Steganography on Synthesizer V SVP File Format

Maria Flora Renata Siringoringo - 13522010[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522010@std.stei.itb.ac.id, [2]mariaflorarenata12@gmail.com

***Abstract*—In modern times, steganography is often done on audio files. The Synthesizer V software uses the SVP file format to store project information. This type of file can be used as a cover file for steganography while still being functional as a project file and can ultimately be rendered into audio. We propose and compare two embedding methods: one utilizing note timing offset values, and another utilizing voice parameter curve points. Both methods modify the least significant bits of floating-point values within the JSON structure, embedding secret information while preserving project functionality.**

***Keywords*—LSB, Steganography, Synthesizer V**

I. Introduction

The need for secret communication has always been present. While making the content of communication incomprehensible to others might be enough most of the time, people sometimes need to hide the existence of the exchange in the first place. This can be done by using steganography.

Steganography is the science of hiding data inside other data. In recent years, digital file types such as audio (WAV or MP3) and images (JPG or PNG) have been used as covers to hide secret messages in. An alternative approach involves embedding data within structured project files used by digital audio workstations and similar software, where information can be hidden in musical parameters before rendering occurs.

This paper applies the LSB steganography technique to the SVP file format used by Synthesizer V, a vocal synthesizing engine and editor. SVP files are structured as JSON documents containing project metadata, tracks, and information related to the tracks such as the notes and vocal parameters.

II. Steganography

Steganography is the science of concealing the existence of data by hiding it inside another media, called the cover. The word steganography itself comes from the Greek combination of the word *steganos*, meaning “hidden”, and *graphien*, meaning “writing”. Unlike cryptography, which focuses on making data unreadable, steganography aims to make the data itself undetectable.

Current steganography methods often include using digital files as the cover media, most often in the form of image or audio. These files usually contain redundant or unimportant information, such as noise. Most steganography methods would embed the secret data inside these noise parts, as changing a few bits in each part would not majorly alter the resulting file.

One of the simpler methods is LSB steganography, in which the least significant bits of the cover file would be replaced by the secret message in order. If LSB steganography is done in the noise component of a file, the change would generally not be noticeable. Further improvements to this technique have also been developed to increase its security, such as embedding in random elements instead of in succession.

There are different ways to measure the success of a steganography method. One of the metrics that could be used is peak signal-to-noise-ratio (PSNR). PSNR is typically used to measure the quality of images and audio, with a higher PSNR meaning a higher quality. In steganography, a higher PSNR means higher similarity between the original media and the steganography output. PSNR is defined by eq. 1.

$$PSNR = 10 \log_{10}\left(\frac{MAX^2}{MSE}\right) \quad (1)$$

Where MAX is the largest fluctuation of the image, and MSE is the mean square error between the original media and the output of the steganography. The formula for MSE is defined by eq. 2

$$\mathrm{MSE} = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2 \quad (2)$$

Where I is the original image of size m x n and K is the steganography output of I. For audio steganography, MSE can be calculated similarly, except without the n as an audio’s value mas one dimension (array) instead of two.

III. Synthesizer V

Synthesizer V (Synth V) is a voice synthesizer software developed by Dreamtonics with focus on singing. Synth V and its editor was first released on 2018 and has since received two major updates to the editor, now called Synthesizer V Studio 2. The Synthesizer V editor functions like a DAW, letting users synthesize vocals and do further editing on them.

To synthesize vocals, the user must make a track, choose the singer (called voicebanks), write notes on to the track, and then put lyrics on the notes. The engine will synthesize vocals by breaking the lyrics into phonemes, As the voicebanks contain recordings of each phoneme by real singers, the lyrics entered by users into the editor will be broken down into phonemes, and the engine will play the corresponding recording. The engine also uses AI to synthesize expressions for the vocals, which the user can customize.

A Synth V project file is saved in the SVP file format. The SVP file format uses JSON format, making it easy to edit outside of the editor. While the file format itself has no official documentation, a list of each name in the JSON and a general description of its value are displayed in table 1.

Table I: Structure of SVP File Format

No.	Name	Value
1	version	Unknown
2	time	Information regarding the time signature and tempo of the song
3	library	Unknown
4	tracks	List of tracks. Each track is a dictionary with the keys: - Name: Name of the track - dispColor: Color of track in the editor - dispOrder: The order of the track in the editor - renderEnabled: Whether the track is chosen for rendering or not - mixer: Values on the track's mixer such as the gain and the pan. - mainGroup: Most of the information related to the track's content is stored here, such as all the parameters of the tracks and the list of notes in the track. The notes hold information such as the duration, the phoneme sounded, etc. - mainRef: Metadata of the track, such as the voice used for the track, the system pitch, blick and pitch offsets, etc. - groups: unknown
5	renderConfig	Saved configuration for rendering the project

IV. Methodology

A. Steganography Design

The large number of fields inside the SVP file format allows for a lot of flexibility on where the secret data will be embedded. A few alternatives were considered for this paper's implementation.

1. Vocal parameters

   A track's vocals have several editable parameters, such as loudness, breathiness, tension, etc. In the editor, this is shown as a line graph which the user can add nodes to.

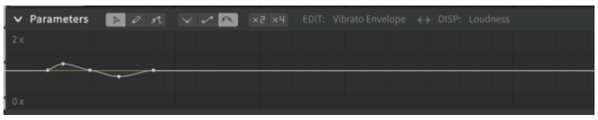


Figure 1: Vocal parameters editor

   Inside the SVP file, these parameters are stored under tracks.mainGroup.parameters as a dictionary with the parameter's name as the key and the mode and a list of points as the value. The list is formatted such that the even indices are the positions of the nodes, and the odd indices are the value of the nodes itself. This list is initially empty, with values inserted as nodes are created in the editor.

   There are eight parameters available for most voicebanks, thus giving a lot of space for data insertion. However, parameters that has nodes and parameters that has not been edited would show up differently in the editor, as the parameter name would be in different colors, and the nodes would be clearly visible. This is not optimal for hiding, as it would be obvious that something has been inserted. To avoid this, data should only be inserted in existing nodes instead of new nodes. The drawback of this method is that steganography could only be done in project files that have already gone through the editing process, usually called tuning, and cannot be done on files distributed through the internet as they typically come untuned.

2. Note offset

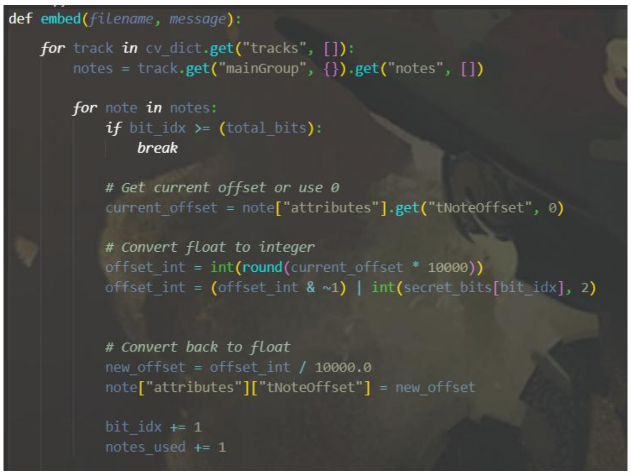
   Each note in a track could have several attributes. One of such attributes is called offset, which controls how early or late a note would be sung compared to its actual position on the track. Small changes to this value would not be noticeable to most people, and a changed offset value would not be visible unless the note itself is selected.

   These values are stored in tracks.mainGroup.notes.attributes. The key for offsets is tNoteOffset. This key and its value would not be present unless the offset has been edited, but it can be inserted through text editors, and the Synth V editor would still be able to open and parse the values normally. The drawback of this method is that there would typically be fewer hiding spots available compared to the first method.

V. IMPLEMENTATION

Both approaches will be implemented on Python with the json library. The program will take two to three inputs: mode (embed or extract), the cover file, and the secret message, if embedding.

During embedding, the program will add a start signature and an end signature to the message. Then, the message will be broken into bits. The program will then insert the bits into either the vocal parameters or the note offsets using the LSB method. As both vocal parameter values and note offsets are stored as floats, the program will convert the float to an integer by multiplying it by 10000 during LSB and then convert it back to float before writing. The program does not count whether the message fits into the file beforehand and will simply throw an error if it ran out of parameters/notes before the whole message is embedded.



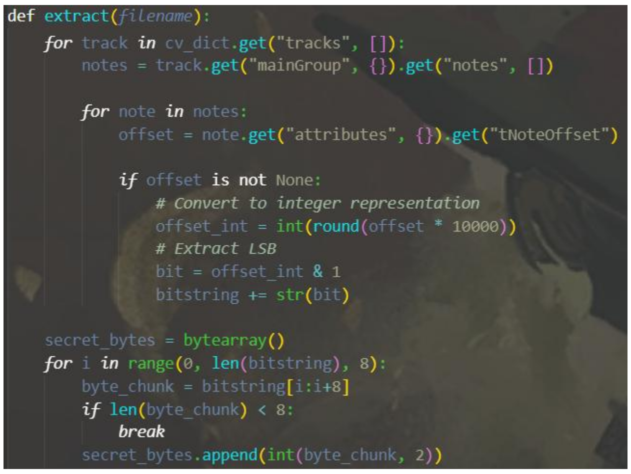
```python
def embed(filename, message):
    for track in cv_dict.get("tracks", []):
        notes = track.get("mainGroup", {}).get("notes", [])

        for note in notes:
            if bit_idx >= (total_bits):
                break

            # Get current offset or use 0
            current_offset = note["attributes"].get("tNoteOffset", 0)

            # Convert float to integer
            offset_int = int(round(current_offset * 10000))
            offset_int = (offset_int & ~1) | int(secret_bits[bit_idx], 2)


            # Convert back to float
            new_offset = offset_int / 10000.0
            note["attributes"]["tNoteOffset"] = new_offset

            bit_idx += 1
            notes_used += 1
```

Figure 2: Embedding with the offset approach

With the vocal parameter approach. The program would loop through every track, get the key “parameters” in “mainGroup”, and then loop every key of the parameters dictionary to get each “points” array. It will then do LSB on the array, starting at index 1 and skipping one index so that it only takes odd indices.

With the note offset approach, the program would similarly loop through every track. It will then get the “notes” array in “mainGroup”. For each note in the array, it will get the value for the key “tNoteOffset” in the “attributes” of the note or use 0 if it doesn’t find it. It will then do LSB on the value and add the tNoteOffset key to the attributes dictionary if it did not have one

Extracting is done similarly to embedding. First, every possible bit is extracted from the cover using the same method as embedding. Second, the program tries to find the start and end signatures and throws an error if either isn’t found. Finally, the program decodes the bytes between the start and end signatures and prints it to the screen.



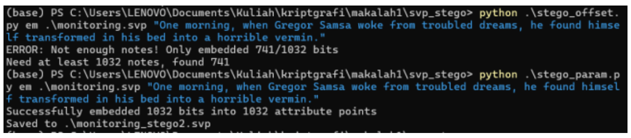
```python
def extract(filename):
    for track in cv_dict.get("tracks", []):
        notes = track.get("mainGroup", {}).get("notes", [])

        for note in notes:
            offset = note.get("attributes", {}).get("tNoteOffset")

            if offset is not None:
                # Convert to integer representation
                offset_int = int(round(offset * 10000))
                # Extract LSB
                bit = offset_int & 1
                bitstring += str(bit)

    secret_bytes = bytearray()
    for i in range(0, len(bitstring), 8):
        byte_chunk = bitstring[i:i+8]
        if len(byte_chunk) < 8:
            break
        secret_bytes.append(int(byte_chunk, 2))
```

Figure 3: Extracting with the offset approach

VI. TESTING AND ANALYSIS

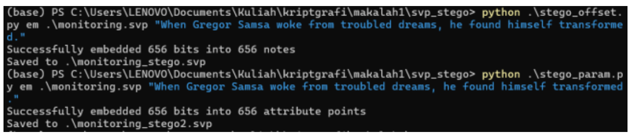
To measure the success of each steganography method, the SVP before and after the steganography will be rendered using the editor, and then the PSNR will be calculated.

For the first test, the file used is called monitoring.svp, which the author made and tuned. It has one vocal track and no instrumental tracks.



```
(base) PS C:\Users\LENOVO\Documents\Kuliah\kriptgrafi\makalah1\svp_stego> python .\stego_offset.
py em .\monitoring.svp "One morning, when Gregor Samsa woke from troubled dreams, he found himse
lf transformed in his bed into a horrible vermin."
ERROR: Not enough notes! Only embedded 741/1032 bits
Need at least 1032 notes, found 741
(base) PS C:\Users\LENOVO\Documents\Kuliah\kriptgrafi\makalah1\svp_stego> python .\stego_param.p
y em .\monitoring.svp "One morning, when Gregor Samsa woke from troubled dreams, he found himsel
f transformed in his bed into a horrible vermin."
Successfully embedded 1032 bits into 1032 attribute points
Saved to .\monitoring_stego2.svp
```

Figure 4: Comparison of length hidden by approaches



```
(base) PS C:\Users\LENOVO\Documents\Kuliah\kriptgrafi\makalah1\svp_stego> python .\stego_offset.
py em .\monitoring.svp "When Gregor Samsa woke from troubled dreams, he found himself transforme
d."
Successfully embedded 656 bits into 656 notes
Saved to .\monitoring_stego.svp
(base) PS C:\Users\LENOVO\Documents\Kuliah\kriptgrafi\makalah1\svp_stego> python .\stego_param.p
y em .\monitoring.svp "When Gregor Samsa woke from troubled dreams, he found himself transformed
."
Successfully embedded 656 bits into 656 attribute points
Saved to .\monitoring_stego2.svp
```

Figure 5: Successful embedding

In a tuned Synth V project, there would typically be more parameter points than notes. Fig. 4 shows the parameter approach being able to hide more bits than the offset approach.

The second test will be done on BUTCHERVANITY.svp, distributed by the creator of the song. The author did minimal tuning to accommodate for the parameter approach. This SVP has two vocal tracks and one instrumental track. The results of both tests are displayed on table II.

Table II: Testing Results

No.	Input	Metric	Parameter	Offset
1	When Gregor Samsa woke from troubled	SVP can be opened	Yes	Yes
		Extracted secret	Yes	Yes

	dreams, he found himself transformed.	= input		
		MSE	9.1616656e-08	1.3613804e-07
		PSNR	70.38026 dB	68.66021 dB
2	Far far away, behind the word mountains, far from the countries Vokalia and Consonantia.	SVP can be opened	Yes	Yes
		Extracted secret = input	Yes	Yes
		MSE	5.0024457e-11	6.8731015e-06
		PSNR	103.00818 dB	51.62847 dB

Testing shows that the parameter approach achieves a smaller MSE than the offset approach, thus also having a higher PSNR. As higher PSNR is better, this shows that the parameter method is better at hiding information that offset.

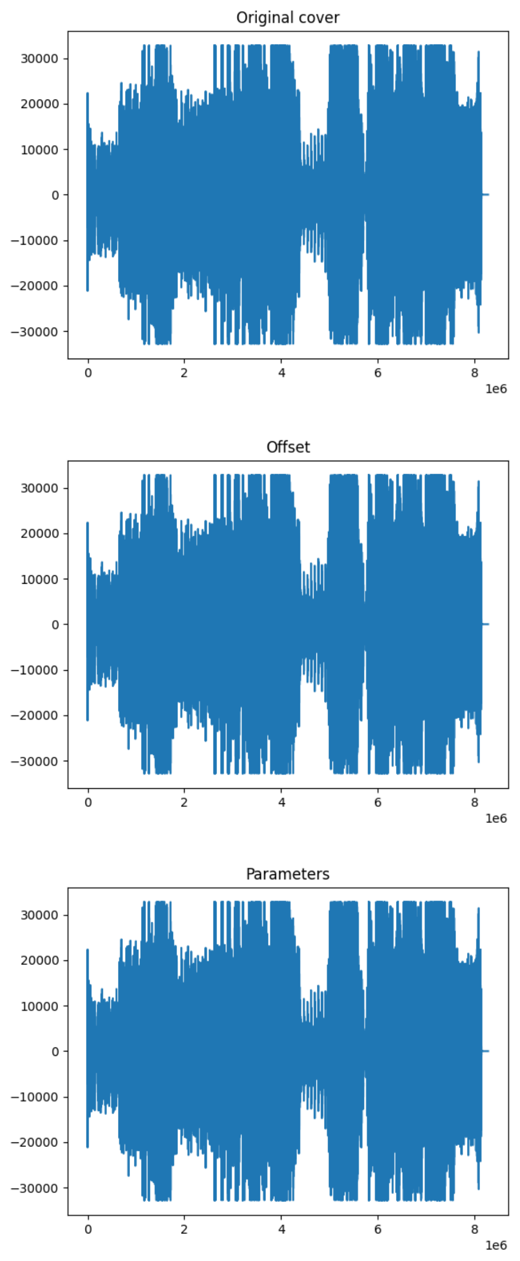


Figure 6: Spectrogram of the audio files

Fig. 6 shows the spectrograms for the original cover and both stego audios for test 2. All spectrograms are similar to each other, which shows that the difference between each audio is not too noticeable, despite the relatively low values of the PSNR during tests.

One thing to note is that during testing, the PSNR of both approaches sometimes change when re-rendering, despite no editing happening. This might be because of the Synth V engine's rendering process, which might introduce noise on its own. Despite sometimes resulting in bad PSNR for the stego audios, this behavior provides cover for the LSB modifications, as PSNR based detection becomes less definitive.

The parameter method consistently achieves a higher PSNR compared to the offset method. This difference might be attributed to how the synthesis engine processes these modifications. Note timing offsets affect a lot of synthesis parameters, as a note starting earlier or later might change how and when each phoneme is sounded by the synthesis engine. Even minimal changes to tNoteOffset might trigger recalculations by the engine. While this change typically isn't too discernable to the human ear, it would result in a smaller PSNR score.

In contrast, vocal parameters such as loudness, tension, and tone shift operate as continuous modulation curves applied during or after phoneme synthesis. While it would change how the vocals sound, it would not affect any timings or the phoneme generations. The changes introduced by manipulating these parameters would also only affect the area bound by the point changed and the points immediately before and after it.

Additionally, the parameter method only modifies pre-existing points created during the original composition and tuning, thus all modified values would still be relevant towards the current musical context. While the offset method could similarly restrict embedding to notes with existing tNoteOffset values to improve stealth, this would severely limit embedding capacity to impractical levels. Conversely, Synth V projects usually contain extensive modifications to parameters with a high number of points, providing higher capacity while maintaining superior audio fidelity.

VI. Conclusion

The SVP file format can be used for steganography, as it is based on JSON with a large number of fields, making insertion and extraction of hidden data relatively easy. We provide two methods of performing steganography on these files, which are by modifying the offset property of the notes, and by modifying the value of existing voice parameter points. Further works can expand on this research by adding features such as random starting points for the insertion, or by experimenting with other fields for data insertion.

VII. Appendix

The source code for this paper can be found in the following GitHub repository:
https://github.com/florars/makalah-kriptografi

VIII. Acknowledgment

Author thanks Mr. Rinaldi Munir for teaching the Cryptography course, which this paper was made as coursework for. Additionally, author thanks her friends for inspiration on this paper, and Flavor Foley for creating and distributing one of the files used for testing.

References

[1] Munir Rinaldi. "Steganografi (Bagian 1)". informatika.stei.itb, Rinaldi Munir, 2025. https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2025-2026/08-Steganografi-Bagian1-2025.pdf. Accessed 22 December 2025.

[2] Amsaveni, A., & Vanathi, P. T. (2015). A comprehensive study on image steganography and steganalysis techniques. International Journal of Information and Communication Technology, 7(4/5), 406. https://doi.org/10.1504/ijict.2015.070300

[3] Dreamtonics. Synthesizer V Studio 2 Pro - Fully Controllable AI Singer Plugin. https://dreamtonics.com/synthesizerv/. Accessed 22 December 2025

[4] Katzenbeisser, S., & Petitcolas, F. A. P. (2000). Information hiding techniques for Steganography and digital watermarking. Artech House.

[5] Mahajan, M., & Kaur, N. (2012). Adaptive Steganography: A survey of recent statistical aware Steganography techniques. International Journal of Computer Network and Information Security, 4(10), 76–92. https://doi.org/10.5815/ijcnis.2012.10.08

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Desember 2025

[illegible]

Maria Flora Renata S. - 13522010