# CHIP RED PILL: HOW WE ACHIEVED TO EXECUTE ARBITRARY [MICRO]CODE INSIDE INTEL ATOM CPUS

Mark Ermolov
Dmitry Sklyarov
Maxim Goryachy

# About us

@_markel__

Mark Ermolov

- Not working for Intel
- Positive Technologies
- Lead Expert
- *mermolov[at]ptsecurity.com*

@_Dmit

Dmitry Sklyarov

- Reversing to live
- Positive Technologies
- Head of Reverse Engineering
- *dsklyarov[at]ptsecurity.com*

@h0t_max

Maxim Goryachy

- ex-Positive Technologies
- ex-Huawei
- Independent researcher
- *h0t_max[at]hotmail.com*

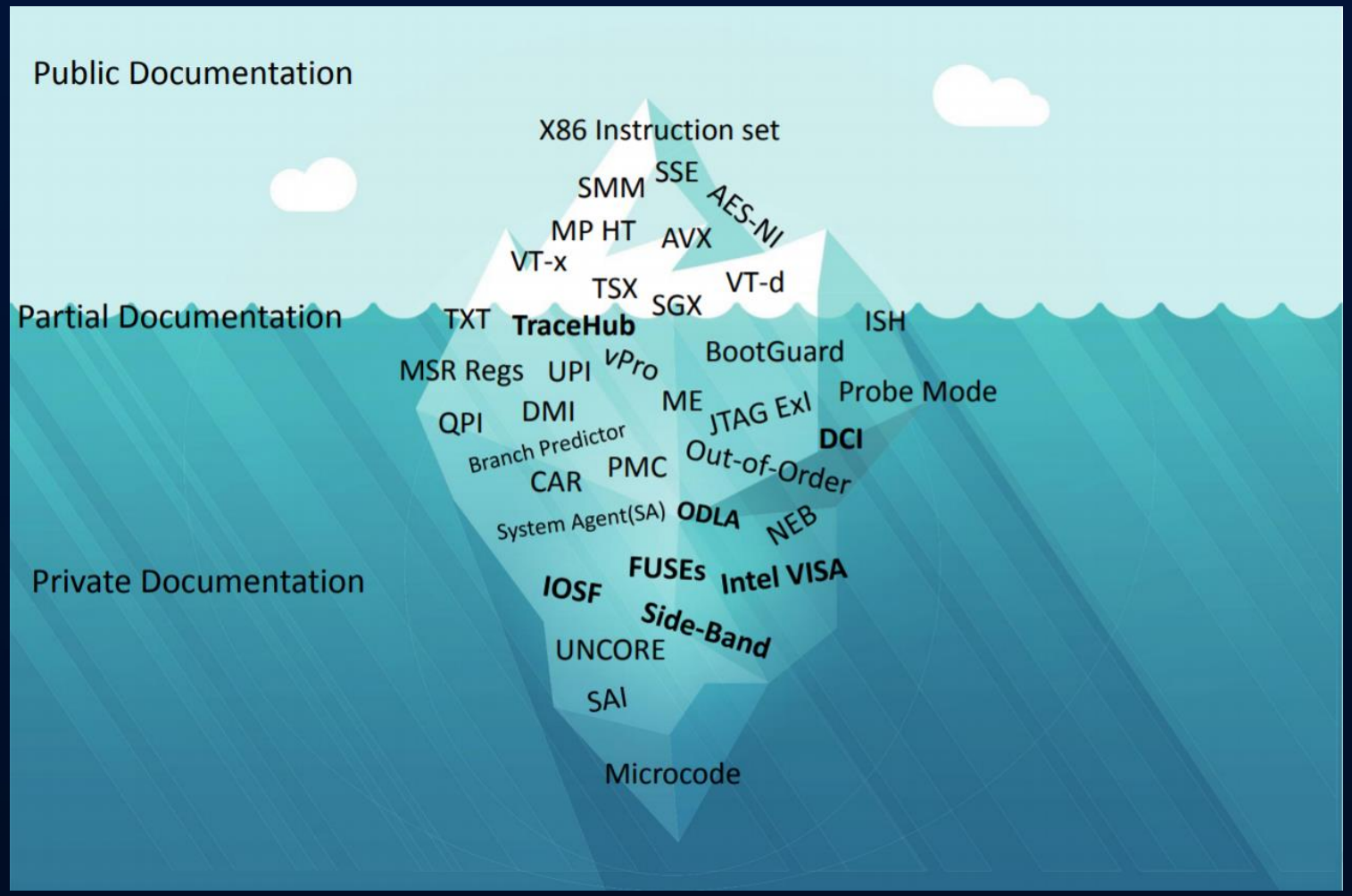
# Agenda

- Microcode Overview
- Access to CPU's internals
- Intel microcode reversing
- Decrypting microcode update

# Agenda

- Microcode Overview
- Access to CPU's internals
- Intel microcode reversing
- Decrypting microcode update

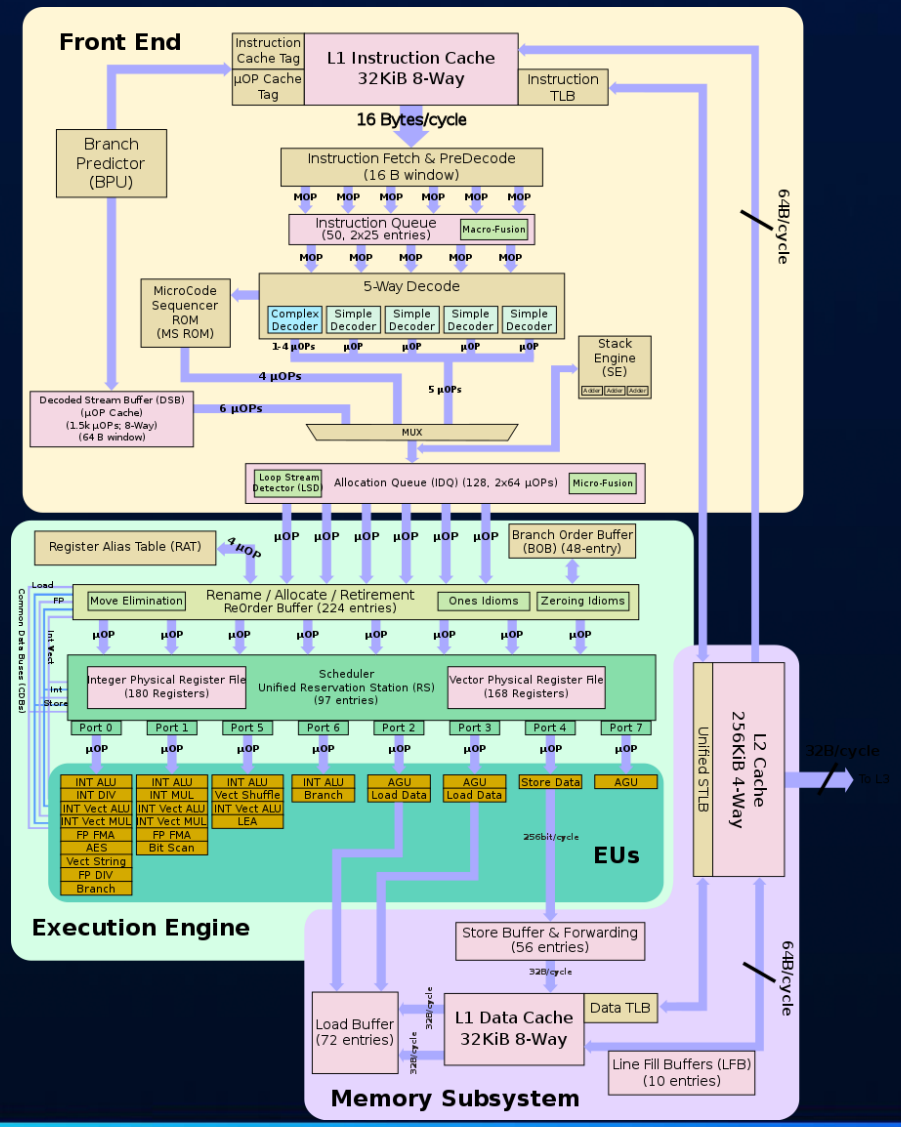
# Deep Intel CPU



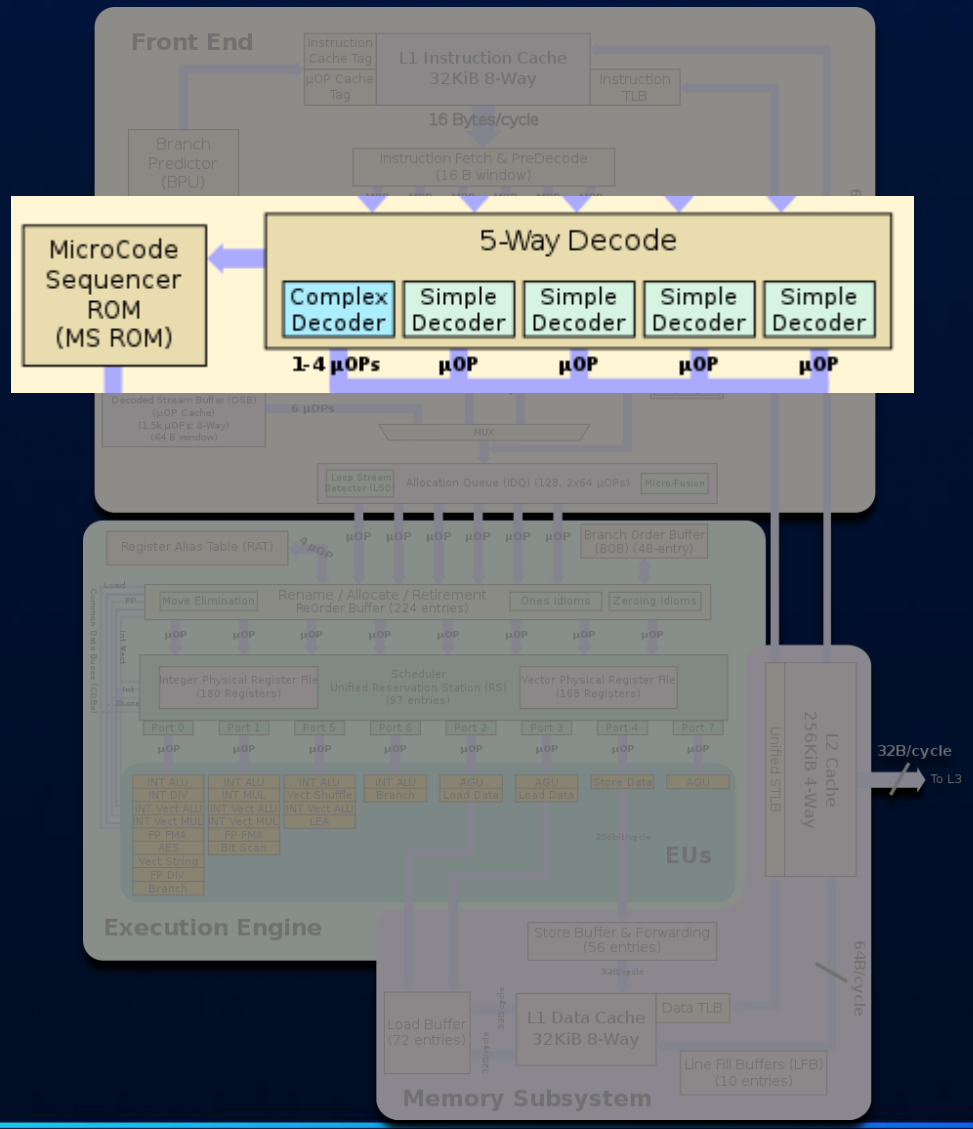
# Microcode Overview

- CPU Core BringUP
- Implements some features (???: SGX, VT-x, MPX, TXT)
- Power Management
- Patches / Update capabilities
- Architecture specific

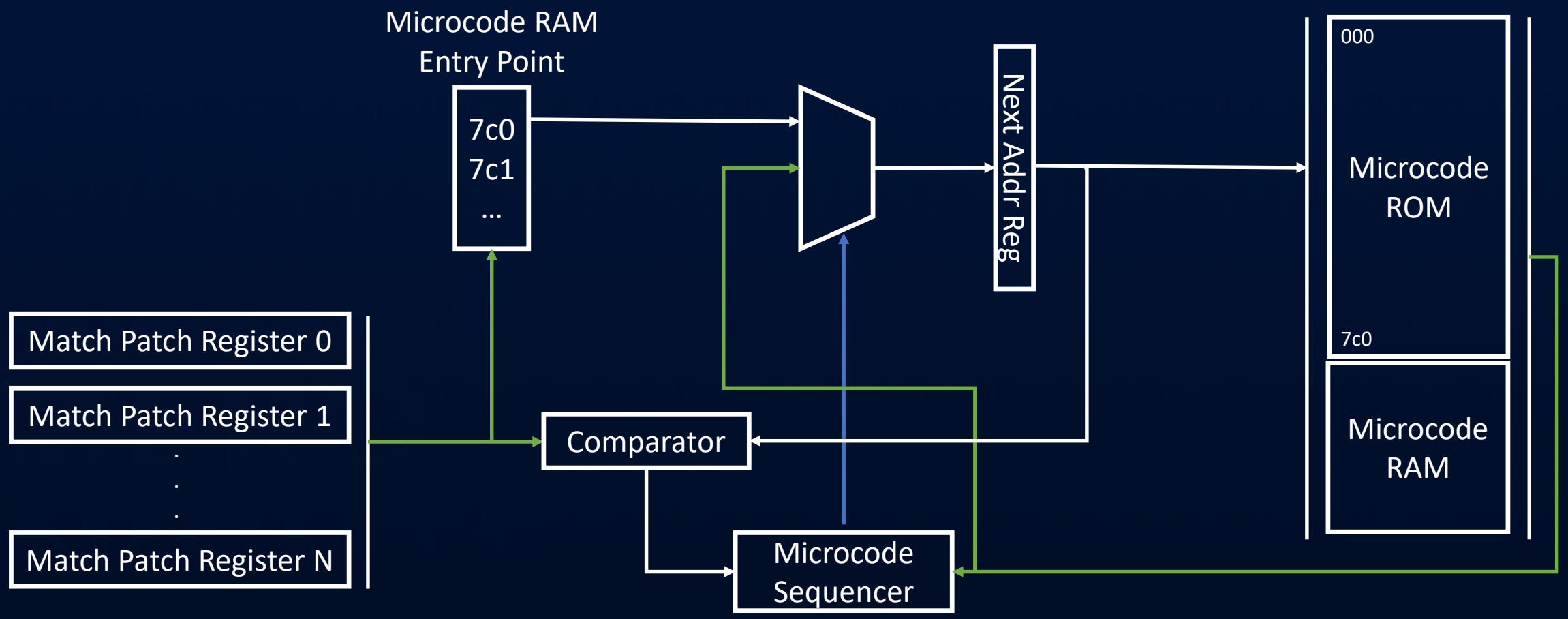
# Intel CPU Core Structure



# Intel CPU Core Structure



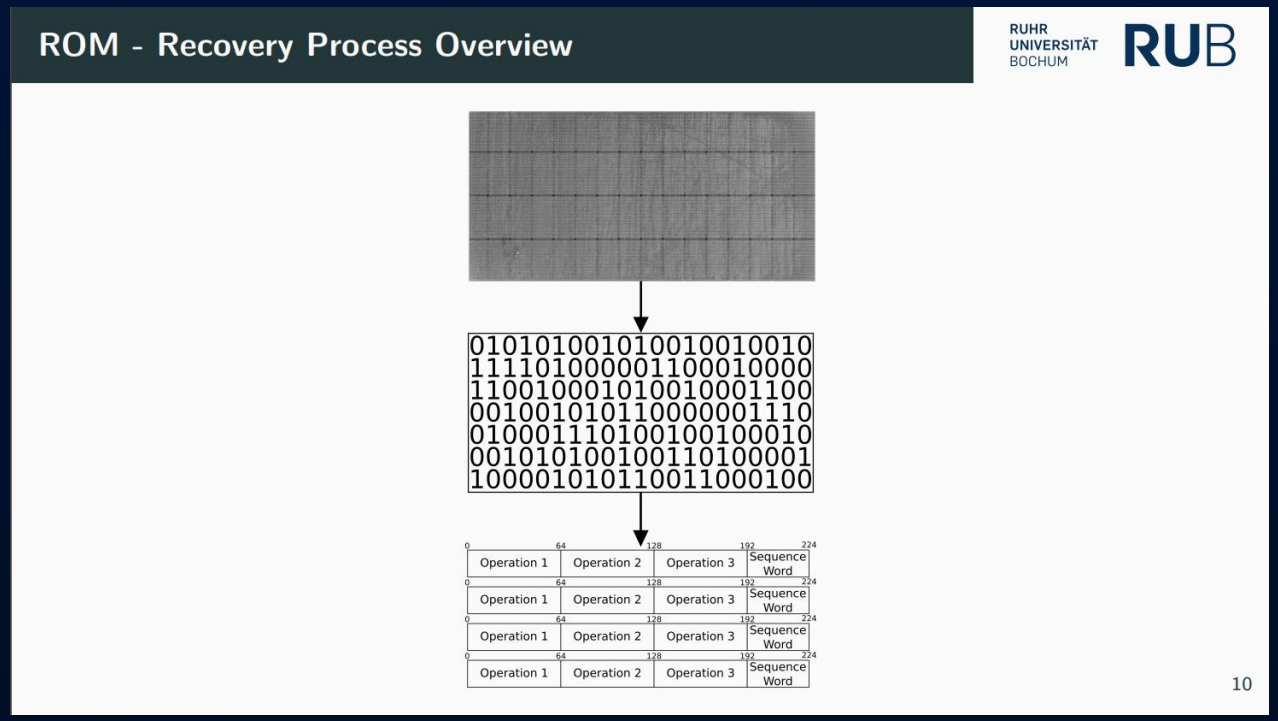
# Microcode Decoder



# Agenda

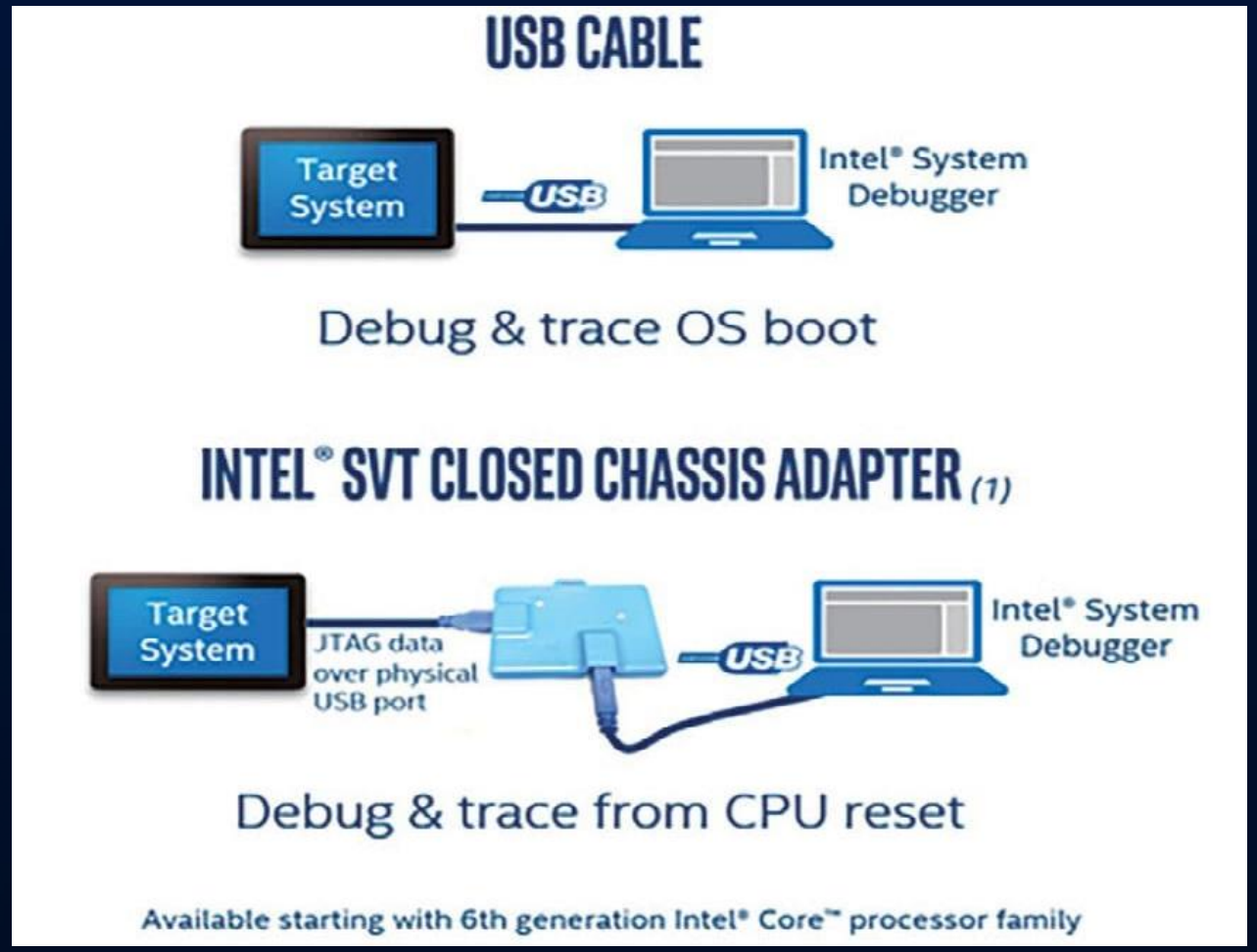
- Microcode Overview
- Access to CPU's internals
- Intel microcode reversing
- Decrypting microcode update

# Research approach: Hardware




*Inside the AMD Microcode ROM - (Ab)Using AMD Microcode for fun and security*

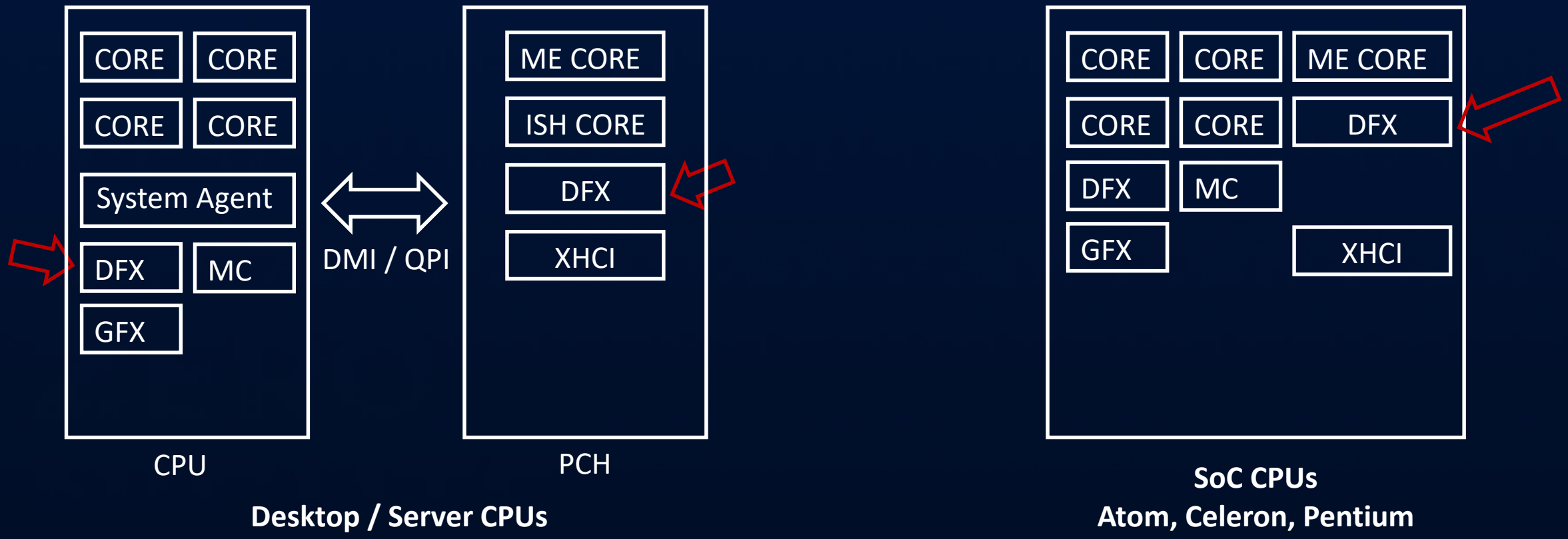
# Research approach: Debugging



# Intel PCH JTAG Unlock

- We've achieved intel *«top secret»* unlock for PCH;
- Debugging Intel Management Engine;
- Intel ME has no microcode core before Ice Lake (2020+);

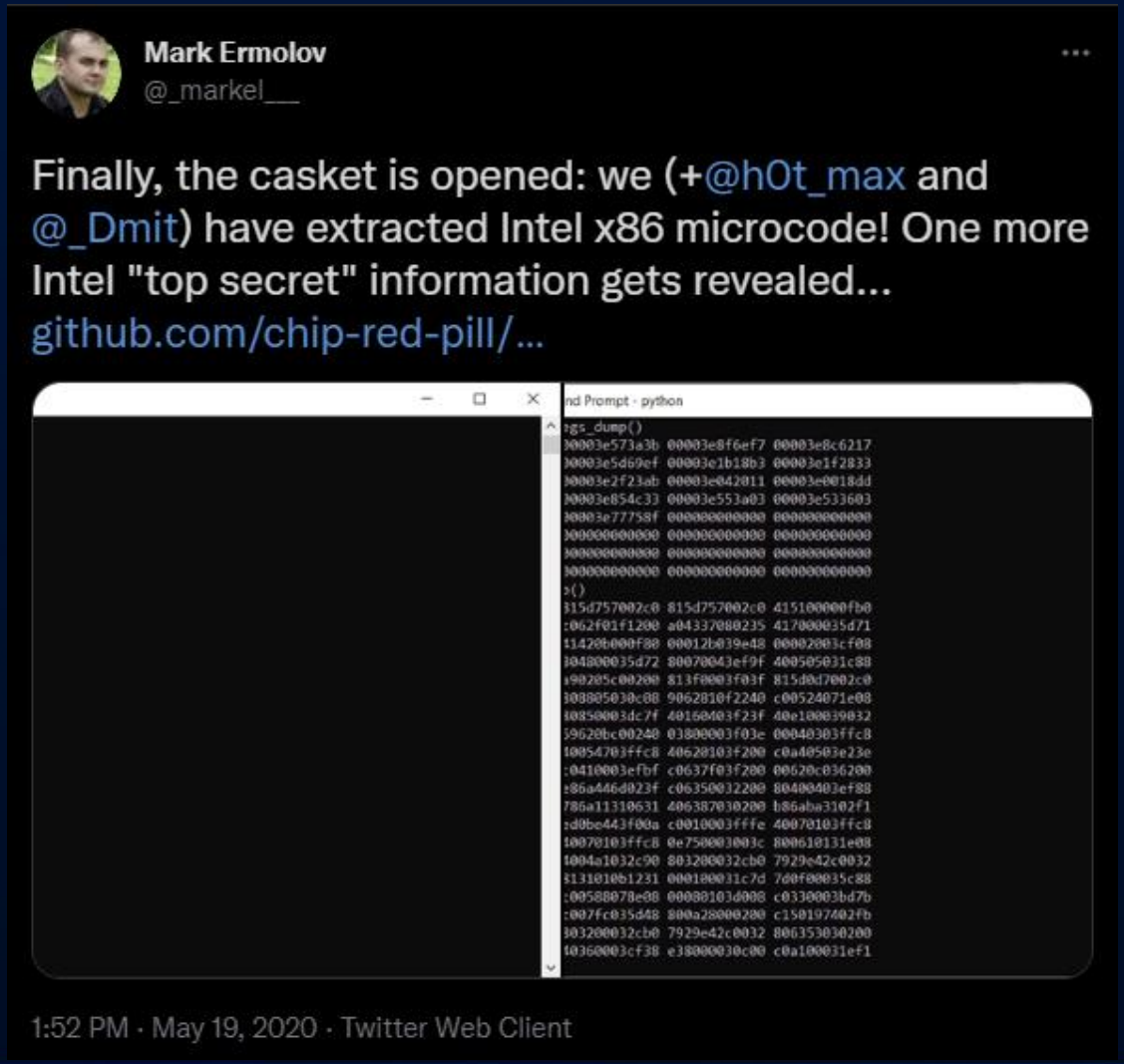

# Intel ME and CPU relationship




**Desktop / Server CPUs**

**SoC CPUs**
**Atom, Celeron, Pentium**

# Intel Atom Microcode Extraction



# Agenda

- Microcode Overview
- Access to CPU's internals
- Intel microcode reversing
- Decrypting microcode update

# LDATs

ARRAY0 — RO

```
0000:  00626803f200 000801030008 004800013000 000000000000
0004:  05b900013000 000a01000200 014800000000 000000000000
...
7ffc:  000000000000 000000000000 000000000000 000000000000
```

ARRAY1 — RO

```
0000:  0000018e5e40 0000018e5e40 0000018e5e40 0000018e5e40
0004:  00000b000240 00000b000240 00000b000240 00000b000240
...
7ffc:  0000018000c0 0000018000c0 0000018000c0 0000018000c0
```

ARRAY2 — RW

```
0000:  0000070000ce 000018201a50 000018201a50 0000384c0600
...
007c:  000000000000 000000000000 000000000000 000000000000
```

ARRAY3 — RW

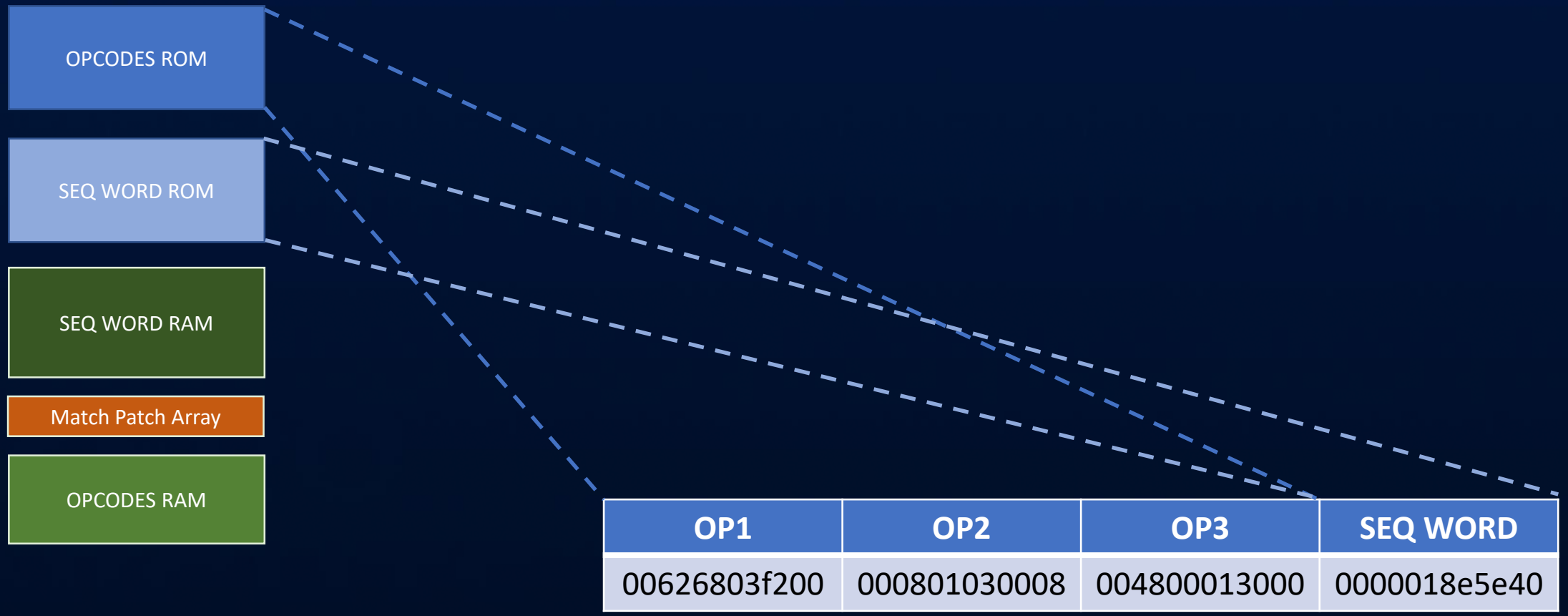
```
0000:  000000000000 00003e5f3a3b 00003e996ef7 00003e966217
...
001c:  000000000000 000000000000 000000000000 000000000000
```

ARRAY4 — RW

```
0000:  c0053d03ffc8 815d857002c0 815d857002c0 415100000fb0
...
01fc:  000000000000 000000000000 000000000000 000000000000
```

https://github.com/chip-red-pill/crbus_scripts

# LDAT and uCode Relationship



# uOps Format

| | 47..46 | 45 | 44 | 43..32 | 31..24 | 23 | 22..18 | 17..12 | 11..6 | 5..0 |
|---|---|---|---|---|---|---|---|---|---|---|
| Field Name | PARITY | M1 | M2 | OPCODE | IMM0 | M0 | IMM1 | DST | SRC1 | SRC0 |
| Size | 2 | 1 | 1 | 12 | 8 | 1 | 5 | 6 | 6 | 6 |

**OPCODE - 12-bit numeric microoperation code of operation**

**SRC0/SRC1/DST - three 6-bits fields which select operands for the operation**

**M0/M1/M2 - bits representing modes of the operation**

**IMM0/IMM1 – represent bits #0-7 and #8-12 of immediate values embedded directly into uops.**

# Sequence Word Format

| | 29..28 | 27..25 | 24..23 | 22..8 | 7..6 | 5..2 | 1..0 |
|---|---|---|---|---|---|---|---|
| Field Name | PARITY | SYNC | UP2 | UADDR | UP1 | EFLOW | UP0 |
| Size | 2 | 3 | 2 | 15 | 2 | 4 | 2 |

**UP0/UP1/UP2** – 2-bit pointers to microoperation inside triad.

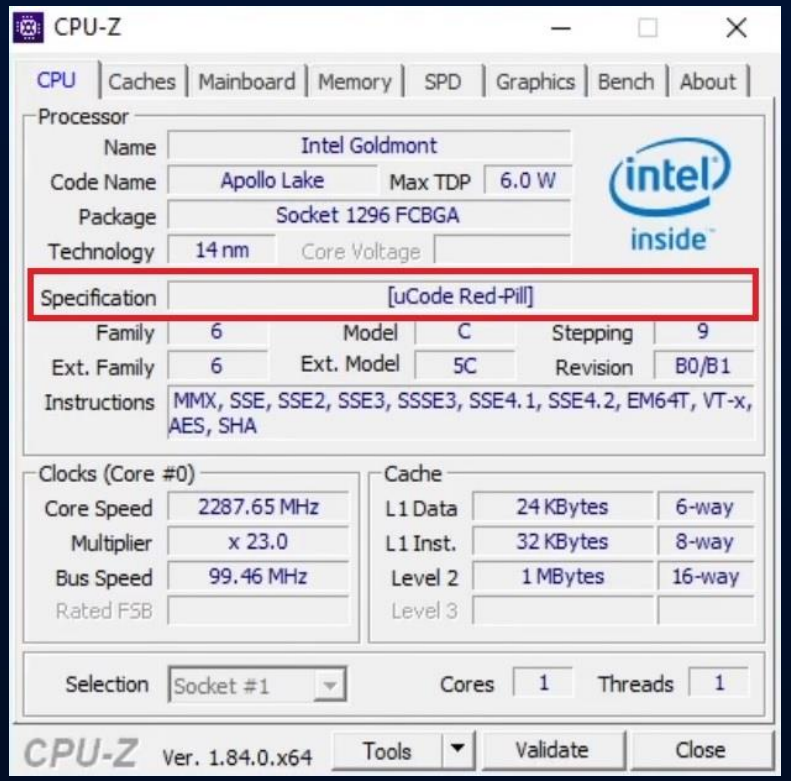
**EFLOW** – 4-bit field that controls execution flow for the microoperations triad.

**UADDR** – 15-bit field that specifies the address in microcode ROM/RAM

**SYNC** – 3-bit field that controls two synchronization aspects those apply for microoperations execution

# uCode Execution

- Find out CPUID[0x8000002.. 0x8000004] leaf entry point
- Develop payload
- Activate CPU RED unlock
- Upload payload to LDAT array[2..4]



# Intel Atom uCode Disassembler

```
U0000: 00626803f200        tmp15:= MOVEFROMCREG_DSZ64(CORE_CR_CUR_UIP)
U0001: 000801030008        tmp0:= ZEROEXT_DSZ32(0x00000001)
       018e5e40            SEQW GOTO U0e5e
-------------------------------------------------------------------------------
U0002: 004800013000        tmp7:= ZEROEXT_DSZ64(0x00000000)
U0004: 05b900013000        mm7:= unk_5b9(0x00000000)
U0005: 000a01000200        TESTUSTATE(UCODE, UST_MSLOOPCTR_NONZERO)
       0b000240            ? SEQW GOTO U0002
U0006: 014800000000        SYNCWAIT-> URET(0x00)
-------------------------------------------------------------------------------
```

https://github.com/chip-red-pill

# Agenda

- Microcode Overview
- Access to CPU's internals
- Intel microcode reversing
- Decrypting microcode update

# Microcode Update

- Runtime update from UEFI/OS
- Encrypted by unknown algorithm
- RSA-2048 signature

# Microcode Update Format

Table 9-8. Microcode Update Format

| 31 | 24 | 16 | 8 | 0 | Bytes |
|---|---|---|---|---|---|
| **Header Version** | | | | | 0 |
| **Update Revision** | | | | | 4 |
| Month: 8 | Day: 8 | Year: 16 | | | 8 |
| **Processor Signature (CPUID)** | | | | | 12 |
| Res: 4 | Extended Family: 8 | Extended Model: 4 | Reserved: 2 | Type: 2 | |
| Family: 4 | Model: 4 | Stepping: 4 | | | |
| **Checksum** | | | | | 16 |
| **Loader Revision** | | | | | 20 |
| **Processor Flags** | | | | | 24 |
| Reserved (24 bits) | | | P7 | P6 | |
| P5 | P4 | P3 | P2 | P1 | |
| P0 | | | | | |
| **Data Size** | | | | | 28 |
| Total Size | | | | | 32 |
| **Reserved (12 Bytes)** | | | | | 36 |
| **Update Data (Data Size bytes, or 2000 Bytes if Data Size = 00000000H)** | | | | | 48 |
| **Extended Signature Count 'n'** | | | | | Data Size + 48 |
| **Extended Processor Signature Table Checksum** | | | | | Data Size + 52 |
| **Reserved (12 Bytes)** | | | | | Data Size + 56 |
| **Processor Signature[n]** | | | | | Data Size + 68 + (n * 12) |
| **Processor Flags[n]** | | | | | Data Size + 72 + (n * 12) |
| **Checksum[n]** | | | | | Data Size + 76 + (n * 12) |

# Decrypt Atom uCode Update: Algorithm

```
5ed5:
    tmp0:= ADD_DSZ8(0x1, tmp0)                          # i := (i + 1) mod 256
    tmp2:= LDPPHYS_DSZ8_ASZ64_SC1(tmp7 + tmp0)          # S[i]
    tmp1:= ADD_DSZ8(tmp2, tmp1)                         # j := (j + S[i]) mod 256
    tmp3:= LDPPHYS_DSZ8_ASZ64_SC1(tmp7 + tmp1)          # swap values of S[i] and S[j]
    STAPPHYS_DSZ8_ASZ64_SC1(tmp7 + tmp0, tmp3)          #
    STAPPHYS_DSZ8_ASZ64_SC1(tmp7 + tmp1, tmp2)          #
    tmp2:= ADD_DSZ8(tmp3, tmp2)                         # (S[i] + S[j]) mod 256
    tmp2:= LDPPHYS_DSZ8_ASZ64_SC1(tmp7 + tmp2)          # K := S[(S[i] + S[j]) mod 256]
    tmp3:= LDPPHYS_DSZ8_ASZ64_SC1(tmp5)                 # *pb
    tmp3:= XOR_DSZ8(tmp2, tmp3)                         #
    STAPPHYS_DSZ8_ASZ64_SC1(tmp5, tmp3)                 # *pb ^= K
    tmp5:= ADD_DSZ64(0x1, tmp5)                         # pb++
    tmp6:= SUB_DSZ32(0x1, tmp6)                         # cb--
    UJMPCC_DIRECT_CONDZ(tmp6, tmp8)                     # if 0 == cb: GOTO tmp8
    SEQWORD GOTO 0x5ed5
```

# Decrypt Atom uCode Update: Plain Text

```
cpu506C9_plat0►  ↓FRO --------                    00000100 Hiew 8.68 (c)SEN
00000:  01 02 00 7C-39 00 0A 00-3F 88 4B ED-C0 00 08 0C  ☺☻ |9 ◙ ?ИКэ└ ◘♀
00010:  0B 01 47 80-00 00 0A 00-3F 88 4F AD-00 03 0A 00  ♂☺GA  ◙ ?ИОн ♥◙
00020:  2F 20 4B 2D-80 02 08 0C-03 22 47 40-A9 03 0A 00  / K-A☻◘♀♥"G@й♥◙
00030:  2F 20 4F 6D-19 02 00 02-03 53 63 80-C0 00 30 02  / Om↓☻ ☻♥ScA└ 0☻
00040:  B8 A6 6B E8-00 00 00 02-03 20 63 C0-00 03 F0 03  ╕жkш   ☻♥ c└ ♥Ё♥
00050:  F8 A6 6B 28-C0 00 08 00-03 C0 0B ED-00 00 0B 10  °жk(└ ◘ ♥└♂э  ♂►
00060:  7F 00 08 00-80 01 31 10-03 00 A1 40-C0 00 31 0C  ⌂ ◘ A☺1►♥ б@└ 1♀
00070:  03 00 07 00-00 00 40 12-0B 30 62 10-00 03 4B 1C  ♥ •   @↕♂0b► ♥K∟
00080:  7F 00 04 40-C0 00 31 12-03 10 24 00-00 00 31 0C  ⌂ ♦@└ 1↕♥►$   1♀
00090:  03 00 01 C0-00 03 08 00-03 C0 0F AD-00 02 00 D2  ♥ ☺└ ♥◘ ♥└☼н ☻ ╥
000A0:  03 20 63 40-A9 03 FD D2-7F FC 84 00-19 00 3D C2  ♥ c@й♥²╥⌂№Д ↓ =┬
000B0:  27 04 00 40-C0 00 08 D0-03 40 08 00-00 00 48 DF  '♦ @└ ◘└♥@◘   H▀
000C0:  03 00 A1 C0-00 03 08 A0-7B A6 3A 83-00 01 08 F0  ♥ б└ ♥◘a{ж:Г ☺◘Ё
000D0:  07 0F A1 40-08 03 3F F2-03 01 01 80-1F 01 0B E0  •☼б@◘♥?Є♥☺☺A▼☺♂р
000E0:  2F 95 08 C0-C0 00 3E E2-03 30 64 C0-00 00 BC EF  /X◘└└ >т♥0d└  ╝я
000F0:  03 00 41 00-00 03 40 B2-0B 2F 62 50-C0 00 3B 02  ♥ A  ♥@▒♂/bP└ ;☻
```

# Next Strike: Intel Gemini Lake + CVE-2018-3643

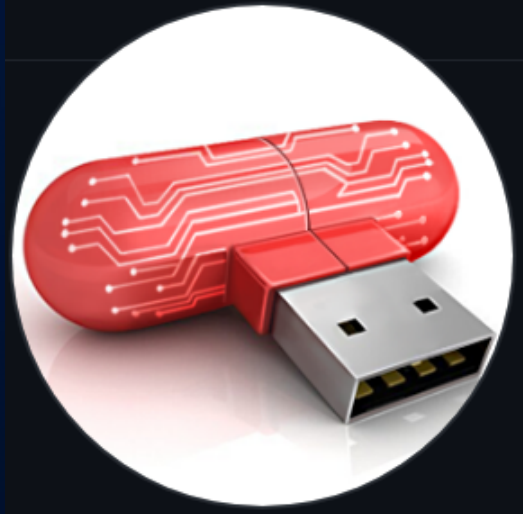

RED UNLOCK

# ELF Inside uCode

```
1  void __fastcall __noreturn main_func(__int64 a1)
2  {
3    unsigned __int32 v1; // eax MAPDST
4    unsigned __int64 v3; // rbp
5    unsigned __int64 v4; // r12
6
7    v1 = __indword(0xCF8u);
8    v3 = __readmsr(0x8Bu);                          // IA32_BIOS_SIGN_ID
9    v4 = __readmsr(0x1A0u);
10   if ( (v4 & 0x400000) != 0 )
11     __writemsr(0x1A0u, v4 & 0xFFFFFFFFFFBFFFFFLL);// IA32_MISC_ENABLE
12   sub_4015EC(a1);
13   if ( (v4 & 0x400000) != 0 )
14     __writemsr(0x1A0u, v4);
15   __writemsr(0x8Bu, v3);
16   __outdword(0xCF8u, v1);
17   __vmx_off();
18 }
```

```
rch\Intel\Microcode\Gemini\uCode.elf
01 01 00-00 00 00 00-00 00 00 00
00 00 00-22 07 40 00-00 00 00 00
00 00 00-40 19 00 00-00 00 00 00
00 38 00-02 00 40 00-05 00 04 00
00 00 00-00 00 00 00-00 00 00 00
00 00 00-00 00 40 00-00 00 00 00
00 00 00-10 19 00 00-00 00 00 00
00 00 00-51 E5 74 64-06 00 00 00
00 00 00-00 00 00 00-00 00 00 00
00 00 00-00 00 00 00-00 00 00 00
00 00 00-10 00 00 00-00 00 00 00
8D 05 A5-16 00 00 48-8D 15 AE 16
B7 16 00-00 F3 0F 6F-2F F3 0F 6F
6F 12 33-D2 F3 0F 6F-19 33 FF F3
66 0F 6F-CD 66 0F 73-F9 04 83 C0
66 0F EF-CD 66 0F 38-00 EC 48 C1
F8 04 66-0F EF C8 66-0F 73 F8 04
66 0F EF-C8 66 0F EF-E9 66 0F 72
6C 16 10-89 C2 83 F8-08 72 B6 48
C5 66 0F-6F D5 83 C1-01 66 0F 73
00 D4 66-0F 6F C8 66-0F EF E8 48
.00000000`00400150:  C1 E0 04 66-0F 73 F9 04-66 0F EF E9-66 0F 73 F9
.00000000`00400160:  04 66 0F 38-DD D3 66 0F-EF E9 66 0F-EF EA 66 0F
.00000000`00400170:  72 F3 01 F3-0F 7F AC 06-90 00 00 00-89 C8 83 F9
```

# Chip Red Pill



Overview | Repositories 5 | Projects | Packages

## uCode Research Team

chip-red-pill

Follow

Research Team Members: Dmitry Sklyarov (@_Dmit), Mark Ermolov (@_markel__), Maxim Goryachy (@h0t)

Popular repositories

**uCodeDisasm**
Python 238 24

**glm-ucode**
GLM uCode dumps
233 37

**crbus_scripts**
IPC scripts for access to Intel CRBUS
Python 105 22

**IntelTXE-PoC**
Forked from ptresearch/IntelTXE-PoC
Intel Management Engine JTAG Proof of Concept
Python 61 13

**udbgInstr**
C++ 59 7

https://github.com/chip-red-pill/

# Intel Feedback

## Bug Bounty Bonus: Pentium®, Celeron®, and Intel Atom® Processors

Intel is announcing a new bonus incentive to our bug bounty program, focusing on firmware and hardware within Intel® Pentium®, Intel® Celeron®, and Intel Atom® processors (see below for full platform listing). This bonus incentive will be open to the public for a period of one year, May 11, 2021 - May 10, 2022 and will pay up to $150,000.00 for novel vulnerabilities (1.5x the normal maximum). Additionally, at the end of the one-year period, the top 10 submissions will be identified and recognized, and the top two researchers will be invited to speak (Virtually) at iSecCon (Intel's internal security conference).

Bonus incentive open to the public –submissions must be received by 11:59pm PST on May 10, 2022 to be eligible for the bonus incentive. Submissions received after that date are not eligible for the bonus incentive but may be eligible under Intel's standard bug bounty program.

Bonus incentive award payout will be multiplier ranging from 1.2-1.5 the standing Bug Bounty payment. (See quick look chart below)

| Vulnerability Severity | Intel Bug Bounty Bonus Firmware | Intel Bug Bounty Bonus Hardware |
|---|---|---|
| Critical | Up to $45,000 | Up to $150,000 |
| High | Up to $21,000 | Up to $42,000 |
| Medium | Up to $3,900 | Up to $6,500 |
| Low | Up to $1,200 | Up to $2,400 |

# Conclusion

- We’ve obtained access to Intel “top secret” debugging level at Apollo and Gemini Lake CPUs family
- We’ve recovered the most part of microcode opcodes for Apollo Lake
- You don’t need physical access for microcode modifying – needs only OS-level code execution and write access to SPI flash
- We developed Intel Atom uCode disassembler and unlock guide - Chip Red Pill (https://github.com/chip-red-pill)

# QUESTIONS?