MGX: Near-Zero Overhead Memory Protection for Data-Intensive Accelerators

*Weizhe Hua*¹, Muhammad Umar¹, Zhiru Zhang¹, G. Edward Suh¹,²
Cornell University¹; Meta AI²

Session 8B: Security IV
Tuesday, June 21
Demands for High-Performance & Secure Computation

- Processing sensitive and private data requires **high performance** and strong **security guarantees**

(a) Self-driving cars  
Credit: The New York Times

(b) Cancer diagnosis  
[Lu Nature’21]

(c) Genomic alignment  
Credit: Wiki
Hardware-based protection (e.g., Intel SGX) offers higher performance than crypto-based solutions

**Specialized** TEEs are emerging for various hardware accelerators

Memory protection in accel. TEE remains a major source of overhead
MGX: Memory Guard for Xelerators

▶ Observations
  – Memory access *patterns* and *granularity* can be largely determined for a particular application
  – *Customized memory protection* leveraging accelerator/application specific characteristics can significantly reduce protection overhead

▶ Key results
  – Reduce the avg. performance overhead of memory protection from 28% to 4% for DNNs and 33% to 5% for graph algorithms
  – Lower the worst-case overhead from 45% to 6%
  – Applicable to other applications such as H.264 decoding and genomic alignment
For confidentiality, must encrypt data blocks in memory
- Randomization required to prevent comparing two blocks

For integrity, must check if a memory read returns the most recent value (that is stored at the address by an accelerator)
Memory Encryption: Counter Mode

Counter

Physical address

Version Number (VN)

$B1$

$B2$

$B3$

$B4$

(Addr1, VN1)

AES_k

One-Time-Pad (OTP)

CPU

VN1

Memory

E$B1$
Memory Encryption: Counter Mode

Memory

\[ \text{Counter} \]

\[ \text{VN2} \quad \text{VN1} \quad \text{E$B_2$} \quad \text{E$B_1$} \]

\[ \text{CPU} \]

\[ \text{AES}_k \]

\[ \text{One-Time-Pad (OTP)} \]

\[ (\text{Addr}_2, \text{VN}_2) \]

\[ \text{$B_2$} \quad \text{$B_3$} \quad \text{$B_4$} \quad \ldots \]
Memory Encryption: Counter Mode

Counter

$B_3$
$B_4$
...

(Addr3, VN3) \rightarrow AES_k

One-Time-Pad (OTP)

CPU

Memory

VN3 VN2 VN1

E$B_3$ E$B_2$ E$B_1$
Memory Encryption: Counter Mode

Counter

$B4$
...
...
...

(Addr4, VN4) → $AES_k$

One-Time-Pad (OTP)

CPU

Root of Merkle tree

Hash of VNs

VN1, VN2, VN3, VN4

Memory

E$B4$, E$B3$, E$B2$, E$B1$
Integrity Verification

$B4$
...
...(Addr4, VN4)

$AES_K$

One-Time-Pad (OTP)

$MAC_K$

Counter

$MAC_1$ $MAC_2$ $MAC_3$ $MAC_4$

Root of Merkle tree

Hash of VNs

VN1 VN2 VN3 VN4

CPU

Memory
Overhead of Memory Protection

- Store per cache-line metadata (VNs and MACs) in the off-chip memory
- Verify the VNs recursively until reaching the on-chip root of the Merkle tree

The avg. memory traffic increase for DNN inference is **36.1%**
Overhead of Memory Protection

- Store per cache-line metadata (VNs and MACs) in the off-chip memory
- Verify the VNs recursively until reaching the on-chip root of the Merkle tree

The avg. memory traffic increase for DNN training is 40.4%
Overhead of Memory Protection

- Store per cache-line metadata (VNs and MACs) in the off-chip memory
- Verify the VNs recursively until reaching the on-chip root of the Merkle tree

The avg. memory traffic increase for PageRank (PR) is **26.3%**
Overhead of Memory Protection

- Store per cache-line metadata (VNs and MACs) in the off-chip memory
- Verify the VNs recursively until reaching the on-chip root of the Merkle tree

The avg. memory traffic increase for Breadth-first search (BFS) is \textbf{25.6\%}
Data movements are often explicitly scheduled and known for a given task
- *Generate VNs* for memory protection *without using off-chip memory*

The memory access pattern is *largely determined* by the network architecture (i.e., control dataflow graph)

Data accesses happen at a coarser granularity
MGX: High-level Idea

- Data movements are often explicitly scheduled and known for a given task
  - Generate VNs for memory protection without using off-chip memory

- Data accesses happen at a coarser granularity
  - Can have each MAC protect a larger memory chunk (e.g., entire feature map)

The memory access pattern is largely determined by the network architecture (i.e., control dataflow graph)

ResNet [He CVPR'16]
MGX Scheme: VN Generation

![Diagram showing the scheme]

$B4 \rightarrow (\text{Addr, } ?) \rightarrow \text{AES}_K \rightarrow \text{Accelerator}

Hash of VNs

- VN1
- VN2
- VN3
- VN4

Memory

- E$B4$
- E$B3$
- E$B2$
- E$B1$
MGX Scheme: VN Generation

![Diagram of MGX Scheme: VN Generation]

- **$B_4**
- **$B_3**
- **$B_2**
- **$B_1**
- (Addr, VN)
- **AES_k**
- Control Proc.
- Accel. state
- Accelerator
- Hash of VNs
  - VN1
  - VN2
  - VN3
  - VN4
- Memory
  - E$B_4$
  - E$B_3$
  - E$B_2$
  - E$B_1$
MGX Scheme: VN Generation

$(\text{Addr, VN}) \xrightarrow{\text{AES}_k} \text{AES}_k\xrightarrow{\text{control proc.}} \text{control proc.}\xrightarrow{\text{accel. state}} \text{accel. state}$

Hash of VNs \rightarrow Memory

E$B_4$, E$B_3$, E$B_2$, E$B_1$
MGX Scheme: VN Generation

$B_4 \rightarrow (\text{Addr, VN}) \rightarrow \text{AES}_K \rightarrow \text{Control Proc.} \rightarrow \text{Accel. state} \rightarrow \text{Accelerator} \rightarrow \text{Memory} \rightarrow E\$B_4, E\$B_3, E\$B_2, E\$B_1
MGX Scheme: Coarse-grained MAC

$B_4 \rightarrow (\text{Addr, VN}) \rightarrow \text{AES}_K \rightarrow \text{MAC}_K' \rightarrow \text{MAC}_1, \text{MAC}_2, \text{MAC}_3, \text{MAC}_4$

\begin{align*}
\text{Control Proc.} & \quad \text{Accel. state}
\end{align*}
MGX Scheme: Coarse-grained MAC

$B_4$  
$B_3$  
$B_2$  
$B_1$  

(Addr, VN)  

Control Proc.  

AES$_K$  

MAC$_K'$  

Accel. state  

Accelerator  

Memory
MGX Scheme: Coarse-grained MAC

![Diagram of MGX Scheme]

- **$B_4$$B_3$$B_2$$B_1**
- **(Addr, VN)**
- **Control Proc.**
- **AES_k**
- **MAC_k’**
- **MAC1**

Accelerator

Memory
MGX for DNN

Accelerator state

Layer ID

VN for read:
Layer ID

Accelerator

Memory

VN for write:
Layer ID + 1
MGX for DNN

Layer ID: 1

VN for read: 1

Accelerator state

VN for write: 2

Accelerator

Memory
MGX for DNN

Accelerator state
Layer ID: 2

VN for read: 2

Accelerator

Memory

VN for write: 3
DNN with Tiling

- Compute a partition of output at a time due to limited size of on-chip memory
  1. Compute over all spatial locations

Input feature maps

![Diagram of convolution](image)

CONV

Output feature maps
DNN with Tiling

- Compute a partition of output at a time due to limited size of on-chip memory
  1. Compute over all spatial locations

![Diagram of DNN with Tiling]

- Input feature maps
- Convolution (CONV)
- Output feature maps
DNN with Tiling

- Compute a partition of output at a time due to limited size of on-chip memory
  1. Compute over all spatial locations
  2. Compute over the rest of input channels (partial results are updated)

Input feature maps -> CONV -> Output feature maps
MGX for DNN with Tiling

Accelerator state
- Layer ID
- Input tile ID
- Output tile ID
- Write mem. CTR

Layer ID || Input tile ID

Layer ID+1 || Write mem. CTR || Output tile ID

Accelerator state:
- Layer ID
- Input tile ID
- Output tile ID
- Write mem. CTR

VN for reading an input tile:
Layer ID || Input tile ID

VN for writing an output tile:
Layer ID+1 || Write mem. CTR || Output tile ID
MGX for DNN with Tiling

Accelerator state

Layer ID: 0
Input tile ID: 0
Output tile ID: 0
Write mem. CTR: 0

Accelerator

Memory

VN: 0 || 0
VN: 1 || 0 || 0

Layer 0

X0

X1
MGX for DNN with Tiling

Accelerator state
Layer ID: 0
Input tile ID: 1
Output tile ID: 1
Write mem. CTR: 0

Layer 0
X0

Layer 1
X1

Accelerator
Memory

VN: 0 || 0
VN: 0 || 1
VN: 1 || 0 || 0
VN: 1 || 0 || 1
MGX for DNN with Tiling

Accelerator state

Layer ID: 0
Input tile ID: 1
Output tile ID: 1
Write mem. CTR: 0

VN: 0 || 0

Layer 0

VN: 0 || 1

Accelerator

Memory

VN: 1 || 0 || 0

VN: 1 || 0 || 1
MGX for DNN with Tiling

Accelerator state

- Layer ID: 0
- Input tile ID: 4
- Output tile ID: 0
- Write mem. CTR: 0
MGX for DNN with Tiling

Accelerator state

Layer ID: 0
Input tile ID: 4
Output tile ID: 0
Write mem. CTR: 1
Graph Algorithm: PageRank

- Each vertex is associated with a feature (i.e., rank)

Graph topology

```
A → B → C → D
```

Adjacency matrix

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Rank

```
A = 1.2
B = 1.2
C = 1.2
D = 1.2
```

Updated rank

```
D = 1.2
C = 1.2
B = 1.2
A = 1.2
```
Irregular access to the adjacency matrix
- Read-only adjacency matrix only needs a constant VN

Updated rank from the previous iteration \((i)\) is used as rank in the next iteration \((i+1)\)
- Keep iteration in accelerator state to construct VN
Experimental Setup

- **Performance simulation**
  - **Accelerator**
    - DNNs: SCALE-Sim [Samajdar ISPASS’20]
    - Graph algorithms: RTL simulation of GraphLily [Hu ICCAD’21]
  - **Memory simulation**
    - Cycle-level memory protection simulator + Ramulator [Kim CAL’15]

- **Protection Baseline** (Intel MEE whitepaper)
  - 8-ary Merkle tree with 56-bit VNs and MACs, and works at a 64-byte granularity
  - 32-KB on-chip cache for VNs and MACs

- **Benchmarks**
  - A variety of DNNs for vision and language tasks
  - Breadth-first search (BFS) and PageRank (PR) on real-world large-scale graphs
Experimental Results on ML Benchmarks

The avg. and max. overhead of BP for DNN inference are 24% and 40%
Experimental Results on ML Benchmarks

The avg. and max. overhead of $\text{MGX}_\text{VN}$ for DNN inference are $6\%$ and $8\%$. 

![Bar chart showing normalized execution time for different models with MGX VN and BP compared. VGG, AlexNet, GoogleNet, ResNet, BERT, DLRM. The avg. and max. overhead of MGX VN for DNN inference are 6% and 8%.]
Experimental Results on ML Benchmarks

The avg. and max. overhead of MGX for DNN inference are 3% and 4%
More in the paper
- Detailed threat model and security analysis
- Detailed MGX scheme and experimental results for DNN and graph processing accelerators
- MGX for other applications such as H.264 decoding and genomic alignment

Conclusion
- MGX provides near-zero overhead memory protection for a wide range of data-intensive accelerators
- TEE on Accelerator with MGX is a promising solution to high-performance and secure computation on private data
Thank You!

MGX: Near-Zero Overhead Memory Protection for Data-Intensive Accelerators

Weizhe Hua¹, Muhammad Umar¹, Zhiru Zhang¹, G. Edward Suh¹,²
Cornell University¹; Meta AI²

Session 8B: Security IV
Tuesday, June 21