A Key-centric Processor Architecture for Secure Computing

David Whelihan, Kate Thurmer, and Michael Vai

HOST 2016

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

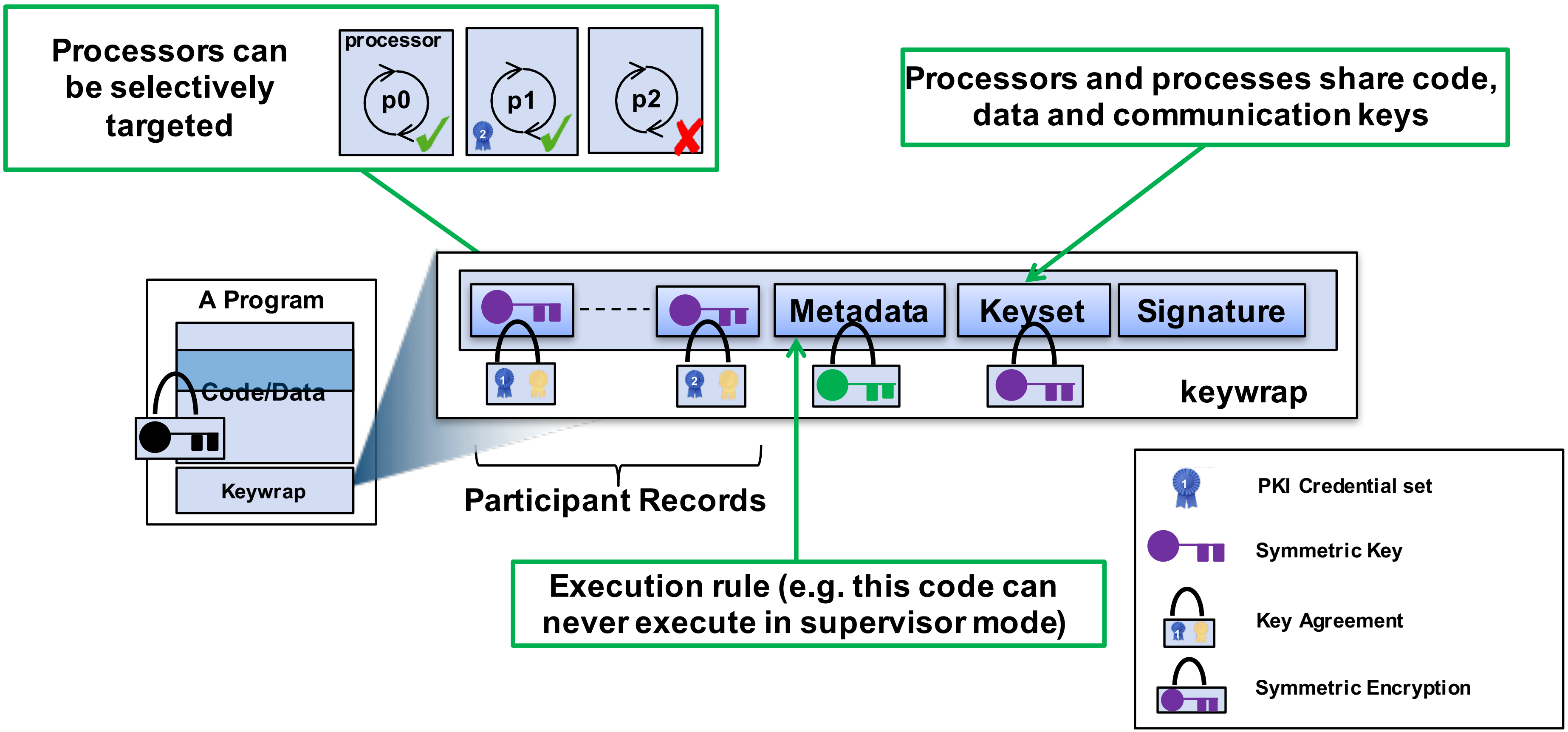
This material is based upon work supported by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Assistant Secretary of Defense for Research and Engineering.

Distributed Computing, Distributed Threat

- **MIT LL is building a synthesizable Sparc v8 compatible processor core that embeds**
  - Stable-key Physical Unclonable Function (PUF)
  - Deeply embedded key management
  - Hardware-enforced mandatory code and data decryption
- **Fosters the creation of trusted groups of computing devices**
  - Dynamic keying

Provides a foundation for holistic data protection, embedding security and encryption technology deeply inside of the processor architecture

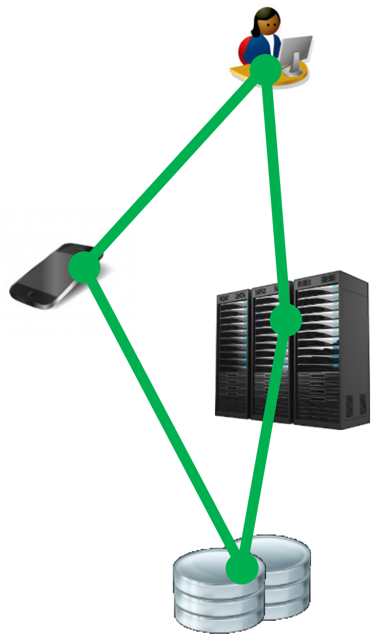
Critical Enabling Technology



PKI-enabled key management locks *keysets* that encrypt code, data and communication to collections of processors

Benefits

- **Features**
  - **Mandatory de/encryption of code and data**
  - **Encrypted, relocatable libraries locked to specific processors**
  - **Security features enabled by the key management system**
- **Enabling**
  - **Trusted networks of cooperating processors**
  - **Separately encrypted functions and libraries**
  - **Progressive security gradations**

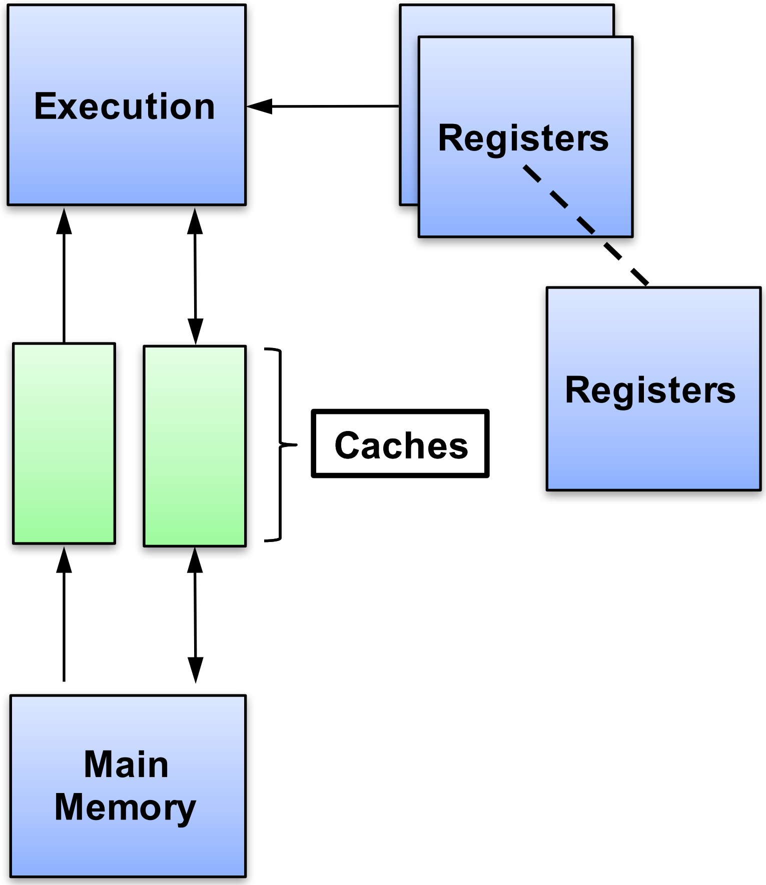


This processor is a vital piece of a distributed and cooperative processing capability with deeply embedded data and code protection

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

How It Works

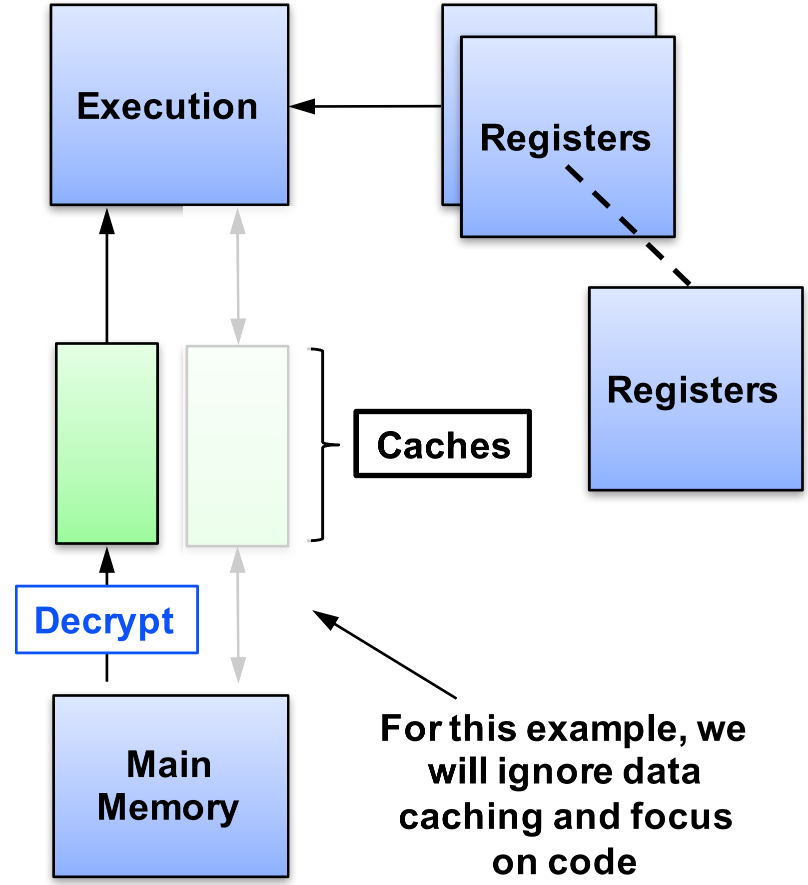
- **Simplified Sparc v8 microprocessor**
  - The *execution* unit performs math on data stored in registers
  - Code and data are pulled from fast memory *caches*
  - Caches fetch code and data from slower, but much larger *main memory*



The Sparc architecture specifies many register *windows*. Programs switch in and out of windows when they need to perform a new operation

How It Works

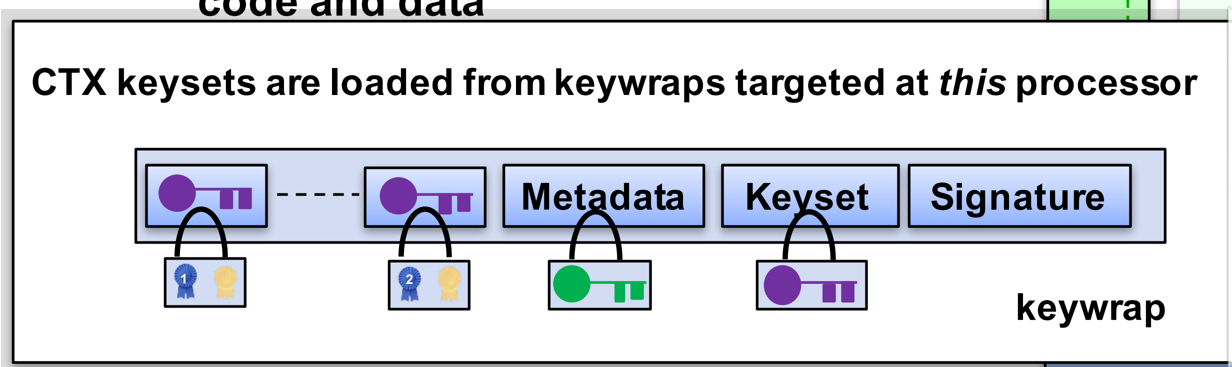
- Simplified Sparc v8 microprocessor
  - The *execution* unit performs math on data stored in registers
  - Code and data are pulled from fast memory *caches*
  - Caches fetch code and data from slower, but much larger *main memory*

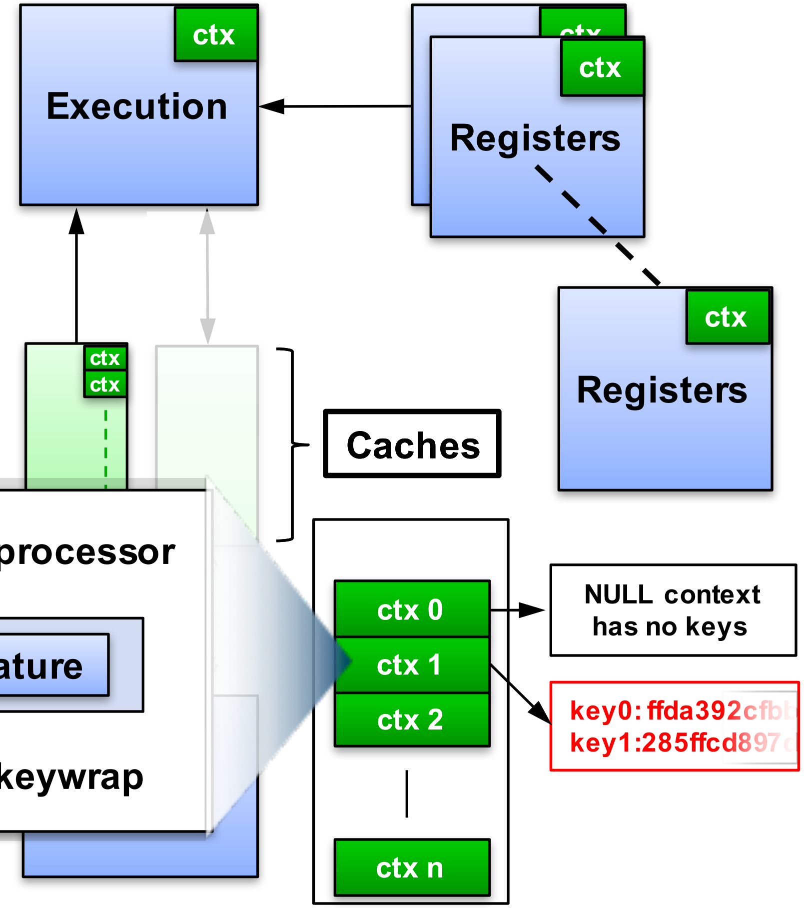


Run-time decryption and encryption is inserted into the code and data paths

How It Works

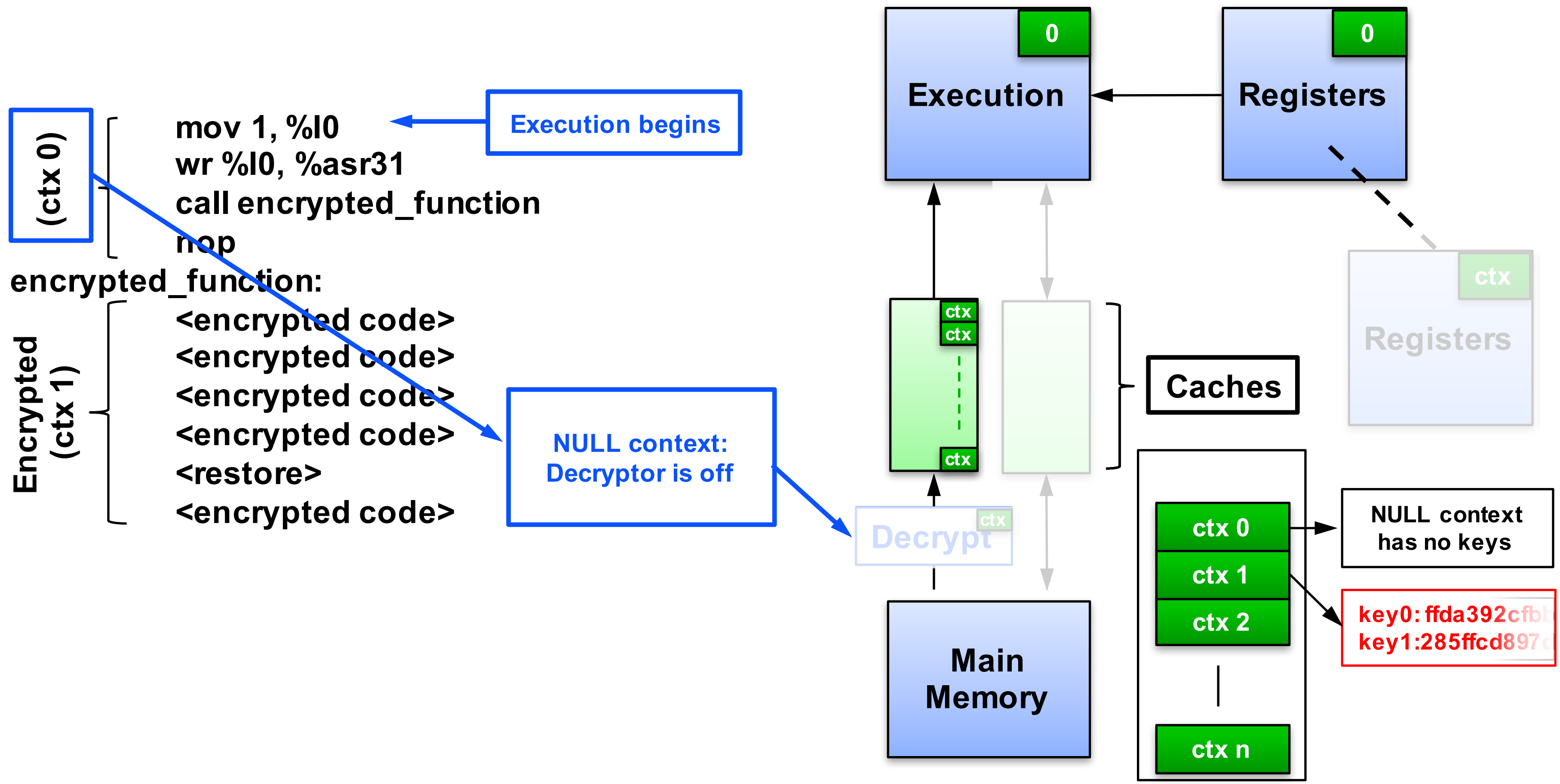
- **Encrypted context (ctx)**
  - An encrypted context is a set of keys bound to a sequence of instructions and its register state
  - The current ctx affects:
    - What keys are used to decrypt code and data





Encrypted contexts are loaded from keywraps

How It Works



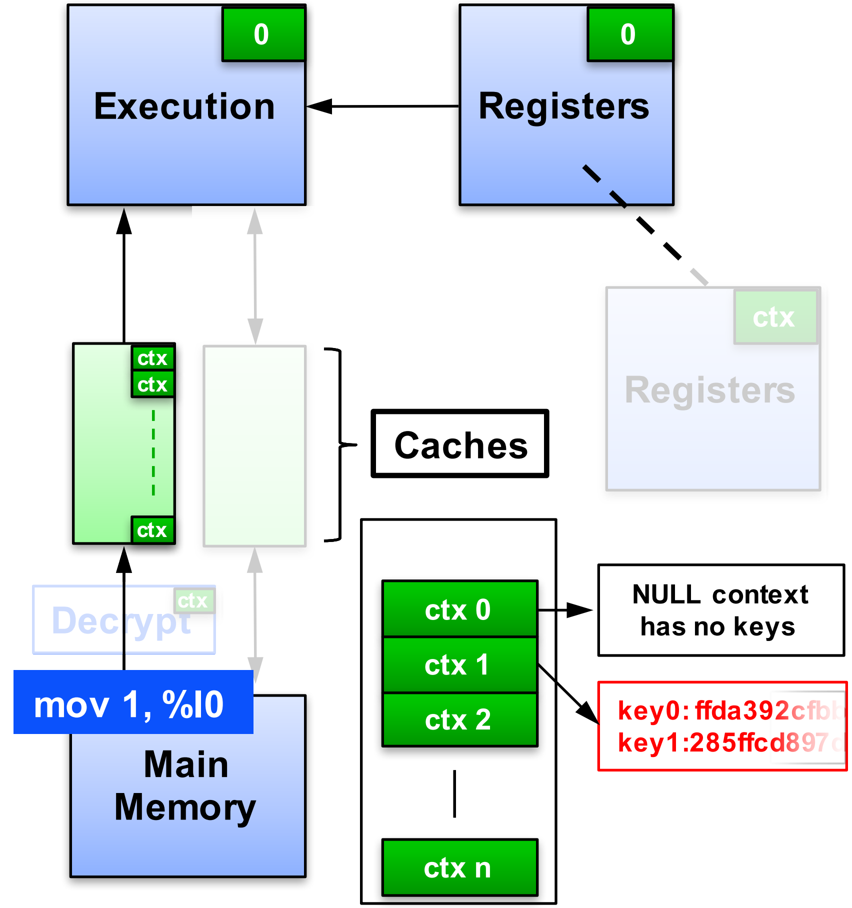
In ctx 0, the NULL context, the decryptor is turned off

How It Works

(ctx 0)

```
        mov 1, %l0
        wr %l0, %asr31
        call encrypted_function
        nop
encrypted_function:
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <restore>
        <encrypted code>
```

Encrypted (ctx 1)



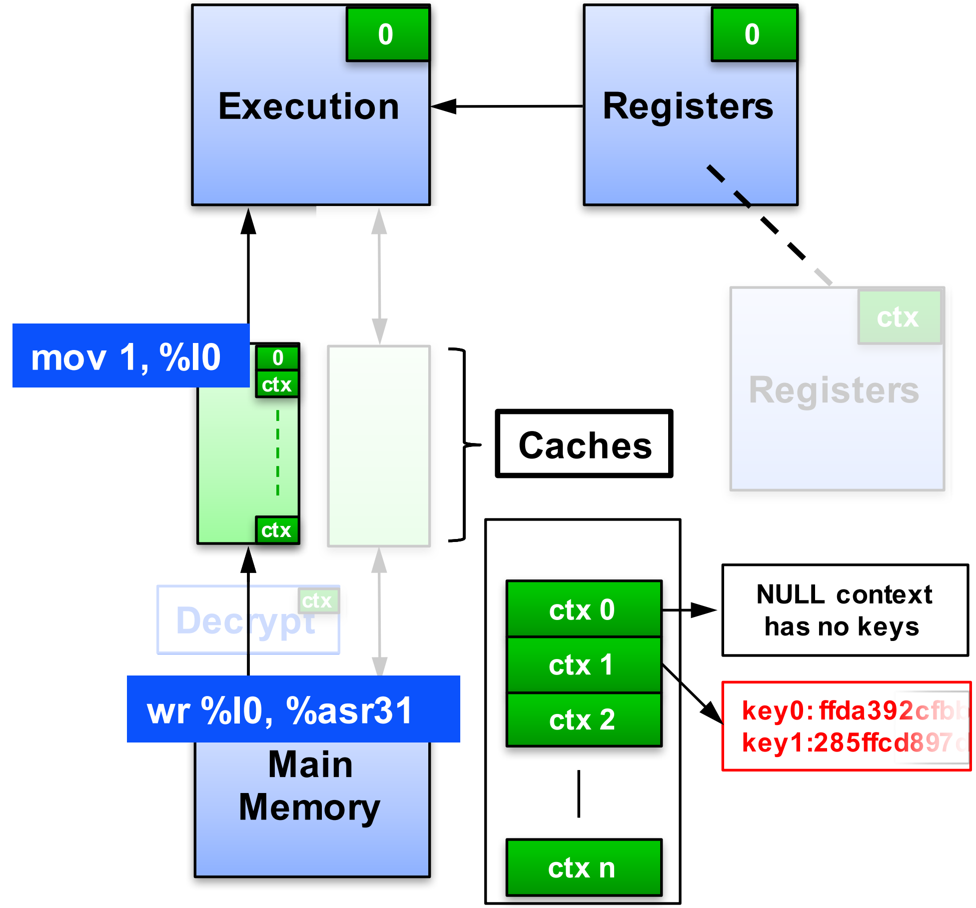
Instructions are executed normally by fetching from main memory, to cache, and into the execution pipeline

How It Works

(ctx 0)

```
        mov 1, %l0
        wr %l0, %asr31
        call encrypted_function
        nop
encrypted_function:
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <restore>
        <encrypted code>
```

Encrypted (ctx 1)



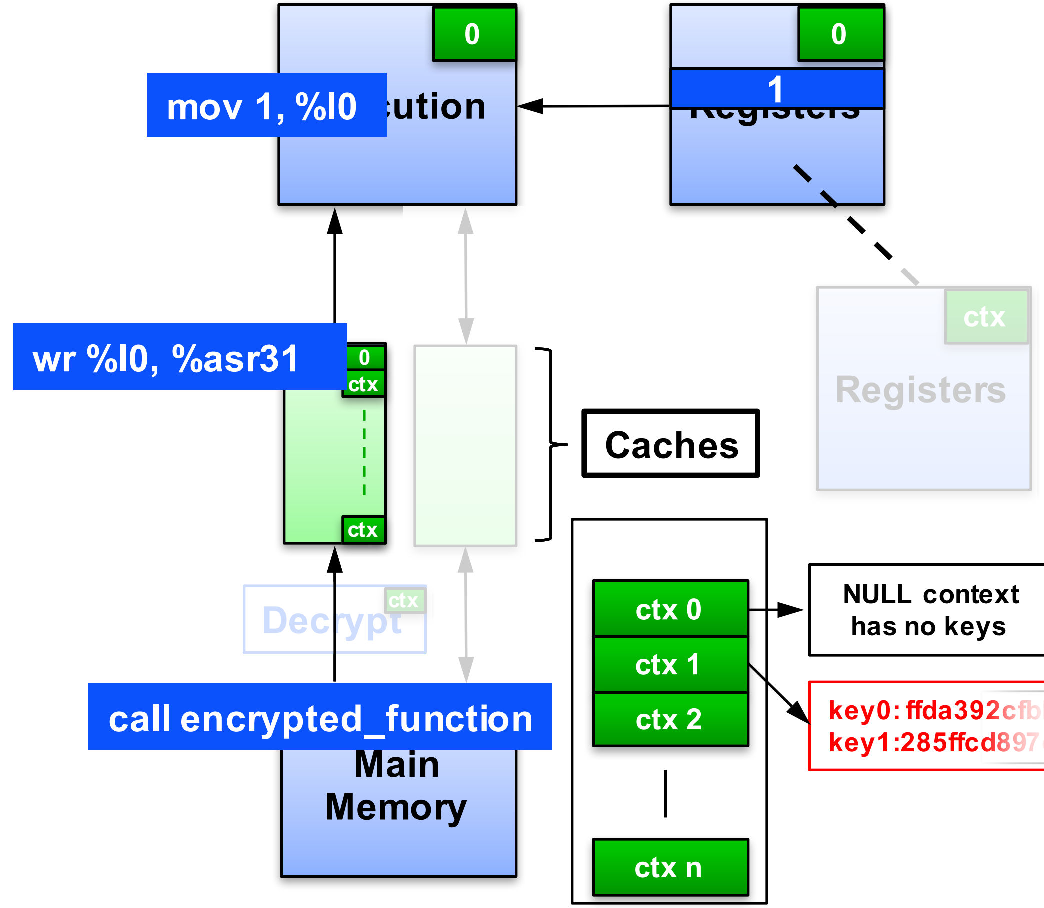
Instructions are executed normally by fetching from main memory, to cache, and into the execution pipeline

How It Works

(ctx 0)

```
        mov 1, %l0
        wr %l0, %asr31
        call encrypted_function
        nop
encrypted_function:
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <restore>
        <encrypted code>
```

Encrypted (ctx 1)



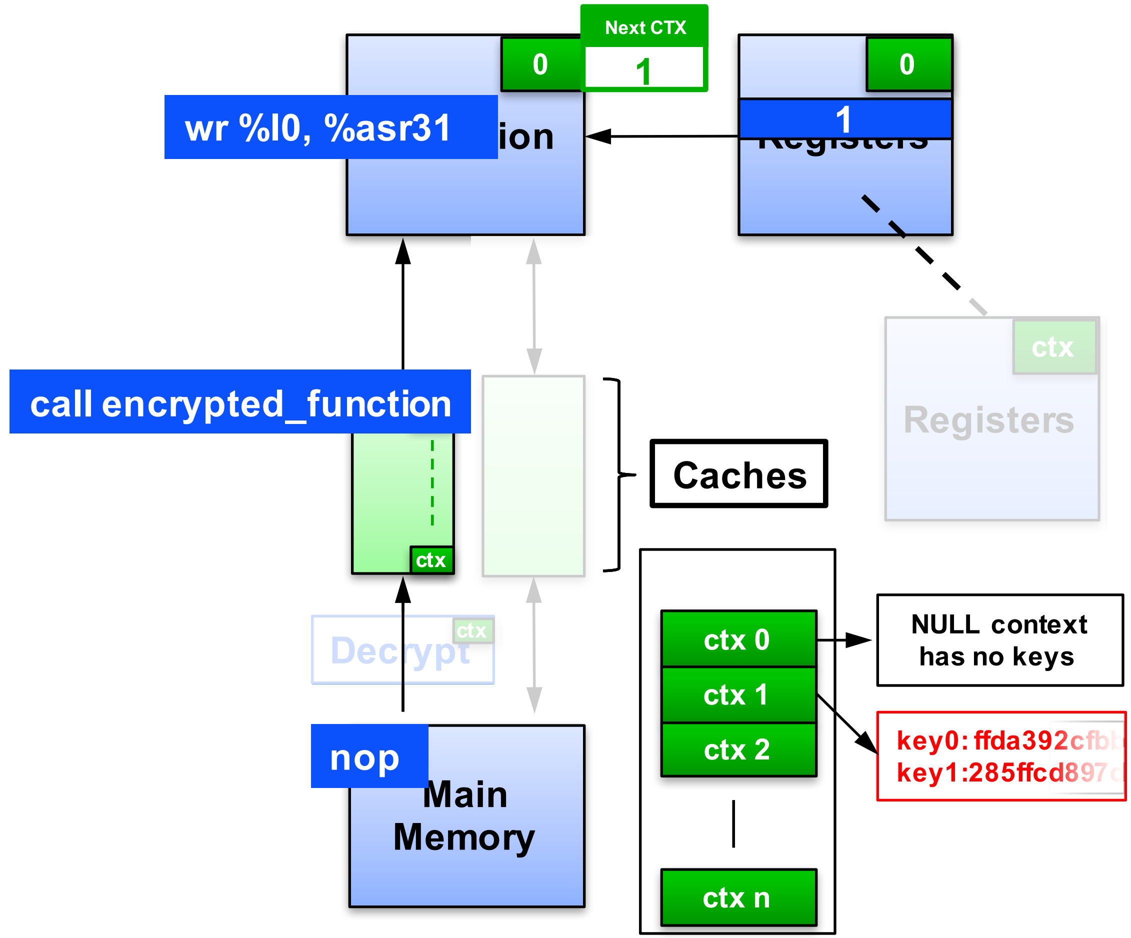
The currently executing instruction places data into one register window, which is bound to the NULL context

How It Works

(ctx 0)

```
        mov 1, %l0
        wr %l0, %asr31
        call encrypted_function
        nop
encrypted_function:
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <restore>
        <encrypted code>
```

Encrypted (ctx 1)



Writing a special register instructs the process or that the next "call" instruction will shift contexts to ctx 1

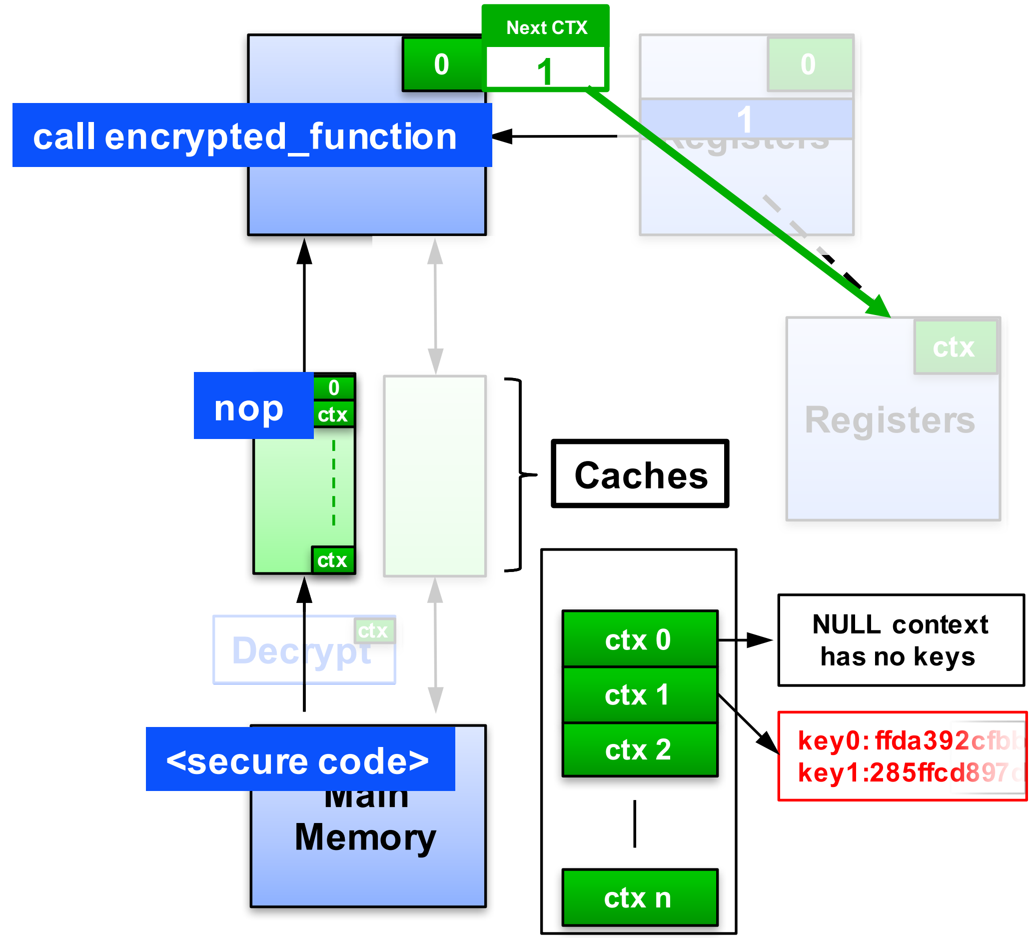
LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

How It Works

(ctx 0)

```
        mov 1, %l0
        wr %l0, %asr31
        call encrypted_function
        nop
encrypted_function:
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <restore>
        <encrypted code>
```

Encrypted (ctx 1)



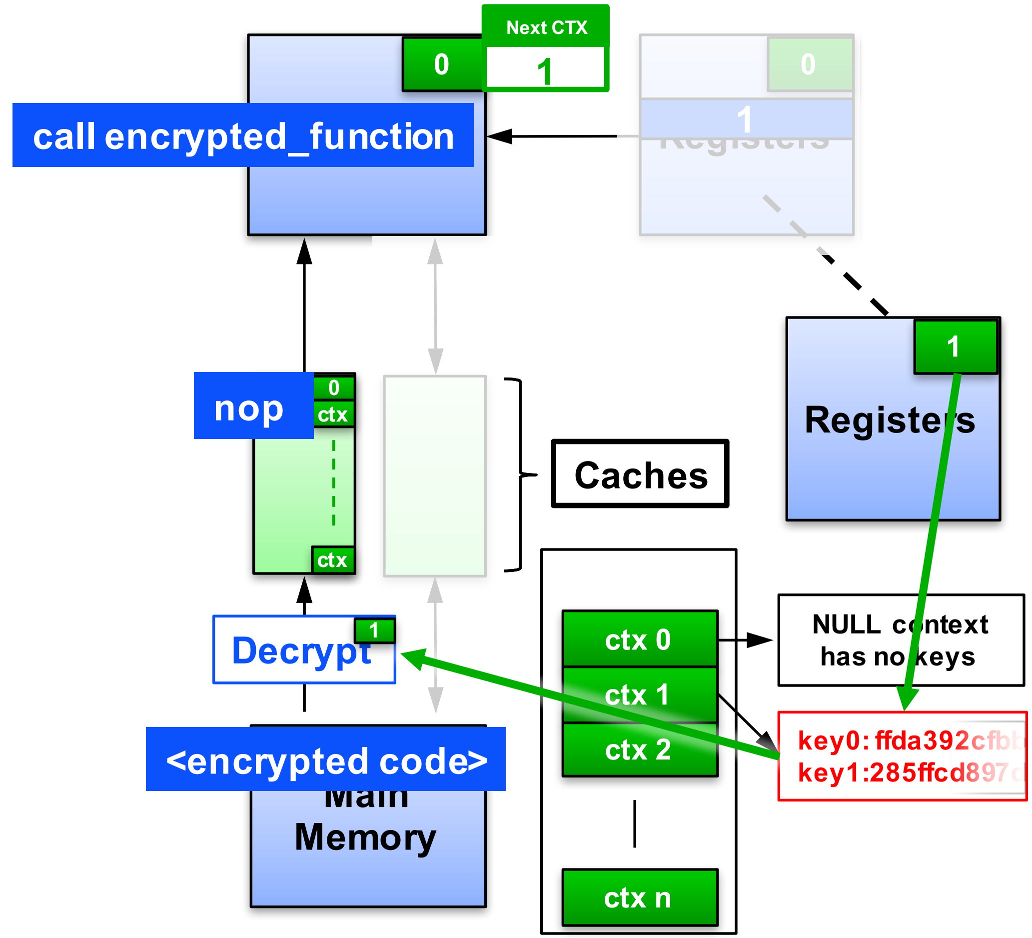
The call shifts the register window, hiding the callers state from the new code

How It Works

(ctx 0)

```
        mov 1, %l0
        wr %l0, %asr31
        call encrypted_function
        nop
encrypted_function:
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <restore>
        <encrypted code>
```

Encrypted (ctx 1)



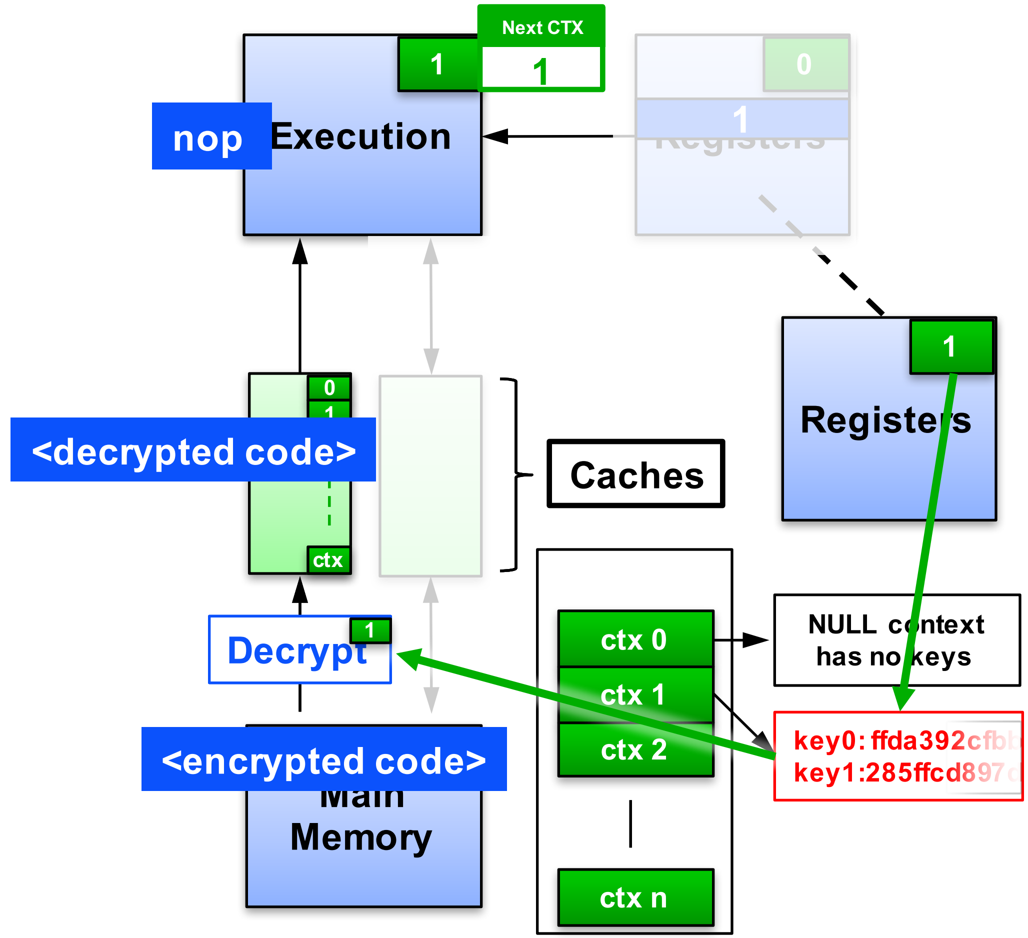
The new window is bound to ctx 1, and therefore shifts the keyset to activate the instruction decryptor

How It Works

(ctx 0)

```
        mov 1, %l0
        wr %l0, %asr31
        call encrypted_function
        nop
encrypted_function:
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <restore>
        <encrypted code>
```

Encrypted (ctx 1)



The called code is decrypted

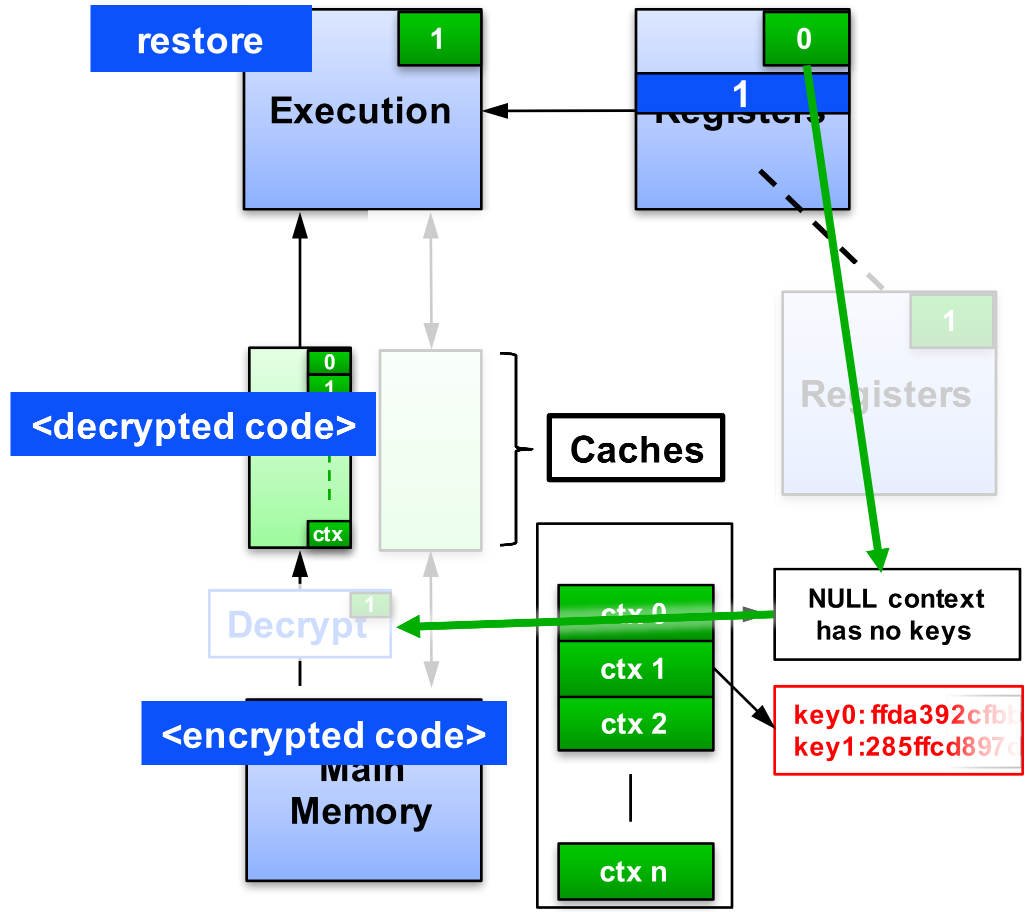
LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

How It Works

(ctx 0)

```
        mov 1, %l0
        wr %l0, %asr31
        call encrypted_function
        nop
encrypted_function:
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <restore>
        <encrypted code>
```

Encrypted (ctx 1)



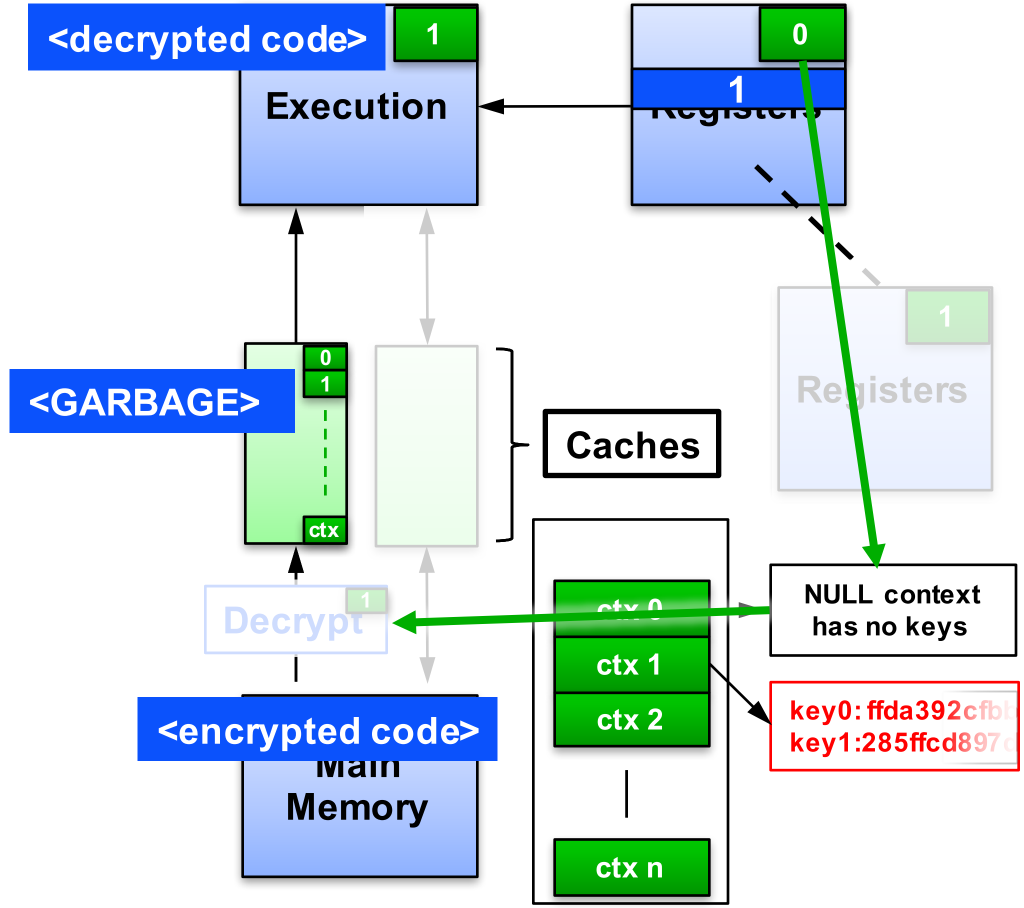
If the encrypted context attempts to access the caller's state (in the other window) by executing a "restore" instruction, the window shifts…

How It Works

(ctx 0)

```
        mov 1, %l0
        wr %l0, %asr31
        call encrypted_function
        nop
encrypted_function:
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <encrypted code>
        <restore>
        <encrypted code>
```

Encrypted (ctx 1)



…and forcibly changes the ctx back to 0, and thus turning off decryption of the code, resulting in a garbage code fetch

Processor Features

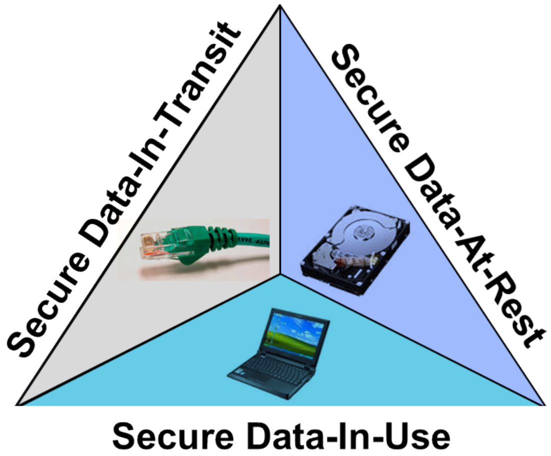
- Fully synthesizable System-On-Chip
- High-assurance Suite-B key management
- Tightly coupled but differently encrypted code streams
- Encrypted, relocatable libraries locked to specific instances of the processor
- High-speed decryption of code and data
  – For AES-128: Little to no performance hit (XOR in the data path)
  – For AES-256: 2 cycles of latency at the start of a missed instruction stream
  – No penalty for cache hits

Vulnerabilities (not exhaustive)

- The processor is not side-channel resistant (currently by choice)
  - Differential Power Analysis
  - Cache timing side-channel
- The data-cache scheme involves changing data under an XOR mask
- Code and data streams currently have no integrity protection
  - Code and data can be read in from AES-GCM encrypted flows, but they are AES-CTR mode encrypted when executing

Summary

- We show a novel embedded processor that protects data-in-use by making cryptographic operations intrinsic to processor execution
- The processor enables:
  - Progressive security gradations
  - Security level specified by the application writer
  - Full encryption and masking of code and data
  - Different code/data keys within a single code stream
  - Relocatable encrypted libraries



This work builds toward a holistic approach to data protection that considers the entire data life-cycle