Verifying Computation in Sequestered Encryption

Meron Z. Demissie, Lauren Biernacki, Todd Austin
University of Michigan
Correspondence: mdemissi@umich.edu

Introduction
- Sequestered Encryption (SE) is a hardware technology that supports secure computation on private data.
  - Unlike trusted execution environments (TEEs), SE computation is not visible to software.
- SE uses two key mechanisms to achieve its goals:
  - Encrypted computation: hides the content of private data from software.
  - Data oblivious programming: eliminates data-dependent control flow and memory accesses.

Challenges
SE does not guarantee correct computation.
- Server can intentionally rearrange secret computation.
- Hardware errors may occur during computation.

Contribution
It is important to think about how a client can check if a computation is done correctly. Since undesired changes to computation cannot be readily detected, we extend SE to verify the correctness of computation.

Summary
- Integrity is an important security property not addressed by SE.
- This work extends SE to verify computation by adding metadata to encrypted types.
- Breaking this mechanism requires a preimage attack on a cryptographic hash.

Usage Model

Introduction
Sequestered Encryption (SE) is a hardware technology that supports secure computation on private data. Unlike trusted execution environments (TEEs), SE computation is not visible to software.

SE uses two key mechanisms to achieve its goals:
- Encrypted computation: hides the content of private data from software.
- Data oblivious programming: eliminates data-dependent control flow and memory accesses.

Challenges
SE does not guarantee correct computation.
- Server can intentionally rearrange secret computation.
- Hardware errors may occur during computation.

Contribution
It is important to think about how a client can check if a computation is done correctly. Since undesired changes to computation cannot be readily detected, we extend SE to verify the correctness of computation.

Summary
- Integrity is an important security property not addressed by SE.
- This work extends SE to verify computation by adding metadata to encrypted types.
- Breaking this mechanism requires a preimage attack on a cryptographic hash.

Conclusion and Future Work
- We utilize the unchanging data flow graph of programs, which stems from data oblivious programming used in SE.
- Once the data flow graph of a program is generated, a hash associated with the data can be passed through the graph using a combining function.
- This ongoing work seeks to find ways of simplifying the cost of client-side verification.

Combining Function (F)
- It captures any changes in register/memory communication between instructions.
- It captures any changes in individual instructions.
- Given an instruction with inputs, it must be difficult to map the hashes of the input to hash of the instruction’s output to ensure strong integrity.

Conclusion and Future Work
- We utilize the unchanging data flow graph of programs, which stems from data oblivious programming used in SE.
- Once the data flow graph of a program is generated, a hash associated with the data can be passed through the graph using a combining function.
- This ongoing work seeks to find ways of simplifying the cost of client-side verification.