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.
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 CPUTMicrocodes 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...
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:

- **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.

- **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.

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

- **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.

- **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 repre-
  sentative 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 Revi-
  sion 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.

- **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.
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.
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.
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 $C^2_{504}$. 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.

| Symbol | Size (B) | MD5 of Feature |
|--------|---------|----------------|
| F1     | 1056    | 999e222464b7ace0ebf1c141ebf5835 |
| F2     | 952     | d2ffaba7c4ddd40e2a655de808d3ef6e |
| F3     | 12      | b71ca2d0ea5297871e5837e3532d305c2 |
| F4     | 28      | 1e9e99e48a95fe81d388fd84b990e59f |
| F5     | 12      | 1e387c8a7046e4bed8d47d735d4f0fd |

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. 

**TABLE 2. Eleven different vectors.**

| Seq | Vector        | Number of samples |
|-----|---------------|-------------------|
| 1   | (912, -1, -1, -1) | 4                 |
| 2   | (992, -1, -1, -1) | 50                |
| 3   | (-1, 992, -1, -1) | 6                 |
| 4   | (-1, 2120, -1, -1) | 2                 |
| 5   | (-1, 3144, -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) | 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)                      |
|----------|--------|------------------------|----------|--------------------------------------|
| 48       | 0      | Unknown 1              | 4        | 0H                                   |
| 52       | 0      | Magic Number           | 4        | a1H                                  |
| 56       | 0      | Unknown 2              | 4        | 20001H                               |
| 60       | 0      | Update Revision        | 4        |                                       |
| 64       | 0      | Unknown 3              | 4        | 📜 Magic Number for 24th update       |
| 68       | 0      | Unknown 4              | 4        | 📜 Magic Number for 3rd update        |
| 72       | 0      | Date                   | 4        | 📜 Magic Number for 20th update       |
| 76       | 0      | Update Length          | 4        | 📜 Magic Number for 20th update       |
| 80       | 0      | Unknown 5              | 4        | 📜 Magic Number for 20th update       |
| 84       | 0      | Processor Signature    | 4        | 📜 Magic Number for 20th update       |
| 88       | 0      | Unknown 6              | 28       | 0H                                   |
| 116      | 0      | Flexible Field 1       | 4        | 📜 Magic Number for 20th update       |
| 120      | 0      | Flexible Field 2       | 4        | 📜 Magic Number for 20th update       |
| 124      | 0      | Unknown 7              | 20       | 0H                                   |
| 144      | 0      | Unknown 8              | 32       | 📜 Magic Number for 20th update       |
| 176      | 0      | RSA Modulus            | 256      | Varies across processor families     |
| 432      | 0      | RSA Exponent           | 4        | 11H                                  |
| 436      | 0      | RSA Signature          | 256      | 📜 Magic Number for 20th update       |

(vecors7, 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_0x00000080_0x00000046, 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 digitizing 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 digitizing 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.

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 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
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.

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.
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                       |
| 06_03H 06_05H |                             | Intel Pentium II Xeon processor                   |
| (2) 06_07H 06_08H 06_0AH 06_0BH |                     | Intel Pentium III Xeon 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_0FHF |                             | Intel Xeon processor 3000, 3200, 5100, 5300, 7300 series |
| 06_1DH |                             | Intel Core 2 Quad processor 6000 series           |
| 06_16H |                             | Intel Core 2 Extreme 6000 series                  |
| Implementation |                             | Intel Core 2 Duo 4000, 5000, 6000, 7000 series processors |
|               |                             | Intel Pentium dual-core processors                |

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/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 27th 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.dcdcc.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://secureteam.com/securityreviews/5PP0M1PDDO/

[17] A. Triulzi. (Mar. 18, 2015). Pneumonia, Shardan, Antibiotics and Nasty MOV: A Dead Hand’s Tale. [Online]. Available: https://www.troopers.de/events/troopers16/655_the_chimaera_processor/axis-15/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.

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.