ARES'16
Salzburg, Austria

HyperCrypt: Hypervisor-based Encryption of Kernel and User Space

Johannes Götzfried, Nico Dörr, Ralph Palutke, and Tilo Müller

Department of Computer Science
FAU Erlangen-Nuremberg, Germany

August 31, 2016

Memory Disclosure

Current practices for protecting sensitive data:

- Security-aware people use full disk encryption
- Only protects data while computer is off
- Does not work with devices in Standby-Mode

RAM contains lots of sensitive data:

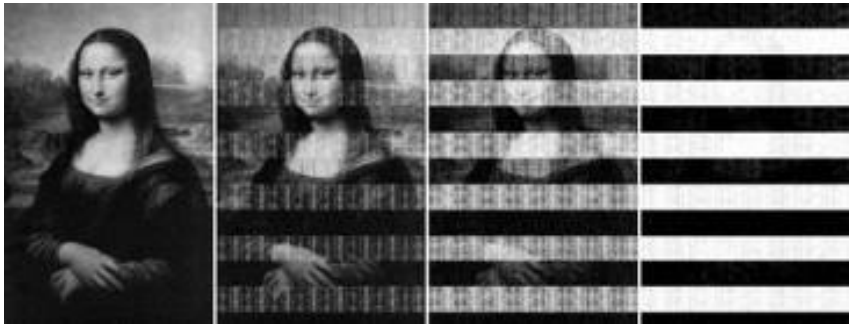
- User passwords or login credentials
- Cryptographic keys
- Personal data and credit card information

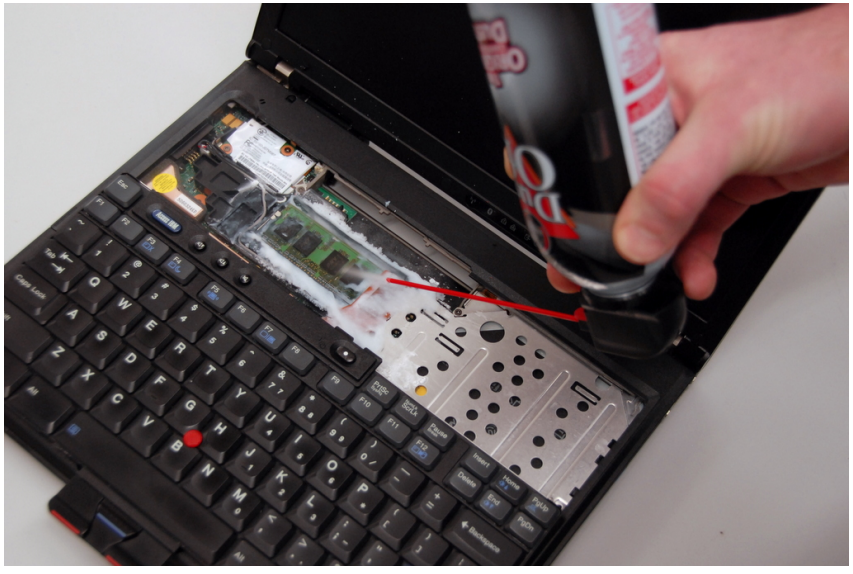
$\rightarrow$ Information is only protected by logical means, e.g., by the OS

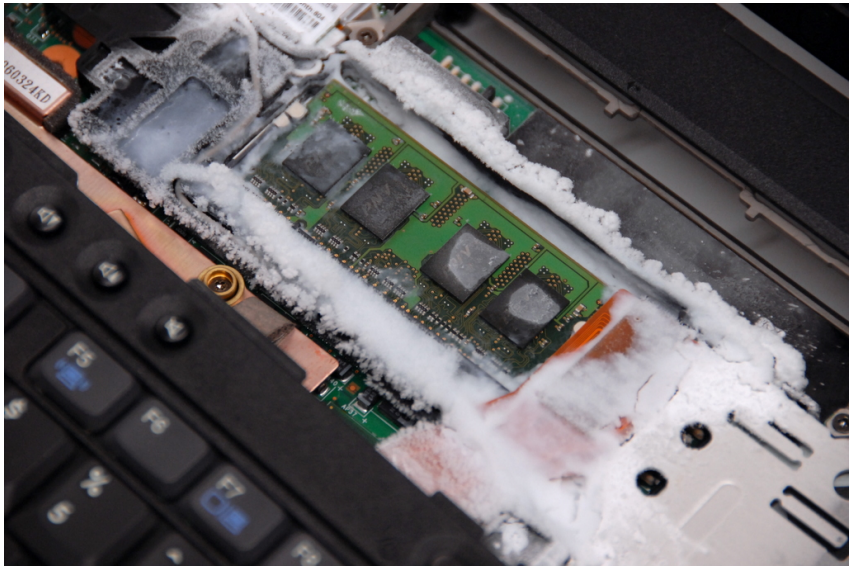
Physical Memory Disclosure

Physical Attacks on RAM:

- By using DMA
  Example: Firewire
- Cold Boot Attacks







Data Lifetime

Goal: Reducing data lifetime of sensitive information within RAM:

- Requires data lifetime knowledge
- Traditional wiping approaches fail (no transparency)

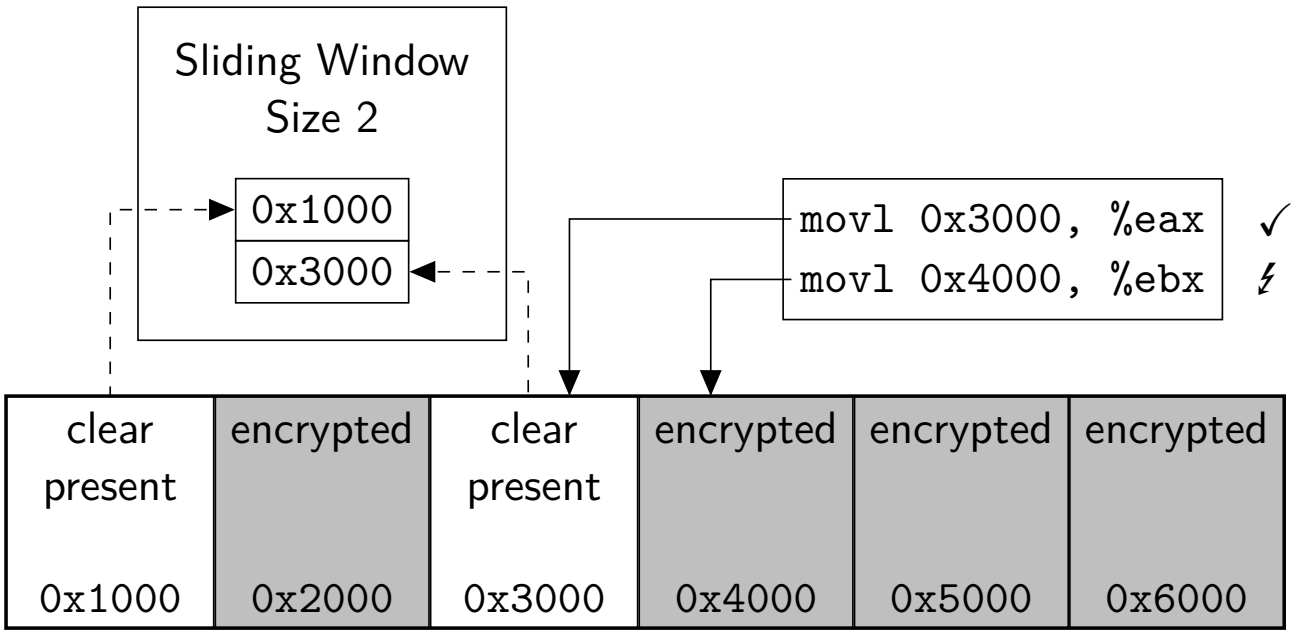
$\rightarrow$ Transparent data encryption effectively hides information

HyperCrypt: Idea

Transparently encrypt data out of the hypervisor:

- Independent from the operating system
- On a per-page basis
- Only a small set of pages remains unencrypted

Sliding window instead of only single page:



→ Sliding window size is a configurable security parameter

HyperCrypt: Background

Prototype implementation as patch for the BitVisor hypervisor:

- Builds upon the BitVisor patch TreVisor
- CPU-bound implementation of AES (TRESOR)
- Stores the key and all intermediate values in CPU registers

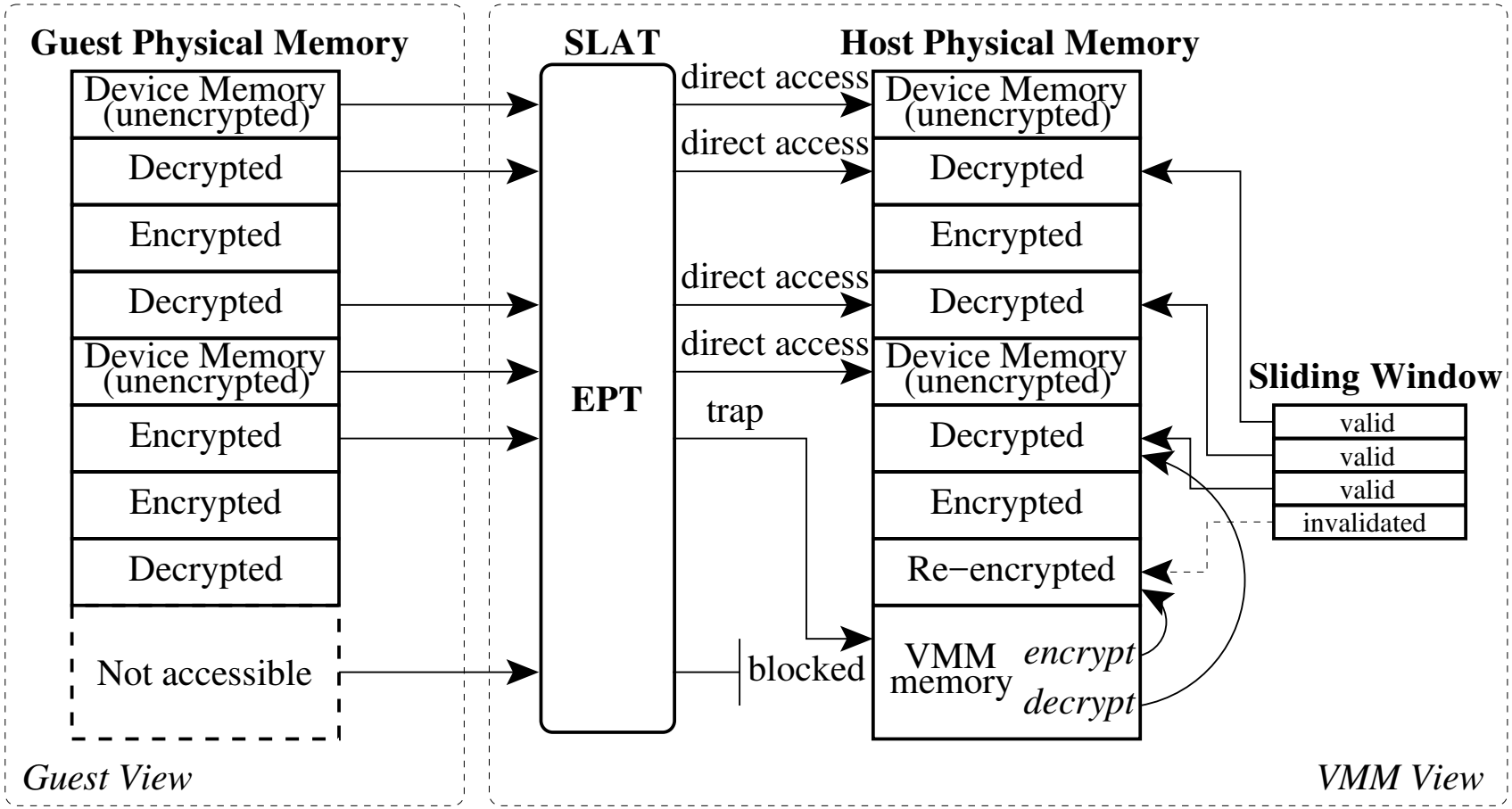
→ No cryptographic keys or key material ever enter RAM

BitVisor memory management:

- BitVisor is a thin hypervisor for I/O device security
- BitVisor uses Second Layer Address Translation (SLAT)
- One-to-one mapping within EPTs
  (only hypervisor is protected)

→ Implement HyperCrypt in the EPT fault handler of BitVisor

HyperCrypt: Workflow



HyperCrypt: Managing Memory Pages

Catching accesses to encrypted memory pages:

- Delete entries from the EPTs to cause faults
- Store flag within a bitmap to identify encrypted pages
- OS is started with empty EPTs

$\rightarrow$ Hooking the EPT fault handler is sufficient

HyperCrypt: Device Memory and DMA

Support for Device Memory:

- Thin hypervisor is not capable of recognizing device memory
- Device memory must not be affected by HyperCrypt
- Check each memory access against system memory map (provided by the BIOS)

→ Provide simple one-to-one mapping for non system RAM

Support for DMA:

- DMA buffers are allocated by the guest OS
- Thin hypervisor is not capable of recognizing DMA buffers
- BitVisor provides drivers for certain devices (hard disk, network card, ...)
- Let BitVisor manage the devices and provide DMA buffers to guest OS

→ Only devices with driver support can be used by the guest

HyperCrypt: Replacement Strategy and Cipher

Replacement Strategy:

- Mostly FIFO for selecting pages for re-encryption
- Second chance algorithm if CPU supports *accessed* bit for EPTs

TreVisor (CPU-bound implementation of AES):

- Configured to behave like AES-128 in XEX mode of operation
- IV to build tweak: Host physical address of the page
- Host physical address cannot be forged by the guest
- Keys can be derived from user password or generated randomly during boot up
- Random number generator of the *Trusted Platform Module* (TPM) is used

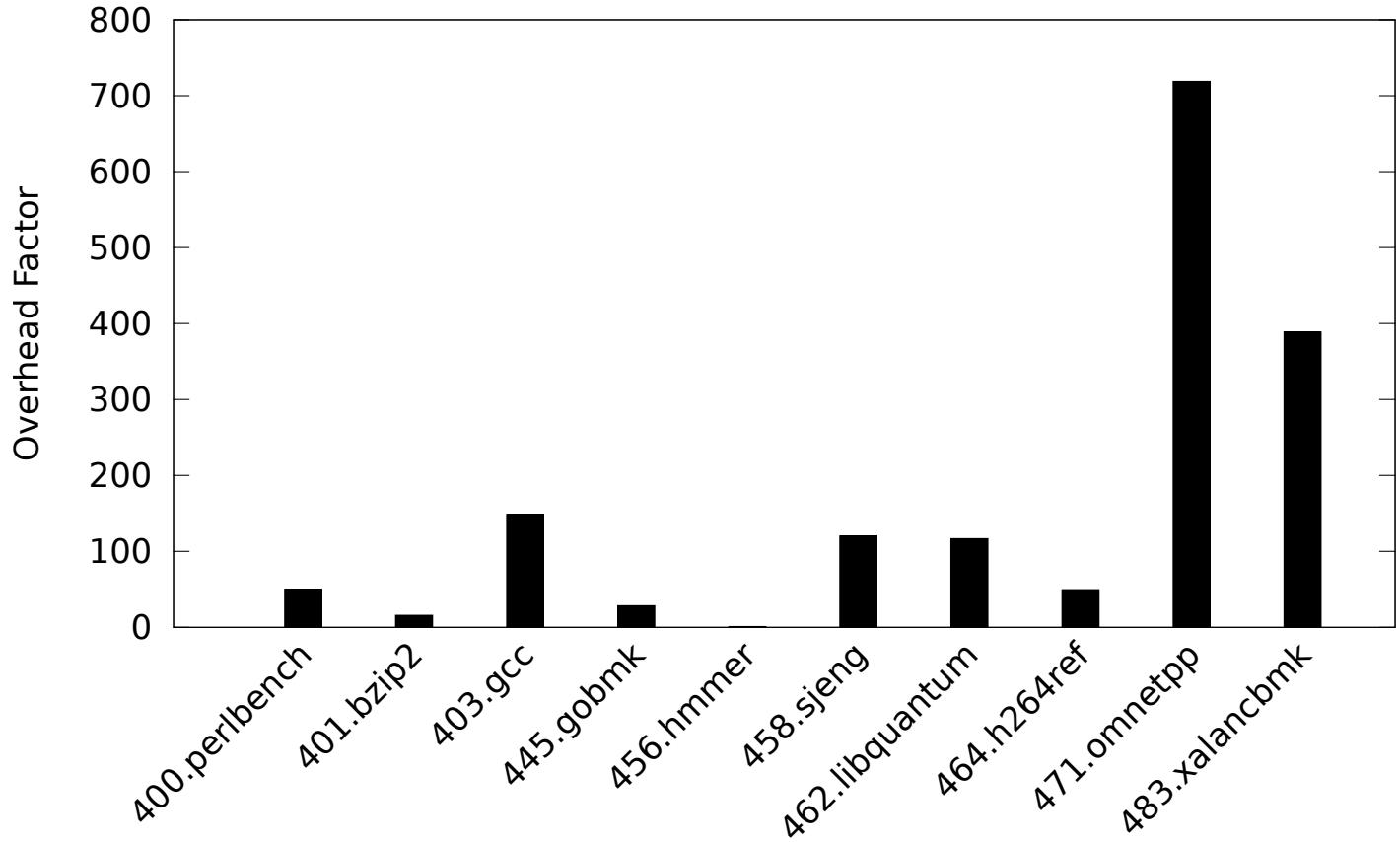
HyperCrypt: Evaluation Setup

System:

- Intel Quadcore CPU (Intel Core i7-2600) running at 3.1GHz
- 8GB of RAM
- Debian Wheezy with base system on top of HyperCrypt

HyperCrypt: Runtime Performance

SPECINT2006 Benchmarks with Sliding Window Size of 1024:



- Overhead of HyperCrypt compared to standard Linux
- Overhead factor between 15 and 148 with three outliers
- 19% performance boost by second chance over FIFO

HyperCrypt: Runtime Performance

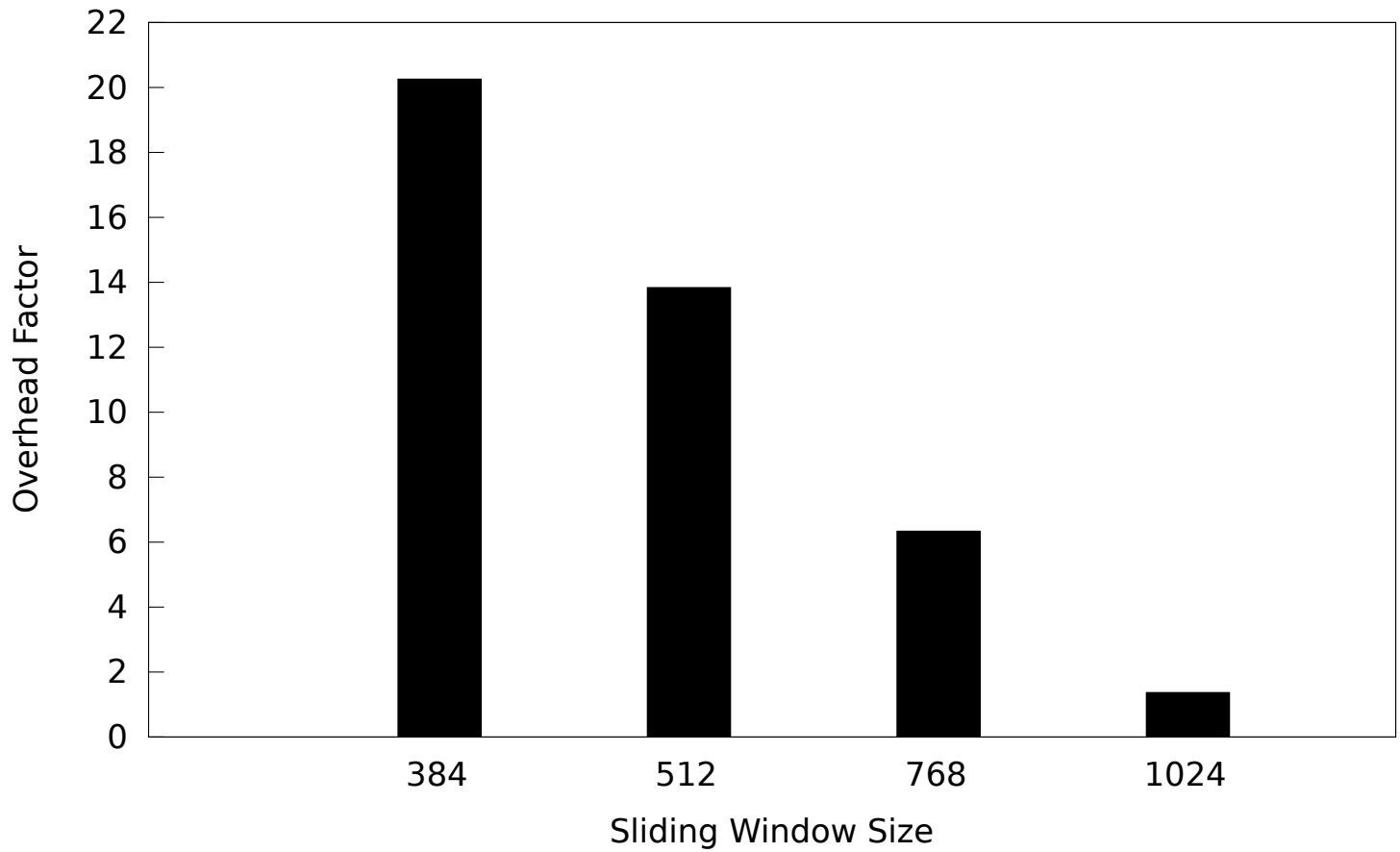
HTTP Server Performance (ngnix web server):

Sliding Window Size	No Encryption	384	512	768	1024
Test duration (s)	81.0	1,641.3	1,120.8	513.5	110.8
Avg. reply rate (replies/s)	123.4	6.1	8.9	19.5	90.2
Avg. connection time (ms)	8.1	164.1	112.1	51.3	11.1

- Httpperf configured to send 10,000 requests
- Avg. reply rate decreases with sliding window size
- Avg. connection time increases with sliding window size

HyperCrypt: Runtime Performance

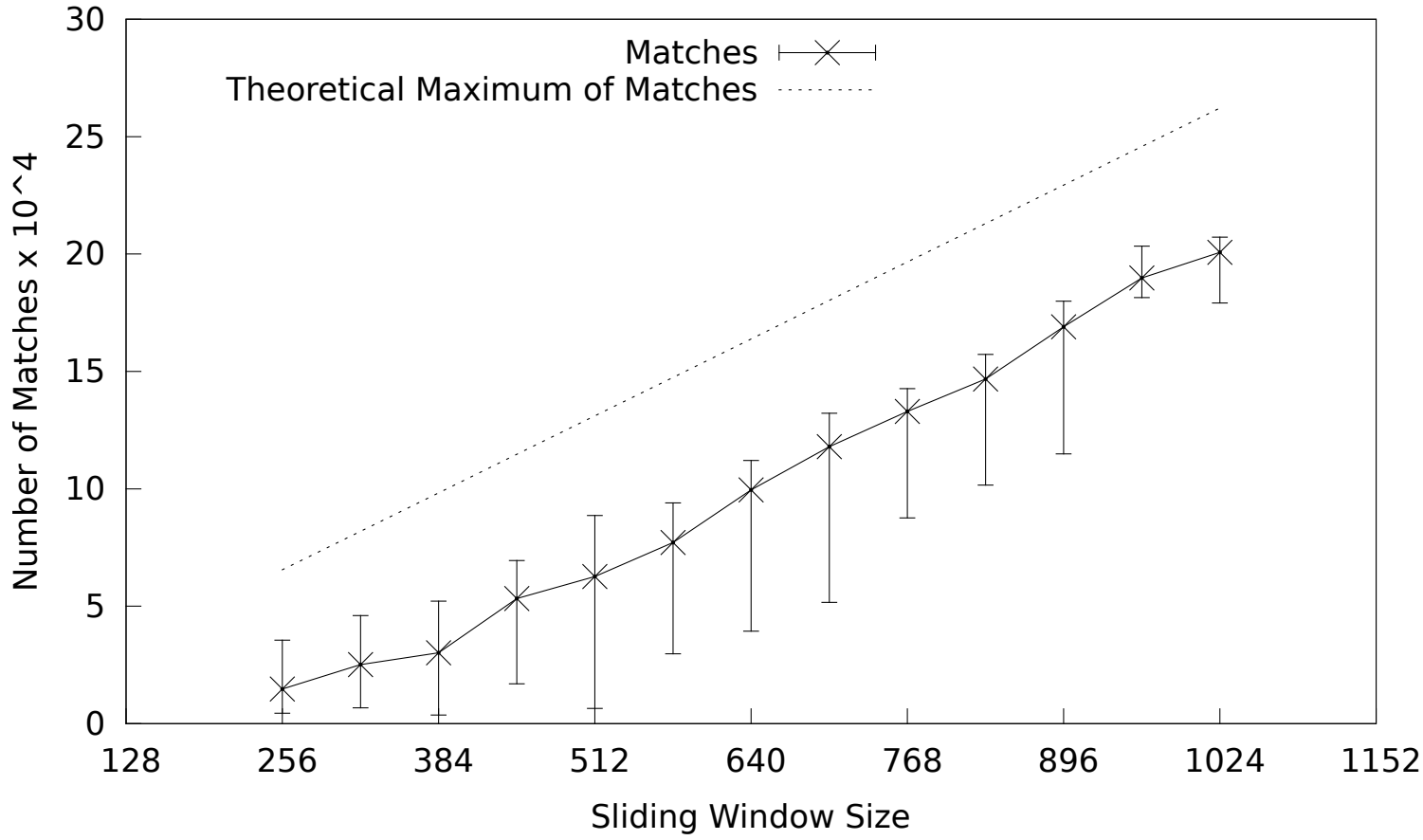
HTTP Server Performance (ngnix web server):



- Overhead of 37% for default sliding window size of 1024
- Overhead factor 6.34 for 768 pages and 13.83 for 512 pages

HyperCrypt: Practical Security Evaluation

Effectiveness:



- Filled 4GB of RAM with random 128 bit pattern
- Searched for the pattern in physical memory after hypercall
- Worst-case scenario (hypercall issued by test program)

HyperCrypt: Practical Security Evaluation

Cache Sizes:

- Cache should be larger than sliding window size
- For our evaluations: Cache: 8MB; default SW size: 4MB
- Unlikely that sensitive information is exposed to RAM at all

$\rightarrow$ Stronger guarantees with Cache as RAM (cf. Coreboot, vCage)

Limitations and Other Approaches

Limitations:

- Performance disaster (SPEC benchmarks)
- Not all devices are supported (BitVisor drivers needed)
- Does not protect against non-physical memory disclosure (swap files, crash reports, vulnerable kernel drivers)

Other Approaches:

- Reducing data lifetime (breaks compatibility)
- Encrypt process address spaces with the help of the OS (kernel space remains exposed)
- Hardware solutions, e.g., Intel SGX (lots of restrictions for software)

Conclusion

Future Work:

- Disable encryption for performance critical tasks
- Turning virtualization on and off during system execution
- Not possible with BitVisor (bare metal hypervisor)

HyperCrypt protects sensitive data within RAM:

- Effectively protects against physical memory disclosure attacks
- Transparently encrypts memory independent of the guest
- Only 37% slowdown for ngnix webserver with default SW size of 1024

Thank you for your attention!

Further Information:

https://www1.cs.fau.de/hypercrypt