Cross-VM Side Channels and Their Use to Extract Private Keys

Yinqian Zhang (UNC-Chapel Hill)

Ari Juels (RSA Labs)

Michael K. Reiter (UNC-Chapel Hill)

Thomas Ristenpart (U Wisconsin-Madison)

Motivation

Bromium™

VMware View

amazon web services™

rackspace

QubesOS

CiTRIX XenDesktop

Windows Azure

openstack™ CLOUD SOFTWARE

Google Compute Engine

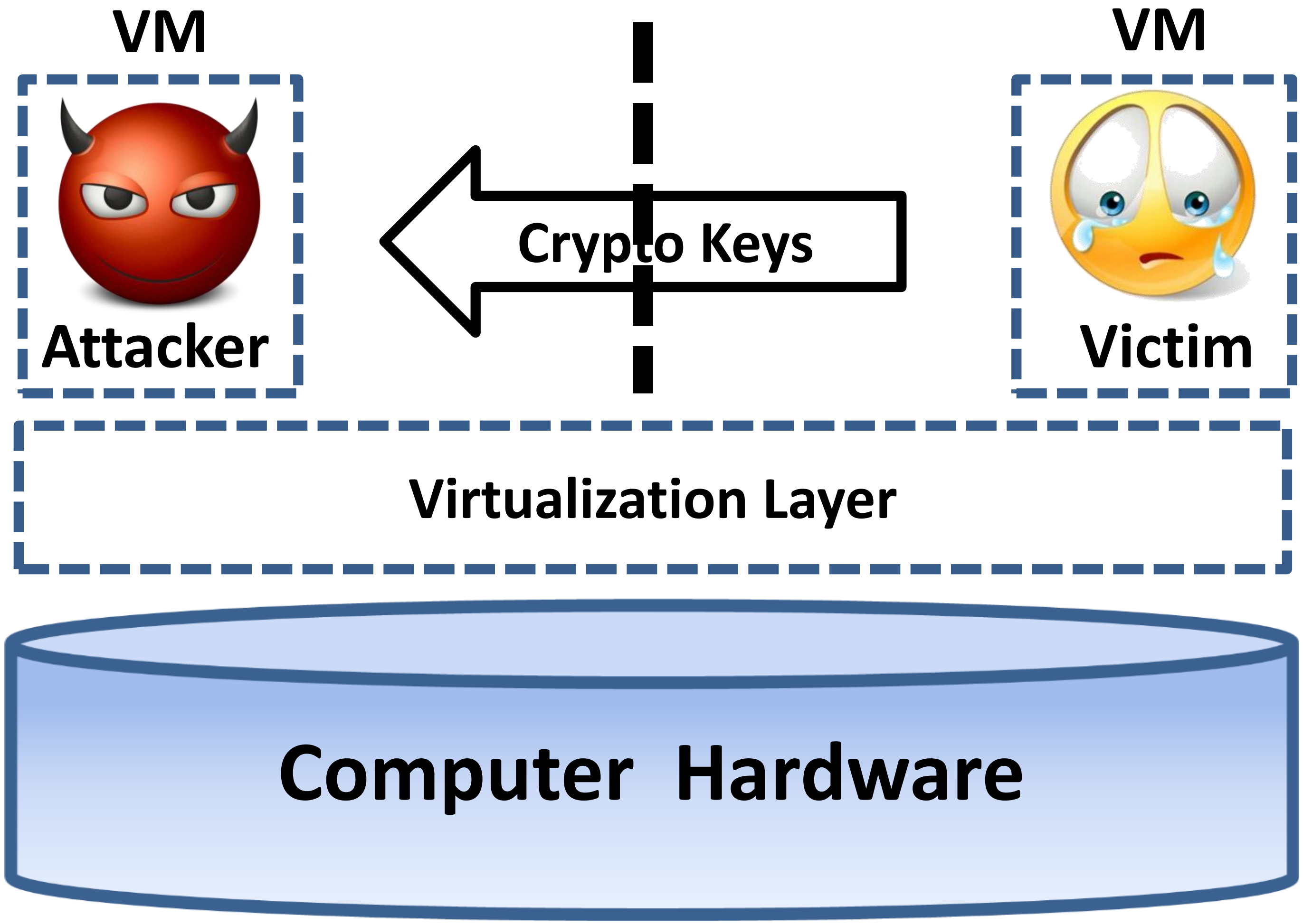
Microsoft® Hyper-V™

Xen™

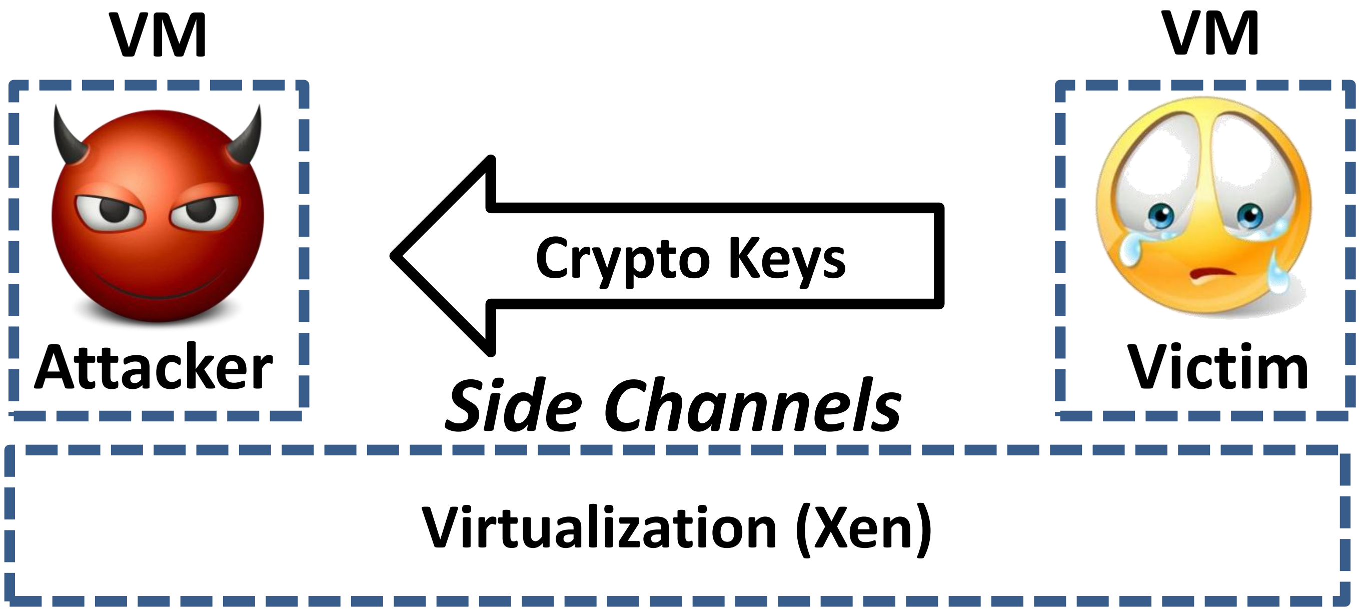
vmware®

KVM

Security Isolation by Virtualization



Access-Driven Cache Timing Channel



An open problem:

Are cryptographic side channel attacks possible in virtualization environment?

Related Work

Publication	Multi-Core	Virtualization	w/o SMT	Target
Percival 2005	✗	✗	✗	RSA
Osvik et al. 2006	✗	✗	✗	AES
Neve et al. 2006	✗	✗	✓	AES
Aciicmez 2007	✗	✗	✗	RSA
Aciicmez et al. 2010	✗	✗	✗	DSA
Bangerter 2011	✗	✗	✓	AES

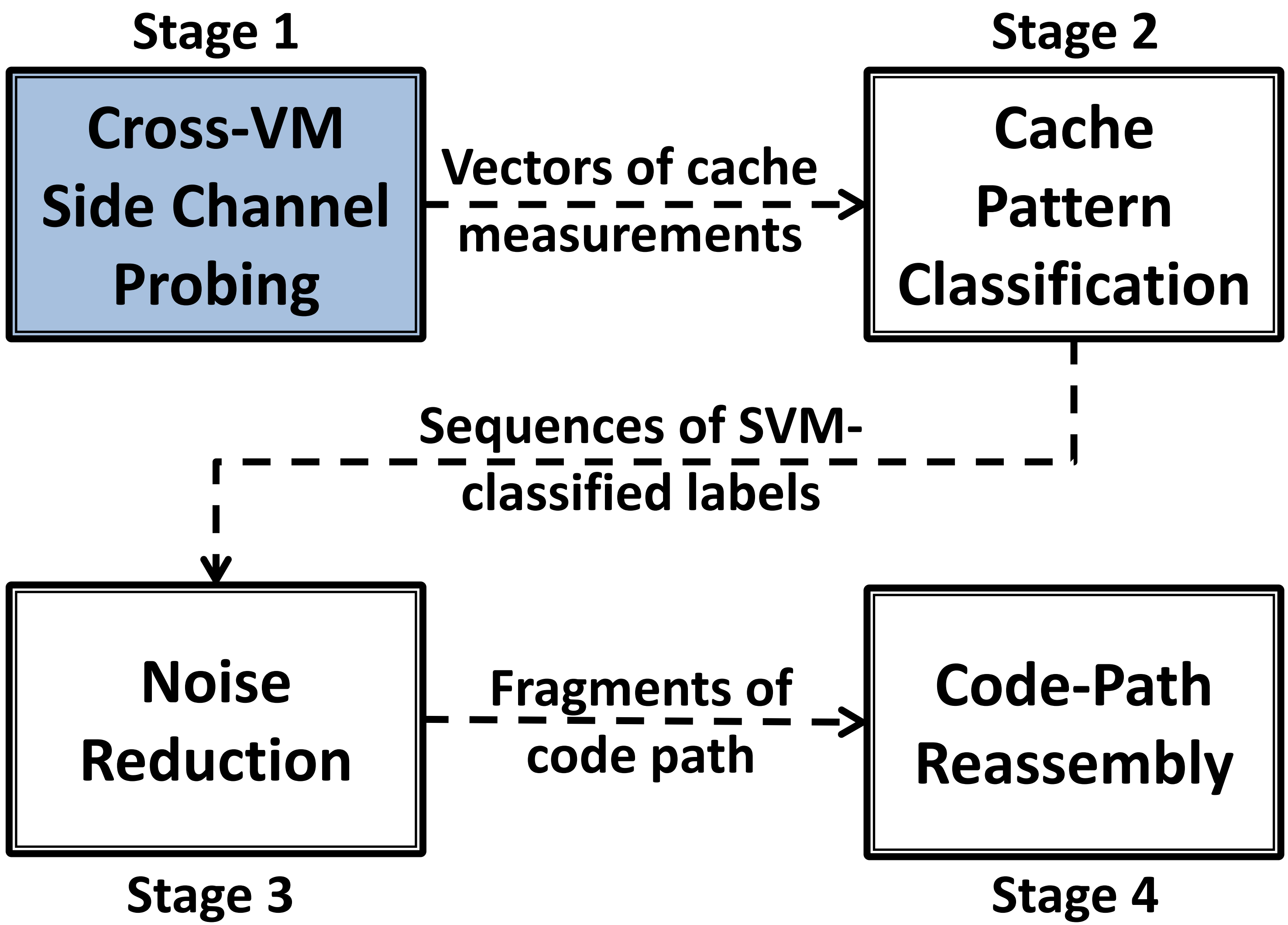
Related Work

Publication	Multi-Core	Virtualization	w/o SMT	Target
Percival 2005	✗	✗	✗	RSA
Osvik et al. 2006	✗	✗	✗	AES
Neve et al. 2006	✗	✗	✓	AES
Aciicmez 2007	✗	✗	✗	RSA
Ristenpart el al. 2009	✓	✓	✓	**load**
Aciicmez et al. 2010	✗	✗	✗	DSA
Bangerter 2011	✗	✗	✓	AES

Related Work

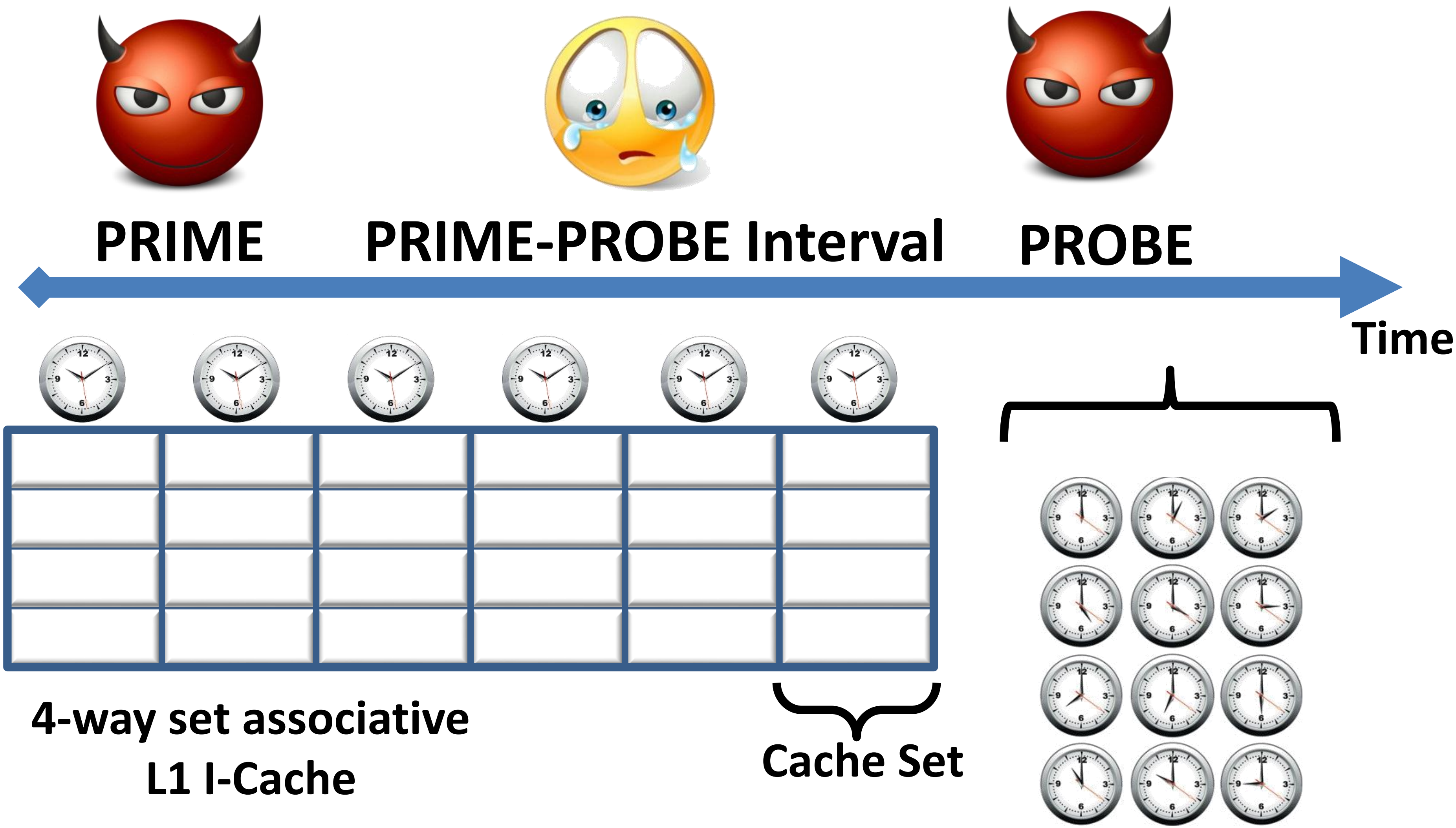
Publication	Multi-Core	Virtualization	w/o SMT	Target
Percival 2005	✗	✗	✗	RSA
Osvik et al. 2006	✗	✗	✗	AES
Neve et al. 2006	✗	✗	✓	AES
Aciicmez 2007	✗	✗	✗	RSA
Ristenpart el al. 2009	✓	✓	✓	load
Aciicmez et al. 2010	✗	✗	✗	DSA
Bangerter 2011	✗	✗	✓	AES
Our work	✓	✓	✓	**ElGamal**

Outline

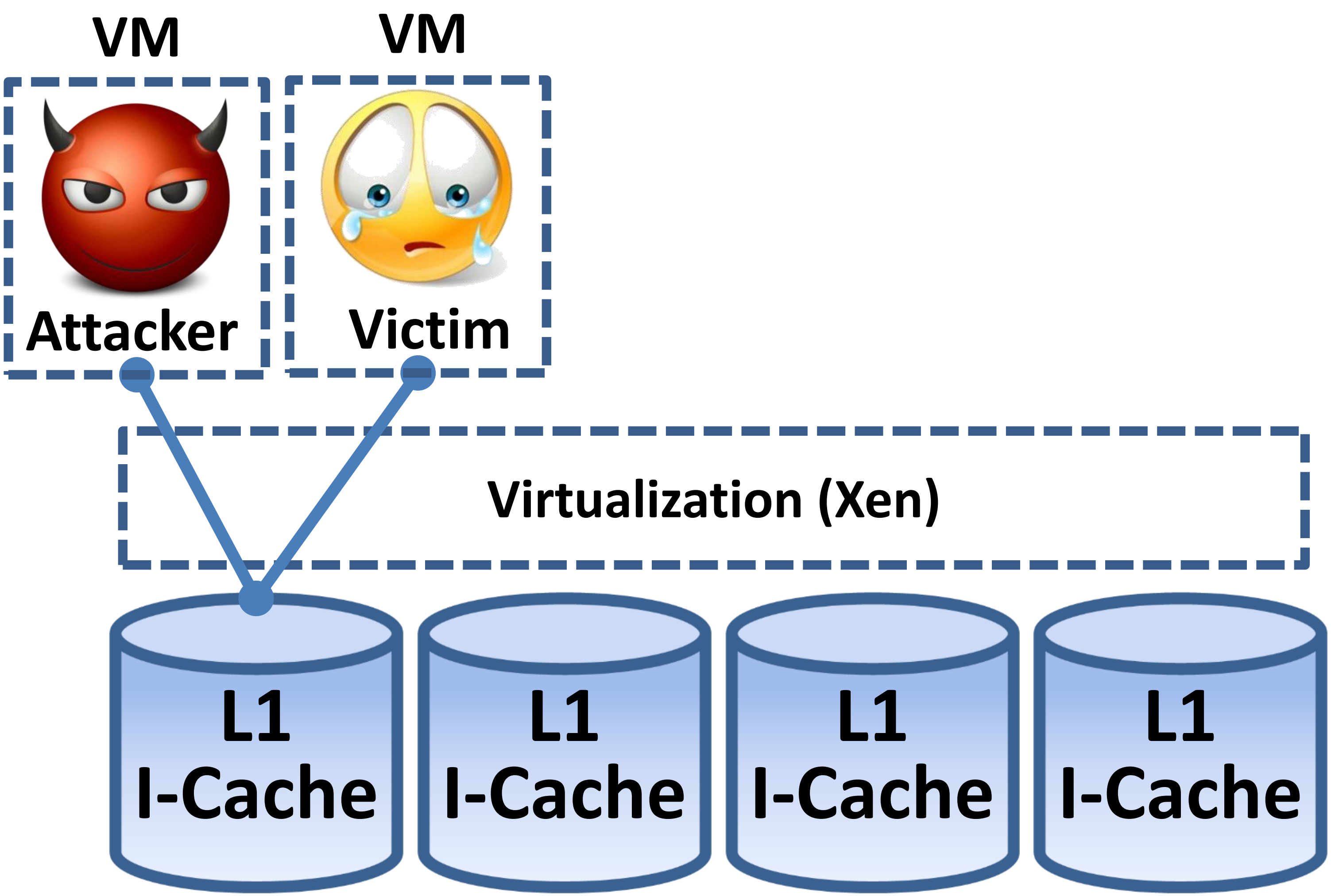


Stage 1
Cross-VM Side Channel Probing
Vectors of cache measurements
Stage 2
Cache Pattern Classification
Sequences of SVM-classified labels
Noise Reduction
Stage 3
Fragments of code path
Code-Path Reassembly
Stage 4

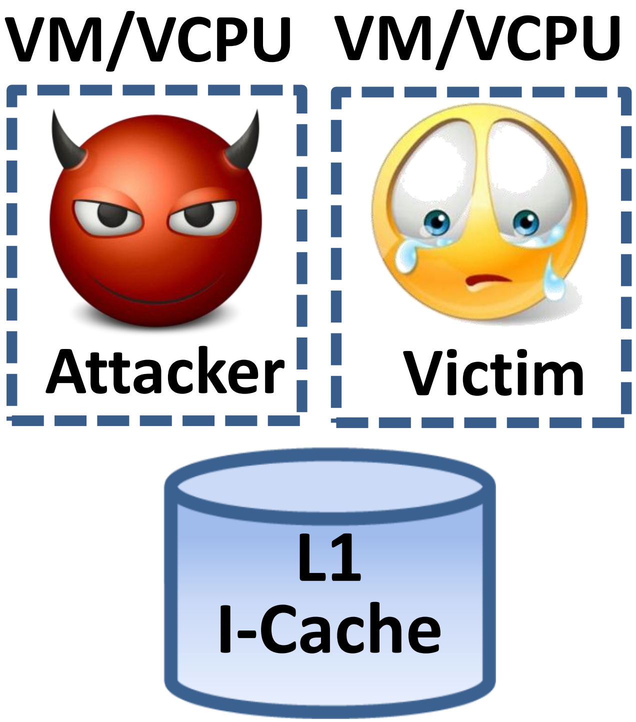
Digress: *Prime-Probe Protocol*



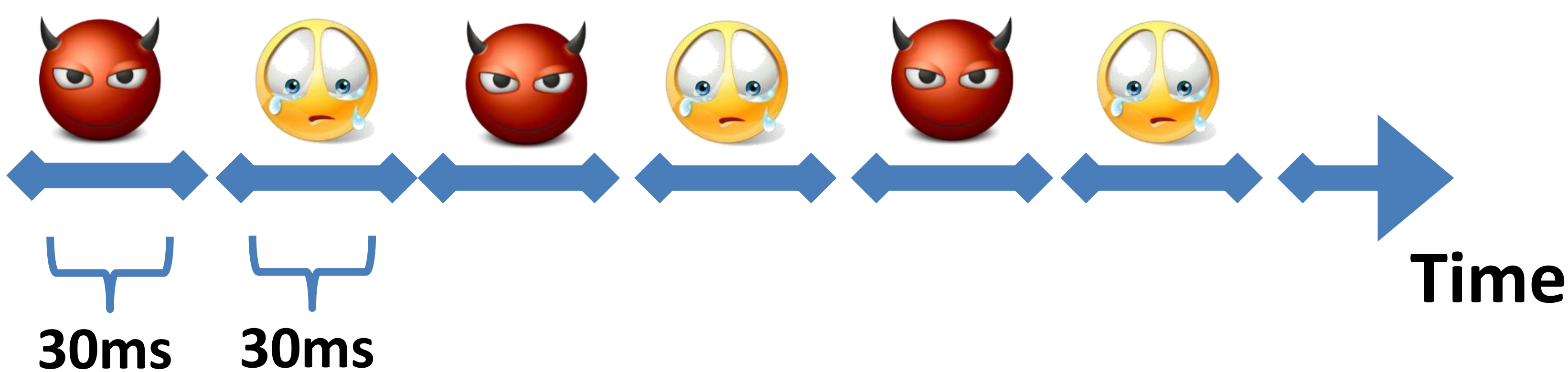
Cross-VM Side Channel Probing



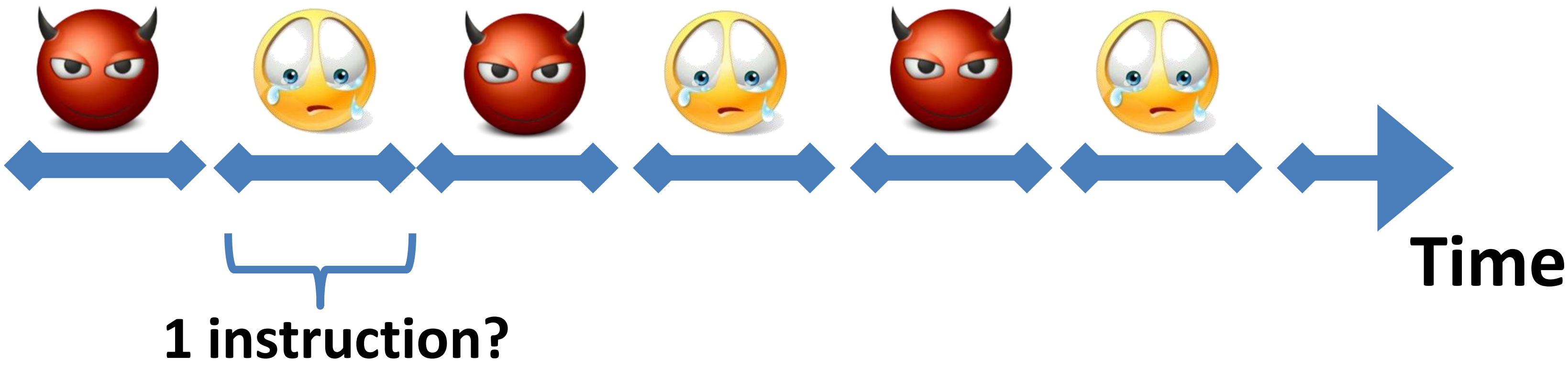
Challenge: Observation Granularity



- **W/ SMT:** tiny prime-probe intervals
- **W/o SMT:** gaming schedulers

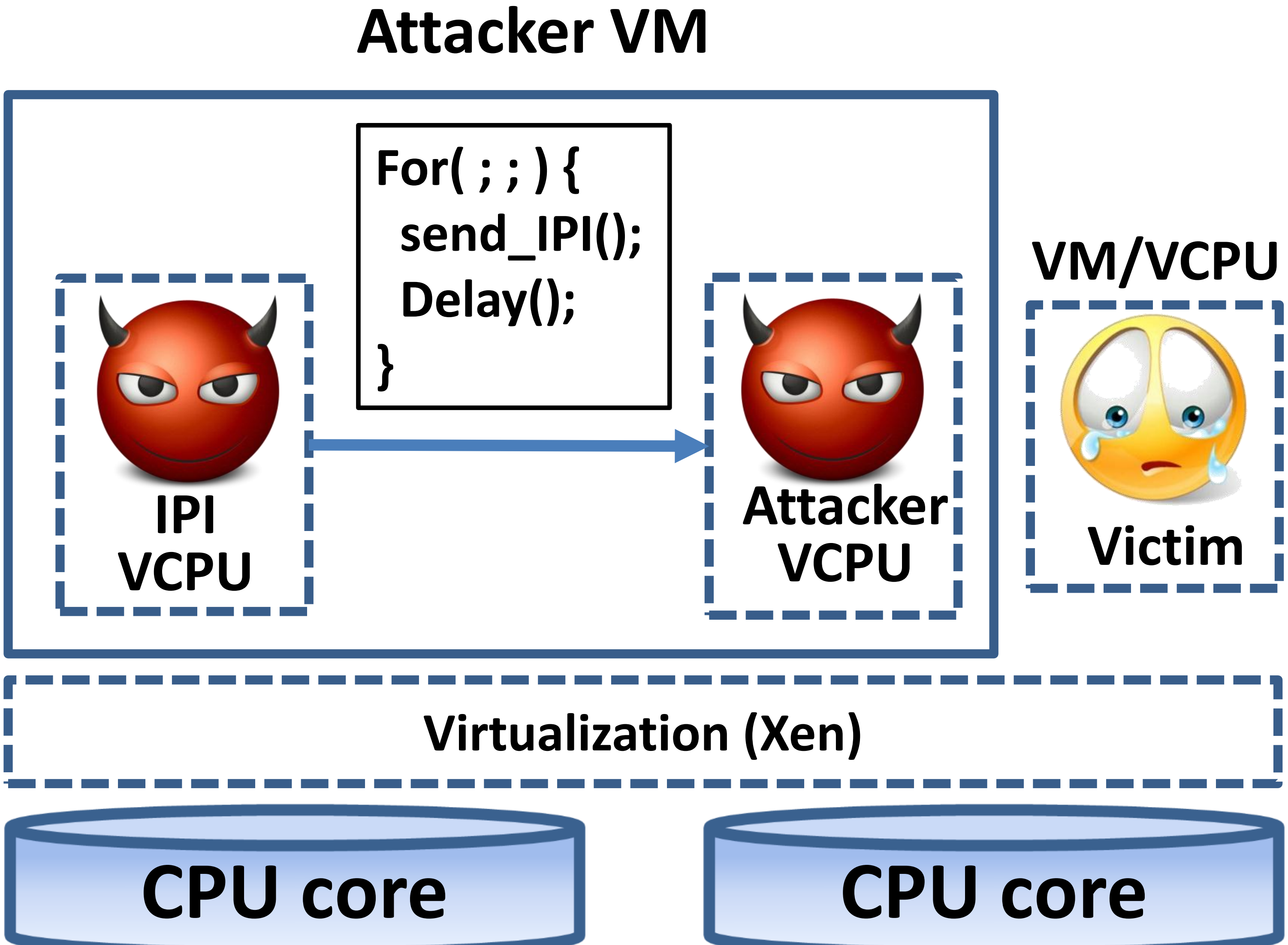


Ideally …

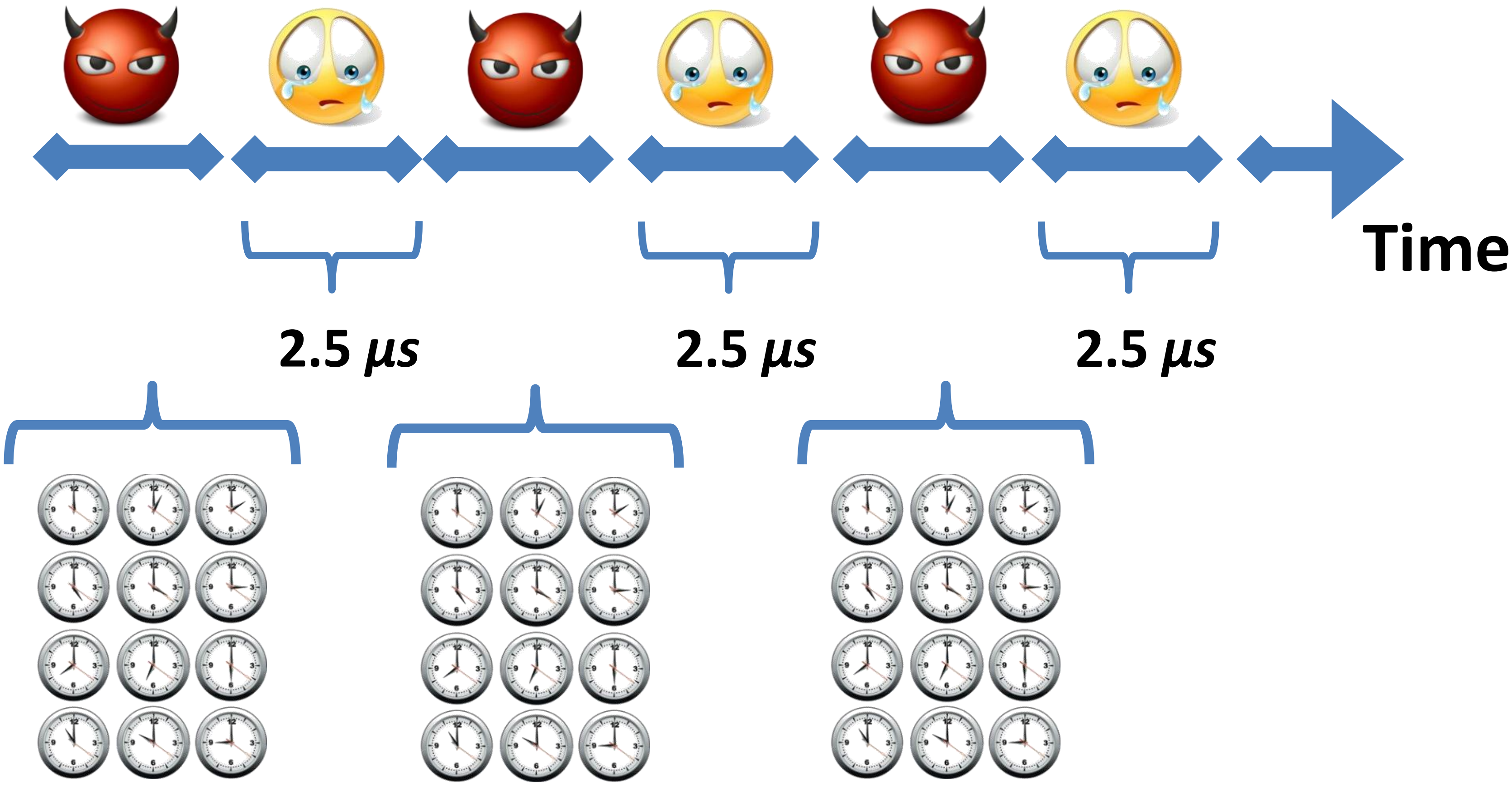


- Use **Interrupts** to preempt the victim:
  - Timer interrupts?
  - Network interrupts?
  - HPET interrupts?
  - **Inter-Processor interrupts (IPI)!**

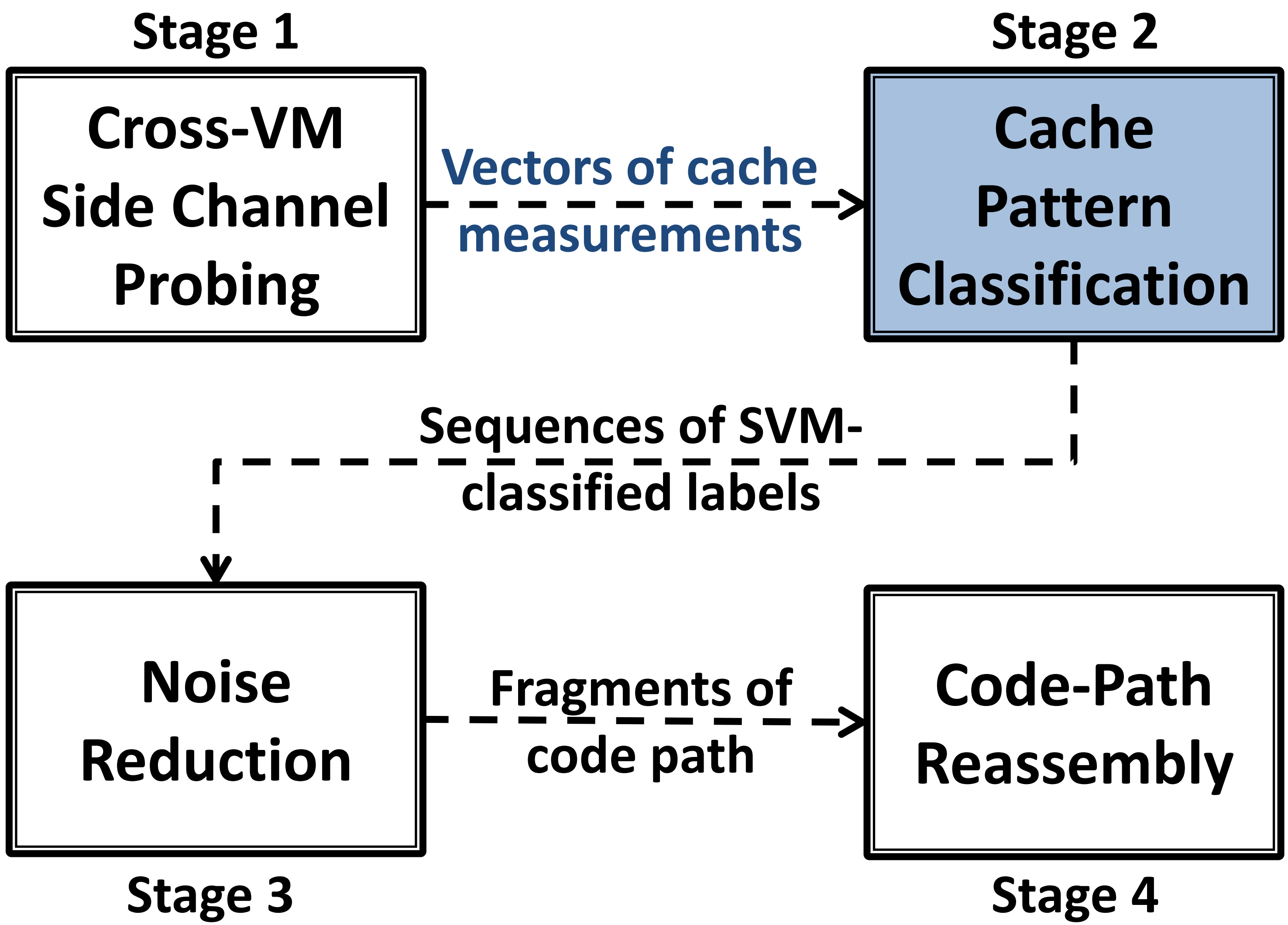
Inter-Processor Interrupts



Cross-VM Side Channel Probing



Outline



Stage 1
Cross-VM Side Channel Probing
Vectors of cache measurements
Stage 2
Cache Pattern Classification
Sequences of SVM-classified labels
Noise Reduction
Stage 3
Fragments of code path
Code-Path Reassembly
Stage 4

Square-and-Multiply

The method is based on the observation that, for a positive integer n, we have

$$x^n = \begin{cases} x\left(x^{\frac{n-1}{2}}\right)^2, & \text{if } n \text{ is odd} \\ \left(x^{\frac{n}{2}}\right)^2, & \text{if } n \text{ is even.} \end{cases}$$

This may easily implemented into the following recursive algorithm:

```
Function exp-by-squaring(x,n)
     if n=1 then return x;
     else if n is even then return exp-by-squaring(x*x, n/2);
     else if n is odd then return x * exp-by-squaring(x*x, (n-1)/2)).
```

Square-and-Multiply (mod)

$$e = \sum_{i=0}^{n-1} a_i 2^i$$

In such notation, the *length* of e is n bits. a_i can take the value 0 or 1 for any i such that $0 \le i < n - 1$. By definition, $a_{n-1} = 1$.

The value b^e can then be written as:

$$b^e = b^{\left(\sum_{i=0}^{n-1} a_i 2^i\right)} = \prod_{i=0}^{n-1} \left(b^{2^i}\right)^{a_i}$$

The solution c is therefore:

$$c \equiv \prod_{i=0}^{n-1} \left(b^{2^i}\right)^{a_i} \pmod{m}$$

```
function modular_pow(base, exponent, modulus)
    result := 1
    while exponent > 0
        if (exponent mod 2 == 1):
           result := (result * base) mod modulus
        exponent := exponent >> 1
        base = (base * base) mod modulus
    return result
```

Square-and-Multiply (libgcrypt)

/* $y = x^e \bmod N$, from **libgcrypt***/

Modular Exponentiation (x, e, N):

let $e_n \ldots e_1$ be the bits of e

$y \leftarrow 1$

for e_i in $\{e_n \ldots e_1\}$

    $y \leftarrow$ **Square**(y) **(S)**

    $y \leftarrow$ **Reduce**(y, N) **(R)**

    if $e_i = 1$ then

        $y \leftarrow$ **Multi**(y, x) **(M)**

        $y \leftarrow$ **Reduce**(y, N) **(R)**

$e_i = 1 \rightarrow$ "SRMR"

$e_i = 0 \rightarrow$ "SR"

Cache Pattern Classification

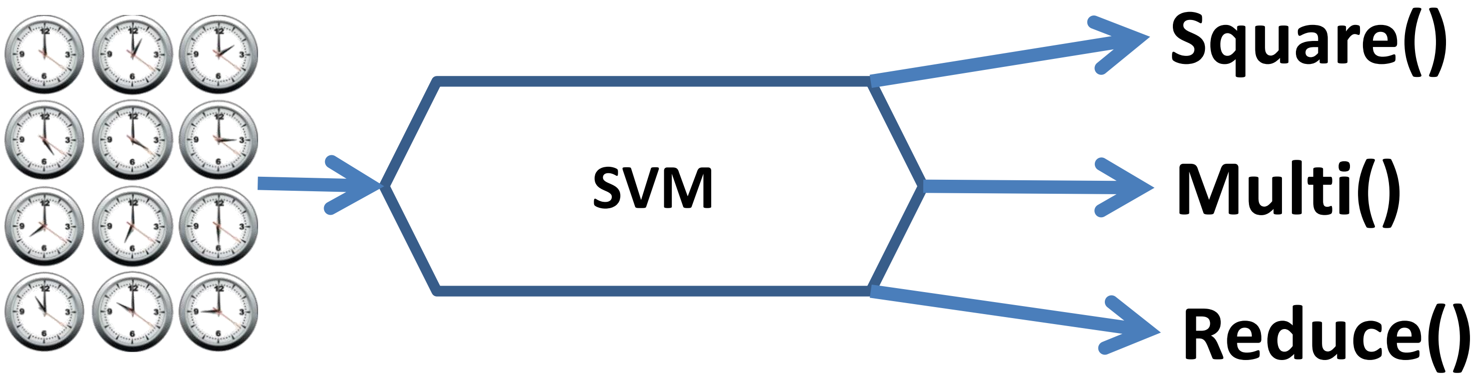
Key observation:

Footprints of different functions are distinct in the I-Cache !

- Square(): cache set 1, 3, …, 59
- Multi(): cache set 2, 5, …, 60, 61
- Reduce(): cache set 2, 3, 4, …, 58

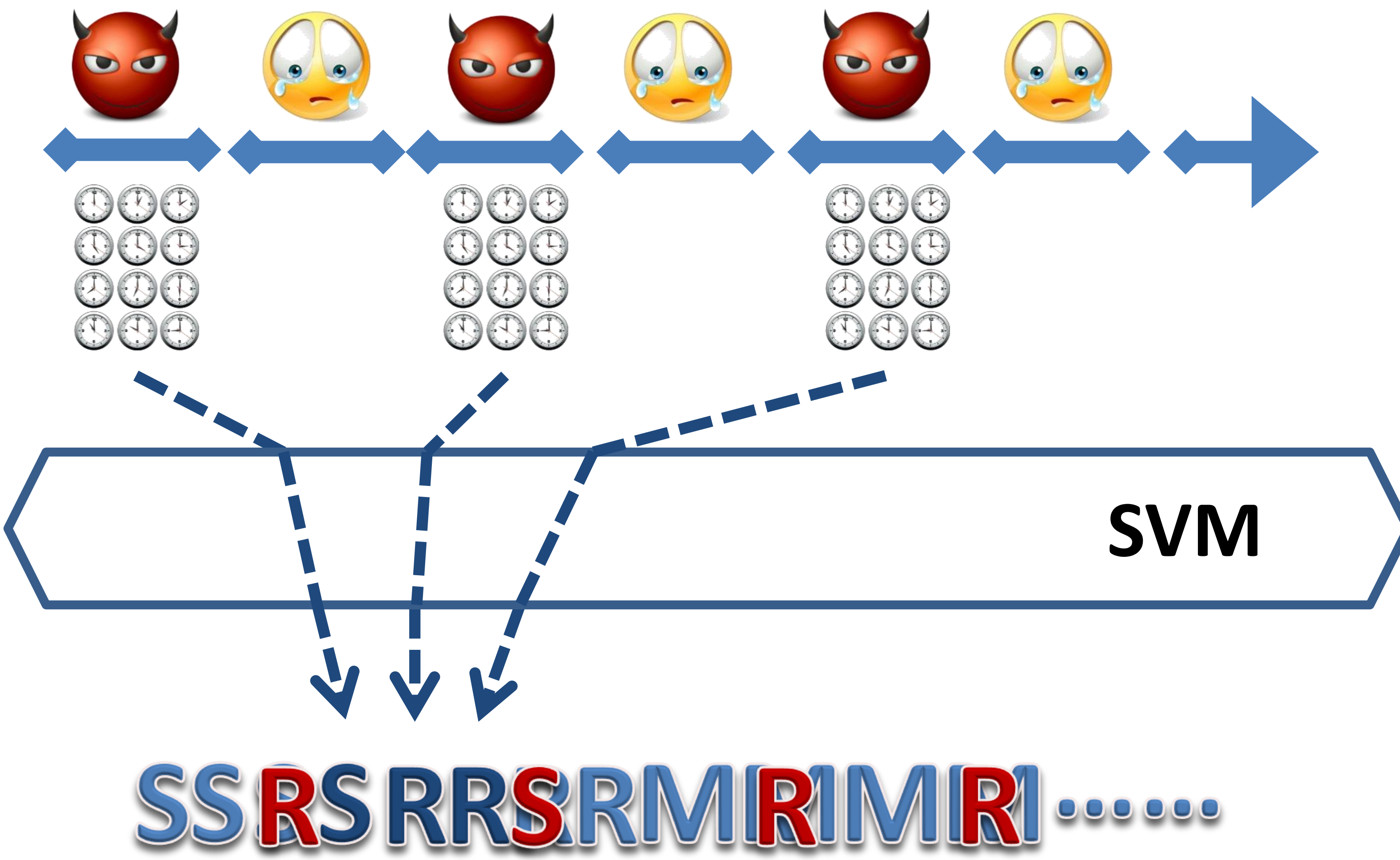
Support Vector Machine

Noise: hypervisor context switch

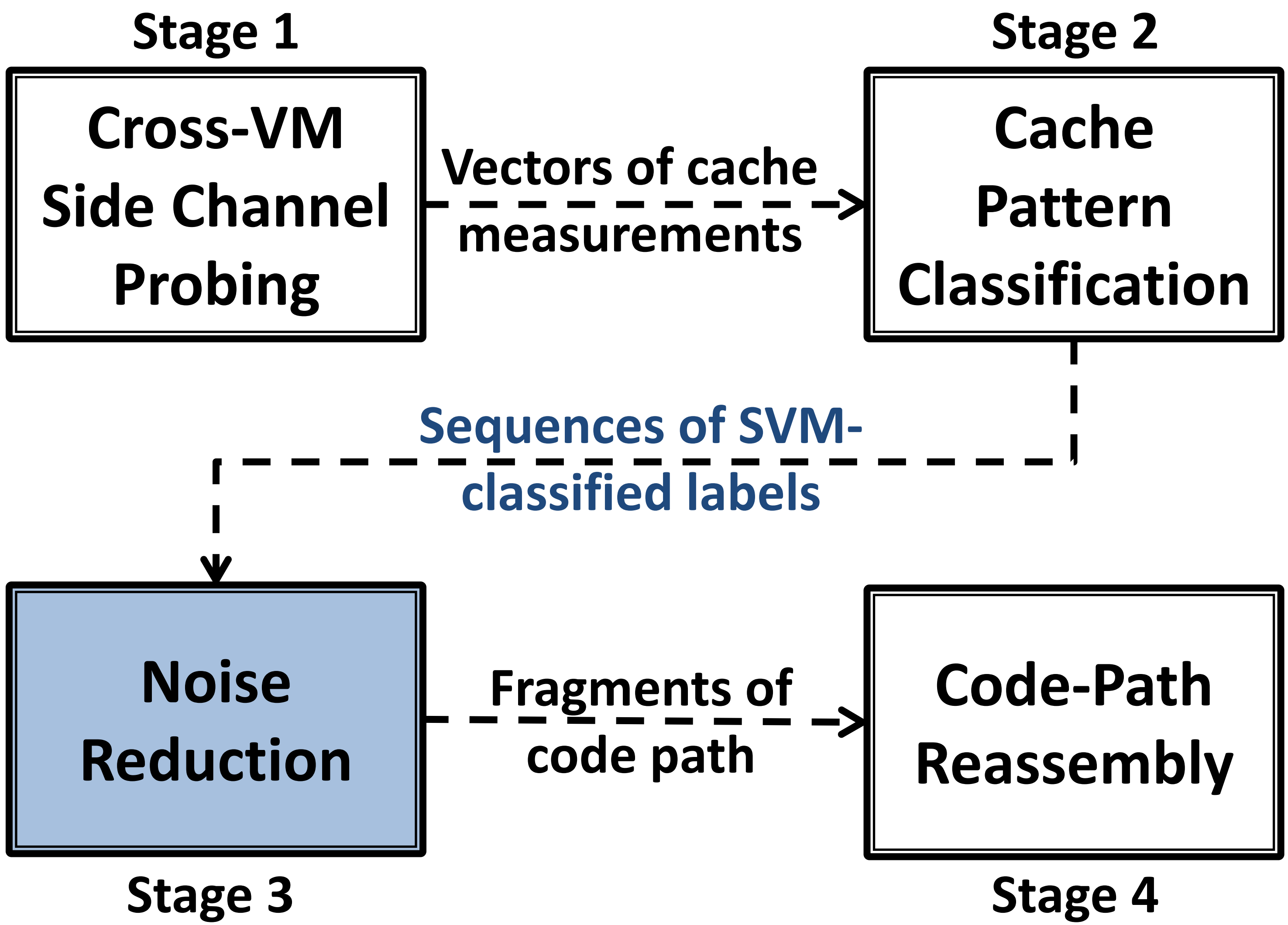


Read more on SVM training

Support Vector Machine



Outline



Stage 1
Cross-VM Side Channel Probing
Vectors of cache measurements
Stage 2
Cache Pattern Classification
Sequences of SVM-classified labels
Noise Reduction
Stage 3
Fragments of code path
Code-Path Reassembly
Stage 4

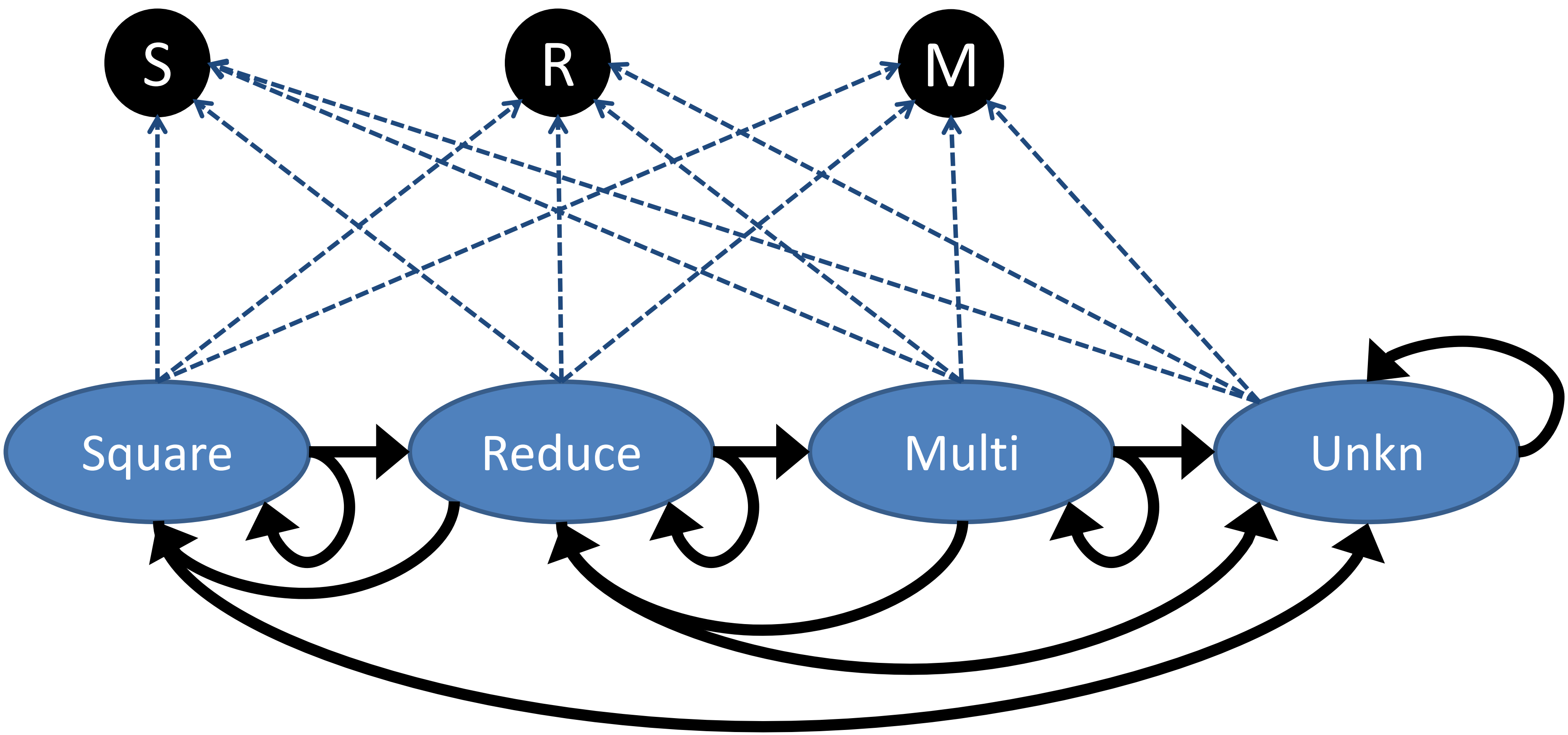
Noise Reduction

SSRSRRSRMRMR ……

Square Reduce Multi

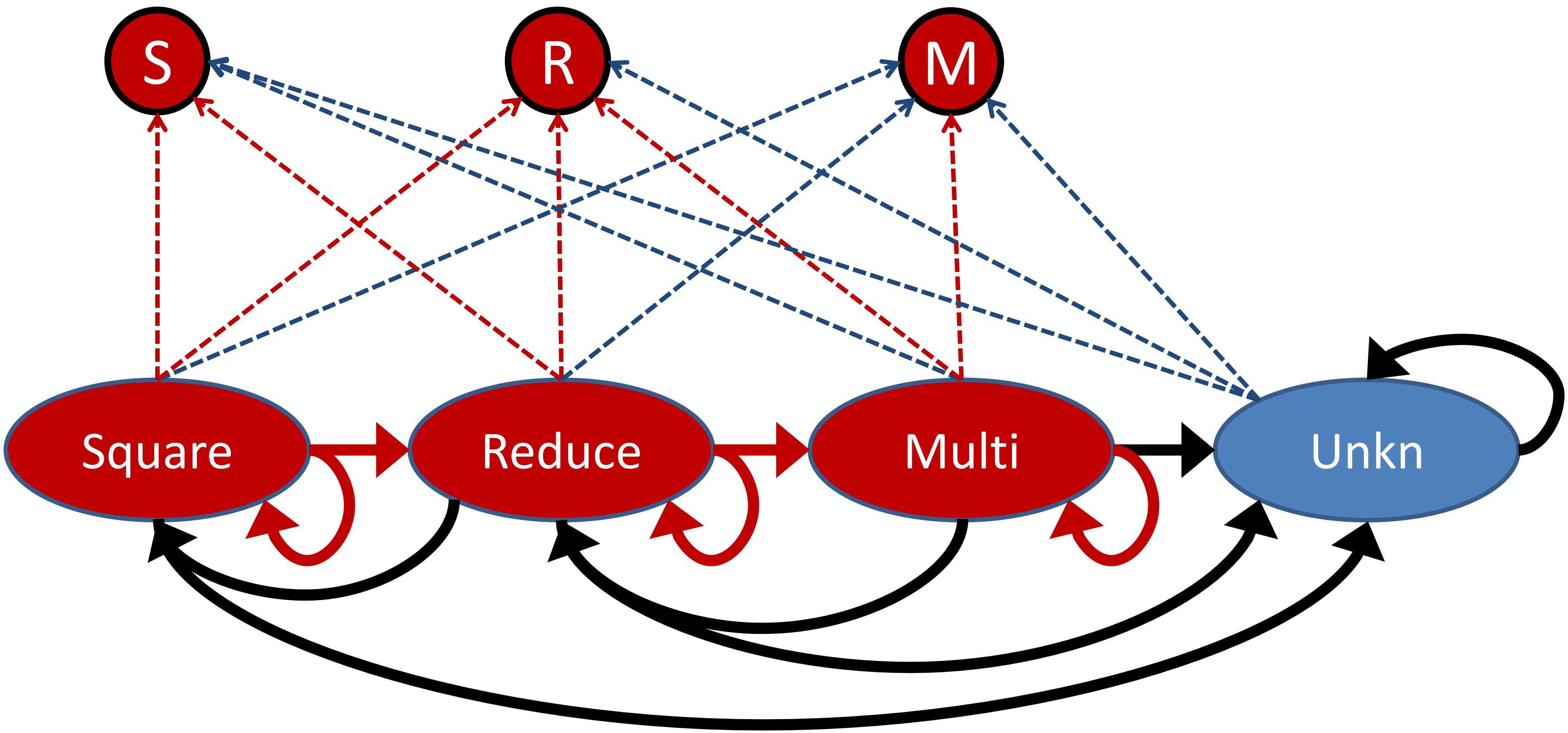
requires robust automated error correction

Hidden Markov Model



Hidden Markov Model

SSRS RRSR MRMR ......



Hidden Markov Model

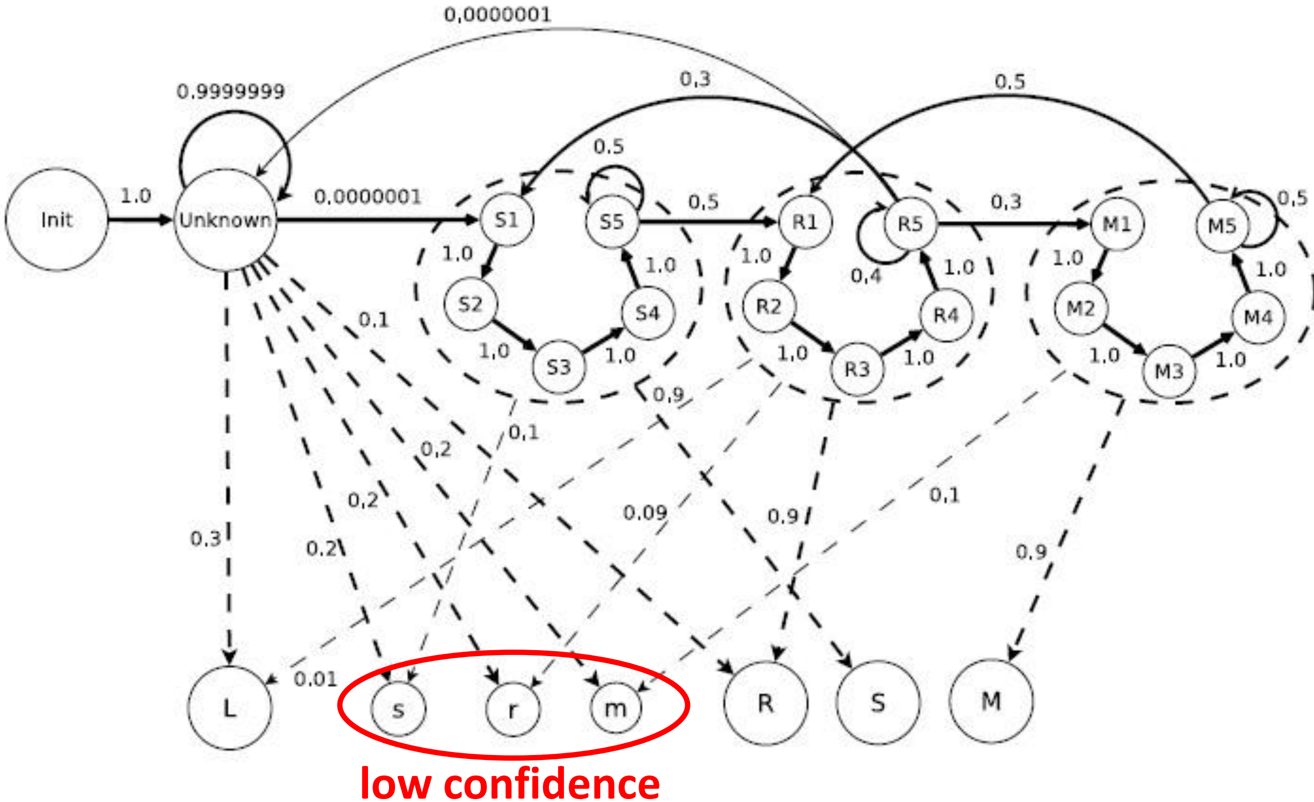
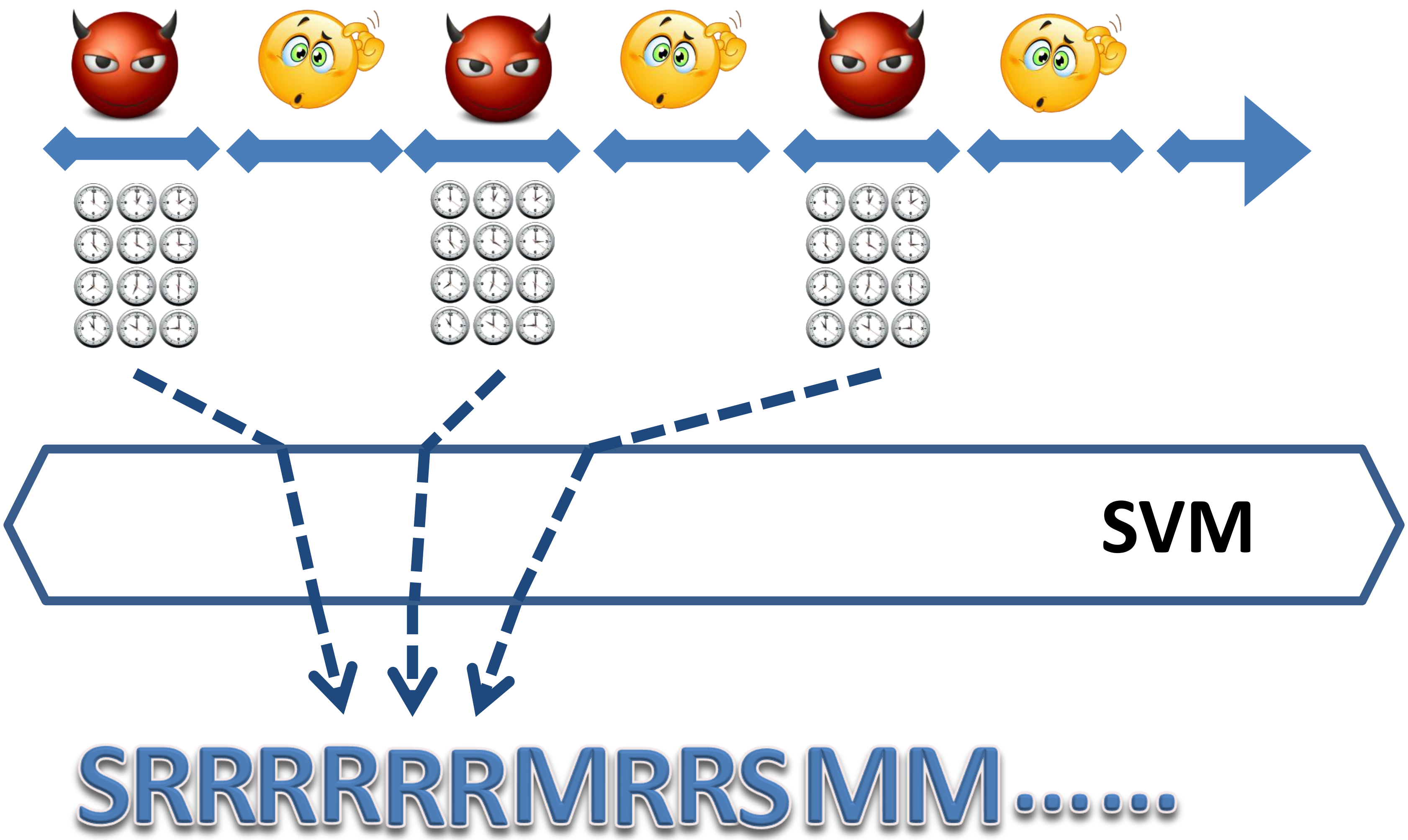


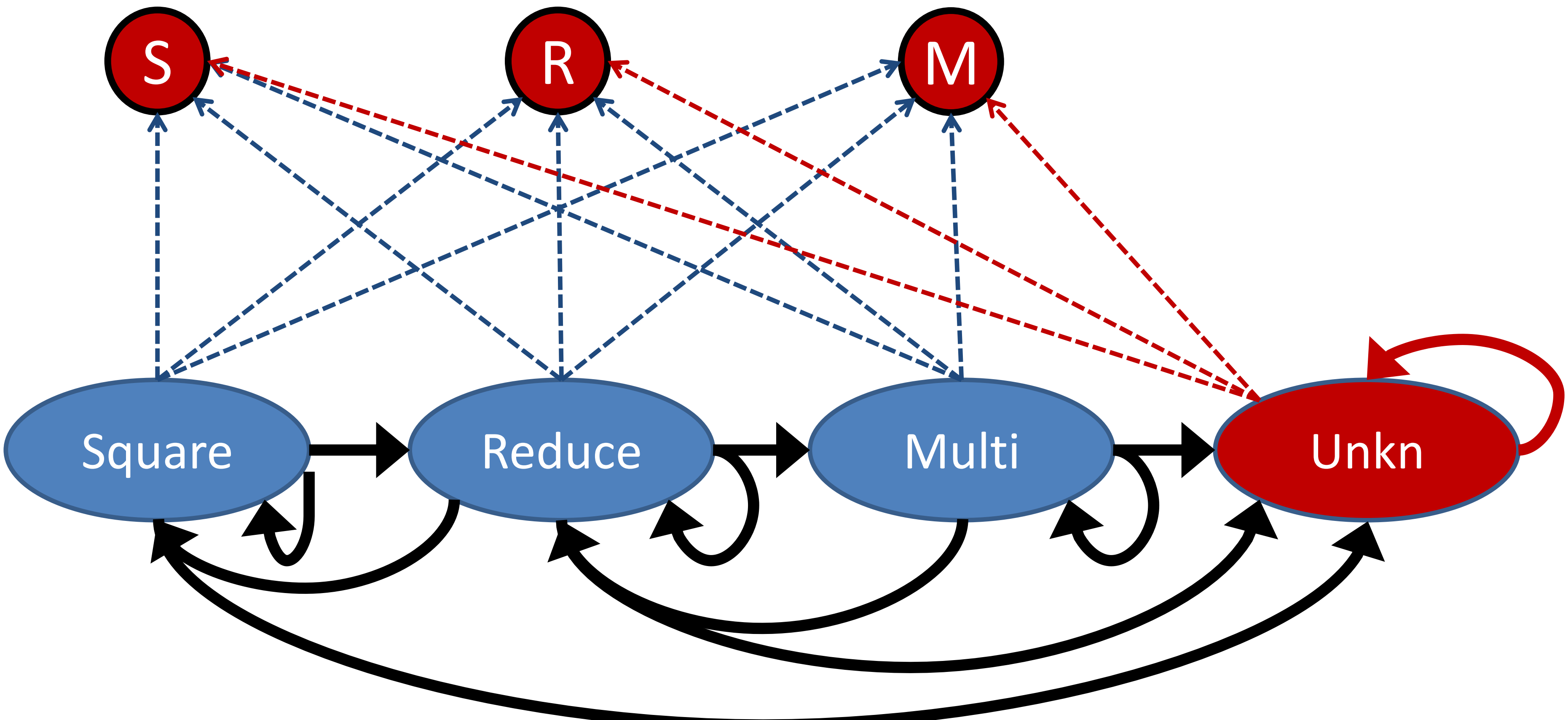
Figure 3: Diagram of the HMM used in our experiments with 4096-bit base x and modulus N. Emission labels are depicted in the lower half, hidden states in the upper half. Solid arrows indicate transitions, dotted arrows denote emissions. Emission probabilities below 0.01 are omitted.

Eliminate Non-Crypto Computation



Eliminate Non-Crypto Computation

SRRRRRRMRRSMM……



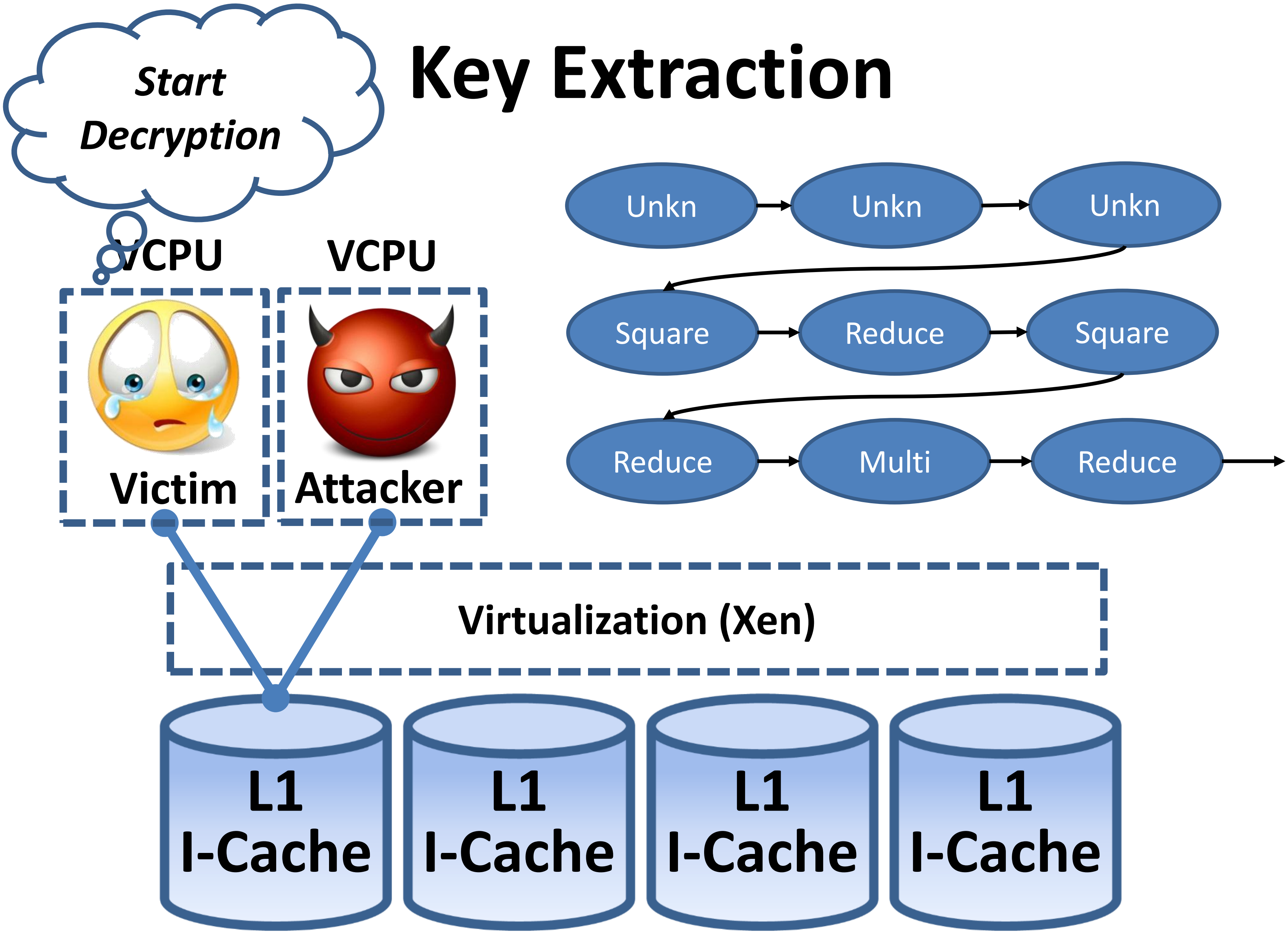
Eliminate Non-Crypto Computation

Key Observations

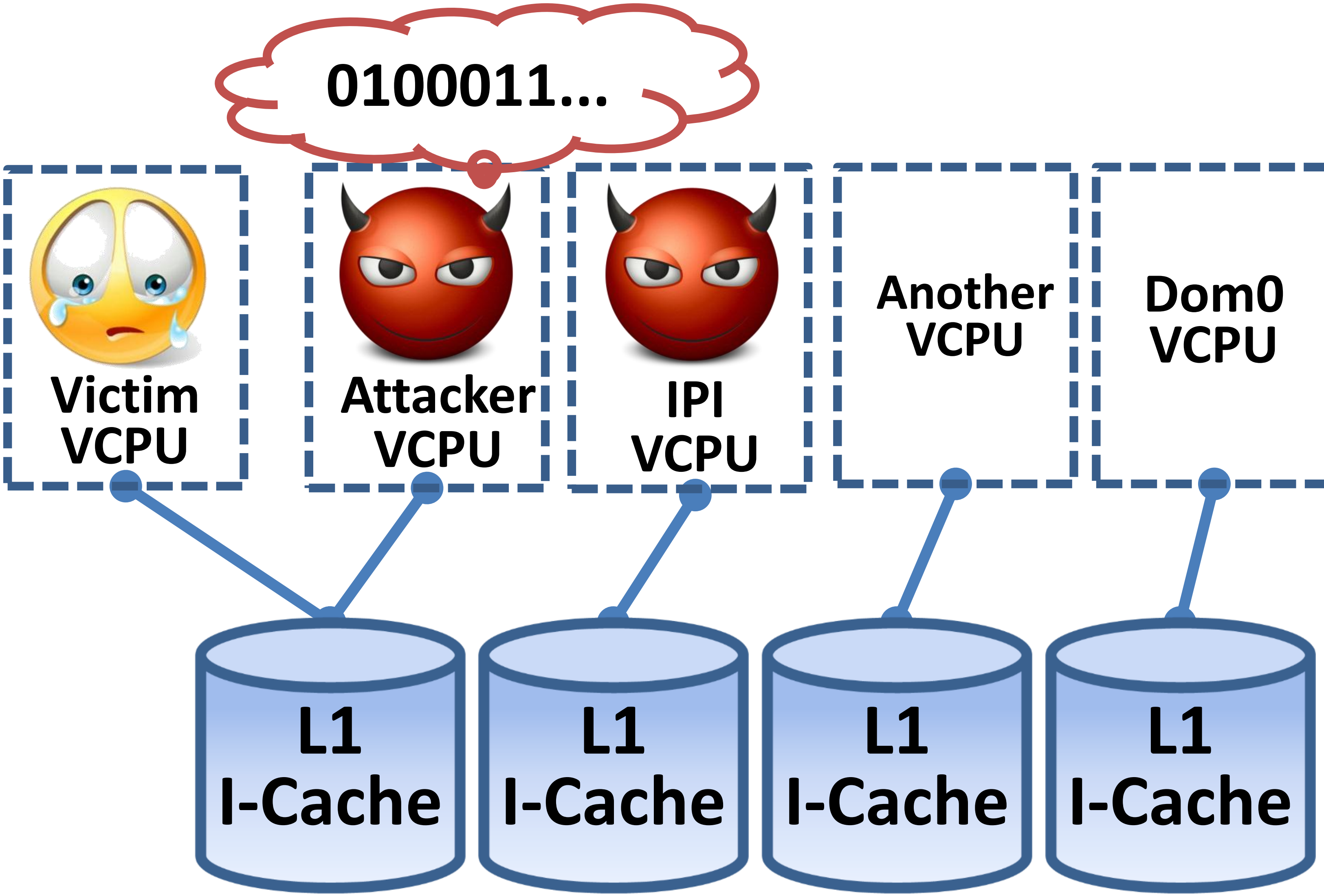
S:M Ratio should be roughly 2:1 for long enough sequences!

“MM” signals an error (never two sequential multiply operations)

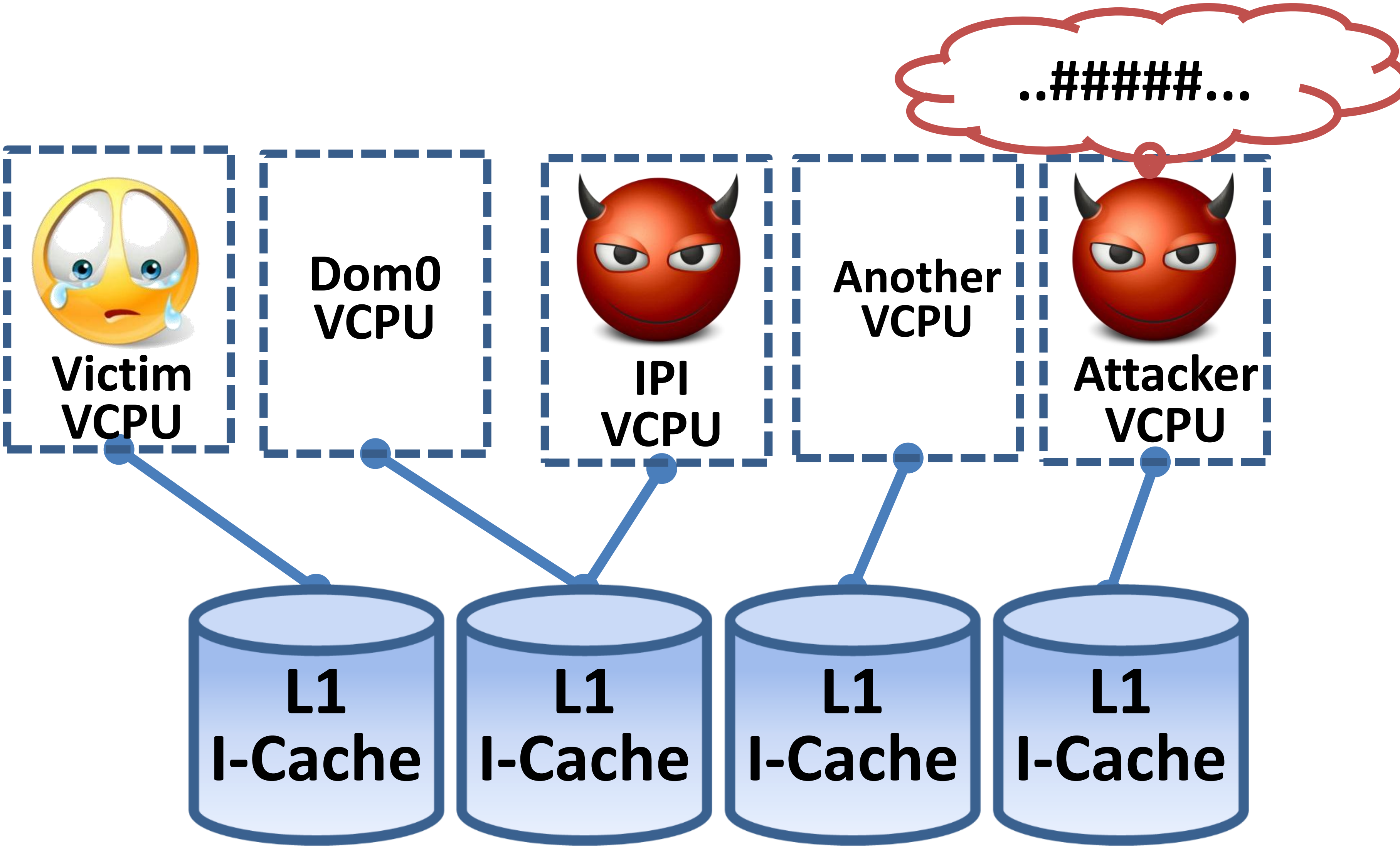
Key Extraction



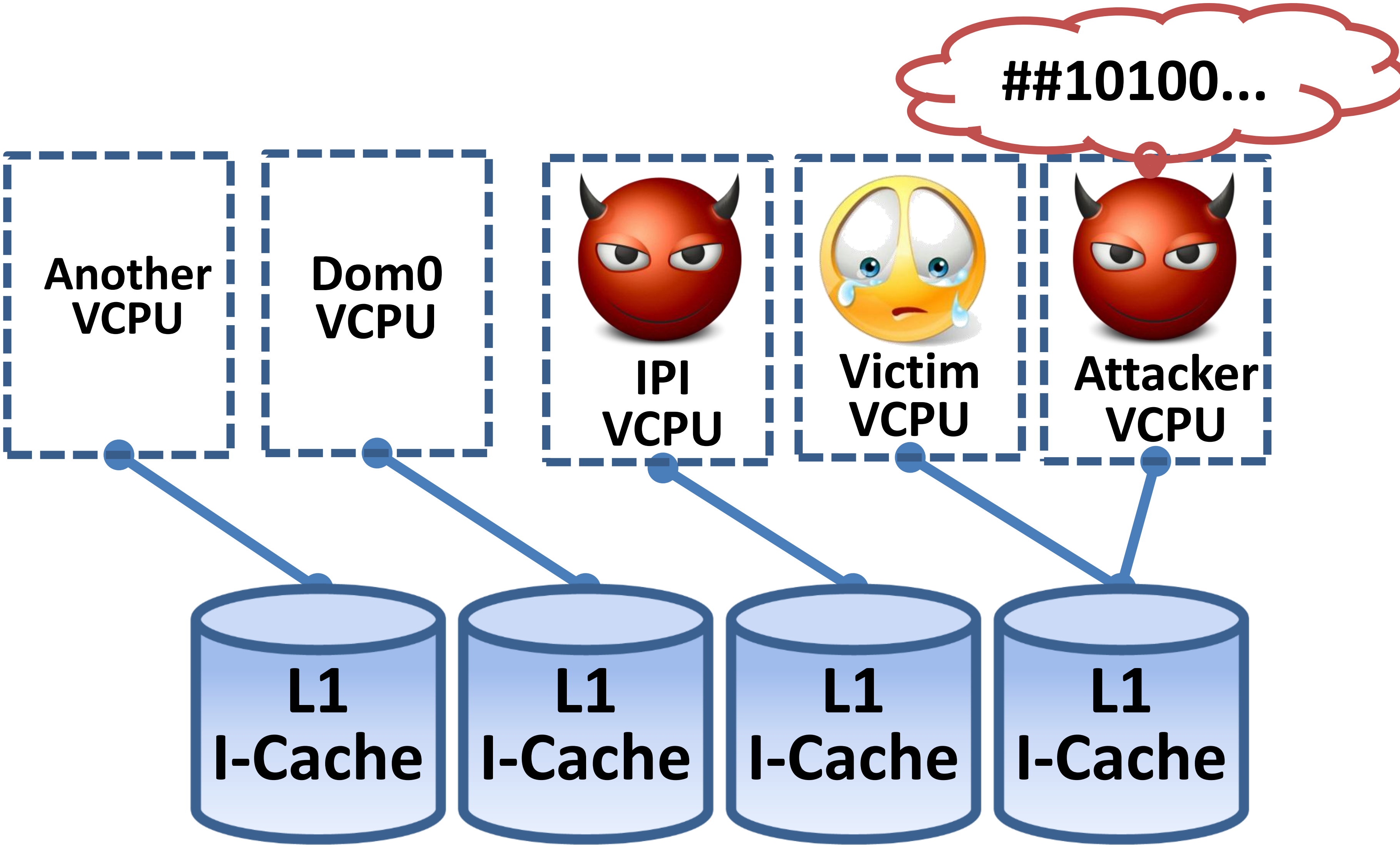
Multi-Core Processors



Multi-Core Processors



Multi-Core Processors



From an Attacker's Perspective

#####1001111010####

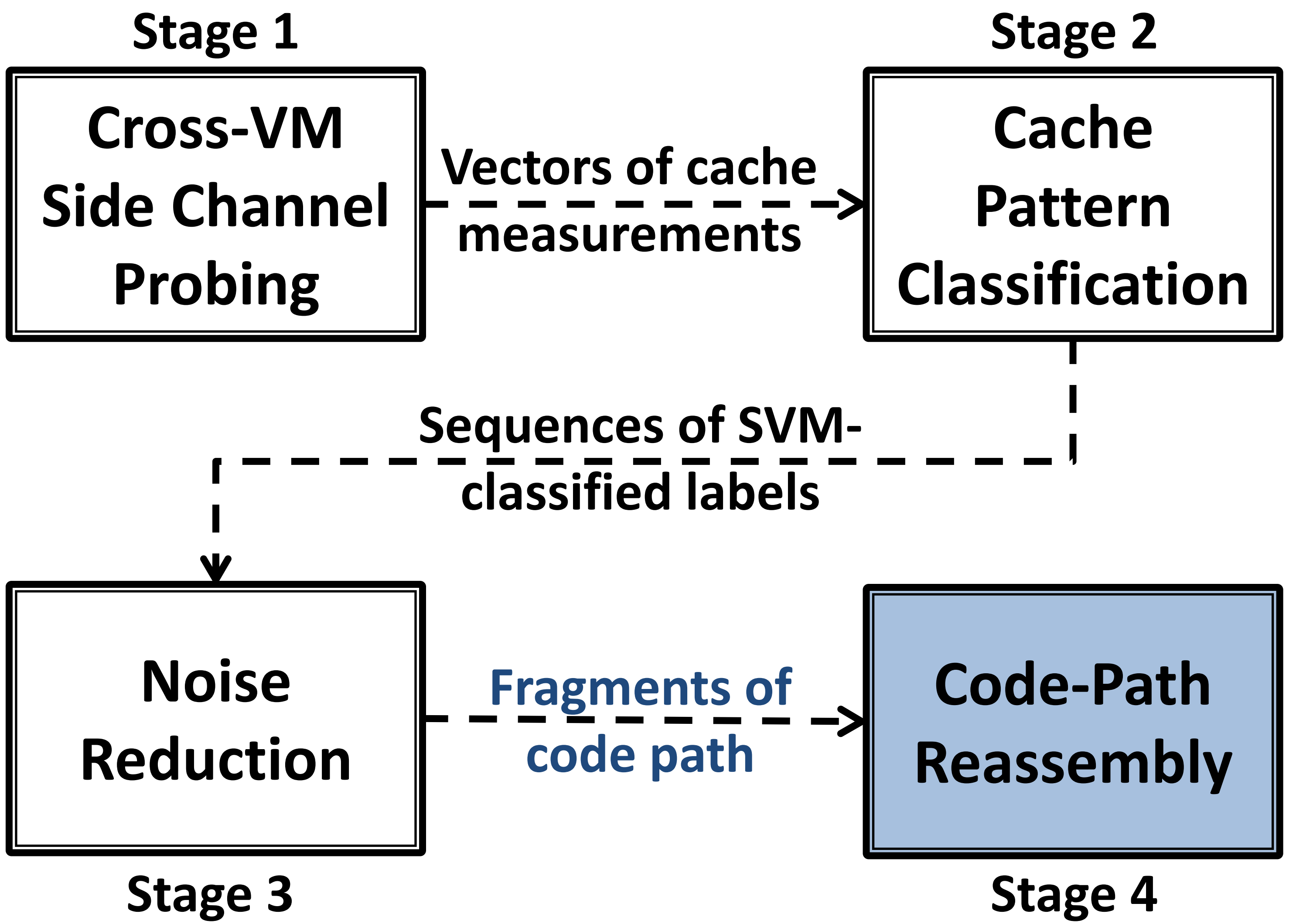
#0111101011########

####110101101#####0

1101110############

###########ooooooooo

Outline


Stage 1
Cross-VM Side Channel Probing
Vectors of cache measurements
Stage 2
Cache Pattern Classification
Sequences of SVM-classified labels
Noise Reduction
Stage 3
Fragments of code path
Code-Path Reassembly
Stage 4

Code-Path Reassembly

1001110010

0111101111

110101101

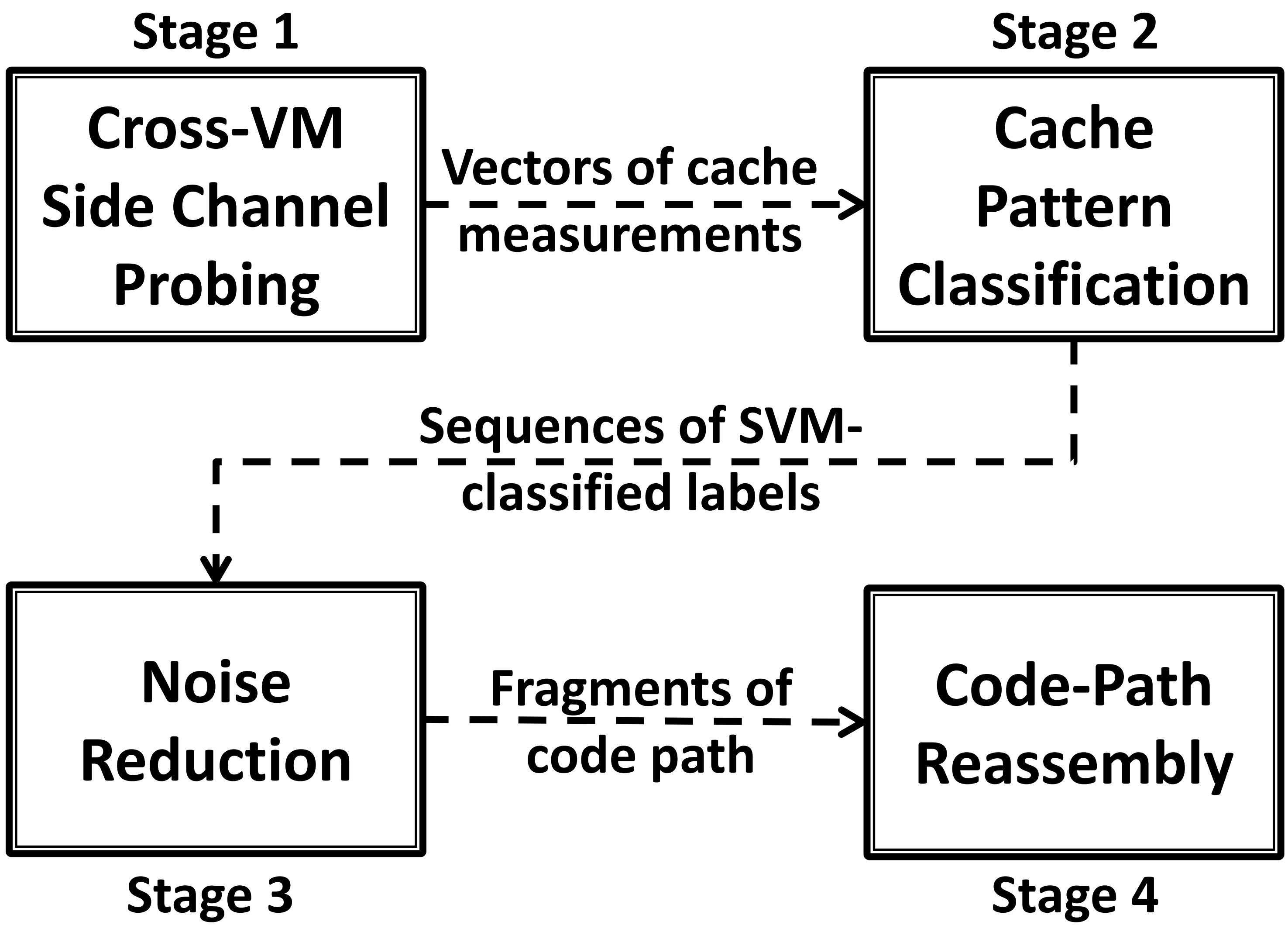
11101110

DNA ASSEMBLY

No error bit!

100111*01*1101110

Outline



Stage 1
Cross-VM Side Channel Probing
Vectors of cache measurements
Stage 2
Cache Pattern Classification
Sequences of SVM-classified labels
Noise Reduction
Stage 3
Fragments of code path
Code-Path Reassembly
Stage 4

Evaluation

- Intel Yorkfield processor
  - 4 cores, 32KB L1 instruction cache
- Xen + linux + GnuPG + libgcrypt
  - Xen 4.0
  - Ubuntu 10.04, kernel version 2.6.32.16
  - Victim runs GnuPG v.2.0.19 (latest)
  - libgcrypt 1.5.0 (latest)
  - ElGamal, 4096 bits

Results

- **Work-Conserving Scheduler**
  – 300,000,000 prime-probe results (6 hours)
  – Over 300 key fragments
  – Brute force the key in ~9800 guesses

- **Non-Work-Conserving Scheduler**
  – 1,900,000,000 prime-probe results (45 hours)
  – Over 300 key fragments
  – Brute force the key in ~6600 guesses

Conclusion

- A combination of techniques
  - **IPI** + **SVM + HMM + Sequence Assembly**
- Demonstrate a cross-VM access-driven cache-based side-channel attack
  - **Multi-core** processors **without SMT**
  - Sufficient fidelity to exfiltrate **cryptographic** keys