"Mircea cel Batran" Naval Academy Scientific Bulletin, Volume XIX – 2016 – Issue 1
Published by "Mircea cel Batran" Naval Academy Press, Constanta, Romania // The journal is indexed in: PROQUEST / DOAJ / DRJI / JOURNAL INDEX / I2OR / SCIENCE LIBRARY INDEX / Google Scholar / Crossref / Academic Keys / ROAD Open Access / OAJI / Academic Resources / Scientific Indexing Services / SCIPIO

RESURFACING OLD DATA COMPRESSION & ENCRYPTION ALGORITHMS FOR EXTRA SECURITY SHELL

Mitruț CARAIVAN[1]
Vasile DOBREF[2]
Paul BURLACU[3]
[1] Lecturer PhD eng., Naval Academy "Mircea cel Batran", Constanța, caraivanmitrut@yahoo.com
[2] Professor PhD eng., Naval Academy "Mircea cel Batran", Constanța, vasile.dobref@anmb.ro
[3]Lecturer PhD eng., Naval Academy "Mircea cel Batran", Constanța, paul.burlacu@anmb.ro

Abstract: *This paper presents a short history of data compression and encryption technologies starting with World War I and their possible value today by resurfacing old and forgotten algorithms as an increased security shell possibility for modern data files storage. It focuses on a case study using available internet tools as of 2016 and emphasizes on the results which relieve a blind eye over old and dusty data compression and encryption algorithms following data encapsulation, therefore showing the possibility of adding easily an extra security layer to any contemporary cutting-edge data protection method.*
Keywords: *data compression, archive, security, data protection, encryption algorithms.*

I. INTRODUCTION

This paper presents shortly some basic notions in the domains of data compression and data encryption, such as redundancy, compression rate, entropy, data set volume and length, so on and so forth. We have studied a classification of classic entropic methods of data compression based upon data loss criteria and symbols encoding procedures and have focused on the principal steps of Huffman variant for static compression. We will prove in the next pages that the use of such classic and long-forgotten algorithms, such as Shannon-Fano [1], adaptive Huffman coding [2]and variations, used together with slight personal alterations in the process of encoding the elements, while keeping a deceptive nomination of the target digital files, may prove to offer a considerable protection nowadays for personal data files, counting exactly on their most obvious quality: age. These 1940's algorithms, because of their age, although not being a commercial – industrial-use viable solution, may in fact prove to be one of the easiest methods of data compression and encryption for the average user, as long as the 'compression' part is not that important for the end-user.

We are positioning this paper's goal algorithm somewhere between obfuscation-hashing and data encryption in the true meaning of the word. Disambiguation [3]: *Hashing* serves the purpose of ensuring data integrity, i.e. if something is changed, the user knows that the data was altered. *Obfuscation* is used to hide the meaning and is often used with computer code to help prevent successful reverse engineering and/or theft of a product's functionality. On the other hand, e*ncryption*is for maintaining data confidentiality and requires the use of a key (kept secret) in order to return to meaningful data.

We are constantly drawing a parallel between the two distinct domains of data compression and data security – cryptology, while keeping a common sense about the personal, home-use applicability of state-of-the art algorithms.

II. DATA COMPRESSION

A tight concept related to data compression is *redundancy*. The primary objective of a compression method is to detect, locate and eliminate redundancy, meaning the repeatability of information within the data set. One other important parameter defined for the compression method performance is the rate of compression (Equation 1):

$$\gamma \triangleq \left[1-\frac{Lc}{Lo}\right]\cdot 100\%$$

Equation 1 Rate of Compression

where Lc is the length of the compressed data set and Lo is the length of the original data set.

Ideally, γ=100% would mean a perfect compression. Obviously, in practice we have only γ<100%. Redundancy however is a constant of the data set and it's not dependent on the compression algorithm used. It can only be measured with a statistical model associated to the initial data set, starting with the frequencies of appearance of the alphabet of the data set. The frequency of appearance is given by the number of appearances of a specific symbol in the data set and the data set volume. This volume is different of the length, as it refers to the total number of symbols of the data set, while the length refers to the total number of bits needed to represent the information.

DOI: 10.21279/1454-864X-16-I1-027

If we consider *v(s)* as the appearance frequency of the symbol *s* and this is an estimation of the appearance probability of symbol *s* within the data set, we can then calculate the minimum number of bits needed to re-encode symbol *s* in order to avoid any confusion with any other symbol. This is called *entropy* (Equation 2) and it can be a fraction, which in practice is rounded up to the nearest integer. The difference between the entropy and the frequency of appearance of a specific symbol is a measure of the intrinsic redundancy of that symbol within the data set.

$$\mathrm{H}(s) \triangleq -\log_2 \nu(s)$$

Equation 2: Entropy of a symbol within a data set

The *source entropy* as a global information characteristic is given by Equation 3:

$$H(x) \triangleq \sum_{i=1}^{M} p_i \cdot \log_2 \frac{1}{p_i}$$

Equation 3: Source Entropy

, where p_i defined in Equation 4 represents the probability distribution over ***X***alphabet.

$$p_i \triangleq P(x_i)$$

Equation 4: Probability Distribution

The source entropy is expressed in bits/symbol and indicates the mean quantity of information of alphabet ***X***.

Apart from this 1940's idea of using alphabet statistical-based models in order to reduce the redundancy of the original data set, starting with the 1970's a new approach was evolving. This was developed based on dictionaries for re-encoding data, which meant replacing a whole string of repeating symbols with only one code, which was in fact an index pointer to the dictionary entry. The link between statistics and the dictionaries was now clear, and it could be defined by entropy and redundancy.

All methods start with the idea of reducing the natural redundancy of the initial data set and this can be done only if some symbols or group of symbols (subsets) are in some sort of repetition.

The methods based upon the appearance frequency are usually parsing the data set twice, as the first loop is necessary for determining the frequencies. The methods based on grouping are counting on the consecutive of separated repeated appearance of the same symbol or group of symbols, which requires only one loop over the data set, with some exceptions on specific areas.

Looking over the criteria for the symbol's graphs construction, there are static and dynamic methods. The static establishes a correspondence between the words and codes, which is constant and fixed during time. The dynamic methods determine this correspondence as variable over time, based upon the relative frequency of the words up to the current encoding moment.

Lossless Data Compression (Fig. 1) is the most frequent case we encounter, as at the end we find a perfect reconstruction of the initial information.

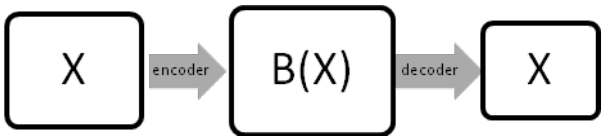

Fig. 1Lossless Data Compression

Lossy Data Compression (Fig. 2) accepts the reduction of the source entropy, therefore the decoder will not be capable of perfect reconstruction of the initial source data.

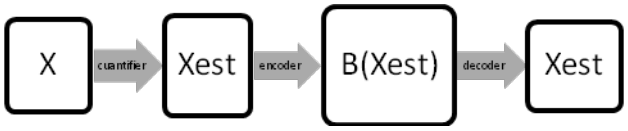

Fig. 2Lossy Data Compression

Xest is sufficient close to *X* (this being defined by a distance function) so the differences between *Xest*and *X* will not exceed a certain threshold.

If γ<0, then the algorithm makes an expansion of the initial data set, not a compression [4]. This is a common practice encountered in hashing.

*Hashing*is used for validating the integrity of content by detecting all modification thereof via obvious changes to the hash output.Some of the well-known examples are: SHA-3, MD5 (now obsolete). Technically, hashing takes arbitrary input and produce a fixed-length string that has the following attributes:

• The same input will always produce the same output;

• Multiple disparate inputs should not produce the same output;

• It should not be possible to go from the output to the input.

• Any modification of a given input should result in drastic change to the hash.

Hashing is used in conjunction with authentication to produce strong evidence that a given message has not been modified. This is accomplished by taking a given input, encrypting it with a given key, hashing it, and then encrypting the key with the recipient's public key and signing the hash with the sender's private key.When the recipient opens

DOI: 10.21279/1454-864X-16-I1-027

the message, they can then decrypt the key with their private key, which allows them to decrypt the message. They then hash the message themselves and compare it to the hash that was signed by the sender. If they match it is an unmodified message, sent by the correct person.

The purpose of *obfuscation* is to make something harder to understand, usually having in mind the purposes of making that specific item more difficult to copy or to be hacked.

It's important to note that obfuscation is not a strong control (like properly employed encryption), but rather an obstacle. Similar to encoding, it can often be reversed by using the same technique that obfuscated it. Otherwise it is simply a manual process that takes time to work through.

Another key aspect about obfuscation is that there is a limitation to how obscure the code can become, depending on the content being obscured. If you are obscuring computer code, for example, the limitation is that the result must still be consumable by the computer or else the application will cease to function. Examples: JavaScript Obfuscator, ProGuard.

III. DATA ENCRYPTION

*Encryption*transforms data in order to keep it secret from other unintended parties. Rather than focusing on usability, the goal is to ensure the data cannot be consumed by anyone other than the intended recipient.It uses a key, which is kept secret, in conjunction with the plaintext and the algorithm, in order to perform the encryption operation. As such, the cypher-text, algorithm, and key are all required to return to the plaintext.

The purpose of *encoding* is to transform data so that it can be properly and safely consumed by a different type of system(e.g.: binary data being sent over email or viewing special characters on a web page). The goal is not to keep information secret, but rather to ensure that it's able to be properly consumed [3].

The first systematic results were found during World War I, when the necessity of data encryption and secret information transmission became paramount. Complex encryption keys usually bear two types of information: fake, which can be deciphered without the key and the real one, which uses the key known beforehand. No matter how sophisticated the method of protection is, there is always the possibility to reveal the data, especially when the message bears information about the protection key. There is no ideal protection key. Using enough processing power, trying different algorithms and methods, any protection can be removed in the end in a certain amount of time. This time represents in fact the ultimate barrier against rogue recipients.

In support of this theory we present a short history of MD5 – which is a hash function long used also for encryption. Initially developed by Ron Rivest in 1991 as a hash function on 128 bits, it was long time considered impenetrable and it made its way as standard RFC1321 for application security and into Cisco routers as main method for encryption, the brand being well known for their high-end intranet security. In 2004 there is the first announcement of a possible security breach, in 2005 we get the first live demonstration for finding the private key and in 2006 the published Klima algorithm finds the key within a single minute on an average notebook.

Encryption is usually done with misleading information encapsulated in the data set creating false tracks. Good compression on the other hand means minimum redundancy and cutting down redundancy means exactly the elimination of any possible addition of misleading track. So, good compression means weak encryption and vice-versa. DES - Data Encryption Standard was once a predominant symmetric-key algorithm for the encryption of electronic data. It was developed in the early 1970's at IBM and it was highly influential in the advancement of modern cryptography in the academic world. DES is now considered to be insecure for many applications.

It was replaced with Triple DES, which is 168-bit gigantic task for the attacked to perform the exhaustive search on e^{2168}. Now these are being replaced by AES – Advanced Encryption Standard.

IV. STATE OF THE ART

NIST – National Institute of Standards and Technology defines SP1800 Cyber-security Practice Guides from 2015 onwards.

The RSA cryptosystem is widely used for secure data transmission. It is considered an asymmetric algorithm due to its use of a pair of keys. A public key is used to encrypt the message and a private key to decrypt it. The result of RSA encryption is a huge batch of data which take quite a bit of time and processing power to break, but with timing and side-channel analysis attacks, adaptive chose cipher-text this can be achieved.

Blowfish is yet another free algorithm designed to replace DES, which claims it has never been defeated. This symmetric cipher splits messages into blocks of 64 bits and encrypts them individually.

Two fish is using a 256 bits long key and is regarded as one of the fastest available symmetric techniques. It is also open-source.Other highly-secure ciphers are Serpent and RC6.

The Advanced Encryption Standard (AES) is the algorithm today trusted as the standard by U.S.

DOI: 10.21279/1454-864X-16-I1-027

Government [5] and numerous organizations[6]. The algorithm is based on several substitutions, permutations and linear transformations, each executed on data blocks of 16 byte – therefore the term "blockcipher". Those operations are repeated several times, called "rounds". During each round, a unique round-key is calculated out of the encryption key, and incorporated in the calculations. Based on the block structure of AES, the change of a single bit, either in the key, or in the plaintext block, results in a completely different cipher-text block, which is a clear advantage over traditional stream ciphers.Although it is extremely efficient in its 128-bit form, AES was also developed to use 192 and 256 bits for heavy duty encryption purposes. AES is largely considered impervious to all attacks except brute force, which attempts to decipher messages using all possible combinations in the 128, 192 or 256-bit cipher. AES using the Rijndael cipher [7] with a key size of 256 is the most widely-accepted encryption algorithm today, although it is not necessarily the most secure from the mathematically point-of-view.

Cyber-attacks are constantly evolving, so security specialists must stay up-to-date with state-of-the art algorithms and their constant variations[8]. A new method called *Honey Encryption* will deter hackers by serving up fake data for every incorrect guess of the key code. This unique approach not only slows down attacks, but potentially buries the correct key in a haystack of false served-data. There are emerging methods like quantum key distribution, which shares keys embedded in photons over fiber optic,that might have viability many years into the future. Successful attacks on renowned targets such as Yahoo, LinkedIn so on and so forth show that no encryption is 100% bulletproof.Encryption keys may be transmissible or not, symmetrical or asymmetrical, public or private. However, presently, we find the best protection method to change the encryption key at a shorter time interval than the estimated time for hacking it.

V. CASE STUDY

Until recently it was considered to be inefficient to use compression algorithms after encryption, as the cipher meant a very well balanced distribution of characters, being almost statistically uniform, which for a compression algorithm is a dead-end, as it looks exactly for patterns and redundancy. A good compression resulted after an encryption process means actually the encryption is quite poor. The natural succession of these operations appears to be the following: compression and therefore redundancy reduction and afterwards, encryption, which involves finding the balance with the first operation. In 1999, 3 teachers in University of California, Berkeley [9] demonstrate that the two processes can be inversed by using the Slepian-Wolf theorem, while the compression algorithm doesn't use the same key in the decompression process. Entropy of the set plays a crucial role, by replacing the encryption key and the demonstration is permanently showing the Hamming distance between the codes. However, the compression is not lossless at all times and the method can be improved, as the code is open-source.

Based on the above we have started an investigation on our own, using a long-forgotten Huffman algorithm, trying to show the benefits of a custom-made compression algorithm in today's operating systems' environment.

Step 1: We have parsed the data set sequentially (symbol by symbol) and constructed the 0 order alphabet A^0, counting the number of appearances of each symbol $N(s)$ and sorted them in descending order (Equation 5):

$$A^0 = \{s_1, s_2, ..., s_n\}$$
$$N(s_1) \geq N(s_2) \geq ... \geq N(s_n)$$

Equation 5: Symbols Alphabet 0 Order

Step 2: (optional, but recommended) we reorder the alphabet elements to allow easier indexing of intermediary binary tree nodes (Equation 6):

$$A^0 = \{t_{n+k-1} = s_k\}_{k \in \overline{1,N}}$$

Equation 6: Symbols used for labelling the binary tree

Step 3: The construction of the binary tree is realized simultaneously with the construction of the new current alphabet. In order to obtain the new alphabet from the old, we apply the following rule: identification and elimination of each 2 symbols, having the smallest counters in the old alphabet, by replacing these with a virtual symbol which has its counter equal to the sum of the previous. If there are more than 2 "weak symbols" with the same counter, we chose the last two in alphabetically order. Therefore, we will not need another reorder of the new alphabet, the result being already sorted out. The cardinal drops a unit compared to the previous alphabet (Equation 7):

$$\begin{cases} A^{0,(N-k-1)} = \left[A^{0,(N-k)} \setminus \{t_p, t_q\}\right] \cup \{t_{N-k-1}\} \\ t_{N-k-1} = \text{'}\sqcup\text{'} \text{ (simbol virtual)} \\ N(t_{N-k-1}) = N(t_p) + N(t_q) \\ card(A^{0,(N-k-1)}) = card(A^{0,(N-k)}) - 1 = N - k - 1 \end{cases}$$

Equation 7: Binary tree construction

DOI: 10.21279/1454-864X-16-I1-027

Step 4: We continuously repeat the previous step in a loop for each current alphabet for N-1 times until the binary tree gets a root (t_1).
Step 5: We index all binary tree nodes with consecutive numbers (left to right), which is also an optional step, but very useful for the decompression process.
This is a classic 1952 algorithm which can be customized and improved. It is commonly used as a didactic method showing an improvement of Shannon-Fano algorithm [10]and while it performs a data compression, it also encodes data and can also encrypt it upon customization and user request. It can be considered an old-fashioned method of encryption, but for our case-study we have tried to use it comparing it to current software.
We have wrote a simple command prompt batch, by simplifying the algorithm up to its classic version, without tampering any of the process steps and got the results in the following table:

File	Initial	SF	Huffman static	Huffman adaptive	Win RAR
Fbitmap .bmp	586 kb	550 kb	542 kb	538 kb	212 kb
Text.doc	104 kb	80 kb	68 kb	60 kb	4 kb
Test.txt	30 bytes	39 bytes	79 bytes	28 bytes	97 bytes
Notepad.exe	50 kb	44 kb	36 kb	34 kb	21 kb

For the word document we have previously used the default password protect function of Microsoft Word, afterwards running our Huffman algorithm on the file and keeping the extension .doc.
We have tried to open the resulting file with the most well-known password cracking and achieving software such as: Brutus, Rainbow Crack, Passware, Wfuzz, Cain and Abel, Appnimi, WinZIP, WinRAR, etc. with absolutely no positive result in finding the "password" for our document, as the default structure of the document was obviously altered. It was also no longer recognized by Microsoft Word, showing the error in Fig. 3:

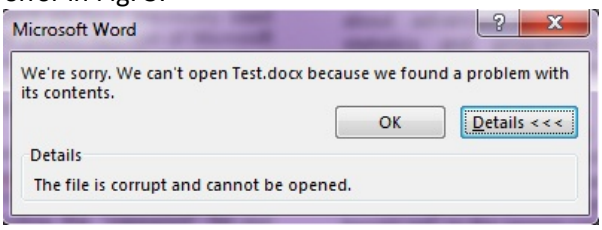

Fig. 3 Screenshot opening word document

The resulting file is in fact an archive with poor compression rate properties, but also a custom-made encryption which basically secures the file. It also deceits any attacker by putting any extension to the file-name, which inserts false-tracks for password cracking software as well.
If finding a password for a word file nowadays is done by cloud computing and tremendous processing power, we have truly found a deceptive way to encrypt our sensitive data for probably 99% of the average attackers. While the compression performances are not as good as well-known WinRAR software, our succession of bits in the data set using old Huffman adaptive algorithm was somehow impenetrable, much like MD5 hash function used for checking correct data transmission in the past. Designed for a totally different purpose, the MD5 was used for encryption for years.

CONCLUSIONS

Considering the time period of about 70-80 years since notable results were recorded in the domains of data compression and data encryption, we can notice the existence of multiple "old algorithms" with numerous variations. If a person doesn't know beforehand which algorithm we have used, nor the initial file type and its extension (decryption methods generally being available for specific types of data: text, pictures, audio/video streams), we can therefore state the chance of discovering the information that really is behind our "compressed" file is quite limited. There is a great number of possible algorithms, with infinite number of variations and customization possible. We are not talking only about advanced knowledge of mathematics, statistics and programming, but also huge processing power needs and impossible brute-force attacks, as this is clearly a dead-end scenario.
By altering the classical design of an old algorithm, usually by different symbol encoding or different dictionary, we can say that we are introducing a new "encryption key", which is known only to the person creating the batch file.
We have successfully decompressed the files, being able to open the word versions afterwards, by entering the password (which is an additional, modern step of encryption).
We can also think of a pattern for even and odd binary tree nodes, with different encoding. The possibilities are infinite. The result is obtained with great ease and if the disk space is not an issue, then the benefit of this encoding method is quite interesting in nowadays modern software environment.
We are aware that our method is not infallible, highly-specialized cryptographers could easily break our code, however, due to the personal, home-use purpose of our case study, we can definitely state that using such

DOI: 10.21279/1454-864X-16-I1-027

algorithms provides more than enough protection against modern hackers and specialized software. These algorithms are too "old", but especially because of their age, they can pose real difficulties in determining the true nature of the hidden data.

BIBLIOGRAPHY

[1] Shannon Fano Coding. Wikipedia.org. [Online]
https://en.wikipedia.org/wiki/Shannon%E2%80%93Fano_coding.
[2] Wikipedia Adaptive Huffman Coding. Wikipedia.org. [Online]
https://en.wikipedia.org/wiki/Adaptive_Huffman_coding.
[3] Disambiguation. Daniel Miessler. [Online]
https://danielmiessler.com/study/encoding-encryption-hashing-obfuscation/.
[4] Dobrescu, Radu and Kevorchian, S.,*"Compresia Datelor",* Editura Academiei Romane, Bucuresti, 2002.
[5] NIST. Federal Information Processing Standards Publication 197, National Institute of Standards and Technology, FIPS PUB 197, Washington DC, 2011.
[6] Boxcryptor. Boxcryptor. [Online] BoxCryptor.
https://www.boxcryptor.com/en/encryption.
[7] AES-Rijndael encryption algorithm. Code Planet. [Online]
http://www.codeplanet.eu/files/flash/Rijndael_Animation_v4_eng.swf.
[8] Ahmed, Mohiuddin, Sazzad, Shahriar T. M. and Mollah, Elias,*"Cryptography and State-of-the-art Techniques"*,IJCSI International Journal of Computer Science Issues, Vol. 9. 3, Bangladesh, 2012.
[9] Johnson, Mark, Wagner, David and Ramchandran, Kannan, *"On Compressing Encrypted Data without the Encryption Key"*, University of California, Berkeley, 1991.
[10] Stefanoiu, Dan,*"Compresia Datelor",* Printech, Bucuresti, 2003.

DOI: 10.21279/1454-864X-16-I1-027