sciendo

Int. J. Appl. Math. Comput. Sci., 2023, Vol. 33, No. 1, 45–55
DOI: 10.34768/amcs-2023-0004

amcs

LIGHTWEIGHT COMPRESSION WITH ENCRYPTION BASED ON ASYMMETRIC NUMERAL SYSTEMS

JAROSŁAW DUDA [a], MARCIN NIEMIEC [b,*]

[a]Institute of Computer Science
Jagiellonian University
ul. Prof. Łojasiewicza 6, 30-348 Kraków, Poland
e-mail: dudajar@gmail.com

[b]Institute of Telecommunications
AGH University of Science and Technology
al. Mickiewicza 30, 30-059 Kraków, Poland
e-mail: niemiec@agh.edu.pl

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 a compression ratio comparable with arithmetic coding at a 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 an additional encryption layer when reaching a satisfactory protection level.

Keywords: symmetric cryptography, lightweight cryptography, data compression, entropy coding.

1. 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 (Huang *et al.*, 2015; El-Douh *et al.*, 2022). Unfortunately, cryptographic algorithms decrease performance. Moreover, it is impossible to implement security services in many systems with limited resources (e.g., resource-constrained IoT devices). 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 (Huffman, 1952), 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 $\{p_s\}$ of probability distributions with a coding optimal for distributions $\{q_s\}$, we use asymptotically $\Delta H = \sum_s p_s \lg(p_s/q_s)$ more bits/symbol than required (Kullback–Leibler divergence). 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

*Corresponding author

bit/symbol.

Arithmetic and range coding (Rissanen, 1976; Martin, 1979) 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 (Marpe *et al*., 2003) 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 (Duda, 2009; 2014b; 2015) for low cost systems: asymmetric numeral systems (ANSs). In contrast to prefix codes, this coding uses nearly accurate probabilities for coded symbols. The high performance and efficiency of ANSs is leading to Huffman and Range being replaced in new compressors (Collet, 2013b; Buzidi, 2014; Francesco, 2014), including the Facebook Zstandard compressor (FZC, 2016), the Apple LZFSE compressor (ALC, 2017) and JPEG XL (Alakuijala *et al*., 2019) to improve performance. Table 1 presents some benchmarks showing advantages especially in decoding time. 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, and the resulting improvements have been already demonstrated for FPGA (Najmabadi *et al*., 2015).

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 resource-constrained IoT devices. As such, there is a growing field of lightweight cryptography (Eisenbarth *et al*., 2007; Poschmann, 2009; Cole and Ranasinghe, 2008), with the 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 (Xie and Kuo, 2005; Kelley and Tamassia, 2014), Burrows–Wheeler transform (Külekci, 2012) and arithmetic coding (Witten and Cleary, 1988; Kim *et al*., 2007). These articles contain some general techniques, whose addition might be considered to improve security of discussed ANS-based encryption.

Huffman coding has also been discussed for adding simultaneous encryption (Tseng *et al*., 2012). An abstract of an article by Gillman *et al*. (1996) 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 (Duda, 2014b). 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 $x \in \{2^R, \ldots, 2^{R+1}-1\}$ 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 (Baptista, 1998; Jakimoski and Kocarev, 2001), the authors discuss three sources of chaos in evolution of this internal state, making its behavior virtually unpredictable while incomplete knowledge. Figure 1 compares it with the standard S-Box approach.

As only a few ciphers like the one-time pad can be formally proven to be safe, practical encryption schemes often require time to gain trust as being secure: by the lack of successful attacks. Hence, while there are some arguments of the 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 improvement in both the compression ratio and performance.

The idea presented in this paper was available as a preprint document before 2021. Therefore, the first papers that developed this approach—encryption based on ANS—were recently published (Camtepe *et al*., 2021; Mahboubi *et al*., 2022; Pieprzyk *et al*., 2022). This confirms usability and a great potential of this solution.

The remainder of the paper proceeds as follows. Section 2 introduces the ANS algorithm: coding and decoding as well as some examples of these steps. Section 3 presents the basic concept of including

Table 1. Benchmarks of data compressors from `https://encode.su/threads/3315-enwik10-benchmark-results` with ANS-based (marked grey) and classical. A great advantage, especially in decoding time, is visible. All but the last one use tANS variant, which allows for discussed simultaneous encryption through perturbation of coding tables.

10GB large text benchmark (2020, i9 9900K), 1GB wiki for 10 languages (ANS)			
Size [bytes]	Encoding [s]	Decoding [s]	Compressor, parameters, remarks
10,000,000,000	---	---	Uncompressed file
5,034,758,325	18.449	**7**.311	lz4 -1 (v1.9.2) LZ
4,666,386,317	26.686	**4**.827	lzturbo -10 -p0 (v1.2)
4,371,496,854	46.907	**7**.282	lz4x -1 (v1.60)
3,909,521,247	32.603	**11**.287	lizard -40 (v1.0.0)
3,823,273,187	136.146	**59**.070	g**zip** -1 (v1.3.12) LZ+Huffman
3,770,151,519	34.216	**26**.236	brotli -q 0 (v1.0.7)
3,660,882,443	767.399	**7**.633	lz4x -9 (v1.60)
3,642,089,943	28.752	**10**.717	**zstd** -1 (v1.4.5) LZ + **tANS**/huf
3,237,812,198	392.835	**53**.771	g**zip** -9 (v1.3.12)
3,095,248,795	137.881	**20**.738	brotli -q 4 (v1.0.7)
3,078,914,611	240.124	**9**.381	**zhuff** -c2 -t1 (v0.99beta), LZ4 + **tANS**
3,065,081,662	50.724	**12**.904	**zstd** -4 --ultra --single-thread (v1.4.5)
2,660,370,879	153.103	**19**.993	**lzturbo** -32 -p0 (v1.2), LZ + **tANS**
2,639,230,515	561.791	**11**.774	**zstd** -12 --ultra --single-thread(v1.4.5)
2,357,818,671	3,953.092	**34**.300	rar -m5 -ma5 -mt1 (v5.80)
2,337,506,087	2,411.038	**11**.971	**zstd** -18 --ultra --single-thread(v1.4.5)
2,220,027,943	7,439.064	**22**.690	brotli -q 10 (v1.0.7)
2,080,479,075	4,568.550	**12**.934	**zstd -22** --ultra --single-thread(v1.4.5)
2,059,053,547	4,909.124	**55**.188	7z -t7z -mx9 -mmt1 (v19.02) - LZMA
1,973,568,508	6,626.946	**89**.762	arc -m9 -mt1 (v0.67)
1,921,561,064	17,200.759	**27**.147	brotli -q 11 --large_window=30 (v1.0.7)
1,899,403,918	1,327.809	**375**.295	nz -c0 -t1 (v0.09 alpha)
1,722,407,658	778.796	**401**.317	m99 -b1000000000 -t1 (beta)
1,675,874,699	781.839	**198**.309	bwtturbo -59 -t0 (v20.2)
1,644,097,084	21,097.196	**93**.130	**razor** (v1.03.7) - **adaptive 4bit rANS**
1,638,441,156	1,030.489	**640**.502	bsc -m0 -b1024 -e2 -T (v3.1.0)
1,632,628,624	1,146.133	**1,284**.451	bcm -9 (v1.40)

encryption in tANS and properties influencing the security level: the set of cryptographic keys, chaotic behavior, etc. Section 4 describes the security features of this encryption method. Finally, Section 5 concludes the paper.

2. Asymmetric numeral systems (ANSs)

This section introduces ANSs, focusing on the tabled variant (tANS). A further discussion and other variants of ANS can be found in the work of Duda (2014b).

2.1. Coding into a large natural number. Let us first consider the standard binary numeral system. It allows us 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 a bit sequence of length approximately equal to $\lg(x)$. This length does not depend on exact values of symbols—this approach is optimized for a symmetric case of $\Pr(0) = \Pr(1) = 1/2$, when both symbols carry one bit of information. In contrast, a symbol sequence $\{p_s\}$ of probability distributions carries asymptotically $\sum_s p_s \lg(1/p_s)$ bits/symbols (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 ANSs.

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': $\{0,\ldots,2^n-1\}$ and $\{2^n,\ldots,2^{n+1}-1\}$. The symmetry of their lengths corresponds to the symmetry of the informational content of both symbols. As depicted in the left panel of Fig. 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, which is analogous to the need to remember the current position n in the standard numeral system.

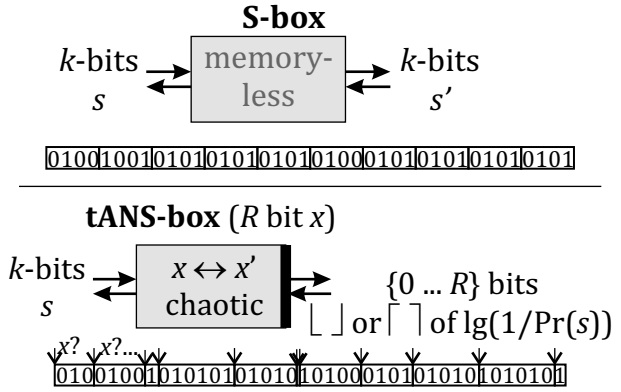

Fig. 1. While S-box performs substitutions/permutations on fixed length bit blocks, the discussed tANS analogue has additional hidden chaotic pseudorandom evolution of internal state x, which decides on both contents and lengths of the processed bit blocks. In this way, an attacker does not know the split of bit sequence into bit blocks.

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}
```

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 $D(x') = (s,x) = (\mod(x',2), \lfloor x'/2 \rfloor)$. 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 be 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

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

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 appearances 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 Fig. 2 depicts an example of such a process for the probability distribution $\Pr(0) = 3/4,\ \Pr(1) = 1/4$. Starting with symbol/state $x = 1$, 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. The ANS results in a lower representation than the standard numeral system, since it better corresponds with the digit distribution of the input sequence 01111.

Arithmetic formulas can be found using multiplication for such coding/decoding functions: uABS and rABS variants for the binary alphabet, and rANS variant for any large alphabet (Duda, 2014b). 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 considerably faster implementations (Giesen, 2014). 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.

2.2. 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 shown as Algorithms 1 and 2.

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

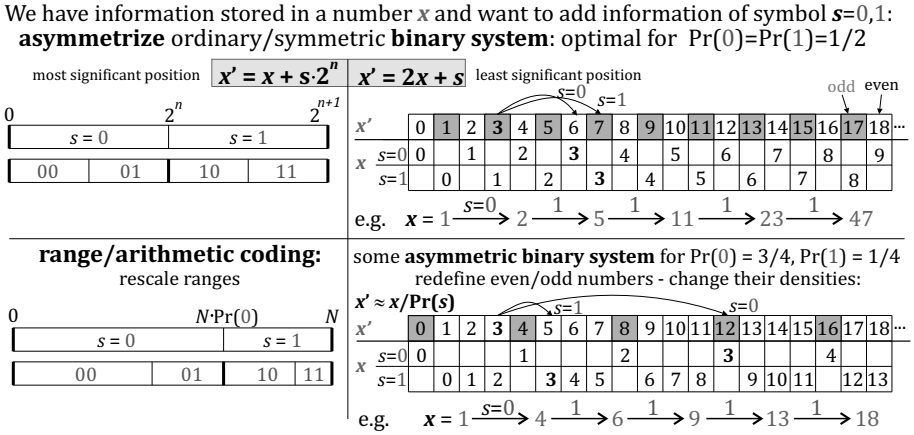

Fig. 2. Arithmetic coding (left) and ANS (right) seen as an asymmetrization of the standard numeral system—in order to optimize them for storing symbols from a general probability distribution.

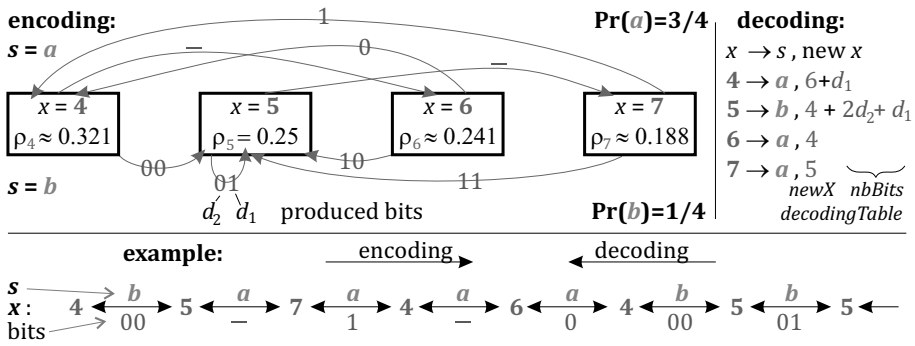

Fig. 3. Example of a four-state tANS (top) and its application for stream coding (bottom). State/buffer x contains $\lg(x) \in [2, 3)$ bits of information. Symbol b carries 2 bits of information, while a carries less than 1—its information is gathered in x until it accumulates to a complete bit of information.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

$I = \{L, \ldots, 2L-1\}$, 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 the uniqueness of the loop in Algorithm 1, we need to use a range of the type $I = \{L, \ldots, 2L-1\}$, where for practical reasons we will use $L = 2^R$. For the uniqueness of the loop in Algorithm 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 then used in Algorithm 2.

2.3. Tabled variant (tANS). In the tabled variant (tANS), which is used in most of compressors in Table 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 automaton with $L = 4$ states optimized for the binary alphabet with $\Pr(a) = 3/4$, $\Pr(b) = 1/4$, as depicted in Fig. 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, the "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 the 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 that the input source is i.i.d. and $\Pr(a) = 3/4$, $\Pr(b) = 1/4$, we can find the probability distribution ρ_x of the visiting states of such an automaton. It allows us to find the expected number of bits/symbol $H' \approx 1 \cdot 1/4 \cdot 2 + (0.241 + 0.188) \cdot 3/4 \cdot 1 \approx H + 0.01$, where $H = \sum_s p_s \lg(1/p_s)$ is the minimal value (Shannon entropy). Generally, as discussed by Duda (2014b), $\Delta H = H' - H$ behave approximately like m^2/L^2, where m is the size of the alphabet.

Connection with prefix codes. Using lookup tables, the decoding procedure can be written as Algorithm 3,

where $X = x - L \in \{0, \dots, 2^R - 1\}$ 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 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: it has a probability of $2^{-nbBits}$. 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, symbol spread "aaaabcdd" would lead to a decoder for prefix code $a \to 0,\ b \to 100,\ c \to 101,\ d \to 11$. Therefore, prefix codes can be regarded as a degenerated case of tANS.

2.4. Algorithms (tANS). Let us now construct the algorithms. Assume that $L = 2^R$: $I = \{L, \dots, 2L - 1\}$, $I_s = \{L_s, \dots, 2L_s - 1\}$ 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 $X := x - L \in \{0, \dots, 2^R - 1\}$ and store the symbol spread as the table $symbol[X] \equiv \overline{s}(X + L)$ of size L.

Algorithm 4 generates the $decodingTable$ for the efficient decoding step from Algorithm 3. For efficient memory handling while the encoding step, the encoding

Algorithm 3. Decoding step for prefix codes and tANS.

1: $t = decodingTable[X]$ $\{X \in \{0, \dots, 2^R - 1\}$ is current state$\}$
2: useSymbol($t.symbol$) {use or store decoded symbol}
3: $X = t.newX$+readBits($t.nbBits$) {state transition}

Algorithm 4. Generating tANS $decodingTable$.

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

table can be stored in the 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 Algorithms 5 and 6, respectively.

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 an optimal way seems a difficult problem. We present only a fast simple way of spreading symbols in a pseudorandom manner in Algorithm 7, which already offers excellent performance. Several symbol spreads can be found and tested in the work of Duda (2014a).

3. Adding encryption

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

3.1. Basic concept. We could use the freedom of choosing the exact coding for encryption purposes. For example, while building a 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.

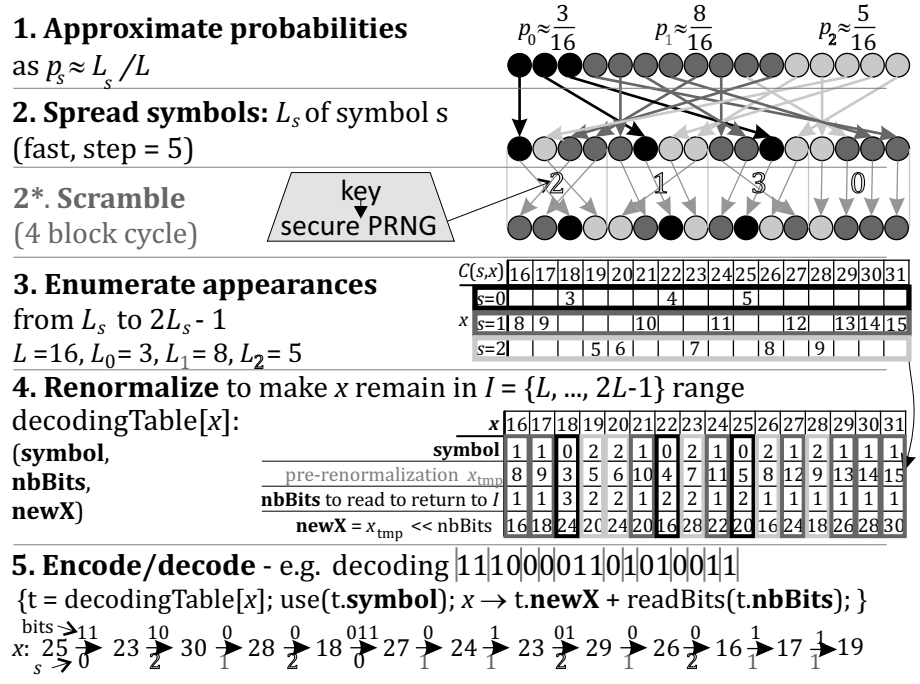

Fig. 4. Example of generation of tANS tables and applying them for stream decoding for an alphabet of size $m = 3$ and $L = 16$ states.

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: $next[s] = L_s$
1: **for** $x = L$ to $2L - 1$ **do**
2: $s = symbol[x - L]$
3: $encodingTable[start[s] + (next[s]++)] = x$;
4: **end for**

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 $\overline{s}$ 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., to put a successive symbol every number of positions specified by $step$ (cyclically). Then we can perturb the obtained symbol spread using a cryptographically-secure pseudorandom 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: $L = 16$, alphabet size $m = 3$, $step = 5$ 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 different locations of symbols result in different forms of encoded messages. The encoded messages depend strongly on the CSPRNG key.

3.2. 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, the alphabet of size $m = 256$ 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, i.e., $L_s = 1$, 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

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

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

Algorithm 7. Example of the fast symbol spread function (Collet, 2013a).

1: $X = 0$; $step = (5/8)L+3$ {some initial position and step}
2: **for** $s = 0$ to $m - 1$ **do**
3: **for** $i = 1$ to L_s **do**
4: $symbol[X] = s$; $X =$mod$(X + step, L)$
5: **end for**
6: **end 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 alignments. The probability that two B blocks of length from the i.i.d. probability distributions $\{p_i\}$ are accidentally equal is approximately $2^{-BH(p_1,\ldots,p_m)}$. Therefore, for practical scenarios (e.g., $m = 256$, $H > 1$), the reduction in the space of possibilities is practically negligible. For the DDP case, approximately $\left(\frac{L-m+1}{L}\right)^B \approx 0.345$ blocks have the dominating symbol only. The remaining ones are always changed by the perturbation: the number of possibilities is $\approx B^{(1-0.345)B/L} \approx 2.49 \cdot 10^{151}$.

3.3. 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).

Another source is the chaos of the dynamics of the internal state x, ensuring that incomplete knowledge leads to a rapid loss in 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 chaos as depicted in Fig. 5.

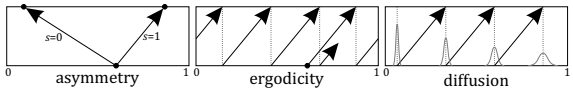

Fig. 5. Three sources of the chaotic behavior of the internal state x: $\lg(x) \to^{\approx} \lg(x) + \lg(1/p) \mod 1$.

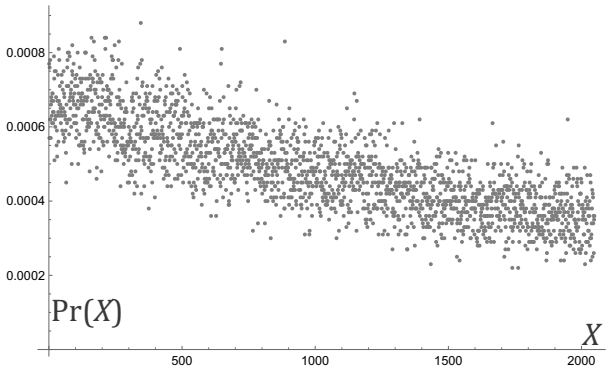

Fig. 6. Example of probability distribution of $X = x - L$ for $L = 2048$ and $m = 256$ and a fast symbol spread.

- 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 Fig. 6.

4. 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 of enhancing the security of this solution. The tests were mainly performed for the DP case: $L = 2048$, $m = 256$ for a pure tANS layer; imperfections can be easily removed, e.g., by a reduced number of layers of AES.

4.1. Balancing. The first question regarding the statistics of the produced bit sequence is the density of

zeros and ones. Are they equal? In general, for the 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 Fig. 6. For the DP case, tests show an 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 the ANS makes the issue $\Pr(0) \approx 0.501$ 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 numbers of zeros and ones.

4.2. Avalanche and nonlinearity. One crucial feature of the secure cipher is the strict avalanche criterion (SAC), which is satisfied if a change in a single bit of the key results, on the average, in a change in 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 the encryption process.

4.3. 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 in the output bits was high.

The same behavior was observed during the tests of completeness. Completeness is satisfied when a change in a single bit of the plaintext causes a change in around one half of bits of ciphertext. The discussed method processes successive single symbols; thus, a change in 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 in a symbol will be identical—their number depends on the probability of the symbol. Tests of further bits were not able to find statistical dependencies.

Operating on short bit blocks (of varying lengths) 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). In this way, the same symbol sequences lead to independent bit sequences. As the number of the initial states may be insufficient, this property can be enhanced by adding a few random symbols at the beginning of the 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 to 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., 10 kB size, which 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 the 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 the 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 the decoder, would strengthen this protection).

During the implementation of the 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. 10 kB) 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.

We can consider Facebook Zstandard which works in approximately 30 kB frames, each starting with ≈ 150 B header storing a probability distribution. To add a nearly free encryption, we can use CSPRNG with a seed (key, frame number, salt) to generate a coding table for each frame. The probability distribution should be encrypted, which can be done with the AES cipher applied to first, e.g., 200 B of the frame including the beginning of the bitstream.

It is worth mentioning that the NIST Statistical Test Suite (Bassham *et al*., 2010) was used to verify the security of the chosen ciphertexts generated by the cryptographic algorithm based on the ANS. This well-known tool provides 15 different test scenarios which assess the level of security of the tested input sequence (the obtained p-value specifies the level of randomness). It was observed that an initial state chosen by a user has no real influence on security—a few states were chosen and all tests were passed.

5. 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 the Huffman coding (due to having inexpensive linear initialization instead of the $n \log n$ cost of sorting in the Huffman coding). Using CSPRNG initialized with a cryptographic key to choose the coding tables means in the message encoded in this way can be simultaneously encrypted at nearly no additional cost. The variable-length nature of this coding makes the 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. Future work on the proposed compression algorithm with an encryption process should focus on advanced cryptanalysis and finding an optimal compromise between security and performance.

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. 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.

Acknowledgment

This work was partially supported by the *ECHO* project, which received funding from the European Union's Horizon 2020 research and innovation programme under the grant agreement no. 830943.

References

Alakuijala, J., Van Asseldonk, R., Boukortt, S., Bruse, M., Comșa, I.-M., Firsching, M., Fischbacher, T., Kliuchnikov, E., Gomez, S., Obryk, R. et al. (2019). JPEG XL next-generation image compression architecture and coding tools, *Applications of Digital Image Processing XLII, San Diego, USA*, pp. 112–124.

ALC (2017). Apple LZFSE compressor, `https://github.com/lzfse/lzfse`.

Baptista, M. (1998). Cryptography with chaos, *Physics Letters A* **240**(1): 50–54.

Bassham, L., Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Leigh, S., Levenson, M., Vangel, M., Heckert, N. and Banks, D. (2010). A statistical test suite for random and pseudorandom number generators for cryptographic applications, *NIST SP 800-22 Rev 1a*, National Institute of Standards and Technology, Gaithersburg, `https://www.nist.gov/publications/statistical-test-suite-random-and-pseudorandom-number-generators-cryptographic`.

Buzidi, H. (2014). LzTurbo compressor, `https://sites.google.com/site/powturbo/`.

Camtepe, S., Duda, J., Mahboubi, A., Morawiecki, P., Nepal, S., Pawłowski, M. and Pieprzyk, J. (2021). CompCrypt—lightweight ANS-based compression and encryption, *IEEE Transactions on Information Forensics and Security* **16**: 3859–3873.

Cole, P.H. and Ranasinghe, D.C. (2008). *Networked RFID Systems and Lightweight Cryptography*, Springer, London.

Collet, Y. (2013a). New generation entropy codecs: Finite state entropy and Huff 0, `https://github.com/Cyan4973/FiniteStateEntropy`.

Collet, Y. (2013b). Zhuff compressor, `http://fastcompression.blogspot.com/p/zhuff.html`.

Duda, J. (2009). Asymmetric numerical systems, *arXiv:* 0902.0271.

Duda, J. (2014a). ANS toolkit, `https://github.com/JarekDuda/AsymmetricNumeralSystemsToolkit`.

Duda, J. (2014b). Asymmetric numeral systems: Entropy coding combining speed of Huffman coding with compression rate of arithmetic coding, *arXiv:* 1311.2540.

Duda, J., Tahboub, K., Gadgil, N.J. and Delp, E.J. (2015). The use of asymmetric numeral systems as an accurate replacement for Huffman coding, *31st Picture Coding Symposium, Cairns, Australia*, pp. 65–69.

Eisenbarth, T., Kumar, S., Paar, C., Poschmann, A. and Uhsadel, L. (2007). A survey of lightweight-cryptography implementations, *IEEE Design & Test of Computers* **24**(6): 522–533.

El-Douh, A.A.-R., Lu, S.F., Elkouny, A.A. and Amein, A.S. (2022). Hybrid cryptography with a one-time stamp to secure contact tracing for COVID-19 infection, *International Journal of Applied Mathematics and Computer Science* **32**(1): 139–146, DOI: 10.34768/amcs-2022-0011.

FZC (2016). Facebook Zstandard compressor, `https://github.com/facebook/zstd`.

Francesco, N. (2014). LZA compressor, `http://heartofcomp.altervista.org/`.

Giesen, F. (2014). Simple rAns encoder/decoder, `https://github.com/rygorous/ryg_rans`.

Gillman, D.W., Mohtashemi, M. and Rivest, R.L. (1996). On breaking a Huffman code, *IEEE Transactions on Information Theory* **42**(3): 972–976.

Huang, Z., Liu, S., Qin, B. and Chen, K. (2015). Sender-equivocable encryption schemes secure against chosen-ciphertext attacks revisited, *International Journal of Applied Mathematics and Computer Science* **25**(2): 415–430, DOI: 10.1515/amcs-2015-0032.

Huffman, D. (1952). A method for the construction of minimum redundancy codes, *Proceedings of the IRE* **40**(9): 1098–1101.

Jakimoski, G. and Kocarev, L. (2001). Chaos and cryptography: Block encryption ciphers based on chaotic maps, *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* **48**(2): 163–169.

Kelley, J. and Tamassia, R. (2014). Secure compression: Theory & practice, *Cryptology ePrint Archive*, Report 2014/113, `https://eprint.iacr.org/2014/113`.

Kim, H., Wen, J. and Villasenor, J.D. (2007). Secure arithmetic coding, *IEEE Transactions on Signal Processing* **55**(5): 2263–2272.

Külekci, M.O. (2012). On scrambling the Burrows–Wheeler transform to provide privacy in lossless compression, *Computers & Security* **31**(1): 26–32.

Mahboubi, A., Ansari, K., Camtepe, S., Duda, J., Morawiecki, P., Pawłowski, M. and Pieprzyk, J. (2022). Digital immunity module: Preventing unwanted encryption using source coding, *TechRxiv*, (preprint).

Marpe, D., Schwarz, H. and Wiegand, T. (2003). Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard, *IEEE Transactions on Circuits and Systems for Video Technology* **13**(7): 620–636.

Martin, G. (1979). Range encoding: An algorithm for removing redundancy from a digitized message, *Institution of Electronic and Radio Engineers International Conference on Video and Data Recording, Southampton, UK*.

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

Pieprzyk, J., Pawlowski, M., Morawiecki, P., Mahboubi, A., Duda, J. and Camtepe, S. (2022). Pseudorandom bit generation with asymmetric numeral systems, *Cryptology ePrint Archive*, Report 2022/005, `https://ia.cr/2022/005`.

Poschmann, A.Y. (2009). Lightweight cryptography: Cryptographic engineering for a pervasive world, *Cryptology ePrint Archive*, Paper 2009/516, `https://eprint.iacr.org/2009/516`.

Rissanen, J.J. (1976). Generalized Kraft inequality and arithmetic coding, *IBM Journal of Research and Development* **20**(3): 198–203.

Tseng, K.-K., Jiang, J.M., Pan, J.-S., Tang, L.L., Hsu, C.-Y. and Chen, C.-C. (2012). Enhanced Huffman coding with encryption for wireless data broadcasting system, *IEEE International Symposium on Computer, Consumer and Control (IS3C), Taichung, Taiwan*, pp. 622–625.

Witten, I.H. and Cleary, J.G. (1988). On the privacy afforded by adaptive text compression, *Computers & Security* **7**(4): 397–408.

Xie, D. and Kuo, C.-C. (2005). Secure Lempel–Ziv compression with embedded encryption, *Electronic Imaging 2005, San Jose, USA* pp. 318–327, DOI: 10.1117/12.590665.

Jarosław Duda (Jarek Duda) is an assistant professor at Jagiellonian University. He holds degrees in computer science (PhD), mathematics (MSc) and physics (PhD). He is mainly focused on information theory, statistical analysis, and is known by his introduction of asymmetric numeral systems.

Marcin Niemiec is an associate professor at the Institute of Telecommunications, AGH University of Science and Technology. His research interests focus on cybersecurity, especially security services, symmetric ciphers, network security, intrusion detection, and quantum cryptography. He has actively participated in the 6th and 7th FP European programs (*ePhoton/ONE+, BONE, SmoothIT, INDECT*), Horizon 2020 Framework Programme (*SCISSOR, ECHO*), Eureka-Celtic (*DESYME*), and many national research projects. He has co-authored over 100 publications.

Received: 7 February 2022
Revised: 22 June 2022
Accepted: 9 November 2022