Information Hiding in Standard MIDI Files Based on Velocity Reference Values

Da-Chun Wu[1], Ming-Yao Chen[2]

(Corresponding author: Da-Chun Wu)

Depart. of Computer & Communication Engineering, National Kaohsiung First University of Science & Technology[1]

Institute of Engineering Science and Technology, National Kaohsiung First University of Science and Technology[2]

No. 1, University Rd, Yanchao Dist., Kaohsiung City, 824 Taiwan, ROC.

(Email: dcwu@nkfust.edu.tw)

(Received May 7, 2015; revised and accepted July 1 & 17, 2015)

Abstract

This paper proposed a novel method to embed information in SMFs (Standard MIDI Files) by slightly adjusting the velocity values of notes. First, this method uses the velocity values of the first note of strong beat, the first of weak beat, and the first second-strong beat with non-zero velocity value in a measure as the referencing values for velocity of other notes in the same measure. Then, the data are embedded in the velocity values of notes excluding the notes with the referencing values. The method uses the difference between the velocity value of each note and its corresponding reference value to decide the number of bits which can be used for embedding data in each note. The proposed method limits the changes of the velocity value of each note to its original value and its corresponding reference value during the data embedding. It can avoid the differences from the original music being heard due to the velocity values excessive change. In addition, the proposed method can also embed data without changing the file sizes of the SMFs. It can also avoid attracting attention. The experimental results show the feasibility of the proposed method.

Keywords: Information hiding, MIDI, standard MIDI file, steganography, velocity

1 Introduction

With the progress of computer and communication technology, Internet access is no longer confined to the use of traditional PCs. Through mobile devices, such as smartphones, tablet PCs, etc., people can access the Internet anytime and anywhere; therefore, the information circulation is increasingly faster. However, security [26] issues also result. Information hiding [18, 19] is a technology whereby secret information is hidden in the images, text, voice, video and other cover-media. A media is called the stego-media after embedding secret information in a cover-media. The most common carrier media are images. Many information hiding techniques are based on images, such as LSB (least-significant-bit) [12, 14], pixel-value difference [13], difference expansion [3, 7], prediction-based [5] and DCT [17] methods.

MIDI [6, 21] (Music Instruction Digital Interface), a communication protocol, a communicate language between digital musical instruments and computers, was created in 1993. SMF (Standard MIDI File) [6, 8, 20, 21] is one of the digital music file formats. The difference between SMF and other digital music files is that it only records relevant performance data of MIDI, such as musical instruments, pitches, tempos and other messages related to the performance. The file size of SMFs is small; it can be seen not only on PCs, but also on a lot of mobile devices. It is also conveniently transferred through the Internet.

Some hiding information methods of SMFs have been proposed and developed. Duration is a parameter related to the performance of an SMF; it indicates the length of time of an event. The duration of a note played means the length of time that a note is pressed till it is released. The actual expression of same music among different players may slightly differ because players cannot precisely play the duration of each note like robots. Adli et al. [2] embed watermarks by using duration parameters. The method uses each two duration values of consecutive notes and adjusts the magnitude of duration values by one increase and one decrease. The sum value of two durations remains unchanged after embedding the watermarks. Yamamoto and Iwakiri [25] propose another similar technology to embed information by duration adjustment. The method adjusts the duration of one note played to overlap the next one. The length of overlapped time indicates the embedded information. Yamamoto and Iwakiri [24] also propose another method of embedding information by using durations. First, the average values of durations of each category of notes (example: quarter note, eighth note, etc.) are calculated. Then, the original duration value of each note is replaced with the average value of duration of the notes category. Finally, to achieve the purpose of embedding information in a note, if the note's new duration value is less than that of the original duration value, the new duration value of the note is added by the embedded message; otherwise, the new duration value is subtracted in the embedded message.

A delta-time value is placed before each event in SMFs to denote the time interval with the previous event. Dittmann and Steinebach [4] propose a method for embedding information by fine-tuning the delta-time values of MIDI files. After embedding information in the delta-time value which is placed before an event, the effect of time when the event appears will be slightly changed. Xu et al. [23] propose a method to embed encrypted watermark information into generated virtual notes. During playback, the generated virtual notes do not affect the original MIDI quality. Program-change messages are used to change the musical instruments playing in MIDI files. If more than one program-change message appears continuously in a MIDI file, only the last instrument will be retained. John [11] uses this feature to achieve the purpose of information embedding by inserting some additional program-change messages before the last program-change message. When MIDI devices read these undefined command codes during playback, they will simply ignore them and will not affect MIDI file playback. Malcolm [15] inserts some undefined command codes in the MIDI specification to deliver secret information. In the MIDI standard specification, if the same command codes next to each other appear repeatedly, only the first command code must appear; the rest of the command codes can be selected to appear or be omitted. Adli and Nakao [1] proposes an embedding method by using repeated command codes which take advantage of the feature of showing or omitting command codes to represent the embedded data, in order to achieve information hiding. Hiding information by this way will not affect the music presentation of MIDI files, but will change the size of MIDI files. In MIDI specification, SysEx commands can be used for transmitting additional messages. Adli and Nakao [1] use this feature to hide information in SMFs; the length of the embedded information is unlimited and the embedded information will not affect the performance of the music. However, the size of the MIDI files after embedding information will get larger.

In MIDI files, tempo events are used to set the actual length of time to play a quarter note of music in microseconds. It controls the playback speed of music. Yamamoto and Iwakiri [25] insert a group of tempo events to represent the embedded data. In MIDI files, several note events, especially note-on events and note-off events, may occur at the same time. The appearing order of these note events does not affect the performance in MIDI files. Inoue and Matsumoto [10] call note events which occur at the same time as simulnotes. They present the coding sequence of note events and rearrange them according to the embedded data to achieve information hiding. In order to increase security, a stegokey is used when embedding and extracting information. Stegokeys can disrupt the coding sequence; the information-hidden mechanism will be more secure. In MIDI files, the quantization function can be used to correct the start time and end time of note events. Therefore, Inoue and Matsumoto [10] use quantization function to correct the timing for increasing the amount of simulnotes and the embedding capacity. Inoue et al. [9] analyze the previous method proposed by Inoue and Matsumoto [10], and found that the SMFs with embedded information are easily inspected. Therefore, he proposed an improved method by preprocessing simulnotes in SMFs before embedding information. The preprocessing works include rearrangement of note events for each simulnote to place all note-off events before note-on events, and divide note events in each simulnote into multiple subsimulnotes which are grouped by the channel numbers of note events. Wiedemer [22] proposes a list steganography algorithm so that any events occurring at the same time are regarded as list items. First, the embedded information is converted into a flexible base number by using flexible base notation. Then, the order of list items is rearranged according to the flexible base number, and the purpose of information embedding is achieved. This method also uses a hash function to increase the security of the embedded data.

In MIDI files, the velocity value of a note event is represented by a parameter which ranges from 0 to 127. The velocity value refers to the volume when the note is played. Dittmann and Steinebach [4] use chords in MIDI files as an information carrier. Additional notes with low velocity value are joined into a chord to represent the embedded watermarks. Adli and Nakao [1] propose a method for embedding information by replacing least significant bits of velocity value; generally speaking, at most three bits are replaced in each note, so it can avoid awareness of the difference between the original file and the file after information is embedded. Adli et al. [2] use every two neighboring notes which have the same velocity values, and adjust the two velocity values by one increasing and the other decreasing for a small magnitude at the same time to embed watermarks. However, it is necessary to refer to the original MIDI file when the watermarks are extracted. Slur is a music symbol which is represented as a curve in musical scores. The curve covers two or more notes as a group and each group should play legato or smoothly without separation. Yamamoto and Iwakiri [24] propose a method to embed information into each note in the group which is covered by a slur.

This paper presents a novel method to hide information by adjusting the velocity values in SMFs. This method first uses the velocity values of the notes of the first strong beat, the first weak beat and the first second-strong beat with non-zero velocity value as reference values for other notes in the same measure. If a note is strong beat, refer to the reference value of the strong beat of the measure; if a note is weak beat, refer to the reference value of the weak beat, etc. Then, the information is embedded in

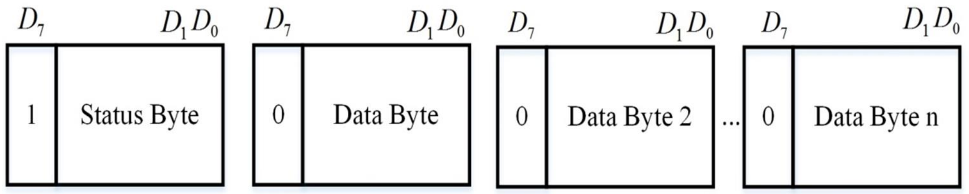

Figure 1: MIDI message structure

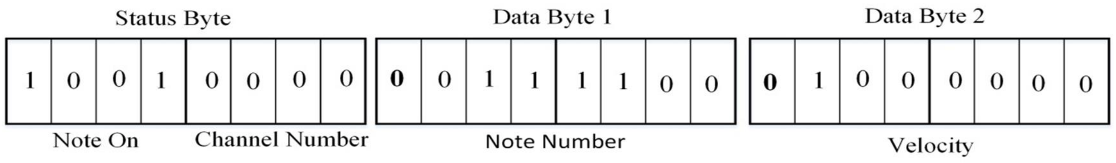

Figure 2: An example of a MIDI message

the velocity value of the notes in each measure. The proposed method can limit the changes of the velocity value of each note to its original value and its corresponding reference value when information is embedded. After embedding data, the changes of velocity values of notes are insignificant. This process can avoid the differences from the original music being heard. The file sizes of SMFs will not change after data embedding. It can also avoid attracting attention.

The rest of the sections of this paper are organized as follows. Section 2 describes MIDI and SMF. The proposed information hiding method is described in Section 3. The experimental results are presented in Section 4. Finally, conclusions are given in Section 5.

2 MIDI and SMF

MIDI is a standard communication protocol that is used to control electronic musical instruments. It allows the player to transmit the details of the performance and associated control information between electronic musical instruments, computers and other devices. The SMF file format is one of the popular digital music formats. This section will introduce MIDI and SMF.

2.1 MIDI

MIDI is composed of hardware interface and communication protocol. MIDI is used to control electronic musical instruments; it connects electronic musical instrument, computer and other digital devices together via hardware interface through transmission lines. Its commands or messages about playing music can thus be transmitted. A MIDI message consists of one status byte and several data bytes. A status byte indicates the type of transmitted message subsequently followed by data bytes with related message. Different MIDI messages may have different numbers of data bytes. The length of a status byte is fixed at one byte, but a data byte can range from 0 to several bytes, as shown in Figure 1. Status bytes and data bytes can be distinguished by their Most Significant Bits (MSBs). The MSB of a status byte is 1, and that of a data byte is 0. For example, the message generated from pressing a key of a keyboard contains one status byte and two data bytes of MIDI messages. The four high bits value of the status byte indicate the message type of note-on, and the four low bits value indicate the channel it used. Messages can be transmitted through channels 1 to 16 in MIDI devices. The information of the first data byte indicates which note (note number) is pressed, while the information of the second data byte indicates the velocity of the pressed note. As shown in Figure 2, the value of the status byte, data byte 1 and data byte 2 are 90_{16}, $3C_{16}$ and 40_{16}, respectively. It represents that a center C is pressed, the velocity value of pressing is 64, and the message is transmitted via Channel 1.

2.2 SMF

SMF is a standard MIDI file format used to store and distribute messages related to MIDI performance. It contains file format, events, timing and other information. SMFs also can be post-produced to make content richer through music sequencers. Because the file size is small, SMFs are widely used in computers, ring tones of cell phone and web pages. SMFs use chunks to store various information of MIDI performance. The structure of an SMF contains header chunk and track chunk, as shown in Figure 3. Both start with four ASCII leading characters to represent the type of chunk, followed by the length information of 32 bits which is used to indicate how many bytes of data are in the rest of the chunk. There is only one header chunk in an SMF, while there are many track

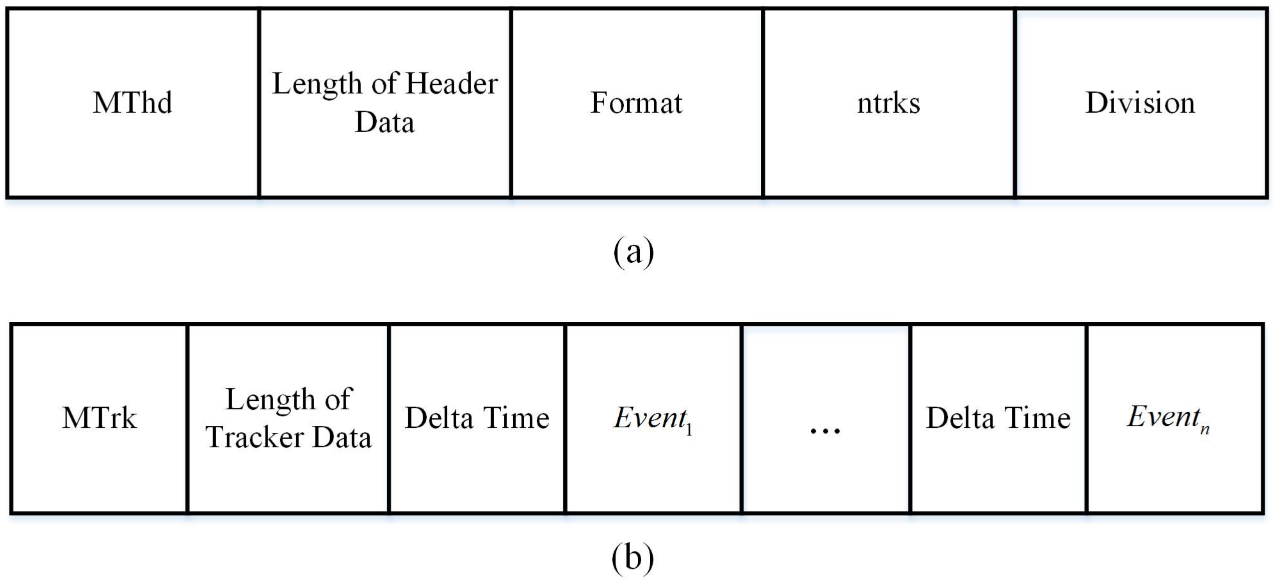

Figure 3: The chunk structure of a MIDI file: (a) Header chunk; (b) Track chunk.

chunks. The structure of an SMF starts with a header chunk followed by several track chunks.

Header chunks store the format, track number, division and other important information of an SMF. The length of division occupies 16 bits, and has two kinds of formats. If the value of MSB of division is 1, it represents that the division is time-code-based format, which is usually used in the synchronization between the video devices and MIDI devices. If the value of MSB of division is 0, it represents that this division is the tick number of a quarter note. Tick is the smallest unit of time in SMFs. This format is the commonly used division format in general SMFs. The division value is relevant to the length of actual time represented by delta time values. The leading character of a track chunk is MTrk. Track chunks are used to store the actual playing music-related information, such as playing musical instruments, note velocities, rhythms, and so on. Delta time is the time interval of two events. The delta time value is 0 when two events occur at the same time. Assuming that the division value is 48, then a delta time value of the quarter note is 48 and the delta time value of a half note is 96.

Time signature is composed of two numbers in the music score. The way they are arranged is much like fractions in mathematics, as shown in Figure 4. The number of the numerator indicates the number of beats per measure, while the number of the denominator indicates the note value of the beat, i.e. the note which is used as a beat. The time signature of SMFs is set by the time signature event. The format of the time signature event of Figure 4 is shown in Figure 5. The first 3 leading bytes of this event are FF 58 04 in hexadecimal, followed by four data bytes. The first and the second data byte denote the numerator and denominator of a time signature, respectively. Among these, the value of the second data byte is represented by negative power of two. In Figure 5, the value of the first data byte is 06; it denotes that there are six beats per measure. The value of the second data bytes is 03; it represents the value of $2^{-3} = \frac{1}{8}$ and indicates that the note value of the beat is the eighth note.

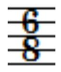

Figure 4: A time signature

3 Information hiding in a SMF

When playing music, an appropriate change of sound velocity can make the music much pleasant. Generally speaking, the strength of the velocity often appears regularly and periodically in music according to the time signature of the scores. Usually, the first beat in each measure of a song is a strong beat, and the second beat is a weak beat. So, the velocity values of the first note beat should be greater than that of the second note beat in the same measure. The velocities of the remaining beats in the measure exhibit different performance strength according to the time signature of the music. Using 4/4 time signature as an example, it represents that a quarter note is used as one beat, and there are four beats per measure. In addition to the first and second beat of each measure, there are strong beat and weak beat, respectively; the third beat of each measure is second-strong beat and the fourth beat of each measure is weak beat. Table 1 lists the common time signature and its strength performance of each beat in a measure.

FF	58	04	06 (Data byte1)	03 (Data byte2)	(Data byte3)	(Data byte4)

Figure 5: The format of the time signature event of Figure 4

Table 1: Strength of beat of common time signature

Time signature	The strength of the expression of each beat in each measure
2/4	Strong, weak
3/4	Strong, weak, weak
4/4	Strong, weak, second-strong, weak
3/8	Strong, weak, weak
6/8	Strong, weak, weak, second-strong, weak, weak
12/8	Strong, weak, weak, second-strong, weak, weak, second-strong, weak, weak, second-strong, weak, weak

This paper proposes a method to hide information in SMFs by embedding data in the velocity values of notes. The method uses time signature to decide the strength of the performance of each note in the music. Then, the two velocity values of the first strong beat note n_s and weak beat note n_w with non-zero value in each measure are used as the reference velocity values: r_s and r_w of strong beat and weak beat of the measure, respectively. If the strength of performance of a song includes a second-strong beat according to its time signature, then assume the reference value of the first second-strong beat note's non-zero value n_{ss} is r_{ss}. The embedding data will be embedded in the notes with non-zero velocity value in the same measure, excluding n_s, n_w and n_{ss}. For each note n which satisfies the above conditions, assume its velocity value is v_n, and the reference velocity value of n is r_n according to the strength of the performance. Then this method can embed $\lfloor \log_2 |v_n - r_n| \rfloor$ bits of secret information at most in the n. The larger difference value between v_n and r_n indicates that the volume of playing the note is not generally expected. It also means that the larger amount of data can be embedded in n. Because the number of bits occupied by a velocity value does not change after embedding data in it, the size of MIDI files remains unchanged while embedding the data. Using a measure of the song 'You can fly' as an example, the score is shown in Figure 6. The time signature of this song is 4/4, which represents that a quarter note is used as one beat, and there are four beats per measure. The strength of the expression of each beat in each measure is strong, weak, second-strong, and weak. The notes a and b shown in the figure represent the first and the second beat of the notes of this measure, respectively. Note c and d represent the notes of the third beat. Note e and f represent the fourth beat of the notes. Among these, notes a and b are the first strong beat note n_s and the first weak beat note n_w with non-zero velocity value for this measure, respectively, while note c is the first second-strong beat note n_{ss} with non-zero velocity value. Note d is the second-strong beat note; both notes e and f are weak beat notes. Assume that the velocity values of note a, b, c, d, e and f are 97, 89, 92, 84, 85 and 88, respectively, and the embedded binary secret bit message is 010, then the velocity values of n_w and n_{ss} are 89 and 92, respectively. The method proposed in this paper can embed information in note d by using r_{ss} as the velocity reference value, and embed information in notes e and f by using r_w as the velocity reference value.

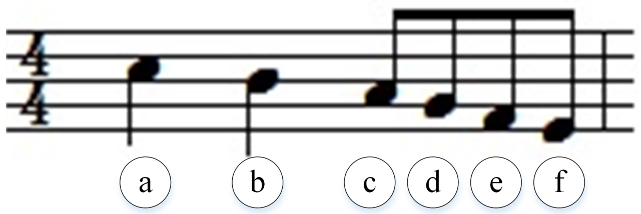

Figure 6: Score of a measure

In order to extract the secret information correctly during the extraction process, the information embedded in each note retains one leading bit with bit value 1, and the remaining embedded bit can be used to embed secret information. In other words, if the total number of data bits can be embedded in the notes is t, then the actual number of bits available for embedding secret information is t-1. The velocity reference value of note d is 92; the total number of data bits can be embedded in this note is $\lfloor \log_2 |84 - 92| \rfloor = 3$. The embedded information of note

d contains a leading bit 1 and the secret binary information 10; therefore, the complete binary information is 110. The velocity reference values of note e and f are 89; the number of secret information bits embedded in note e, f are $\lfloor \log_2 |85 - 89| \rfloor = 2$, $\lfloor \log_2 |88 - 89| \rfloor = 0$, respectively. The embedded information of note e contains a leading bit 1 and the secret binary information 0; therefore, the complete binary information is 10. Because the number of bits which can be embedded in note f is 0, this note is not used for embedding information. Detailed descriptions of embedding and extraction procedures are shown in the following section.

3.1 Information Embedding Procedure

This section will describe how secret information is embedded in the SMFs. The embedding procedures are as follows:

Input: the original cover SMF C, bit length l of secret information $I = i_1 i_2 \cdots i_l$.
Output: stego-SMF S after embedding I in C.

Step 1. According to the time signature of music and strength of beat of common time signature in Table 1, we obtain the information about strong beat, weak beat, and second strong beat in each measure of music. Set j to 1, and perform Step 1.1 for each measure M_j of music one by one until the secret information is completely embedded.

Step 1.1: Let n_s and n_w be the first strong beat note and the first weak beat note with non-zero velocity value in M_i, respectively. Assume the velocity values of n_s and n_w are r_s and r_w, respectively. If there are a second strong beat in M_i, let the first second strong beat note with non-zero velocity value be n_{ss}, and the velocity value of n_{ss} be r_{ss}. Perform Steps 1.1.1 through 1.1.7 by using each note n_i with non-zero velocity value in M_i as the note variable n one by one except n_s, n_w, n_{ss} and if any.

Step 1.1.1: Let the corresponding velocity reference value of n be r_n; r_n can be represented as

$$r_n = \begin{cases} r_s, & if\ n_i \ \in \ strong\ beat \\ r_w, & if\ n_i \ \in \ weak\ beat \\ r_{ss}, & if\ n_i \ \in \sec ond\ strong\ beat \end{cases}$$

Step 1.1.2: Assume the original velocity value of n is v_n. If the value of $|v_n - r_n|$ is less than 2, it indicates that n cannot be used for embedding data. Then, go to Step 1.1.7.

Step 1.1.3: The number of bits that can be embedded in n is e, e can be expressed as

$$e = \lfloor \log_2 |v_n - r_n| \rfloor$$

where $\lfloor \bullet \rfloor$ denotes the floor function.

Step 1.1.4: The embedded data of note n contains a leading bit 1 and e-1 bit secret binary information. Assume f is the embedded data, f can be expressed as

$$f = \begin{cases} 1, & if\ e = 1 \\ 1 \bullet i_j i_{j+1} \cdots i_{j+e-2}, & if\ e \geqslant 2 \end{cases}$$

where $\bullet$ denotes the symbol of bit concatenation.

Step 1.1.5: Let $v_n^{'}$ be the new velocity value after embedding f into v_n, $v_n^{'}$ can be expressed as

$$v_n' = \begin{cases} r_n + BinToDec(f), & if\ v_n > r_n \\ r_n - BinToDec(f), & if\ v_n < r_n \end{cases}$$

where $BinToDec(x)$ denotes the function which converts the binary bit stream x into a decimal number.

Step 1.1.6: Set j to j+e-1.

Step 1.1.7: Continue.

Step 2. Obtain stego-SMF S after embedding I.

3.2 Information Extraction Procedure

Extraction procedure is essentially the reverse process of embedding procedure. If the difference value between the velocity value of a note and its corresponding reference value is more than 1, it indicates that the note is with embedded data. Therefore, the data can be extracted through the reverse process of embedding information.

4 Experimental Results

This section presents the experimental results of the proposed method. Because this method uses the velocity values of the first strong beat, the first weak beat and the first second-strong beat note as the velocity reference values, the difference value between the velocity reference value and the velocity value of a note is used to embed data. While a large amount of difference value represents that it can provide more bits for embedding data, this method cannot embed data in the velocity values of notes if the velocity values are the same as the corresponding velocity reference values. The tested MIDI files were collected from a web site [16]. There are 75 MIDI files which contain different time signatures. These MIDI files originate from the original work of eight authors. Six tested files are chosen from among them to show the detailed experimental results. They are dvoe1, chonorain, chonoc10, han-fir-2-bourree, pcanon-in-d-for-guitar and romance-in-f-op-51. Table 2 shows time signature, file size and number of measures that can be embedded data by using six tested MIDI files, respectively.

Table 3 shows beats per note, number of notes, number of notes that can be embedded data, ratio of notes

Table 2: Time signature, file size and number of measures that can be embedded data of six tested MIDI files

File name	Time signature	File size (K Bytes)	Number of measures that can be embedded data
dvoe1	4/4	29.7	574
chonorain	4/4	11.6	82
chonoc10	4/4	23.8	278
han-fir-2-bourree	4/4	20.3	120
pcanon-in-d-for-guitar	4/4	4.82	35
romance-in-f-op-51	4/4	25.2	105

Table 3: Beats per note, number of notes, number of notes that can be embedded data, ratio of notes that can be embedded data and embedding capacity of six tested MIDI files

File name	Beats per note	Number of notes	Ratio of notes that can be embedded data	Embedding capacity		
				Bits	Bits per note	Bits per embedding note
devoe1	0.4459	5149	0.3650	5165	1.0031	2.7488
chonorain	0.1696	1934	0.3475	2141	1.1070	3.1860
chonoc10	0.6387	1741	0.3297	1234	0.7088	2.1498
han-fir-2-bourree	0.2155	2227	0.7800	2961	1.3240	1.7047
pcanon-in-d-for-guitar	0.2439	574	0.2875	415	0.7230	2.5151
romance-in-f-op-51	0.1317	3187	0.7358	5994	1.8807	2.5561

that can be embedded data, and embedding capacity of six tested MIDI files. Beats per note represents the average number of beats of notes in a MIDI file. Generally speaking, the smaller the value of beats per note, the more notes per measure, i.e. the more notes that can be used for embedding data. The number of notes that can be embedded data refers to the sum of the number of notes with non-zero velocity values, except for the velocity reference notes of strong beat, weak beat and second-strong beat of each measure in the MIDI file. The ratio of notes that can be embedded data represents the proportion of the notes that can be embedded data to the whole amount of notes in a MIDI file. The larger the value of ratio of notes that can be embedded data, the smaller the value of beats per note. For example, romance-in-f-op-51 MIDI file in Table 4 has a relatively high ratio of notes that can be embedded data and has relatively low beats per note. There are several ways to represent the embedding capacity of each MIDI file in Table 3. Bits per note indicate the amount of the embedded bits per note in a MIDI file. It can be expressed as:

$$bits-per-note = \frac{total\ no.\ of\ embedded\ bits}{total\ no.\ of\ notes\ in\ the\ MIDI\ file}$$

Bits per embedding note' refers to the total number of notes with embedded data in a MIDI file. It can be expressed as:

$$bits-per-embedding-note = \frac{total\ no.\ of\ embedded\ bits}{total\ no.\ of\ notes\ with\ embedded\ data\ in\ a\ MIDI\ file}$$

The larger the bits per embedding note' signifies the larger the average difference between the velocities of the notes and their reference values in the MIDI file, and the more capacity that can be used for embedding data. As shown in Table 4, the value of bits per embedding note of chorain MIDI file is relatively larger than those of the others. It signifies that the differences between the velocity values and their reference values of the notes in the chorain MIDI file are relatively larger than those of the others.

Lastly, Table 4 shows the total number of MIDI files for each author, the average number of notes of MIDI files, the average beats per note of MIDI files, the average bits per note of MIDI files after embedding data and the average bits per embedding note of the whole 75 MIDI files which are all classified by the authors. First, the results of the average number of notes, the average bits per note of MIDI files after embedding data and the average beats per note of each author are summed up the number of notes, bits per note, and beats per note of each author. Second, divide the sum values by the total number of MIDI files for each author, respectively. In Table 4, the average bits per note after data were embedded in MIDI files of author Brams is relatively small. This is because the differences

Table 4: Beats per note, number of notes, number of notes that can be embedded data, ratio of notes that can be embedded data and embedding capacity of six tested MIDI files

Author	Number of MIDI files	Average number of notes	Average beats per note	Average bits per note after data embedded	Average bits per embedding note
Brahms	1	11921.0	0.7932	0.0084	2.1333
Chopin	12	1369.4	0.7971	0.5246	1.8767
Dvarok	10	8172.7	0.7532	0.7223	2.7638
Handel	17	5519.5	0.7823	0.3834	2.7424
Pachelbel	5	3146.0	1.1864	0.6073	3.1772
Schubert	1	1316.0	0.5091	0.3989	1.9590
Tchaikovsky	11	17732.4	0.6772	0.9232	2.3192
Vavilda	18	5853.2	0.8016	0.2240	2.2541

between the velocity values and their reference values of the notes in the MIDI files of author Brams are near zero for most of the notes in his MIDI files. In Table 4, the average beats per note for an individual author may represent the expression of the music for each author. The larger value represents that one's music is played fast, and may present a brisk and dancing style. On the contrary, if the music is played slowly, it may present a lyrical and restrained feeling.

5 Conclusions

This paper proposed a new method to hide information in SMFs. This method uses the velocity values of the first note of strong beat, the first of weak beat, and the first second-strong beat with non-zero velocity value in a measure as the referencing values for velocity of other notes in the same measure. The proposed method can limit the changes of the velocity value of each note to its original value and its corresponding reference value when data is embedded. It can avoid the differences from the original music due to excessively changed velocity value, being heard.

In addition, the proposed method can embed data without changing the file sizes of SMFs, and also avoid attracting attention. Therefore, the proposed method provides a secure way to embed data in SMFs.

Acknowledgments

This work was supported partially by National Science Council, R. O. C. under Grant No. 100-2221-E-327-029.

References

[1] A. Adli and Z. Nakao, "Three steganography algorithms for midi files," in *IEEE International Conference on Machine Learning and Cybernetics*, vol. 4, pp. 2401–2404, Aug. 2005.

[2] A. Adli, H. Mirza, and Z. Nakao, "A watermarking approach for midi file based on velocity and duration modulation," in *Knowledge-Based Intelligent Information and Engineering Systems*, pp. 133–140, 2008.

[3] O. M. Al-Qershi and B. Ee Khoo, "Two-dimensional difference expansion (2D-de) scheme with a characteristics-based threshold," *Signal Processing*, vol. 93, pp. 154–162, Jan. 2013.

[4] J. Dittmann and M. Steinebach, "A framework for secure midi ecommerce," in *German National Research Center for Information Technology*, 2002.

[5] I. C. Dragoi and D. Coltuc, "Local-prediction-based difference expansion reversible watermarking," *IEEE Transactions on Image Processing*, vol. 23, pp. 1779–1790, Apr. 2014.

[6] R. Guerin, *MIDI Power: The Comprehensive Guide*, Cengage Learning PTR, 2005.

[7] Li C. Huang, L. Yu Tseng, and M. S. Hwang, "A reversible data hiding method by histogram shifting in high quality medical images," *Journal of Systems and Software*, vol. 86, pp. 716–727, Mar. 2013.

[8] D. M. Huber, *The MIDI Manual: A Practical Guide to MIDI in the Project Studio*, Focal Press, 3 edt., 2007.

[9] D. Inoue, M. Suzuki, and T. Matsumoto, "Detection-resistant steganography for standard mimi files," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 86, pp. 2099–2106, 2003.

[10] D. Inoue and T. Matsumoto, "A scheme of standard midi files steganography and its evaluation," in *Proceedings of Security and Watermarking of Multimedia Contents IV*, SPIE 4675, Apr. 29, 2002.

[11] C. John, *Steganography V - Hiding Messages in MIDI Songs*, Aug. 5, 2004. (`http://www.codeproject.com/Articles/5390/Steganography-V-Hiding-Messages-in-MIDI-Songs`)

[12] M. Juneja and P. S. Sandhu, "Improved lsb based steganography techniques for color images in spatial domain," *International Journal of Network Security,*, vol. 16, pp. 452–462, Nov. 2014.

[13] Y. Po Lee, J. C. Lee, W. K. Chen, Ko C. Chang, I. J. Su, and C. P. Chang, "High-payload image hiding with quality recovery using tri-way pixel-value differencing," *Information Sciences*, vol. 191, pp. 214–225, May. 2012.

[14] T. C. Lu, C. Ya Tseng, and J. H. Wu, "Dual imaging-based reversible hiding technique using lsb matching," *Signal Processing*, vol. 108, pp. 77–89, Mar. 2015.

[15] J. W. Malcolm, *Method and Apparatus for Encoding Security Information in a MIDI Datastream*, Patent: US 6798885 B1, Sep. 2004.

[16] MIDI Music, "http://goo.gl/ryqel4,".

[17] S. Onga, K. S. Wonga, and K. Tanaka, "Scrambling embedding for JPEG compressed image," *Signal Processing*, vol. 109, pp. 38–53, Apr. 2015.

[18] K. Patel, S. Utareja, and H. Gupta, "A survey of information hiding techniques," *International Journal of Emerging Technology and Advanced Engineering*, vol. 3, pp. 347–350, Jan. 2013.

[19] C. P. Sumathi, T. Santanam, and G. Umamaheswari, "A study of various steganographic techniques used for information hiding," *International Journal of Computer Science & Engineering Survey*, vol. 4, pp. 9–25, 2013.

[20] The International MIDI Association, *Standard MIDI-file Format Spec. 1.1, Updated*," 2003.

[21] The MIDI Manufacturers Association, *The Complete MIDI 1.0 Detailed Specification Ver. 96.1*, 1996.

[22] M. Wiedemer, *MIDI File Steganography*, Patent: US 7402744 B1, July 22, 2008.

[23] C. Xu ane Y. Zhu and D. D. Feng, "Content protection and usage control for digital music," in *First IEEE International Conference on Web Delivering of Music*, pp. 43–50, 2001.

[24] K. Yamamoto and M. Iwakiri, "An information hiding technique to music code with musical performance rendering," in *Youngnam-Kyushu Joint Conference*, 2009.

[25] K. Yamamoto and M. Iwakiri, "A standard midi file steganography based on fluctuation of duration," in *IEE Availability, Reliability and Security*, pp. 774–779, Mar. 2009.

[26] Y. Zhang, X. Li, H. Li, and H. Zhu, "Subliminal-free variant of schnorr signature with provable security," *International Journal of Electronics and Information Engineering*, vol. 2, pp. 59–68, June 2015.

Da-Chun Wu was born in Taiwan on June 6, 1959. He received the B. S. degree in the Department of Computer Science from Tamkang University, Taipei, Taiwan, 1983, the M. S. degree in the Institute of Information Engineering from Tamkang University in 1985. He received the Ph.D. degree in computer science from Chiao Tung University, Hsinchu, Taiwan in 1999. Dr. Wu joined the faculty of the Department of Information Management, Ming Chuan University, in 1987. He joined the faculty of National Kaohsiung First University of Science and Technology (NKFUST), Kaohsiung, Taiwan in August 2002. He is currently the Head and Associate Professor in the Department of Computer and Communication Engineering. Professor Wu's current research interests include image processing, database, multimedia security.

Ming-Yao Chen was born in Kaohsiung on May 26, 1967. He received the M.S. degree in Department of Computer and Communication Engineering from the National Kaohsiung First University of Science and Technology, Kaohsiung, Taiwan, in 2003. He is currently a Ph.D. student at Institute of Engineering Science and Technology, National Kaohsiung First University of Science and Technology, Kaohsiung, Taiwan. His current research interests include image processing, multimedia steganography.