New Results on Instruction Cache Attacks

Onur Acıiçmez Billy Bob Brumley Philipp Grabher

Samsung Electronics, USA

Aalto University School of Science and Technology, Finland

University of Bristol, UK

August 18, 2010

Main Contributions

- improve instruction cache analysis techniques,
- mount a lattice attack on OpenSSL's DSA implementation using this improved analysis,
- present results of instruction cache analysis in a real-world attack settings,
- and outline possible countermeasures to prevent instruction cache attacks and measure their performance impact.

Micro-Architectural (MA) Attacks

- Standard processor components can be the source of sensitive information leakage for otherwise secure software
- MA attacks typically observe timing or power variations caused by processor components
- Processor components that have been targeted by MA attacks include:
  - Different types of caches
  - Branch prediction units
  - (Shared) functional units
- Difficult to prevent

Cache Attacks

- Cache is a fast memory block close to the CPU that holds frequently accessed data and instructions
  - Cache Hit Data in the cache ⇒low latency
  - Cache Miss Data not in the cache ⇒ high(er) latency
  - Timing difference between these two events is observable
- Time-driven versus Access-driven Attacks
- Parallel application execution on modern CPUs
  - Quasi-parallel execution enabled by OS support
  - Explicitly-parallel execution enabled by extra hardware

Previous Work

Hu 1992	- Caches as covert channels
Page 2002	- Theoretical attack via power trace analysis
Percival 2005	- Access-driven attack on RSA
Acıiçmez 2007	- Instruction cache analysis of RSA
Brumley et al. 2009	- Cache template attack on ECDSA

Digital Signature Algorithm

DSA

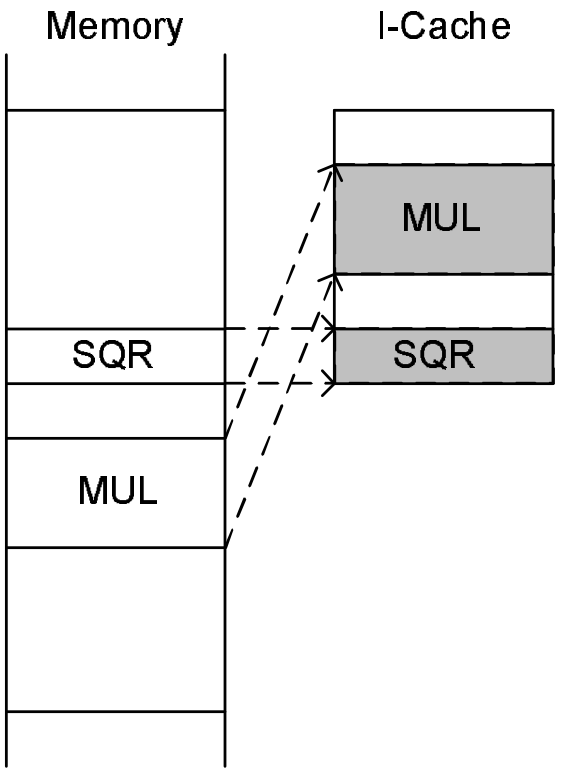
$$r = g^k \bmod p \bmod q$$
$$s = k^{-1}(h(m) + xr) \bmod q$$

Sliding Window Exponentation

Input: M, window size w and an l-bit exponent $k = (k_{l-1}, k_{l-2}, \ldots, k_0)$
Output: $X = M^k$
Pre-compute M^i for all odd $i < 2^w$;
$X \leftarrow 1$, $i \leftarrow l - 1$;
while $i \geq 0$ **do**
  if $e_i = 0$ **then** $t \leftarrow 1$, $u \leftarrow 0$;
  else Find the largest $t \leq w$ such that $u \leftarrow (k_i, \ldots, k_{i-t+1})$ is odd;
  $X \leftarrow X^{2^t}$;
  if $u > 0$ **then** $X \leftarrow X \cdot M^u$;
  $i \leftarrow i - t$;
end
return X

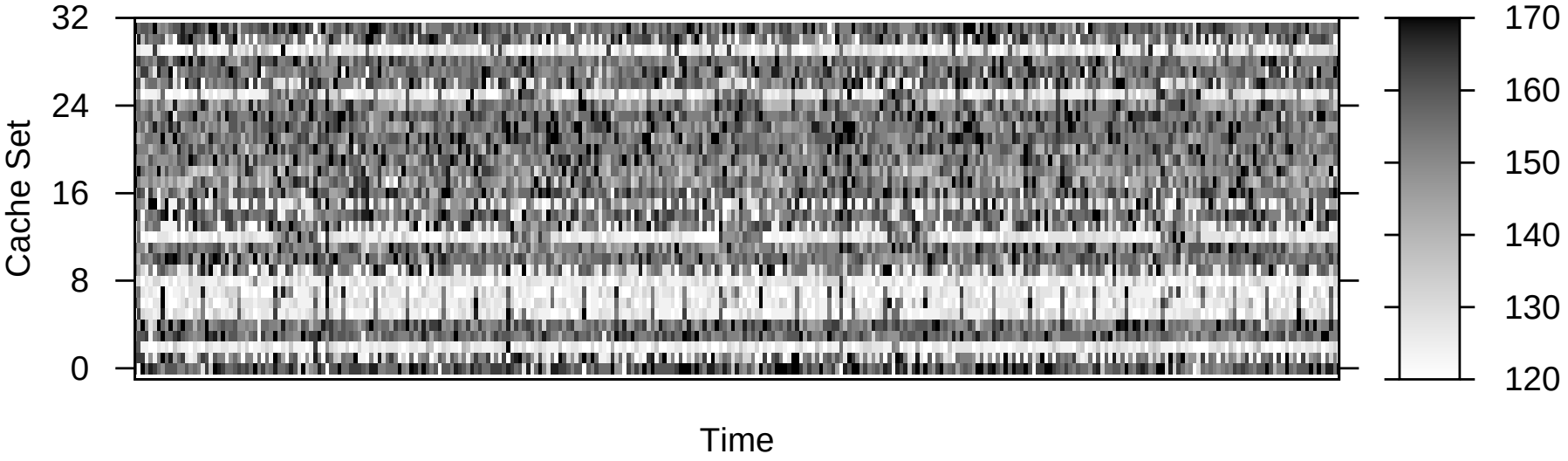
Instruction Cache Attacks: A Primer

- Probe selectively pollutes cache regions
- Crypto process executes one iteration
- Probe re-executes the same code
- Execution time reveals cache access pattern of crypto process
- Theoretical because of synchronisation issue



Example: Instruction Cache Timing Data

- Trace produced by probe:
  1. Read through all cache lines in a cache set.
  2. Measure the time (clock cycles) it took.
  3. Repeat for all cache sets.
  4. Repeat forever . . .



- Use framework of Brumley and Hakala (2009) to analyse such data quickly and automatically

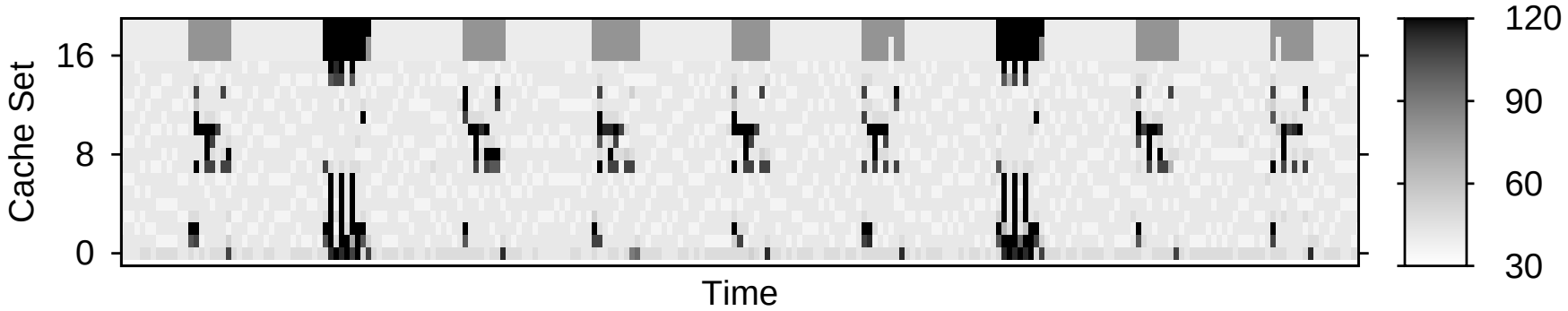
Framework Overview

Automates cache-timing data analysis to recover algorithm state by:

- Collect exemplar timing vectors (templates) reflecting algorithm cache access behavior
- With these templates, create codebook vectors for the different operations we wish to distinguish
- Use Vector Quantisation (VQ) to match incoming cache-timing data to these codebook vectors
- Use output as observation input to Hidden Markov Model (HMM) that accurately models the control flow of the algorithm
- Obtain most likely state sequence

Analysis of Instruction Cache Timing Data

- Data produced by a probe running in parallel with an OpenSSL/DSA sign operation



- Top two rows are meta data of the framework analysis:

  Bottom metadata VQ classification
  Top metadata HMM state guess

Key Recovery with Lattice Attack

- Attacker collects tuples $(r_i, s_i, m_i, \hat{k}_i)$
- We obtain a system of equations of the form:

$$k_i = s_i^{-1}(h(m_i) + r_i x) \bmod q$$

- j equations, $j + 1$ unknowns; but for the k_i we know a portion
- Use known results on lattice attacks (Howgrave-Graham and Smart 01, etc.) to recover private key x
- Use only tuples where LSBs of $\hat{k}_i$ are `000000`; Probability: 2^{-6}

Results (17K signatures)

- 3200 iterations, 75 tuples (59 guesses correct), less than 1 hour

Countermeasures

At the kernel and/or compiler level:

- (Partial) cache disabling
- (Partial) instruction cache flushing evicts sensitive data
- Cache-conscious layout ensures that security-critical code sections map to same regions in the cache

SMT-specific mitigations:

- Disabling SMT functionality
- Security-aware scheduling algorithm

Performance Impact on the Crypto Process

Throughput analysis of OpenSSL/RSA on an Intel Core Duo

	Throughput (Decryptions/s)		
Implementation	1024-bit	2048-bit	4096-bit
Baseline OpenSSL/RSA	576.3	104	17.3
...with cache disabled	0.8	0.1	0.02
... with cache flushing	530	89.3	16.5
... with partial flushing	576.8	104.9	17.1
...with cache-conscious layout	570	102.8	17.2

- Cache-conscious memory layout and partial cache flushing have minimal impact on performance

Conclusion

- Improved instruction cache analysis by using Vector Quantisation and Hidden Markov Models
- Demonstrated its practicality by running a key recovery attack on current version of DSA in OpenSSL
- Cache-conscious memory layout and partial flushing are two low-cost and generic countermeasures
- Considering security as first-class goal in system design could simplify the development of such countermeasures

Fin

Any Questions?