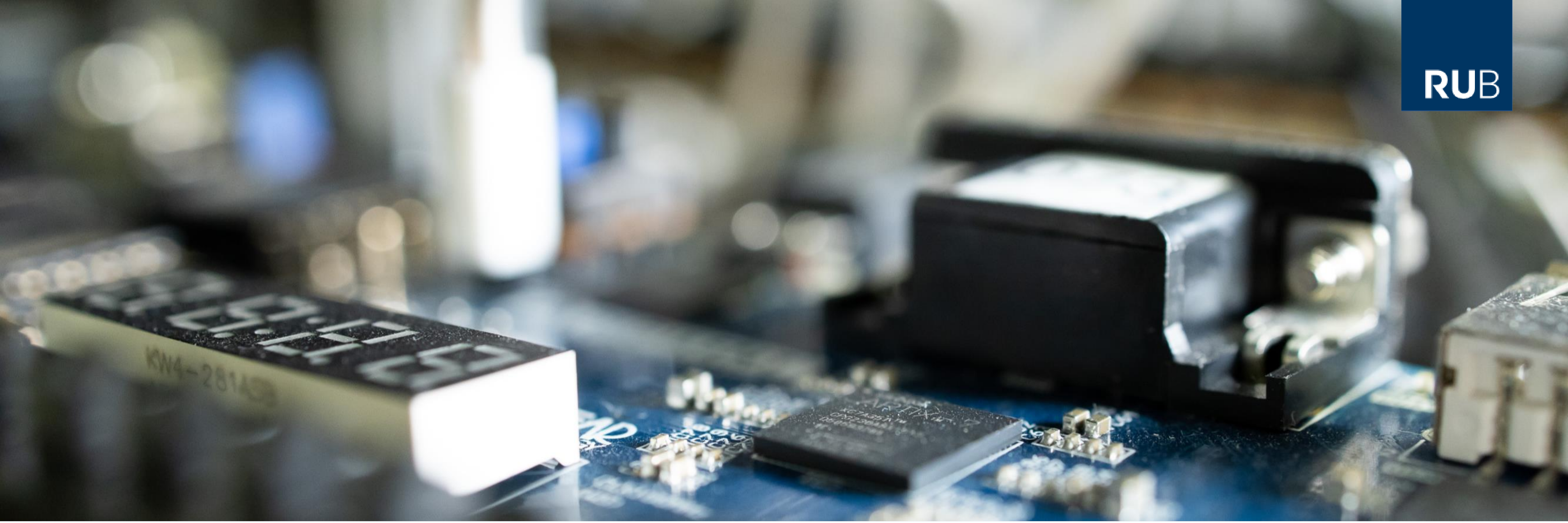


RUHR-UNIVERSITÄT BOCHUM

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

Federico Canale[1], Tim Güneysu[1], Gregor Leander[1], Jan Philipp Thoma[1], Yosuke Todo[2], Rei Ueno[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.

TOHOKU UNIVERSITY

NTT

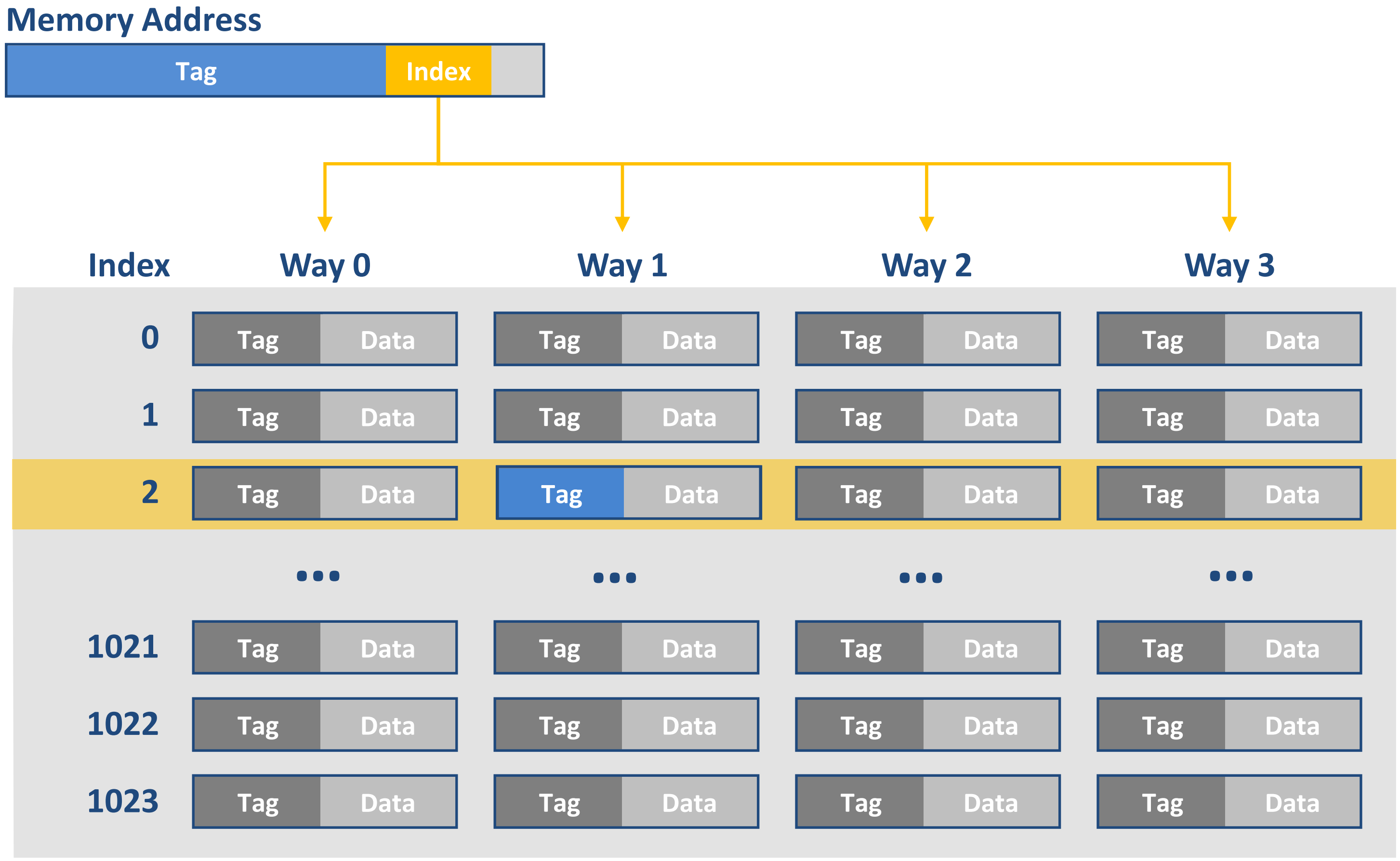
RUHR UNIVERSITÄT BOCHUM RUB

CASA CYBER SECURITY IN THE AGE OF LARGE-SCALE ADVERSARIES

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

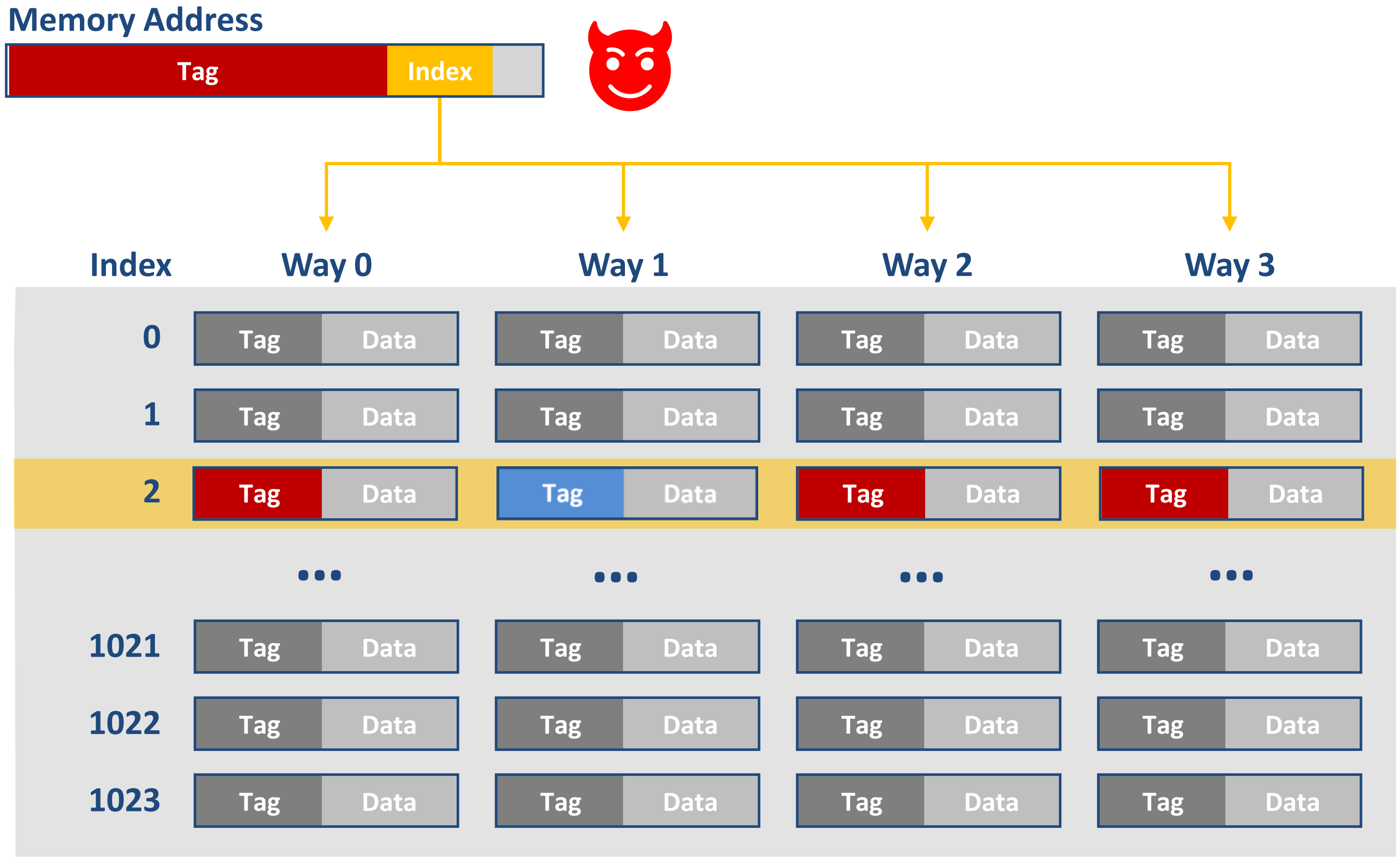


Motivation

Prime + Probe Attack

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

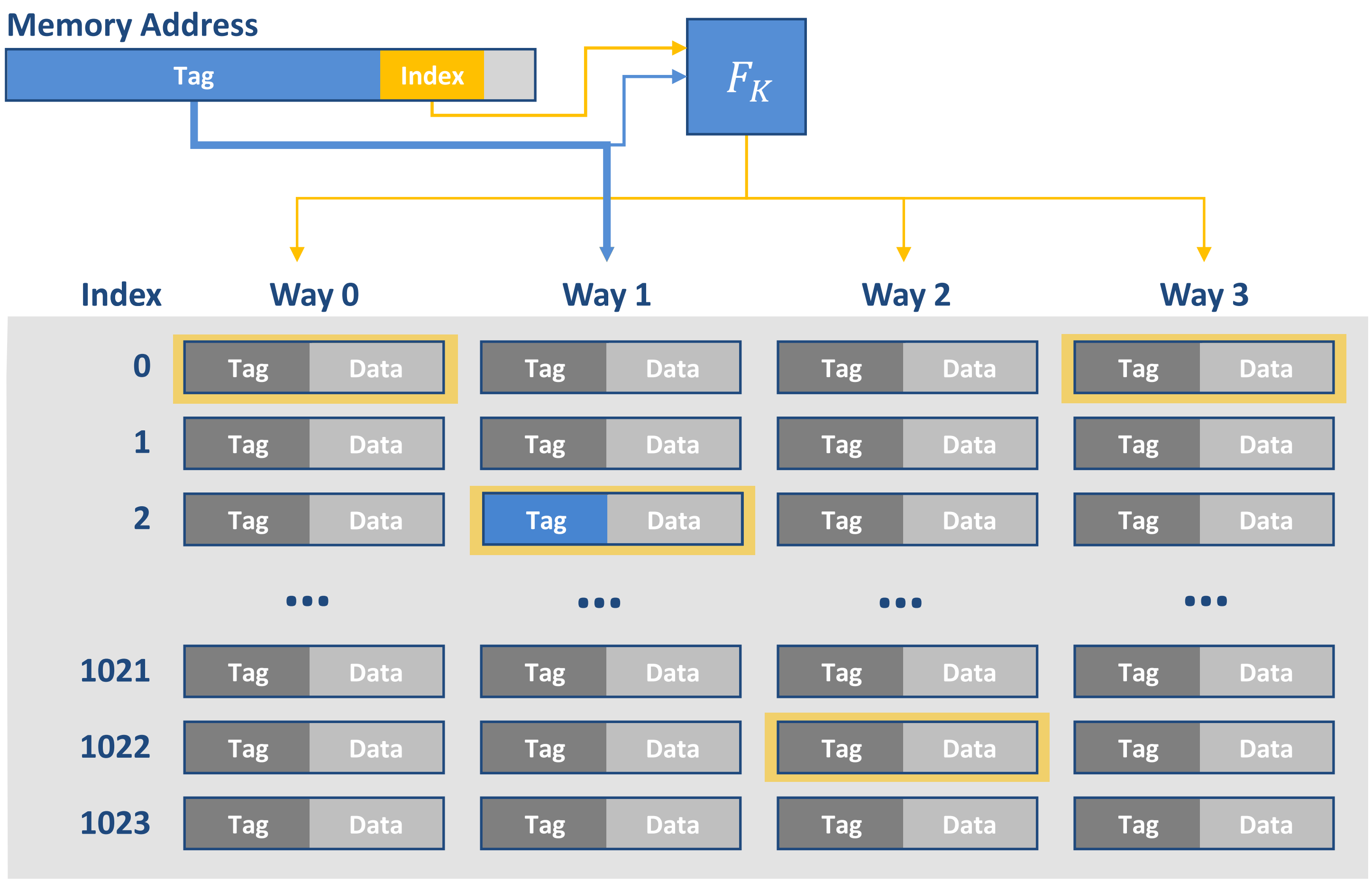
    → Cache miss = access
- **Prime + Probe**



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?**



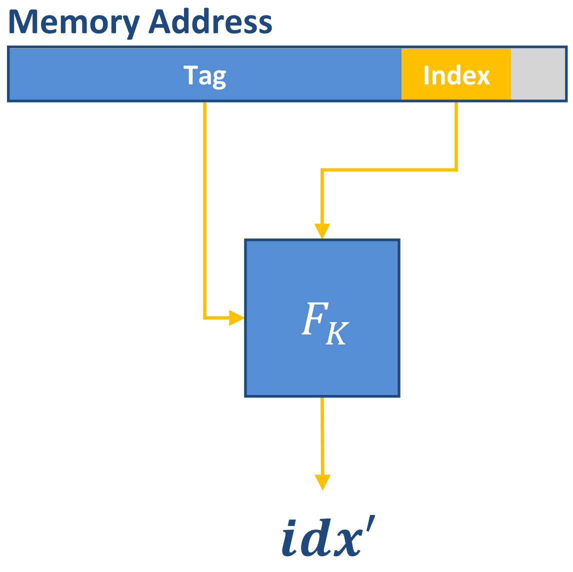
SCARF

Secure Cache Randomization Function

Design Rationale

Functional Requirements

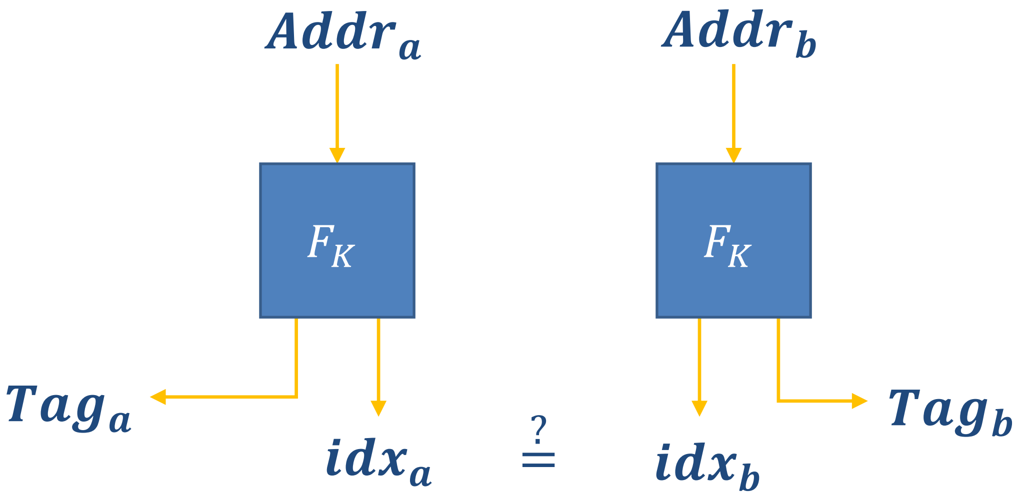
1. **Low Latency**
2. **Key Dependency**
3. **Invertibility (given the tag)**
   - For write-back caches
4. **We will focus on caches with 1024 sets**
   - Map 48-bit tag + 10-bit index to 10-bit randomized index
   - Offset bits must be ignored!



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**



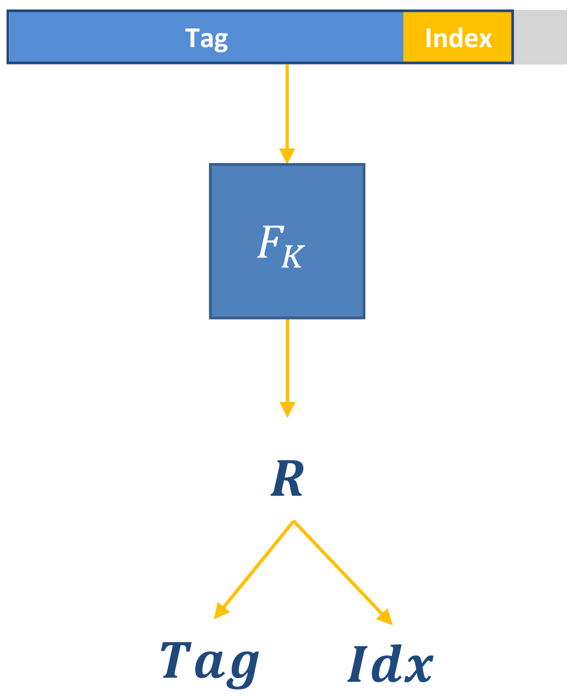
Design Rationale

Function Selection

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

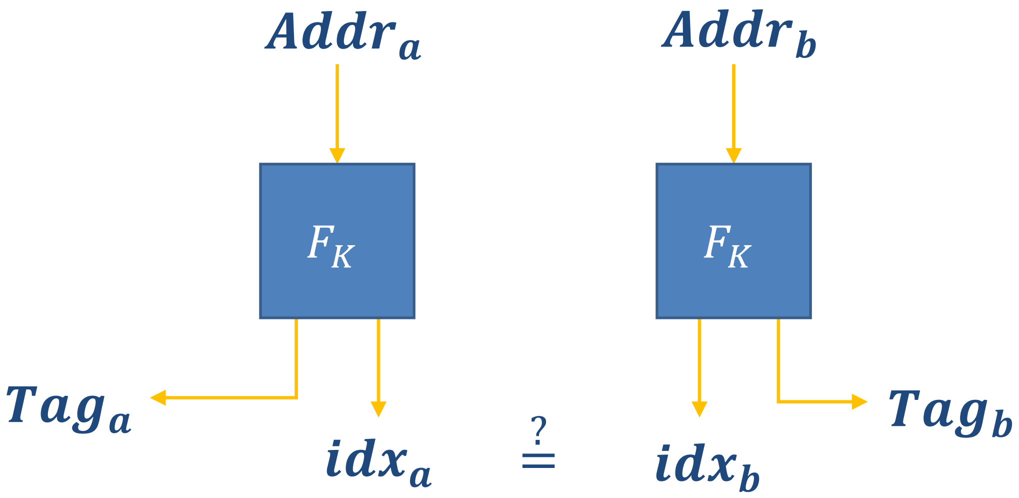
How do we design F_K?

- 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

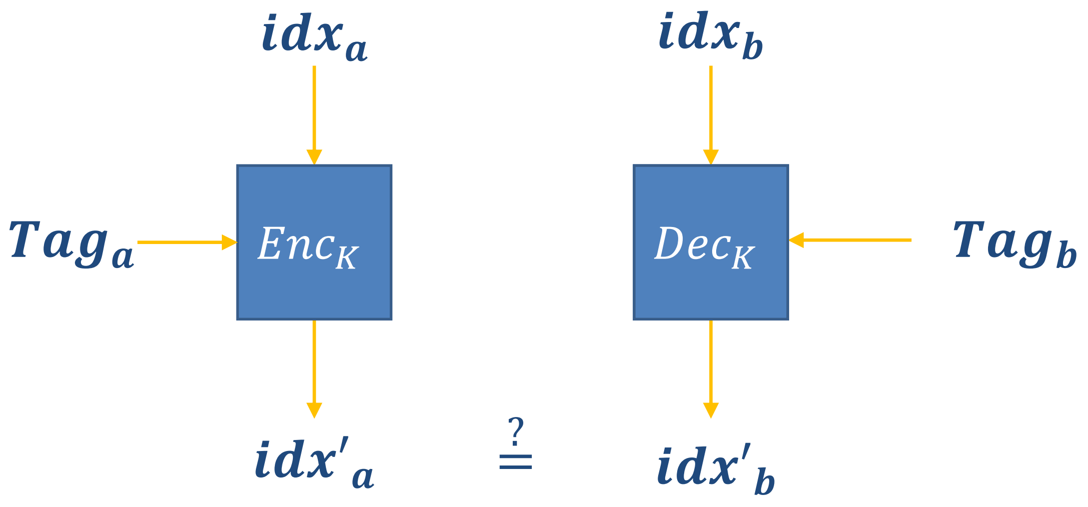
How do we design F_K?

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

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

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

Attacker's view



We can reduce the number of rounds by half!

SCARF

Function Selection

How do we design F_K?

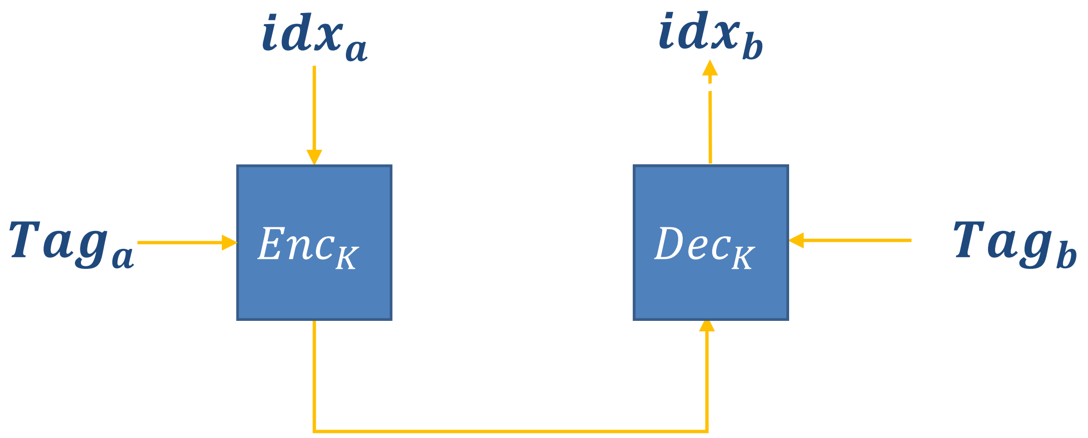
- SCARF: Create a 10-bit tweakable block cipher!
  - 1024 sets is a common choice
  - Simple attacker model allows latency optimization!
- The attacker learns if

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

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

We can reduce the number of rounds by half!

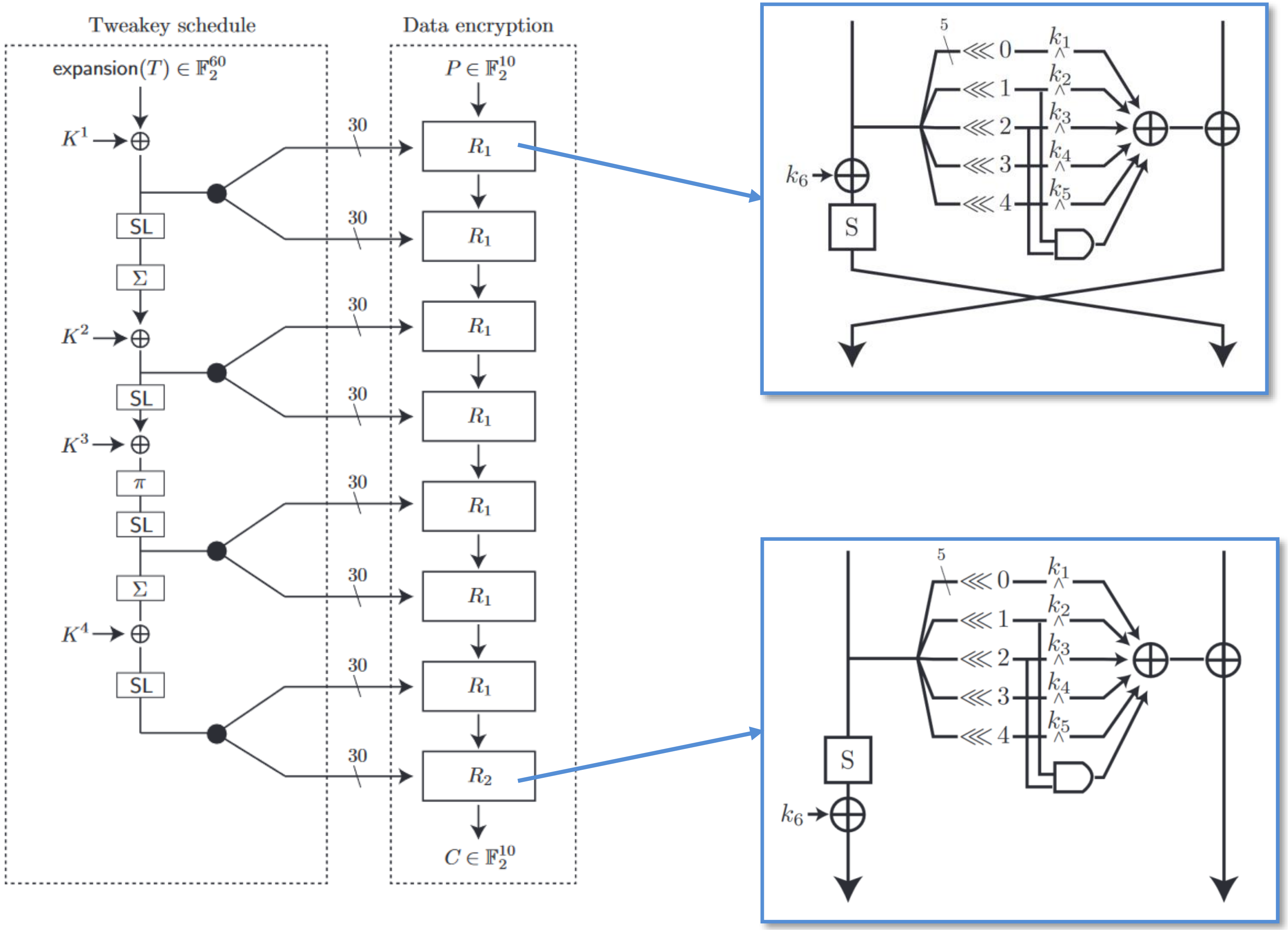
Designer‘s view



SCARF

Design

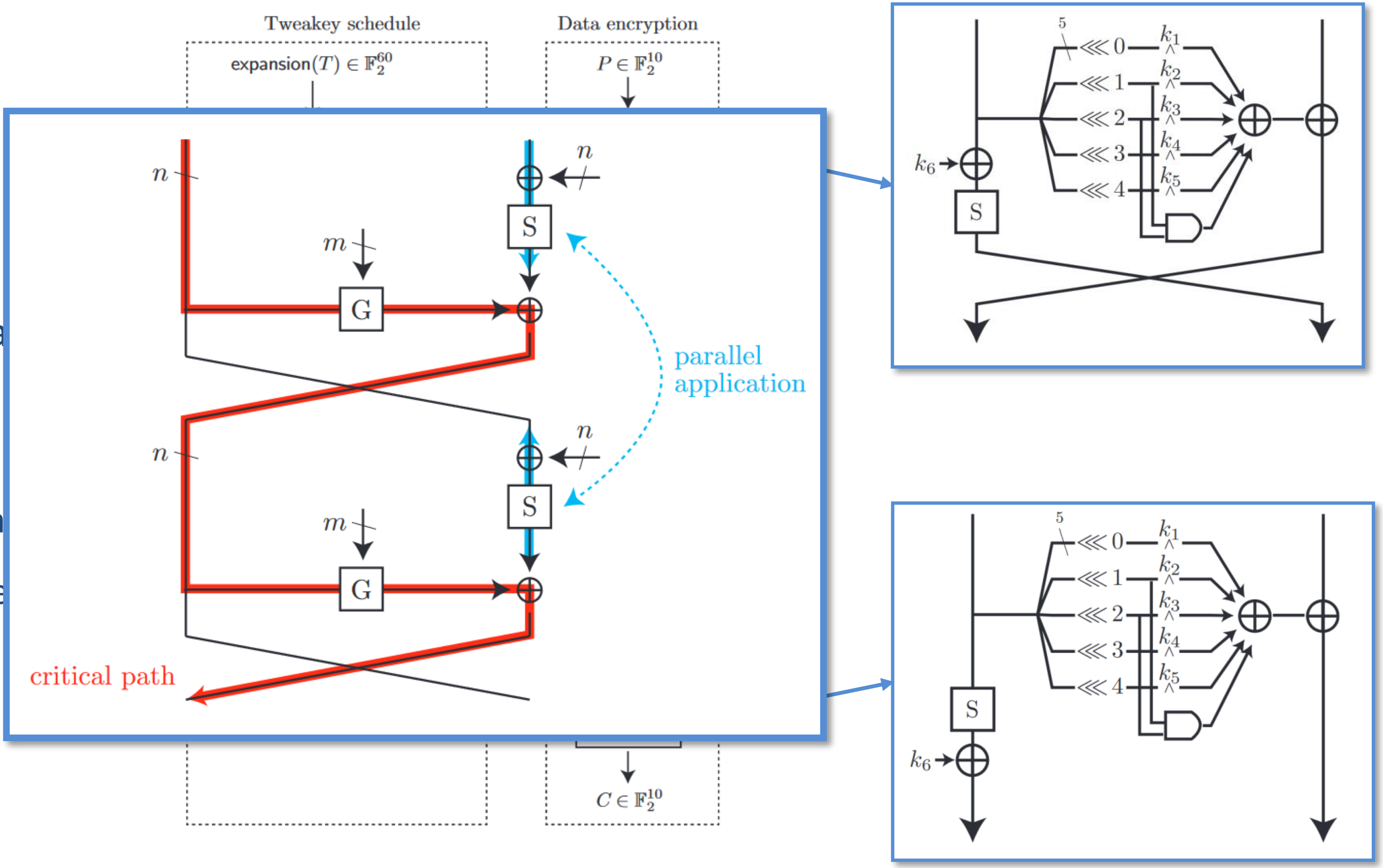
- **SCARF** is a 10-bit tweakable block cipher with 48-bit tweak
- 7 + 1 rounds
  - Latency optimized combination of SPN and Feistel structure
- 240 Bit key



SCARF

Design

- **SCARF** is a 10-bit tweakable block cipher with 48-bit twea
- 7 + 1 rounds
  - Latency optimized combina 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 1. Synthesis results using Nangate OCLs

Technology	45 nm		15 nm	
	Latency [ns]	Area [GE]	Latency [ps]	Area [GE]
PRINCE	4.74	12,554	628.49	17,484
MANTIS6	4.73	13,129	630.07	17,641
QARMA5	4.40	13,915	563.62	18,455
SCARF	**2.26**	**7,335**	**305.76**	**8,118**

Questions?

Jan Philipp Thoma

jan.thoma@rub.de