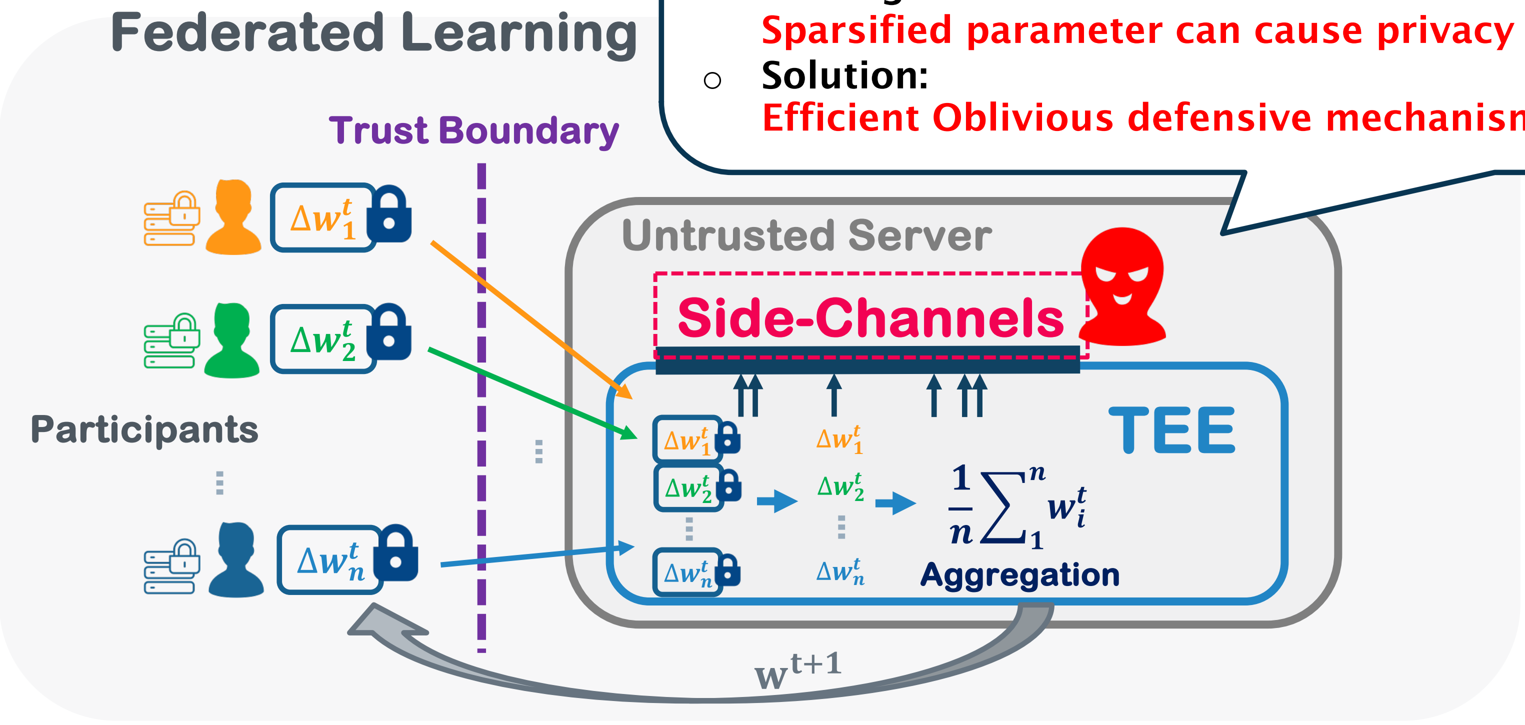
2023.8.29 VLDB'23

OLIVE: Oblivious Federated Learning on TEE against the risk of Sparsification

Fumiyuki Kato [1], Yang Cao [2], Masatoshi Yoshikawa [3]
[1] Kyoto University, [2] Hokkaido University, [3] Osaka Seikei University,

Summary

- Motivation:
  FL with server-side TEE (Figure below)
- Problem:
  Privacy risk of Side-channels in FL with TEE
- Findings:
  Sparsified parameter can cause privacy leak
- Solution:
  Efficient Oblivious defensive mechanism



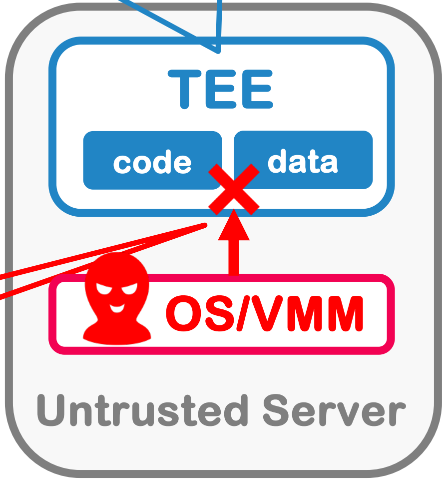
Background

Trusted Execution Environment (TEE)

- **TEE enables secure computation on remote machine**
- **Intel SGX – one of TEE implementations**
  - **1. Memory encryption**
    - Can hide code and data against privileged software (OS/VMM)
  - **2. Remote Attestation**
    - Can verify the integrity of the code and data externally

- **Memory access pattern leakage via side-channels**
  - **Cache-based (*Prime+Probe*) [1]**
  - **Page-based [2]**

TEE is (hardware-assisted) isolated environment for code and data

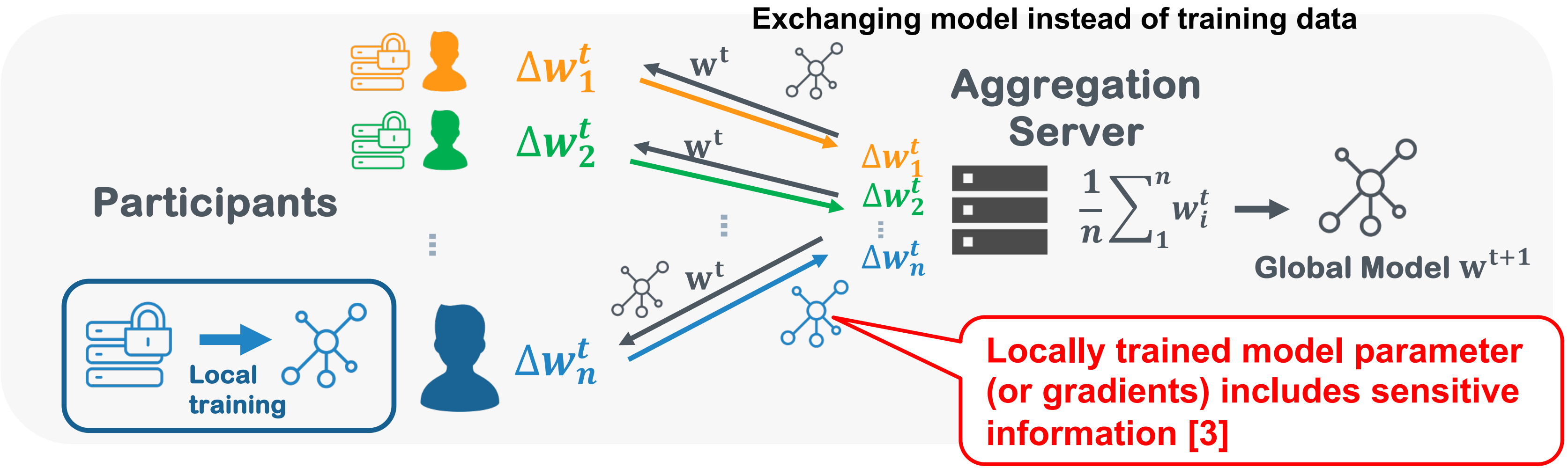


Access patterns can be visible regardless of memory encryption

[1] Moghimi, el.al., How SGX amplifies the power of cache attacks." International Conference on Cryptographic Hardware and Embedded Systems . Springer, Cham, 2017.
[2] Y. Xu, el.al., "Controlled-channel attacks: Deterministic side channels for untrusted operating systems." 2015 IEEE Symposium on Security and Privacy.

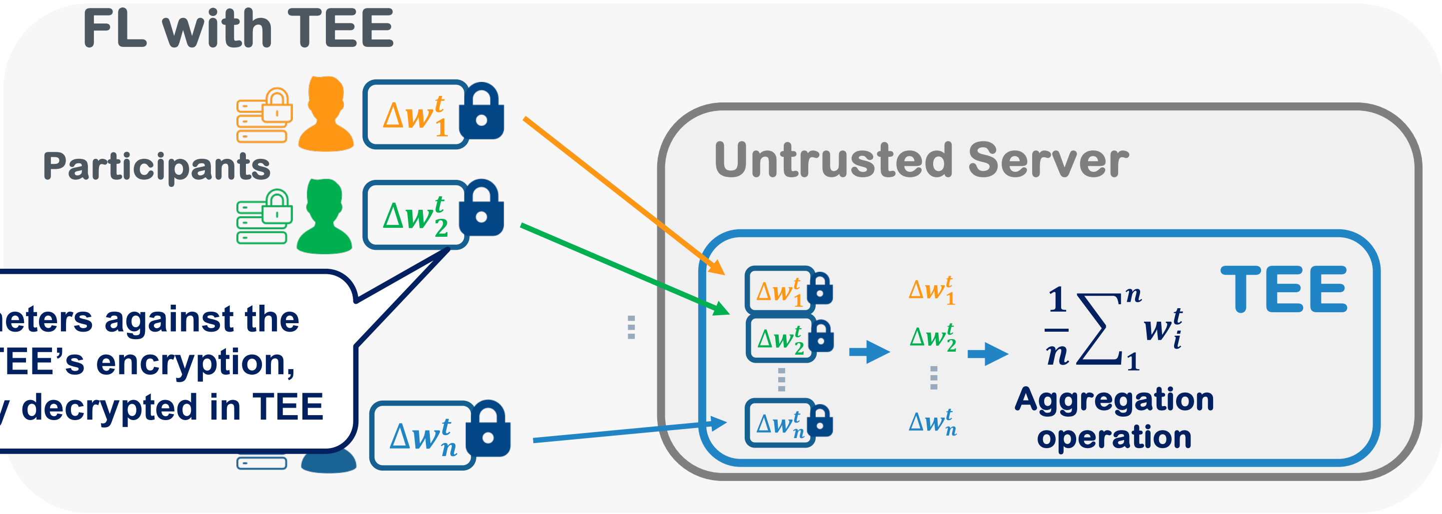
Federated Learning (FL)

- Collaborative ML scheme with
  - many participants
  - a central aggregation server
- **Problem: Locally trained model is sufficient to leak sensitive information**



[3] Zhu, Ligeng, Zhijian Liu, and Song Han. "Deep leakage from gradients." Advances in neural information processing systems 32 (2019).

FL with server-side TEE

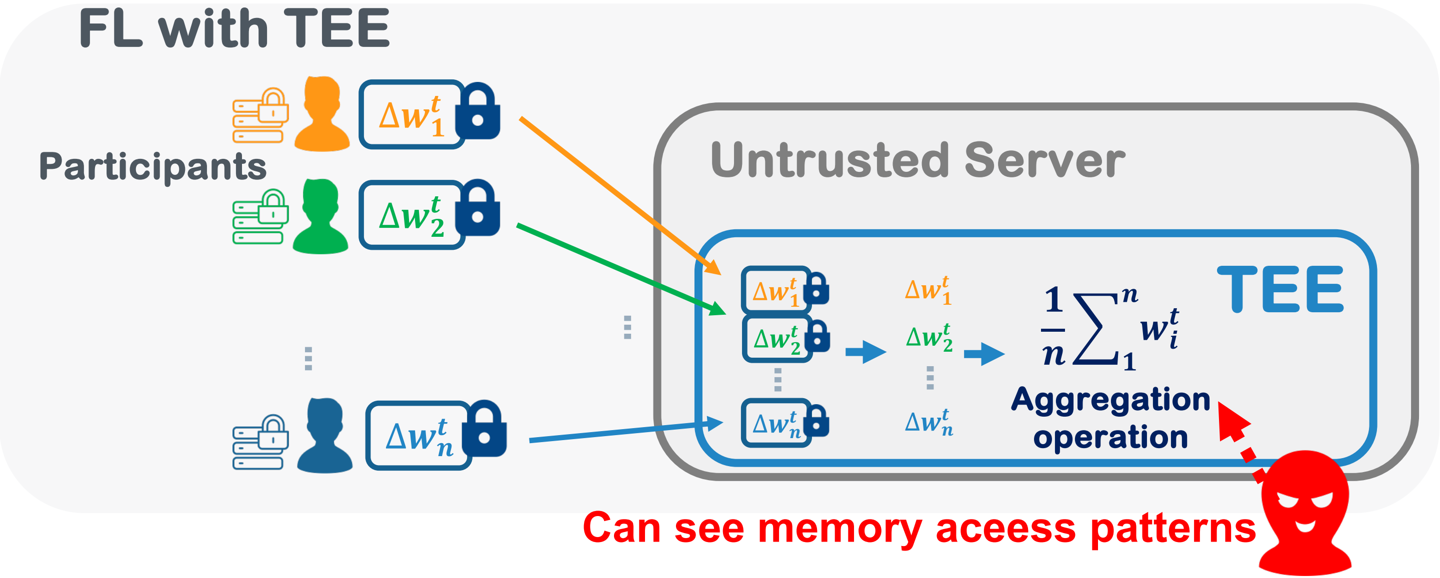


Hide parameters against the server by TEE's encryption, which is only decrypted in TEE

The combination of CDP improves the utility of DP-FL. (vs Shuffle DP)

	Trust model	Utility
CDP-FL [4, 28, 50, 84]	Trusted server	Good
LDP-FL [45, 74, 81, 92]	Untrusted server	Limited
Shuffle DP-FL [29, 44]	Untrusted server + Shuffler	Shuffle DP-FL ≤ CDP-FL
Olive (Ours)	**Untrusted Server with TEE**	**Olive = CDP-FL**

Problem of FL with server-side TEE



Problem: The impact of side-channels of TEE is unknown

- What is the specific privacy risks?
- What is practical protection against the attacks?

Contributions

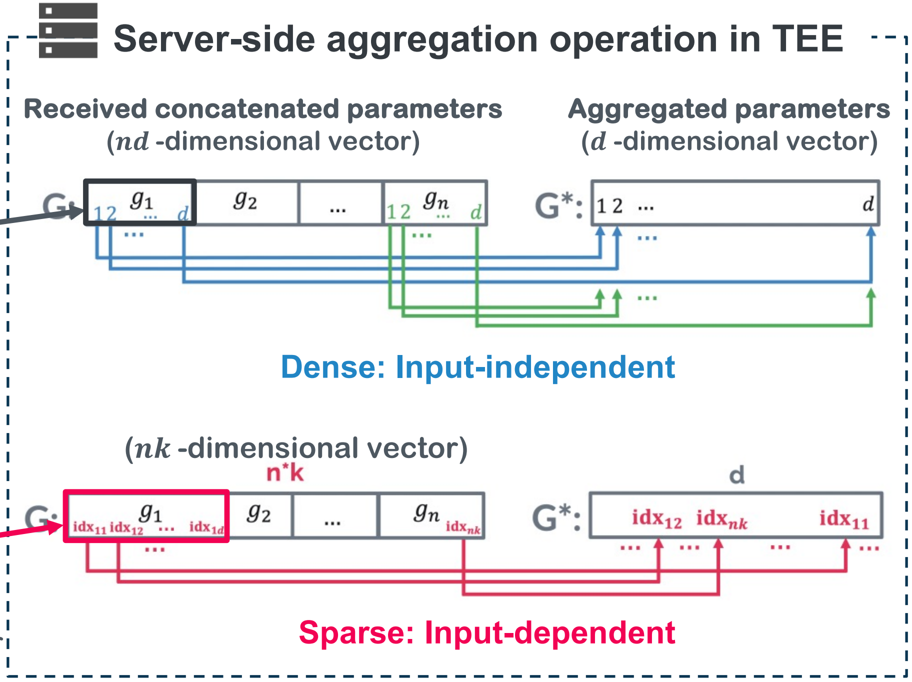
- **In FL with server-side TEE, we study both of Attack and Defense in terms of memory access pattern leaks**
- **Attack**
  - We show that the sparsified parameters often used in FL can leak sensitive information via memory access patterns
  - We demonstrate that privacy attacks are possible using information obtained from memory access patterns
- **Defense**
  - We design an efficient oblivious FL aggregation algorithm
  - We evaluate the proposed defensive mechanism on real-world scales

Attack analysis

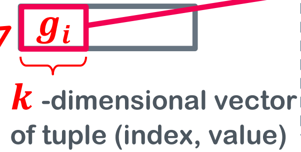
Memory Access Pattern Analysis on Aggregation Operation of FL

- If parameter is *Dense*, the memory access of the aggregation operation is independent from inputs

- *In Sparsified setting*, memory access pattern can leak the selected parameter indices (data-dependent)



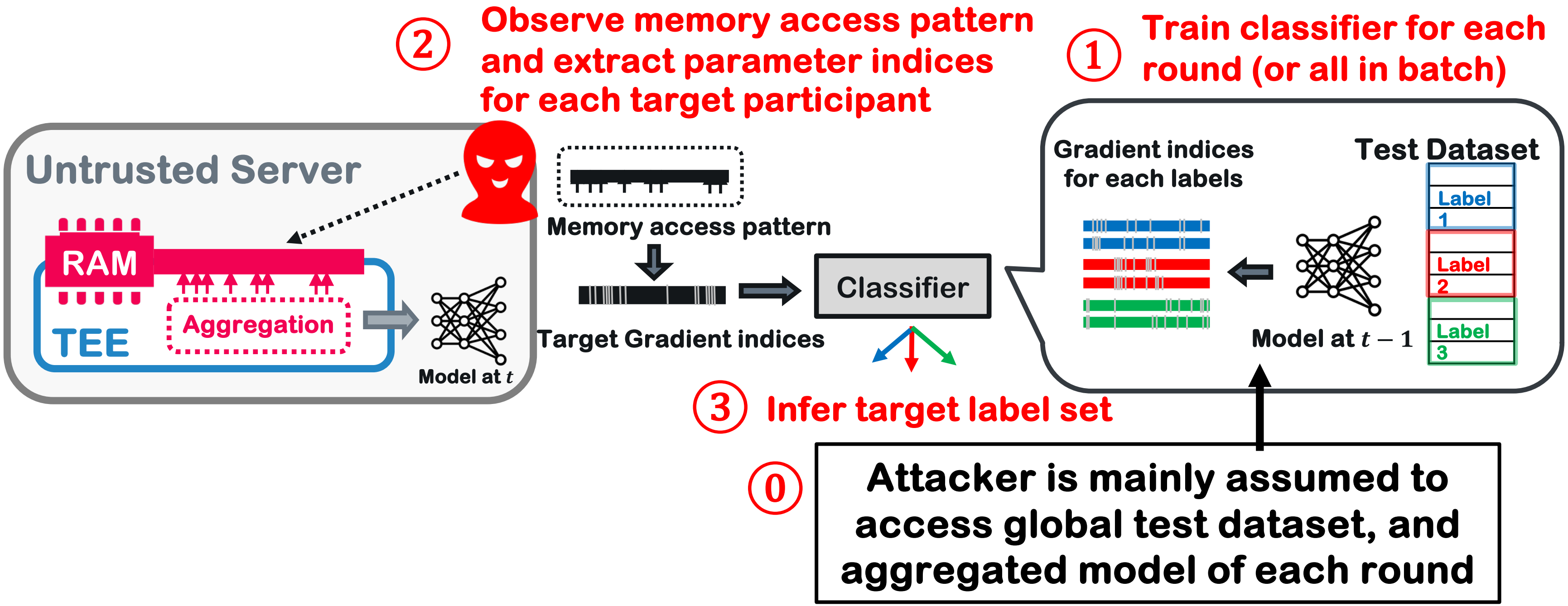
(Data-dependent) Top-k sparsification is often used in FL for better Comms-cost and better utility [4]



[4] Mitra, Aritra, et al. "Linear convergence in federated learning: Tackling client heterogeneity and sparse gradients." Advances in Neural Information Processing Systems 34 (2021)

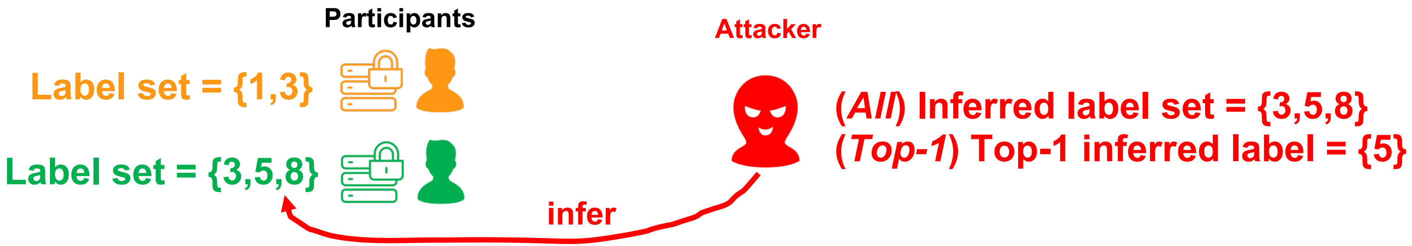
Overview of Attack Design

- To show the leaked information can leak private information
- The goal is to infer the sensitive label set of the target participant

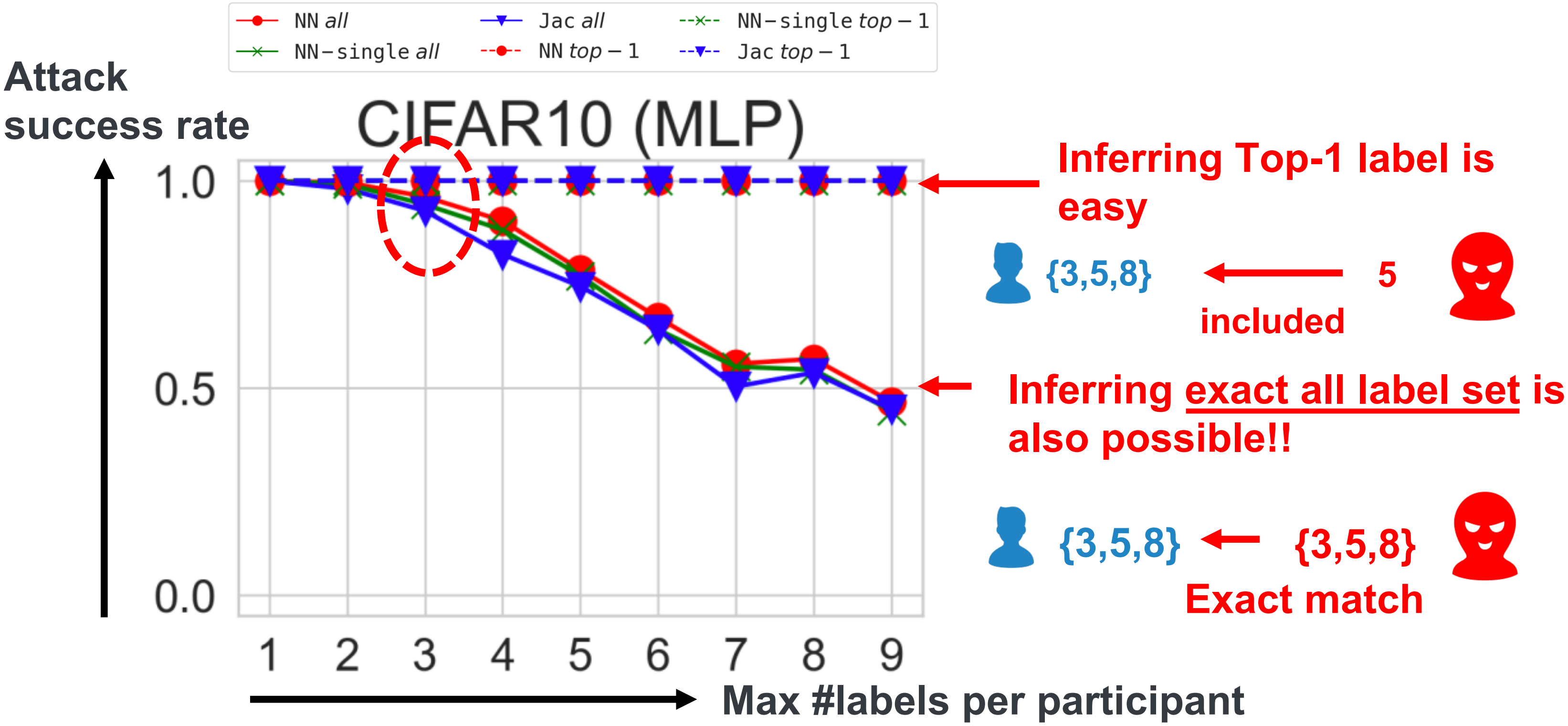


Empirical Evaluation: Setup

- Dataset
  - MNIST and CIFAR100 (, and Purchase100 (tabular dataset) in the paper)
- FL setting
  - Sparsification: with Top-10% sparsification
  - The maximum #Labels of each participant is controlled (#Participants: 1000)
- **Evaluation Metrics**
  - *All*: The ratio that predicted labels exactly match target label set
  - *Top-1*: The ratio that top-1 predict-scored label is included in target label set
    - Weakest privacy leak



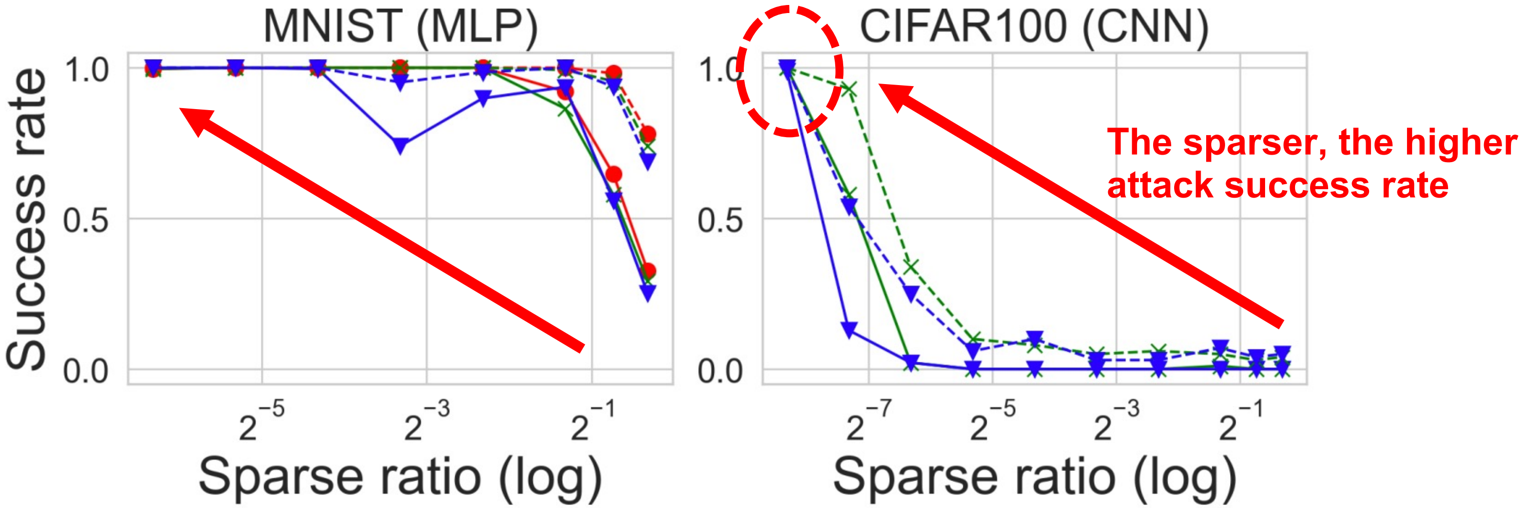
Result: Attack is overall successful



The more labels a participant has, the harder it is to infer

The sparser, the easier the attack

When sparse ratio=0.3%, attacks are almost 100% successful even we have 100 class labels.
(Common sparse ratio in FL is, for example, 0.1% [5])



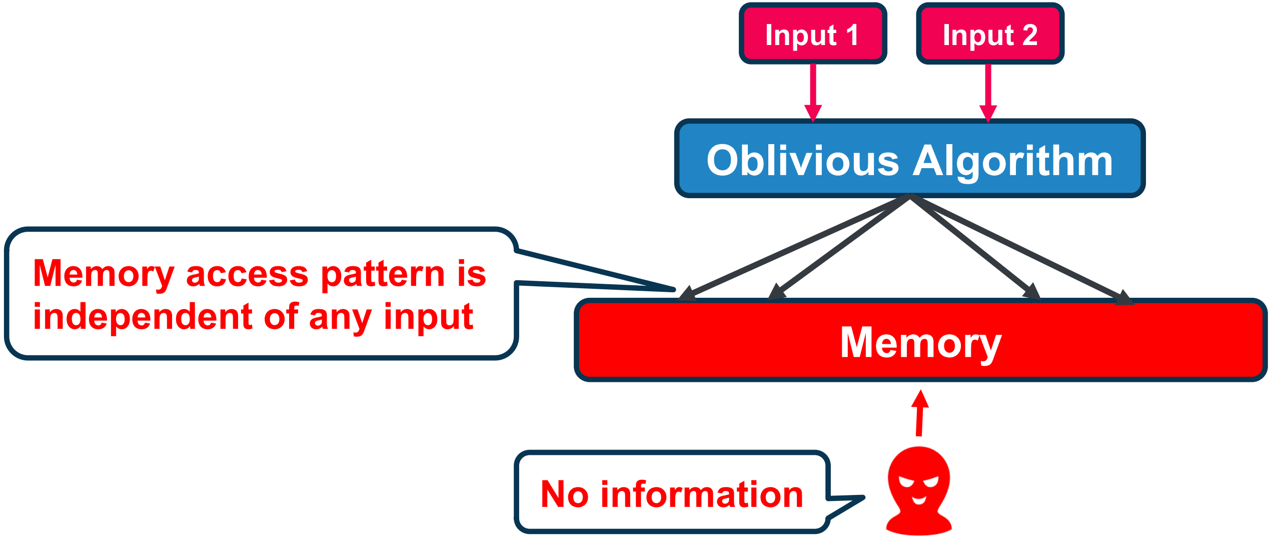
The number of labels of participants is fixed at 2.

[5] Shi, Shaohuai, et al. "Understanding top-k sparsification in distributed deep learning." arXiv preprint arXiv:1911.08772 (2019).

Defensive mechanism

Oblivious Algorithm

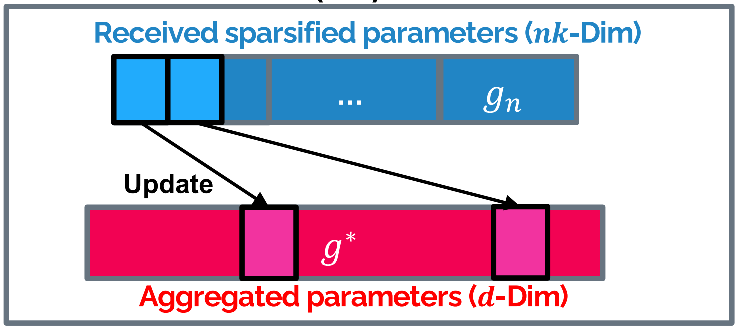
- **Oblivious algorithm** is an algorithm whose memory access pattern is independent of the input values
  - No problem if memory access pattern leaks from side-channels



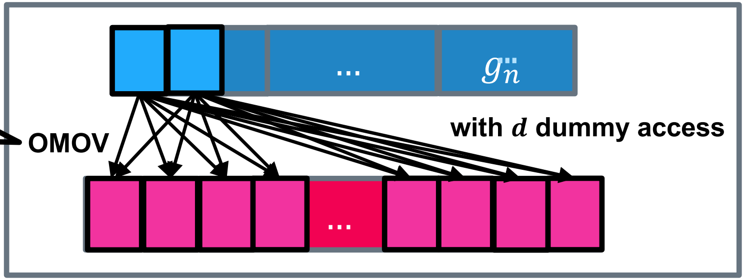
Oblivious Algorithm: Baseline

- Non-oblivious method
  - $O(nk)$
- Baseline method (oblivious)
  - Full memory access approach
  - $O(nkd)$
    - n: #Participants
    - k: Dimensions of the sparsified model
    - d: Dimensions of the dense model

Non-oblivious $O(nk)$



Baseline oblivious $O(nkd)$

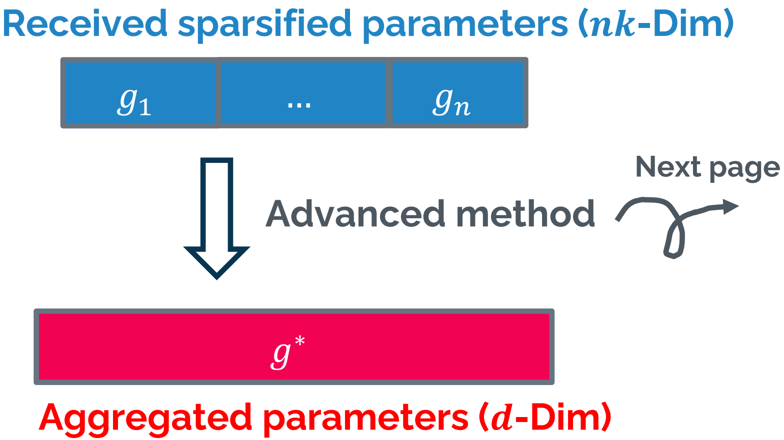


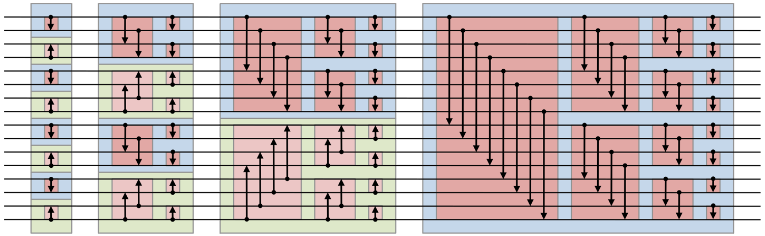
Using CMOV (of x86 instruction)-based oblivious primitive (**OMOV**) to ensure the program's execution path, including conditional branches, oblivious

But, can we make more efficient oblivious algorithm for this purpose?

Oblivious Algorithm: Advanced

- Advanced method
  - $O\left((nk+d)\log^2(nk+d)\right)$
  - Using ***oblivious sort***
    - Bitonic sort causes fixed memory access pattern





Sorting network of Bitonic sort
(https://en.wikipedia.org/wiki/Bitonic_sorter)

Overview of Advanced method

Received sparsified parameters (nk-Dim)

Zero-initialized parameter (d-Dim)

$\Rightarrow$ concatenate

G $\boxed{g_1 \mid \dots \mid g_n}$ + $\boxed{(1,0),(2,0),(3,0),\dots,(d-1,0),(d,0)}$

$\Rightarrow$ oblivious sort

$(1,0),(1.3),(1.2),(1.3),(2,0),(2.1),(3,0),\ \dots,(d-1,0),(d-1,1)(d,0),(d,2)$

$\Rightarrow$ oblivious fold

\+ \+ \+ \+ \+ \+

$(M,0),(M,0),(M,0),(1.8),(M,0),(2.1),(M,0),\dots,(M,0),(d-1,1)(M,0),(d,2)$

$O(nk+d))$

OMOV OMOV OMOV …

$\Rightarrow$ oblivious sort

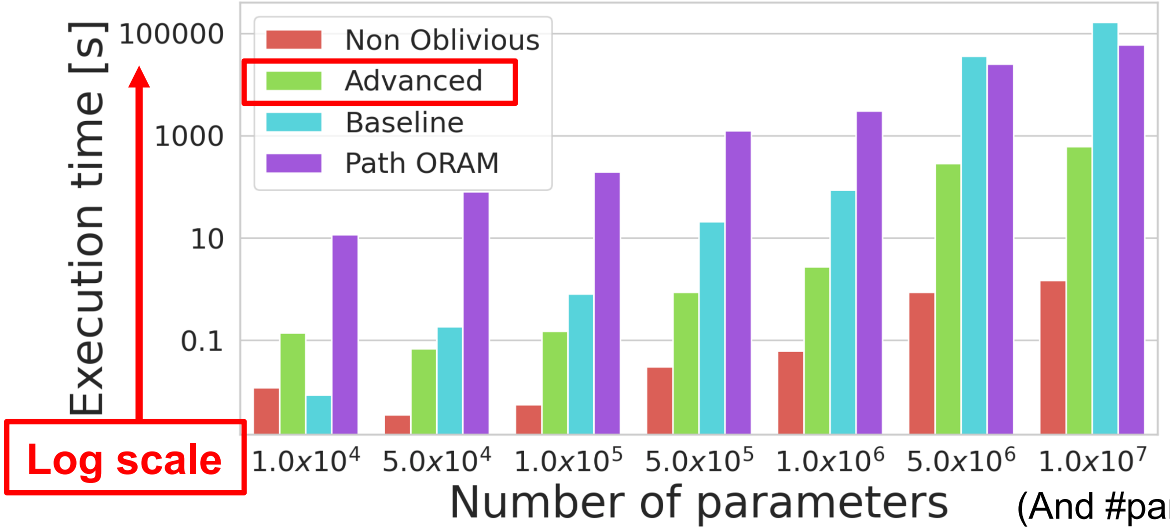
$\boxed{(1.8),(2.1),\dots,(d-1,1),(d,2),}(M,0),(M,0),\ \dots\ ,(M,0)$

Aggregated parameters (the first d-Dim vector)

Evaluation

Most efficient ORAM implementation
But, with some modification to
implement on SGX's security model [6]

- Non Oblivious, Path ORAM (ZeroTrace [6]), Baseline, Advanced
- with Intel SGX (PRM: 128 MB)



(And #participant =100)

Advanced method is reasonably fast with over millions of parameters

[6] Sajin Sasy, Sergey Gorbunov, and Christopher Fletcher. "ZeroTrace: Oblivious memory primitives from Intel SGX." Symposium on Network and Distributed System Security (NDSS). 2018.

Conclusions

- **In FL with server-side TEE, we studied both of Attack and Defense**
- **Attack**
  - We show that the sparsified parameters often used in FL can leak sensitive information via memory access patterns
  - We demonstrated that privacy attacks are possible using information obtained from memory access patterns
- **Defense**
  - We designed an efficient oblivious FL aggregation algorithm
  - We evaluated the proposed defensive mechanism on real-world scales

Appendix

Assumptions

- Attacker can access
  - Test dataset
  - Aggregated global model in each round
  - Observe access patterns (from side-channels)

- Top-k sparsification is used

Attack: Cache line-level

- Even if the granularity of the observation becomes a cachet line, the results don't change much

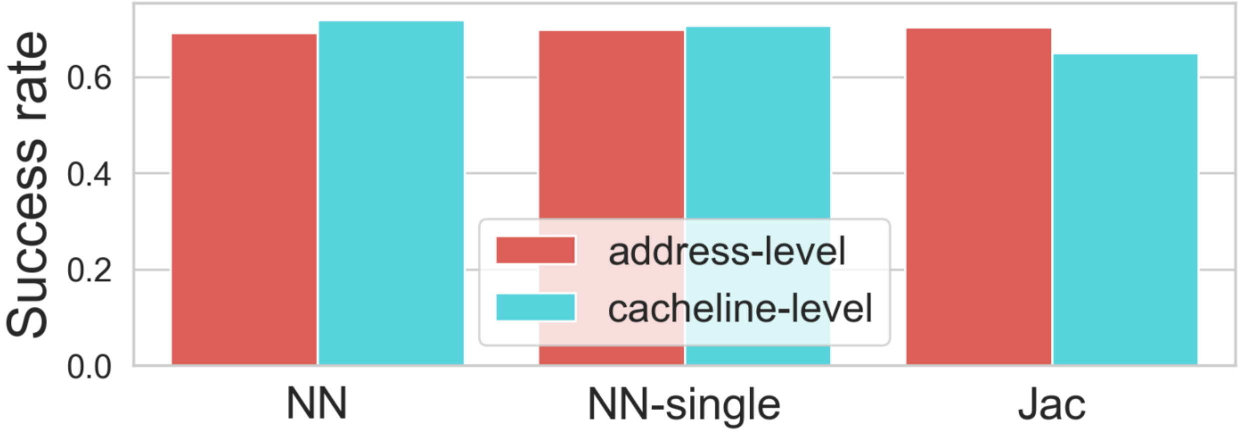
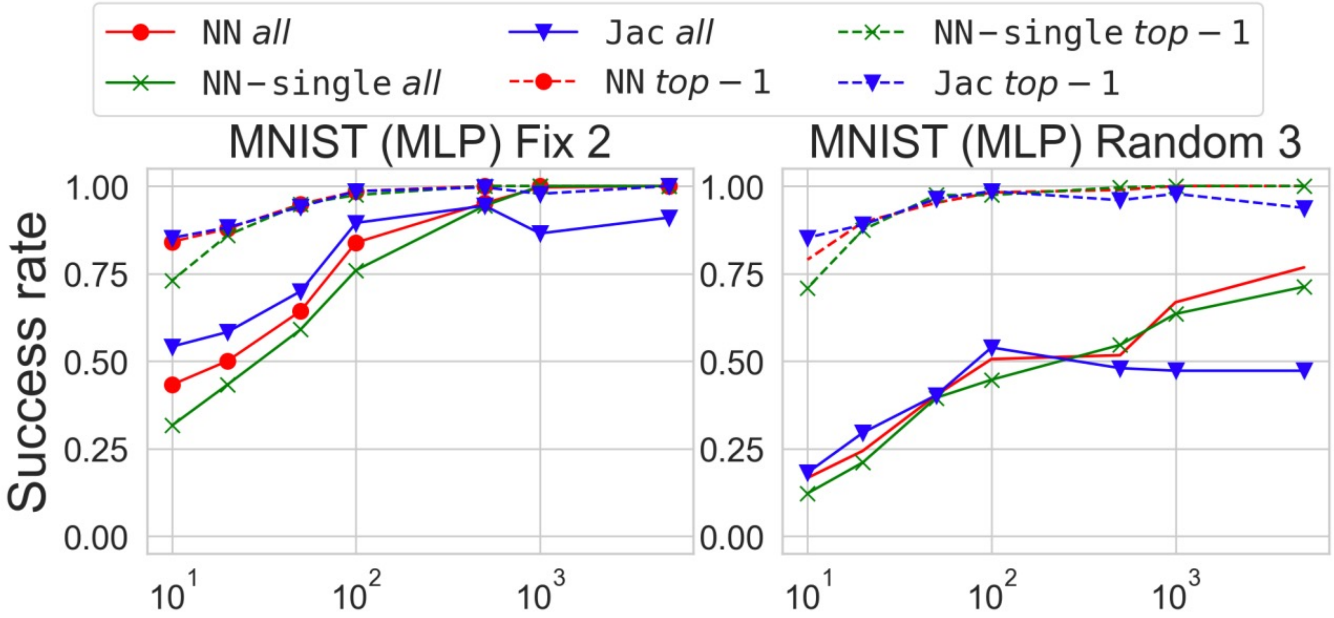


Figure 7: Cacheline-level leakage on CNN of CIFAR10: Attacks are possible with at least slightly less accuracy.

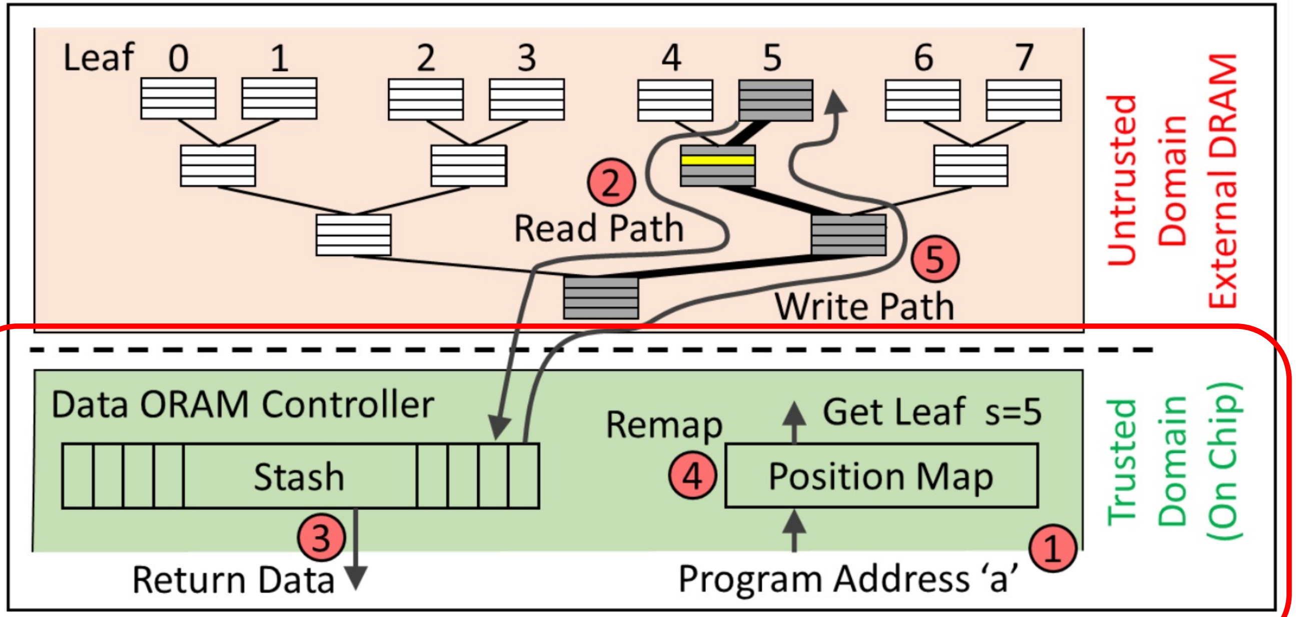
Attack: Various test dataset

- Test dataset can be very small



Path ORAM [A.1]

Access cost can be reduced to O(log N)



In the trust model of SGX, Trusted Domain of ORAM also needs to be oblivious [A.2]

[Image source: *https://scl.engr.uconn.edu/research/oram.php]*

[A.1] Stefanov, E., Van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X. and Devadas, S., 2013, November. Path ORAM: an extremely simple oblivious RAM protocol. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (pp. 299-310). ACM.
[A.2] Sajin Sasy, Sergey Gorbunov, and Christopher Fletcher. "ZeroTrace: Oblivious memory primitives from Intel SGX." Symposium on Network and Distributed System Security (NDSS). 2018.

Discussion: Differentially Obliviousness

- Chen et al [A.3] formalize DO algorithms

Definition 2.1 (Differentially oblivious (stateless) algorithms). *Let ϵ, δ be functions in a security parameter λ. We say that the stateless algorithm M satisfies (ϵ, δ)-differential obliviousness, iff for any neighboring inputs I and I', for any λ, for any set S of access patterns, it holds that*

$$\Pr[\mathbf{Accesses}^M(\lambda, I) \in S] \leq e^{\epsilon(\lambda)} \cdot \Pr[\mathbf{Accesses}^M(\lambda, I') \in S] + \delta(\lambda)$$

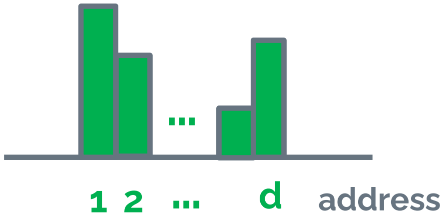
where $\mathbf{Accesses}^M(\lambda, I)$ is a random variable denoting the ordered sequence of memory accesses the algorithm M makes upon receiving the input λ and I.

- NIPS '19 [A.4] and CCS '18 [A.5] proposed similar algorithms to gurantee DO

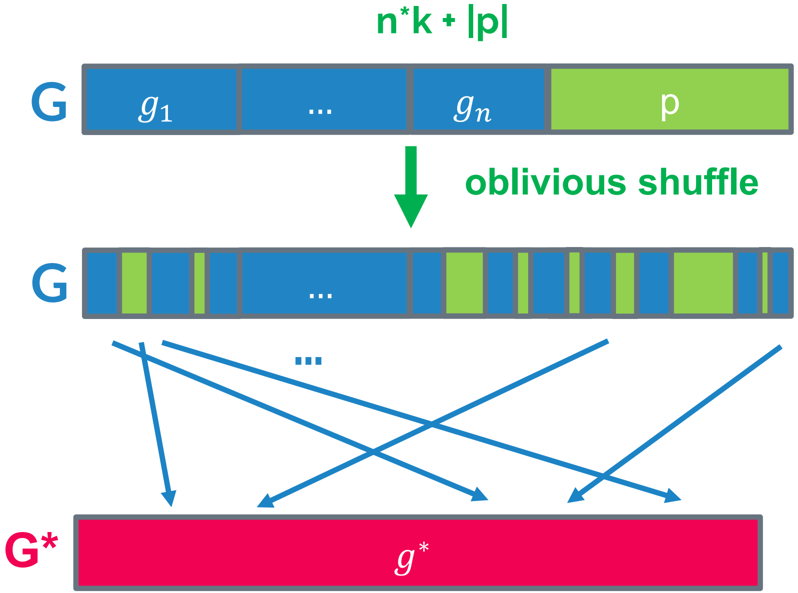
[A.3] Chan et al, Foundations of differentially oblivious algorithms. SIAM 2019.
[A.4] Joshua et al, An Algorithmic Framework For Differentially Private Data Analysis on Trusted Processors. NIPS 2019
[A.5] Mazloom et al. Secure Computation with Differentially Private Access Patterns. CCS 2018

Discussion: Differentially Obliviousness

- NIPS ’19 [A.4]
  - **p** padding, **oblivious shuffle**
  - $O\left((\mathrm{nk} + |\mathrm{p}|)\log^2(\mathrm{nk} + |\mathrm{p}|)\right)$
  - leaks differenitally private histogram of all indices



Access pattern histogram with DP

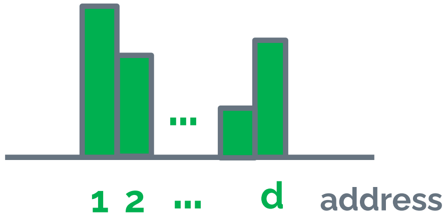


It doesn’t work due to huge padding size
(1) Sensitivity (i.e., d) can be too large (|p| is O(kd))
(2) Can only use one-sided noise

[A.4] Joshua et al, An Algorithmic Framework For Differentially Private Data Analysis on Trusted Processors. NIPS 2019

Discussion: Differentially Obliviousness

- NIPS '19 [A.4]
  - p padding, oblivious shuffle
  - $O\left((\mathrm{nk} + |\mathrm{p}|)\log^2(\mathrm{nk} + |\mathrm{p}|)\right)$
  - leaks diffferenitally private histogram of all indices



Access pattern histogram with DP

p is too large in FL setting

- noise vector **z** $\in \mathbf{R}^{\mathbf{d}}$
  - z_i ~ **Lap(2k/ε)**
  - **d**-dimensional
- Then, **p** is O**(kd)**
  - **kd** is very large
- Remember
  - $O\left((\mathrm{nk} + \mathrm{p})\log^2(\mathrm{nk} + \mathrm{p})\right)$
- Moreover, it is necessary to allocate memory for the padded data, which is very incompatible with SGX that has poor memory

[A.4] Joshua et al, An Algorithmic Framework For Differentially Private Data Analysis on Trusted Processors. NIPS 2019

Algorithms

Algorithm 3 Baseline

Input: $g = g_1 \,\|\, \dots \,\|\, g_n$: concatenated gradients, nk length
Output: g^*: aggregated parameters, d length

initialize aggregated gradients g^*
for each $(idx, val) \in g$ **do**
  /* c is the number of weights included in one cacheline */
  /* offset indicates the position of idx in the cacheline */
  for each $(idx^*, val^*) \in g^*$ if $idx^* \equiv \text{offset} \pmod{c}$ **do**
    $flag \leftarrow idx^* == idx$ ▷ target index or not
    $val' \leftarrow \texttt{o\_mov}(flag, val^*, val^* + val)$
    write val' into idx^* of g^*
return g^*

Algorithm 4 Advanced

Input: $g = g_1 \,\|\, \dots \,\|\, g_n$: concatenated gradients, nk length
Output: g^*: aggregated parameters, d length

/* *initialization*: prepare zero-valued gradients for each index */
$g' \leftarrow \{(1, 0), \dots, (d, 0)\}$ ▷ all `values` are zero
$g \leftarrow g \,\|\, g'$ ▷ concatenation
/* *oblivious sort* in $O((nk + d) \log^2 (nk + d))$ */
oblivious sort g by `index`
/* *oblivious folding* in $O(nk + d)$ */
$idx \leftarrow$ `index` of the first weight of g
$val \leftarrow$ `value` of the first weight of g
for each $(idx', val') \in g$ **do** ▷ Note: start from the second weight of g
  $flag \leftarrow idx' == idx$
  /* M_0 is a dummy index and very large integer */
  $idx_{prior}, val_{prior} \leftarrow \texttt{o\_mov}(flag, (idx, val), (M_0, 0))$
  write $(idx_{prior}, val_{prior})$ into idx' - 1 of g
  $idx, val \leftarrow \texttt{o\_mov}(flag, (idx', val'), (idx, val + val'))$
/* *oblivious sort* in $O((nk + d) \log^2 (nk + d))$ */
oblivious sort g by `index` again
return take the first d `values` as g^*

Oblivious Primitives

- **O_MOV**
  - Using CMOV (x86 instruction)
  - set the value of either “a” or “b” in the register depending on the conditional flag
    - **adversary cannot see**
  - constructs o_mov (oblivious move), o_swap (oblivious swap), o_write (oblivious write)

```
#[inline]
pub fn o_swap<T>(flag: isize, x: &T, y: &T) {
    unsafe {
        llvm_asm!(
            "test %rax, %rax \n\t
             movq (%r8), %r10 \n\t
             movq (%rdx), %r9 \n\t
             mov %r9, %r11 \n\t
             cmovnz %r10, %r9 \n\t
             cmovnz %r11, %r10 \n\t
             movq %r9, (%rdx) \n\t
             movq %r10, (%r8) \n\t"
            :
            : "{rax}"(flag), "{rdx}" (x), "{r8}" (y)
            : "rax", "rdx", "r8", "r9", "r10", "r11"
            : "volatile"
        );
    }
}
```

o_swap by x86

Oblivious Primitives

- O_WRITE
  - n times oblivious mov (o_mov)
  - x16 faster by cache-line optimization in Baseline method

```
for i in 0..(num_of_weights / CACHE_LINE_NUM_OF_WEIGHT) {
    // Only one of the values to be read needs to be written.
    // Since it is not known from the outside which value was written
    let cache_line_addr = dst_base.offset(CACHE_LINE_NUM_OF_WEIGHT * i + ith) as *mut f32;
    let flag = (cache_line_addr == addr) as isize;

    let ret: f32;
    llvm_asm!(
        "xor %rcx, %rcx \n\t
        mov %r8, %rcx \n\t
        test %rcx, %rcx \n\t
        cmovnz %rdx, %rax \n\t"
      : "={rax}"(ret)
      : "{r8}"(flag), "{rax}" (*cache_line_addr), "{rdx}" (val)
      : "rax", "rcx", "rdx", "r8"
      : "volatile"
    );
    *cache_line_addr = ret;
    last += 1;
}
```

o_mov