SCARF – A Low-Latency Block Cipher for Secure Cache Randomization

Federico Canale\textsuperscript{1}, Tim Güneysu\textsuperscript{1}, Gregor Leander\textsuperscript{1}, Jan Philipp Thoma\textsuperscript{1}, Yosuke Todo\textsuperscript{2}, Rei Ueno\textsuperscript{3}

1 – Ruhr-University Bochum, Bochum, Germany
2 – NTT Social Informatics Laboratories, Tokyo, Japan
3 – Tohoku University, Sendai-shi, Japan

Jan Philipp Thoma, M. Sc.
Motivation

Caches

- Caches are **set-associative**
  - Table structure with **ways** and **sets**
  - **Set** is determined by part of the address
  - **Way** is determined by the replacement policy
**Motivation**

**Prime + Probe Attack**

- An attacker can observe cache accesses
  1. Fill a cache set
  2. Trigger victim access
  3. Re-Access **eviction set**
     - Cache miss = access

- **Prime + Probe**
Motivation
Cache Randomization

• Cache randomization
  • Prevents efficient Prime + Probe attacks
  • Index is pseudorandomly generated from the address
  • Data is placed in one of the candidate entries
• What do we use as $F_K$?
SCARF

Secure Cache Randomization Function
Design Rationale

Functional Requirements

1. Low Latency
2. Key Dependency
3. Invertibility (given the tag)
   - For write-back caches
4. We will focus on caches with 1024 sets
   - Map 48-bit tag + 10-bit index to 10-bit randomized index
   - Offset bits must be ignored!
Design Rationale

Attacker Model

- The attacker aims to find colliding addresses.
- The attacker can observe if two addresses collide.
- The attacker never sees the output.
How do we design $F_K$?

- Idea 1: Use a low-latency block cipher! (e.g. PRINCE)
  - We need to cut the offset bits
    - Zero-pad 58-bit input
    - Sample index-bits from ciphertext
    - Use remainder as tag

→ 6 Bit storage overhead for the tag and comparison logic
Design Rationale

Function Selection

How do we design $F_K$?

• Idea 2: Create a 58 bit block cipher!
  • The attacker cannot see the output!
    → Can we reduce the latency further?

→ Attacker model now involves partial ciphertext collisions

→ Wired attacker model makes latency optimization hard
**SCARF**

**Function Selection**

How do we design $F_K$?

- SCARF: Create a 10-bit tweakable block cipher!
  - 1024 sets is a common choice
  - Simple attacker model allows latency optimization!

- The attacker learns if

  $E_{T_1}(P_1) \overset{?}{=} E_{T_2}(P_2)$

  $E_{T_2}^{-1}(E_{T_1}(P_1)) \overset{?}{=} P_2$

We can reduce the number of rounds by half!
SCARF
Function Selection

How do we design $F_K$?

- SCARF: Create a 10-bit tweakable block cipher!
  - 1024 sets is a common choice
  - Simple attacker model allows latency optimization!

- The attacker learns if
  \[ E_{T_1}(P_1) \overset{?}{=} E_{T_2}(P_2) \]
  \[ E_{T_2}^{-1}(E_{T_1}(P_1)) \overset{?}{=} P_2 \]

We can reduce the number of rounds by half!
SCARF

Design

- **SCARF** is a 10-bit tweakable block cipher with 48-bit tweak
- 7 + 1 rounds
  - Latency optimized combination of SPN and Feistel structure
- 240 Bit key
**SCARF**

**Design**

- **SCARF** is a 10-bit tweakable block cipher with 48-bit tweak.
- 7 + 1 rounds
  - Latency optimized combination of SPN and Feistel structure
- 240 Bit key
Hardware Implementation

Nangate OCLs

- **Half the latency** compared to low-latency block ciphers
- **Half the area** compared to low-latency block ciphers
- **No additional overhead** for larger tags
- Evaluated software performance using PARSEC benchmarks, results in the paper

<table>
<thead>
<tr>
<th>Technology</th>
<th>45 nm Latency [ns]</th>
<th>45 nm Area [GE]</th>
<th>15 nm Latency [ps]</th>
<th>15 nm Area [GE]</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRINCE</td>
<td>4.74</td>
<td>12,554</td>
<td>628.49</td>
<td>17,484</td>
</tr>
<tr>
<td>MANTIS6</td>
<td>4.73</td>
<td>13,129</td>
<td>630.07</td>
<td>17,641</td>
</tr>
<tr>
<td>QARMA5</td>
<td>4.40</td>
<td>13,915</td>
<td>563.62</td>
<td>18,455</td>
</tr>
<tr>
<td><strong>SCARF</strong></td>
<td><strong>2.26</strong></td>
<td><strong>7,335</strong></td>
<td><strong>305.76</strong></td>
<td><strong>8,118</strong></td>
</tr>
</tbody>
</table>