Lightweight compression with encryption based on Asymmetric Numeral Systems

Jarek Duda
Jagiellonian University
Golebia 24, 31-007 Krakow, Poland
Email: *dudajar@gmail.com*

Marcin Niemiec
AGH University of Science and Technology
Mickiewicza 30, 30-059 Krakow, Poland
Email: *niemiec@kt.agh.edu.pl*

***Abstract*—Data compression combined with effective encryption is a common requirement of data storage and transmission. Low cost of these operations is often a high priority in order to increase transmission speed and reduce power usage. This requirement is crucial for battery-powered devices with limited resources, such as autonomous remote sensors or implants. Well-known and popular encryption techniques are frequently too expensive. This problem is on the increase as machine-to-machine communication and the Internet of Things are becoming a reality. Therefore, there is growing demand for finding trade-offs between security, cost and performance in lightweight cryptography. This article discusses Asymmetric Numeral Systems – an innovative approach to entropy coding which can be used for compression with encryption. It provides compression ratio comparable with arithmetic coding at similar speed as Huffman coding, hence, this coding is starting to replace them in new compressors. Additionally, by perturbing its coding tables, the Asymmetric Numeral System makes it possible to simultaneously encrypt the encoded message at nearly no additional cost. The article introduces this approach and analyzes its security level. The basic application is reducing the number of rounds of some cipher used on ANS-compressed data, or completely removing additional encryption layer if reaching a satisfactory protection level.**

	Enwiki8 100,000,000B	encode time [ns/byte]	decode time [ns/byte]
ZSTD 0.6.0 –22 --ultra	25,405,601	701	2.2
Brotli (Google Feb 2016) –q11 w24	25,764,698	3400	5.9
LZA 0.82b –mx9 –b7 –h7	26,396,613	449	9.7
lzturbo 1.2 –39 –b24	26,915,461	582	2.8
WinRAR 5.00 –ma5 –m5	27,835,431	1004	31
WinRAR 5.00 –ma5 –m2	29,758,785	228	30
lzturbo 1.2 –32	30,979,376	19	2.7
zhuff 0.97 –c2	34,907,478	24	3.5
gzip 1.3.5 –9	36,445,248	101	17
pkzip 2.0.4 –ex	36,556,552	171	50
WinRAR 5.00 –ma5 –m1	40,565,268	54	31
ZSTD 0.4.2 –1	40,799,603	7.1	3.6

Figure 1. Comparison of some well known compressors based on Huffman coding (marked red) with those using ANS (marked green) from [5] benchmark. ZSTD, lzturbo and zhuff use the tANS variant, which allows to add encryption by perturbing coding tables.

I. Introduction

Reliable and efficient data transmission is a crucial aim of communications. Modern telecommunication systems are facing a new challenge: security. Usually, data confidentiality is implemented by additional services, which are able to protect sensitive data against disclosure. Unfortunately, cryptographic algorithms decrease performance. Moreover, it is impossible to implement security services in many systems with limited resources (e.g., the Internet of Things). Therefore, system architects must find other ways to ensure data protection. One such possibility is integration of encryption with other data processing steps, such as source coding.

Prefix codes, such as the well-known Huffman coding [1], Golomb, Elias, unary and many others, are the basis of data storage and transmission due to their low cost. They directly translate a symbol into a bit sequence. As the symbol of probability p generally contains $\lg(1/p)$ bits of information ($\lg \equiv \log_2$), prefix codes are perfect for probabilities with a power of $1/2$. However, this assumption is rarely true in practice. While encoding a sequence of $\{p_s\}$ probability distribution with a coding optimal for $\{q_s\}$ distribution, we use asymptotically $\Delta H = \sum_s p_s \lg(p_s/q_s)$ more bits/symbol than required. This cost of inaccuracy is especially significant for highly probable symbols. They can carry nearly 0 bit/symbol of information, while prefix codes have to use at least 1 bit/symbol.

Arithmetic and range coding ([2], [3]) avoid this cost by operating on nearly accurate probabilities. However, they are more costly and usually require multiplication, which is an operation with a high computational complexity. Using lookup tables to avoid multiplication is achieved for example by CABAC [4] in H.264, H.265 video compressors. However, it operates on the binary alphabet, requiring eight steps to process a byte.

Recently, a new multiplication-free large alphabet entropy coder was proposed for low cost systems: Asymmetric Numeral Systems (ANS) ([6], [7], [8]). In

contrast to prefix codes, this coding uses nearly accurate probabilities for coded symbols. The high performance and efficiency of ANS is leading to Huffman and Range being replaced in new compressors ([9], [10], [11]), including Apple LZFSE [12] and Facebook Zstandard [13] to improve performance. Figure 1 presents a comparison of well known compressors based on Huffman coding with compressors using the ANS algorithm. It shows that this new entropy coder allows for compressors which are many times faster both decoding and encoding for comparable compression ratios. One of the reasons is that, while Huffman coding requires costly sorting of symbols to build the prefix tree, ANS initialization is cheap: with linear time and memory cost. This advantage is especially important for the cost of hardware implementations, which improvements have been already demonstrated for FPGA [14].

As well as providing effective data compression, another basic requirement of data storage and transmission is confidentiality. We are able to ensure data confidentiality using symmetric ciphers (asymmetric cryptography is not an appropriate solution in environments with limited resources because of its high computational cost). However, popular symmetric ciphers such as the Advanced Encryption Standard (AES), turn out to be too costly for many applications, especially battery-powered, such as autonomous remote sensors or the Internet of Things. As such, there is a growing field of lightweight cryptography [15], [16], [17] – with a focus on low cost, at a trade-off for having lower protection requirements.

Since a combination of compression and encryption is a common requirement, the cost priority suggests a natural solution of combining these two steps. Many approaches were considered for adding encryption into methods which are already a part of data compressors: Lempel-Ziv substitution schemes [18], [19], Burrows-Wheeler transform [20] and arithmetic coding [21], [22]. These articles contain some general techniques, which addition might be considered to improve security of discussed ANS-based encryption.

Huffman coding has also been discussed for adding simultaneous encryption [23]. An abstract of an article by Ronald Rivest et al. [24] concludes that: *"We find that a Huffman code can be surprisingly difficult to cryptanalyze"*. The main problem is the lack of synchronization – the attacker does not know how to split the bit sequence into blocks corresponding to symbols. Additionally, data compression offers auxiliary protection by reducing redundancy which could be used for cryptanalysis.

A Huffman decoder can be viewed as a special case of the tabled variant of an ANS decoder, referred as tANS [7]. This generalization allows for more complex behavior and other features, which suggest that secure encryption could be included inside the entropy coding process. While the prefix code is a set of rules: "symbol $\rightarrow$ bit sequence", tANS also has a hidden internal state $x \in \{2^R, ..., 2^{R+1}-1\}$ for some $R \in \mathbb{N}$, which acts as a buffer containing $\lg(x) \in [R, R+1)$ bits of information. The transition rules have the form

$$(\text{symbol, state}) \quad \rightarrow \quad (\text{bit sequence, new state})$$

Therefore, in comparison with Huffman coding, there is an additional hidden variable x, which controls the bit sequence to produce, including the number of produced bits in this step: floor or ceiling of $\lg(1/p)$. As chaos is seen as strongly linked to the security of cryptography [25], [26], the authors discuss three sources of chaos in evolution of this internal state, making its behavior virtually unpredictable while incomplete knowledge.

As only a few ciphers like one-time pad can be formally proven to be safe, practical encryption schemes often require time to gain trust as being secure: by lack of successful attacks. Hence, while there are some arguments of strength of the proposed encryption scheme, until gaining such trust it is suggested to be used together with a convenient cipher like AES, for example with a reduced number of rounds. Comparing Huffman-based compression plus 10 round of AES, with tANS-based compression+encryption plus 5 rounds of AES, we get gain in both compression ratio and performance.

The remainder of the paper proceeds as follows. Section II introduces the ANS algorithm: coding and decoding as well as some examples of these steps. Section III, presents the basic concept of including encryption in tANS, and properties influencing security level: set of cryptographic keys, chaotic behavior, etc. Section IV describes the security features of this encryption method. Finally, Section V concludes the paper.

II. Asymmetric Numeral Systems (ANS)

This section introduce ANS, focusing on the tabled variant (tANS). Further discussion and other variants of ANS can be found in [7].

A. Coding into a large natural number

Let us first consider the standard binary numeral system. It allows to encode a finite sequence of symbols from the binary alphabet ($s_i \in \mathcal{A} = \{0, 1\}$) into $x = \sum_{i=0}^{n-i} s_i 2^i \in \mathbb{N}$. This number can be finally written as length $\approx \lg(x)$ bit sequence. This length does not depend on exact values of symbols – this approach is optimized for $\Pr(0) = \Pr(1) = 1/2$ symmetric case, when both symbols carry 1 bit of information. In contrast, a $\{p_s\}$

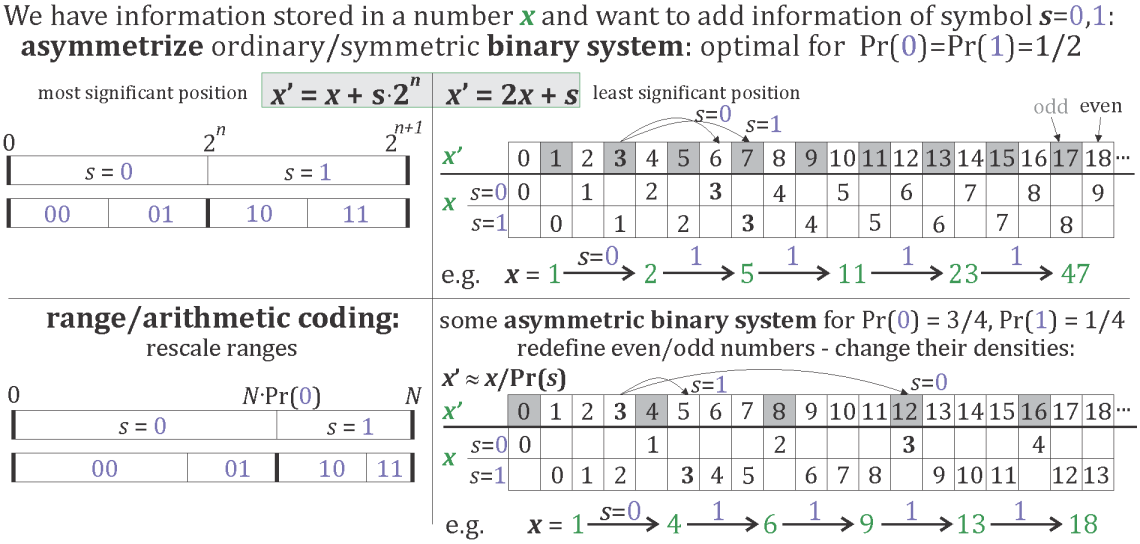

Figure 2. Arithmetic coding (left) and ANS (right) seen as an asymmetrization of the standard numeral system - in order to optimize them for storing symbols from a general probability distribution.

probability distribution symbol sequence carries asymptotically $\sum_s p_s \lg(1/p_s)$ bits/symbol (Shannon entropy), and a general symbol of probability p carries $\lg(1/p)$ bits of information. Hence, to add information stored in a natural number x to information from a symbol of probability p, the total amount of information will be $\approx \lg(x) + \lg(1/p) = \lg(x/p)$ bits of information. This means that the new number $x' \in \mathbb{N}$ containing both information should be approximately $x' \approx x/p$, which is the basic concept of ANS.

Having a symbol sequence encoded as $x = \sum_{i=0}^{n-i} s_i 2^i$, we can add information from a new symbol $s \in \mathcal{A}$ in two positions: the most or the least significant. The former means that the new number containing both information is $x' = x + s2^n$. The symbol s chooses between two large ranges for x': $\{0, ..., 2^n - 1\}$ and $\{2^n, ..., 2^{n+1} - 1\}$. The symmetry of their lengths corresponds to the symmetry of informational content of both symbols. As depicted in the left panel of Figure 2, arithmetic or range coding can be viewed as an asymmetrization of this approach to make it optimal for different probability distributions. They require operating on two numbers, defining the currently considered range, what is analogues to the need to remember the current position n in the standard numeral system.

We can avoid this inconvenience by adding a new symbol in the least significant position: $C(s, x) = x' = 2x + s$. Old digits are shifted one position up. To reverse this process, the decoding function is $D(x') = (s, x) = (\mathrm{mod}\ (x', 2), \lfloor x'/2 \rfloor)$. This approach can be viewed that x' is x-th appearance of an even $(s = 0)$ or odd $(s = 1)$ number. We can use this rule to asymmetrize this approach to make it optimal for a different probability distribution. In order to achieve this, we need to redefine the division of natural numbers into even and odd numbers, such that they are still distributed uniformly, but with the density corresponding to the assumed probability distribution. More formally, for a probability distribution $\{p_s\}$ we need to define a *symbol distribution* $\overline{s} : \mathbb{N} \to \mathcal{A}$, such that: $|\{0 \le x < x' : \overline{s}(x) = s\}| \approx x' p_s$. Then the encoding rule is

$$x' = C(s, x) \text{ is } x\text{-th appearance of symbol } s.$$

and correspondingly for the decoding function D, such that $D(C(s, x)) = (s, x)$. The decoded symbol is $\overline{s}(x')$ and x is the number of appearance of this symbol. More formally:

$$C(s, x) = x' : \overline{s}(x') = s,\ |\{0 \le y < x' : \overline{s}(y) = s\}| = x$$

$$D(x') = (\overline{s}(x'),\ |\{0 \le y < x' :\ \overline{s}(y) = \overline{s}(x')\}|)$$

The right panel of Figure 2 depicts an example of such a process for the $\Pr(0) = 3/4,\ \Pr(1) = 1/4$ probability distribution. Starting with $x = 1$ symbol/state, we encode successive symbols: 01111 into $x = 47$ or $x = 18$. Then we can can successively use the decoding function D to decode the symbol sequence in the reverse order. ANS results in a lower representation than the standard numeral system, since it better corresponds with the digit

distribution of the input sequence 01111.

There can be found arithmetic formulas using multiplication for such coding/decoding functions: uABS and rABS variants for the binary alphabet, and rANS variant for any large alphabet [7]. The range variant (rANS) can be viewed as a direct alternative to range coding with some better performance properties such as a single multiplication per symbol instead of two, leading to many times faster implementations [27]. However, since it requires multiplication and is not suited for encryption, this paper only discusses the tabled variant (tANS), in which we put the entire coding or decoding function for a range $x \in I$ into a table.

B. Streaming ANS via Renormalization

Using the C function multiple times allows us to encode a symbol sequence into a large number x. Working with such a large number would be highly demanding. In AC, renormalization is used to allow finite precision, an analogous approach should be used for ANS. Specifically, we enforce x to remain in a fixed range I by transferring the least significant bits to the stream (we could transfer a few at once, but this is not convenient for tANS). A basic scheme for the decoding/encoding step with included renormalization is:

Algorithm 1 ANS decoding step from state x

```
(s, x) = D(x)                          { the proper decoding function }
useSymbol(s)                           { use or store decoded symbol }
while x < L do
    x = 2·x + readBit()                {read bits until returning to I}
end while
```

Algorithm 2 ANS encoding of symbol s from state x

```
while x > maxX[s] do                   {maxX[s] will be found later}
    writeBit(mod(x, 2)); x = ⌊x/2⌋      {write youngest bits}
end while                              {until we can encode symbol}
x = C(s, x)                            { the proper encoding function }
```

To ensure that these steps are the inverse of each other, we need to make sure that the loops for writing and reading digits end up with the same values. For this purpose, let us observe that if a range has the form $I = \{L, \ldots, 2L-1\}$, when removing $(x \to \lfloor x/2 \rfloor)$ or adding $(x \to 2x + d)$ the least significant bits, there is exactly one way of achieving range I. For uniqueness of the loop in Method 1, we need to use I range of this type: $I = \{L, \ldots, 2L-1\}$ where for practical reasons we will use $L = 2^R$. For uniqueness of the loop in Method 2 we need to additionally assume that

$$I_s = \{x : C(s,x) \in I\} \qquad \left(I = \bigcup_s C(s, I_s)\right)$$

are also of this form: $I_s = \{L_s, \ldots, 2L_s - 1\}$ and therefore $maxX[s] = 2L_s - 1$ which is used in Method 2.

C. Tabled variant (tANS)

In the tabled variant (tANS), which is used in most of compressors in Fig. 1 and is interesting for cryptographic purposes, we put the entire behavior into a lookup table. Let us start with the following example: we construct an $L = 4$ state automaton optimized for the $\Pr(a) = 3/4$, $\Pr(b) = 1/4$ binary alphabet, depicted in Figure 3. We need to choose a symbol distribution $\overline{s} : I \to \{a, b\}$ for $I = \{4, 5, 6, 7\}$. To correspond to the probability distribution, the number of symbol appearances should be chosen as $L_a = 3$, $L_b = 1$. There now remain four options to choose the $\overline{s}$ function. Let us focus on the choice $\overline{s}(5) = b$, $\overline{s}(4) = \overline{s}(6) = \overline{s}(7) = a$, or in other words: "abaa" symbol spread. We need to enumerate the appearances using the numbers $I_a = \{3, 4, 5\}$, $I_b = \{1\}$, getting the decoding function $D(4) = (a, 3)$, $D(5) = (b, 1)$, $D(6) = (a, 4)$, $D(7) = (a, 5)$. It allows us to obtain the decoded symbol and a new state. However, some of these states are below the I range, therefore we need to apply renormalization by reading some youngest bits to return to I range. For example for $x = 5$, the decoding function takes us to $x = 1$, so we need to read two bits from the stream (d_1, d_2) to return to I, leading to state $x = 4 + 2d_2 + d_1$.

Assuming the input source is i.i.d. $\Pr(a) = 3/4$, $\Pr(b) = 1/4$, we can find the probability distribution of the visiting states of such an automaton: ρ_x in this figure. It allows us to find the expected number of bits/symbol: $H' \approx 1 \cdot 1/4 \cdot 2 + (0.241 + 0.188) \cdot 3/4 \cdot 1 \approx H + 0.01$ bits/symbol, where $H = \sum_s p_s \lg(1/p_s)$ is the minimal value (Shannon entropy). Generally, as discussed in [7], $\Delta H = H' - H$ behave approximately like m^2/L^2, where m is the size of the alphabet.

a) Connection with prefix codes: Using lookup tables, the decoding procedure can be written as:

Algorithm 3 Decoding step for prefix codes and tANS

```
t = decodingTable[X]          {X ∈ {0, .., 2^R − 1} is current state}

useSymbol(t.symbol)           { use or store decoded symbol }
X = t.newX + readBits(t.nbBits)   { state transition }
```

where $X = x - L \in \{0, .., 2^R - 1\}$ is a more convenient representation. It should be noted that this method can also be used for decoding prefix codes such as Huffman coding. In this case R should be chosen as the maximal length of the bit sequence corresponding to a symbol. The state X should be viewed as a buffer containing the last R bits to process. It directly determines

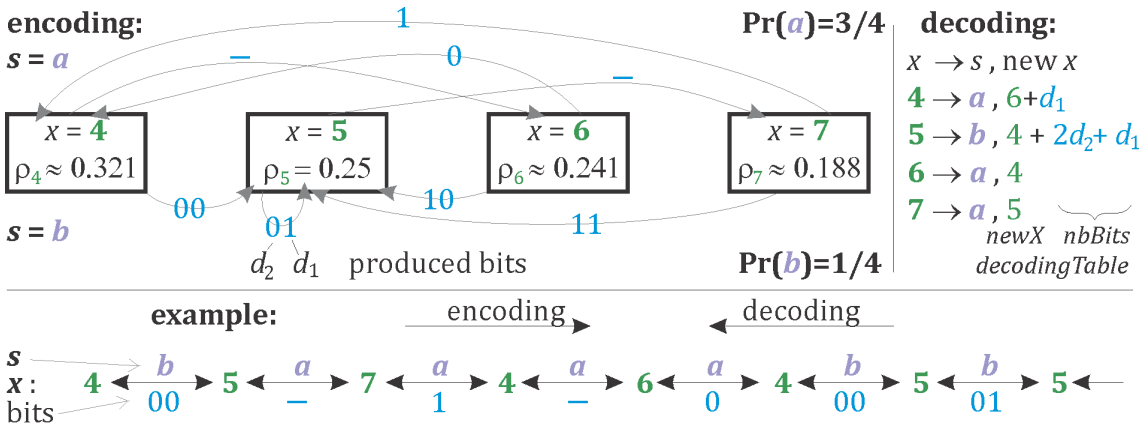

Figure 3. Example of 4 state tANS (top) and its application for stream coding (bottom). State/buffer x contains $\lg(x) \in [2,3)$ bits of information. Symbol b carries 2 bits of information, while a carries less than 1 - its information is gathered in x until it accumulates to a complete bit of information. ρ_x are probabilities of visiting state x assuming the i.i.d. input source.

the symbol, which uses $nbBits \leq R$ bits of the buffer. The remaining bits should be shifted and $nbBits$ should be read from the stream to refill the buffer:

$$decodingTable[X].newX = (X << nbBits)\ \&\ mask$$

where $<<$ denotes left bit-shift operation and $\&mask$ denotes restriction to the least significant R bits.

Just shifting the unused bits corresponds to assuming that the produced symbol carried indeed $nbBits$ bits of information: has $2^{-nbBits}$ probability. tANS works on fractional amounts of bits by not only shifting the unused bits, but also modifying them according to the fractional amount of bits of information.

It should be noted that if we choose $L_s = 2^{r_s}$ for symbol of probability $\approx 2^{r_s - R}$, and spread symbols in ranges, our tANS decoder would become a decoder for a prefix code. For example, "aaaabcdd" symbol spread would lead to decoder for $a \to 0$, $b \to 100$, $c \to 101$, $d \to 11$ prefix code. Therefore, prefix codes can be regarded as a degenerated case of tANS.

D. Algorithms (tANS)

对区间[8,15]而言，Ia={6,7,8,9,10,11}，其中Ls=6

Let us now formulate the algorithms. Assume that $L = 2^R$: $I = \{L, \ldots, 2L-1\}$, $I_s = \{L_s, \ldots, 2L_s - 1\}$ and that $q_s := L_s/L \approx p_s$ approximates the probability distribution of the symbols. There are $|I| = 2^R$ positions for spreading symbols with $|\{x \in I : \overline{s}(x) = s\}| = L_s$ appearances of symbol s. For convenient table handling, we use $X := x - L \in \{0, \ldots, 2^R - 1\}$ and store the symbol spread as $symbol[X] \equiv \overline{s}(X + L)$ size L table.

Method 4 generates the $decodingTable$ for efficient decoding step from Method 3. For efficient memory handling while encoding step, the encoding table can be stored in one dimensional form $C(s,x) = encodingTable[x + start[s]] \in I$ for $x \in I_s$, where $start[s] = -L_s + \sum_{s'<s} L_{s'}$. To encode symbol s from state x, we first need to transfer $k[s] - 1$ or $k[s]$ bits, where $k[s] = \lceil lg(L/L_s) \rceil$. This choice can be simplified to $nbBits = (x + nb[s]) >> r$ using a prepared table $nb[]$. Finally, the preparation and encoding step are written as Methods 5 and 6 respectively.

Algorithm 4 Generating tANS $decodingTable$

Require: $next[s] = L_s$ {number of next appearance of symbol s}

for $X = 0$ to $L-1$ **do**
  $t.symbol = symbol[X]$ { symbol is from spread function }
  $x = next[t.symbol]++$ { $D(X+L) = (symbol, x)$ }
  $t.nbBits = R - \lfloor \lg(x) \rfloor$ { number of bits to return to I }
  $t.newX = (x << t.nbBits) - L$ { properly shift x }
  $decodingTable[X] = t$
end for

++表示某行中的这个状态值是不断增加的，这与实际一致

通过枚举x=0,1,2,3,4,5...可以很明显的看出来，x 自身占用了（logx个向下取整+1）个比特。
为了移动到[L,2L-1]这个段内，必须r-v=(R+1)-logx个向下取整+1），证毕。

Algorithm 5 Preparation for tANS encoding, $L = 2^R$, $r = R + 1$

Require: $k[s] = R - \lfloor \lg(L_s) \rfloor$ $\{nbBits = k[s]$ or $k[s] - 1\}$
Require: $nb[s] = (k[s] << r) - (L_s << k[s])$
Require: $start[s] = -L_s + \sum_{s'<s} L_{s'}$
Require: $next[s] = L_s$
for $x = L$ to $2L-1$ **do**
  $s = symbol[x - L]$
  $encodingTable[start[s] + (next[s]++)] = x;$
end for

将来会x-Ls+(L0+L1+...+Ls-1)这样用。其中x-Ls是在符号 s 对应的小段内的偏移量。后面的多项和表示编码表内部的之前的所有小段的总长度。

Symbol Spread Function: We need to choose $symbol[X] = \overline{s}(X + L)$ distributing symbols over the I range: L_s appearances of symbol s. Finding the optimal way seems is a difficult problem. We present only a fast


```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15   数组下标
-------------------------------------
0   1 2 3   4 5 6   7   8   9     10 11   输入字符 a
  0       1       2           3          输入字符 b
-------------------------------------
```

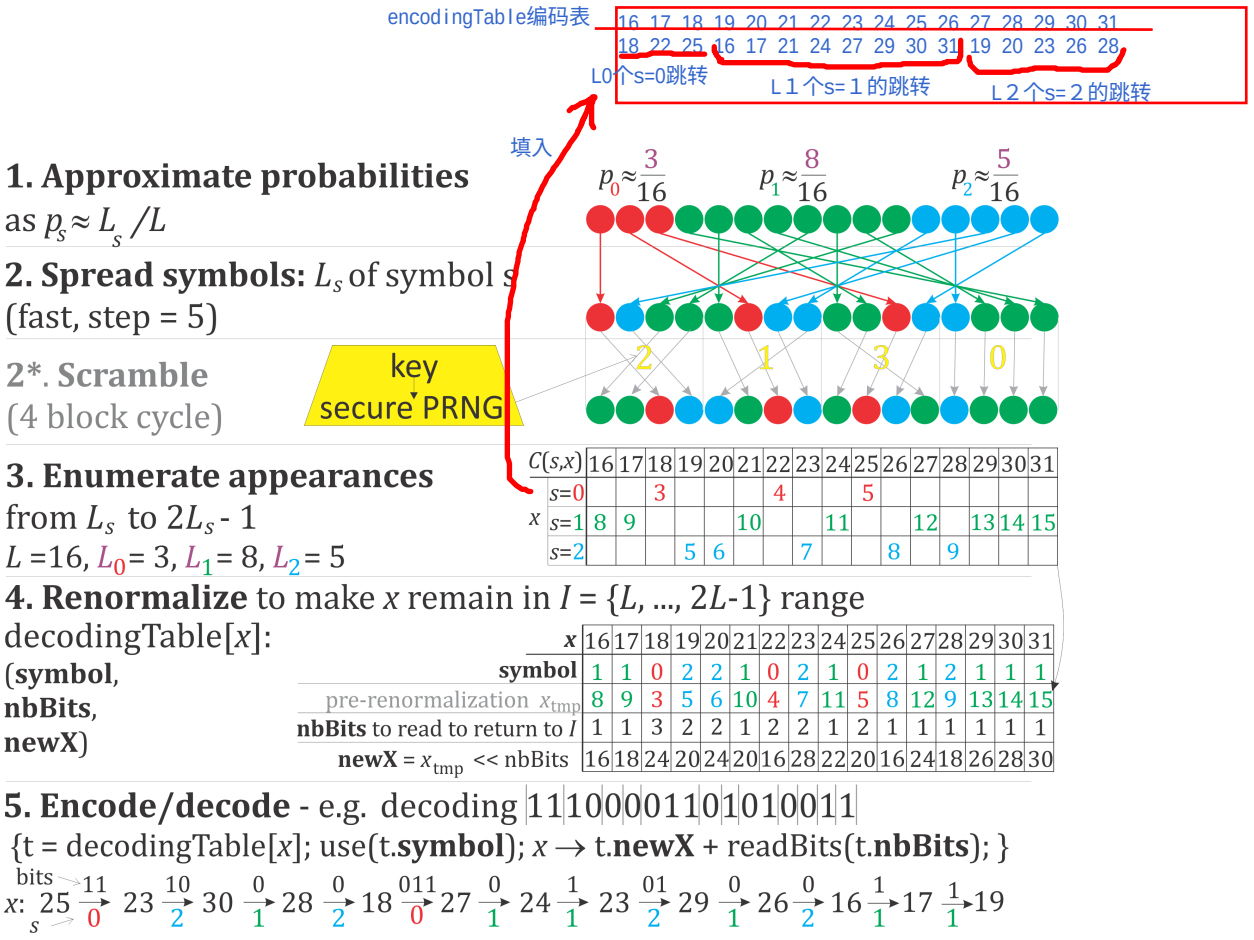

Figure 4. Example of generation of tANS tables and applying them for stream decoding for $m = 3$ size alphabet and $L = 16$ states.

x>>nbBits可以定位到编码表中的某个输入符号所在的状态值，因此(x>>nbBits - Ls)的值其实就是x状态在编码表内符号s对应的小段内的偏移量，这个偏移量再加上之前的各个小段的长度，就可以得到相对于整个编码表开始位置的偏移量，从而查表后可得到编码后的目标状态。
例如23>>2得到的是5，它的输入符号是s=0，因此在第一个小段内的偏移量是2，encodingTable[2]刚好就是25．再例如29状态输入符号2时，29>>2得到7，7−L2+L0+L1 = 7-5+3+8 = 13，因此查询编码表encodingTable[13]，得到13,也符合。

Algorithm 6 tANS encoding step for symbol s and state $x = X + L$

$nbBits = (x + nb[s]) >> r$ {$r = R + 1,\ 2^r = 2L$}
useBits($x, nbBits$) {use $nbBits$ of the youngest bits of x}
$x = encodingTable[start[s] + (x >> nbBits)]$

simple way of spreading symbols in a pseudorandom way in Method 7, which already offers excellent performance. Several symbol spreads can be found and tested in [28].

Algorithm 7 Example of fast symbol spread function [29]

$X = 0$; $step = 5/8L + 3$ { some initial position and step}
for $s = 0$ to $m - 1$ **do**
  for $i = 1$ to L_s **do**
    $symbol[X] = s$; X =mod$(X + step, L)$
  end for
end for

需求：显然各个符号的分布需要每次出现的X位置都不同，然后逐步填充满[0, L-1]或者[L, 2*L-1]这个目标段
做法：
（1）当n为偶数时，例如n＝0，step=n+1这个步长step=1,显然满足条件；n=2时，也显然满足条件，依次类推，n为偶数时step=n+1满足条件；进一步推导n+(2+2+...+2)+1也满足条件,不失一般性n+n+1也满足条件，即2n+1满足条件；继续推导4*n+n+1也满足条件，最后得到4*n+n+2+1应该也满足条件；因此step=(5/8)*L+3就相当于5*(L/8)+3，当L=16时，L/8是偶数，就相当于5*n+3，符合填充需求；
（2）当n位奇数时，例如n=1，则step=n应该满足条件，也可以进行类似的分析。

III. Adding encryption

The construction of tANS code gives us an opportunity to ensure data confidentiality. The concept of encryption in tANS coder is described in this section.

A. Basic concept

We could use the freedom of choosing the exact coding for encryption purposes. For example while building prefix tree for a size m alphabet, there are $m-1$ internal nodes. Switching their left and right children gives us 2^{m-1} options of encoding our message.

As discussed, prefix codes can be viewed as a tANS for L_s being powers of 2 and symbols spread in ranges. Without this restriction, there are much more options of choosing the $\bar{s}$ function:

$$\binom{L}{L_1, \ldots, L_m} \approx 2^{L \cdot H(L_1/L, \ldots, L_m/L)}$$

where $H(p_1, \ldots, p_m) = \sum_i p_i \lg(1/p_i)$ is entropy.

Each option defines a different coding. Therefore we need a method of spreading symbols according to the cryptographic key. One way is first to use an independent

method, e.g. put successive symbol every $step$ number of positions (cyclically). Then we can perturb the obtained symbol spread using a cryptographically-secure pseudo-random number generator (CSPRNG) seeded with the key, for example by taking blocks and cyclically shifting symbols inside such blocks by a shift from the CSPRNG.

Figure 4 depicts an example of coding and encryption processes for the following parameters: $L = 16$, $m = 3$ size alphabet, $step = 5$ and size $B = 4$ blocks. After step 2, where we spread all symbols (globally), the scrambling process in blocks is performed (locally). This is crucial from the security point of view, since a different locations of symbols results in different forms of encoded messages. The encoded messages depend strongly on the CSPRNG key.

B. Numbers of possibilities

The key space is a crucial element for protecting the secure cipher against brute-force attacks, therefore we analyze the number of ways of encoding messages.

As default parameters (DP), we consider $L = 2048$ states, $m = 256$ size alphabet and $B = 8$ blocks, which requires 8kB of lookup tables (or 6kB with simple bit compression). As degenerated default parameters (DDP), we consider the worst case scenario: when there is one dominating symbol and the remaining ones have the minimal $L_s = 1$ number of appearances.

The number of different symbol spreads for DP is 2^{2048H} and depends on the entropy of the sequence. We can use DDP to find the lower bound: the number of symbol spreads here is $\frac{L!}{(L-m+1)!} \approx 1.65 \cdot 10^{837}$ for $L = 2048,\ m = 256$.

The assumed perturbation using cyclic shifts by values from PRNG reduces these numbers. For DP, this number is $B^{L/B} = 8^{256} \approx 1.55 \cdot 10^{231}$. Some cyclic shifts of such blocks may accidentally lead to identical symbol alignment. The probability that two B length blocks from the i.i.d. $\{p_i\}$ probability distribution are accidentally equal is approximately $2^{-BH(p_1,\ldots,p_m)}$. Therefore, for practical scenarios (e.g. $m = 256$, $H > 1$), the reduction of space of possibilities is practically negligible. For the DDP case, approximately $\left(\frac{L-m+1}{L}\right)^B \approx 0.345$ of blocks have the dominating symbol only. The remaining ones are always changed by the perturbation: the number of possibilities is $\approx B^{(1-0.345)B/L} \approx 2.49 \cdot 10^{151}$.

C. Chaotic state behavior

Having a large number of possible codings is not sufficient; strong dependence on the key is also required. One source is relying on the security of CSPRNG, which ensures that changing a single bit in the key produces a completely independent perturbation of the symbol spread. Additionally, eventual inferring the coding function would give no information about the key (seed).

Figure 5. Three sources of chaotic behavior of the internal state x: $\lg(x) \to^{\approx} \lg(x) + \lg(1/p) \mod 1$.

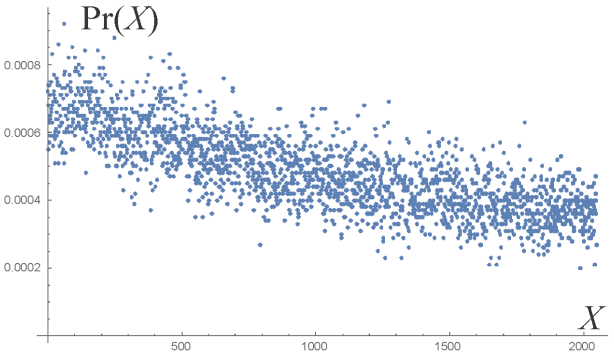

Figure 6. Example of probability distribution of $X = x - L$ in $L = 2048$ and $m = 256$ case and fast symbol spread.

Another source is chaos of dynamics of the internal state x, ensuring that incomplete knowledge leads to a rapid loss of any information about the state of the coder. State x can be viewed as a buffer containing $\lg(x)$ bits of information, and adding a symbol of probability s increases it by $\lg(1/p)$ bits. Due to renormalization, this addition is modulo 1 - accumulated complete bits are send to the stream. Finally the approximate behavior is $\lg(x) \to^{\approx} \lg(x) + \lg(1/p) \mod 1$.

This cyclic addition formula contains three sources of chosity as depicted in Figure 5:

- asymmetry: each position may correspond to a different symbol and so to a different shift,
- ergodicity: $\lg(1/p)$ is usually irrational, so even a single symbol tends to cover the range uniformly,
- diffusivity: this formula is approximate, so even knowing the symbol sequence, information about the exact position is quickly lost.

These properties suggest that we should expect an approximately uniform probability distribution of $lg(x)$, which corresponds to $\Pr(x) \propto 1/x$ distribution of x. Better symbol spreads are close to this behavior. For the discussed fast symbol spread, the noise around this $1/x$ curve can be high, as shown in Figure 6.

IV. Features and limitations

The presented concept of lightweight compression with encryption should be verified from the security point of view. In this section, we discuss results of standard cryptographic tests of tANS encoding with encryption as well as presenting ways to of enhancing the security of this solution. The tests were mainly performed for the DP case: $L = 2048$, $m = 256$ for pure tANS layer - imperfections can be easily removed for example by a reduced number of layers of AES.

A. Balancing

The first question regarding the statistics of the produced bit sequence is the density of "0" and "1". Are they equal? In general, for ANS algorithm it is not exactly fulfilled. This is due to the fact that the probability distribution of used states x prefers lower states: approximately $\Pr(x) \propto 1/x$, such as in Figure 6. For the DP case, tests show approximately 0.001 difference ($\Pr(0) \approx 0.501$), it has never exceeded 0.002. For different parameters, an approximate general behavior of this difference is that it is proportional to m/L.

For higher correlations, the probability of a length k bit sequence in the produced stream should be 2^{-k}. Beside the above difference, our tests could not detect further disagreements with this rule.

The variable-length nature of ANS makes the $\Pr(0) \approx 0.501$ issue unlikely to be useful for cryptanalysis (due to the lack of synchronization). Additionally, this small imbalance can be easily removed by adding an inexpensive additional operation, such as XOR with a mask (or set of masks) having equal number of zeros and ones.

B. Avalanche and nonlinearity

One crucial feature of the secure cipher is the Strict Avalanche Criterion (SAC), which is satisfied if a change of a single bit of the key results, on average, in a change of one half of bits of ciphertext. The tANS approach uses CSPRNG which has a similar property: changing a single bit of the seed leads to a statistically independent random stream, which means an independent tANS coding table. We tested a property which is even stronger than SAC: by encoding the same symbol sequence using the same coding tables, but starting with a different initial state. We were not able to detect statistical dependencies between such two streams.

Additionally, we verified the nonlinearity of the encryption process (defined as the Hamming distance to the nearest affine function). The tests confirmed the nonlinear behavior of encryption process.

C. Diffusion and completeness

The next important feature is the diffusion of changes during the encryption process. We verified that even when the number of changes in the entry were low, the change of the output bits was high.

The same behavior was observed during the tests of completeness. Completeness is satisfied when a change of a single bit of the plaintext causes a change of around one half of bits of ciphertext. The discussed method processes successive single symbols, so a change of a symbol can influence only bits corresponding to the current and successive positions. We have performed tests with two encoding streams starting with the same state x. We first encode a single symbol different for each streams, followed by a sequence of symbols identical for both streams. Encoding a symbol of probability p produces the youngest $\lfloor \lg(1/p) \rfloor$ or $\lceil \lg(1/p) \rceil$ bits of x. This means that the first few bits after the change of symbol will be identical – their number depends on the probability of symbol. Tests of further bits were not able to find statistical dependencies.

Operating on short bit blocks (of varying length) leaves an option for adaptive attacks by exploring ciphertexts differing by single symbols. To protect against this, the initial state can be chosen entirely randomly. We can use such an initial state by analogy to the initial vector in many modes of encryption, e.g. Cipher Block Chaining (CBC). This way, the same symbol sequences lead to independent bit sequences. As the number of the initial state may be insufficient, this property can be enhanced by adding a few random symbols at the beginning of plaintext.

We can enhance this protection by making sure that we use an independent coding table each time. This can be achieved by using what is referred as 'salt': a random number which affects the seed of CSPRNG and is stored in the header of a file. Additionally, the stream is usually divided into frames of e.g. 10kB size, what is common in data compression applications for updating probability distributions. For encryption purposes, new independent coding tables can be generated for each frame, using the number of frame also as a seed. Finally, we could use triple data: a cryptographic key, the number of the frame and a random number (salt) as the seed of CSPRNG.

Summarizing, the tests of features confirm that presented solution is able to protect confidentiality at a high level of security. We suggest the following principles:

- using a relatively large number of states and a large

alphabet (protecting against brute-force attacks),
- encrypting the final state, which is required for decoding,
- using a completely random initial state to protect against adaptive attacks (additionally, appending a few random symbols at the beginning, which are discarded by decoder, would strengthen this protection).

During the implementation of proposed solution, it is possible to strengthen the security level by:

- using three parameters: the cryptographic key, the number of the data frame (e.g. 10kB) and a random number stored in an encrypted file (salt) as a seed for CSPRNG to make all coding tables completely independent,
- using an inexpensive additional encryption layer, such as XOR with a set of masks (generated using CSPRNG), a simple substitution-permutation cipher, or AES with a reduced number of rounds.

Future work on proposed compression algorithm with encryption process should focus on advanced cryptanalysis and finding the optimal compromise between security and performance.

V. Conclusion

This paper proposes a new concept of compression with simultaneous encryption. From the data compression perspective, it provides a nearly optimal compression ratio (such as arithmetic coding) at an even lower cost than Huffman coding (due to having inexpensive linear initialization instead of the $n \log n$ cost of sorting in Huffman coding). Using CSPRNG initialized with a cryptographic key to choose the coding tables means the message encoded this way can be simultaneously encrypted at nearly no additional cost. The variable-length nature of this coding makes eventual cryptanalysis extremely difficult as the attacker does not know how to split the bit sequence into blocks corresponding to successive symbols. These blocks and even their lengths depend on the internal state of the coder, which is hidden from the attacker. The behavior of this state is chaotic, rapidly eliminating any incomplete knowledge of the attacker. Using CSPRNG ensures that even if an attacker would obtain the applied coding table, no information about the cryptographic key is acquired.

Such lightweight compression with encryption is crucial in many situations, for example in battery-powered remote sensors which should transmit the gathered data in a compressed and secure manner. We are entering the age of the Internet of Things, where the use of such types of devices will be widespread. The hundreds of potential applications of this solution include medical implants transmitting diagnostic data, smart RFIDs powered by electromagnetic impulses only, smartphones or smartwatches with improved performance and extended battery life, and many other situations for data storage and transmission.

References

[1] D. Huffman, "A method for the construction of minimum redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, September 1952.

[2] J. J. Rissanen, "Generalized kraft inequality and arithmetic coding," *IBM Journal of research and development*, vol. 20, no. 3, pp. 198–203, 1976.

[3] G. Martin, "Range encoding: an algorithm for removing redundancy from a digitized message," *Proceedings of Institution of Electronic and Radio Engineers International Conference on Video and Data Recording Conference*, July 1979, Southampton, England.

[4] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 620–636, July 2003.

[5] M. Mahoney, "Data Compression Programs website," http://mattmahoney.net/dc.

[6] J. Duda, "Asymmetric numerical systems," *arXiv:0902.0271*.

[7] ——, "Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding," *arXiv:1311.2540*.

[8] J. Duda, K. Tahboub, N. J. Gadgil, and E. J. Delp, "The use of asymmetric numeral systems as an accurate replacement for huffman coding," *31st Picture Coding Symposium*, 2015.

[9] Y. Collet, "Zhuff compressor," http://fastcompression.blogspot.com/p/zhuff.html.

[10] H. Buzidi, "lzturbo compressor," https://sites.google.com/site/powturbo/.

[11] N. Francesco, "Lza compressor," http://heartofcomp.altervista.org/.

[12] "Apple lzfse compressor," https://github.com/lzfse/lzfse.

[13] "Facebook zstandard compressor," https://github.com/facebook/zstd.

[14] S. M. Najmabadi, Z. Wang, Y. Baroud, and S. Simon, "High throughput hardware architectures for asymmetric numeral systems entropy coding," in *2015 9th International Symposium on Image and Signal Processing and Analysis (ISPA)*. IEEE, 2015, pp. 256–259.

[15] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A survey of lightweight-cryptography implementations," *IEEE Design & Test of Computers*, vol. 24, no. 6, pp. 522–533, 2007.

[16] A. Y. Poschmann, "Lightweight cryptography: cryptographic engineering for a pervasive world," in *Ph. D. Thesis*. Citeseer, 2009.

[17] P. H. Cole and D. C. Ranasinghe, "Networked rfid systems and lightweight cryptography," *London, UK: Springer. doi*, vol. 10, pp. 978–3, 2008.

[18] D. Xie and C.-C. Kuo, "Secure lempel-ziv compression with embedded encryption," in *Electronic Imaging 2005*. International Society for Optics and Photonics, 2005, pp. 318–327.

[19] J. Kelley and R. Tamassia, "Secure compression: Theory & practice," Cryptology ePrint Archive, Report 2014/113, 2014.

[20] M. O. Külekci, "On scrambling the burrows–wheeler transform to provide privacy in lossless compression," *Computers & Security*, vol. 31, no. 1, pp. 26–32, 2012.

[21] I. H. Witten and J. G. Cleary, "On the privacy afforded by adaptive text compression," *Computers & Security*, vol. 7, no. 4, pp. 397–408, 1988.

[22] H. Kim, J. Wen, and J. D. Villasenor, "Secure arithmetic coding," *Signal Processing, IEEE Transactions on*, vol. 55, no. 5, pp. 2263–2272, 2007.
[23] K.-K. Tseng, J. M. Jiang, J.-S. Pan, L. L. Tang, C.-Y. Hsu, and C.-C. Chen, "Enhanced huffman coding with encryption for wireless data broadcasting system," in *Computer, Consumer and Control (IS3C), 2012 International Symposium on*. IEEE, 2012, pp. 622–625.
[24] D. W. Gillman, M. Mohtashemi, and R. L. Rivest, "On breaking a huffman code," *IEEE Transactions on Information theory*, vol. 42, no. 3, pp. 972–976, 1996.
[25] M. Baptista, "Cryptography with chaos," *Physics Letters A*, vol. 240, no. 1, pp. 50–54, 1998.
[26] G. Jakimoski, L. Kocarev *et al.*, "Chaos and cryptography: block encryption ciphers based on chaotic maps," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 48, no. 2, pp. 163–169, 2001.
[27] F. Giesen, https://github.com/rygorous/ryg_rans.
[28] J. Duda, https://github.com/JarekDuda/AsymmetricNumeralSystemsToolkit.
[29] Y. Collet, https://github.com/Cyan4973/FiniteStateEntropy.