Hardware Implementation of Secure Communication in a Bus-Based Multi-Core System Using Tiny Encryption Algorithm

Team Dja Dja: Jianchao Gao, Dike Zhou, Ameya Rane
University of Michigan

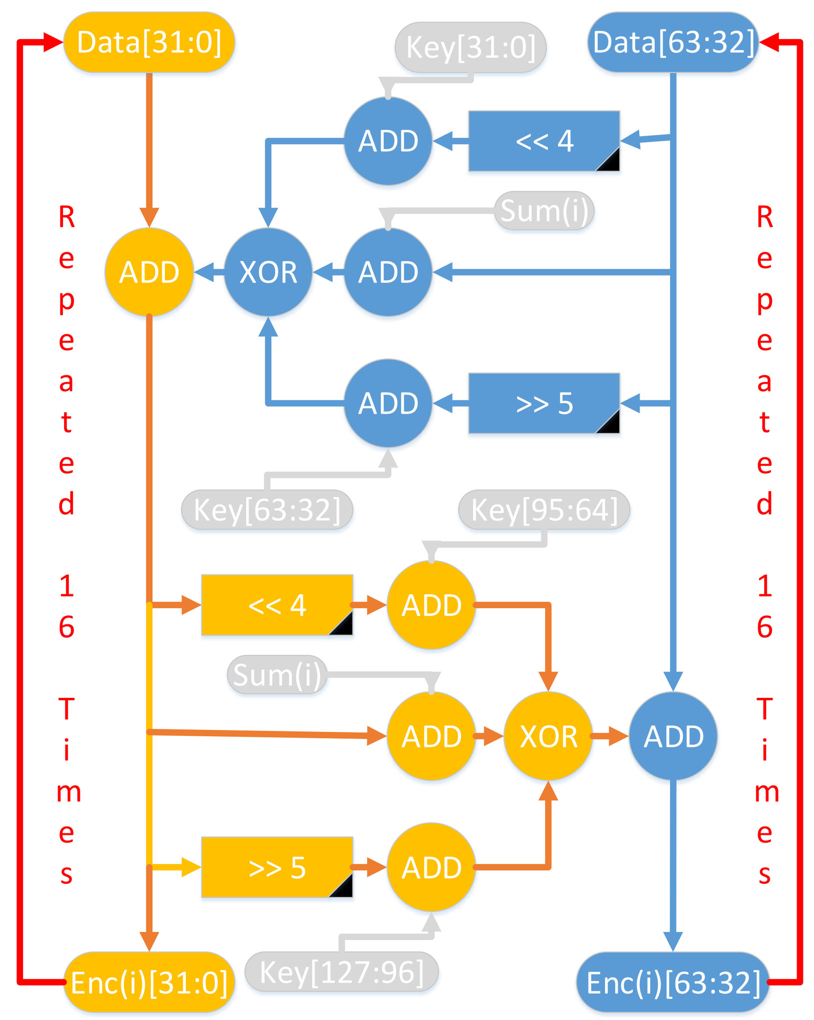
Introduction

- Security attacks- most common way of phishing today
- The interface (bus) more prone to attack, as compared to the cores
- Illegal snooping makes data vulnerable on the bus

Background

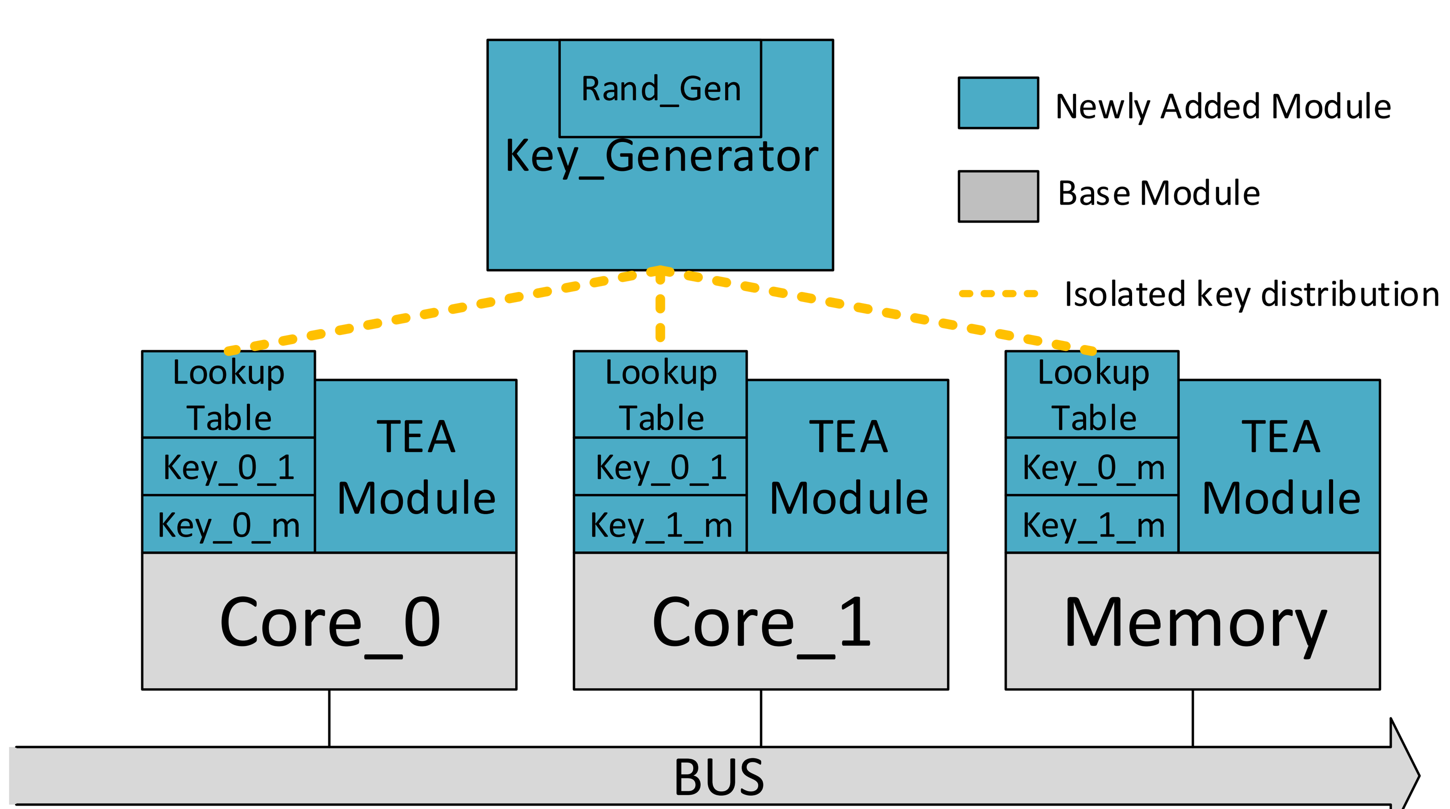
Tiny Encryption Algorithm (TEA)

- Simple symmetric encryption
- Fast to compute, easy logic
- Implemented to do 16 rounds of encryption in 2 cycles, can also be done in 1 cycle, or
- For our design (64 bit processor), one set of keys contains one 128-bit key



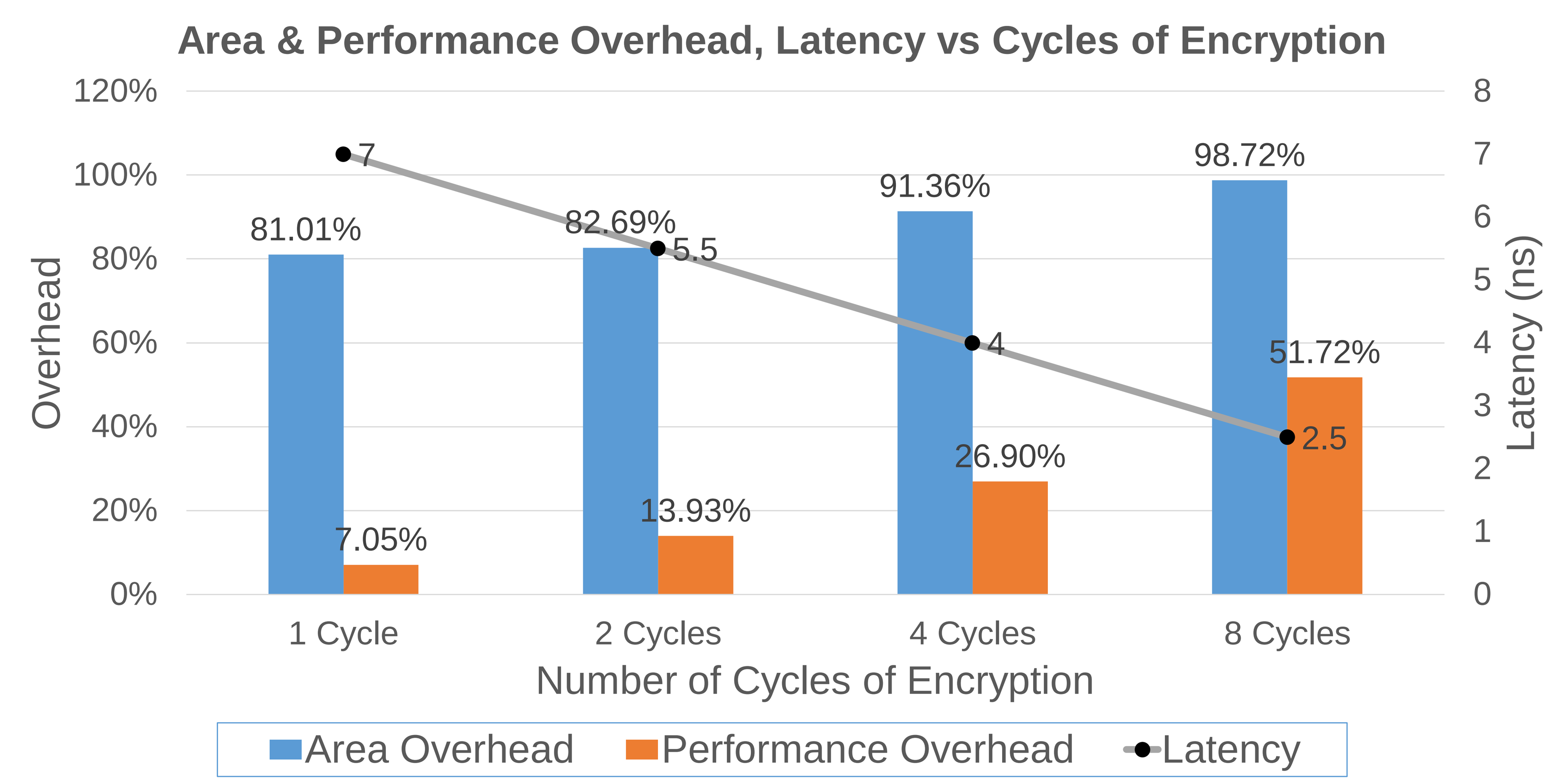
Proposed Architecture

- Dedicated encryption and decryption modules in each node
- Each pair of nodes will use a unique set of keys
- The "Cloud" generates and distribute keys to each pair of nodes
- Keys refreshed after a regular interval
- Lookup table helps to reduce penalty of refreshing keys



Core to bus: Encrypt with TEA encryption module
Bus to core: Decrypt with TEA decrypt module

Area & Performance Overhead



- Baseline case is an R10K-style two-core system with snooping bus and MESI cache coherence protocol
- Performance & area overhead go worse as number of cycles increase, but latency is improved
- Baseline design has a clock period of 6 ns, so 2-cycle module is our best choice in this case

Experiment Setup & Results

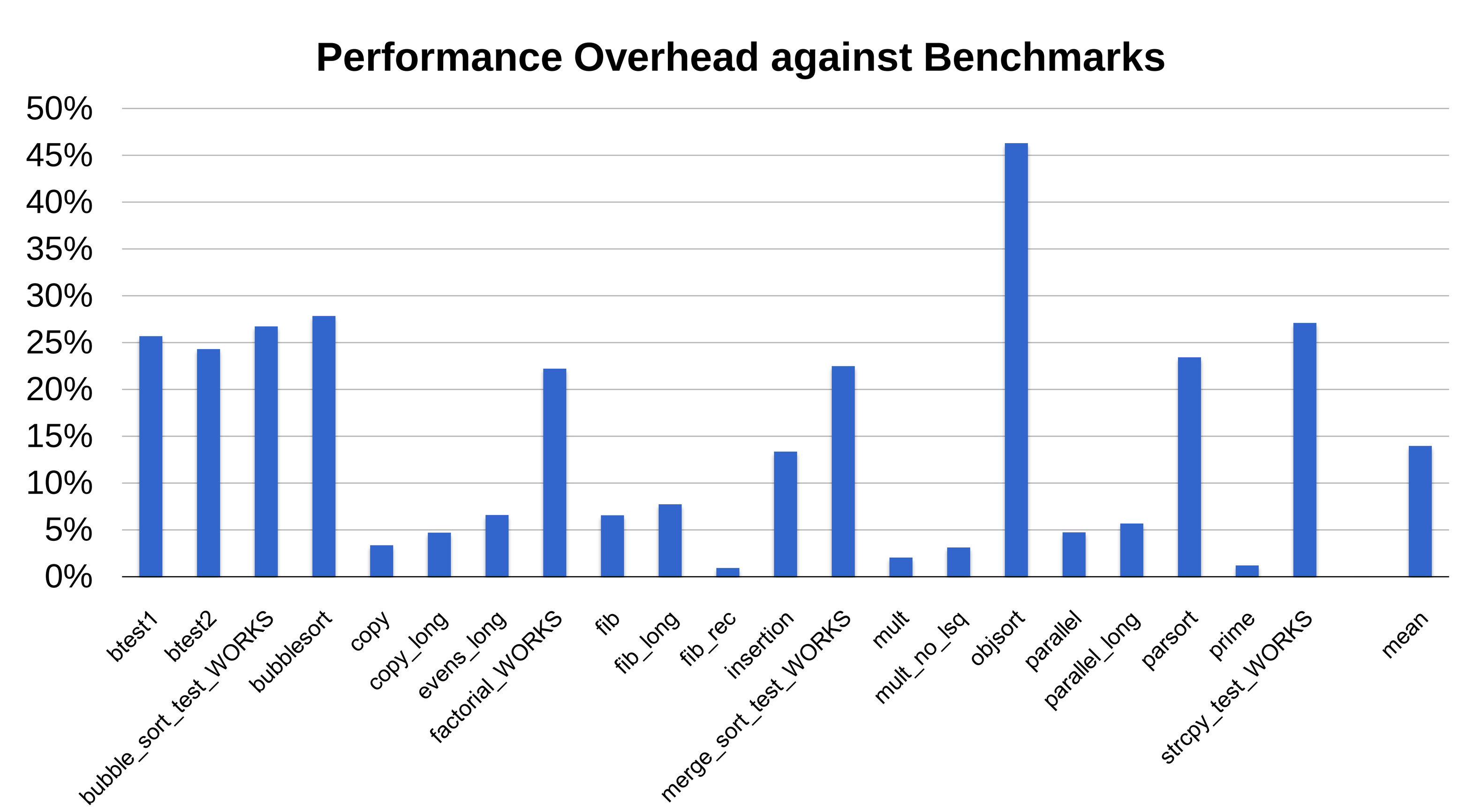
Baseline

- Two cores and one memory
- Snooping bus with MESI coherence protocol

Novelty

- Two-cycle encryption / decryption modules
- Keys are refreshed every 2000 cycles

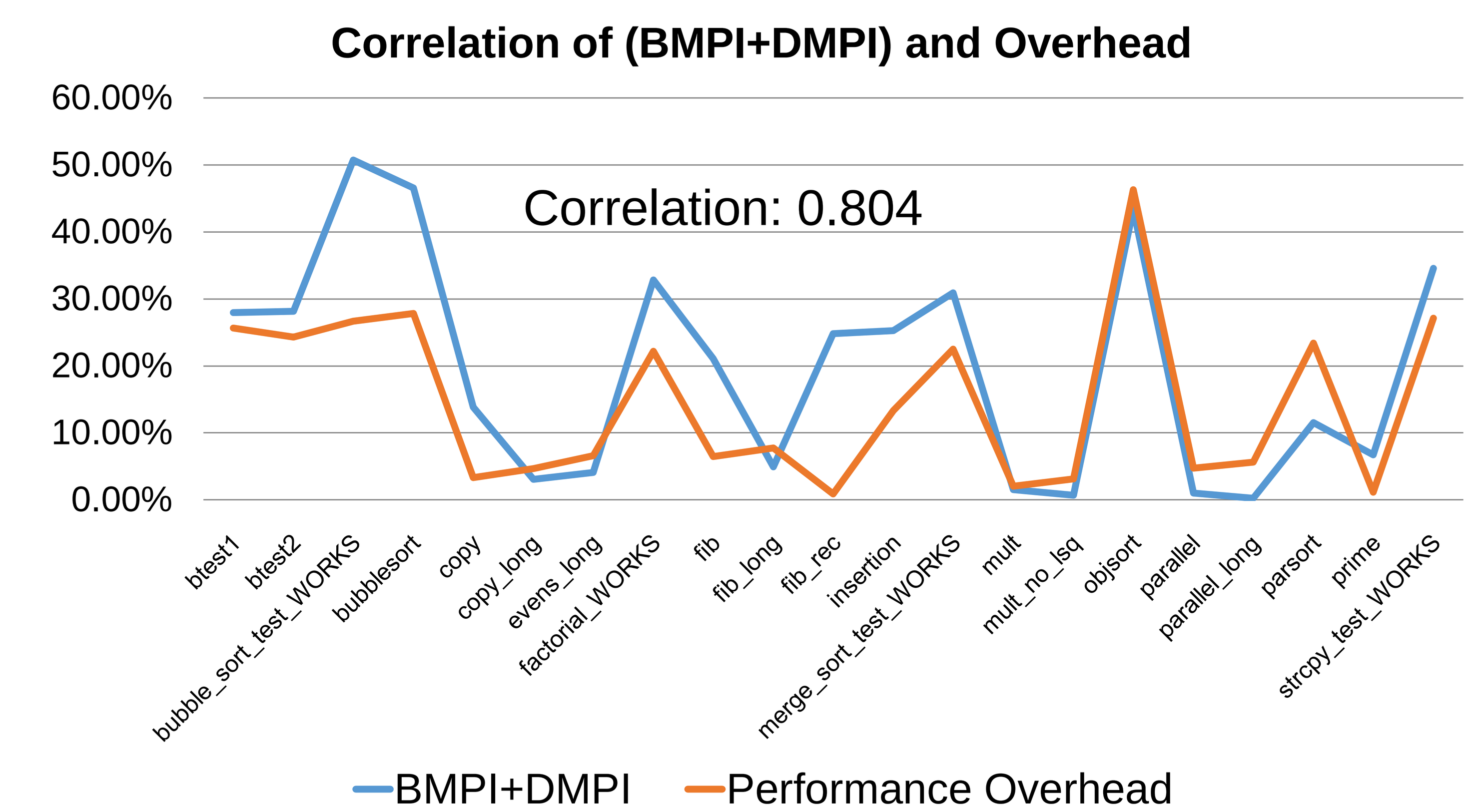
Result: Average performance overhead: **13.9%**; best case **0.9%**



Discussion

Two metrics:

- Branch Misprediction Per Instruction (BMPI)
- DCache Miss Per Instruction (DMPI)



Two solutions to reduce overhead:

- Reduce BMPI: better branch predictor
- Reduce DMPI: larger L1 cache; better replacement policy, etc.

Security Analysis

- Naive brute force: 10^{37} cycles to break
  Probability of hacking- 10^{-32} if refresh after 1M cycles
- Impossible Differential Analysis: 10^{18} cycles to break
  Probability of hacking- 10^{-17} if refresh after 1M cycles
- Keys safe to be replaced in less than 10^{17} cycles
  Relaxed time period for key generation

Conclusions

- Performance overhead: **13.9%** on average, **0.9%** in the best case
- Area overhead: 82.7% in proposed 2 cycle implementation

Given 1M refresh period

- Probability of **10^{-32}** to hack the algorithm using naïve brute force
- Probability of **10^{-17}** using Impossible Differential Analysis;