IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received July 14, 2020, accepted September 10, 2020, date of publication September 15, 2020,
date of current version September 25, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3024243

Reverse Engineering of Intel Microcode Update Structure

ZHIGUO YANG, QINGBAO LI, PING ZHANG, AND ZHIFENG CHEN
State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China
Corresponding author: Zhifeng Chen (xiaohouzi06@163.com)

ABSTRACT Microcode update mechanism have been widely used in modern processors. Due to the implementation details are not public, researchers are prevented from gaining any sort of further understanding currently. The microcode update binary which uploaded into Central Processing Unit (CPU) is the only accessible node in this update chain by researchers, but previous manual reverse analysis for a small amount of microcode updates has the disadvantages of incomplete coverage, slow speed, and low accuracy. Therefore, we first build a Sample Repository containing 504 Intel official microcode updates, then propose a semiautomatic analytical method named SJNW-MA to analyze samples. This work has the following merits: (1) automatic methods of similarity analysis and candidate feature mining improve the speed; (2) manual-assisted analysis based on expert knowledge can filter important features, to avoid redundant features or valuable common data blocks missing; (3) analysis for 504 microcode updates make the results of reverse engineering are more complete. Finally, we extract eleven structures of Intel microcode updates and group them into four categories. In addition, we also identify and describe some new metadata in microcode updates of the third and the fourth category, including a new 3072-bit *RSA Modulus* as well as corresponding *RSA Exponent* which indicates upgrade of security technology inside update mechanism.

INDEX TERMS Microcode update, reverse engineering, central processing unit.

I. INTRODUCTION

Processor manufactures have introduced microcode into CPU interior to achieve greater performance and efficiency since the 1970's. Although microcode was initially implemented in Read-Only Memory (ROM), an update mechanism was introduced by means of Random-Access Memory (RAM) to implement dynamic debugging capabilities and correcting processor errata, especially after the infamous Intel Pentium FDIV bug of 1994.

Intel has supported the ability to apply stability and security updates to the CPU in the form of microcode updates for well over decades. In fact, once erroneous CPU behavior is discovered, manufacturers publish a microcode update immediately. The most famous is that Intel, OEM of Motherboard and Operating System successively delivered solution for Spectre and Meltdown vulnerabilities [1], [2] by microcode updates in 2018. On the basis of microcode updates, processor manufacturers obtain flexibility and reduce costs of correcting erroneous behavior. But due to the volatility of RAM, microcode updates are not persistent and have to be reloaded after each processor reset. The normal preferred method to apply microcode updates is using the system Basic Input Output System (BIOS), but for a subset of Intel's processors this can be done at runtime using the operating system [3].

Although microcode update mechanism can be leveraged to rectify some erroneous behavior of CPU, it is not a panacea. Microcode updates would degrade performance due to additional condition checks and they cannot be applied in all cases, such as some complex design errors of hardware. Ironically, K7 processor of AMD fails to properly validate RAM where microcode update is loaded in during built-in self-test (BIST), causing the microcode update mechanism itself to be listed as a processor erratum [4].

More unfortunately, CPU vendors keep information about all microcode secret including update and work principle of microcode update. Publicly available documentation and patents merely state vague claims about how real-world microcode might actually look like, but provide little other meaningful insight [5]. Although no vulnerability, backdoor, or error is discovered up to now, it is not enough to prove the reliability and security of the microcode update mechanism.

The associate editor coordinating the review of this manuscript and approving it for publication was Zhitao Guan.

Instead, Because the implementation details of microcode update mechanism are not public, researchers are prevented from gaining any sort of further understanding currently, which means that it is impossible to study the update to clearly establish whether any security issues are being fixed like what developer or vendors claimed. That Intel and AMD are reducing their internet resource of microcode updates seems to indicate that vendors still try to strengthen technology protection and microcode update security through closed-source instead of cooperation.

We try to conduct an in-depth analysis for the structure of microcode update binary which uploaded into CPU to provide basis for researchers who focus on this field to do some further research, because they are the only accessible node in this update chain by researchers now. In fact, vendors deliberately hide details considering security and patents, resulting in less research results and a lack of knowledge in this field. The structure of microcode update has been documented and published in Intel software developer's manual (as shown in Fig.1), which consists of three parts: *Header*, *Data* and *Extended Signature Table* (supported since 0F_03H). However, the *Extended Signature Table* is not appeared in all collected microcode updates, which also verified the work of Chen and Ahn [6].

It seems that the structure of microcode update is very clear at present, especially the documented metadata of *Header* and *Extended Signature Table*, but in fact other metadata has been found in *Data*. Hawkes has found that there is a 96-byte prefix and other encryption-related fields in *Data* of some microcode updates [7]. In addition, because *Data Size* in *Header* must be a multiple of 4 bytes while *Total Size* must be a multiple of 1024 bytes, which indicates that there may be random padding data in *Data*, i.e. *Data* is not all valid data.

A key to solve challenges and difficulties of research without knowledge or basis is collecting enough microcode updates and filtering some features or criterions that are important for classification and structure reversion from them. First, we build a Sample Repository of microcode updates from Intel. Above of all, a semiautomatic analytical method called SJNW-MA is proposed to analyze samples in the Sample Repository, which helps us select five features and other interesting common data blocks for classification and analysis. Finally, we complete reverse engineering based on summary for the results from the above method, and list the possible structures of Intel microcode updates as well as some significant information inside update mechanism. In summary, our main contributions in this article are as follows:

(1) **Sample Repository** of Intel microcode updates containing 504 samples is built, capacity of which is more than existing repository (MCE DB r153, 411) supported by *CPUMicrocodes* project [8], and those samples supported to be uploaded into CPU by loader.

(2) **SJNW-MA** semiautomatic analytical method for samples is proposed to improve efficiency, accuracy, and lower-granularity. Compared to manual analysis [6], it can search more common data blocks, among of them are used as features for classification.

(3) Analysis for 504 samples in the Sample Repository is completed. The results of reverse engineering, such as 11 structures and some significant discoveries inside Intel microcode updates, are explained in detail in chapter 4, which provides basis for follow-up researchers to continue to expand research.

(4) Especially, some new metadata of *Data* is identified and described in some specific structures, such as *Flexible Field 1*, *Flexible Field 2*, etc. In addition, 24 public keys of RSA used for signature are determined which is 12 more than that found by Hawkes [7], and the implement of recovering the originally signed data is also finished.

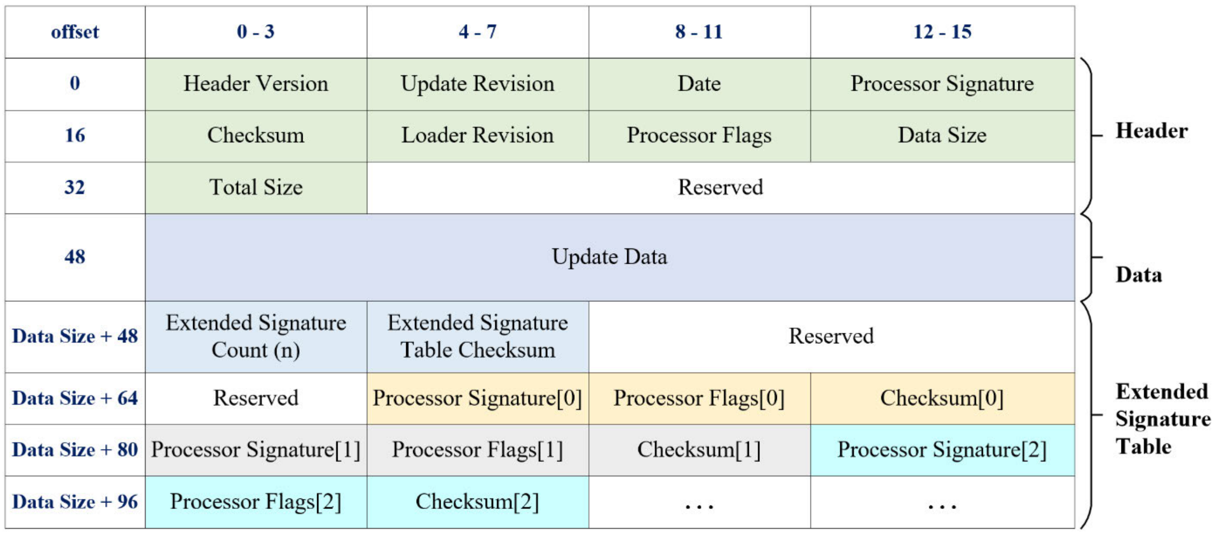
(5) A new structure is extracted and a 3072-bit *RSA modulus* which used for signature is identified in updates with this structure. Although not stored in updates, we determine that the value of *RSA Exponent* is 65537 (10001H) by enumeration method and recover the original signed data successfully.

II. RELATED WORK

Although Wilkes first introduced the concept of microcode into computer engineering as early as 1951 [9], researchers have paid most attention to diverse branches of research related to micro-programming and optimization of microcode for a long time [10]–[12]. No one had realized that microcode may be threatening until Thompson proposed the concept of a malicious compiler which is undetectable even by source code analysis in 1984 [13]. Regrettably, likely due to the proprietary nature of processor microarchitecture and microcode, little published work has analyzed processor microcode from a security perspective.

There has been some reverse analysis of microcode updates. Molina and Arbaugh published the metadata accompanying Intel microcode updates in 2000, determining the purpose of certain fields within the microcode update header [14], but Intel published details of microcode update header on their software developer's manual [15] after soon. Likewise, an anonymous report published in 2004 provided similar information about microcode updates of AMD [16]. Hawkes published a technical report in 2013 to disclosure the presence of additional metadata within *Data* of Intel microcode updates, suggesting that RSA signature with a non-standard Secure Hash Algorithm (SHA) is used for verification of some microcode updates [7]. Chen's work [6] attempted a high-level security analysis for microcode updates for both Intel and AMD, which guiding some ideas in this article. Although Triulzi presented that he had been able to patch the microcode of AMD K8 microarchitecture at TROOPERS 15 and 16 [17], [18], no details of his reverse engineering nor the microcode encoding is published. A real landmark work is Proof-of-Concept Microprograms of AMD K8 and K10 microarchitecture completed by Koppe et al in 2018 [5], [19].

Overall, few studies of microcode update security seem to have attracted little experts' attention on this field, and the



offset	0 - 3	4 - 7	8 - 11	12 - 15	
0	Header Version	Update Revision	Date	Processor Signature	Header
16	Checksum	Loader Revision	Processor Flags	Data Size	
32	Total Size	Reserved			
48	Update Data				Data
Data Size + 48	Extended Signature Count (n)	Extended Signature Table Checksum	Reserved		Extended Signature Table
Data Size + 64	Reserved	Processor Signature[0]	Processor Flags[0]	Checksum[0]	
Data Size + 80	Processor Signature[1]	Processor Flags[1]	Checksum[1]	Processor Signature[2]	
Data Size + 96	Processor Flags[2]	Checksum[2]	. . .	. . .	

FIGURE 1. The structure of Intel microcode update documented in Intel software developer's manual. The size of Header is fixed at 48 bytes. Note: the triads (*Processor Signature[i], Processor Flags[i], Checksum[i]*) in the Extended Signature Table will be used by utility software to decompose a microcode update into multiple microcode updates where each of the new updates is constructed without the optional *Extended Signature Table*.

intensifying processor security issues also have not strongly promoted the research on the effectiveness and security of microcode updates.

III. METHOD AND EXPERIMENT

In this chapter, we will describe implementation details of the Sample Repository and the SJNW-MA method. First, how to build the Sample Repository of Intel microcode updates is described in section 3.1. Then, the SJNW-MA method for analysis of samples is introduced in section 3.2. Finally, the process and results of experiment is described in section 3.3. Although we only analyze available microcode updates for Linux operating system, the data on microcode_ctl, BIOS or other operating systems is identical.

A. SAMPLE REPOSITORY OF MICROCODE UPDATES

There are two types of available microcode update packages for Linux operating system from Intel official distribution. DAT files for older version were released from around 2000, which with ASCII or UTF-8 encoded text format and big-endian endianness. Multiple independent microcode updates are encapsulated in one DAT file. The little-endian binary files containing a single microcode update have been applied since 2018. So, we would make treatment for microcode update packages to obtain microcode update samples with uniform naming and format, which facilitates the reverse analysis of structure and application later. As shown in Fig.2, the workflow of packages processing is as follows:

(1) **Extract**: This step as well as (2) Format Transform is only executed for DAT files. Because storing multiple microcode updates and other notation, such as copyright and title, DAT files must be parsed to extract independent microcode updates.

(2) **Format Transform**: The process of Format Transform can restore microcode update to binary from DAT, which also ran at official microcode update loading.

(3) **Deduplicate**: Since microcode update packages which released in various periods often stores the same microcode updates, Deduplication is necessary to avoid duplicate microcode updates in the Sample Repository.

(4) **Rename**: Renaming rule must make sure that name of microcode update samples is unique and unambiguity, thus *Processor Signature*, *Processor Flags* and *Update Revision* are chosen to make up name of samples in the layout of *Processor Signature_Processor Flags_Update Revision*.

Processor Signature, a 32-bit words, is a unique representative of the CPU hardware model that the microcode update will apply to; *Processor Flags*, also called *Platform ID*, is used to determine ''slot'' information for the processor and the proper microcode update to load [15]; *Update Revision* indicates version information of the microcode update. All of these can be obtained from microcode update self. Both of *Processor Signature* and *Processor Flags* are strongly associated with checking compatibility of microcode update, encoding of which is shown as Fig.3.

(5) **Checksum Check**: Finally, Checksum would be checked to verify the integrity of the microcode update. Checksum is correct when the summation of all the DWORDs that comprise the microcode update result in 0H. If checksum passes, the update will be stored into the Sample Repository, while the error update will be logged and abandoned.

In addition, considering the strong coupling between microcode updates and hardware, the samples are labeled with *DisplayFamily_DisplayModel*, which referring to the description of the correspondence between *Processor Signature* and processor products in the Intel technical manual [20]. The values of *DisplayFamily_DisplayModel* could

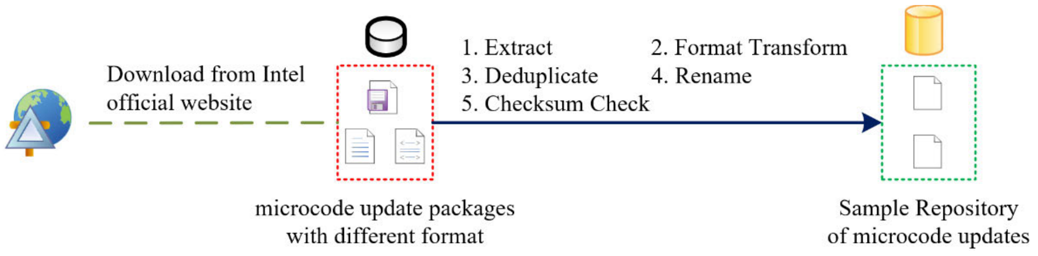
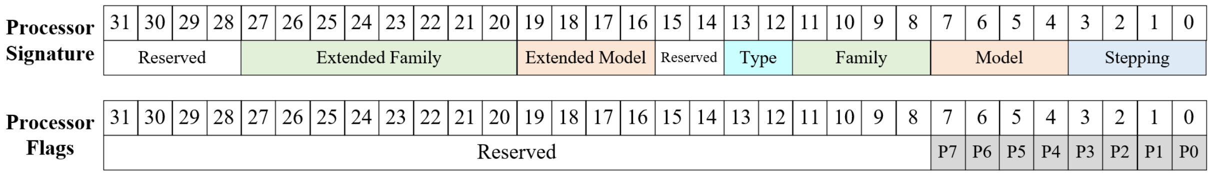

Download from Intel official website
1. Extract
3. Deduplicate
5. Checksum Check
2. Format Transform
4. Rename
microcode update packages with different format
Sample Repository of microcode updates

FIGURE 2. The collection and processing of microcode update packages. We download Intel microcode update packages from Intel download center (closed) and project supported by Intel in Github to ensure credibility.



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Processor Signature	Reserved				Extended Family								Extended Model				Reserved		Type		Family				Model				Stepping			

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Processor Flags	Reserved																								P7	P6	P5	P4	P3	P2	P1	P0

FIGURE 3. Encoding of ***Processor Signature*** and ***Processor Flags*** in microcode update.

be extracted from fixed fields in *Processor Signature* (as shown in Fig.3). It is worth mentioning that the binary files released since 2018 are also named after *DisplayFamily_DisplayModel_Stepping*, but other fields such as *Stepping* only represents small change within the specific processor model. If introducing them into label, the results will be more complicated, which is not convenient for manual-assisted analysis later.

B. SJNW-MA SEMIAUTOMATIC ANALYTICAL METHOD

Further research will be based on the following reasonable assumptions: At least, the structures of microcode updates applied to processors within the same product model should have similarity or even share some common data [6], [7], [15]. To this end, an SJNW-MA semiautomatic analytical method aimed to similarity calculation and feature selection is proposed, the results from which are finally used to achieve effective samples classification. The complete framework of SJNW-MA method is shown as Fig.4, which consists of three stages:

(1) At SJ stage (section 3.2.1), Jaccard index between two samples is calculated based on k-shingling algorithm, and stored in Jaccard index matrix to characterize the similarity between any sample pairs. Meaningfully, Jaccard index matrix and samples can be regarded as adjacency matrix and node of graph respectively.

(2) At NW stage (section 3.2.2), during traverse of Jaccard index matrix based on Depth First Search (DFS) algorithm, the best matching sequence of two nodes (i.e. samples) with edge is determined using Needleman-Wunsch algorithm, which then is separated into multiple common data blocks as candidate features.

(3) At MA stage (section 3.2.3), researchers conduct manual-review for candidate features (common data blocks) from NW stage. The main purpose is to slicing, deduplicate and evaluate, and finally obtain an efficacious and simplified feature set for sample classification.

1) SIMILARITY ANALYSIS BASED ON JACCARD INDEX

The Jaccard index, also known as Intersection over Union and the Jaccard similarity coefficient (originally coined *coefficient de communauté* by Paul Jaccard), is a statistic used for comparing the similarity and diversity of sample sets [21]. The Jaccard index measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{1}$$

Obviously, the value range of Jaccard index is [0, 1] and higher value means higher similarity between two sets. 0 means that set A is completely different from set B and 1 means that set A is exactly same as set B, while set A and set B are both empty sets, Jaccard index of A and B is 1.

The calculation objects of Jaccard index are sets, so, the data of little-end binary samples must be converted to sets using k-shingling algorithm [22], [23], which is also the segmentation method commonly used in document similarity research. Considering that the size of field which represent certain information independently in microcode updates is 4 bytes, i.e. the smallest unit of microcode updates may be 4 bytes, we take the value of k as 4. The complete calculation algorithm is shown in Listing.1. It must be stated that *Header* of microcode update (the offset is 0 to 48 bytes) would not be processed and analyzed because has been disclosed.

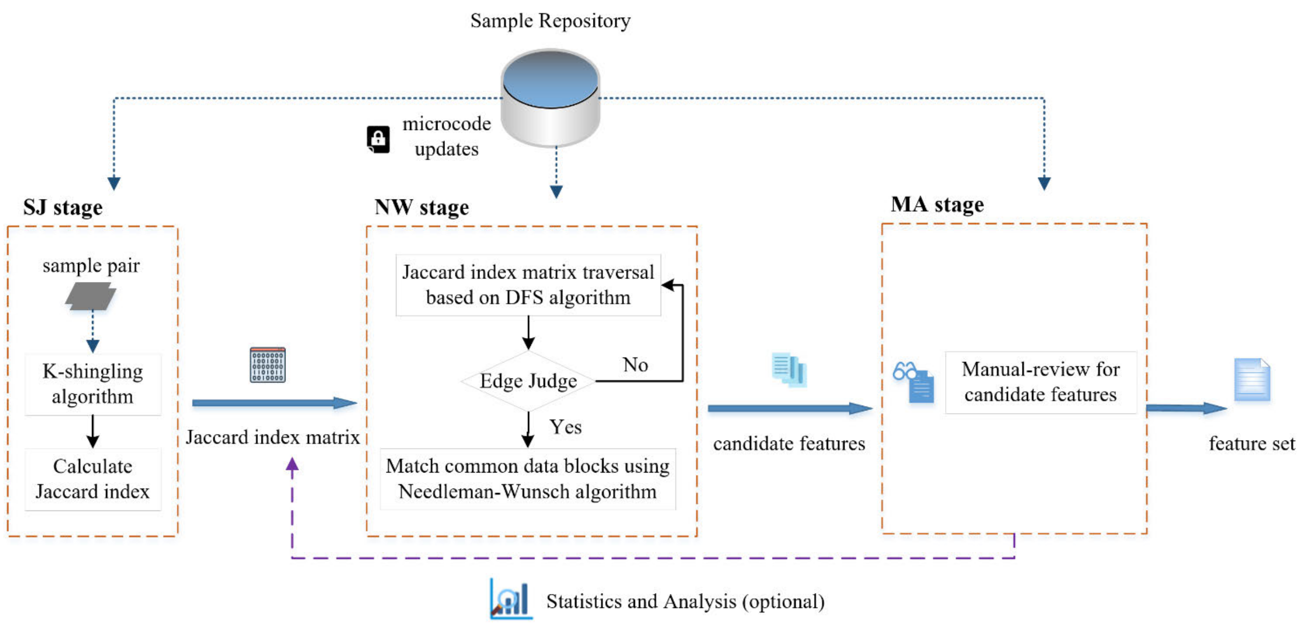

Sample Repository
microcode updates
SJ stage
sample pair
K-shingling algorithm
Calculate Jaccard index
Jaccard index matrix
NW stage
Jaccard index matrix traversal based on DFS algorithm
Edge Judge
No
Yes
Match common data blocks using Needleman-Wunsch algorithm
candidate features
MA stage
Manual-review for candidate features
feature set
Statistics and Analysis (optional)

FIGURE 4. **Framework of SJNW-MA method.**

```
Algorithm 1: the calculation of Jaccard index
Input  : samples in Sample Repository
Output: Jaccard index matrix
1. k = 4
2. Jaccard_matrix[N][N]
3. while sample_A, sample_B in Sample Repository do
4.     if (Jaccard index of sample_A and sample_B is not calculated)
5.         set_A = set(k-shingling(sample_A))
6.         set_A = set(k-shingling(sample_B))
7.         if (set_A is not empty set and set_B is not empty set)
8.             J(A, B) = intersection(set_A, set_B) / union(set_A, set_B)
9.             Jaccard_matrix[sample_A][sample_B] = J(A, B)
10.        else
11.            throw out error
12. end
```

LISTING 1. **Jaccard index calculation algorithm of sample pairs at SJ stage.**

2) CANDIDATE FEATURE MINING USING NEEDLEMAN-WUNSCH ALGORITHM

Although able to indicate the similarity between samples, the results obtained at SJ stage can't mark key information about offset, size, order, etc., since Jaccard index is based on disordered and non-repetitive sets. So, we introduce a new method to seek key information at NW stage.

Needleman-Wunsch algorithm is a method to match protein or DNA sequence based on the knowledge of bioinformatics, also known as optimization matching algorithm or overall sequence comparison method [24]. The introduction of this algorithm can help to search for common data blocks as candidate features between sample pairs. we regard Jaccard index matrix as Adjacency Matrix and single sample as node of graph. if Jaccard index is greater than 0 and less than 1, there is an edge between sample_A and sample_B.

During traverse of Jaccard index matrix using DFS algorithm, if edge existed between sample_A and sample_B, Needleman-Wunsch algorithm will be called to seek common data blocks between them, the overall algorithm design is shown in Listing.2.

```
Algorithm 2: candidate feature mining algorithm
Input  : samples in Sample Repository, Jaccard index matrix
Output: candidate feature set
1.  function NW_AddSet(sample_A, sample_B)
2.      match_seq = Needleman_Wunsch(sample_A, sample_B)
3.      return set(match_seq is divided into common data blocks)
4.  end
5.  function DFS_Jaccard(node)
6.      for (i in nodes)
7.          if (i is unmarked and 0 < Jaccard_index_matrix[node][i] < 1)
8.              add NW_AddSet(node, i) to candidate feature set
9.              stack.push(node)
10.             mark node is accessed
11.             DFS_Jaccard(i)
12.         if (stack is not empty)
13.             DFS_Jaccard(stack.pop())
14.     end
15. end
16. function main()
17.     while (there is unmarked node in nodes) do
18.         select unmarked node
19.         DFS_Jaccard(node)
20.     end
21. end
```

LISTING 2. **Candidate feature mining algorithm at NW stage. Note: it is considered that no edge between the sample pair when Jaccard index is equal to 1.**

Particularly, we do not deal with the situation where Jaccard index is equal to 1 at this stage. One of reasons is to avoid meaningless common data blocks which is the whole data of sample, another is to try to find the reason for this situation during manual analysis at MA stage.

3) MANUAL-ASSISTED ANALYSIS BASED ON EXPERT KNOWLEDGE

The most important reason for the necessity of manual-assisted analysis based on expert knowledge is that, there is no way to automatically recognize or know the meaning and role of these common data blocks. In fact, we have determined some questions which may make some elements in candidate feature set invalid during the feasibility analysis of this method: (1) some common data blocks are substrings of others; (2) multiple common data blocks with different sizes are zero, which may be used as reserved fields. A redundant and incorrect feature set will interfere with the analysis and reverse engineering, so the candidate feature set from NW stage cannot be used to the analysis of samples directly.

For common data blocks with Q-1, the common data block containing others is divided into two or more common data blocks; For common data blocks with Q-2, we will ignore temporarily and decide whether to select it as feature during evaluation; For common data blocks whose signification is known, we will filter and judge whether it can be selected as a classification or analytical criterion based on expert knowledge. Finally, we will consider size, data, and position of common data blocks, simultaneously utilizing knowledge and achievement from [6], [7], to determine the feature set.

C. EXPERIMENT

1) SETUP

The research object of this article is Intel microcode update, so the experiment of SJNW-MA method must depend on the Sample Repository. We download 42 microcode update packages (2009.03.30-2020.06.16) from Intel official way and process them as described in section 3.1 to build a Sample Repository containing 504 little-endian binary files of Intel microcode update samples, which are labeled by 56 values of *DisplayFamily_DisplayModel* corresponding to the specific processor models. Interestingly, only 54 labels have been documented in Intel developer's manual [20], and 06_06H and 06_16H are undocumented. But for unknown reasons, these microcode updates are also released publicly. Although Intel no longer supports update of some microcode updates for early processors, we still analyze all samples in the Sample Repository using SJNW-MA method.

We have implemented the SJNW-MA method based on python 3.7, and ran it on hardware environment with Intel Core i7-9700T and 16GB memory.

2) PROCESS AND RESULTS

We will describe in detail the intermediate results and analysis of each stage of the SJNW-MA method experiment. These valuable results have greatly guided the breakthrough of reverse engineering.

First, 126756 valid data of Jaccard index which stored in matrix are obtained at SJ stage. The number of data is equal to C_{504}^2. We use mathematical statistics to visually display these data, the heatmap and histogram of Jaccard index are shown in Fig.5. From heatmap (left), it can be intuitively found that some Jaccard index is higher and shows the characteristics of dense distribution, which verifies the rationality of previous assumption preliminary; From histogram (right), the amount of Jaccard index equaling to 0 is 58885, only accounting for 46.5% of all; there are even 33 cases where Jaccard index is 1, which means that the data except *Header* of two samples is the same. Those statistical results fully validate our assumption that there is a large amount of common data blocks between sample pairs, and thus they are similar.

Then, A total of 127 common data blocks, which as candidate feature set, are obtained at NW stage. Sizes of common data blocks range from 4 bytes to 72064 bytes and increments are irregular. These common data blocks do not appear in all samples. On the contrary, some special ones only appear between a sample pair.

Next, we completed the analysis of candidate features and 5 features were selected (As shown in Tab.1). In addition, we identify 24 RSA public keys from common data blocks, size of which is 260 bytes (256-byte *RSA modulus* and 4-byte *RSA Exponent*), and they will be explained in Section 4.2. Similarly, we also do not abandon those interesting and mysterious common data blocks which were not be selected into feature set, we will give our views on them in Section 4.1.

The cases where Jaccard index is equal 1, i.e., the data except *Header* of samples is identical, appear in some sample pairs with the same *Processor Signature* but different *Processor Flags*. This may be due to these microcode updates being interchangeable on these platforms.

TABLE 1. Feature set.

Symbo	Size (B)	MD5 of Feature
F1	1056	999ef222464b7acc0ebf1e141ebf5835
F2	952	d2ffaba7c4dd40e2a655de8b8d3efd6e
F3	12	b71ca2d0ea5297871e5837ce32d305c2
F4	28	1c9e99e48a495fe81d388fdb4900e59f
F5	12	1e387c8a7046e4bed8bd47d735d4ff5d

Finally, we create a vector $\{x_1, x_2, x_3, x_4, x_5\}$ for every sample. If equation $x_i = -1$ holds, the feature does not exist in this sample, while x_i is greater than 0, it indicates the offset of feature. 11 different vectors are extracted from all samples (as shown in Tab.2).

IV. RESULTS OF REVERSE ENGINEERING

Based on the work in chapter 3, we will give the detailed and objective results of reverse engineering about Intel microcode updates in this chapter. The structure analysis of microcode update and security technologies applied to update mechanism are described in section 4.1, 4.2 respectively.

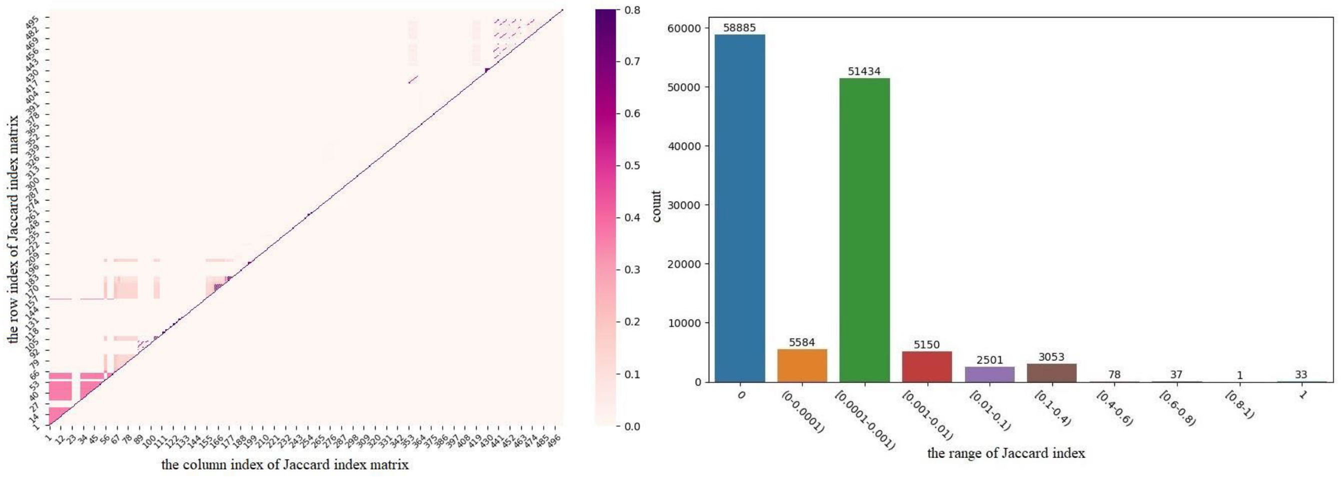


FIGURE 5. Heatmap (left) and histogram (right) of Jaccard index. Note: To show clearly, the step of axis scale in the heatmap (left) is not in units of 1, but it does not insinuate that the data has not participated in the calculation. The axis labels in the histogram (right) indicate the range of Jaccard index, e.g. [0.0001-0.001) represents the value of Jaccard index is equal to or greater than 0.001 but less than 0.001.

TABLE 2. Eleven different vectors.

Seq	Vector	Number of samples
1	{912, -1, -1, -1, -1}	4
2	{992, -1, -1, -1, -1}	50
3	{-1, 992, -1, -1, -1}	6
4	{-1, 2120, -1, -1, -1}	2
5	{-1, 3144, -1, -1, -1}	34
6	{-1, -1, 48, 88, -1}	336
7	{-1, 3124, 48, 88, -1}	12
8	{-1, 3188, 48, 88, -1}	1
9	{-1, 4148, 48, 88, -1}	6
10	{-1, -1, -1, 88, 48}	2
11	{-1, -1, -1, -1, -1}	51

A. STRUCTURE

The 504 Intel microcode updates and their structures are grouped into four categories according to the feature representation. The first category includes updates with vectors as 1, 2, 3, 4, and 5; the third category includes updates with vectors as 6, 7, 8, 9; and the second, fourth category respectively includes updates with vector as 11, 10 in Tab.2.

The sizes of updates belonged to the earliest and first category are 2048 bytes or 4096 bytes, which containing an unknown data block of 2000 bytes or 4048 bytes respectively. The detailed structures are shown in Fig.6, (1), (2) and (3) are the structures of 2048-byte updates. (4) and (5) are the structures of 4096-byte updates. Compared to (2), the offset of feature F1 in (1) is moved forward to 912 and a 80-byte data block (only shared in (1)) is added at the end; and in (3), F1 is replaced by another 1056-byte common data block consisting of F2 and a 104-byte common data block. Compared to (5), the offset of feature F2 in (4) is moved forward to 2120 and a 920-byte data block (only shared in (4)) following a 104-byte common data block is added at the end; there is a 1056-byte common data block containing F2 between (3) and (4). Those common data blocks have been preliminary confirmed as garbage data to deter reverse engineering, which do not affect acceptance of microcode update by processor [6], [25].

The mapping relationship of processor products and structures is shown in Appendix C. It is emphasized that there are two structures of updates targeting 06_0EH, (4) is used by processors whose *Stepping* is CH and (5) is used by that whose *Stepping* is 8H. In addition, the latest update of the first category is released in 2010, i.e., Intel is no longer supporting its update.

The microcode updates of the second category target processors whose *DisplayFamily_DisplayModel* are 06_06H, 0F_00H, 0F_01H, 0F_02H, 0F_03H, 0F_04H or 0F_06H (not all). Although there is no obvious characteristic in updates of this category like that in the first category, we still discern some irregular and different-size common data blocks (one of them is shown in Appendix A). Above of all, "EFFFFFFFH" frequently appears in the first four bytes in *Data* of updates. Similarly, Intel is no longer supporting update of such updates.

The earliest microcode update of the third category is released in 2006 (0x00000.65_0x00000001_0x00000008, 0F_06H), and this structure is still in use now. The most obvious characteristic is that a 692-byte prefix in *Data*, the data structure of which is shown in Tab.3. the value of *Flexible Field 1* is not static and we give the detailed value in Appendix B. when the value of *Flexible Field 1* is not equal to *Processor Flags*, which is a sufficient and unnecessary condition, the value of *Flexible Field 2* must be taken as 0H.

Surprisingly, a common data block after the range marked by *Update Length* usually appears in updates with size more than 71680 bytes, whose size is different and some are even shared with microcode updates of the first category

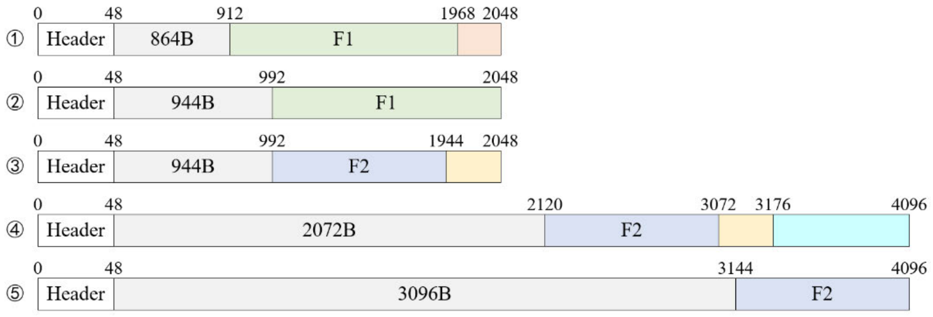


FIGURE 6. Structures of microcode update (The First Category). The numbers above bar are offset of the data blocks, and the numbers in bar are size of data blocks, e.g. "2072B" represents the size of this data block is 2072 bytes.

TABLE 3. Prefix of microcode update (the third category).

Prefix	Offset	Field	Size (B)	Value (default)
Description (128B)	48	Unknown 1	4	0H
	52	Magic Number	4	a1H
	56	Unknown 2	4	20001H
	60	Update Revision	4	
	64	Unknown 3	4	06_1AH 06_1EH 06_25H 06_2AH 06_2CH 06_2DH 06_2EH
	68	Unknown 4	4	06_2FH 06_37H 06_3AH 06_3EH 06_4CH 06_4DH are 0H
	72	Date	4	No later than *Date* in *Header*
	76	Update Length	4	In units of 4 bytes and count from offset 48
	80	Unknown 5	4	1H (May be *Loader Revision*)
	84	Processor Signature	4	
	88	Unknown 6	28	0H
	116	Flexible Field 1	4	*Processor Flags*, *Update Length*, 0 or others
	120	Flexible Field 2	4	
	124	Unknown 7	20	0H
	144	Unknown 8	32	
Encryption (516B)	176	RSA Modulus	256	Varies across processor families
	432	RSA Exponent	4	11H
	436	RSA Signature	256	

(vectors7, 8, 9). But we do not know the role of it currently. This may indicate the presence of shared loader or exit handler code, or alternatively common patches for the same processor errata.

At present, only two microcode updates for Ice Lake microarchitecture is grouped into the fourth category, the structure of which also is the newest (the filename of earliest one is 0x000706.5_0x 00000080_0x 00000046, released on September 5, 2019). The first part of prefix, *Description*, is also discovered and it should be inherited from updates of the third category, but the values of some fields have changed, such as *Magic Number*, *Unknown 2*. The most important change is that *RSA Exponent* does not exist.

B. ENCRYPTION, DIGITING DIGEST AND SIGNATURE

Although all microcode updates adopt checksum for integrity verification, this mechanism is relatively simple and public, attackers can easily modify the update data and forge a new checksum. Therefore, encryption, digiting digest, signature, or other technologies are also used into the microcode update.

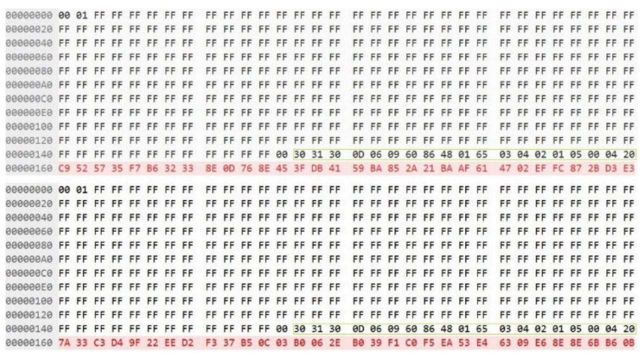
Microcode updates of the first category may be encrypted using a block cipher with a block size of 64 bits or less, this block cipher does not appear to be chained [6], our results show that $gcd(864, 944, 2072, 3096) = 8$ bytes. No much is known about updates of the second category, but we think it may still use the similar encryption algorithm as the first category.

The technology that using RSA signature for digiting digest to protect the integrity of encrypted data has been discovered in updates of the third category. Referring to the work of Hawkes, a total of 24 RSA public keys are extracted, which are shared among microcode updates for different processors. The detailed information is seen in Appendix D. The recovered results show that digiting digest is padded following PKCS#1 v1.5, with a private key operation set for the block type. The 160-bit digiting digest (SHA-1) is used for earlier processor, including 0F_06H, 06_0FH (*Stepping* is BH) and 06_17H; Others use a 256-bit digiting digest (SHA-2 even SHA-3). It can be confirmed that Encryption-then-MAC has been applied in microcode updates of the third category as described in the patent [26], but the specific encryption algorithm of block cipher is unknown.

When analyzing the microcode updates for 06_0FH (*Stepping* is BH), it was found that the results of RSA Signature extracted from updates with the same *Processor Signature* and *Update Revision* while different *Processor Flags* are also the same, but data out after the range marked by *Update Length* is not unanimous. This fact proves that the digiting digest is only for the data within the range which marked by *Update Length*, due to considering efficiency of microcode update loading.

Although no visible *RSA Exponent* is found in updates of the fourth category, a 3072-bit common data block whose offset is 176 to 560 bytes may be a potential *RSA Modulus*. It meets the following conditions:

- 3072 bits is a size of commercial *RSA Modulus* with higher security level,
- The value is odd numbered,
- The value is not factorable by any value between 2 and 2^{32}.



```
00000000 00 01 FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
00000020 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
00000040 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
00000060 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
00000080 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
000000A0 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
000000C0 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
000000E0 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
00000100 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
00000120 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
00000140 FF FF FF FF FF FF FF FF  FF FF FF FF 00 30 31 30  0D 06 09 60 86 48 01 65  03 04 02 01 05 00 04 20
00000160 C9 52 57 35 F7 B6 32 33  8E 0D 76 8E 45 3F DB 41  59 BA 85 2A 21 BA AF 61  47 02 EF FC 87 2B D3 E3

00000000 00 01 FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
00000020 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
00000040 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
00000060 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
00000080 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
000000A0 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
000000C0 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
000000E0 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
00000100 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
00000120 FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF
00000140 FF FF FF FF FF FF FF FF  FF FF FF FF 00 30 31 30  0D 06 09 60 86 48 01 65  03 04 02 01 05 00 04 20
00000160 7A 33 C3 D4 9F 22 EE D2  F3 37 B5 0C 03 B0 06 2E  B0 39 F1 C0 F5 EA 53 E4  63 09 E6 8E 8E 6B B6 0B
```

FIGURE 7. **The original signed data (red) of microcode update (the fourth category). The data block marked by green frame is the 19-byte common data block.**

The value of 3072-bit data block whose offset is 560 to 944 bytes is less than the potential *RSA Modulus*, so it may be an RSA signature. We try to determine the value of *RSA Exponent* using enumeration method, and when the value is taken as 65537 (10001H), the original signed data meeting PKCS#1 v1.5 is obtained (shown in Fig.7). Due to avalanche effect, the 19-byte common data block appearing in the original signed data seems to be additional information rather than part of the digital digest. Therefore, updates of the fourth category still use SHA-2 or SHA-3 to generate the digital digest whose size is 256 bits.

V. EVALUATION AND DISCUSSION

A. THE NEW DISCOVERIES

The previous manual analysis as well our semiautomatic method cannot give precise time, accuracy, or coverage to compare. We only introduce some new discoveries to illustrate the effectiveness and advanced of our method.

Compared with the traditional and sporadic manual analysis [6], [7], SJNW-MA is a fine-grained, accurate, and efficient method to analyze and reverse microcode updates. Base on the results from the analysis of 504 Intel microcode updates using it, we have completed the reverse engineering. Some new structures of updates are discovered and the values of metadata are determined, and we also extract 24 RSA public keys more 12 than that found by Hawkes [7]. The most significant result is that we extract a new structure, and identify a new 3072-bit *RSA Modulus* in microcode updates of the fourth category and recover the original signed data successfully.

B. ACTUAL ATTACK CHAIN

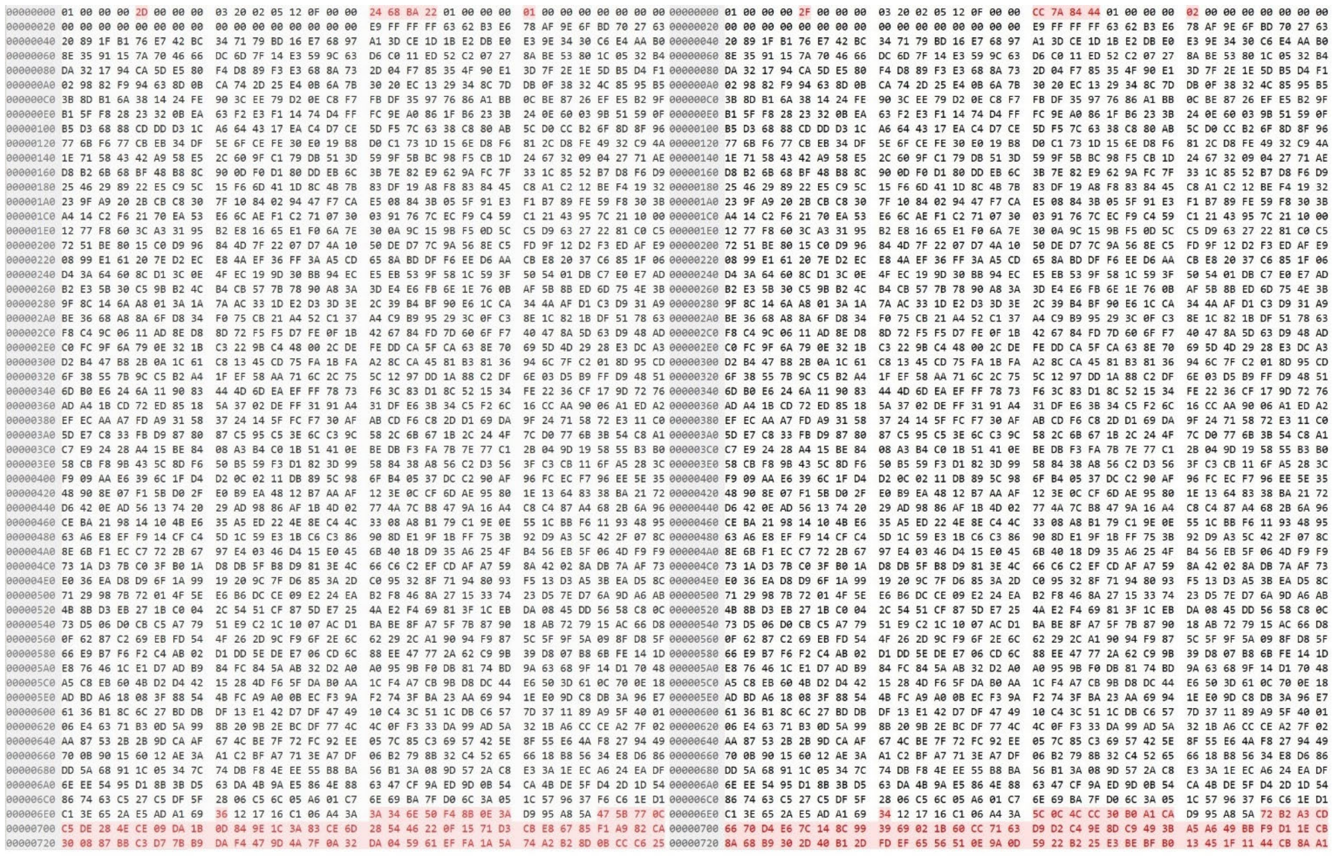
Of course, our work is only a preliminary study for the security of microcode updates. There is still much work to be done for actual available attack chain, such as crack of encryption algorithm and secure hash algorithm used in microcode update, reverse engineering of micro-ops encoding, etc. In addition, it is also to consider whether the security of microcode update is affected by BIOS or operating system, which will also be our next work. All in all, we hope that results of this article can provide a basis for further research.

C. THE CONFRONTATION BETWEEN ATTACK AND DEFENSE

From our results of reverse engineering, Intel has adopted a variety of methods to strengthen the security of microcode update mechanism, such as encryption, signature, checksum, even on-chip execution. The microcode updates for early processor do not seem to be so secure, but these technical measures have achieved their goals to a certain extent.

Many new technologies have been used for hardware or software reverse analysis, which may be affect research of microcode updates. Side channel attack has been shown effective in disclosing secret encryption keys stored within commercial FPGAs, but may be difficult to apply to processors due to increased silicon complexity. Hardware-based epoxy decapping and analysis under a microscope with fuming sulfuric acid has been successful in revealing the contents of secret memories within microcontrollers. There also exist proprietary physical debugging interfaces for processors via exposed surface mount pads or land grid array pins, e.g. Intel's XDP or other JTAG debuggers, which may be useful for obtaining more information about processor internals [6].

Moreover, it is not ruled out that there are security risks in the microcode update supply chain. As stated in introduction, researchers are difficult to study the updates to clearly establish whether any security issues are being fixed like what developer claimed or new security issues are introduced, even backdoors reserved by manufacturers. What is more terrible is that all users, including professionals, have no effective way to verify whether the microcode update in their devices is malicious. This means that once attacked, the problem will



```
00000000 01 00 00 00 2D 00 00 00  03 20 02 05 12 0F 00 00  24 68 BA 22 01 00 00 00  01 00 00 00 00 00 00 00
00000020 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  E9 FF FF FF 63 62 B3 E6  78 AF 9E 6F BD 70 27 63
00000040 20 89 1F B1 76 E7 42 BC  34 71 79 BD 16 E7 68 97  A1 3D CE 1D 1B E2 DB E0  E3 9E 34 30 C6 E4 AA B0
00000060 8E 35 91 15 7A 70 46 66  DC 6D 7F 14 E3 59 9C 63  D6 C0 11 ED 52 C2 07 27  8A BE 53 80 1C 05 32 B4
00000080 DA 32 17 94 CA 5D E5 80  F4 D8 89 F3 E3 68 8A 73  2D 04 F7 85 35 4F 90 E1  3D 7F 2E 1E 5D B5 D4 F1
000000A0 02 98 82 F9 94 63 8D 0B  CA 74 2D 25 E4 0B 6A 7B  30 20 EC 13 29 34 8C 7D  DB 0F 38 32 4C 85 95 B5
000000C0 3B 8D B1 6A 38 14 24 FE  90 3C EE 79 D2 0E C8 F7  FB DF 35 97 76 86 A1 BB  0C BE 87 26 EF E5 B2 9F
000000E0 B1 5F F8 28 23 32 0B EA  63 F2 E3 F1 14 74 D4 FF  FC 9E A0 86 1F B6 23 3B  24 0E 60 03 9B 51 59 0F
00000100 B5 D3 68 88 CD DD D3 1C  A6 64 43 17 EA C4 D7 CE  5D F5 7C 63 38 C8 80 AB  5C D0 CC B2 6F 8D 8F 96
00000120 77 6B F6 77 CB EB 34 DF  5E 6F CE FE 30 E0 19 B8  D0 C1 73 1D 15 6E D8 F6  81 2C D8 FE 49 32 C9 4A
00000140 1E 71 58 43 42 A9 58 E5  2C 60 9F C1 79 DB 51 3D  59 9F 5B BC 98 F5 CB 1D  24 67 32 09 04 27 71 AE
00000160 D8 B2 6B 68 BF 48 B8 8C  90 0D F0 D1 80 DD EB 6C  3B 7E 82 E9 62 9A FC 7F  33 1C 85 52 B7 D8 F6 D9
00000180 25 46 29 89 22 E5 C9 5C  15 F6 6D 41 1D 8C 4B 7B  83 DF 19 A8 F8 83 84 45  C8 A1 C2 12 BE F4 19 32
000001A0 23 9F A9 20 2B CB C8 30  7F 10 84 02 94 47 F7 CA  E5 08 84 3B 05 5F 91 E3  F1 B7 89 FE 59 F8 30 3B
000001C0 A4 14 C2 F6 21 70 EA 53  E6 6C AE F1 C2 71 07 30  03 91 76 7C EC F9 C4 59  C1 21 43 95 7C 21 10 00
000001E0 12 77 F8 60 3C A3 31 95  B2 E8 16 65 E1 F0 6A 7E  30 0A 9C 15 9B F5 0D 5C  C5 D9 63 27 22 81 C0 C5
00000200 72 51 BE 80 15 C0 D9 96  84 4D 7F 22 07 D7 4A 10  50 DE D7 7C 9A 56 8E C5  FD 9F 12 D2 F3 ED AF E9
00000220 08 99 E1 61 20 7E D2 EC  E8 4A EF 36 FF 3A A5 CD  65 8A BD DF F6 EE D6 AA  CB E8 20 37 C6 85 1F 06
00000240 D4 3A 64 60 8C D1 3C 0E  4F EC 19 9D 30 BB 94 EC  E5 EB 53 9F 58 1C 59 3F  50 54 01 DB C7 E0 E7 AD
00000260 B2 E3 5B 30 C5 9B B2 4C  B4 CB 57 7B 78 90 A8 3A  3D E4 E6 FB 6E 1E 76 0B  AF 5B 8B ED 6D 75 4E 3B
00000280 9F 8C 14 6A A8 01 3A 1A  7A AC 33 1D E2 D3 3D 3E  2C 39 B4 BF 90 E6 1C CA  34 4A AF D1 C3 D9 31 A9
000002A0 BE 36 68 A8 8A 6F D8 34  F0 75 CB 21 A4 52 C1 37  A4 C9 B9 95 29 3C 0F C3  8E 1C 82 1B DF 51 78 63
000002C0 F8 C4 9C 06 11 AD 8E D8  8D 72 F5 F5 D7 FE 0F 1B  42 67 84 FD 7D 60 6F F7  40 47 8A 5D 63 D9 48 AD
000002E0 C0 FC 9F 6A 79 0E 32 1B  C3 22 9B C4 48 00 2C DE  FE DD CA 5F CA 63 8E 70  69 5D 4D 29 28 E3 DC A3
00000300 D2 B4 47 B8 2B 0A 1C 61  C8 13 45 CD 75 FA 1B FA  A2 8C CA 45 81 B3 81 36  94 6C 7F C2 01 8D 95 CD
00000320 6F 38 55 7B 9C C5 B2 A4  1F EF 58 AA 71 6C 2C 75  5C 12 97 DD 1A 88 C2 DF  6E 03 D5 B9 FF D9 48 51
00000340 6D B0 E6 24 6A 11 90 83  44 4D 6D EA EF FF 78 73  F6 3C 83 D1 8C 52 15 34  FE 22 36 CF 17 9D 72 76
00000360 AD A4 1B CD 72 ED 85 18  5A 37 02 DE FF 31 91 A4  31 DF E6 3B 34 C5 F2 6C  16 CC AA 90 06 A1 ED A2
00000380 EF EC AA A7 FD A9 31 58  37 24 14 5F FC F7 30 AF  AB CD F6 C8 2D D1 69 DA  9F 24 71 58 72 E3 11 C0
000003A0 5D E7 C8 33 FB D9 87 80  87 C5 95 C5 3E 6C C3 9C  58 2C 6B 67 1B 2C 24 4F  7C D0 77 6B 3B 54 C8 A1
000003C0 C7 E9 24 28 A4 15 BE 84  08 A3 B4 C0 1B 51 41 0E  BE DB F3 FA 7B 7E 77 C1  2B 04 9D 19 58 55 B3 B0
000003E0 58 CB F8 9B 43 5C 8D F6  50 B5 59 F3 D1 82 3D 99  58 84 38 A8 56 C2 D3 56  3F C3 CB 11 6F A5 28 3C
00000400 F9 09 AA E6 39 6C 1F D4  D2 0C 02 11 DB 89 5C 98  6F B4 05 37 DC C2 90 AF  96 FC EC F7 96 EE 5E 35
00000420 48 90 8E 07 F1 5B D0 2F  E0 B9 EA 48 12 B7 AA AF  12 3E 0C CF 6D AE 95 80  1E 13 64 83 38 BA 21 72
00000440 D6 42 0E AD 56 13 74 20  29 AD 98 86 AF 1B 4D 02  77 4A 7C B8 47 9A 16 A4  C8 C4 87 A4 68 2B 6A 96
00000460 CE BA 21 98 14 10 4B E6  35 A5 ED 22 4E 8E C4 4C  33 08 A8 B1 79 C1 9E 0E  55 1C BB F6 11 93 48 95
00000480 63 A6 E8 EF F9 14 CF C4  5D 1C 59 E3 1B C6 C3 86  90 8D E1 9F 1B FF 75 3B  92 D9 A3 5C 42 2F 07 8C
000004A0 8E 6B F1 EC C7 72 2B 67  97 E4 03 46 D4 15 E0 45  6B 40 18 D9 35 A6 25 4F  B4 56 EB 5F 06 4D F9 F9
000004C0 73 1A D3 7B C0 3F B0 1A  D8 DB 5F B8 D9 81 3E 4C  66 C6 C2 EF CD AF A7 59  8A 42 02 8A DB 7A AF 73
000004E0 E0 36 EA D8 D9 6F 1A 99  19 20 9C 7F D6 85 3A 2D  C0 95 32 8F 71 94 80 93  F5 13 D3 A5 3B EA D5 8C
00000500 71 29 98 7B 72 01 4F 5E  E6 B6 DC CE 09 E2 24 EA  B2 F8 46 8A 27 15 33 74  23 D5 7E D7 6A 9D A6 AB
00000520 4B 8B D3 EB 27 1B C0 04  2C 54 51 CF 87 5D E7 25  4A E2 F4 69 81 3F 1C EB  DA 08 45 DD 56 58 C8 0C
00000540 73 D5 06 D0 CB C5 A7 79  51 E9 C2 1C 10 07 AC D1  BA BE 8F A7 5F 7B 87 90  18 AB 72 79 15 AC 66 D8
00000560 0F 62 87 C2 69 EB FD 54  4F 26 2D 9C F9 6F 2E 6C  62 29 2C A1 90 94 F9 87  5C 5F 9F 5A 09 8F D8 5F
00000580 66 E9 B7 F6 F2 C4 AB 02  D1 DD 5E DE E7 06 CD 6C  88 EE 47 77 2A 62 C9 9B  39 D8 07 B8 6B FE 14 1D
000005A0 E8 76 46 1C E1 D7 AD B9  84 FC 84 5A AB 32 D2 A0  A0 95 9B F0 DB 81 74 BD  9A 63 68 9F 14 D1 70 48
000005C0 A5 C8 EB 60 4B D2 D4 42  15 28 4D F6 5F DA B0 AA  1C F4 A7 CB 9B D8 DC 44  E6 50 3D 61 0C 70 0E 18
000005E0 AD BD A6 18 08 3F 88 54  4B FC A9 A0 0B EC F3 9A  F2 74 3F BA 23 AA 69 94  1E E0 9D C8 DB 3A 96 E7
00000600 61 36 B1 8C 6C 27 BD DB  DF 13 E1 42 D7 DF 47 49  10 C4 3C 51 1C DB C6 57  7D 37 11 89 A9 5F 40 01
00000620 06 E4 63 71 B3 0D 5A 99  8B 20 9B 2E BC DF 77 4C  4C 0F F3 33 DA 99 AD 5A  32 1B A6 CC CE A2 7F 02
00000640 AA 87 53 2B 2B 9D CA AF  67 4C BE 7F 72 FC 92 EE  05 7C 85 C3 69 57 42 5E  8F 55 E6 4A F8 27 94 49
00000660 70 0B 90 15 60 12 AE 3A  A1 C2 BF A7 71 3E A7 DF  06 B2 79 8B 32 C4 52 65  66 18 B8 56 34 E8 D6 86
00000680 DD 5A 68 91 1C 05 34 7C  74 DB F8 4E EE 55 B8 BA  56 B1 3A 08 9D 57 2A C8  E3 3A 1E EC A6 24 EA DF
000006A0 6E EE 54 95 D1 8B 3B D5  63 DA 4B 9A E5 86 4E 88  63 47 CF 9A ED 9D 0B 54  CA 4B DE 5F D4 2D 1D 54
000006C0 86 74 63 C5 27 C5 DF 5F  28 06 C5 6C 05 A6 01 C7  6E 69 BA 7F D0 6C 3A 05  1C 57 96 37 F6 C6 1E D1
000006E0 C1 3E 65 2A E5 AD A1 69  36 12 17 16 C1 06 A4 3A  3A 34 6E 50 F4 8B 0E 3A  D9 95 A8 5A 47 5B 77 0C
00000700 C5 DE 28 4E CE 09 DA 1B  0D 84 9E 1C 3A 83 CE 6D  28 54 46 22 0F 15 71 D3  CB E8 67 85 F1 A9 82 CA
00000720 30 08 87 BB C3 D7 7B B9  DA F4 47 9D 4A 7F 0A 32  DA 04 59 61 EF FA 1A 5A  74 A2 B2 8D 0B CC C6 25
```

0x00000f12_0x00000001_0x0000002d

```
00000000 01 00 00 00 2F 00 00 00  03 20 02 05 12 0F 00 00  CC 7A 84 44 01 00 00 00  02 00 00 00 00 00 00 00
00000020 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  E9 FF FF FF 63 62 B3 E6  78 AF 9E 6F BD 70 27 63
00000040 20 89 1F B1 76 E7 42 BC  34 71 79 BD 16 E7 68 97  A1 3D CE 1D 1B E2 DB E0  E3 9E 34 30 C6 E4 AA B0
00000060 8E 35 91 15 7A 70 46 66  DC 6D 7F 14 E3 59 9C 63  D6 C0 11 ED 52 C2 07 27  8A BE 53 80 1C 05 32 B4
00000080 DA 32 17 94 CA 5D E5 80  F4 D8 89 F3 E3 68 8A 73  2D 04 F7 85 35 4F 90 E1  3D 7F 2E 1E 5D B5 D4 F1
000000A0 02 98 82 F9 94 63 8D 0B  CA 74 2D 25 E4 0B 6A 7B  30 20 EC 13 29 34 8C 7D  DB 0F 38 32 4C 85 95 B5
000000C0 3B 8D B1 6A 38 14 24 FE  90 3C EE 79 D2 0E C8 F7  FB DF 35 97 76 86 A1 BB  0C BE 87 26 EF E5 B2 9F
000000E0 B1 5F F8 28 23 32 0B EA  63 F2 E3 F1 14 74 D4 FF  FC 9E A0 86 1F B6 23 3B  24 0E 60 03 9B 51 59 0F
00000100 B5 D3 68 88 CD DD D3 1C  A6 64 43 17 EA C4 D7 CE  5D F5 7C 63 38 C8 80 AB  5C D0 CC B2 6F 8D 8F 96
00000120 77 6B F6 77 CB EB 34 DF  5E 6F CE FE 30 E0 19 B8  D0 C1 73 1D 15 6E D8 F6  81 2C D8 FE 49 32 C9 4A
00000140 1E 71 58 43 42 A9 58 E5  2C 60 9F C1 79 DB 51 3D  59 9F 5B BC 98 F5 CB 1D  24 67 32 09 04 27 71 AE
00000160 D8 B2 6B 68 BF 48 B8 8C  90 0D F0 D1 80 DD EB 6C  3B 7E 82 E9 62 9A FC 7F  33 1C 85 52 B7 D8 F6 D9
00000180 25 46 29 89 22 E5 C9 5C  15 F6 6D 41 1D 8C 4B 7B  83 DF 19 A8 F8 83 84 45  C8 A1 C2 12 BE F4 19 32
000001A0 23 9F A9 20 2B CB C8 30  7F 10 84 02 94 47 F7 CA  E5 08 84 3B 05 5F 91 E3  F1 B7 89 FE 59 F8 30 3B
000001C0 A4 14 C2 F6 21 70 EA 53  E6 6C AE F1 C2 71 07 30  03 91 76 7C EC F9 C4 59  C1 21 43 95 7C 21 10 00
000001E0 12 77 F8 60 3C A3 31 95  B2 E8 16 65 E1 F0 6A 7E  30 0A 9C 15 9B F5 0D 5C  C5 D9 63 27 22 81 C0 C5
00000200 72 51 BE 80 15 C0 D9 96  84 4D 7F 22 07 D7 4A 10  50 DE D7 7C 9A 56 8E C5  FD 9F 12 D2 F3 ED AF E9
00000220 08 99 E1 61 20 7E D2 EC  E8 4A EF 36 FF 3A A5 CD  65 8A BD DF F6 EE D6 AA  CB E8 20 37 C6 85 1F 06
00000240 D4 3A 64 60 8C D1 3C 0E  4F EC 19 9D 30 BB 94 EC  E5 EB 53 9F 58 1C 59 3F  50 54 01 DB C7 E0 E7 AD
00000260 B2 E3 5B 30 C5 9B B2 4C  B4 CB 57 7B 78 90 A8 3A  3D E4 E6 FB 6E 1E 76 0B  AF 5B 8B ED 6D 75 4E 3B
00000280 9F 8C 14 6A A8 01 3A 1A  7A AC 33 1D E2 D3 3D 3E  2C 39 B4 BF 90 E6 1C CA  34 4A AF D1 C3 D9 31 A9
000002A0 BE 36 68 A8 8A 6F D8 34  F0 75 CB 21 A4 52 C1 37  A4 C9 B9 95 29 3C 0F C3  8E 1C 82 1B DF 51 78 63
000002C0 F8 C4 9C 06 11 AD 8E D8  8D 72 F5 F5 D7 FE 0F 1B  42 67 84 FD 7D 60 6F F7  40 47 8A 5D 63 D9 48 AD
000002E0 C0 FC 9F 6A 79 0E 32 1B  C3 22 9B C4 48 00 2C DE  FE DD CA 5F CA 63 8E 70  69 5D 4D 29 28 E3 DC A3
00000300 D2 B4 47 B8 2B 0A 1C 61  C8 13 45 CD 75 FA 1B FA  A2 8C CA 45 81 B3 81 36  94 6C 7F C2 01 8D 95 CD
00000320 6F 38 55 7B 9C C5 B2 A4  1F EF 58 AA 71 6C 2C 75  5C 12 97 DD 1A 88 C2 DF  6E 03 D5 B9 FF D9 48 51
00000340 6D B0 E6 24 6A 11 90 83  44 4D 6D EA EF FF 78 73  F6 3C 83 D1 8C 52 15 34  FE 22 36 CF 17 9D 72 76
00000360 AD A4 1B CD 72 ED 85 18  5A 37 02 DE FF 31 91 A4  31 DF E6 3B 34 C5 F2 6C  16 CC AA 90 06 A1 ED A2
00000380 EF EC AA A7 FD A9 31 58  37 24 14 5F FC F7 30 AF  AB CD F6 C8 2D D1 69 DA  9F 24 71 58 72 E3 11 C0
000003A0 5D E7 C8 33 FB D9 87 80  87 C5 95 C5 3E 6C C3 9C  58 2C 6B 67 1B 2C 24 4F  7C D0 77 6B 3B 54 C8 A1
000003C0 C7 E9 24 28 A4 15 BE 84  08 A3 B4 C0 1B 51 41 0E  BE DB F3 FA 7B 7E 77 C1  2B 04 9D 19 58 55 B3 B0
000003E0 58 CB F8 9B 43 5C 8D F6  50 B5 59 F3 D1 82 3D 99  58 84 38 A8 56 C2 D3 56  3F C3 CB 11 6F A5 28 3C
00000400 F9 09 AA E6 39 6C 1F D4  D2 0C 02 11 DB 89 5C 98  6F B4 05 37 DC C2 90 AF  96 FC EC F7 96 EE 5E 35
00000420 48 90 8E 07 F1 5B D0 2F  E0 B9 EA 48 12 B7 AA AF  12 3E 0C CF 6D AE 95 80  1E 13 64 83 38 BA 21 72
00000440 D6 42 0E AD 56 13 74 20  29 AD 98 86 AF 1B 4D 02  77 4A 7C B8 47 9A 16 A4  C8 C4 87 A4 68 2B 6A 96
00000460 CE BA 21 98 14 10 4B E6  35 A5 ED 22 4E 8E C4 4C  33 08 A8 B1 79 C1 9E 0E  55 1C BB F6 11 93 48 95
00000480 63 A6 E8 EF F9 14 CF C4  5D 1C 59 E3 1B C6 C3 86  90 8D E1 9F 1B FF 75 3B  92 D9 A3 5C 42 2F 07 8C
000004A0 8E 6B F1 EC C7 72 2B 67  97 E4 03 46 D4 15 E0 45  6B 40 18 D9 35 A6 25 4F  B4 56 EB 5F 06 4D F9 F9
000004C0 73 1A D3 7B C0 3F B0 1A  D8 DB 5F B8 D9 81 3E 4C  66 C6 C2 EF CD AF A7 59  8A 42 02 8A DB 7A AF 73
000004E0 E0 36 EA D8 D9 6F 1A 99  19 20 9C 7F D6 85 3A 2D  C0 95 32 8F 71 94 80 93  F5 13 D3 A5 3B EA D5 8C
00000500 71 29 98 7B 72 01 4F 5E  E6 B6 DC CE 09 E2 24 EA  B2 F8 46 8A 27 15 33 74  23 D5 7E D7 6A 9D A6 AB
00000520 4B 8B D3 EB 27 1B C0 04  2C 54 51 CF 87 5D E7 25  4A E2 F4 69 81 3F 1C EB  DA 08 45 DD 56 58 C8 0C
00000540 73 D5 06 D0 CB C5 A7 79  51 E9 C2 1C 10 07 AC D1  BA BE 8F A7 5F 7B 87 90  18 AB 72 79 15 AC 66 D8
00000560 0F 62 87 C2 69 EB FD 54  4F 26 2D 9C F9 6F 2E 6C  62 29 2C A1 90 94 F9 87  5C 5F 9F 5A 09 8F D8 5F
00000580 66 E9 B7 F6 F2 C4 AB 02  D1 DD 5E DE E7 06 CD 6C  88 EE 47 77 2A 62 C9 9B  39 D8 07 B8 6B FE 14 1D
000005A0 E8 76 46 1C E1 D7 AD B9  84 FC 84 5A AB 32 D2 A0  A0 95 9B F0 DB 81 74 BD  9A 63 68 9F 14 D1 70 48
000005C0 A5 C8 EB 60 4B D2 D4 42  15 28 4D F6 5F DA B0 AA  1C F4 A7 CB 9B D8 DC 44  E6 50 3D 61 0C 70 0E 18
000005E0 AD BD A6 18 08 3F 88 54  4B FC A9 A0 0B EC F3 9A  F2 74 3F BA 23 AA 69 94  1E E0 9D C8 DB 3A 96 E7
00000600 61 36 B1 8C 6C 27 BD DB  DF 13 E1 42 D7 DF 47 49  10 C4 3C 51 1C DB C6 57  7D 37 11 89 A9 5F 40 01
00000620 06 E4 63 71 B3 0D 5A 99  8B 20 9B 2E BC DF 77 4C  4C 0F F3 33 DA 99 AD 5A  32 1B A6 CC CE A2 7F 02
00000640 AA 87 53 2B 2B 9D CA AF  67 4C BE 7F 72 FC 92 EE  05 7C 85 C3 69 57 42 5E  8F 55 E6 4A F8 27 94 49
00000660 70 0B 90 15 60 12 AE 3A  A1 C2 BF A7 71 3E A7 DF  06 B2 79 8B 32 C4 52 65  66 18 B8 56 34 E8 D6 86
00000680 DD 5A 68 91 1C 05 34 7C  74 DB F8 4E EE 55 B8 BA  56 B1 3A 08 9D 57 2A C8  E3 3A 1E EC A6 24 EA DF
000006A0 6E EE 54 95 D1 8B 3B D5  63 DA 4B 9A E5 86 4E 88  63 47 CF 9A ED 9D 0B 54  CA 4B DE 5F D4 2D 1D 54
000006C0 86 74 63 C5 27 C5 DF 5F  28 06 C5 6C 05 A6 01 C7  6E 69 BA 7F D0 6C 3A 05  1C 57 96 37 F6 C6 1E D1
000006E0 C1 3E 65 2A E5 AD A1 69  34 12 17 16 C1 06 A4 3A  5C 0C 4C CC 30 B0 A1 CA  D9 95 A8 5A 72 B2 A3 CD
00000700 66 70 D4 E6 7C 14 8C 99  39 69 02 1B 60 CC 71 63  D9 D2 C4 9E 8D C9 49 3B  A5 A6 49 BB F9 D1 1E CB
00000720 8A 68 B9 30 2D 40 B1 2D  FD EF 65 56 51 0E 9A 0D  59 22 B2 25 E3 BE BF B0  13 45 1F 11 44 CB 8A A1
```

0x00000f12_0x00000002_0x0000002f

FIGURE 8. A common data blocks between microcode updates of the second category.

TABLE 4. The value of flexible field 1.

Value	Processor Families / Processor Number Series	Number of updates
Processor Flags	06_4EH 06_55H 06_5CH 06_5EH 06_5FH 06_66H 06_7AH 06_8EH 06_9EH 06_A6H	122
Update Length	06_2AH 06_2DH 06_3EH	35
0H	06_0FH 06_17H 06_1AH 06_1CH 06_1EH 06_25H 06_26H 06_2CH 06_2EH 06_2FH 06_36H 06_3AH 06_3CH 06_3DH 06_3FH 06_45H 06_46H 06_47H 06_4CH 06_4FH 06_56H 0F_06H	194
others	06_37H 06_4DH	4

not be fixed and perceived by user, until the next time a new version of the microcode update is installed.

VI. CONCLUSION

In this article, we build a Sample Repository containing 504 Intel microcode updates and propose a semiautomatic analytical method SJNW-MA. Compared with previous manual analysis, this method improves efficiency and accuracy, which is used to make a more precise division and analysis of the microcode update structure. Finally, 11-kind structures of Intel microcode updates are identified from 504 samples, and are grouped into four categories. Only updates of the third and fourth category are still in use now. In addition, we do not find that *Extended Signature Table* documented in the Intel technical manual was used in all updates.

All microcode updates adopt encryption technology to protect the data. Updates of the third category are currently the most widely used. The values of *Flexible Field 1*, *Flexible Field 2* and other meaningful metadata are identified and described in this article. Digiting digest and RSA signature used for integrity in updates of the third category are also reversed, we obtain 24 public keys of RSA which is 12 more than that found by Hawkes, and determine that digiting digest is only for the data marked by *Update Length*.

Microcode updates of the fourth category only target Ice Lake microarchitecture and structure of which is the newest. The structure of the fourth category is similar with that of the third category, but no *RSA Exponent*. Fortunately, we identify a new 3072-bit *RSA Modulus* in updates of the fourth category

TABLE 5. Processors with microcode updates of the first category.

Structure	DisplayFamily_DisplayModel	Processor Families / Processor Number Series
(1)	06_01H	Intel Pentium Pro processor
(2)	06_03H 06_05H	Intel Pentium II Xeon processor Intel Pentium II processor
	06_07H 06_08H 06_0AH 06_0BH	Intel Pentium III Xeon processor Intel Pentium III processor
(3)	06_09H 06_0DH	Intel Pentium M processor
(4)	06_0EH (*Stepping* is CH)	Intel Core Duo Intel Core Solo processors
	06_0EH (*Stepping* is 8H)	Intel Core Duo Intel Core Solo processors
(5)	06_0FH	Intel Xeon processor 3000, 3200, 5100, 5300, 7300 series Intel Core 2 Quad processor 6000 series Intel Core 2 Extreme 6000 series Intel Core 2 Duo 4000, 5000, 6000, 7000 series processors Intel Pentium dual-core processors
	06_1DH	Intel Xeon processor MP 7400 series
	06_16H	undocumented

TABLE 6. RSA modulus in microcode updates of the third category.

MD5 of RSA Modulus	Processor Families / Processor Number Series	Number of updates
abd5c77f69526a248effcc0c5b12ef89	0F_06H	1
549eae1fa9ce99fe34334d010bb10330	06_1CH 06_26H	12
3aa1a2cd5ca553005644ed9fe703d1cd	06_36H	3
35d55ce4da88b080df60718f4c7c1628	06_4DH 06_37H	4
875844a569aa83265e0a5233b3f2bfa7	06_4CH	2
d653dc71463f50cec32867a1d0537bd7	06_5CH 06_5FH	10
1e3fd4b845342eb4578507df6b4201dc	06_7AH	7
13de963ff2892846f6f12bc0f40c1462	06_0FH	16
9f3760ff39ca73885062138c047c2ac9	06_17H	20
eae1670d9d534696e429aa493e018a67	06_1AH 06_1EH	16
fed1b98a5fb182b8333d3d852eecba48	06_25H 06_2CH	12
1d9d8ee879f322191be7fd9f24022955	06_2EH	3
09a729febe94ad64140c613bdcef2253	06_2AH	8
6b9e6b3f73559d24293c4d3b4058623c	06_2FH	5
ec34605e3fcea02b5be5514b2bc68c2e	06_2DH	14
a18ee2f73a0138eece08cbbe8f5e8cf8	06_3AH	9
416c6714c9fe099fa4ecfcc401d3f179	06_3EH	13
88ca270bb879e4149ec27af68f077609	06_3CH 06_45H 06_46H	33
8f1d4361ac62c70d2749bf92766e9f41	06_3FH	18
519a8f3357c18b67ae7a07a02e36d463	06_3DH 06_47H	17
d09eeba4cf5a621c8e89ad85a7f64315	06_4FH 06_56H	27
2f66117846dff5e8df2319f3696bbc15	06_4EH 06_5EH 06_8EH 06_9EH 06_A6H	86
7d5ac5adbbfc57612aaf3a7a8fd7636b	06_55H	18
a28b32d207903292d5e1632f2c86fd8b	06_66H	1

and recover the original signed data successfully by taking *RSA Exponent* as 65537.

APPENDIX A
See Figure 8.

APPENDIX B
See Table 4.

APPENDIX C
See Table 5.

APPENDIX D
See Table 6.

REFERENCES
[1] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *Proc. IEEE Symp. Secur. Privacy (SP)*, Washington, DC, USA, 2019, May 2019, pp. 19–37.

[2] M. Lipp, M. Schwarz, and D. Gruss, "Meltdown: Reading kernel memory from user space," presented at the 27th USENIX, Aug. 2018. [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-lipp.pdf

[3] Intel, Releasenote. (Jun. 16, 2020). *Intel-Linux-Processor-Microcode-Data-Files*. [Online]. Available: https://github.com/intel/Intel-Linux-Processor-Microcode-Data-Files/blob/master/releasenote

[4] *AMD AthlonTM Processor Model 6 Revision Guide, C ed*, Advanced Micro Devices, San Francisco, CA, USA, 2001.

[5] P. Koppe, B. Kollenda, and M. Fyrbiak, "Reverse engineering x86 processor microcode," presented at the 26th USENIX, Aug. 2017. [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-koppe.pdf

[6] D. D. Chen and G.-J. Ahn, "Security analysis of x86 processor microcode," Arizona State Univ., Tempe, AZ, USA, Tech. Rep., Dec. 2014, pp. 1–18. [Online]. Available: http://www.dcddcc.com/docs/2014_paper_microcode.pdf

[7] B. Hawkes. (Mar. 2013). *Notes on Intel Microcode Updates*. [Online]. Available: http://www.inertiawar.com/microcode/

[8] Intel, Platomav. (Jul. 5, 2020). *CPUMicrocodes*. [Online]. Available: https://github.com/platomav/CPUMicrocodes/tree/master/Intel

[9] M. Wilkes, "The best way to design an automatic calculating machine," *Microprocess. Microprogramm.*, vol. 8, nos. 3–5, pp. 141–144, Oct. 1981, doi: 10.1016/0165-6074(81)90018-1.

[10] A. K. Agrawala and T. G. Rauscher, *Foundations of Microprogramming: Architecture, Software, and Applications*. New York, NY, USA: Academic, 1976.

[11] Rauscher and Adams, "Microprogramming: A tutorial and survey of recent developments," *IEEE Trans. Comput.*, vol. C-29, no. 1, pp. 2–20, Jan. 1980, doi: 10.1109/TC.1980.1675451.

[12] L. H. Jones, "A survey of current work in microprogramming," *Computer*, vol. 8, no. 8, pp. 33–38, Aug. 1975, doi: 10.1109/C-M.1975.219050.

[13] K. Thompson, "Reflections on trusting trust," in *Proc. Commun. ACM*, 1975, pp. 33–38.

[14] J. Molina and W. Arbaugh, "P6 family processor microcode update feature review," College Park, MD, USA, Tech. Rep., Sep. 2000, pp. 1–12.

[15] *Intel 64 and IA-32 Architectures Software Developer's Manual Volume.3 (3A, 3B, 3C & 3D): System Programming Guide, 325384-072US ed*, Intel, Silicon Valley, CA, USA, 2020.

[16] Anonymous, (Jul. 26, 2004). *Opteron Exposed: Reverse Engineering AMD K8 Microcode Updates*. [Online]. Available: https://securiteam.com/securityreviews/5FP0M1PDFO/

[17] A. Triulzi. (Mar. 18, 2015). *Pneumonia, Shardan, Antibiotics and Nasty MOV: A Dead Hand's Tale*. [Online]. Available: https://www.troopers.de/media/filer_public/1c/a1/1ca1c699-2820-46fc-846a-2938011d7897/arrigotriulzi-troopers15-shardan.pdf

[18] A. Triulzi. (Mar. 16, 2016). *The Chimaera Processor*. [Online]. Available: https://www.troopers.de/events/troopers16/655_the_chimaera_processor/

[19] B. Kollenda, P. Koppe, M. Fyrbiak, C. Kison, C. Paar, and T. Holz, "An exploratory analysis of microcode as a building block for system defenses," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Toronto, ON, Canada, Jan. 2018, pp. 1649–1666.

[20] *Intel 64 and IA-32 Architectures Software Developer's Manual Volume. 4: Model-Specific Registers, 335592-072US ed*, Intel, Silicon Valley, CA, USA, 2020

[21] Wikipedia. *Jaccard Index*. Accessed: Aug. 14, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Jaccard_index

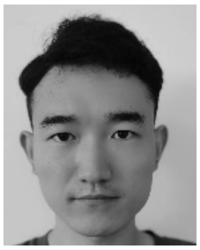
[22] G. Samb Lo and S. Dembele, "Probabilistic, statistical and algorithmic aspects of the similarity of texts and application to gospels comparison," 2015, *arXiv:1508.03772*. [Online]. Available: http://arxiv.org/abs/1508.03772

[23] C. Varol and S. Hari, "Detecting near-duplicate text documents with a hybrid approach," *J. Inf. Sci.*, vol. 41, no. 4, pp. 405–414, Aug. 2015, doi: 10.1177/0165551515577912.

[24] P. R. George, "Needleman-Wunsch algorithm," in *Encyclopedia of Genetics, Genomics, Proteomics, and Informatics*. Berlin, Germany: Springer, 2008, p. 562.

[25] L. Gwenapp, "P6 microcode can be patched," Microprocessor Rep., Sep. 1997, vol. 11, no. 12.

[26] J. A. Sutton, "Microcode patch authentication," U.S. Patent 2003 0 196 096 A1, Oct. 16, 2003. [Online]. Available: http://www.google.com/patents/WO2003088019A2?cl=en



ZHIGUO YANG is currently pursuing the master's degree with the State Key Laboratory of Mathematical Engineering and Advanced Computing, China. His research interests include reverse engineering and vulnerability of hardware and trusted computing.

QINGBAO LI is currently a Professor with the State Key Laboratory of Mathematical Engineering and Advanced Computing, China. His research interests include trusted computing and security of hardware.

PING ZHANG is currently a Professor with the State Key Laboratory of Mathematical Engineering and Advanced Computing, China. Her research interests include binary analysis and disassembly.

ZHIFENG CHEN is currently a Lecturer with the State Key Laboratory of Mathematical Engineering and Advanced Computing, China. His research interest includes network and information security.