# A Secure and Robust Approach to Software Tamper Resistance

***Sudeep Ghosh***, Jason D. Hiser and
Jack W. Davidson
*University of Virginia*

# Motivation

- Software performs critical functionality.
  - Needs to be protected

- Limitations with current AT technology
  - Vulnerable to dynamic analysis.
  - Require special hardware.
  - A number of solutions incur high overhead or require strict resource guarantees.

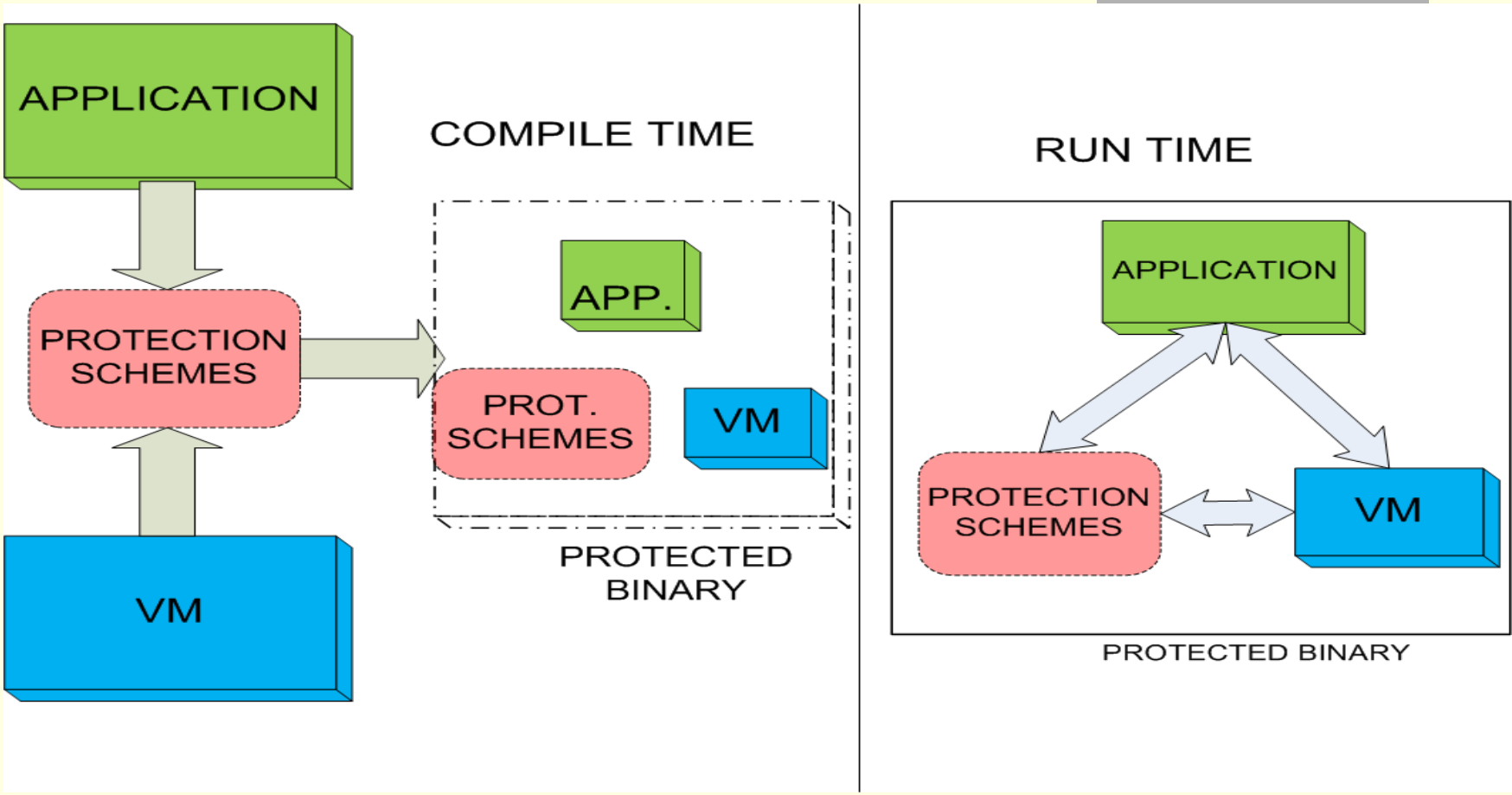
# Threat Model

- White-box attack model
  - Adversary can inspect, modify or forge any information.
  - They can modify the OS to return inaccurate information.
  - Hardware cannot be trusted.
- Given enough time and resources, the adversary can succeed in manually inspecting and modifying programs.
- Most attacks are supported by automated analysis.

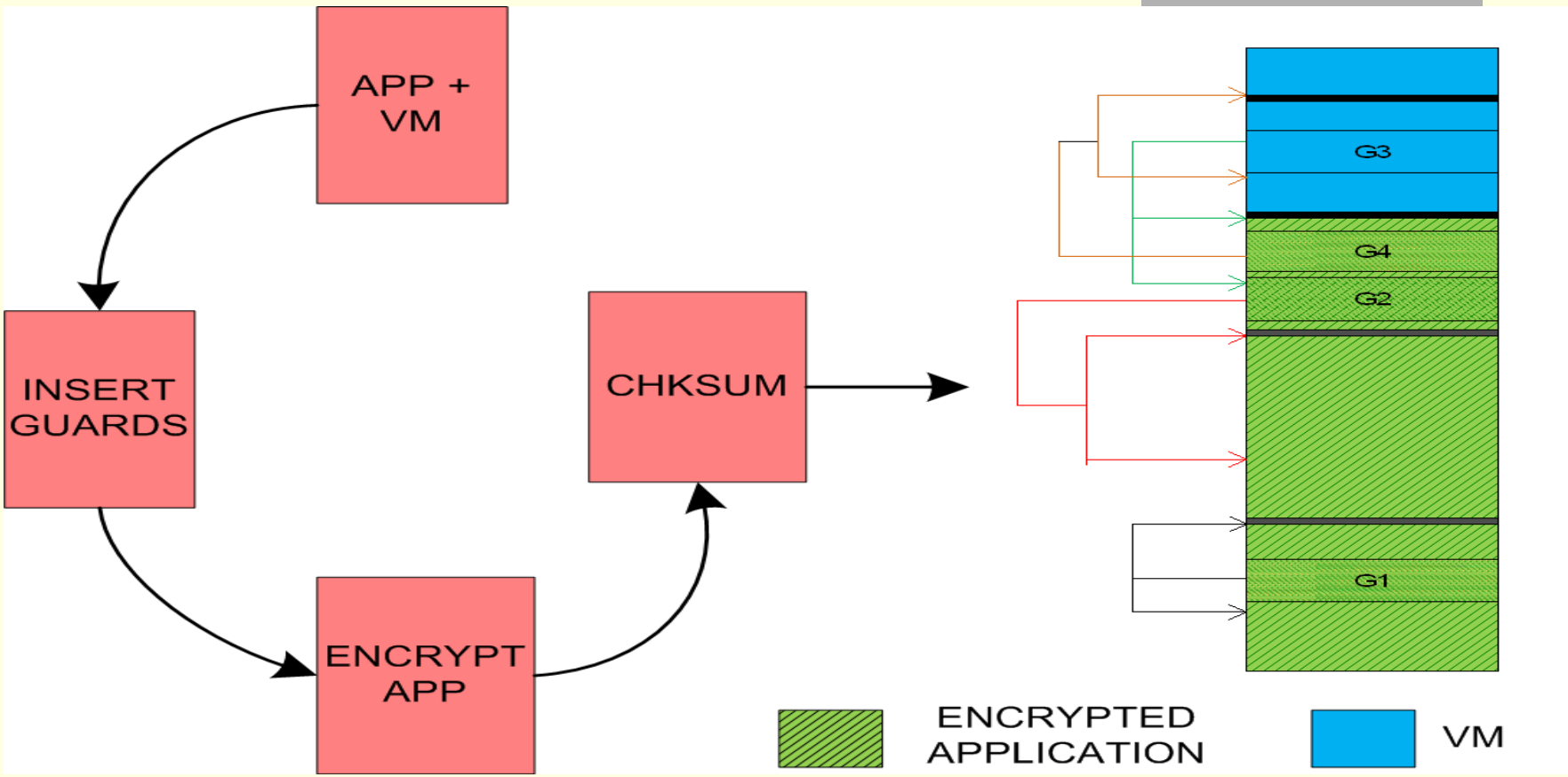
# Goal

The goal of this research is to hamper automated analysis and modification of programs.

# Security Architecture

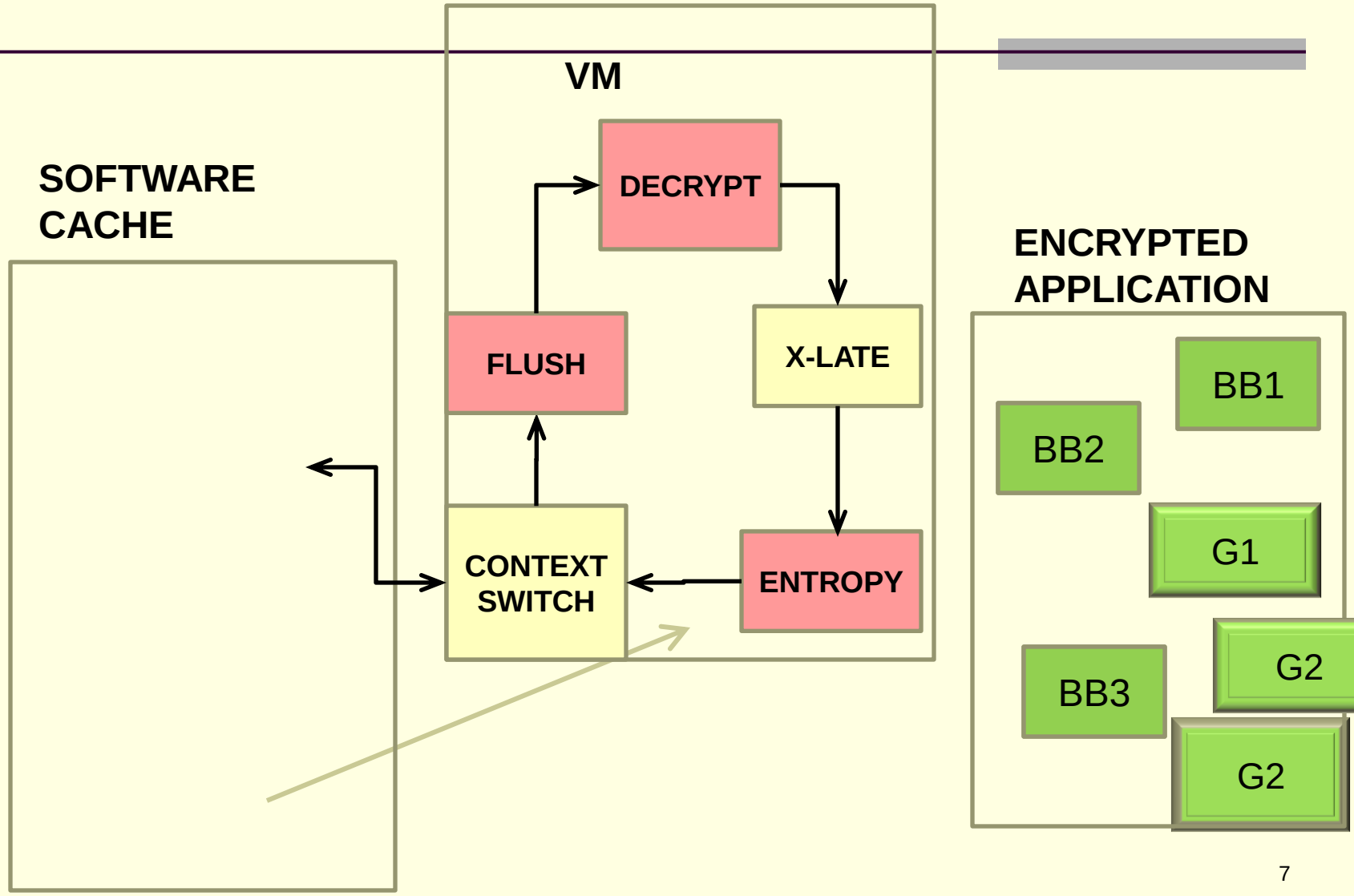


# Compile Time




Chang, H., Atallah, M.: Protecting software code by guards. In: Proceedings of the ACM Workshop on Security and Privacy in Digital Rights Management. (2002)

# Run time


VM
SOFTWARE CACHE
DECRYPT
FLUSH
X-LATE
CONTEXT SWITCH
ENTROPY
ENCRYPTED APPLICATION
BB1
BB2
G1
G2
BB3
G2

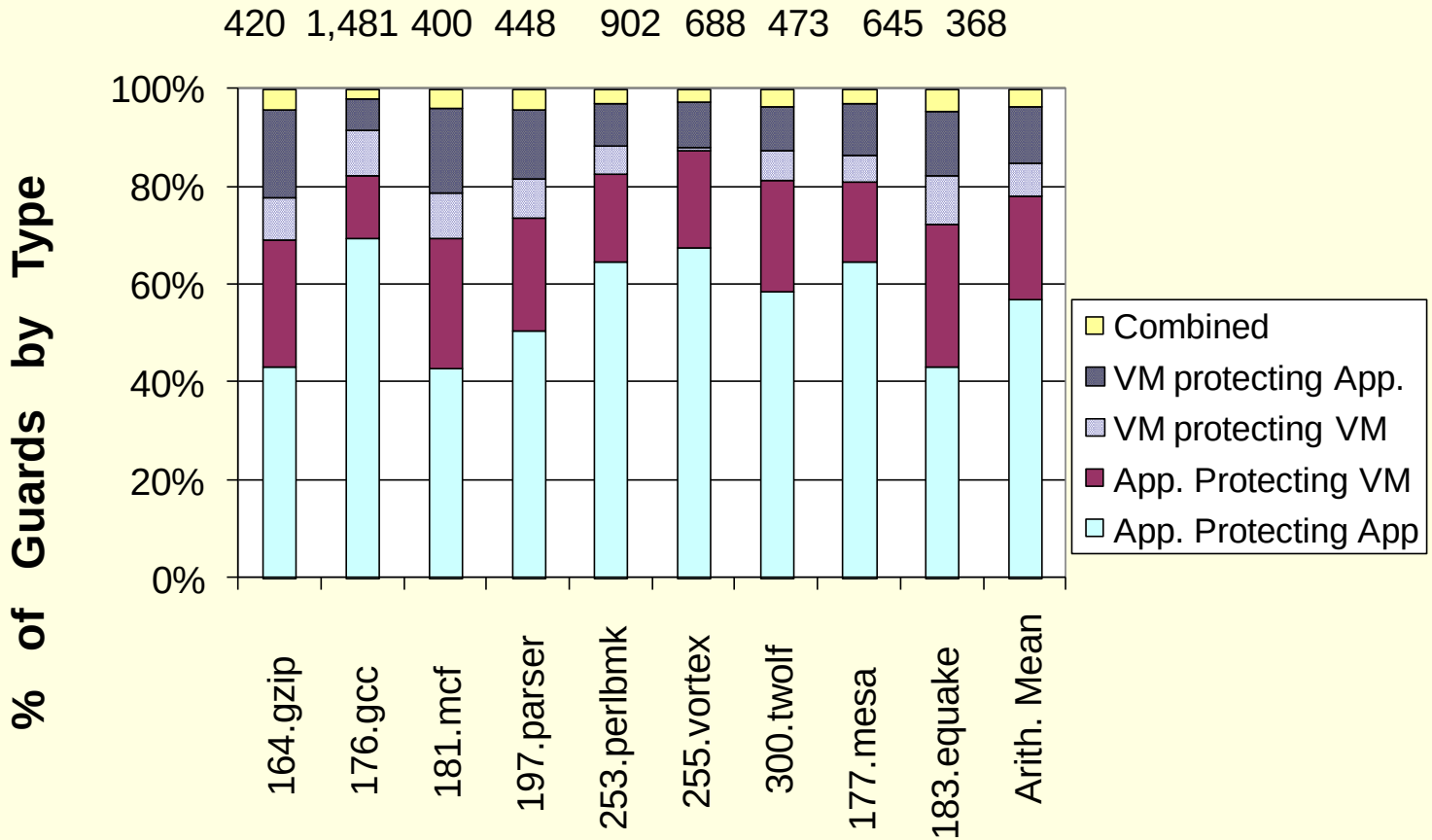
# Contributions

- Mutual Reinforcement
  - Encryption protects guards
  - Guards protect the decryption and flushing code
  - Flushing provides variance in the location of guards and prevents persistent changes to application code
- Imparts missing dynamism
  - Can be combined with other protection mechanisms (e.g. branch functions)

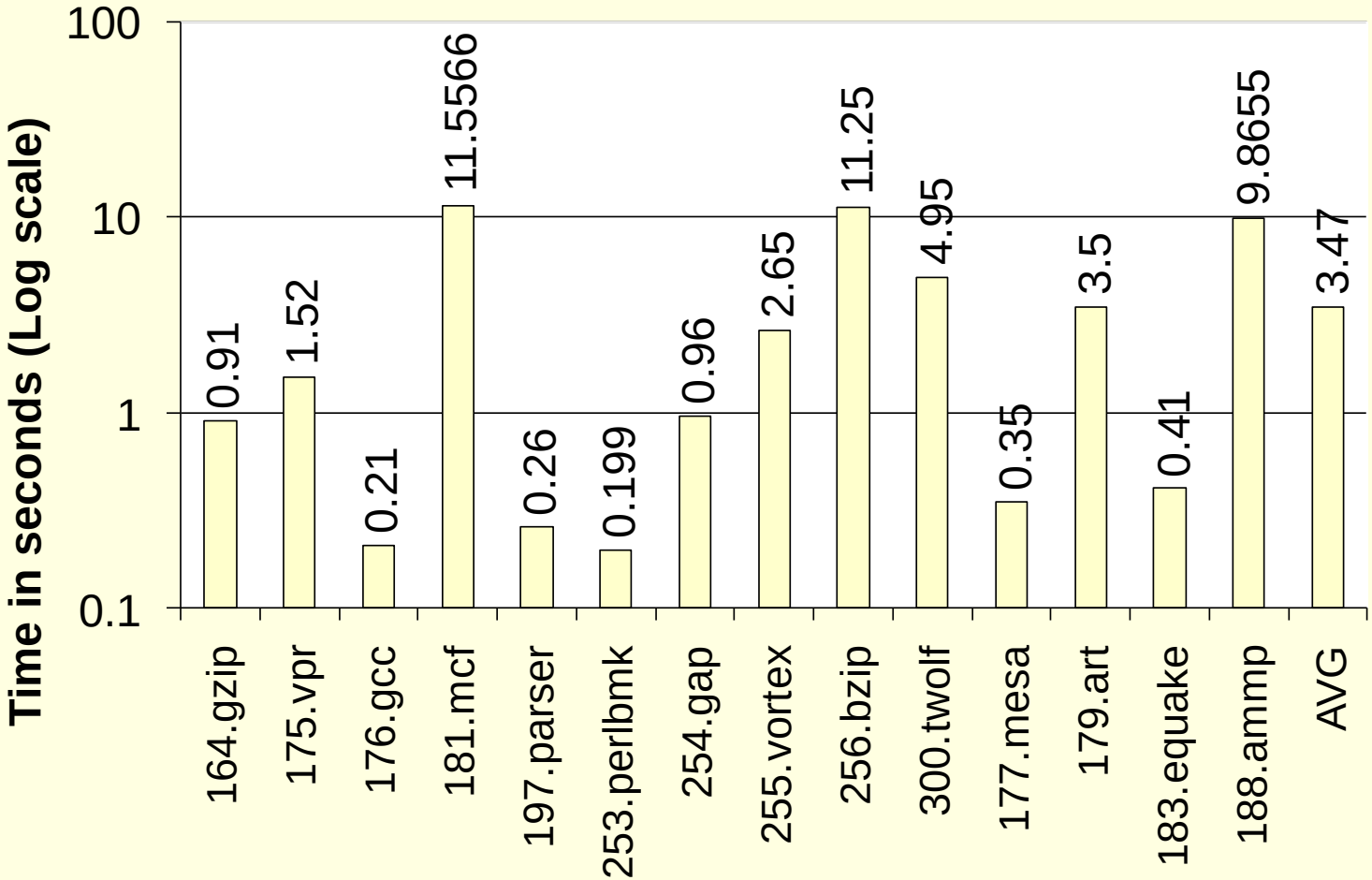
# Experimental Setup

- Automated prototype for evaluating and assessing our ideas.
- X86 platform
- Linux OS
- GCC toolchain
- Max Code Cache Size = 4 MB
- SPEC2000 C language benchmarks
- Guard coverage (number of guards protecting each byte of application code) = 4

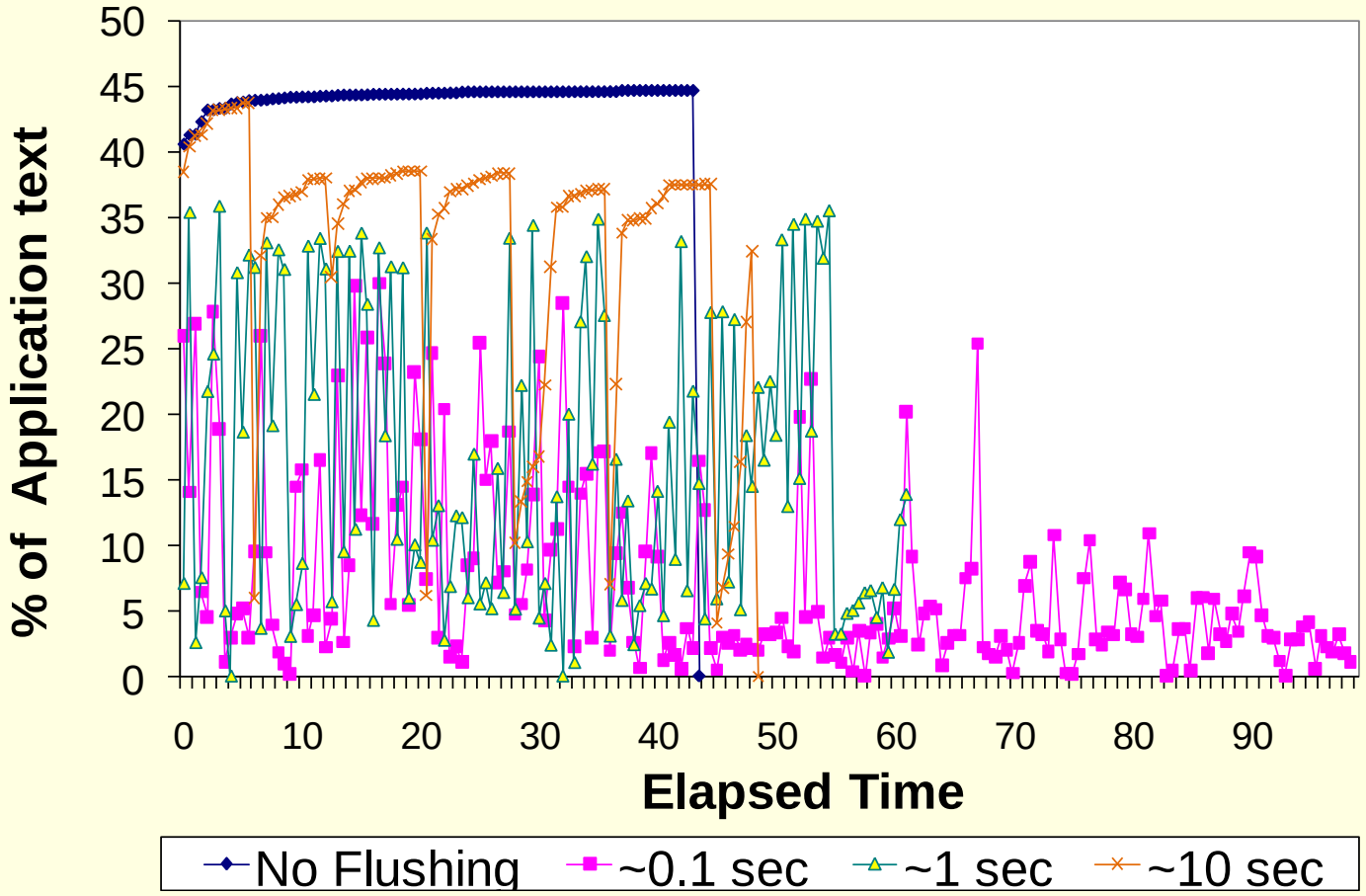
# Distribution of Guards



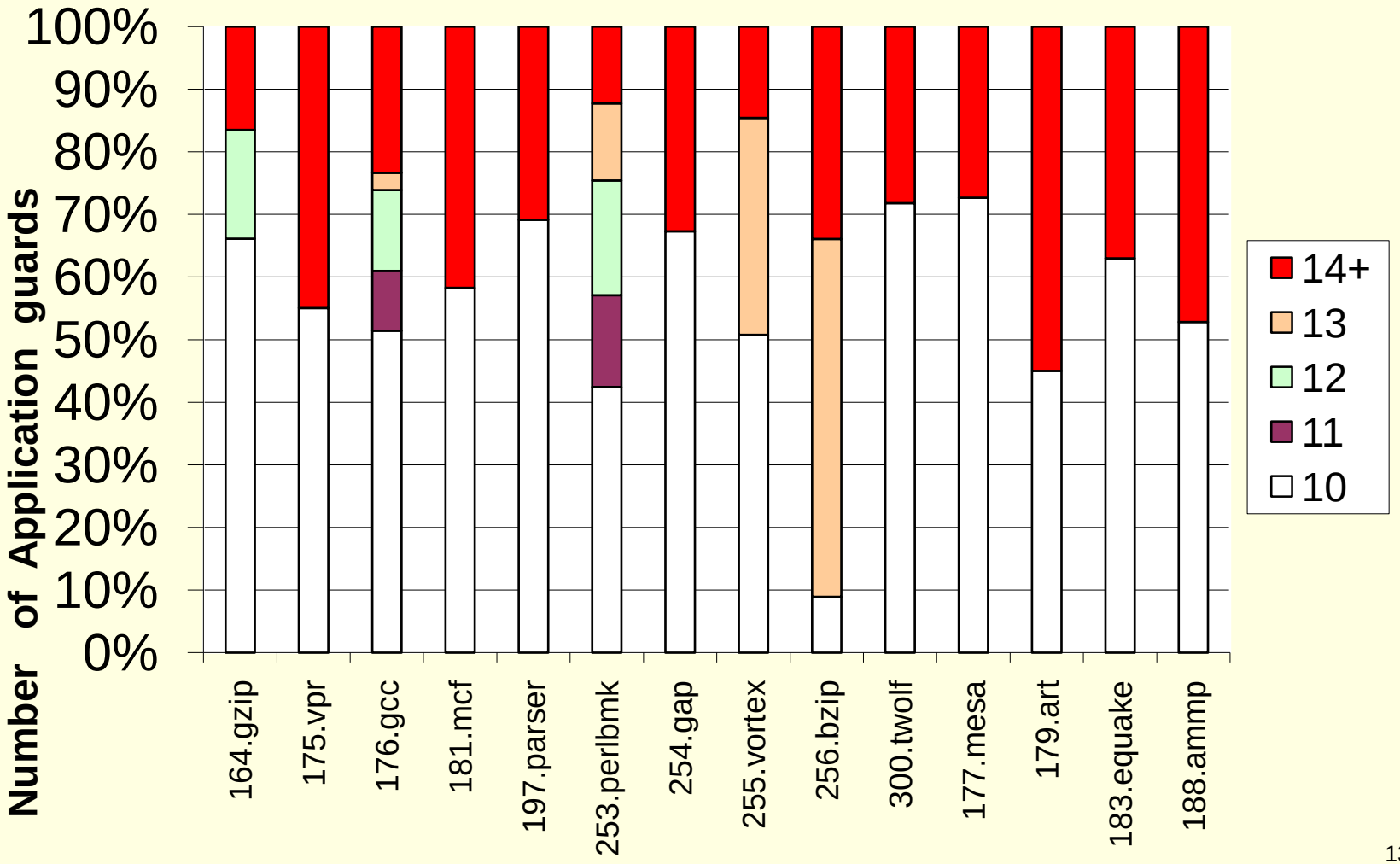
# Effectiveness



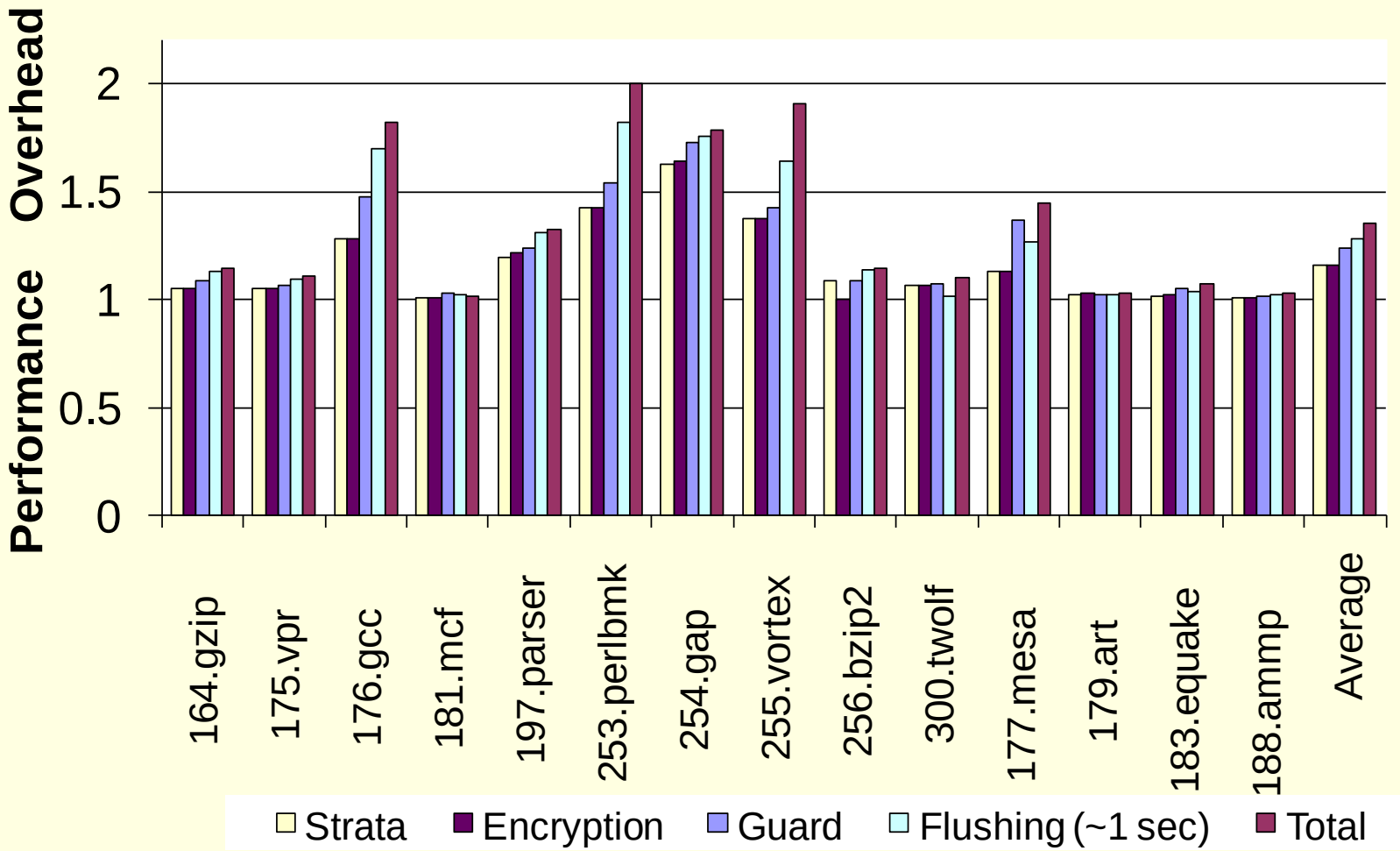
# Flushing


% of Application text
50
45
40
35
30
25
20
15
10
5
0
0
10
20
30
40
50
60
70
80
90
Elapsed Time
No Flushing
~0.1 sec
~1 sec
~10 sec

# Code Shifting



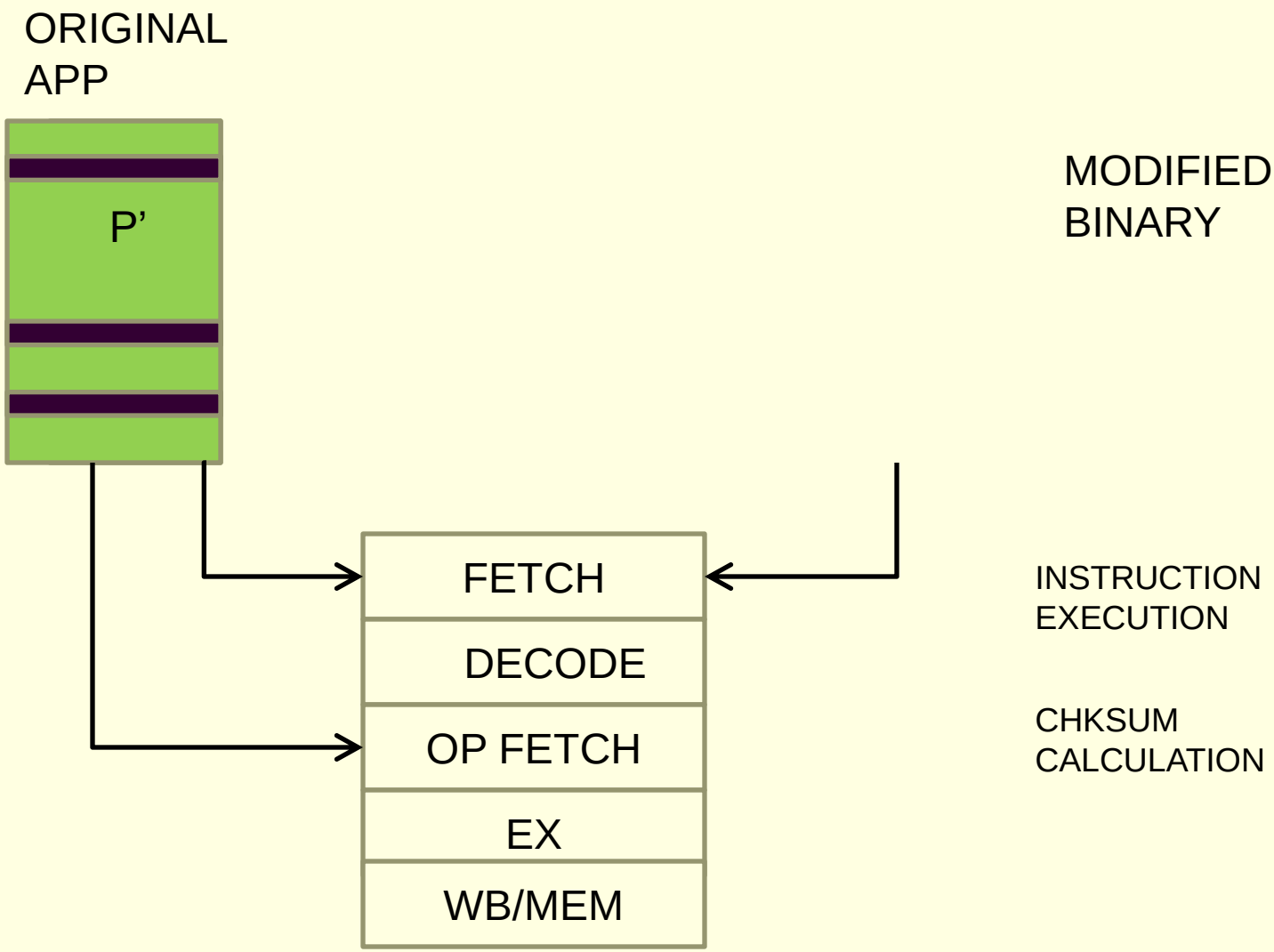
# Performance



# Security Discussion

- Effectiveness against static and dynamic analysis
  - Encryption prevents static analysis
    - Knowledge of VM code irrelevant
  - Dynamic code relocation hampers analysis at run time
    - Difficult to launch iterative attacks.

# Split Memory Attack on Guards



# Skype Case Study

- Security Problems
  - Bulk decryption
  - Collusion attack on guards
- Our defenses
  - On demand decryption and periodic flushing makes it hard for the adversary to analyze whole application.
  - Guards execute from different locations across runs, so collusion attacks will not be successful.

Biondi, P., Fabrice, D.: Silver needle in the skype. In: Black Hat Europe, Amsterdam, the Netherlands (2006)

# Related Works

- Software Tamper Resistance
  - Aucsmith, D.: Tamper resistant software:An implementation.
  - Chang, H., Atallah, M.: Protecting software code by guards.
- Code Encryption
  - Cappaert, J., Preneel, B., Anckaert, B., Madou, M., Bosschere, K.D.: Towards tamper resistant code encryption: Practice and experience
- Remote Tamper-proofing
  - Collberg, C., Nagra, J., Snavely, W.: bianlian: Remote tamper-resistance with continuous replacement.
- VM based approaches
  - Anckaert, B., Jakubowski, M., Venkatesan, R.: Proteus: virtualization for diversified tamper-resistance.

# Conclusions

- We have introduced a novel approach to software tamper resistance, using process-level virtualization.
- Contributions
  - Dynamic code obfuscation.
  - Granularity of decryption is much finer than previous work
  - Increased resistance against OS based attacks on guards
  - Periodic flushing ensures any successful modification is temporary.