NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

TAD

Time Side-Channel Attack Defense of Obfuscated Source Code

Alexander Fell, Thinh Hung Pham, Siew-Kei Lam

January 22, 2019

Outline

Side-Channel Attacks

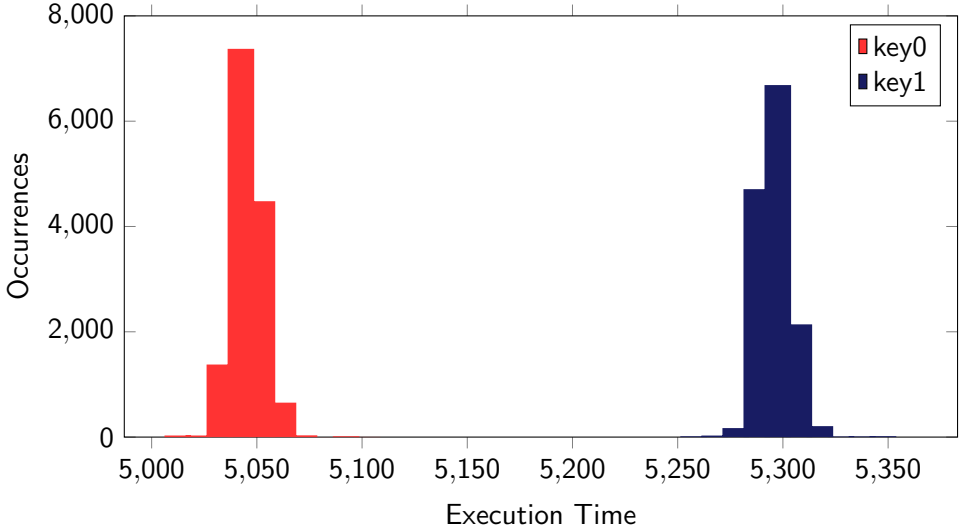
- Unintentional leakage of information through side-channels caused by variations in:
  - power consumption of the processor
  - temperature
  - electromagnetic emittance
  - duration of program execution
  - ...

Example Program

Input: data y, private key k, integer N
Output: $y^k \mod N$

```
1:  function ModExp(y, k, N)
2:      r ← 1
3:      for all ⟨k_i | i ∈ k⟩ do
4:          if (k_i = 1) then                    ▷ Critical Condition
5:              r ← (r × y) mod N
6:          end if
7:          y ← y^2 mod N
8:      end for
9:      return r mod N
10: end function
```

Execution Times of the Example Program



Reverse-Engineering Attacks

- Low-cost extraction, disassembly and analysis of binary program code of an obtained embedded system
- Obfuscation function O modifies the source code P such that

$$P(x) = O(P(x)) \text{ for all inputs } x$$

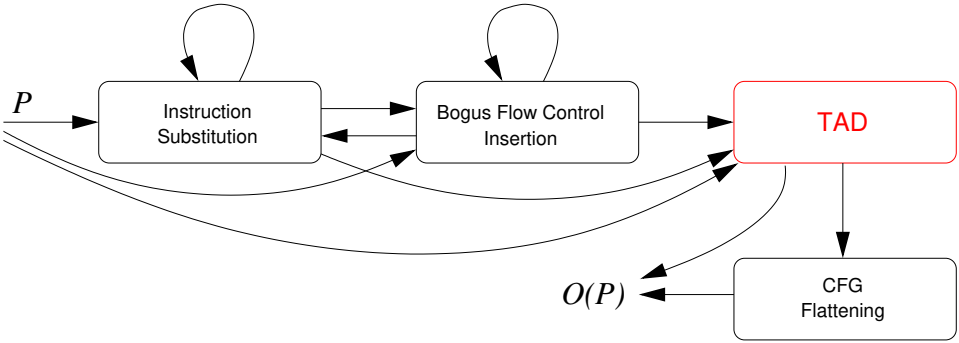
- Examples for O [1, 8, 7, 4, 5]:
  - Instruction substitution
  - Instruction set randomization
  - Flattening of control flow graph (CFG)
  - Insertion of bogus flow control
  - ...
- No cryptographic guarantee of security! [2, 6, 13, 12]

Motivation

A compiler that **automatically** modifies a program such that its functionality remains the same with an additional protection against

1. Reverse-Engineering **and**
2. Time Side-Channel Attacks

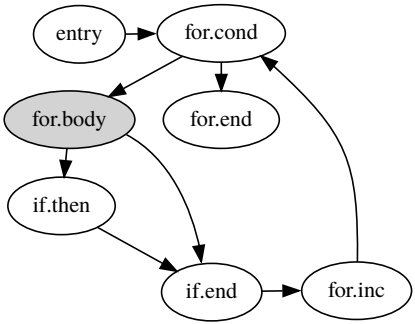
Obfuscation Function O



Time Side-Channel Attack Defense (TAD) I

Input: data y, private key k, integer N
Output: $y^k \mod N$

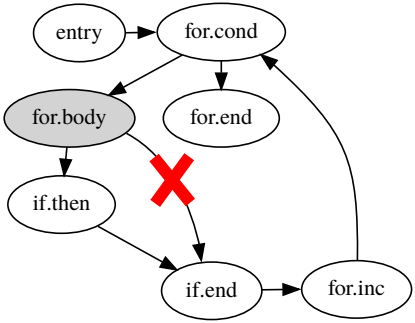
```
1:  function MODEXP(y, k, N)
2:      r ← 1
3:      for all ⟨k_i | i ∈ k⟩ do
4:          if (k_i = 1) then
5:              r ← (r × y) mod N
6:          end if
7:          y ← y^2 mod N
8:      end for
9:      return r mod N
10: end function
```



Time Side-Channel Attack Defense (TAD) II

Input: data y, private key k, integer N
Output: $y^k \mod N$

```
1:  function ModExp(y, k, N)
2:      r ← 1
3:      for all ⟨k_i | i ∈ k⟩ do
4:          if (k_i = 1) then
5:              r ← (r × y) mod N
6:          end if
7:          y ← y^2 mod N
8:      end for
9:      return r mod N
10: end function
```



Time Side-Channel Attack Defense (TAD) III

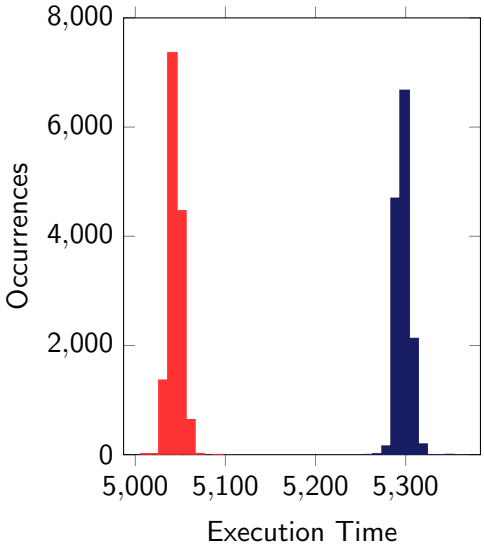
Input: data y, private key k, integer N
Output: $y^k \mod N$

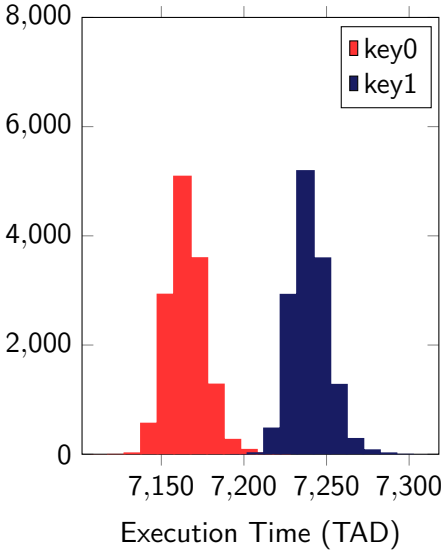
1: **function** MODEXP(y, k, N)
2: $r \leftarrow 1$
3: **for all** $\langle k_i | i \in k \rangle$ **do**
4: **if** $(k_i = 1)$ **then**
5: $r \leftarrow (r \times y) \mod N$
6: **end if**
7: $y \leftarrow y^2 \mod N$
8: **end for**
9: **return** $r \mod N$
10: **end function**

Input: data y, private key k, integer N
Output: $y^k \mod N$

1: **function** MODEXP(y, k, N)
2: $r \leftarrow 1$
3: **for all** $\langle k_i | i \in k \rangle$ **do**
4: $m \leftarrow k_i \times (-1)$
5: $rr \leftarrow (r \times y) \mod N$
6: $r \leftarrow (m \wedge rr) \vee (\neg m \wedge r)$
7: $y \leftarrow y^2 \mod N$
8: **end for**
9: **return** $r \mod N$
10: **end function**

TAD - Results (bare-metal)





TAD - Conclusion

- Removing divergence in CFG is insufficient.
  - Cache
  - Variable latency instructions
  - Memory access times
  - Branch prediction
  - ...

Solution:
Instructions independent of operand values $\rightarrow$ TAD with Custom Instructions and hardware diversification (TADCI)

TADCI - Modified Example Program

Let $CI_1 = \times$ and $CI_2 = \text{mod}$

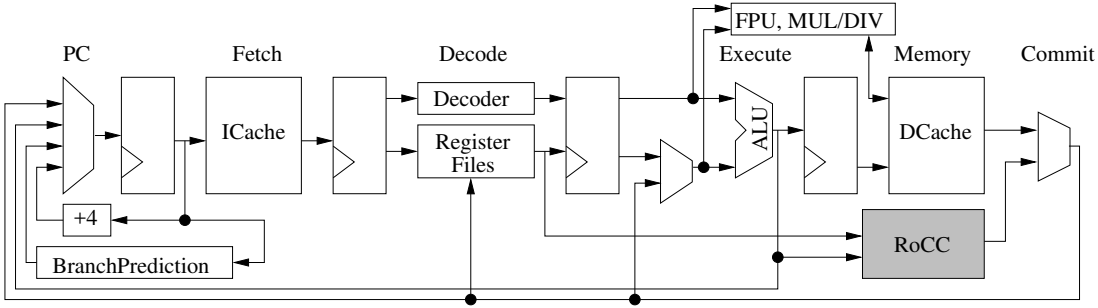
Input: data y, private key k, integer N
Output: $y^k \mod N$

1: **function** MODEXP(y, k, N)
2: $r \leftarrow 1$
3: **for all** $\langle k_i | i \in k \rangle$ **do**
4: $m \leftarrow k_i \times (-1)$
5: $rr \leftarrow (r \times y) \mod N$
6: $r \leftarrow (m \wedge rr) \vee (\neg m \wedge r)$
7: $y \leftarrow y^2 \mod N$
8: **end for**
9: **return** $r \mod N$
10: **end function**

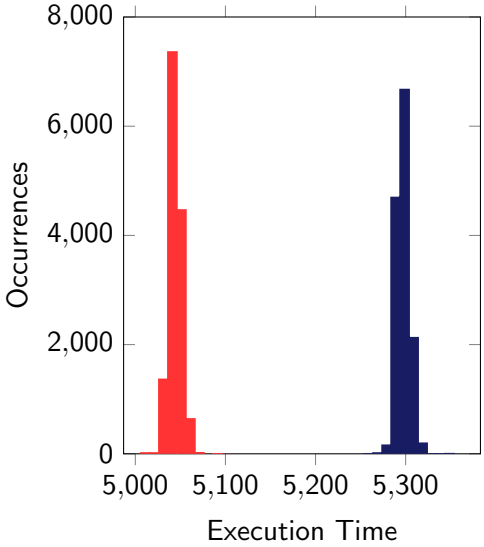
Input: data y, private key k, integer N
Output: $y^k \mod N$

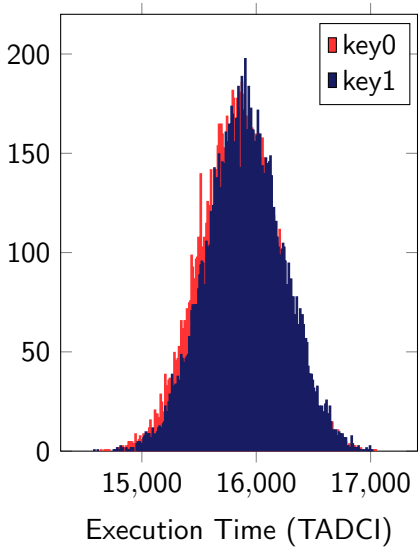
1: **function** MODEXP(y, k, N)
2: $r \leftarrow 1$
3: **for all** $\langle k_i | i \in k \rangle$ **do**
4: $m \leftarrow k_i \times (-1)$
5: $rr \leftarrow (r \, CI_1 \, y) \, CI_2 \, N$
6: $r \leftarrow (m \wedge rr) \vee (\neg m \wedge r)$
7: $y \leftarrow (y \, CI_1 \, y) \, CI_2 \, N$
8: **end for**
9: **return** $r \, CI_2 \, N$
10: **end function**

TADCI - Architecture Overview



TADCI - Results (bare-metal)

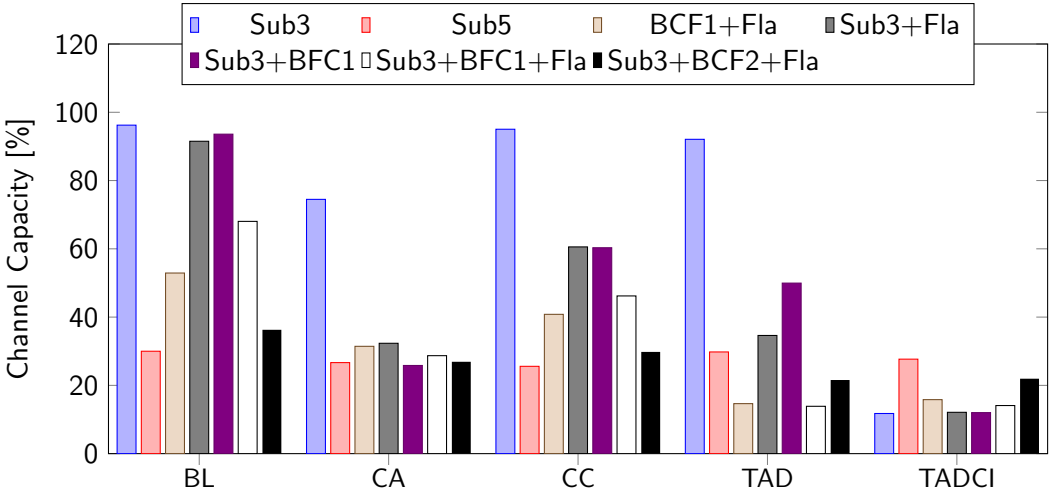




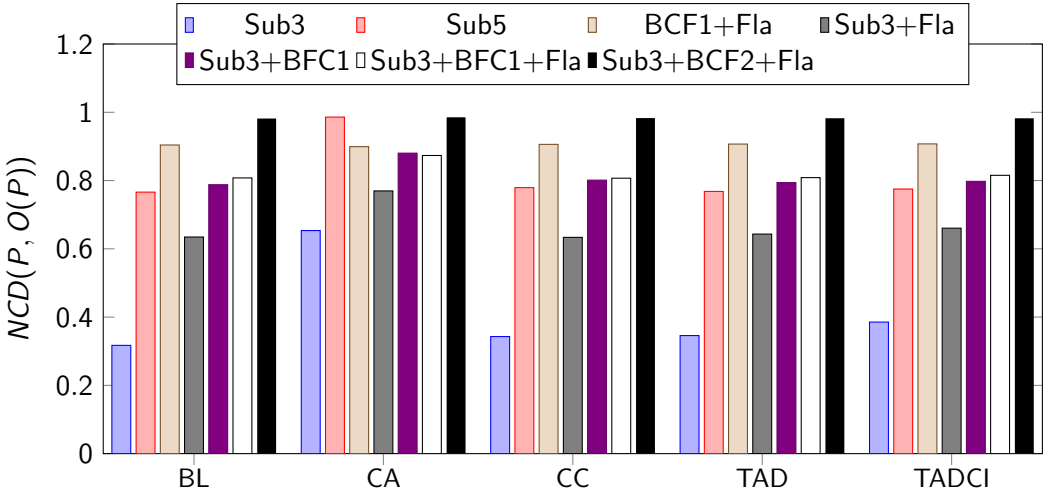
Test Environment

- RISC-V Rocket Core [3] on a Zynq7000 XC7Z020 FPGA
  - 64bit
  - 50MHz
  - Branch Prediction
  - L1 Instruction and Data Cache
  - L2 Cache
  - FPU
  - H/W Multiplier and Division Unit with Early Out
  - Rocket Chip Co-Processor
- Bare Metal Program Execution
  - RSA modular exponentiation (MODEXP) to encrypt or decrypt a message [11] from the benchmark suite introduced in [10]
  - Modular multiplication (MULMOD16) from the IDEA cipher [9]

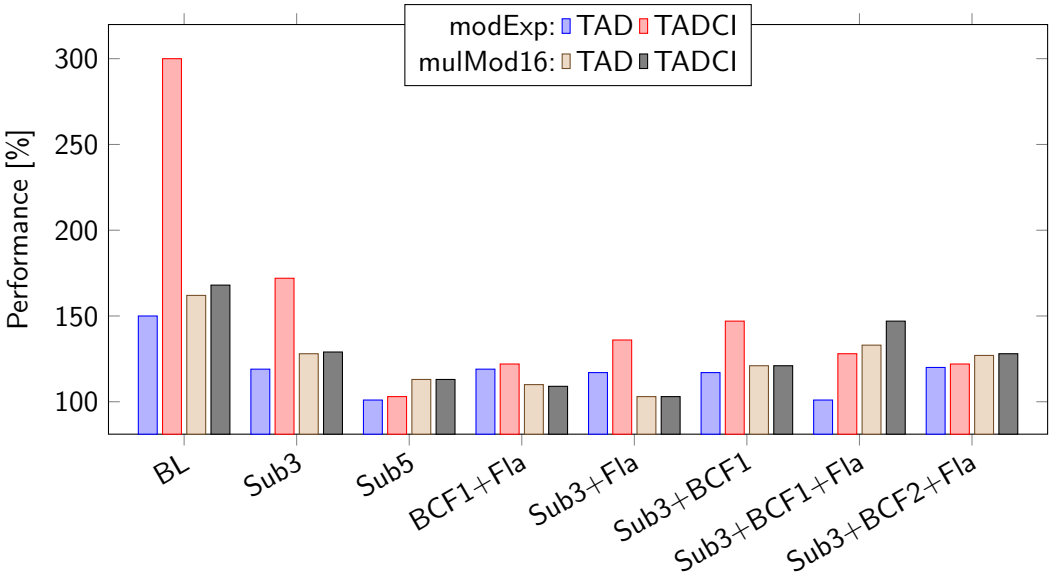
Results - Channel Capacity (modExp)



Results - Normalized Compression Distance *NCD* (modExp)



Results - Performance Impact



Hardware Requirements

Table: Hardware Resource Utilization

	RISC-V	RISC-V with RoCC	Change
LUTs	32310	32953	+2%
DSPs	15	25	+67%
BRAM	24	24	–

Conclusion

- Impact on performance, obfuscation strength and time side-channel attack mitigation depends highly on the input program.
- Minimal invasive to existing RISC-V architectures (RoCC)
- Low additional hardware resource requirements
- Suitable for low-noise environments/embedded systems
- External factors such as periphery influence time side-channel leakage

Thank You

References I

[1] B. Anckaert, M. Jakubowski, and R. Venkatesan.
Proteus: Virtualization for Diversified Tamper-resistance.
In *Proceedings of the ACM Workshop on Digital Rights Management*, pages 47–58, New York, NY, USA, 2006. ACM.

[2] D. Apon, Y. Huang, J. Katz, and A. J. Malozemoff.
Implementing Cryptographic Program Obfuscation.
IACR Cryptology ePrint Archive, 2014:779, 2014.

[3] K. Asanović, R. Avižienis, and J. e. a. Bachrach.
The Rocket Chip Generator.
Technical Report UCB/EECS-2016-17, EECS Department, University of California, Berkeley, 2016.

[4] C. S. Collberg and C. Thomborson.
Watermarking, tamper-proofing, and obfuscation - tools for software protection.
IEEE Transactions on Software Engineering, 28(8):735–746, Aug 2002.

[5] M. Fyrbiak, S. Rokicki, N. Bissantz, R. Tessier, and C. Paar.
Hybrid Obfuscation to Protect Against Disclosure Attacks on Embedded Microprocessors.
IEEE Transactions on Computers, 67(3):307–321, March 2018.

[6] M. Horváth.
Survey on Cryptographic Obfuscation.
IACR Cryptology ePrint Archive, 2015:412, 2015.

[7] P. Junod, J. Rinaldini, J. Wehrli, and J. Michielin.
Obfuscator-LLVM – Software Protection for the Masses.
In *Proc. Int. Workshop on Software Protection*, pages 3–9. IEEE, 2015.

[8] M. Kainth, L. Krishnan, C. Narayana, S. G. Virupaksha, and R. Tessier.
Hardware-Assisted Code Obfuscation for FPGA Soft Microprocessors.
In *DATE*, pages 127–132. ACM, 2015.

[9] X. Lai.
On the design and security of block ciphers, 1992.

References II

[10] H. Mantel and A. Starostin.
Transforming Out Timing Leaks, More or Less, pages 447–467.
Springer, ESORICS 2015, 2015.

[11] R. L. Rivest, A. Shamir, and L. Adleman.
A Method for Obtaining Digital Signatures and Public-key Cryptosystems.
Commun. ACM, 21(2):120–126, Feb. 1978.

[12] S. Schrittwieser and S. Katzenbeisser.
Code Obfuscation against Static and Dynamic Reverse Engineering.
In *13th International Conference on Information Hiding (IH)*, volume 6958, pages 270–284. Springer, May 2011.

[13] S. Schrittwieser, S. Katzenbeisser, J. Kinder, G. Merzdovnik, and E. Weippl.
Protecting Software Through Obfuscation: Can It Keep Pace with Progress in Code Analysis?
ACM Computing Surveys, 49(1):4:1–4:37, Apr. 2016.