A Method to Compactly Store Scrambled Data Alongside Standard Unscrambled Disc Images of CD-ROMs

Jacob Hauenstein

Computer Science Department, The University of Alabama in Huntsville, Huntsville, Alabama, USA

Abstract

When archiving and preserving CD-ROM discs, data sectors are often read in a so-called "scrambled mode" before being unscrambled and further processed into a standard disc image. Processing of scrambled data into a standard disc image is potentially lossy, but standard disc images exhibit greater software compatibility and usability compared to scrambled data. Consequently, for preservation purposes, it is often advantageous to store both the scrambled data and the corresponding standard disc image, resulting in high storage demands. Here, a method that enables compact storage of scrambled data alongside the corresponding (unscrambled) standard CD-ROM disc image is introduced. The method produces a compact representation of the scrambled data that is derived from the standard disc image. The method allows for storage of the standard unscrambled disc image in unmodified form, easy reconstruction of the scrambled data, and a substantial space savings compared standard data compression techniques.

Keywords

compact disc, compression, data archival, data preservation, scrambled

1. Introduction

In recent years, there has been increased interest in archiving and preserving software and other data produced during the previous years of computing, with especially strong interest in the archival and preservation of video games data [1–4]. Much of the work to archive and preserve such data has historically been accomplished through community efforts [2], in which a community of users work to first extract data from aging storage media (a process called *dumping* or *imaging*) and then preserve the extracted data by storing copies of it on modern storage media. The resulting data is often called a *dump*, *image*, or *disc image* [5]. Because the goal of these archival and preservation projects is to preserve the dumps over a long period of time, such dumps are typically stored on multiple media and/or in multiple locations as required for long term data storage. As such, the data storage requirements for preservation communities may grow very large, especially when dumping large media such as compact disc read-only memory discs (often denoted CD-ROMs or simply CDs), the preservation of which is the focus of this work.

Dumps are typically stored in a standard format, with the specific standard used decided upon by the community. Usage of a standard format guarantees that all community member's dumps are in the same format, ensuring high software compatibility for each dump and enabling easy comparison between dumps from different community members via standard file hashing algorithms. In some cases, the process of dumping a medium produces two sets of data: the final dump in the standard format, and the intermediate data that is processed into the final dump. Unlike the final dump, the intermediate data is often in a format that has relatively narrow software compatibility and may be difficult to compare between community members. However, both the final dump and the intermediate data have potential importance in preservation. While the final dump is important

David C. Wyld et al. (Eds): SIPP, NLPCL, BIGML, SOEN, AISC, NCWMC, CCSIT - 2022
pp. 207-215, 2022. CS & IT - CSCP 2022 DOI: 10.5121/csit.2022.121318

because it allows easy comparison of dumps between community members and has wide software compatibility (e.g., with emulators and disc image processing software that enables exploration and study of the data), the intermediate data is important because it may contain data that, due to limitations of the standard used for final dumps, is not included in the final dump. E.g., in Section 2.3, we describe in detail how data may be lost when CD-ROM dumps are processed from the often-used intermediate *scrambled* data into the standard *unscrambled* disc image used for final dumps. Thus, for the case of CD-ROM dumps, there is a need for community members to store both intermediate data and final dumps, imposing even greater storage requirements on top of the already demanding storage requirements of CD-ROM archival and preservation.

The primary contributions of this work are (1) a novel method for compactly storing the intermediate scrambled data alongside the final dump when archiving and preserving CD-ROM discs, and (2) a study of the space savings afforded by our method compared to naively storing the intermediate scrambled data. Our method can help ease the storage requirements of the CD-ROM preservation community. Our method works by attempting to reconstruct the intermediate scrambled data from the unscrambled final dump and then creating a binary diff between the reconstructed intermediate data and the original intermediate data produced during the dump.

The remainder of this work is organized as follows. Section 2 provides details about data storage on CD-ROM and the file format used for CD-ROM disc images are presented, including details about scrambling and why some data may not be preserved when the final dump is built from the scrambled intermediate data. Section 2 also presents some details about how scrambled data is dumped from CD-ROM discs, and why it is valuable to do so. Section 3 describes our method for compactly preserving the scrambled data alongside the unscrambled final dump. Section 4 describes the experiments performed to analyze the space savings of our method and the results of those experiments. Section 5 concludes the work.

2. Background

This section presents some background details about how data is stored on CD-ROMs, why and how such data is scrambled, why there is value in dumping / preserving the scrambled data, and why it may be the case that there is data present in the (intermediate) scrambled data that is removed when a dump is converted from its scrambled form into a standard (unscrambled) image file (i.e., the final dump).

2.1. Data storage on CDs / disc images

In this section, we present some necessary background details about how information is stored on CDs and in CD disc images / dumps. Note that, because the CD specifications are quite lengthy and complex (e.g., as partially seen in [6]), we present here only enough details to aid understanding of this work. Additionally, our focus here is on the way that CDs are presented at the software level when such discs are read by standard, widely available computer optical disc drives (such as those used for dumping CDs). CDs also contain a large amount of other data (e.g., [7], [8]) at the physical layer that is not exposed / accessible at the software level by such drives and is thus outside the scope of this work and not discussed here. Since our focus here is on archival and preservation, we also assume that discs will be read in a mode that returns the most data possible from the disc. There exist reading modes that discard some error detection and correction data when reading from discs [9], but we assume those reading modes are not being used here.

Compact discs are divided into *sectors*, and each sector contains 2352 bytes. (N.B., there are some additional bytes present in the so-called *subchannels*, but we do not make use of these subchannels in this work.) When the contents of a CD are dumped and stored in a standard disc image, the disc image simply contains the 2352 bytes of every sector contained on the CD (starting with the first

sector). Thus, the logical format of CD sectors presented in this section also applies to CD disc images. (N.B., CD disc images often contain, in addition to the file that holds the sector data, other files that are used to store metadata, but we do not make use of these other files / data in this work.)

Originally, compact discs were designed for storage of stereo audio at a 16-bit sampling resolution and a 44.1Khz sampling rate, and thus 75 sectors represents 1 second of audio [9]. When storing data within a sector, the sector is divided into a number of fields. The first 12 bytes (bytes 0 through 11) of the sector are used to store the *sync field* value of 00 FF FF FF FF FF FF FF FF FF FF 00 hexadecimal. The remaining fields store the sector address and mode (collectively called the *header field*), user data, error correction (ECC) and detection (EDC) data, and other items.

The presence of a regular bit pattern (i.e., many more bits with a value of zero than one, or vice versa) on the physical disc is problematic for the CD decoding hardware within the optical disc drive [6]. Because such sequences may naturally occur in data, before each data sector is stored on the disc surface, the data sector is subjected to a process known as *scrambling*. In the scrambling process, each byte within the sector, except the 12 bytes that comprise the sync field, is XORed with the corresponding byte in a standardized scrambling table. The byte values contained in the scrambling table are designed to, when XORed with the sector data, avoid problematic bit patterns. The algorithm used to generate the scrambling table is standardized and described in various standards documents (e.g., [6]). This scrambling process is reversible by simply performing the same XOR a second time, and thus data is easily scrambled before the sector is written to the disc (to avoid the problematic bit patterns) and unscrambled when the sector is read from the disc (to return the data to its original state).

2.2. Reading scrambled data

When a sector is read from a CD using a standard optical drive, a sequence of 2352 bytes is returned by the drive. If the optical drive is instructed to read the sector in data mode, the optical drive (typically) automatically (1) unscrambles the data and (2) performs error detection and correction using any EDC / ECC bytes present within the data sector. Finally, the drive returns a sequence of 2352 bytes representing the unscrambled, error-corrected sector starting at the 12 byte sync field. In contrast, if the drive is instructed to read the sector in audio mode, the drive does not attempt to perform unscrambling or use any EDC / ECC bytes within the sector prior to returning a sequence of 2352 bytes representing the audio sector. In the case of reading in audio mode, the sequence of bytes returned by the optical drive typically does not begin exactly at the start of the sector. Instead, the data returned by the drive is offset by some number of bytes from the true start of the sector, with the offset amount depending on the specific optical drive model used. This audio offset has been studied widely within the optical disc archival community (e.g., [10], [11]). In addition to the audio offset exhibited by the specific optical disc drive used, some otherwise identical discs exhibit different audio offsets due to variations in manufacturing (often called the *factory offset* or *write offset*) [10]. These offsets complicate the process of archiving discs and comparing dumps between different community members / different copies of a disc, especially for discs containing both data and audio sectors (where it may be necessary to manually correct the difference in offsets between the two types of sectors [12]).

In general, optical disc drives refuse to read audio sectors in data mode (or vice versa), as this is the behavior required by the standard optical drive reading commands [9]. However, some optical drives are able to read both data and audio sectors in audio mode [13], bypassing the drive's data sector processing logic. This ability to read data sectors in audio mode (sometimes called *scrambled mode* [14]) is useful for multiple reasons. First, it ensures that the drive returns both audio and data sectors using the same offset, obviating the need for users to manually correct the offset difference between audio and data sectors. Second, it bypasses the optical drive's data unscrambling

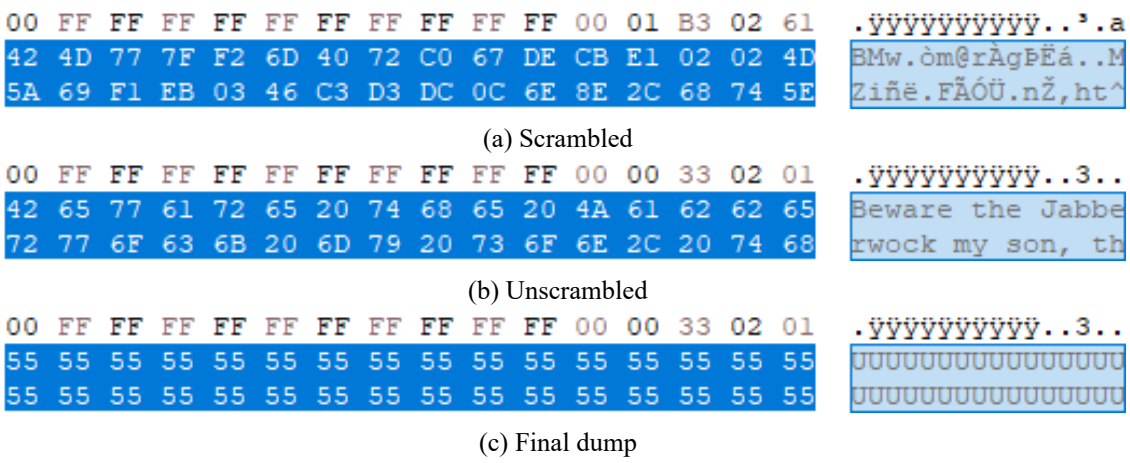

Figure 1: Snippet of a sector from the game Rune [16] as seen in a hex editor (left shows raw byte values, right shows text given by those bytes). The highlighted portion of the scrambled intermediate data of the sector (shown in (a)) contains a string of text that can be seen clearly when the sector is unscrambled (shown in (b)). Because the sector contains intentional EDC/ECC errors, the string of text is removed when the sector is converted into the final dump (shown in (c)).

and error correction logic. This is useful because some discs contain data sectors with intentional EDC/ECC errors (often called *error sectors*) as a form of copy protection [15], and bypassing the optical drive's error correction logic allows these error sectors to be processed in software with minimal modification by the drive's error correction logic. The community-developed DiscImageCreator software [14] uses scrambled mode for CD dumping and is capable of automatically correcting for offsets and dumping CDs containing a wide variety of copy protection schemes.

2.3. Converting from scrambled data to the final dump

While reading data in scrambled mode is useful for CD archival, the scrambled mode data has relatively limited software compatibility and, because the optical disc drive does not use any error correction to verify / correct errors when data sectors are read in scrambled mode, the scrambled data may contain undetected errors. Consequently, the community-dictated standards typically used for preserving and comparing CD disc images require that the data sectors be further processed and stored in unscrambled form for the final dump. Thus, the scrambled mode data is an intermediate format. Building the final dump requires that the scrambled data be unscrambled. In addition, any EDC/ECC data is verified during build of the final dump. According to the community standards, for any sectors containing EDC/ECC errors (intentional or otherwise), all bytes except the sync field and header field are replaced with the hex sequence 0x55 [17]. This dummy sector standard has a number of benefits for the community, including (1) it matches the behavior of software previously used for archival of optical discs [12], ensuring that dumps made with newer software match those dumps made with older software, (2) it makes it possible to easily match dumps between users even in the case of discs with intentional errors by making the byte values in error sectors consistent between dumps, and (3) it was previously assumed that error sectors do not contain any useful data [18], and it was thus believed to be the case that there is no harm in replacing the data in such sectors.

The assumption that error sectors do not contain any useful data has been found to be incorrect for some discs [18]. For example, some copies of the PC video game Rune [16] have hidden text data stored inside of at least one error sector [18]. This hidden text string is present (in scrambled

form) in the intermediate data, but it is destroyed when the intermediate data is processed into the final dump, as shown in Fig. 1. Because as much data as possible should be preserved for archival purposes, it is thus necessary for community members to preserve both the intermediate data and the final dump, and, because the intermediate data and the final dump are each equal to the total size of the disc being dumped (i.e., they both contain all the sectors on the disc), this requirement essentially doubles the data storage requirements for each CD-ROM disc dumped (compared to storing only the final dump).

3. METHOD

In this section, we introduce our method to compactly store the intermediate scrambled data alongside the final dump when preserving CD-ROM discs. To ensure that the convenience of the final dump is not lessened when our method is applied, our method leaves the final dump unmodified and converts the intermediate scrambled data to a more compact form. This compact form can easily be used to fully reconstruct the original intermediate data.

Our method takes advantage of the fact that, for data sectors in which no EDC/ECC errors are present, the unscrambling process is exactly reversible, and such sectors can be rescrambled from the final dump into a byte sequence identical to the corresponding sector in the intermediate data. In contrast, for sectors in which EDC/ECC errors are present, the sectors are replaced with dummy sectors in the final dump (as noted in Section 2.3), and, consequently, the intermediate data for these sectors cannot be reconstructed by rescrambling the final dump. Thus, to preserve the intermediate data alongside the final dump, our method's compact representation of the intermediate scrambled data stores the intermediate data only for those sectors that cannot be exactly reconstructed from the final dump (i.e., sectors with EDC/ECC errors). Because it is typically the case that the vast majority of data sectors on a CD-ROM do not have any EDC/ECC errors, our method assumes that most data sectors can be exactly reconstructed into their intermediate format. (Our method also works for discs with a large number of EDC/ECC errors, though the space savings will be reduced.)

In the following two subsections, we describe how our method creates the compact representation of the intermediate data and how our method recreates the intermediate data from this compact representation, respectively.

3.1. Creating the compact representation

To create a compact representation of the intermediate data from the final dump, our method works in two phases. The first phase creates an approximate reconstruction of the intermediate scrambled data from the final dump. For convenience, we use ϵ to denote the file containing the original intermediate data produced during the dump and $\hat{\epsilon}$ to denote the file containing the approximately reconstructed intermediate data. For this phase, the input to our method is the disc image file containing the final dump, denoted ω, and the output is $\hat{\epsilon}$. This first phase works as follows. For each sector in ω, the sector is first checked to see if the first 12 bytes of the sector contain the sync field value. If the sync field value is not present, the sector is assumed to be an audio sector, and the sector is copied unmodified into $\hat{\epsilon}$. If the sync field value is present, each byte in the sector (excluding the 12 bytes in the sync field) is XORed with the corresponding byte in a table of the 2340 scrambling values (denoted **T**) and then written into $\hat{\epsilon}$. (Note that, because the first 12 bytes of the sector are not scrambled, the 13th byte of the sector is the first byte that is scrambled, and it is scrambled by XORing with the 1st byte of **T**.) This scrambling table is generated from the algorithm given in [6]. This process is performed for each sector present in ω. Upon conclusion of the first phase, $\hat{\epsilon}$ contains an approximate reconstruction on the intermediate data.

The second phase uses the *xdelta3* binary diff software [19] to encode the differences between ϵ and $\hat{\epsilon}$ into a new diff file, denoted Δ. Because, in phase 1, most sectors are exactly reconstructed from

Algorithm 1: Creating Δ from ω via $\hat{\epsilon}$

```
Data: T
Input: ϵ, ω
Output: ε̂, Δ
foreach 2352 byte sector s in ω do
    if first 12 bytes of s equal sync field value
     then
        // data sector, XOR with T
        for i ← 12 to 2351 do
            // XOR byte i of s with byte i − 12 of T
            s[i] = s[i] ⊕ T[i − 12]
        end
        // copy scrambled s into ε̂
        copy s into ε̂
    else
        // audio sector, just copy into ε̂
        copy s into ε̂
    end
end
// Now that ε̂ is constructed, use xdelta3 to diff ε̂ and ϵ, giving Δ
Δ ← output of "xdelta3 -e -9 -s ε̂ ϵ"
```

the final dump into their intermediate form, ϵ and $\hat{\epsilon}$ typically differ in relatively few byte positions (as few as 0 byte positions may differ), and, as a result, Δ is typically substantially smaller than ϵ. And, because the output of *xdelta3* is Δ, a binary diff file that can be used to reconstruct ϵ from $\hat{\epsilon}$, and, because $\hat{\epsilon}$ can be reconstructed from ω, just the binary diff file Δ is sufficient to reconstruct ϵ from ω. Thus, ϵ can be discarded and the smaller Δ kept instead. Note that this approach to encoding ϵ from $\hat{\epsilon}$ is robust, because, even if some of the sectors were processed incorrectly when $\hat{\epsilon}$ was created in phase 1, Δ still contains the necessary information to rebuild ϵ from $\hat{\epsilon}$. That is, as long as phase 1 results in a $\hat{\epsilon}$ that *approximately* reconstructs ϵ at most byte positions, Δ will be smaller than ϵ. (We study the amount of space savings achieved by our method in Section 4.)

The pseudocode for both these phases is shown in Fig. 1.

3.2. Recreating the intermediate data from the compact representation

To reconstruct the intermediate data from the final dump, our method again works in two phases. The first constructs $\hat{\epsilon}$ from ω, and this phase is identical to the first phase described in the previous section. The second phase uses *xdelta3* to reconstruct ϵ using Δ.

The pseudocode for both these phases is shown in Fig. 2.

4. Experiments and Results

In this section, we describe our experiments to evaluate the space savings of our method (compared to naively storing the intermediate data, with and without compression, alongside the final dump) and the results of those experiments.

4.1. Experiments

To study the space savings of our method, we first selected and dumped four CD-ROM discs using DiscImageCreator. As previously mentioned, DiscImageCreator dumps using scrambled mode,

Algorithm 2: Recreating ϵ from ω via $\hat{\epsilon}$ using Δ

```
Data: T
Input: Δ, ω
Output: ε̂, ε
foreach 2352 byte sector s in ω do
    if first 12 bytes of s equal sync field value
     then
        // data sector, XOR with T
        for i ← 12 to 2351 do
            // XOR byte i of s with byte i − 12 of T
            s[i] = s[i] ⊕ T[i − 12]
        end
        // copy scrambled s into ε̂
        copy s into ε̂
    else
        // audio sector, just copy into ε̂
        copy s into ε̂
    end
end
// Now that ε̂ is constructed, use xdelta3 to apply Δ to ε̂, giving ε
ε ← output of "xdelta3 -d -s ε̂ Δ"
```

producing both scrambled intermediate data and a final unscrambled dump. We then, for each of the four discs, used our method to generate the compact representation (i.e., Δ) of the intermediate scrambled data. Finally, we compared the size of the compact representation generated by our method with the size of the original scrambled data from the corresponding final dump. In addition, we compared the size of our compact representation with the size of the intermediate data when it is compressed using 7-Zip's "Ultra" mode [20]. To compare sizes, the *space saving*, denoted k, was calculated from the new size (i.e., the size of the 7-Zip compressed file or Δ) and the original size (i.e., the size of the original scrambled data) according to

$$k = 1 - \frac{\text{New Size}}{\text{Original Size}}. \tag{1}$$

Thus, k is equal to 0 if the new size and original size are equal (i.e., when there is no space savings) and increases as the amount of space savings goes up.

The four discs dumped, denoted **D1**, **D2**, **D3**, and **D4**, were selected such that they represent a variety of possible disc types that may be input to our method. **D1** contains data sectors only and does not contain any intentional error sectors. **D2** contains both data sectors and audio sectors and does not contain any intentional error sectors. **D3** and **D4** both contain data sectors only, and both contain intentional error sectors.

4.2. Results

The results are summarized in Table 1. There, the original size and number of error sectors are shown for each disc. In addition, the size of the 7-Zip compressed scrambled data is shown and the size of Δ when the scrambled data is compacted according to our method is shown. Finally, the space savings value k is shown for both 7-Zip and our method.

As can be seen in the table, our method achieves a much higher space savings value compared to 7-Zip. As seen here, because the scrambling process helps to ensure that data has similar numbers

Disc	Original Size	No. Err. Sec.	7-Zip Size	Our Method Size	7-Zip k	Our Method k
D1	493,146,192	0	459,200,084	1,718	0.069	0.999
D2	752,425,968	0	580,962,585	2,612	0.228	0.999
D3	788,886,672	585	661,481,962	1,261,927	0.161	0.998
D4	830,822,832	583	828,265,184	1,297,480	0.003	0.998

Table 1: Results of our method and 7-Zip Ultra compression on the four discs. All sizes in bytes.

of bits with values of 1 and 0, the scrambled intermediate data has a high level of entropy and typically does not compress well via standard compression algorithms. In contrast, our method exhibits a very high space savings.

5. Conclusion

In this work, we introduced a new method for compactly storing intermediate scrambled data alongside final dumps. Our method takes advantage of the fact that the intermediate scrambled data can be approximately reconstructed from the final dump. Our method first builds an approximate reconstruction of the scrambled intermediate data from the final dump, and then encodes the differences between the approximate reconstruction and the actual intermediate data.

Our method achieved a substantial space savings increase compared to storing the intermediate data without compression and compared to storing the intermediate data using 7-Zip's Ultra compression. Thus, we believe our method will prove useful for easing the data storage burden encountered by those archiving and preserving CD-ROMs.

Acknowledgments

We wish to thank the members of the data archival and preservation communities. We also wish to thank the reviewers for their helpful feedback.

References

[1] M. Guttenbrunner, C. Becker, and A. Rauber, "Keeping the game alive: Evaluating strategies for the preservation of console video games," *International Journal of Digital Curation*, vol. 5, no. 1, Jun. 2010. [Online]. Available: https://doi.org/10.2218/ijdc.v5i1.144

[2] F. Cifaldi, ""It's just emulation!" - The challenge of selling old games," in *Game Developers Conference*, 2016. [Online]. Available: https://www.gdcvault.com/play/1023470/contactUs

[3] J. Newman, "The music of microswitches: Preserving videogame sound—a proposal," *The Computer Games Journal*, vol. 7, no. 4, pp. 261–278, 2018. [Online]. Available: https://doi.org/10.1007/s40869-018-0065-8

[4] N. Nylund, P. Prax, and O. Sotamaa, "Rethinking game heritage–towards reflexivity in game preservation," *International Journal of Heritage Studies*, vol. 27, no. 3, pp. 268–280, 2021. [Online]. Available: https://doi.org/10.1080/13527258.2020.1752772

[5] Redump.org Community, "Redump.org," 2022, accessed Jun. 15, 2022. [Online]. Available: http://wiki.redump.org/index.php?title=Redump.org&oldid=48927

[6] "Data interchange on read-only 120 mm optical data disks (CD-ROM)," Ecma International, Geneva, Switzerland, Standard, Jun. 1996. [Online]. Available: https://www.ecma-international.org/publications-and-standards/standards/ecma-130/

[7] L. B. Vries and K. Odaka, "CIRC-the error-correcting code for the compact disc digital audio

system," in *Audio Engineering Society Conference: 1st International Conference: Digital Audio*. Audio Engineering Society, 1982.

[8] K. Immink, "Modulation systems for digital audio discs with optical readout," in *ICASSP '81. IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 6, 1981, pp. 587–589.

[9] "Multimedia command set - 5 (MMC-5)," T10, Washington, D.C., USA, Standard, Oct. 2006. [Online]. Available: http://www.t10.org/cgi-bin/ac.pl?t=f&f=mmc5r04.pdf

[10] Redump.org Community, "Combined offset in eac," 2010, accessed Jun. 15, 2022. [Online]. Available: http://forum.redump.org/topic/7649/combined-offset-in-eac/

[11] Accuraterip.com, "Accuraterip," 2010, accessed Jun. 15, 2022. [Online]. Available: http://www.accuraterip.com/

[12] Redump.org Community, "CD dumping guide with audio tracks (old)," 2022, accessed Jun. 15, 2022. [Online]. Available: http://wiki.redump.org/index.php?title=CD_Dumping_Guide_with_Audio_Tracks_(Old)&oldid=46420

[13] ——, "DiscImageCreator: Optical disc drive compatibility," 2022, accessed Jun. 15, 2022. [Online]. Available: http://wiki.redump.org/index.php?title=DiscImageCreator:_Optical_Disc_Drive_Compatibility&oldid=48878

[14] sarami, "DiscImageCreator," https://github.com/saramibreak/DiscImageCreator, 2022, accessed Jun. 15, 2022.

[15] K. Kaspersky, *CD Cracking Uncovered: Protection Against Unsanctioned CD Copying*. Wayne, PA, USA: A-List Publishing, 2004.

[16] Human Head Studios, "Rune," 2000, accessed Jun. 15, 2022. [Online]. Available: https://web.archive.org/web/20130622083143/http://www.rune-world.com/

[17] Redump.org Community, "Moderating guidelines for IBM PC and other systems," 2021, accessed Jun. 15, 2022. [Online]. Available: http://wiki.redump.org/index.php?title=Moderating_guidelines_for_IBM_PC_and_other_systems&oldid=45839

[18] ——, "Issues dumping pc disc with "code lock" copy protection," 2021, accessed Jun. 15, 2022. [Online]. Available: http://forum.redump.org/topic/29842/issues-dumping-pc-disc-with-code-lock-copy-protection/page/2/

[19] J. P. MacDonald, "xdelta: open-source binary diff, differential compression tools, VCDIFF (RFC 3284) delta compression," 2016, accessed Jun. 15, 2022. [Online]. Available: http://xdelta.org/

[20] I. Pavlov, "7-Zip," 2021, accessed Jun. 15, 2022. [Online]. Available: https://www.7-zip.org/