Rochester Institute of Technology

Audio-Based Covert Communications Channel Documentation

Isha Mistry, Nia Poor, Christopher Brooks, Domenic Lo Iacono
Dr. Matthew Wright, Dr. Christopher Schwartz, Dr. Nate Mathews
CSEC-559/659: Generative AI in Cybersecurity 2235
March 4, 2024

Product Description

Our product is an audio-based covert channel that embeds hidden messages within audio files. It takes unmodified WAV files and alters them in a manner imperceptible to the human ear. By mapping alphanumeric characters at a very high frequency, it creates visual messages. To decode these messages, the user simply needs to visually plot the audio file and read the embedded message.

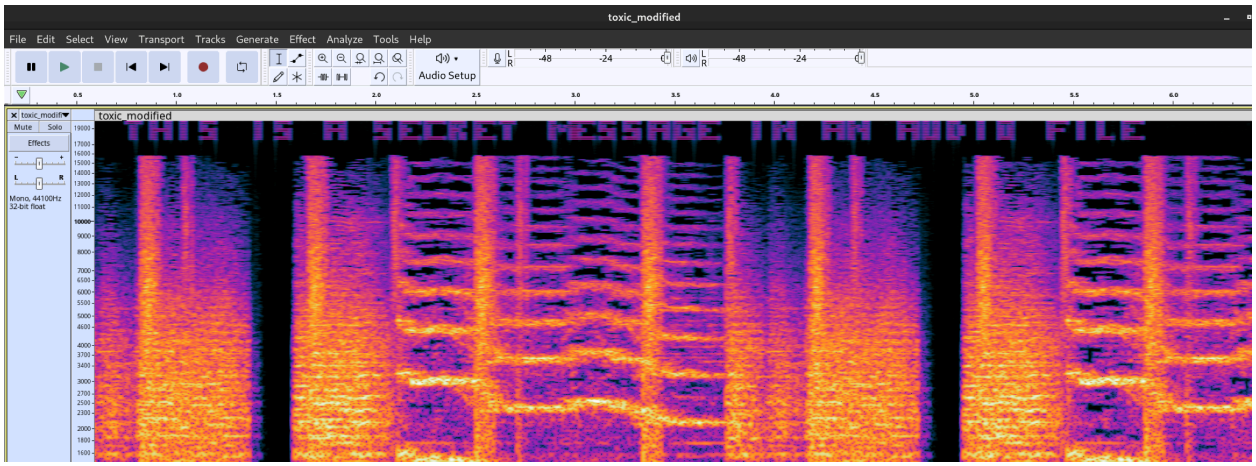


Image: Spectrogram showcasing the frequency vs. time for Toxic by Britney Spears with the message "THIS IS A SECRET MESSAGE IN AN AUDIO FILE" revealed at the top.

Encoding

To embed a message within an audio file, use the embed command followed by the necessary arguments:

python audio_covert.py embed <wav_file> <outfile> --message "Your secret message"
<wav_file>: Path to the input WAV file.
<outfile>: Path where the output WAV file with the embedded message will be saved.
--message: The secret message you wish to embed within the audio file.

Decoding

To plot the audio spectrum of a file, potentially revealing any hidden messages, use the plot command:

python audio_covert.py plot <wav_file> [--xlim X] [--ylim Y]
<wav_file>: Path to the WAV file whose spectrum you wish to plot.
--xlim: (Optional) The amount of time in seconds to display in the plot.
--ylim: (Optional) The lower bound of the frequency range to include in the plot.

Test Case Groups

We used three distinct categories of test cases to demonstrate the various strengths of our audio-based covert communications channel.

Case 1 Group: Popular Songs

Our first group of test cases demonstrates the covertness of our channel. Specifically, it shows that an individual unaware of the existence of a covert message is unlikely to notice that we have subtly modified popular songs to embed these messages.

Case 2 Group: AI Generated Music

Our second group of test cases demonstrates further automation possibilities with this script. To fully automate obtaining, encoding, and decoding WAV files, one could connect our code to an API to a website that produces AI-generated music. For our AI-generated music, we used a free account with the website creators.aiva.ai.

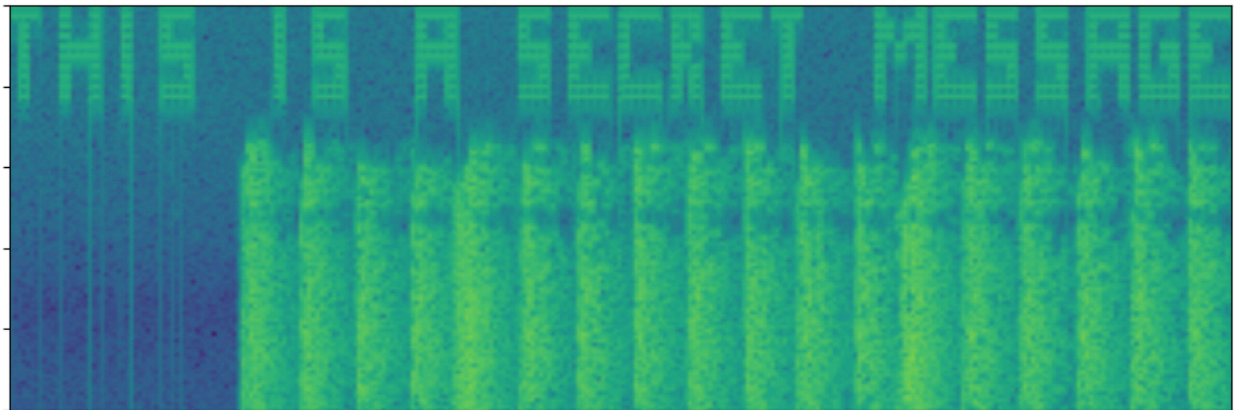
Case 3 Group: Group Conversations

Our third group of test cases illustrates our ability to overlay covert messages over seemingly innocent group conversations. A practical application of this could involve sending audio messages embedded within text communications—a common functionality that can be exploited by our covert channel.

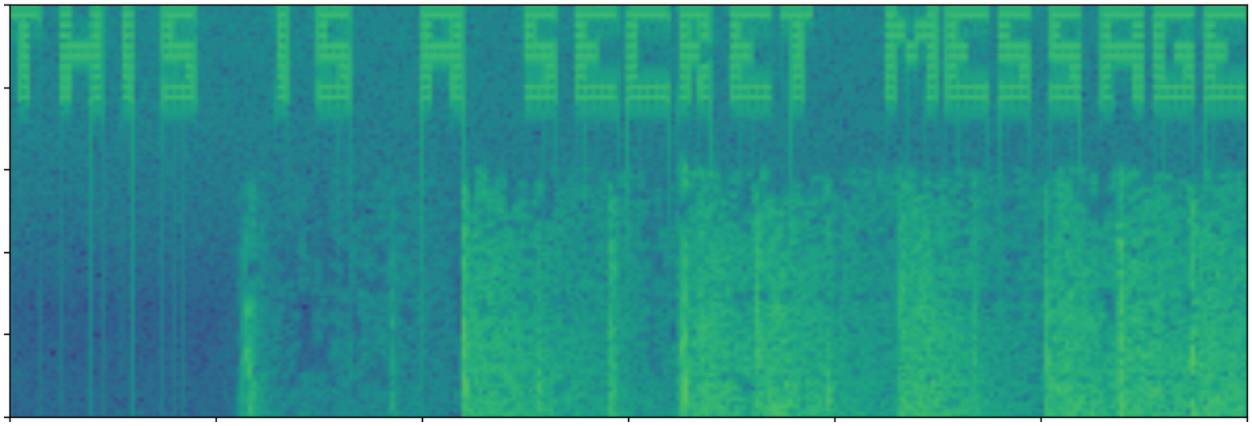
Test Case Hidden Messages

```
PS C:\Users\loiac\AudioCovertChannel> python .\audio_covert_channel.py embed .\audio_clips\ai_generated_music\AI_Generated_1.wav .\encoded_audio_clips\AI_num_1.wav --message "This is a secret message"

Encoding message: This is a secret message
Progress: Encoding 'E'
Done
PS C:\Users\loiac\AudioCovertChannel> python .\audio_covert_channel.py embed .\audio_clips\ai_generated_music\AI_Generated_2.wav .\encoded_audio_clips\AI_num_2.wav --message "This is a secret message"

Encoding message: This is a secret message
Progress: Encoding 'E'
Done
PS C:\Users\loiac\AudioCovertChannel> python .\audio_covert_channel.py embed .\audio_clips\ai_generated_music\AI_Generated_3.wav .\encoded_audio_clips\AI_num_3.wav --message "This is a secret message"

Encoding message: This is a secret message
Progress: Encoding 'E'
Done
PS C:\Users\loiac\AudioCovertChannel> python .\audio_covert_channel.py plot .\encoded_audio_clips\AI_num_1.wav --xlim 3 --ylim 10000
PS C:\Users\loiac\AudioCovertChannel> python .\audio_covert_channel.py plot .\encoded_audio_clips\AI_num_2.wav --xlim 3 --ylim 10000
PS C:\Users\loiac\AudioCovertChannel> python .\audio_covert_channel.py plot .\encoded_audio_clips\AI_num_3.wav --xlim 3 --ylim 10000
PS C:\Users\loiac\AudioCovertChannel>
```

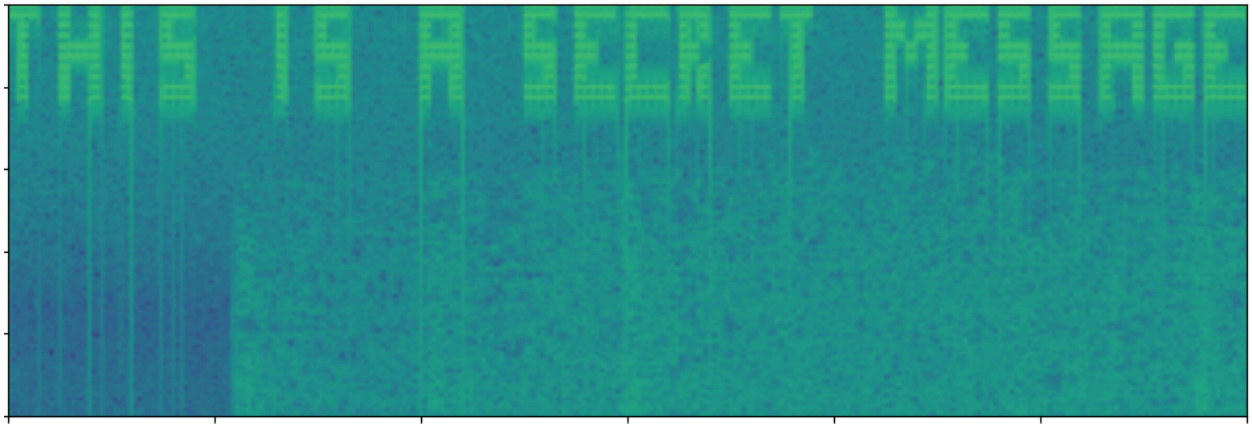
AI_Generated_1.wav :



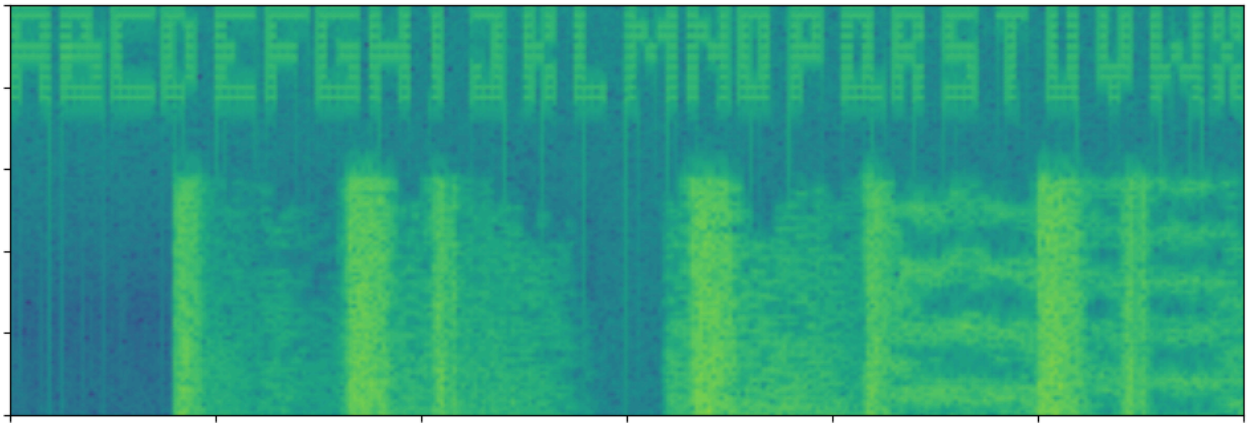
AI_Generated_2.wav :



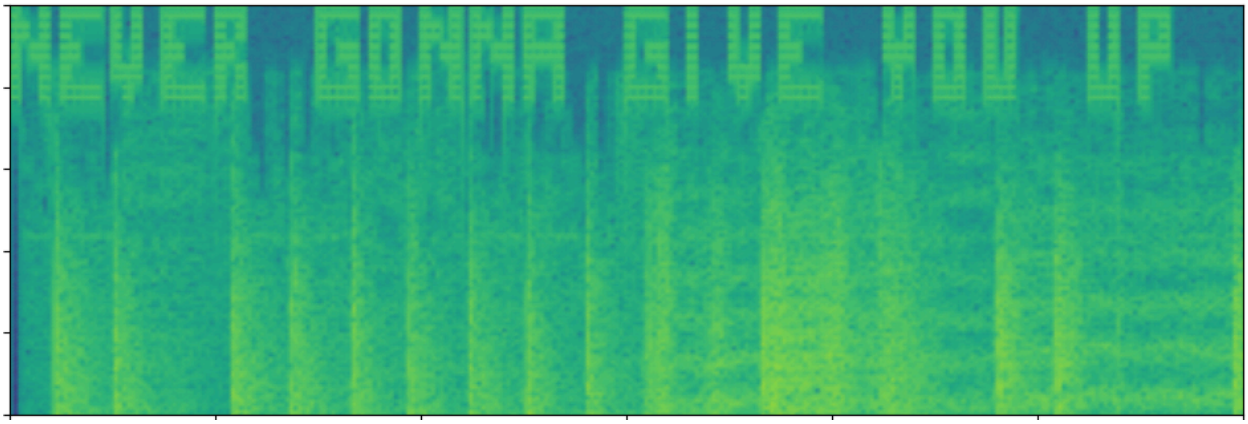
AI_Generated_3.wav :



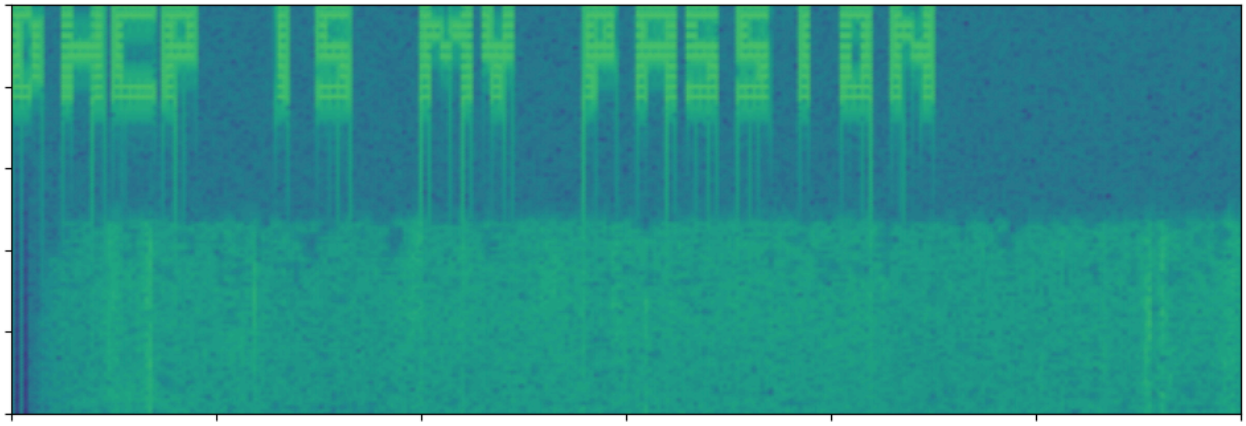
Toxic A-Z.wav :



never_gonna_what.wav :



dhcp_for_life.wav:

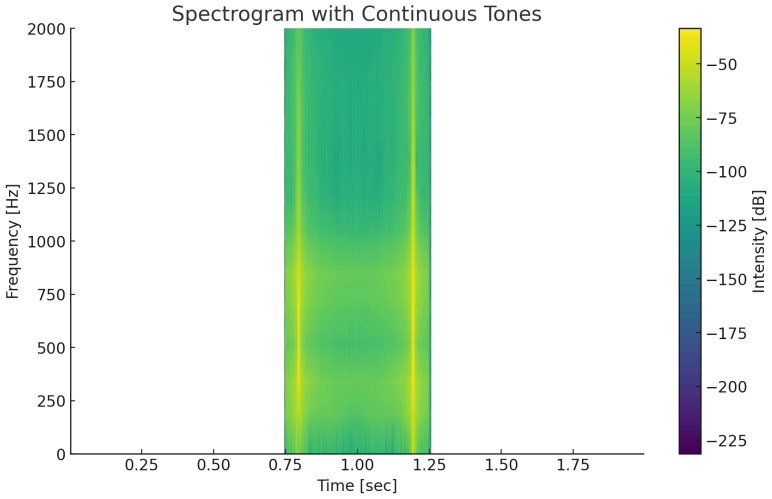


Generative AI Use

We used Generative AI for much of the math in this script. The code that creates the individual squares, that are the building blocks of the letters, was done entirely by AI. The math to create these squares is in-depth and would have taken a significant amount of time. The prompt and result of our first successful “square” are as follows:

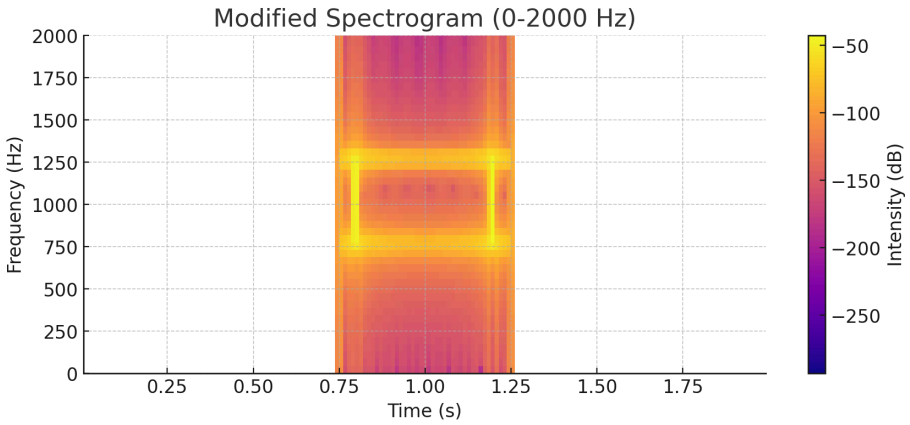
Create an audio file that, when mapped to a spectrogram, has the following qualities:

- A vertical line spanning from (0.75 seconds, 750 Hz) to (0.75 seconds, 1250 Hz)
- A vertical line spanning from (1.25 seconds, 750 Hz) to (1.25 seconds, 1250 Hz)
- Vertical lines of the same height (750 Hz to 1250 Hz) every 0.05 seconds between the intervals 0.75 seconds and 1.25 seconds
- The frequency axis ranges from 0 to 2000

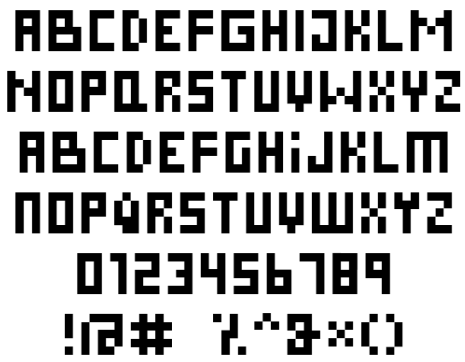


While they may not be very intense (in dB), ChatGPT *did* create horizontal lines for this square.

From there we were able to continually prompt ChatGPT to build off of and improve this square. We notably wanted the intensity to be more condensed to the necessary ranges.



When we were initially creating the messages in the audio file, the user was able to hear them. We used generative AI to lower the volume (intensity in dB) of our message's WAV file through a function written by ChatGPT. We were able to get this script after our first prompt, which makes sense considering there are likely more online sources on this topic.



We found out that asking ChatGPT to make letters out of filled out squares was significantly easier than getting it to output letters using lines and curves. We found a basic “block font” image from Google and then changed our message letters to look the same.