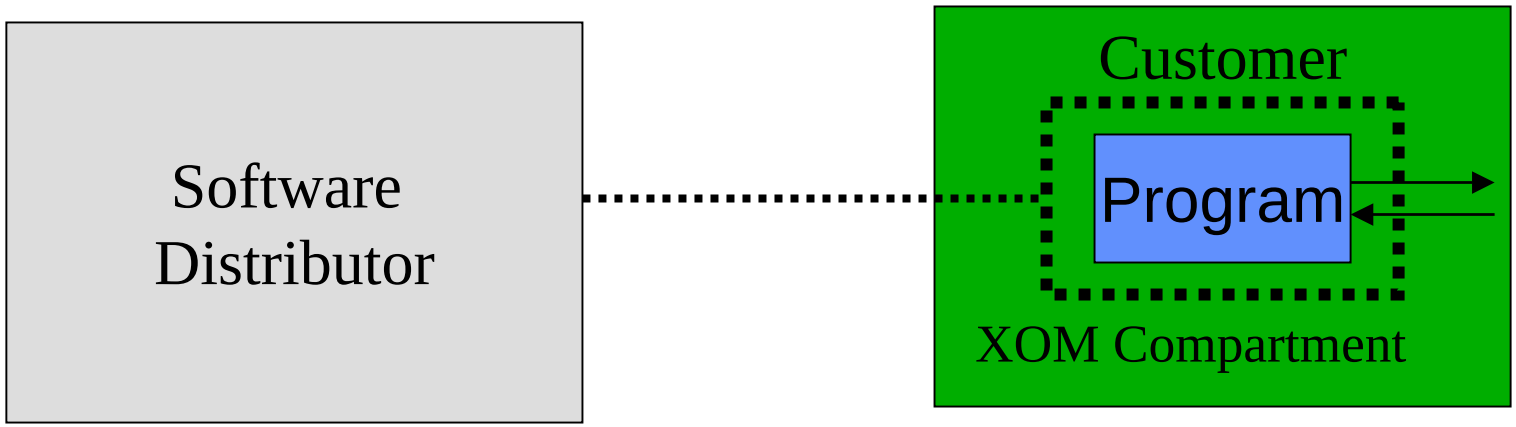
Architectural Support for Copy and Tamper Resistant Software

David Lie, Chandu Thekkath, Mark Mitchell,
Patrick Lincoln, Dan Boneh, John Mitchell and Mark Horowitz
Computer Systems Laboratory
Stanford University

XOM

- XOM: eXecute Only Memory
- Would like to support an environment where programs are protected from copying and tampering



- Use this to:
  - Prevent Hackers
  - Protect Intellectual Property
  - Combat Software Piracy

General Strategy

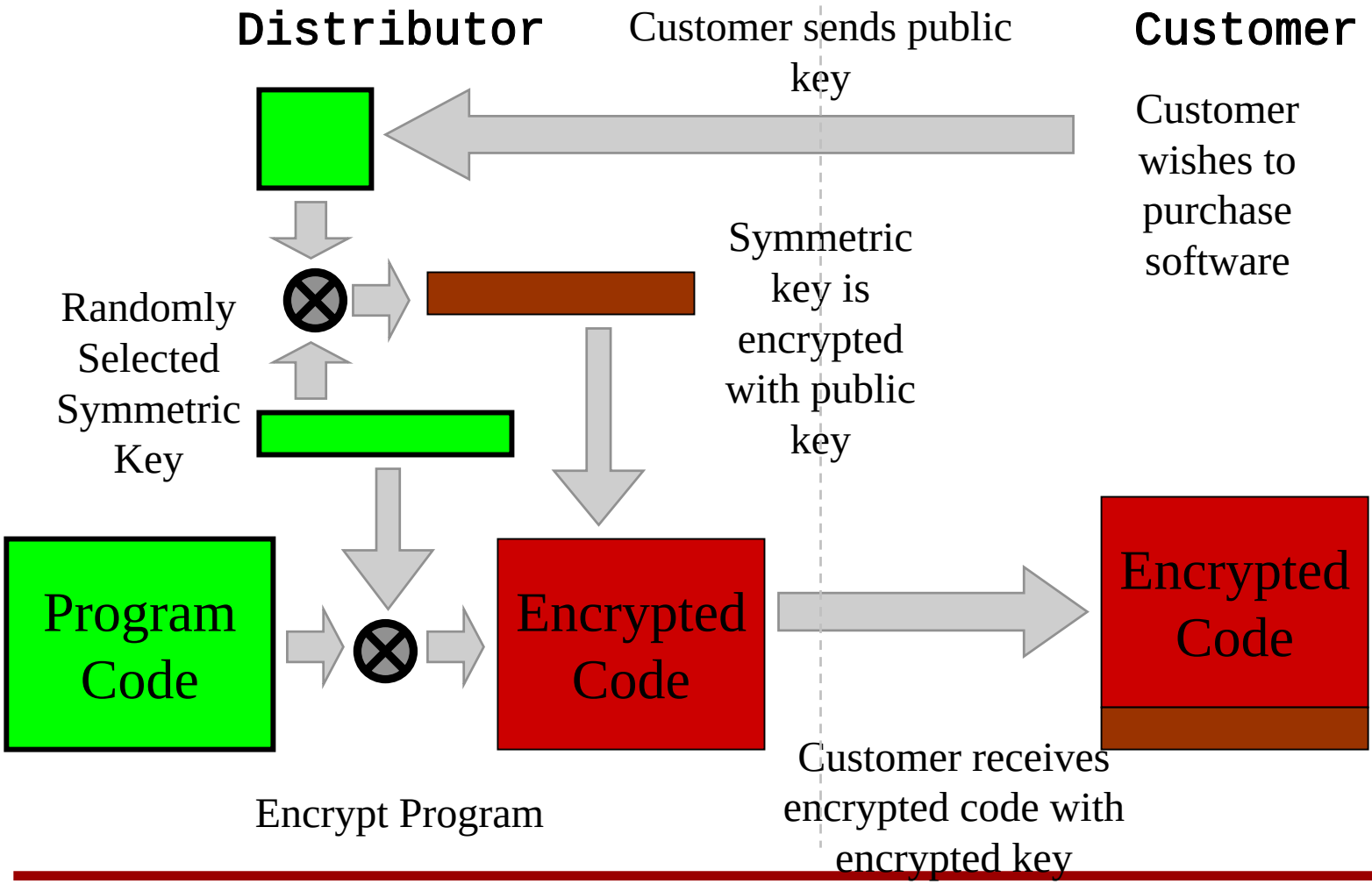
- Interfaces are suspect
  - Disk and memory are considered insecure
- On chip storage can be trusted
- How compartments are implemented
  - Data is always protected by some mechanism
  - With hardware tags when on chip
  - With crypto when off chip

Crypto Review

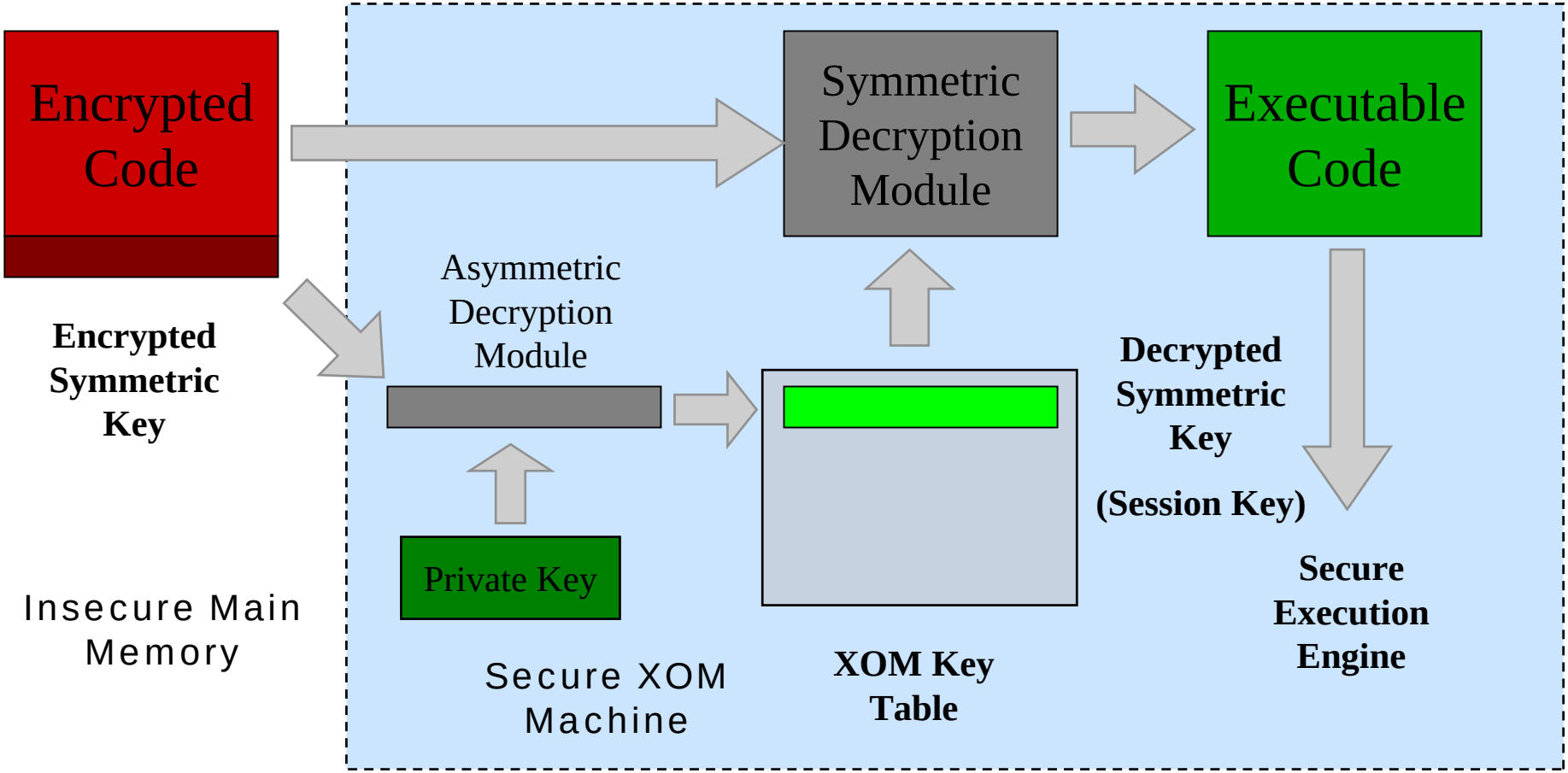
- Asymmetric Ciphers or public-key ciphers (RSA, El Gamal)
  - Pairs of keys
  - Public key and is used to encrypt data
  - Private key and is used to decrypt data
  - Are much slower than symmetric ciphers

- Symmetric Ciphers (3DES, Blowfish)
  - Single key used for encryption and decryption
  - Pretty fast when implemented in hardware
  - Advanced Encryption Standard ciphers will be even faster

Software Distribution Method



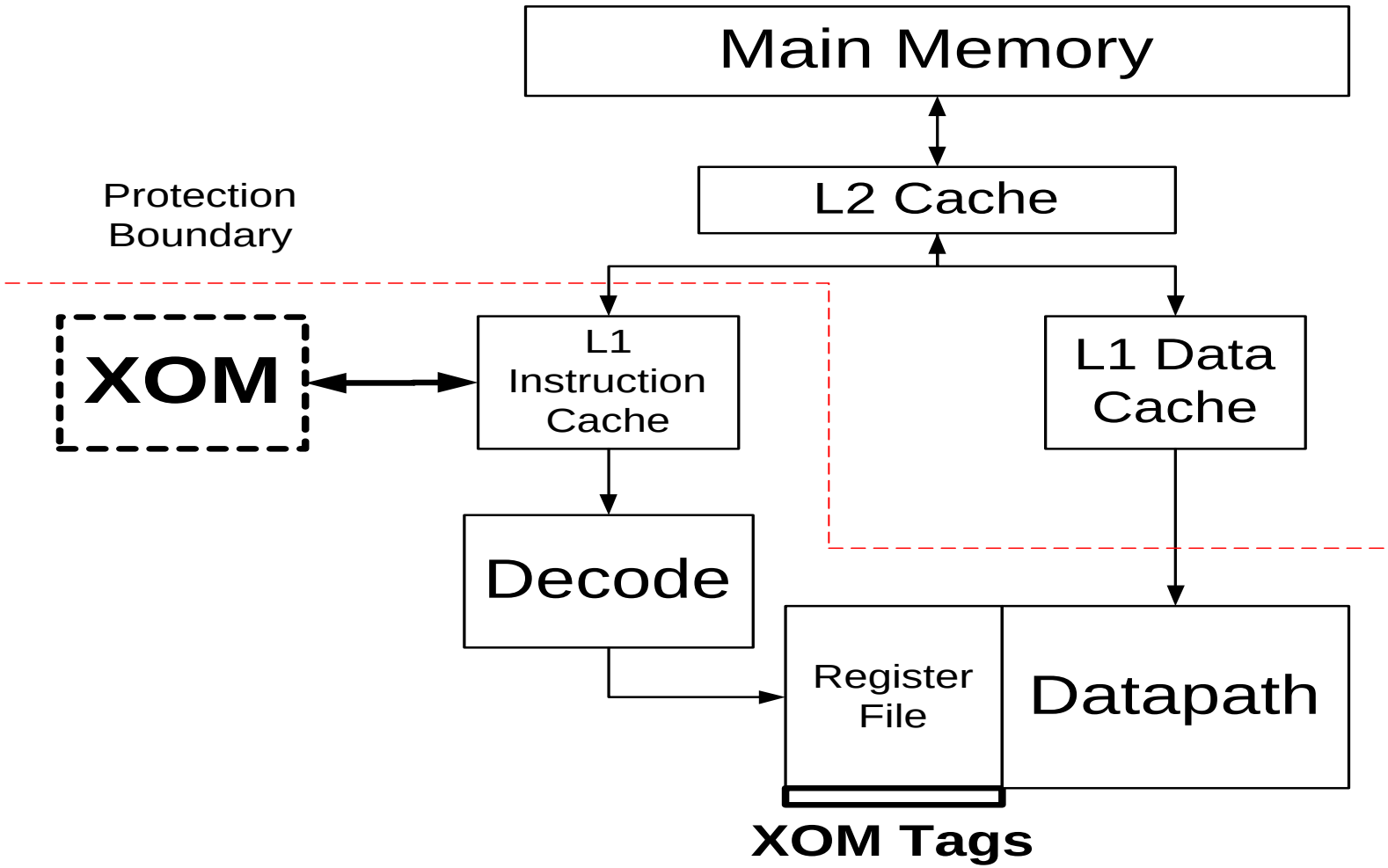
Loading Secure Code



Managing Data

- Simple hardware rule: a tag check on every access
- But the model is too rigid, applications cannot pass out data for other applications to read
- Have a special compartment with a tag ID of zero called the “null” compartment
- Special instructions are required to move data to and from null to a program’s private compartment
  - Allows for communication with principals outside of compartment

A Simple XOM Machine

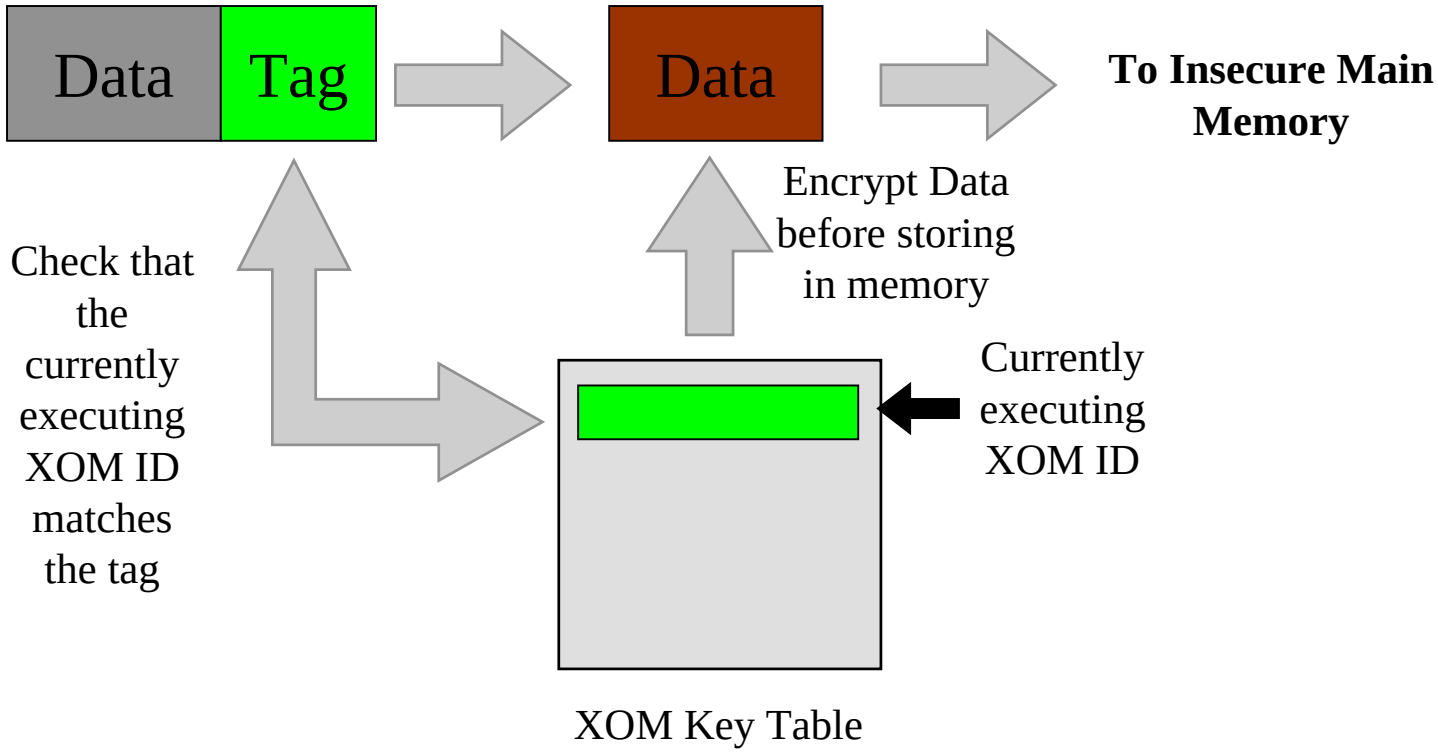


Two problems

- Memory is insecure
  - All sensitive program data must fit in registers
  - Too restrictive a programming model
- Programs can’t read or write data that doesn’t belong to them
  - But OS needs to do this when performing a context switch
- Use encryption to solve both problems
  - This was the same technique used to protect sensitive code

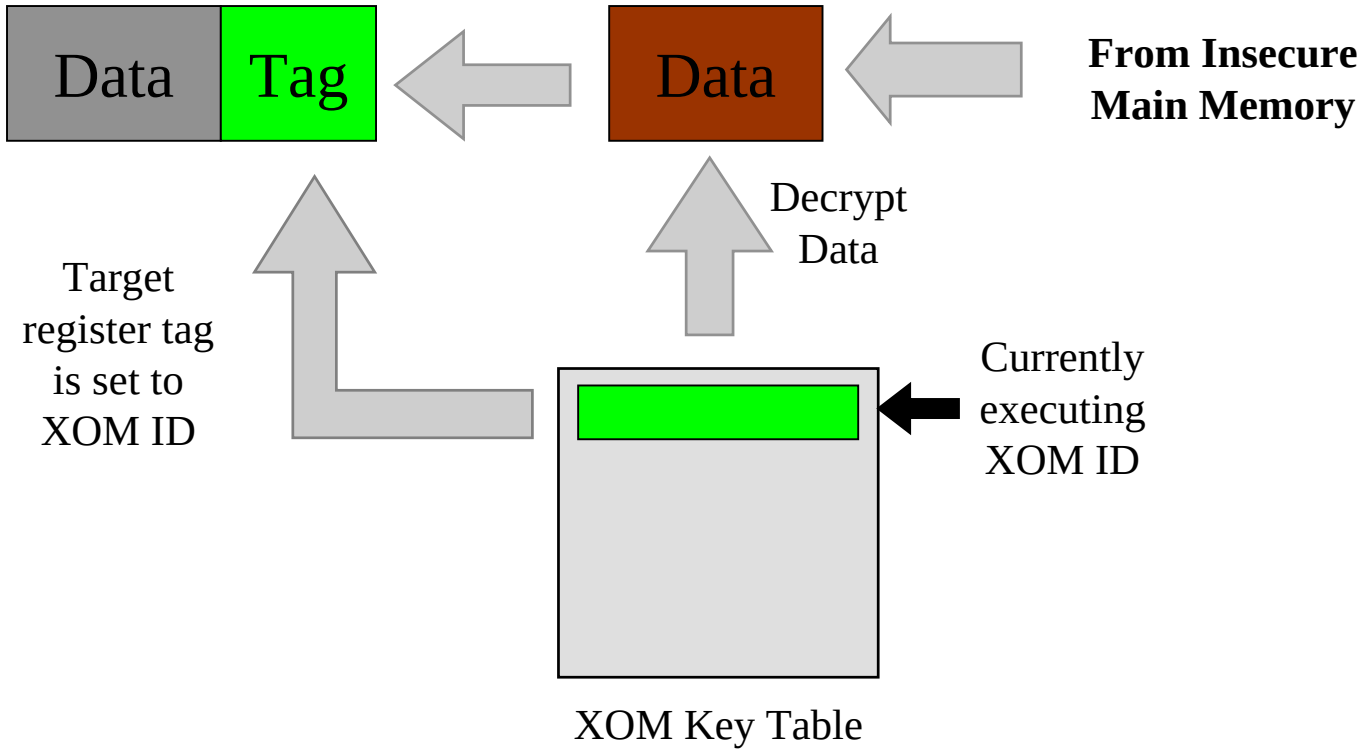
Supporting Main Memory

- *store_secure* instruction



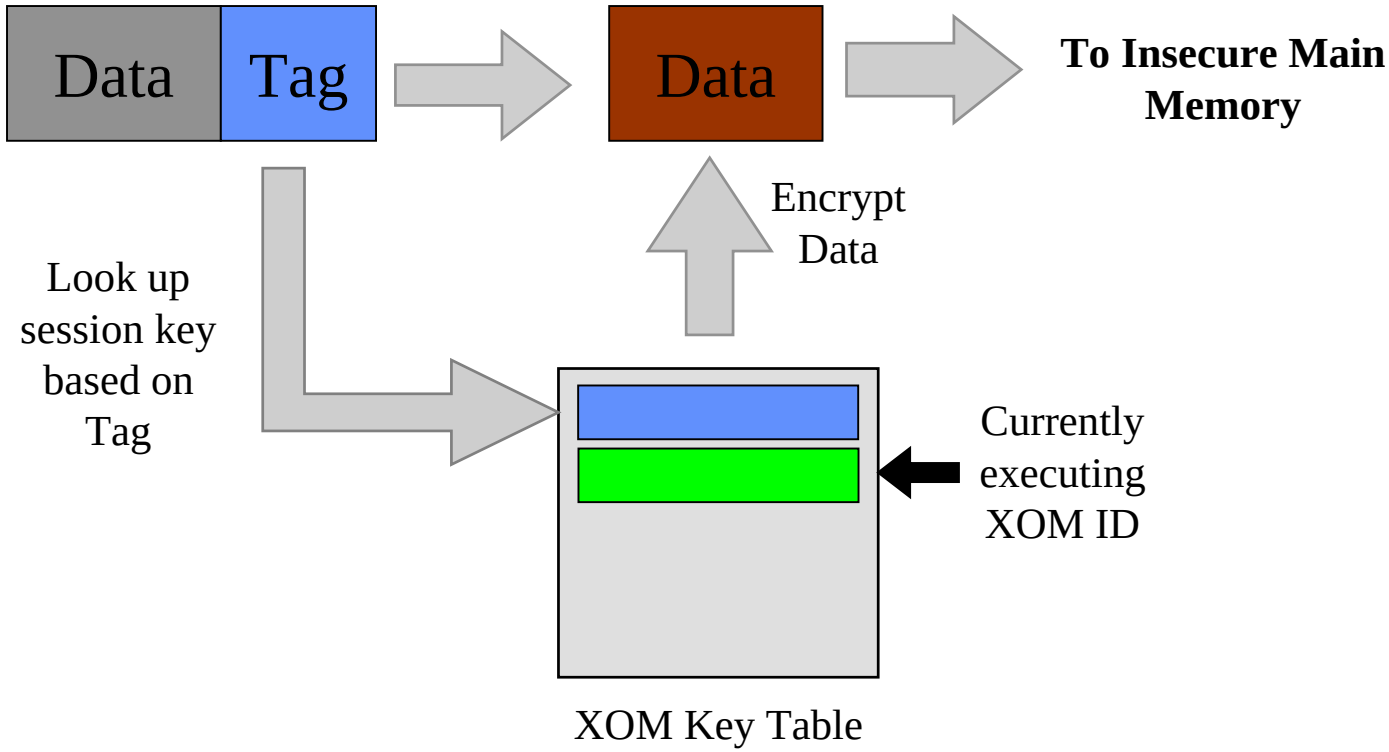
Supporting Main Memory

- *load_secure* instruction



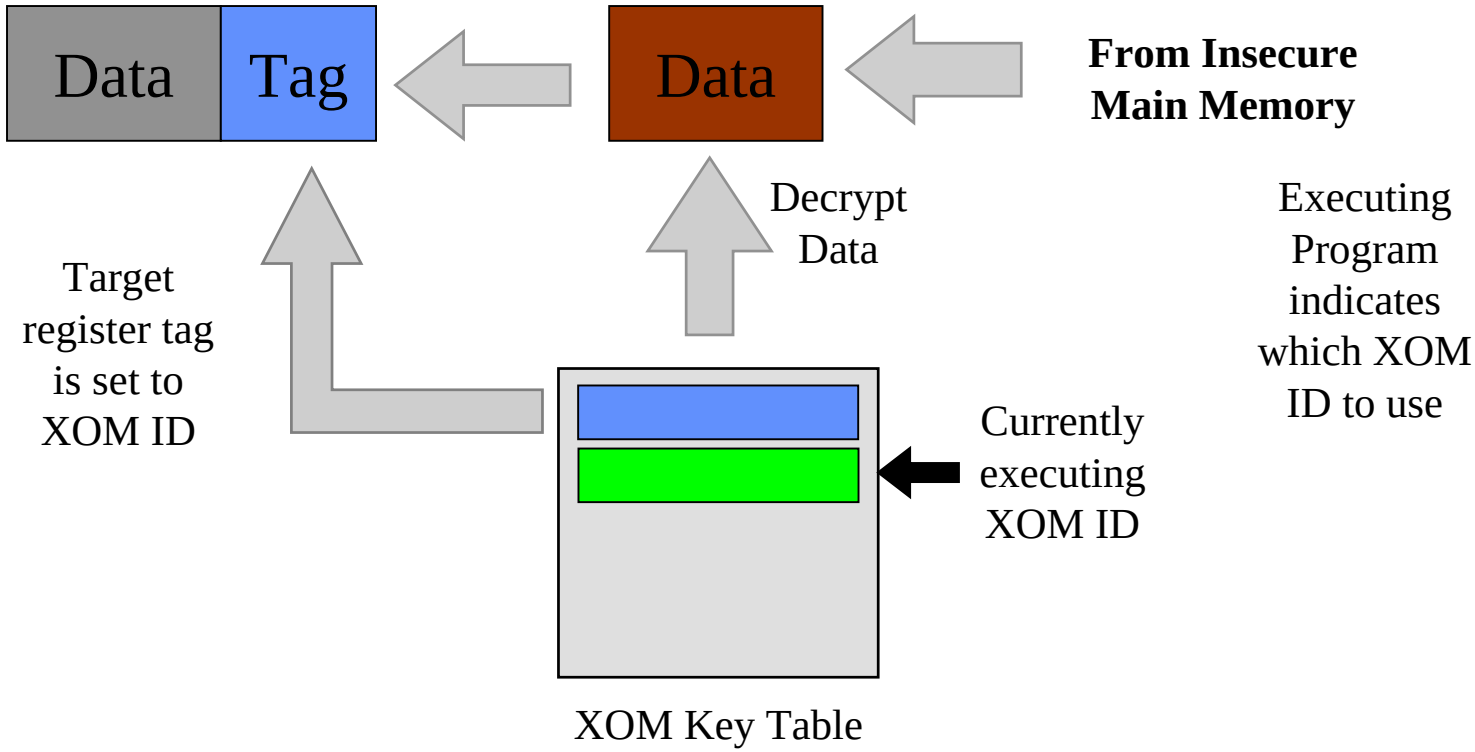
Supporting Interrupts

- *save_secure* instruction



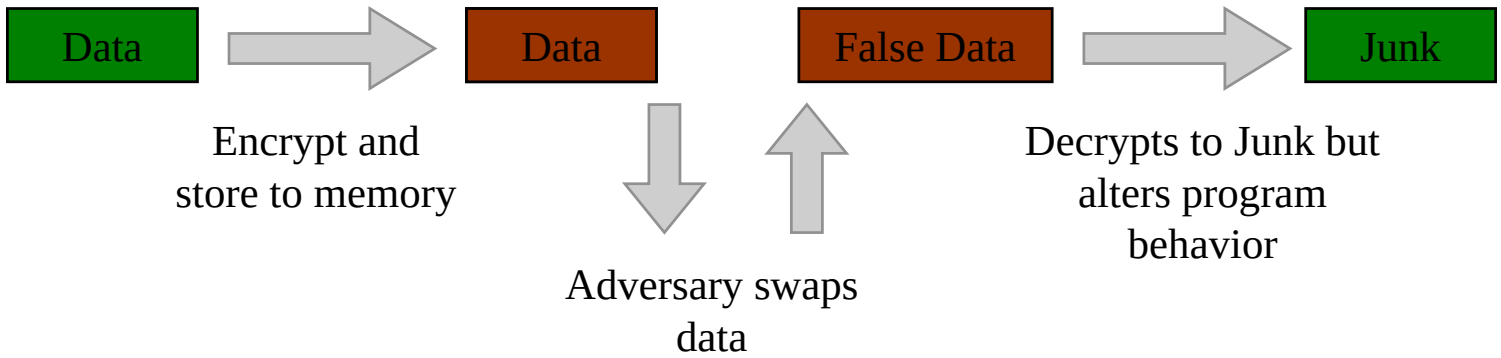
Supporting Interrupts

- *restore_secure* instruction



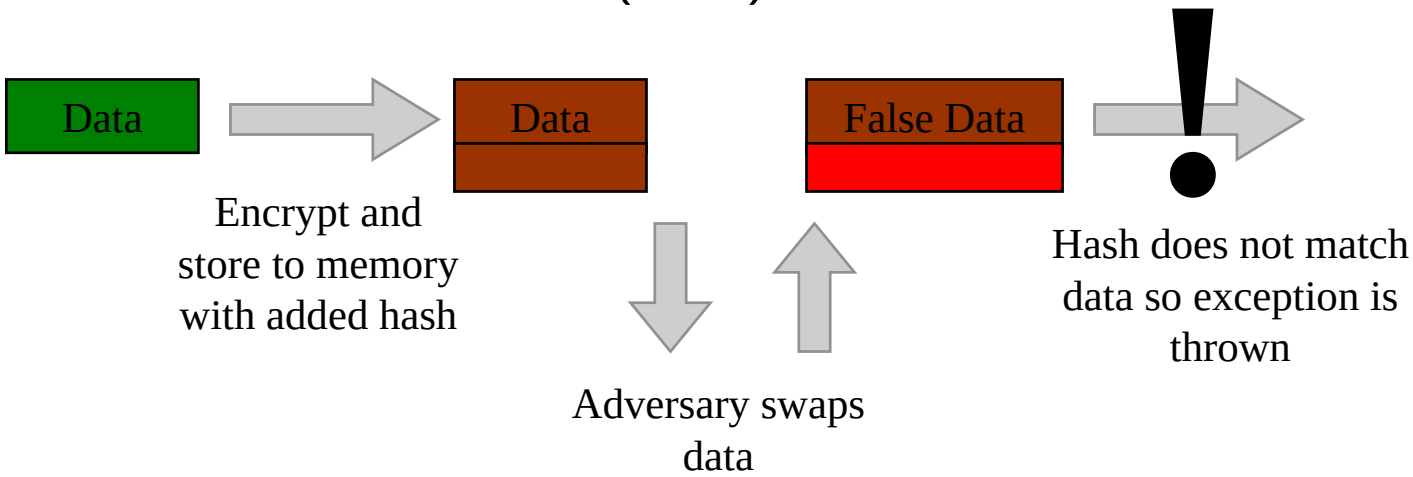
Spoofing Attacks

- Spoofing attack:
  - Adversary tries to substitute fake ciphertext to alter behavior
- Tags are able to catch spoofed attacks because tag ID changes
- Encryption alone is not sufficient for memory



Spoofing Prevention

- Solution is to add an integrity hash to the encryption
  - In cryptography terms, this is called a Message Authentication Code (MAC)

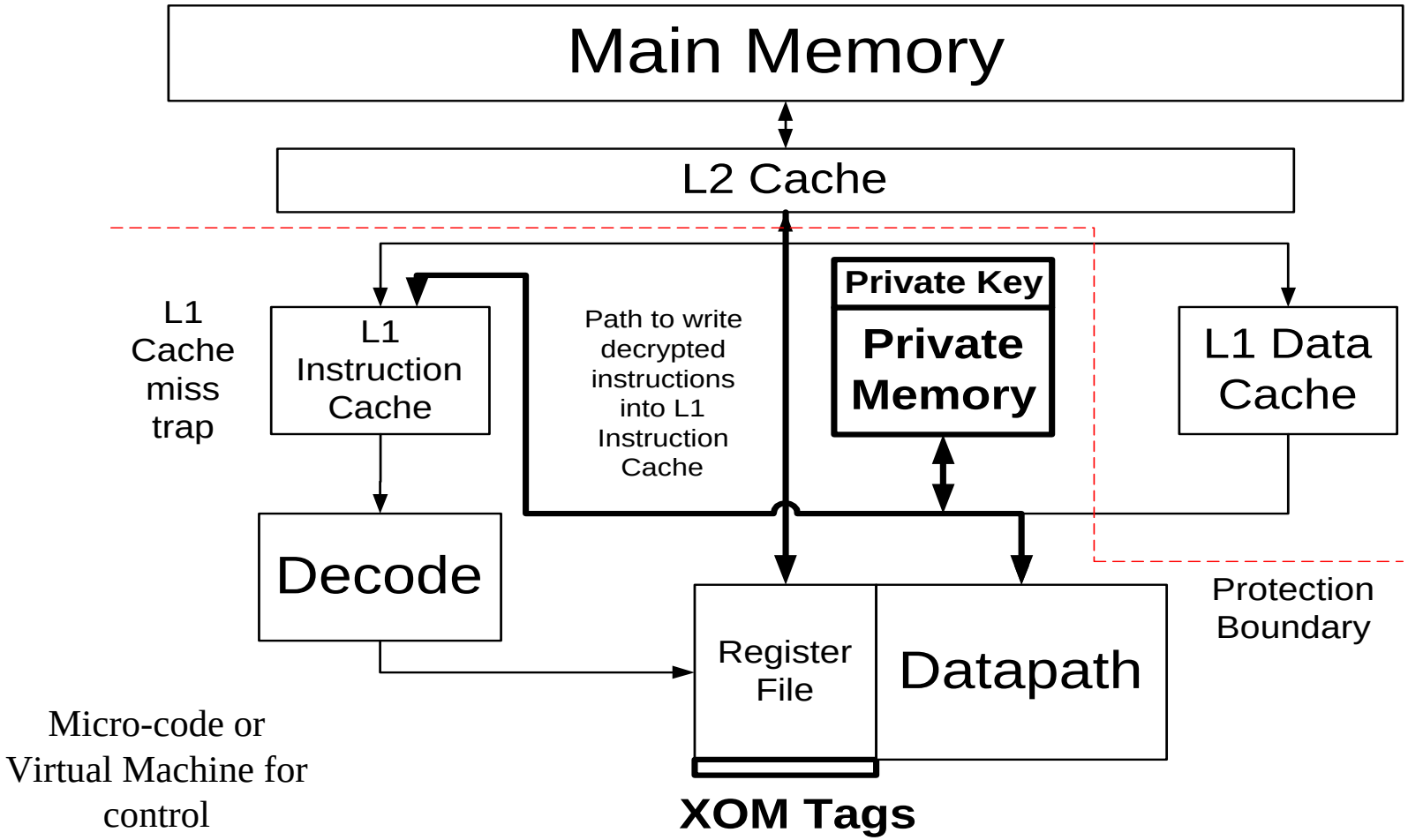


Splicing Attacks and Replay Attacks

- Splicing Attacks
  - Attacker moves valid data from one location to another location
  - Add position dependent hash:
    - Virtual Address for secure load/stores
    - Register number for secure save/restores

- Replay Attacks
  - Attack records and reuses old register and memory values
  - Add a regenerative key to Key Table that is used for save/restores
  - Use protected registers to protect memory values

Required Hardware



Performance Issues

- Performance hit is going to come from the cryptographic operations
  - XOM start-up
  - Instruction load path from memory
  - Data loads and stores to and from memory
  - Register saves and restores to and from memory

- The accesses to memory occur the most often

- We want to speed up the symmetric and hashing operations, as well as optimize access to memory

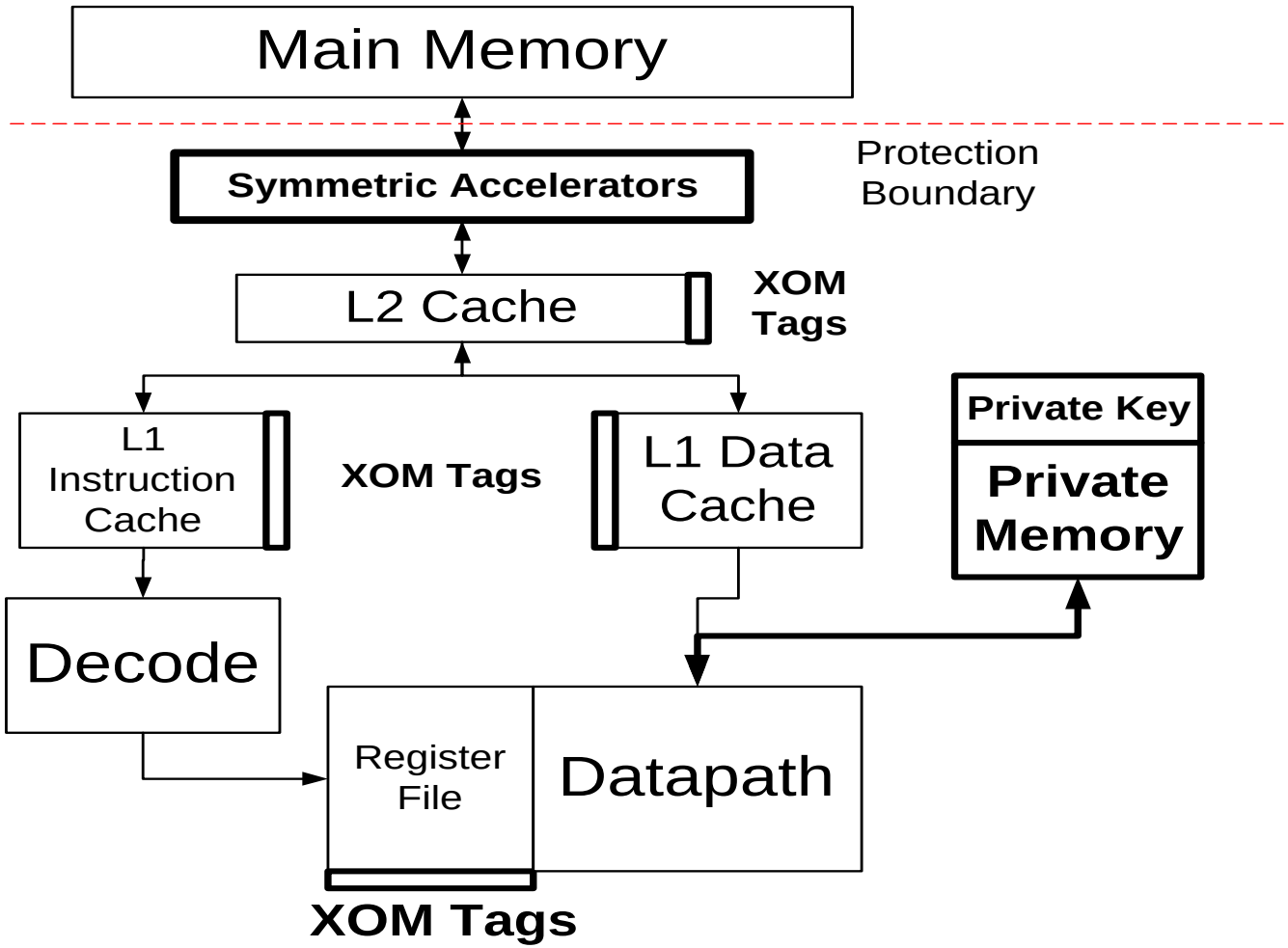
Additional Hardware

- Cache decrypted data
  - Add tags to caches
- Speed up symmetric operations
  - Add special symmetric cryptography hardware
- Speed up hash calculation
  - Select a fast hash calculation such as 128 bit CRC

Caching Values

- Caching reduces the number of cryptographic operations
- The size of each message is increased
- The granularity of ownership is increased
  - Need to add per word valid bits
  - Clear all valid bits when tag changes

Full XOM Machine



Summary

- Show how to implement compartments with architectural support
- Trust only the processor and assume memory and OS are insecure
- Use:
  - Data tagging on-chip
  - Crypto off-chip
- Required hardware is modest
  - Private Memory and Key
  - XOM Tags on registers
- Additional hardware can be added to improve performance
  - Symmetric hardware
  - XOM Tags in caches