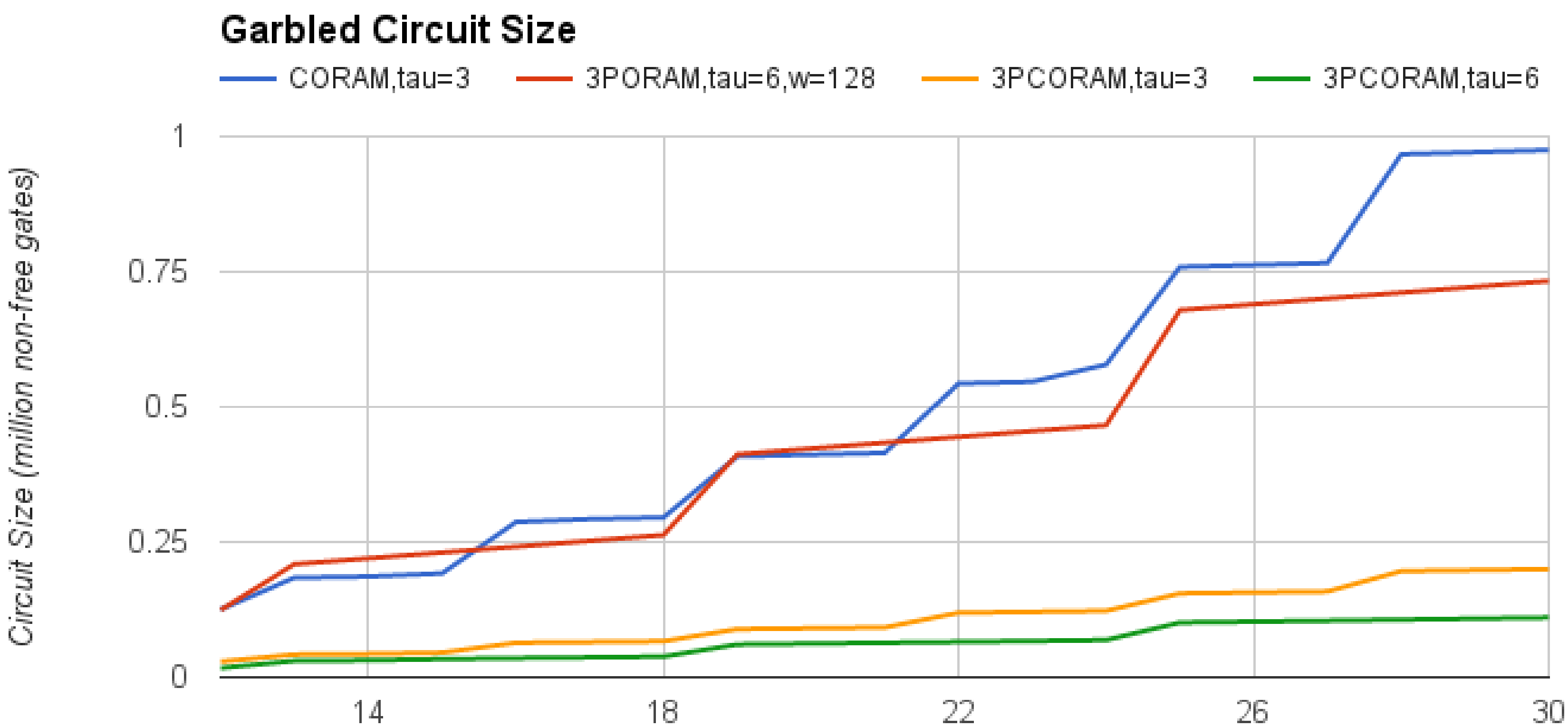
CORAM: 2PC [WCS'15]: higher α_{CT}, w=3, α_{CM}=2
3PORAM: 3PC [AC'15]: low α_{CT}, w=O(m+k) ≤ 128
3PCORAM: 3PC [new]: same α_{CT} (~1.2x) and w as in CORAM

Path-ORAM: 2PC vs. 3PC

2PC-ORAM: $bndw = m^2 w \cdot \alpha k$
$|circ| = m^2 w \cdot (\alpha + \alpha_{CT} + \alpha \cdot \alpha_{CM})$

3PC-ORAM: $\text{bndw} = m^2 w \cdot (\alpha + k)$
$|\text{circ}| = m^2 w \cdot \alpha_{CT}$

k : sec.par (=128). m : address size w : bucket width (=3)
$\alpha = \max(2^{\tau}, D/m) = 2^{\tau}$ τ : addr. chunk size D : record size (=4B)
α_{CT} (= ?) α_{CM} (=2) : circ.comp. of C_T, C_M (=circuit size / input length)



m: address size

3PC: 5-10x

CORAM: 2PC [WCS'15]: higher α_{CT}, w=3, α_{CM}=2
3PORAM: 3PC [AC'15]: low α_{CT}, w=O(m+k) ≤ 128
3PCORAM: 3PC [new]: same α_{CT} (~1.2x) and w as in CORAM

Path-ORAM: 2PC vs. 3PC

2PC-ORAM: $bndw = m^2 w \cdot \alpha k$

$|circ| = m^2 w \cdot (\alpha + \alpha_{CT} + \alpha \cdot \alpha_{CM})$

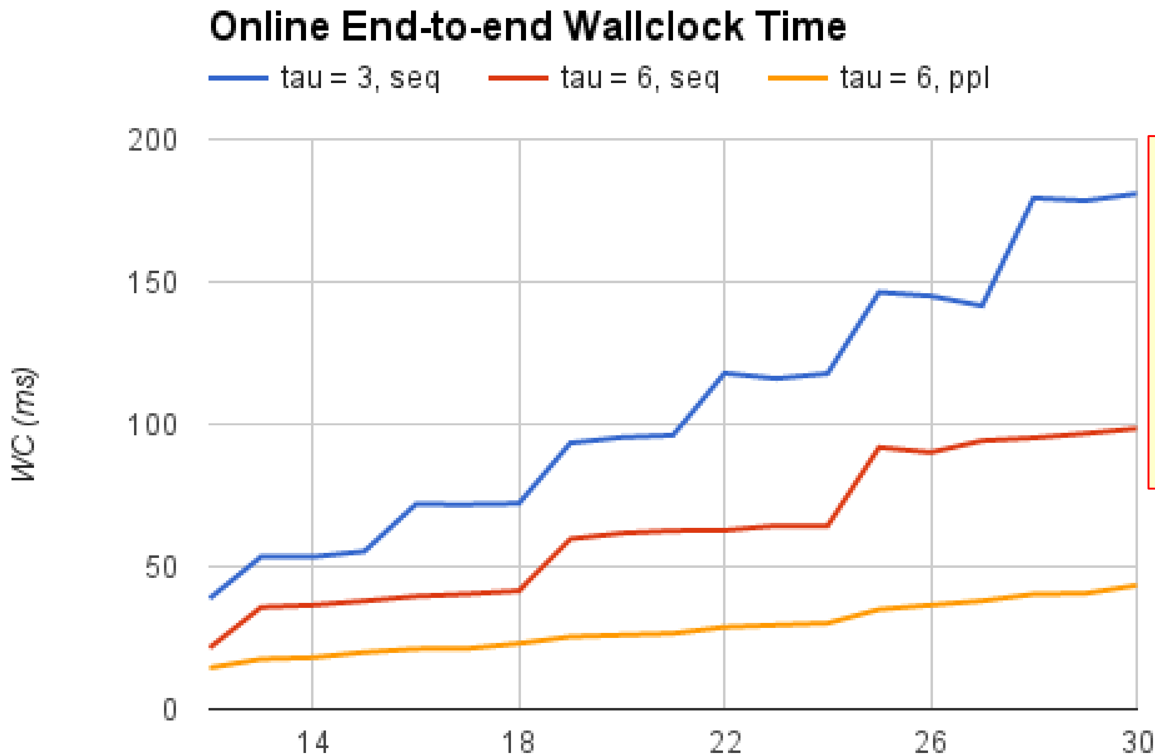
3PC-ORAM: $bndw = m^2 w \cdot (\alpha + k)$

$|circ| = m^2 w \cdot \alpha_{CT}$

k : sec.par (=128). m : address size w : bucket width (=3)

$\alpha = \max(2^{\tau}, D/m) = 2^{\tau}$ τ : addr. chunk size D : record size (=4B)

α_{CT} (= ?) α_{CM} (=2) : circ.comp. of C_T,C_M (=circuit size / input length)



m: address size

3PC:

Larger τ → 2x

- **1.5x for CPU**
- **2-3x in rounds**

Pipelining → 2x

3PCORAM: 2PC [WCS'15]: higher α_{CT}, w=3, α_{CM}=2

3PORAM: 3PC [AC'15]: low α_{CT}, w=O(m+k) ≤ 128

3PCORAM: 3PC [new]: same α_{CT} (~1.2x) and w as in CORAM

Questions, Directions

Examples:

- pipelining, batched access with postponed eviction, *parallel access*
- MPC for other data-structures
- general (t,n): the "P_1/P_2 permute & P_3 gets outputs" idea doesn't scale...
- malicious security? covert security?
- secure-computation-friendly multi-server ORAM ([LO'14]: client uses PRF)