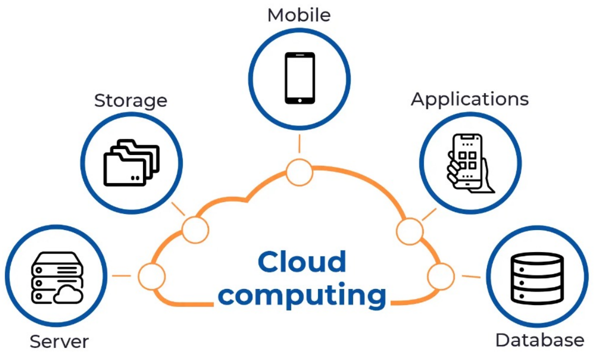
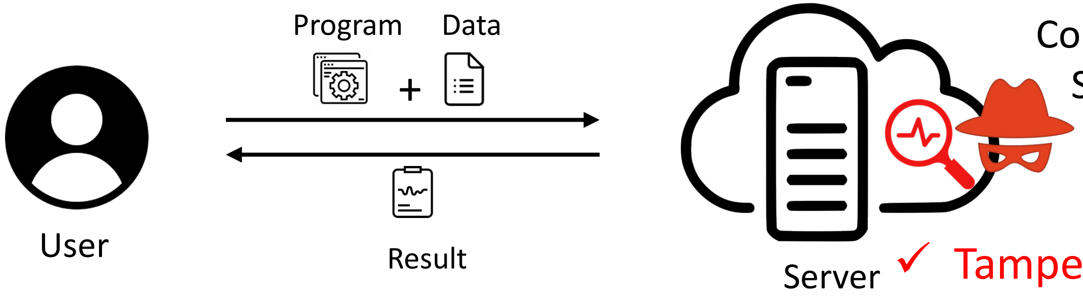
Making Your Program Oblivious: a Comparative Study for Side-channel-safe Confidential Computing

AKM Mubashwir Alam, Keke Chen
Trustworthy and Intelligent Computing Lab (TIC)
Department of Computer Science
Marquette University

Cloud Computing and Security Concerns



Security concerns:



Compromise Security?

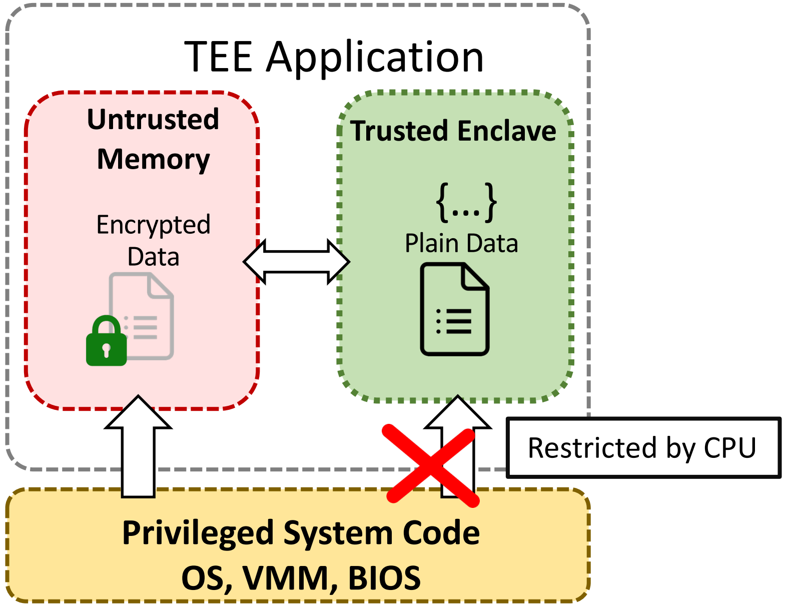
- ✓ Tamper program
- ✓ Tamper data
- ✓ Steal sensitive information

Goal

- **Confidentiality**: Server learns nothing
- **Integrity**: Server returns accurate result
- **Efficiency**: Faster execution time

Confidential Computing with TEE

- Hardware assisted approach
- Provides:
  - Confidentiality
  - Integrity
  - Efficiency
- Faster Computation than existing crypto approaches
  - Homomorphic Encryption
  - Secure Multiparty Computation
  - Other hybrid crypto-protocols



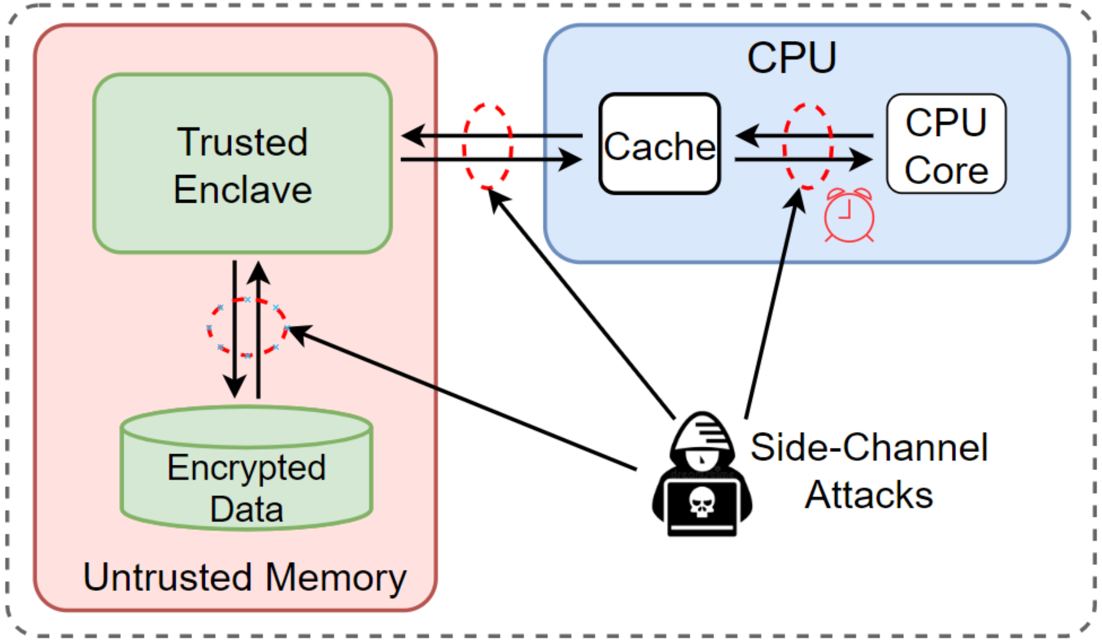
TEE Security

Side-Channel Attacks

Adversary cannot access Enclave’s restricted memory

However, adversary may observe:

- ✓ Untrusted memory interactions
- ✓ Enclave Page loading
- ✓ CPU Cache access time



Researchers discovered series of attacks by exploiting these side-channels

Side Channel Attacks on TEEs

- Memory-targeted attacks
  - Page Fault Attack [1] [2]
  - Data/Page Access Pattern Attacks [3]
  - Branch Shadowing Attack [4]
- Cache Attacks
  - Cache Attack [5]
- Micro-architectural Attacks
  - Speculative Execution Attack [6]
  - Injection based Attack [7]

[1] - Y. Xu, W. Cui, and M. Peinado, Controlled-channel attacks: Deterministic side channels for untrusted operating systems, 2015

[2] - S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena. Preventing page faults from telling your secrets, 2016.

[3] - J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx, "Telling Your Secrets Without Page Faults" Proc. 26th USENIX Conf. Secur. Symp.

[4] - S. Lee, M. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, Inferring fine-grained control flow inside SGX enclaves with branch shadowing, 2017.

[5] - F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, Software grand exposure: SGX cache attacks are practical, 2017

[6] - J. Van Bulck, M. Minkin, O. Weisse et al., "FORESHADOW: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution," Proc. 27th USENIX Secur. Symp., 2018

[7] - Jo Van Bulck, Daniel Moghimi, Michael Schwarz, et al, "LVI: Hijacking transient execution through microarchitectural load value injection. In: 2020 IEEE Symposium on Security and Privacy (SP)

Solution for Side Channel Attacks

- Manufacturers’ guideline for developers
- Micro-architectural patches
- Access pattern protection – data obliviousness helps!

Data oblivious solutions for Side-Channel Protection

- Execution path doesn’t change for different inputs
- Data Access either fixed or randomized pattern

Goal

- To protect any data dependent operation
- To achieve either fixed or randomized access pattern

Data dependent code

```
if (a >= b ) {
    larger = a
} else {
    larger = b
}
```

Oblivious code

```
bool cond = (a >=b)?
CMOV(cond, larger, a)
CMOV(!cond, larger, b)
//Access both memory locations
//copy only if the condition is true
```

TEE Side-channel Attacks that Data Obliviousness can address

- Memory Targeted Attacks
- Cache Attacks
- Micro-Architectural Attacks

Challenges of Implementing Oblivious Programs

- Unclear how complex is to develop oblivious program
    - Developers' effort is unclear in manual composition
    - Automated/semi-automated approaches are still immature
    - Quality of generated oblivious programs
- No systematic Study

Our Contribution

- Comprehensive analysis on constructing data oblivious solutions
  - Manual
  - Compiler
  - Circuit
  - Framework

- Characterize the approaches
  - Performance
  - Ease of use
  - Maturity for applications

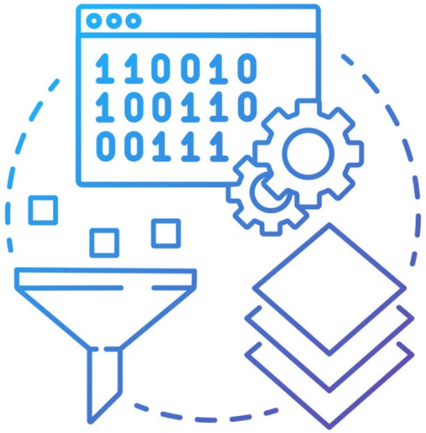
- Develop evaluation benchmark on
  - Oblivious operations
  - Computation intensive tasks
  - Data Intensive tasks

Making Your Program Oblivious: Manual Approach

- To detect and apply manually
- Require depth knowledge on
  - Access pattern problem
  - Oblivious Algorithms, Data Structure
  - Oblivious primitives in TEE
- Require analyzing for vulnerability
  - High-level design of the program
  - Every line of code
- Require mitigation for
  - High-level interaction
  - Detail level code

Making Your Program Oblivious: Fully Automated Approach (Compiler)

- Minimize the manual effort
- Accelerate development process



Compiler

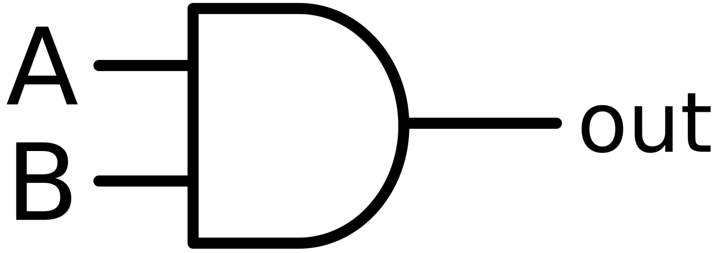
Automate manual composition

Hide pattern via randomization

- Still Experimental
- Often does not provide efficient solution
  - Unnecessary obfuscation due to false positive result in Static Analysis
  - Memory randomization and shuffling incurs a significantly high cost

Making Your Program Oblivious: Fully Automated Approach (Circuit)

- Boolean circuits
  - Used in crypto for years, e.g., garbled circuit
  - Naturally data independent (Oblivious)
  - Executes all the paths



- Concerns
  - Generated circuit is large, proportional to input data size
  - Simulating (hardware) circuits in software mode
  - Inherently slow

Making Your Program Oblivious: Semi-automated (framework) Approach

- Regulate the application’s data flow
- Handle most sensitive access pattern shared by applications
- Significantly reduce developers’ effort

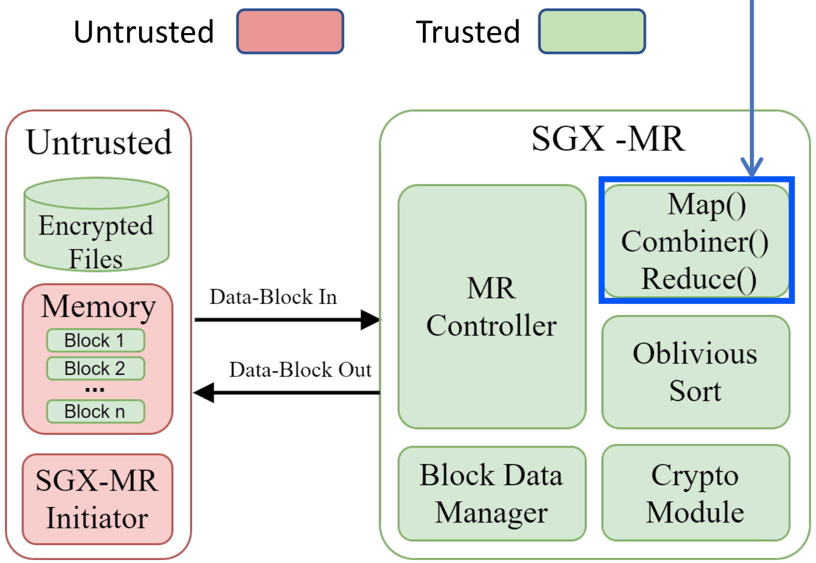
Making Your Program Oblivious: Semi-automated (framework) Approach

- Regulate the application's data flow
- Handle most sensitive access pattern shared by applications
- Significantly reduce developers' effort

E.g., SGX-MR

- Handles oblivious branching, sorting, group-size, etc.
- Developer only provides map and reduce function
- Covers a wide range of data-analytics applications

Developer only handles access patterns here



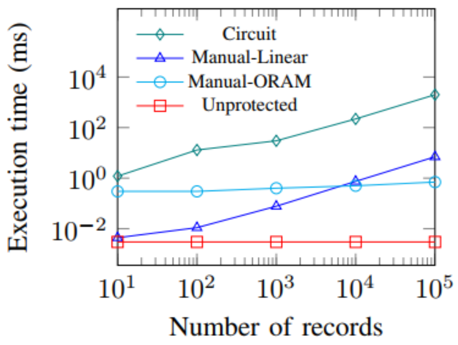
SGX-MR Framework

Experimental Evaluations

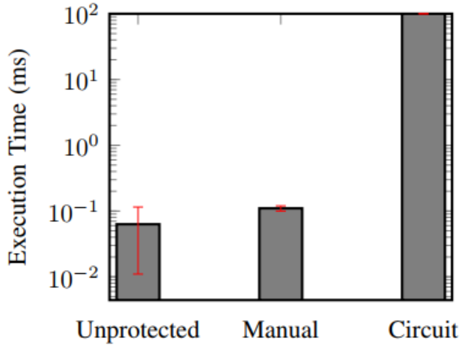
- **System Configuration**
  - Intel(R) Core(TM) i7-8700K CPU
  - 3.70GHz processor
  - 16 GB of DRAM.
  - Intel SGX
  - Linux version is Ubuntu 22.04
- **Implementation**
  - Manual Approach – State of the art oblivious techniques
  - Circuit – HyCC Circuit generator [1]
  - Framework – SGX-MR [2]
- **Oblivious Operation**
  - Oblivious array access, conditional branching, oblivious sorting
- **Sample Application**
  - Compute Intensive – Edit Distance, All-pair shortest path(Floyd Warshal)
  - Data Intensive – WordCount, KMeans

[1] Büscher, Niklas, et al. "HyCC: Compilation of hybrid protocols for practical secure computation." *ACM SIGSAC Conference on Computer and Communications Security*. 2018.
[2] A. K. M. M. Alam, et al. SGX-MR: Regulating dataflows for protecting access patterns of data-intensive SGX applications. Proceedings on PETS, 2021(1):5 – 20, 01 Jan. 2021

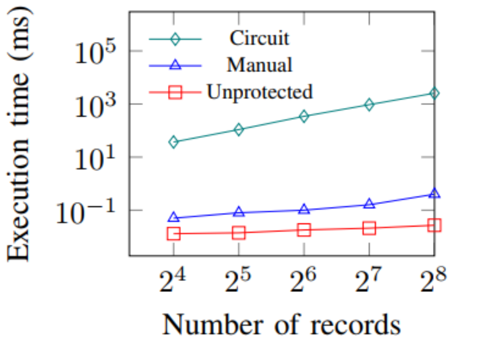
Compute-Intensive: Building Blocks



(a) Oblivious array access. Record size 8 bytes.



(b) Oblivious branching.

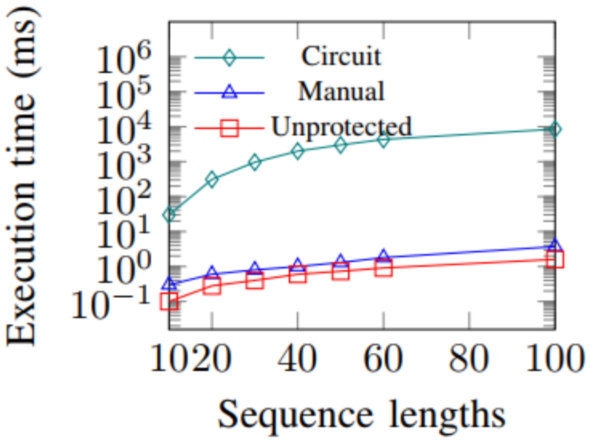


(c) Oblivious sorting. Record size 8 bytes.

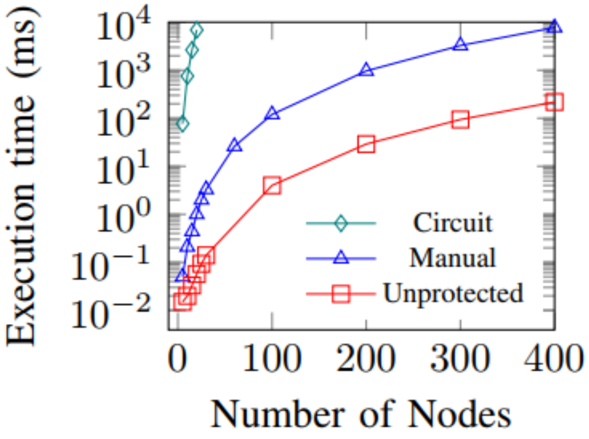
- Oblivious solution is costly compared to unprotected versions
- Linear scan performs better than ORAM for < 10,000 records
- Manual approach is much efficient compared to circuit-based approach

Experimental Results

Compute-Intensive: Applications



(a) Edit Distance

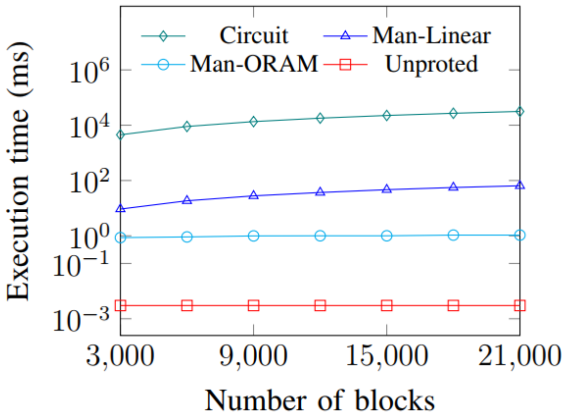


(b) All-pair shortest path

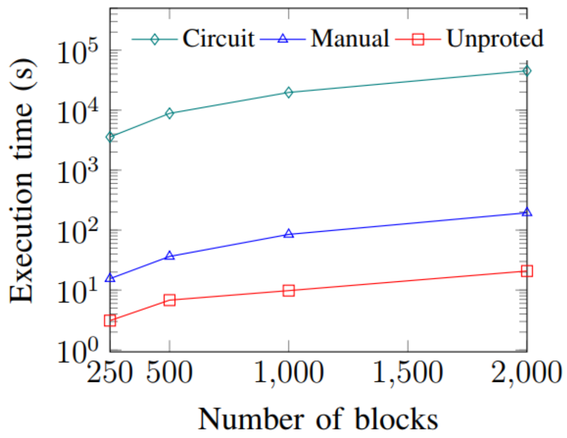
- Manual approach is effective: closer to unprotected version
- Circuit cost is extremely high compared to manual approach

Experimental Results

Data-Intensive: Building Blocks



(a) Comparing random access over blocks. Block size 1 KB.

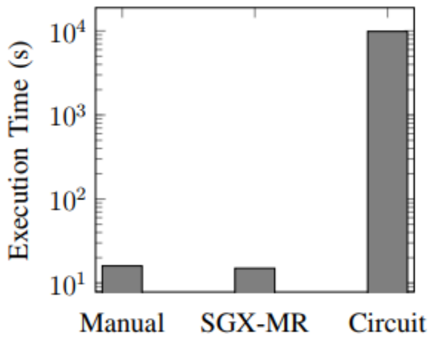


(b) Block-based oblivious sorting. Block size 1KB with 75 words per block.

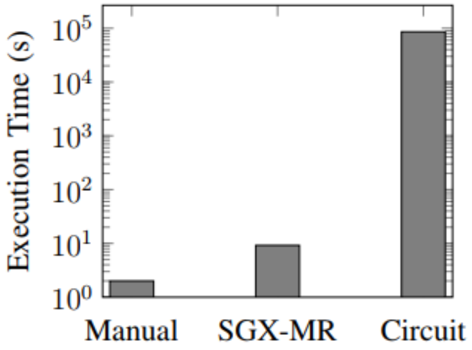
- ORAM performs much better than linear scan
- Circuit approach is still expensive

Experimental Results

Data-Intensive: Applications



(a) Application-level perfromance for Wordcount. Number of blocks 500 with 75 words/block.

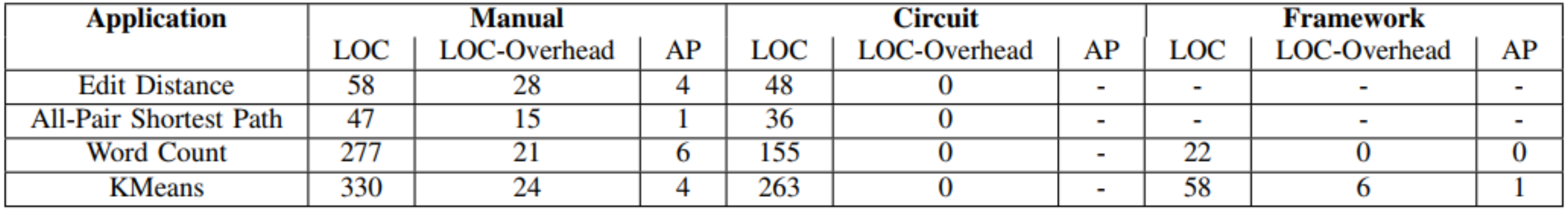


(b) Application-level performance for KMeans. 4000 1KB-Blocks with eight bytes per record, and five clusters.

- Framework approach is effective
- With minimal effort close to manually crafted solution

Developers’ Effort

LOC: Total lines of code
AP: Access-pattern sensitive code segments
LOC-overhead: Lines used to hide access-patterns



Application	Manual			Circuit			Framework		
	LOC	LOC-Overhead	AP	LOC	LOC-Overhead	AP	LOC	LOC-Overhead	AP
Edit Distance	58	28	4	48	0	-	-	-	-
All-Pair Shortest Path	47	15	1	36	0	-	-	-	-
Word Count	277	21	6	155	0	-	22	0	0
KMeans	330	24	4	263	0	-	58	6	1

- Circuit approach require no additional overhead for access pattern protection.
- Manual approach require high effort to analyze sensitive code and write mitigations
- Frameworks, e.g., SGX-MR, protect major access-pattern issues, require minimal effort

Conclusion

- Manual approach is difficult to handle
- Fully automated approaches are not yet ready
- Framework approach is effective and more practical

Thank You

Making Your Program Oblivious

a Comparative Study for Side-channel-safe Confidential Computing