MDPI

Article

Hide45: A Method for Optimal Payload Data Hiding in Base45 Encoded Strings

Marco Botta , **Davide Cavagnino** and **Alessandro Druetto** *

Dipartimento di Informatica, Università degli Studi di Torino, Corso Svizzera 185, 10149 Torino, Italy; marco.botta@unito.it (M.B.); davide.cavagnino@unito.it (D.C.)
* Correspondence: alessandro.druetto@unito.it

Abstract: Base45 encodes pairs of octets using 3 characters from an alphabet of 45 printable symbols. Previous works showed the ability to hide payload data into encoded strings by exploiting the unused configurations of the Base45 encoding. In this paper, we present Hide45, an algorithm for hiding data into Base45 encoded strings that optimizes the embedding capacity, according to the frequency distribution of the input data. Experimental tests show that an optimal assignment of bit configurations to the most frequent pairs of octets allows to reach a payload capacity very close to the theoretical capacity of the method, improving over a baseline assignment by up to 53%.

Keywords: Base45 printable encoding; data hiding; digital watermarking; optimal embedding

Citation: Botta, M.; Cavagnino, D.; Druetto, A. Hide45: A Method for Optimal Payload Data Hiding in Base45 Encoded Strings. *Appl. Sci.* **2023**, *13*, 9993. https://doi.org/10.3390/app13179993

Academic Editor: Andrea Prati

Received: 4 August 2023
Revised: 28 August 2023
Accepted: 1 September 2023
Published: 4 September 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Digital watermarking and steganography are part of the field of data hiding where some piece of information is stored in digital objects for various purposes ranging from simple information saving to copyright protection, from integrity protection or origin authentication to covert communication [1]. The digital objects involved in data hiding may be of any type, like images [2–4], audio files [5,6], movies [7–9], 3D models [10–12], neural networks [13–16], and textual data [17–19].

In the field of data hiding in textual data, Ref. [20] proposes a method to embed information into binary strings encoded in sequences built using a printable alphabet: there, the authors focus on two encodings, namely Base45 [21] and Base85 [22], that have spare configurations that can be used to save one bit of information. An improvement for the Base45 encoding is presented in [23] where bits are reversibly embedded by exploiting the most probable printable sequences depending on the type of binary data of the cover object.

In [24], several applications for the data hiding potentiality of those unused sequences are suggested: for example, digital signatures, MACs, CRCs, and other security information.

In this work, the frequency distribution of the input binary sequences for a file type is computed on a large data sample and used to optimize the number of bits embedded in every printable sequence, thus improving the performance of the method described in [23]. Note that the proposed approach can be applied to any encoding method pertaining to the framework defined in [20], the only limit being the size of the unused sequences as it will be defined. In the Base45 case here studied, for example, our approach improves the average payload obtained in [23] by up to 53%.

The remainder of the paper is organized as follows: Section 2 introduces some notation to be used throughout the paper, whilst Section 3 recalls some works related to the presented encoding improvement. Section 4 gives a brief description of the Base45 printable encoding of binary data and Section 5 presents the original data embedding algorithm and the following improvement, then proposes an optimization based on the input data frequency distribution. The performance of the method is shown in Section 6 and conclusions are drawn in the last Section 7. Finally, the Appendix A presents a possible compression

method to efficiently save the table recording the optimal assignment of bit configurations to the most frequent pairs of octets.

2. Notation

In this paper, sets are denoted with calligraphic capital letters (e.g., $\mathcal{B}$), the empty set is written with the symbol $\varnothing$, alphabets with uppercase boldface italic letters, like $\boldsymbol{A}$, scalar values and optimization variables with lowercase italic letters (e.g., n) and functions and mappings between sets with Greek lowercase letters, like μ. The set of all non-negative integer numbers is denoted with $\mathbb{Z}^+$.

A sequence (or configuration) of symbols, or characters, from an alphabet is represented with uppercase letters, eventually with a subscript, like S and S_0. Binary string or binary sequence denote a sequence of symbols from an alphabet of cardinality 2. The function $\mathbf{card}(.)$ represents the cardinality of a set.

The name BaseYY, where YY is a number, will represent a method of encoding binary strings using an alphabet composed of YY symbols.

3. Related Works

The use of printable encodings of binary data is required in some contexts like e-mail transfer or QR-codes, thus many such encodings have been developed for various purposes.

One of the first implementations of an encoding method to transparently transmit data through systems that may modify some binary sequences (i.e., are not 8-bit clean because use some binary strings as control messages) has been the pair of programs uuencode–uudecode for UNIX systems: essentially, this method reversibly transforms in printable characters, which are not modified by intermediate systems, the binary data to be transmitted using a subset of the ASCII character set.

In the present days the most widely known printable encoding is Base64 [25] that encodes three binary octets with four printable characters. The same RFC specifies a Base32 encoding that represents five binary octets with eight characters from the Base32 alphabet and the Base16 encoding which is the case-insensitive hexadecimal representation of binary data.

Nonetheless, many other encodings of bit strings into printable characters have been developed; in the following we recall some of them and their possible applications in a list ordered by the numerical base used to transform the binary data.

Base36 uses an alphabet of ten digits, from 0 to 9, and twenty-six letters of the English alphabet, from A to Z: many programming languages have library functions for conversions from and to Base36, for example, PHP [26], JavaScript [27], and Python [28].

Forty-one symbols alphabets are used in the two proposals [29,30] for using the minimum number of characters needed to represent two octets with three symbols. The two proposals differ for the alphabets used and for the possibility of [30] to encode bit strings of any length.

Base45 is the foundation of the RFC 9285 [21] that defines a method for encoding binary data with forty-five symbols, namely the twenty-six letters of the English alphabet, the ten Arabic digits and eight special characters along with the space. More details on this encoding are given in Section 4. In [20,23] the unused sequences of printable symbols for the Base45 and Base85 encodings are exploited for data embedding.

A Base56 encoding has been developed in [31] where the author has written PHP encoding/decoding functions for shortening URLs using an alphabet of fifty-six characters only.

An alphabet with fifty-eight printable characters (the ten Arabic digits and the twenty-six uppercase and lowercase letters of the English alphabet without zero, capital O, capital I, and lowercase L to avoid graphical misinterpretation by humans) is used in the Bitcoin system to write addresses and keys (see [32] and the expired Internet Draft [33]).

Using an alphabet composed by the ten decimal digits and the twenty-six letters of the English alphabet, both uppercase and lowercase, allows encoding data with sixty-two

symbols. The paper [34] proposes a transformation format for ISO 10646 based on the ten decimal Arabic digits and the twenty-six English letters, both uppercase and lowercase: using sixty-two characters this format is called UTF-62. UCS-2 codes (16 bits) are encoded with three Base62 characters and UCS-4 codes (32 bits) are transformed in six Base62 characters: UTF-62 provides for distinguishing the two cases by the different first encoded character. The work [35] uses the same alphabet of [34] (even if the positions of the characters are interchanged) and encodes the bit stream in chunks of six bits: the binary configurations from 000000 to 111011 are directly mapped to one of the sixty symbols of the alphabet, whilst if the first five bits are 11110 or 11111 then the mapping will be on the sixty-first or the sixty-second alphabet symbol respectively, and the sixth bit will be delayed to the next chunk to be encoded. In this way the encoding has the maximum efficiency and no configurations are wasted.

Alphabets with eighty-five printable ASCII characters are used in [36], for the representation of IPv6 addresses, and in Ascii85 [22], for the encoding of binary data mapping groups of four octets in five Base85 characters improving to 5/4 the inflation rate 4/3 of Base64.

Base91 [37] presents an efficient method to encode bit strings with an alphabet of ninety-one ASCII printable characters (ten digits, twenty-six English uppercase and lowercase letters, and twenty-nine special characters like ampersand, parentheses, and at sign). Thirteen bits at a time (2^{13} possible bit strings) are encoded with two Base91 characters (leading to 91^2 possible pairs) leaving eighty-nine Base91 pairs unused: thus, if the value of the thirteen bits is less than 89 one more bit is encoded for a total of fourteen bits transformed (note that the maximum possible encoded value is $2^{13} + 88$). Additionally, Ref. [38] encodes data chunks of thirteen bits with pairs of ninety-one printable characters employing twelve of the $91^2 - 2^{13} = 89$ unused configurations to indicate the number of bits to ignore in the last chunk in case the original bit string has a bit length not multiple of thirteen.

A Base122 encoding (Base-122 [39]) has been developed to reduce the inflation rate of Base64 from 4/3 to 8/7. Base-122 encodes data using Unicode UTF-8 [40] which represents characters with one to four octets. It saves blocks of seven bits in one UTF-8 octet and, to deal with six characters that will not be allowed by HTML, Base-122 encodes them in two UTF-8 octets having form 110AAA1B 10BBBBBB, where the three bits AAA define one of the six characters to be encoded and the B bits may be used to represent other bits; care is also taken for the termination of a bit string having length not multiple of seven.

Some other references on the printable encoding topic of binary data may be found in [41].

4. The Base45 Encoding

The Base45 encoding [21] is defined in an RFC that proposes a method for encoding pairs of binary octets using an alphabet $\boldsymbol{A}$ of forty-five printable symbols mainly aimed at representing binary data stored into QR-codes. The Base45 alphabet is composed of the ten decimal digits, the twenty-six letters of the English alphabet, the space, and other eight special characters (listed in the following double quoted string "$%*+-./:").

Every pair of octets is considered to represent a sixteen-bit binary number which is converted in Base45 and expressed with three symbols from the alphabet $\boldsymbol{A}$. In case of a binary string having length an odd number of octets the last octet is interpreted as an eight-bit binary number, converted in Base45 and encoded using two symbols from $\boldsymbol{A}$.

In the present context the main thing to note is that with three Base45 symbols it is possible to represent $45^3 = 91{,}125$ sequences that are much more than the $2^{16} = 65{,}536$ required to encode two octets. The following section will briefly present the encoding and data embedding proposed in [20], discuss the improvement based on pairs of octets statistics developed in [23], and introduce the novelty of the present paper with the enhancement obtained by an optimized bit assignment.

5. Proposed Algorithm

In [20] we proposed to consider a printable encoding method based on an alphabet $\boldsymbol{A}$ with a characters that uses a sequence of t symbols to represent binary strings of n bits. The total number of sequences of length t is a^t and there are 2^n binary strings of length n: if $a^t > 2^n$, like in the case of Base45 [21] or Base85 [22], the unused (sometimes called "illegal") $a^t - 2^n$ sequences can be configured to carry extra information, possibly in a reversible manner.

Consider an alphabet $\boldsymbol{A}$ of a symbols. Call $\mathcal{S}$ the set of the a^t possible sequences of length t made of symbols from $\boldsymbol{A}$. Call $\mathcal{B}$ the set of the 2^n binary strings of length n. If $\mathbf{card}(\mathcal{S}) \geq \mathbf{card}(\mathcal{B})$ then it is possible to use the sequences in $\mathcal{S}$ to encode the strings in $\mathcal{B}$ defining a bijective mapping μ (and μ^{-1}) between $\mathcal{B}$ and a subset $\mathcal{W} \subseteq \mathcal{S}$ such that $\mathbf{card}(\mathcal{W}) = \mathbf{card}(\mathcal{B})$ (see Figure 1).

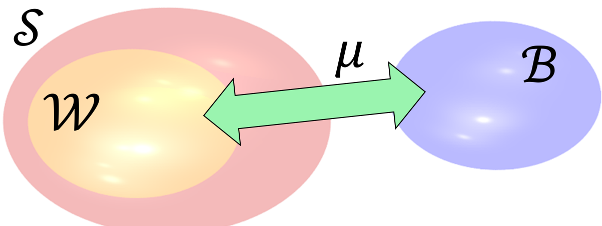

Figure 1. Mapping μ between binary strings of length n and sequences of t symbols from an alphabet $\boldsymbol{A}$.

In case $\mathcal{W} \subset \mathcal{S}$ let us call $\mathcal{D} = \mathcal{S} - \mathcal{W} \neq \varnothing$ the set of unused sequences in $\mathcal{S}$. In [20] we proposed to build a one-to-one mapping η between $\mathcal{D}$ and a subset $\mathcal{E}$ of $\mathcal{W}$ that may be used for reversible data hiding into the sequences of $\mathcal{S}$ (see Figure 2). Note that [20] assumes that $\mathbf{card}(\mathcal{D}) \leq \mathbf{card}(\mathcal{W})$ but we relax this constraint in the present work.

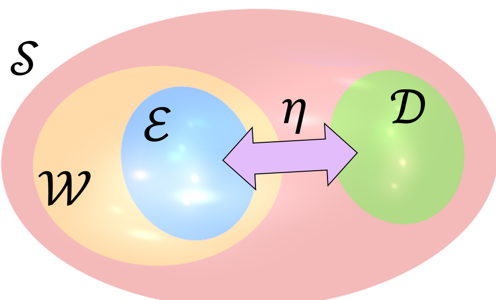

Figure 2. Mapping η between the unused sequences of $\mathcal{S}$ and a subset of the sequences of $\mathcal{W}$; note that $\mathcal{W} \subset \mathcal{S}, \mathcal{E} \subseteq \mathcal{W}, \mathcal{S} = \mathcal{D} \cup \mathcal{W}, \mathcal{D} \cap \mathcal{W} = \varnothing, \mathbf{card}(\mathcal{D}) = \mathbf{card}(\mathcal{E})$.

As an example, let us consider the Base41 encoding proposal [30]: there, three symbols from an alphabet of 41 characters are used to represent 65,536 two octet strings, 256 one octet strings, and 255 bit strings of length from 0 to 7 bits. We may thus see a generalization of the concepts shown in Figure 1. In this embodiment $\mathcal{S}$ is composed of the $41^3 = 68{,}921$ sequences of symbols of Base41; $\mathcal{B}$ consists of the previously listed $65{,}536 + 256 + 255 = 66{,}047$ binary strings represented by Base41. The subset $\mathcal{W}$ of $\mathcal{S}$ is composed by 66,047 strings, each one associated to a binary string with a function μ specified in [30]. The remaining $68{,}921 - 66{,}047 = 2874$ Base41 sequences constitute the set $\mathcal{D}$ (Figure 2).

When a binary string in $\mathcal{B}$ is encoded with a sequence S in $\mathcal{E}$ then it is possible to store a bit b of data by encoding the binary string with S if $b = 0$ or with $\eta(\mathrm{S}) \in \mathcal{D}$ if $b = 1$. Instead, if the binary string is encoded with a sequence belonging to $\mathcal{W} - \mathcal{E}$ no bit is embedded.

This method is proposed in [20] where it is also shown the reversibility of the process and the data extraction: please refer to that paper for more details.

To improve the data payload, in [23] we proposed to build $\mathcal{E}$ using the sequences in $\mathcal{W}$ mapped through μ to the most probable binary strings in $\mathcal{B}$: this depends on the statistic distribution of the binary strings in the input data (e.g., a file) which may be built analyzing a large set of files of the same type (for example, PDF or PNG format).

Starting from the observation that if a sequence S_0 in $\mathcal{E}$ is associated to a sequence S_1 in $\mathcal{D}$ it may reversibly carry one bit of extra data (during decoding S_0 means a 0 valued bit payload and restore S_0, S_1 means a 1 valued bit payload and restore S_0), then if a sequence S_{00} in $\mathcal{E}$ is associated to the sequences S_{01}, S_{10}, S_{11} in $\mathcal{D}$ then during encoding/decoding two bits of extra data may be embedded/extracted depending on which one of the four sequences is used and the original sequence S_{00} can be restored during decoding.

In the present paper, we propose to optimize the construction of the set $\mathcal{E}$ according to the frequency distribution of the data and having the following characteristics:

- The mapping η may associate a sequence in $\mathcal{W}$ to $2^k - 1, k \in \mathbb{N}$, sequences in $\mathcal{D}$, thus, in general, $\mathbf{card}(\mathcal{D}) \geq \mathbf{card}(\mathcal{E})$ and η is not one-to-one anymore: a sequence in $\mathcal{W}$ may carry k bits if it is associated to $2^k - 1$ sequences in $\mathcal{D}$;
- Given the relative frequencies f_i of the binary strings in $\mathcal{B}$, the objective is to associate n_i bits of payload to the i-th sequence to maximize the average payload p per binary sequence:

$$p = \sum_{i \in \mathcal{B}} n_i f_i. \tag{1}$$

As an example, consider the encoding proposed in [20] for Base45 [21]: two octets are mapped on three symbols from a base 45 alphabet, thus having $45^3 = 91{,}125$ configurations. Two octets result in $2^{16} = 65{,}536$ binary configurations each having its own relative frequency f_i.

A one-to-one mapping from binary configuration to Base45 configuration leaves $45^3 - 2^{16} = 91{,}125 - 65{,}536 = 25{,}589$ Base45 configurations unused.

Assigning n_i bits payload to the i-th binary configuration the average payload p_{45} per binary string is:

$$p_{45} = \sum_{i=0}^{2^{16}-1} n_i f_i \tag{2}$$

which must be maximized.

Assigning k bits payload to a binary configuration requires to use 2^k Base45 configurations: one Base45 sequence is the one normally assigned whilst the $2^k - 1$ are taken from the unused 25,589 Base45 sequences. This poses a constraint on the previous value (2):

$$\sum_{i=0}^{2^{16}-1} 2^{n_i} \leq 45^3 = 91{,}125. \tag{3}$$

It is obvious that the 25,589 sequences must be assigned to the highest frequency binary configurations maximizing (2). Another way to write constraint (3) is to look only at the unused sequences; if any configuration i is assigned n_i bits, it requires in total $2^{n_i} - 1$ extra configurations excluding itself. For example, if $n_i = 0$ we assign $2^0 = 1$ configurations to i (only itself), and it requires $2^0 - 1 = 0$ extra sequences; this is correct, since no payload will be carried by configuration i.

Knowing that the total number of extra (unused) sequences is 25,589 we can then write:

$$\sum_{i=0}^{2^{16}-1} (2^{n_i} - 1) \leq 25{,}589 \tag{4}$$

which is equivalent to Equation (3) since:

$$\sum_{i=0}^{2^{16}-1}(2^{n_i}-1) \leq 25{,}589 \Leftrightarrow \sum_{i=0}^{2^{16}-1} 2^{n_i} - \sum_{i=0}^{2^{16}-1} 1 \leq 25{,}589$$
$$\Leftrightarrow \sum_{i=0}^{2^{16}-1} 2^{n_i} - 2^{16} \leq 25{,}589$$
$$\Leftrightarrow \sum_{i=0}^{2^{16}-1} 2^{n_i} \leq 25{,}589 + 2^{16} \Leftrightarrow \sum_{i=0}^{2^{16}-1} 2^{n_i} \leq 91{,}125$$

that is exactly the aforementioned constraint.

In the following Section 5.1 we present the optimization method used to assign unused configurations in $\mathcal{D}$ and, consequently, bits to the legal Base45 sequences with the purpose of maximizing the average payload p_{45} per pair of octets (2).

5.1. Optimization Method

Before delving into the optimization problem, let us introduce the rationale behind its definition. As an example, let us consider a binary file of 25,589 unique pairs of octets, so that $25{,}589 - 1$ appear only once and one appears 100 times. Then, by assigning one bit to each of them, the payload obtained is exactly $100 + 25{,}589 - 1 = 25{,}688$.

If we decide to assign three bits to the one configuration that appears 100 times and one bit to all others (without exceeding the number of available unused configurations), the total payload obtained with this different approach will be $3 \cdot 100 + 25{,}589 - (2^3 - 1) = 25{,}884$, that is larger than the one bit case previously described.

This simple example motivates the search for the best possible assignment of bits to configurations in order to maximize the payload.

5.1.1. Problem Description

Given a function that needs to be optimized in the presence of constraints, one might think to solve the problem by means of combinatorial optimization [42]. In fact, there is the need to maximize the average payload (2) by assigning a certain number of bits n_i to the i-th binary configuration without exceeding the number of available configurations (3).

As a preliminary, let us say we only consider the most frequent (for each file type) Base45 configurations: baseline for our optimization is, in fact, the one-bit assignment to each one of the $45^3 - 2^{16} = 25{,}589$ most frequent configurations [23]. In this setting suppose, without loss of generality, that we sort all configurations in non-increasing order of their frequency f_i, such that $f_i \geq f_j$ when $i \leq j$. Then we can define the conditional probability to find a (frequent) configuration i as:

$$e_i = \frac{f_i}{\sum_{j=0}^{25{,}589-1} f_j} \qquad 0 \leq i < 25{,}589, \tag{5}$$

that is, the probability to find configuration i, subject to $i < 25{,}589$, in any file of a given type having a specific frequency distribution.

Our aim is then to maximize the *expected payload*, that is:

$$\text{maximize} \sum_{i=0}^{25{,}589-1} e_i n_i; \tag{6}$$

we note that maximizing (6) is equivalent to maximize (2) over the 25,589 most frequent configurations since:

$$\begin{aligned}\text{maximize} \sum_{i=0}^{25{,}589-1} e_i n_i &\Leftrightarrow \text{maximize} \sum_{i=0}^{25{,}589-1} \frac{f_i}{\sum_{j=0}^{25{,}589-1} f_j} n_i \\ &\Leftrightarrow \frac{1}{\sum_{j=0}^{25{,}589-1} f_j} \cdot \text{maximize} \sum_{i=0}^{25{,}589-1} f_i n_i \\ &\Leftrightarrow \text{maximize} \sum_{i=0}^{25{,}589-1} f_i n_i\end{aligned}$$

given that the term $\sum_{j=0}^{25{,}589-1} f_j$ is a constant and so we can remove it from the equation.

The only modification required on constraint (4) is the sum index that becomes $25{,}589 - 1$ from the old $2^{16} - 1$; this is the same thing we already did in (6).

It is easy to see that the objective function (6) under the one-bit assignment hypothesis [23] equals to 1, since $n_i = 1$ for all of the i most frequent configurations. Our optimization approach, then, aims to maximize this value: if we can find an assignment with an objective value above 1, it means that there is a possible gain w.r.t. the one-bit baseline.

5.1.2. Model Description

The previous considerations lead to the formulation (7)–(9) of a *nonlinear program* [43].

$$\text{maximize} \sum_{i=0}^{25{,}589-1} e_i n_i \tag{7}$$

$$\text{subject to} \sum_{i=0}^{25{,}589-1} (2^{n_i} - 1) \leq 25{,}589 \tag{8}$$

$$\text{where} \quad n_i \in \mathbb{Z}^+ \tag{9}$$

The amount of assigned bits n_i is not known beforehand, since we need to optimize the assignment in such a way that the objective function is maximized. These quantities are then represented as integer variables, and while the objective function itself (7) is linear w.r.t. the variables, constraint (8) is not.

In fact, (8) is *exponential* w.r.t. the variables, and having such constraints in an optimization model forbids the use of well-known and effective techniques [44] applicable only to models with linear objective functions and constraints, the so-called *linear programs*.

5.1.3. Constraint Linearization

Luckily, several common techniques are known (and can be easily found in any of the already cited books) to *linearize* this family of exponential and binary-related constraints, as we will see. Let us define a new *binary* variable $x_{ik} \in \{0, 1\}$ that is tightly linked to the different integer values that n_i can take.

$$x_{ik} = \begin{cases} 1 & \text{if } n_i = k \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

By the definition (10) we know that $x_{ik} = 1$ if and only if $n_i = k$; then, $x_{ik} = 1$ implies that we assign k bits to binary configuration i.

For constraint (8) we can derive a limit on the maximum power of 2, because no single n_i can be higher than a certain amount: given that there are a maximum of 25,589 unused Base45 configurations, we have that $2^{n_i} - 1 \leq 25{,}589$. This power limit is, in fact, 14: since $2^{14} - 1 = 16{,}383 < 25{,}589 < 2^{15} - 1 = 32{,}767$, we know that $n_i \leq 14$ for all i. This information can be propagated to variables (10) by defining only such variables for values of $k \in \mathbb{N}$ in the range $0 \leq k \leq 14$.

These new variables allow us to rewrite constraint (8) in a linear fashion: if $x_{ik} = 1$ we know that $n_i = k$, exactly k bits are assigned to the i-th binary configuration, and the

number of Base45 configurations required is 2^{n_i}. But $n_i = k$, and k is a known integer (a number, not a variable), so we can evaluate it for every term.

$$\sum_{i=0}^{25{,}589-1} \sum_{k=0}^{14} \left(2^k - 1\right) x_{ik} \leq 25{,}589 \tag{11}$$

Equation (11) will be used in place of (8) in the linear model. Additionally, the objective function needs to be changed: for the same reason, maximizing Equation (12) written with the x_{ik} variables

$$\sum_{i=0}^{25{,}589-1} \sum_{k=0}^{14} e_i k x_{ik} \tag{12}$$

is equivalent to maximize the objective function (7).

There is a need for an additional constraint, nevertheless. The value assumed by n_i is *unique*; in other words, we cannot assign both 0 bits and 3 bits to a configuration. Hence, exactly one variable x_{ik} for every configuration i can be 1, all the others must be 0. Constraints (13) are then required and will be added to the model.

$$\sum_{k=0}^{14} x_{ik} = 1 \qquad \forall\, 0 \leq i < 25{,}589 \tag{13}$$

5.1.4. Linear Model

Taking into consideration the previous discussion the new variable definition and the added constraints, we can write down the complete *linear* program.

$$\text{maximize} \quad \sum_{i=0}^{25{,}589-1} \sum_{k=0}^{14} e_i k x_{ik} \tag{14}$$

$$\text{subject to} \quad \sum_{i=0}^{25{,}589-1} \sum_{k=0}^{14} \left(2^k - 1\right) x_{ik} \leq 25{,}589 \tag{15}$$

$$\sum_{k=0}^{14} x_{ik} = 1 \qquad \forall\, 0 \leq i < 25{,}589 \tag{16}$$

$$\text{where} \quad x_{ik} \in \{0, 1\} \tag{17}$$

The model (14)–(17) can then be solved by means of standard and efficient well-known optimization methods and approximated with good quality heuristics. In the empirical analysis part, Section 6, we are going to obtain the optimum, hence the best possible assignment given the frequencies, by solving this model with `CPLEX` commercial solver [45]. A *greedy approximation algorithm* that gives results indistinguishable from the real optima, given the particular structure [46] of the model, is under development.

It is worth noting that the input dimension for the optimization model never changes: since for Base45 we are going to have always 25,589 unused configurations and the maximum possible required power of 2 is 14, the number of variables x_{ik} is always $25{,}589 \times 14$ and the number of constraints (16) is always 25,589. Given this consideration, we remark that for these relatively *small* dimensions it is easy for any solver to efficiently find the real optima in a small amount of time.

Considering also the fact that, given the configuration frequencies, the entire optimization must be done only *once* for each file type to build the table that will be later used for the embedding, the few seconds of execution time required by the solver to find the best possible assignment has no impact over the encoding/embedding procedure.

5.2. Encoding and Decoding Algorithms

The encoding and decoding algorithms of our proposed **Hide45** method make use of a table *EncDec* that implements the mappings μ and η. *EncDec* contains a row (i.e., a record) for every binary string in $\mathcal{B}$ and its corresponding sequence in $\mathcal{W}$: this realizes the one-to-one function μ (and its inverse μ^{-1}) in Figure 1.

Table 1 reports the two types of record needed by the table *EncDec* to store the information for the encoding and the decoding: a Type 1 record is used for sequences carrying no payload bits, that is sequences in $\mathcal{W} - \mathcal{E}$. On the other hand, a Type 2 record is used for a binary string that is mapped by μ to a sequence S_0 in $\mathcal{E}$ that is capable of carrying payload bits, e.g., k bits: in this case S_0 and other $2^k - 1$ sequences in $\mathcal{D}$ make up an ordered list $[S_0, S_1, S_2, \ldots, S_{2^k-1}]$ that can be indexed by the value expressed by the k bits to reversibly encode them. Function η (Figure 2) operates in this way: if the k bits have value i then η outputs S_i in place of S_0; given S_i in $[S_0, S_1, S_2, \ldots, S_{2^k-1}]$ its inverse η^{-1} returns k bits valued i and the sequence S_0. It is obvious that:

- *EncDec* is composed of $2^n = 2^{16}$ rows;
- The sequences in the last column of *EncDec* form a partition of $\mathcal{S}$.

The **encoding procedure** has as input the stream of payload data bits to be embedded and processes one binary string $B \in \mathcal{B}$ at a time to be converted in printable form as follows:

- If $S_0 = \mu(B) \in \mathcal{W} - \mathcal{E}$ (Type 1 entry in *EncDec*) then no payload bits can be embedded and S_0 is output;
- Otherwise $S_0 = \mu(B) \in \mathcal{E}$, thus its Type 2 entry in *EncDec* is used getting k, the number of bits that can be stored with S_0, and k payload bits are read from the stream whose decimal value f is used as an index by η to select and output the sequence S_f from the list $[S_0, S_1, S_2, \ldots, S_{2^k-1}]$ found in the corresponding entry.

The **decoding procedure** performs the inverse mapping and extraction working on an input sequence $S \in \mathcal{S}$ after the other to obtain the original data in binary form and the payload bits:

- If S is associated to a Type 1 entry in *EncDec* then no payload data is stored and the binary string $B = \mu^{-1}(S)$ is output;
- Otherwise S is searched in the last field of the Type 2 entries of *EncDec*: when found, its index in the list is used to recover the k payload bits and the corresponding sequence $S_0 = \eta^{-1}(S) \in \mathcal{E}$ is used to output $B = \mu^{-1}(S_0)$.

Table 1. Type of records of table *EncDec* for storing the functions μ, η and their inverses μ^{-1} and η^{-1}.

Entry Type	Symbol $\in \mathcal{B}$	Sequence $\in \mathcal{S}$	Payload Bits	Sequences for Embedding
Type 1	B	$S \in \mathcal{W} - \mathcal{E}$	0	$\varnothing$
Type 2	B	$S_0 \in \mathcal{E}$	k	$[S_0, S_1, S_2, \ldots, S_{2^k-1}]$, $S_1, S_2, \ldots, S_{2^k-1} \in \mathcal{D}$

6. Empirical Analysis

The set of experiments we set up was aimed at verifying the improvement over the one-bit per (most frequent) symbol encoding presented in [23]: thus we considered the same file formats and incidentally used the same files, but other files of the same format should present the same results due to the large number of files we employed.

The files and formats we used were:

- JPEG: 500 images 768 × 576 pixels 24 bpp, quality 80, 2 × 2 subsampling;
- TIFF: 500 images 768 × 576 pixels 24 bpp, uncompressed;
- PNG: 500 images 768 × 576 pixels 24 bpp, compressed;
- ZIP: 7000 files of 500,000 octets;
- BZ2: 1100 files of 100,000 octets;

- GZ: 7000 files of 500,000 octets;
- PDF: 2000 files of 500,000 octets;
- MP3: 3000 audio files having size 500,000 octets.

For every file format we performed a 5-fold cross validation with a proportion of 4:1 using the first set to compute the statistic distribution of the $2^{16} = 65{,}536$ symbols and the second set to test the payload capability of the optimized bit assignment. Experiments were run on laptop with CPU Intel® Core™ i7-1165G7 2.80 GHz with 16 GB RAM.

Comparisons between the method [23] and the improvement described in this paper are reported in Table 2. The first column of these tables reports the file format, and the second and the third report the average payloads (and standard deviation) for the 5-fold cross validation obtained by method developed in [23] and by the proposed Hide45 algorithm, respectively. The payload is expressed in bit per output symbol (bps), that is average number of bits carried by a Base45 character of the embedded output file. The "Th. Gain" (theoretical gain) column refers to the expected payload gain computed by the optimization function (6). The "Emp. Gain" (empirical gain) column reports the percentage advantage of the proposed method over the one-bit baseline payload, that is used in [23].

Table 2. Set of files in JPEG, TIFF, and PNG; ZIP, BZ2, and GZ; PDF and MP3 formats: average payloads per output symbol and gains computed for [23] and the proposed method Hide45.

File Format	Avg Payload [23] [bps]	Avg Payload Hide45 [bps]	Th. Gain [%]	Emp. Gain [%]
JPEG	0.17025 ± 0.01270	0.17807 ± 0.00494	4.6	4.6
TIFF	0.26252 ± 0.04132	0.31376 ± 0.01758	22.0	19.5
PNG	0.17091 ± 0.01251	0.17235 ± 0.00503	1.4	0.8
ZIP	0.14177 ± 0.01708	0.14251 ± 0.00325	2.5	0.5
BZ2	0.13959 ± 0.01207	0.14140 ± 0.00895	3.4	1.3
GZ	0.13404 ± 0.00540	0.13410 ± 0.00203	0.07	0.04
PDF	0.16998 ± 0.02877	0.26002 ± 0.04044	57.4	53.0
MP3	0.16989 ± 0.01656	0.24231 ± 0.03960	45.2	42.6

To account for the different improvements in payload gain depending on the file format we analyzed, for each format, the relative frequency distributions of the symbols sorting them in decreasing order. The plots are reported in Figures 3 and 4: the first one shows the cumulative distribution of the relative frequencies for all the 65,536 symbols while the second magnifies on the first 128 most frequent symbols of each file format (note that the symbols are not the same being the use of the various symbols a peculiarity of each format).

From these figures it may be observed that for the distributions having a steepest start (for example, the MP3 file type distribution) there is a bigger gain in assigning a variable number of bits to every symbol: this is due to the fact that when there are symbols more frequent than others then allocating more bits to them increases the average payload (2).

We remark that the execution times for the encoding/embedding procedures do not depend upon the given map, therefore there is no difference between [23] and the proposed method Hide45 with regard to the required time. As already stated in Section 5.1, the optimization procedure that gives the best possible assignment table given the configuration frequencies is computed only once for every file type. Considering the fact that this optimization, in our testing environment, requires in all cases between 4 s and 8 s with a median of 5 s, it has no effective impact over the whole procedure.

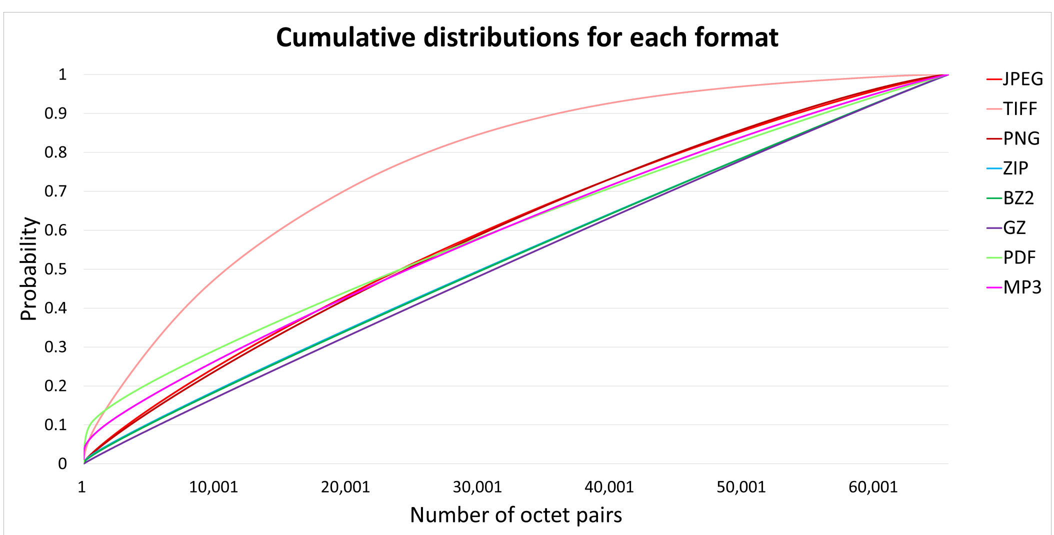

Figure 3. Cumulative distributions of all octet pairs sorted in decreasing order of frequency.

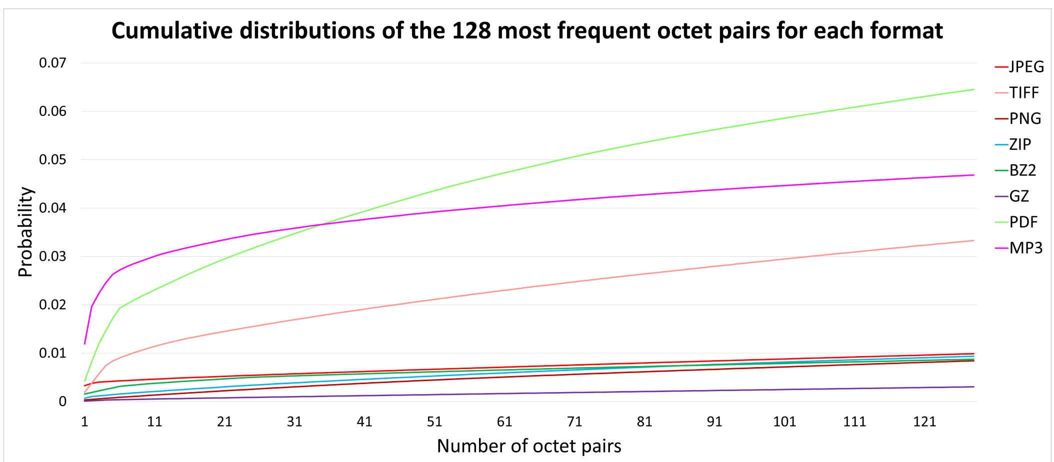

Figure 4. Cumulative distributions of the most frequent 128 octet pairs (zoom on the leftmost area of graph in Figure 3) sorted in decreasing order of frequency.

7. Conclusions

This work has presented **Hide45**, an improvement over a previously published method for embedding data into printable encoded strings using a Base45 alphabet. The proposed solution optimizes the bit assignments for the sequences normally used for the encoding exploiting the unused sequences to give more bits to the highest frequency sequences.

Modeling and solving the problem as an integer linear program leads to significant improvement upon previously published methods. Experimental results showed a meaningful improvement (up to 53%) for those data types having skewed distributions of their double octets binary data. Moreover, it may be shown (see Appendix A) that the encoding/decoding table can be compressed in less than 11 Ko.

Since **Hide45** is able to store a higher amount of information by an optimal assignment of unused sequences, it can be useful to embed into files longer and safer security fingerprints and digital signatures, among more general data. Furthermore, part of the embedded information can be employed to check whether the transmitted file has undergone to an attack aiming at its modification.

A limitation of this method is related to the fact that the optimization is done using a statistic of sequence frequencies over several files. Even though the average payloads exhibit a very low standard deviation among file types over the tested files (see Table 2), in the case where a file exhibits a frequency distribution different from the one considered by the statistic, it will result in a reduced payload. A possible development of this technique

can be the building of a encoding/decoding table *EncDec* specific for every file, by optimizing over its own frequency distribution; this has a disadvantage nevertheless, since the table must be transmitted with the file. This can lead to advantages only for bigger files, since the encoding/decoding table will occupy some of the embedding capacity of the file itself; the objective for future research will be to analyze how to embed this table, compute the required space, and check whether this approach will be more convenient than using a precomputed table.

Further developments of the method will be also towards the analysis of different optimizations methods.

Author Contributions: All authors contributed equally to this work, in the following categories. M.B., D.C., A.D.: Conceptualization, methodology, software, validation, writing—original draft, writing—review and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been supported by the Italian Ministero dell'Università e della Ricerca.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No data sharing is required as any set of files having a format analyzed in this work will produce comparable and similar results.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

In this appendix a possible compression method for the coding table *EncDec* is presented.

The 45^3 Base45 configurations and the 2^{16} binary strings of 16 bits are sorted in ascending order. After that, the first 2^{16} of the 45^3 Base45 configurations are one-to-one mapped to the 2^{16} binary strings of 16 bits according to the ascending order.

The remaining $45^3 - 2^{16}$ Base45 configurations will be assigned, from first to last, to the binary strings carrying payload bits.

From these considerations it is obvious that to save the *EncDec* table it is sufficient to specify the number of payload bits $0 \leq k \leq 14$ that are encoded by every binary string, from the first to the last in the ordered list.

A shrunk version of the *EncDec* table is thus a sequence of values $0 \leq k \leq 14$ each one saved with 4 bits: the resulting compressed table will be $2^{16}/2 = 32$ Ko in size. A more wise encoding will perform a Huffman compression because the smaller values of k will be more frequent than the larger ones (e.g., it is obvious that a value of $k = 14$, if present, will occur only once): simple tests showed that a typical distribution Huffman encoded requires less than 11 Ko (see Algorithm A1).

Algorithm A2 presents the pseudo-code to recover the *EncDec* table (refer to Table 1) from the shrunk version.

Algorithm A1 Pseudo-code of the algorithm for saving a compressed version of the *EncDec* table

Require: $0 \leq n \leq 14$
  n = number of payload bits associated to B
  Encode n represented with 4 bits or using a pre-computed Huffman code
  Write n on file

Algorithm A2 Pseudo-code of the algorithm for building the *EncDec* table

```
UnusedPointer = 65,536
for B = 0 to 2^16 − 1 do
  Read n (number of payload bits associated to B) from file
  EncDec[B].Symbol = BinaryRepresentationOn16Bits(B)
  EncDec[B].Sequence = Base45Encoding(B)
  EncDec[B].PayloadBits = n
  if n == 0 then
    EncDec[B].EntryType = Type 1
    EncDec[B].SeqForEmbed = ∅
  else if n ≤ 14 then
    EncDec[B].EntryType = Type 2
    if UnusedPointer +2^n − 1 ≥ 91,125 then
      error("Number of Base45 configurations exceeded")
    else
      for c = 1 to 2^n − 1 do
        EncDec[B].SeqForEmbed ∪ = Base45Encoding(UnusedPointer)
        UnusedPointer ++
      end for
    end if
  else
    /* n > 14 */
    error("Wrong number of bits associated to a configuration")
  end if
end for
```

References

1. Cox, I.J.; Miller, M.L.; Bloom, J.A.; Fridrich, J.; Kalker, T. *Digital Watermarking and Steganography*, 2nd ed.; Morgan Kaufmann: Burlington, MA, USA, 2007. [CrossRef]
2. Botta, M.; Cavagnino, D.; Pomponiu, V. Image Fragile Watermarking through Quaternion Linear Transform in Secret Space. *J. Imaging* **2017**, *3*, 34. [CrossRef]
3. Singh, A.K.; Kumar, B.; Singh, G.; Mohan, A. *Medical Image Watermarking*; Springer: Berlin/Heidelberg, Germany, 2017.
4. Begum, M.; Uddin, M.S. Digital Image Watermarking Techniques: A Review. *Information* **2020**, *11*, 110. [CrossRef]
5. Salah, E.; Amine, K.; Redouane, K.; Fares, K. A Fourier transform based audio watermarking algorithm. *Appl. Acoust.* **2021**, *172*, 107652. [CrossRef]
6. Hua, G.; Huang, J.; Shi, Y.Q.; Goh, J.; Thing, V.L. Twenty years of digital audio watermarking—A comprehensive review. *Signal Process.* **2016**, *128*, 222–242. [CrossRef]
7. Bloom, J.A.; Polyzois, C. Watermarking to track motion picture theft. In Proceedings of the Conference Record of the Thirty-Eighth Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, USA, 7–10 November 2004; Volume 1, pp. 363–367. [CrossRef]
8. Dubey, N.K.; Kumar, S. A Review of Watermarking Application in Digital Cinema for Piracy Deterrence. In Proceedings of the 2014 Fourth International Conference on Communication Systems and Network Technologies, Bhopal, India, 7–9 April 2014; pp. 626–630. [CrossRef]
9. Asikuzzaman, M.; Pickering, M.R. An Overview of Digital Video Watermarking. *IEEE Trans. Circuits Syst. Video Technol.* **2018**, *28*, 2131–2153. [CrossRef]
10. Botta, M.; Cavagnino, D.; Gribaudo, M.; Piazzolla, P. Fragile Watermarking of 3D Models in a Transformed Domain. *Appl. Sci.* **2020**, *10*, 3244. [CrossRef]
11. Vasic, B.; Raveendran, N.; Vasic, B. Neuro-OSVETA: A Robust Watermarking of 3D Meshes. In Proceedings of the International Telemetering Conference Proceedings, Las Vegas, NV, USA, 21–24 October 2019; International Foundation for Telemetering: San Diego, CA, USA, 2019; Volume 55.
12. Wang, Y.P.; Hu, S.M. A New Watermarking Method for 3D Models Based on Integral Invariants. *IEEE Trans. Vis. Comput. Graph.* **2009**, *15*, 285–294. [CrossRef] [PubMed]
13. Adi, Y.; Baum, C.; Cisse, M.; Pinkas, B.; Keshet, J. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. In Proceedings of the 27th USENIX Security Symposium (USENIX Security 18), Baltimore, MD, USA, 15–17 August 2018; pp. 1615–1631.

14. Chen, H.; Rouhani, B.D.; Fu, C.; Zhao, J.; Koushanfar, F. DeepMarks: A Secure Fingerprinting Framework for Digital Rights Management of Deep Learning Models. In Proceedings of the 2019 on International Conference on Multimedia Retrieval, Ottawa, ON, Canada, 10–13 June 2019; pp. 105–113.
15. Botta, M.; Cavagnino, D.; Esposito, R. NeuNAC: A novel fragile watermarking algorithm for integrity protection of neural networks. *Inf. Sci.* **2021**, *576*, 228–241. [CrossRef]
16. Tartaglione, E.; Grangetto, M.; Cavagnino, D.; Botta, M. Delving in the loss landscape to embed robust watermarks into neural networks. In Proceedings of the 2020 25th International Conference on Pattern Recognition (ICPR), Milan, Italy, 10–15 January 2021; pp. 1243–1250. [CrossRef]
17. Kamaruddin, N.S.; Kamsin, A.; Por, L.Y.; Rahman, H. A Review of Text Watermarking: Theory, Methods, and Applications. *IEEE Access* **2018**, *6*, 8011–8028. [CrossRef]
18. Ali, A.E. A New Text Steganography Method By Using Non-Printing Unicode Characters. *Eng. Tech. J.* **2010**, *28*, 72–83.
19. Liu, T.Y.; Tsai, W.H. A New Steganographic Method for Data Hiding in Microsoft Word Documents by a Change Tracking Technique. *IEEE Trans. Inf. Forensics Secur.* **2007**, *2*, 24–30. [CrossRef]
20. Botta, M.; Cavagnino, D. A Framework for Reversible Data Embedding into Base45 and Other Non-Base64 Encoded Strings. *Appl. Sci.* **2022**, *12*, 241. [CrossRef]
21. Fältström, P.; Ljunggren, F.; van Gulik, D.W. *The Base45 Data Encoding*; RFC 9285; RFC Editor: Phoenix, AZ, USA, 2022. [CrossRef]
22. Incorporated, A.S. *PostScript Language Reference*, 3rd ed.; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1999.
23. Botta, M.; Cavagnino, D. Improving data embedding capacity into Base45 encoded strings. *Eng. Rep.* **2023**, *5*, e12622. [CrossRef]
24. Botta, M.; Cavagnino, D. Escaping Printable Encoded Streams to Embed Out-of-Band Data. *Appl. Sci.* **2023**, *13*, 6926. [CrossRef]
25. Josefsson, S. *The Base16, Base32, and Base64 Data Encodings*; RFC 4648; RFC Editor: Phoenix, AZ, USA, 2006. [CrossRef]
26. Group, T.P. PHP Math Functions, Base_Convert() Function. Available online: https://www.php.net/manual/en/function.base-convert.php (accessed on 10 July 2023).
27. Corporation, M. JavaScript Reference, Number Constructor, toString() Method. Available online: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number (accessed on 10 July 2023).
28. Foundation, P.S. Python 3.11.2 Documentation, int() Class. Available online: https://docs.python.org/3/library/functions.html (accessed on 10 July 2023).
29. Veljkovic, S. Base41. 2014. Available online: https://github.com/sveljko/base41 (accessed on 10 July 2023).
30. Botta, M.; Cavagnino, D. Base41: A proposal for printable encoding of bit strings. *Eng. Rep.* **2023**, *5*, e12606. [CrossRef]
31. Duggan, R. Base-56 Integer Encoding in PHP. 2009. Available online: http://rossduggan.ie/blog/codetry/base-56-integer-encoding-in-php/index.html (accessed on 10 July 2023).
32. Antonopoulos, A.M. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2014.
33. Nakamoto, S.; Sporny, M. The Base58 Encoding Scheme. Internet-Draft Draft-Msporny-Base58-03, Internet Engineering Task Force. Expired. 2021. Available online: https://datatracker.ietf.org/doc/draft-msporny-base58/03/ (accessed on 10 July 2023).
34. Wu, P.C. A base62 transformation format of ISO 10646 for multilingual identifiers. *Softw. Pract. Exp.* **2001**, *31*, 1125–1130. [CrossRef]
35. He, K.; Xu, X.; Yue, Q. A secure, lossless, and compressed Base62 encoding. In Proceedings of the 2008 11th IEEE Singapore International Conference on Communication Systems, Guangzhou, China, 19–21 November 2008; pp. 761–765. [CrossRef]
36. Elz, R. *A Compact Representation of IPv6 Addresses*; RFC 1924; RFC Editor: Phoenix, AZ, USA, 1996. [CrossRef]
37. Henke, J. basE91 Encoding. 2006. Available online: https://base91.sourceforge.net/ (accessed on 10 July 2023).
38. He, D.; Sun, Y.; Jia, Z.; Yu, X.; Guo, W.; He, W.; Qi, C.; Lu, X. A Proposal of Substitute for Base85/64–Base91. In Proceedings of the SUMMER 8th International Conference on Computing, Communications and Control Technologies: CCCT, Orlando, FL, USA, 29 June–2 July 2010.
39. Albertson, K. Base-122 Encoding. Available online: https://blog.kevinalbs.com/base122 (accessed on 10 July 2023).
40. Yergeau, F. *UTF-8, a Transformation Format of ISO 10646*; RFC 3629; RFC Editor: Phoenix, AZ, USA, 2003. [CrossRef]
41. Wikipedia. Binary-to-Text Encoding. Available online: https://en.wikipedia.org/wiki/Binary-to-text_encoding (accessed on 10 July 2023).
42. Papadimitriou, C.H.; Steiglitz, K. *Combinatorial Optimization: Algorithms and Complexity*; Prentice-Hall: Hoboken, NJ, USA, 1982.
43. Bertsekas, D.P. *Nonlinear Programming*; Athena Scientific: Nashua, NH, USA, 1995.
44. Vanderbei, R.J. *Linear Programming: Foundations and Extensions*; Springer: New York, NY, USA, 1998.
45. IBM ILOG CPLEX Optimization Studio. 2023. Available online: https://www.ibm.com/products/ilog-cplex-optimization-studio (accessed on 10 July 2023).
46. Martello, S.; Toth, P. *Knapsack Problems: Algorithms and Computer Implementations*; John Wiley & Sons: Hoboken, NJ, USA, 1990.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.