Comments on "Arithmetic Coding as a non-linear dynamical system"

Amit Pande[a,*], Joseph Zambreno[b], Prasant Mohapatra[a]

[a]*Department of Computer Science, University of California, Davis, CA, USA*
[b]*Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, USA*

Abstract

Nagaraj et al.[1, 2] present a skewed-non-linear Generalized Luroth Series (s-nGLS) framework. S-nGLS uses non-linear maps for GLS to introduce a security parameter *a* which is used to build a keyspace for image or data encryption. The map introduces non-linearity to the system to add an "encryption key parameter". The skew is added to achieve optimal compression efficiency. s-nGLS used as such for joint encryption and compression is a weak candidate, as explained in this communication. First, we show how the framework is vulnerable to known plaintext based attacks and that a key of size 256 bits can be broken within 1000 trials. Next, we demonstrate that the proposed non-linearity exponentially increases the hardware complexity of design. We also discover that s-nGlS can't be implemented as such for large bitstreams. Finally, we demonstrate how correlation of key parameter with compression performance leads to further key vulnerabilities.

Keywords:

1. Introduction

Multimedia communication with efficient compression and security has become an increasing concern for wide applications in commercial and defense applications. The pervasive use of multimedia communications in entertainment (HDTV, mobile, Internet video [3]), public domain (surveillance [4], tele-medicine) and defense applications (Unmanned air vehicles [5]) has increased concern for efficient multimedia encryption. The challenge of enabling both compression and security by a single operation is gaining importance given the ubiquitous nature of compressed media files, challenging demands of video compression systems and huge popularity of mobile videos (v.i.z. mobile phones, ipods, notebooks, HDTV etc). In 2011, more than 50% of data traffic in cellular networks was videos and this trend is going to increase.

Arithmetic Coding (AC) is widely used for the entropy coding of text and multimedia data. The range [0,1) is iteratively partitioned into sub-intervals according to the relative probabilities of occurrence of the input symbols (can be binary or multiple symbols). However, as conventionally implemented, it is not particularly secure.

*Corresponding author

Email addresses: amit@cs.ucdavis.edu (Amit Pande), zambreno@iastate.edu (Joseph Zambreno), prasant@cs.ucdavis.edu (Prasant Mohapatra)

In order to perform source coding (data compression), Nagaraj et al. [1, 2] present a new scheme called as Generalized Luroth Seroies (GLS). GLS treats messages as coming from independent and identically distributed (i.i.d.) sources. They are represented as imprecise measurements (symbolic sequence) of a chaotic system which is ergodic, preserves the Lebesgue-measure and is a nonlinear-dynamical system. GLS achieves Shannon's entropy bound and turns out to be a generalization of arithmetic coding, the popular source coding algorithm used in international compression standards such as JPEG2000 and H.264. The authors use a skew version of non-linear GLS coding (s-nGLS coding) for data encryption.

GLS is an ergodic and Lebesgue measure-preserving discrete dynamical system. Skewed non-linear GLS exhibits Robust Chaos, has positive Lyapunov exponents and preserves the Lebesgue measure and has high key sensitivity. However, the security of the scheme has not been proven. In our study, we found several vulnerabilities of the proposed scheme, as enumerated below.

Weakness of s-nGLS coding

1. **Known Plaintext Attack:** An adversary may make a wrong guess in value of the skew parameter a. However, this may lead to imperfect reconstruction and not necessarily to completely random output, which can be exploited for vulnerabilities. *A closely related value of a can lead to perfect reconstruction of first few symbols of a binary string even when it is not exactly the same as a.* Therefore, *it is possible for an adversary to launch a known plaintext attack and thus successfully iterate to guess the value of a.* As the guess of a gets closer to the actual value, more and more symbols will be reconstructed properly.
2. **Small bitstreams:** Traditionally Arithmetic coding based schemes can't be used to encode large strings because of 'big-number' arithmetic involved in sub-division of interval. However, re-normalization allows efficient implementation of arithmetic coding in software and hardware. GLS can't be re-normalized, hence unsuitable for large bitstreams.
3. **Computational Complexity:** Arithmetic coding is computationally expensive, as compared to other variable length coding schemes and thus less popular in baseline profiles of video encoders. *The skew proposed in [2] further increases the complexity of coder exponentially by adding a squaring and square root operation.*
4. **Compression-based attack:** The authors introduce a loss in compression efficiency to introduce what they call as "scrambling" the length of compressed data. However, this overhead is directly proportional to magnitude of key parameter a. This discrepancy can be exploited to leak the key.

In this paper, we briefly discuss these weaknesses with experimental results and observations. The paper is organized as follows: Section 2 gives a brief overview of s-nGLS scheme followed by discussion of weaknesses and experimental setup in subsequent sections.

2. Skewed-nGLS framework

GLS or Generalized-Luroth-Coding is a framework for joint source coding and encryption. The properties of a good encryption such as mixing and sensitivity to the key parameter are provided by a chaotic ergodic map [6]. GLS-coding is Shannon optimal and thus presented as an ideal candidate for joint source coding and encryption.

A measure preserving non-linear skew extension of GLS (known as s-nGLS) exhibits Robust Chaos [7] and can improve the key efficiency. The skew is obtained by approximating the straight

lines in GLS with parabolic trajectories. The skew is proportional to the probability of input symbol while non-linearity is proportional to key parameter.

s-n GLS coding exhibits robust chaos, which refers to the absence of attracting periodic orbits in the neighborhood of the parameter space. This feature makes Robust Chaos very desirable for cryptographic purposes. The piecewise non-linear generalization of GLS is given by the following equation:

$$s-nGLS(a,p,x) = \frac{(a-p)+\sqrt{(p-a)^2+4ax}}{2a}, 0 \le x < p. \quad (1)$$

$$= \frac{(1+a-p)+\sqrt{(p-a-1)^2+4a(1-x)}}{2a}, p \le x < 1 \quad (2)$$

where $0 \le x \le 1$ and $0 < a \le min\{p, 1-p\}$. Here p stands for $p(A)$ (for i.i.d binary source) and a is the private-key. Notice that in the limit as $a \to 0$, s-nGLS reduces to GLS (Skewed-Tent map). S-nGLS is a 2D map which exhibits Robust Chaos in both a and p dimensions.In a typical joint source coding and encryption application, for a given source (p is given), a acts as the private key (the key space is $(0,k)$, where $k = min\{p, 1-p\}$). Encoding and decoding are exactly the same as we did with GLS. Owing to Robust Chaos, two different keys a_1 and a_2 which are very near to each other would produce uncorrelated symbolic sequences after a few iterations. Thus, it can been seen that the same message (symbolic sequence) would get widely different initial conditions for different parties (different a). There is a slight loss of compression optimality since it is important to ensure that the length of the compressed and encrypted data are not predictable.

The inverse iteration over the map, is used for encoding symbols '0' and '1' on the chaotic map. We begin with initial interval [0,1) and iterate it over inverse map to get the final interval. The smallest codeword is chosen from the final interval and transmitted as compressed bitstream. The inverse map is given by:

$$s-nGLS^{-1}(a,p,y) = ay^2 - y(a-p), \text{ when encoding '0'} \quad (3)$$

$$= 1 - ay^2 - y(1+a-p), \text{ when encoding '1'} \quad (4)$$

s-n GLS is readily generalizable to non-binary alphabets. GLS can also be used for adaptive coding by appropriately changing the skew of the GLS to account for the changing source symbol probabilities at every iteration. GLS and its non-linear extensions (s-nGLS) exhibit properties of ergodicity (mixing), chaos (sensitive dependence on initial conditions), and strong pseudo-randomness (due to Robust Chaos). All these are desirable for joint source coding and encryption application. Unlike GLS, s-n GLS has higher computational complexity.

This work has inspired further works in the community. In [8], authors build a chaotic convolution coder using chaotic maps to alternate connection matrices. Cryptanalysis of these papers is presented in [9, 10]. [11] presents a modified scheme based on Nagaraj's work where they use M-ary codes (using piece-wise chaotic maps) and use a simple mechanism to data encryption.

3. Related works in joint compression and encryption

A Randomized Arithmetic Coding (RAC) scheme was presented by Grangetto et al. [12] in 2006. It achieves encryption by inserting some randomization in the arithmetic coding procedure

at no expense in terms of coding efficiency. RAC needs a key of length 1-bit per encoded symbol. The applications targeted by Grangetto et al. [12] was specifically an JPEG2000-encoder. Thus, a potential adversary will not have access to the original image or blocks of image coder nor be in a position to provide a particular image to be encoded. Thus, robustness to plaintext attacks was not a goal in original design of RAC. The RAC encoder, by itself, without the scrambling, confusion and diffusion offered by the image coder, willbe very vulnerable. The number of trials needed to determine an N-bit shuffling sequence would be on the order of N , since the output pairs corresponding to inputs that differ in exactly one symbol can be compared to get the result.

Wen et al. [13] presented Secure Arithmetic Coding (SAC) in 2007 where a arithmetic-coding based encoder is combined with a pre and post scrambling operation for improved security. The SAC based encoder is constructed over a Key-Splitting Arithmetic Coding (KSAC) presented by the same authors in 2006 [14]. IN KSAC, a key is used to split the intervals of an arithmetic coder based on choice of a key.

However, SAC introduces loss in coding efficiency particularly for small sized inputs, which are later restricted to a small value by putting some constraints on the keyspace. The SAC encoder may introduce multiple sub-intervals (which can be restricted by the algorithm) which significantly increasing the computational cost of encoder. Many papers have been published demonstrating successful attacks against SAC scheme [15, 16, 17, 18]. These works consider plaintext based attacks on these schemes.

4. Known-plaintext attack

Let us assume a case where the encoder uses M bits key (M can assume any value such as 32, 64, 128 or 256). If we split the interval $[0, p)$ ($p \leq 0.5$) into 2^M intervals, one key value represent one unique interval. We setup a simple experiment where we encode a symbol PT of length N and encode it using key value'a'.

$$CT = \text{GLS-encode}(PT, a)$$

An attacker has access to the encryption oracle, but no knowledge of the key. He sets up a trial experiment wherein he guesses the input keys. Let us call the guess key as a_g.

$$CT_g = \text{GLS-encode}(PT, a_g)$$

A metric $SigWt$ was defined to indicate the correlation between the coded output with random trials and the coded output with chosen value of 'a'. $SigWt$ is obtained by assigning a binary number to string obtained on XORing the trial output with coded output. Thus, the initial bits of SigWt (corresponding to initial bits of key) are given higher weight.

$$SigWt = \sum_{i=1}^{len} (CT(i) \oplus CT_g(i)) \times 2^{len-i}$$

where len is given by larger of the length of CT and CT_g. We vary the value a_g by varying it across the interval $[0, p)$ in $N + 1$ uniform steps. Thus, the step size is $\frac{1}{N}$. For GLS coding, the larger the difference $a_g - a$, the larger will be the value of $SigWt$. After, one iteration, we can reduce the interval to $[a_n - \frac{1}{N}, a_n + \frac{1}{N})$ by finding a_n corresponding to the minimum value of

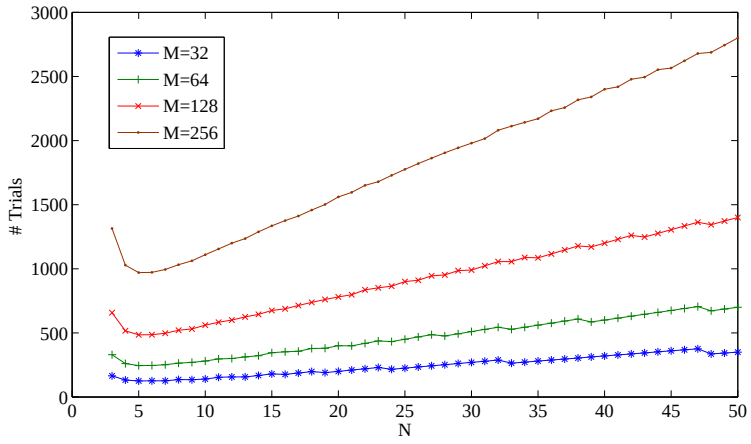


Figure 1: Known plaintext attack: Plot showing the number of trials required for worst case scenario as a function of N.

$SigWt$ in previous iteration. The number of iterations α required to find the exact key can be obtained by solving the following inequality:

$$(\frac{2}{N})^{\alpha} \leq 2^{-M}$$

which can be simplified to:

$$(\alpha + M)\log 2 - \alpha \log N \leq 0$$

The number of guesses is approximated by $N \times \alpha = 2\alpha(K.2^M)^{1/\alpha} = \frac{N\log(k2^M)}{\log(N/2)}$, which is a function with minimum at origin. The approximation was made by equating the inequality and considering fractional values. The exact value was found through numerical simulations where N was varied from 3 to 50 and the least value of α was found satisfying the inequality. The curve was plotted for different values of N (splits of interval) and M (length of key) (see Figure 1). The figure shows a dip for N=5 (not 3) as the exact solution for different scenarios. We also observe that a key of length 1000 can also be broken by around 1000 trials.

In the above discussion, we assumed the result that $SigWt$ metric shows a gradual dip across different values of a. For an ideal cipher this should actually be a straight line with a slight dip exactly at correct 'a' value, and all other values should give almost same values (zero correlation). To validate our assumption, we carried some simulations and the results are plotted in Figure 4. Figure 4(a-b) show the result for the case when original key value a is 0.3101 and the source probability value p is 0.6. Figure 4(c-d) show the result for the case when original key value a is 0.111101 and the source probability value p is 0.8. The metric $SigWt$ shows a gradual dip near the correct key value, which can be exploited to launch attacks on the system. Thus, GLS coding is prone to known plaintext attacks.

The authors propose to append some random data to the beginning of the message in order to ensure that nothing is revealed by observing a few iterations [2]. However, appending of some random data is found to have significant degradation in compression performance of the coding algorithm. We obtain a performance degradation of up to 80% for extreme symbol probabilities ($p > 0.85$ or $p < 0.15$) when XORing random bits to the beginning of message. The losses

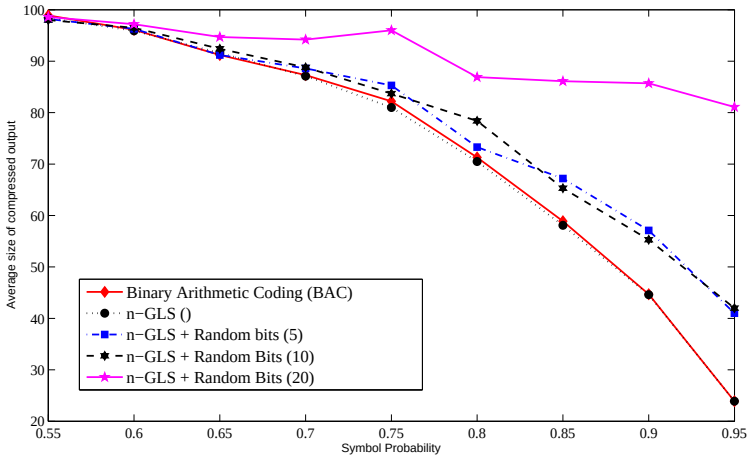


Figure 2: Compression losses with XORing random bits in beginning of message.

will be much higher in appending instead of XORing. Therefore, it is not reasonable to loose compression inefficiency at the cost of increased security strength. More details are given in Figure 2. Obviously, if we use large string size - say N=1000 or 10000, appending a few bits in the beginning will have negligible effect in compression efficiency.

However, we found that GLS scheme has limitations that it can't be, as such applied to large strings because of absence of renormalization proceedure. It is detailed in next section.

5. Small string restrictions

In practise, arithmetic coders operate at a fixed limit of precision. They know the approximate precision the decoder will be able to match and the calculations of fractions is approximated at both ends to same precision [19, 20, 21, 22]. As such, it is not possible to operate a simple arithmetic coder with strings larger than the precision offered by the decoder.

The process called as renormalization comes to rescue. It keeps the finite precision from becoming a bottleneck on the total number of symbols that can be encoded. Whenever the range is reduced to the point where all values in the range share certain beginning digits, those digits are sent to the output. As stated in [23]: "One of the major bottlenecks in any arithmetic encoding and decoding process is given by the renormalization procedure. Renormalization in the M coder is required whenever the new interval range R after interval subdivision does no longer stay within its legal range. Each time a renormalization operation must be carried out one or more bits can be output at the encoder or, equivalently, have to be read by the decoder. This process, as it is currently specified in the standard, is performed bit-by-bit, and it is controlled by some conditional branches to check each time if further renormalization loops are required. Both conditional branching and bitwise processing, however, constitute considerable obstacles to a sufficiently high throughput." Many schemes have been proposed for re-normalization [23, 24]

This *'renormalization'* step is required not just to accelerate the arithmetic coding on standard software and hardware platforms but also to make it feasible on these architectures. The reason

being as follows: Arithmetic coding (or GLS coding) techniques require splitting of intervals at every stage. In GLS coding this split is interpreted as back-iteration over chaotic maps and iterative shrinking of interval (initially [0,1)). The shrinking of these intervals is so fast that we can't actually compute GLS coding values for large strings such as 1000 or 10000 bits on fixed point arithmetic. We conducted experiments to find out the maximum length of strings which can be encoded using GLS coding (or arithmetic coding without renormalization). The results are presented in Figure 3 where we show the mean and standard deviations of runs over 32 and 64 bit IEEE single and double precision arithmetic coding values.

Thus, even with 64 bit arithmetic it is not possible to implement GLS coding in hardware for a string longer than 60 bits. An obvious solution to this problem is to apply the renormalization proceedure, same way as arithmetic coding. However, this is not possible.

Renormalization works for arithmetic coding because, in successive iterations - the window (range) of final codeword keeps shrinking and thus we can correspondingly shift our operating window also. For example, the initial interval [0,1) may shrink to [0.626, 0.663) and so on. In this case, we can output portion of final codeword ($0.101_{bin} = 0.625_{dec}$) and renormalize the intervals. This is not possible in chaotic mapping (GLS mapping) because there is no localization of codeword although the difference between beginning and final interval keeps shrinking.

It is not possible to implement GLS coding on long strings without this re-normalization even on hardware accelerators such as FPGA. Related research work [25, 26]have also raised this issue.

For experimental evaluation, in this work we use variable precision arithmetic in Matlab (similar to use of Big numbers in C) to find solutions with bigger strings (N=100 or 1000).

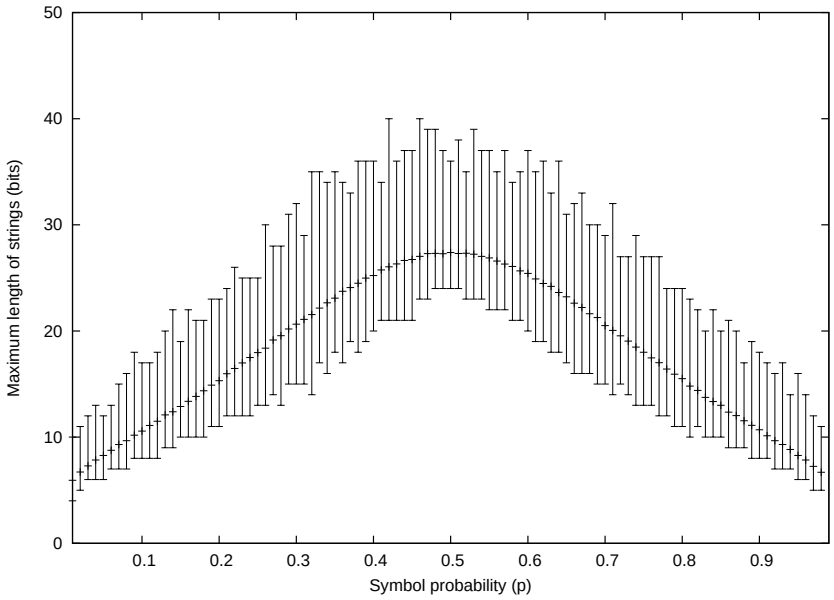
6. Computational Complexity

Joint source-coding frameworks such as GLS coding are presented with a motivation to reduce the computation cost of compression-then-encryption operation. However, as we shall identify in this section, s-nGLS coding massively increase the computational complexity of arithmetic coding and reduce the system throughput. Binary Arithmetic Coding (BAC) followed by encryption with AES (American Encryption Standard) algorithm is the naive candidate which should provide best security. AES was designed keeping in mind the requirements of both hardware and software and is therefore extremely fast when it is fully pipelined in custom hardware such as FPGA [27]. The authors of that paper were able to achieve a clock frequency of 184 MHz, and a net throughput of 23.57 MHz on Xilinx XC2V4000 FPGA. However, Binary Arithmetic Coding (in its original form) and GLS are sequential in nature. This becomes the bottleneck in a combined BAC+AES system. A combined system also needs dedicate hardware for both compression and encryption operation.

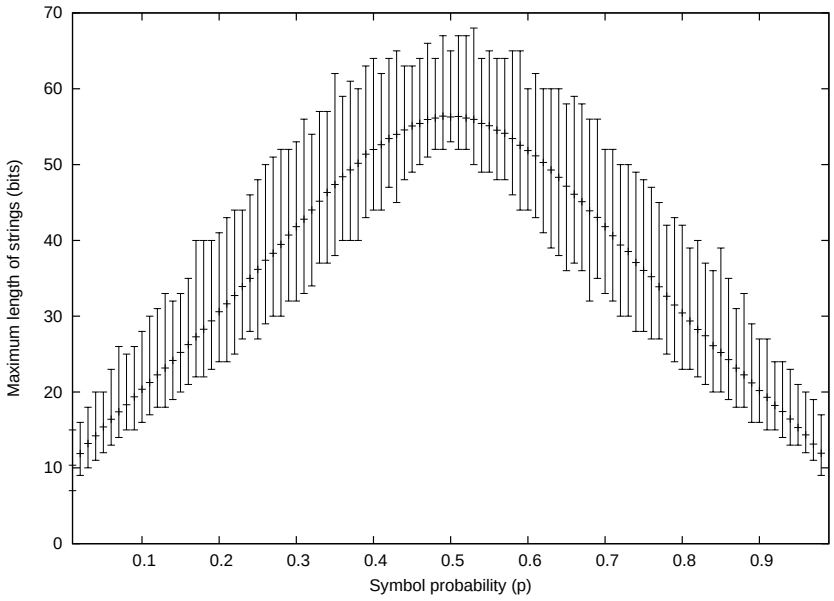
The arithmetic operations required for one bit encoding or decoding using BAC is 2 adders and 1 multipliers. The 128 bits AES encryption involves simple and basic operations such as table lookups, shifts, and XORs. In total, requires 40 sequential transformation steps are requreid by the operation. This approximately requires 336 bytes of memory, 608 XOR operations per cycle of AES or roughly 3 bytes memory and 5 XOR operations per single bit of encoding. However, s-nGLS encoding increases the computational complexity of Arithmetic Coding by 10 times (shown in results later) which is much more than the computational complexity of BAC+AES.

We used a Xilinx Virtex-6 VLX75 FPGA for synthesis using Xilinx ISE 12.4 design suite.

The s-nGLS decoder operation has one square and one square-root operation. Both operations have a significant overhead when implemented in hardware. The synthesis results illustrate

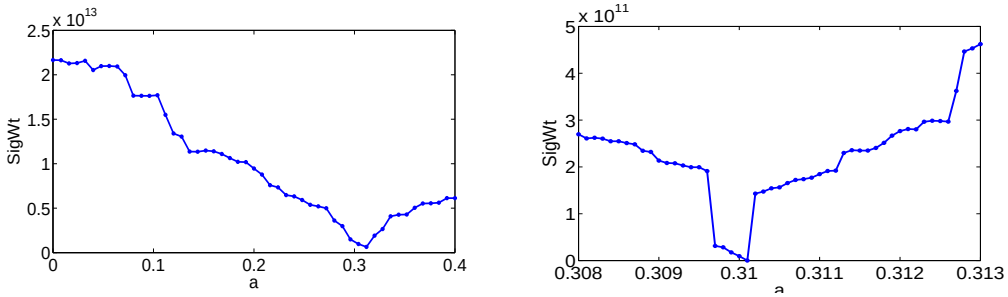


(*a*) Implementation on 32 bits single precision arithmetic (IEEE single precision format)



(*b*) Implementation on 64 bits double precision arithmetic (IEEE Standard 754, double precision format)

Figure 3: Figure showing the maximum length of a bitstream which can be encoded using GLS coding with standard fixed point arithmetic on most 'of-the-shelf' processors. The bars show the average (over 1000 simulations) length of encodes after which software precision is unable to resolve the sub-intervals in GLS iterations. The error bars show the minimum and maximum values over 1000 iterations.



(a) Original Key $a = 0.3101$, $p = 0.6$

(b) Magnified version of (a)

(c) Original Key $a = 0.111101$, $p = 0.8$

(d) Magnified version of (c)

Figure 4: Corelation values for GLS encoding (N=20, averaging over 1000 simulations)

the overhead in hardware implementation of s-n GLS coding. We were able to design a 48 bit fixed point implementation of s-n GLs coding, which take over 10x more logic slices than a direct implementation of Binary Arithmetic Coding. The maximum achievable clock frequency for the design reduces to 240 MHz which is a drop by 2.5 times, as compared with implementation of Binary Arithmetic Coding.

The approximate implementation results are shown in Table 1. We used Xilinx CORE generator to generate square root and multiplier/ squarer. The Xilinx CORDIC LogiCORE is a module for generation of the generalized coordinate rotational digital computer (CORDIC) algorithm which iteratively solves trigonometric, hyperbolic and square root equations. The core is fully synchronous using a single clock. However, it can generate square root core for upto 48 bit fixed point implementation only.

Table 1: Computational Complexity: 10x increase in hardware resources and 2.5x slowdown in s-nGLS implementation over FPGA

	bits	Logic Slices	Frequency (MHz)
Binary Arithmetic Coding	48	600	581
Square root	48	2860	241
Multiplier	48	2424	450
s-n GLS (approx)	**48**	**5884 (10x)**	**241 (0.4x)**
Binary Arithmetic Coding	64	1685	500
Square root	64	-	-
Multiplier	64	4256	450
s-n GLS (approx)	**64**	-	-

The computational complexity of n-GLS coding is significantly higher than binary arithmetic coding for large strings (say, N=10000) because the renormalization process traditionally used in arithmetic coding can't be used for GLS framework [26]. The increased computational complexity of s-nGLS framework adds to this complexity.

7. Compression-based attacks

The authors introduce a loss in compression efficiency to introduce what they call as "scrambling" the length of compressed data. The introduction of key parameter "a" quadratically affects the width of final interval (see equation 3 and 4). The larger the value a, the larger is the loss of compression efficiency.

In GLS coding, as in arithmetic coding, there is no direct prediction or correlation between source symbol probability and size of compressed bitstream. However, the average length of coded bitstream (for large number of plaintexts) is determined by the source symbol probability (p) according to Shannon's entropy theorem. As we go on increasing the value of a, the average size of compressed bitstream keeps increasing proportional to magnitude of a. This property can be used to localize the value of a using compression statistics, although it is not possible to obtain the precise value of a using this approach.

To test this property of s-nGLS coding, we set a sample test where 50 random plaintexts were encoded with increasing values of a. 50 sample values of a were chosen, uniformly distributed over range [0,p). The result is shown in Figure 5. The gradual shift from left to right (blue

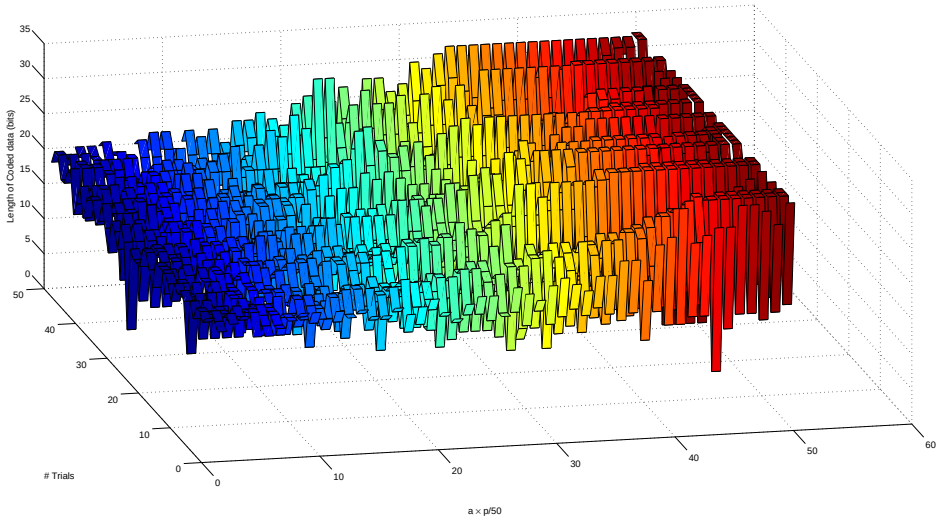


Figure 5: Compression based attack: The length of compressed bitstream is proportional to the magnitude of key parameter a (reported over 50 trials)

to red) indicates gradual increase of a values. We can also observe gradual increase in size of compressed bitstream over different trials.

8. Conclusions

In this letter, we discuss some weaknesses of s-nGLS to use as a framework for joint video compression and encryption. There is a great scope of future work - to use the attractive features of GLS encoding positively by alleviating the above mentioned limitations.

References

[1] N. Nagaraj, P. Vaidya, K. Bhat, Joint Entropy Coding and Encryption using Robust Chaos, Arxiv preprint nlin/0608051.

[2] N. Nagaraj, P. Vaidya, K. Bhat, Arithmetic coding as a non-linear dynamical system, Communications in Nonlinear Science and Numerical Simulation 14 (4) (2009) 1013–1020.

[3] W. I. Grosky, C. Zhang, S.-C. Chen, Guest editors' introduction: Intelligent and pervasive multimedia systems, IEEE MultiMedia 16 (2009) 14–15. doi:http://doi.ieeecomputersociety.org/10.1109/MMUL.2009.10.

[4] V. Singh, M. Kankanhalli, Adversary aware surveillance systems, IEEE Trans. on Information Forensics and Security 4 (3) (2009) 552–563.

[5] J. Y. S. Gorman, A. Cole, Insurgents hack U.S. drones, Wall Street Journal Online http://online.wsj.com /article/SB126102247889095011.html (2009).
URL `http://online.wsj.com /article/SB126102247889095011.html`

[6] G. Alvarez, S. Li, Some basic cryptographic requirements for chaos-based cryptosystems, International Journal of Bifurcation and Chaos in Applied Sciences and Engineering 16 (8) (2006) 2129.

[7] S. Banerjee, J. Yorke, C. Grebogi, Robust chaos, Physical Review Letters 80 (14) (1998) 3049–3052.

[8] T. Y. Ng, L. Chew, S. Puthusserypady, Error correction with chaotic switching between convolutional codecs, Circuits and Systems I: Regular Papers, IEEE Transactions on 55 (11) (2008) 3655 –3662. doi:10.1109/TCSI.2008.925938.

[9] J. Zhou, O. Au, Cryptanalysis of chaotic convolutional coder, in: Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on, 2010, pp. 145 –148. doi:10.1109/ISCAS.2010.5536959.
[10] J. Zhou, O. Au, On the security of chaotic convolutional coder, Circuits and Systems I: Regular Papers, IEEE Transactions on 58 (3) (2011) 595 –606. doi:10.1109/TCSI.2010.2073852.
[11] K. Wong, Q. Lin, J. Chen, Simultaneous arithmetic coding and encryption using chaotic maps, Circuits and Systems II: Express Briefs, IEEE Transactions on 57 (2) (2010) 146–150.
[12] M. Grangetto, E. Magli, G. Olmo, Multimedia selective encryption by means of randomized arithmetic coding, IEEE Trans. Multimedia 8 (5) (2006) 905–917. doi:10.1109/TMM.2006.879919.
[13] H. Kim, J. Wen, J. Villasenor, Secure arithmetic coding, IEEE Trans. Signal Processing 55 (5) (2007) 2263–2272. doi:10.1109/TSP.2007.892710.
[14] J. Wen, H. Kim, J. Villasenor, Binary arithmetic coding with key-based interval splitting, IEEE Trans. Signal Processing Letters 13 (2) (2006) 69–72. doi:10.1109/LSP.2005.861589.
[15] G. Jakimoski, K. Subbalakshmi, Cryptanalysis of some multimedia encryption schemes, IEEE Trans. Multimedia 10 (3) (2008) 330–338. doi:10.1109/TMM.2008.917355.
[16] J. Zhou, O. C. Au, X. Fan, P. H. W. Wong, Joint security and performance enhancement for secure arithmetic coding, in: ICIP, 2008, pp. 3120–3123.
[17] J. Zhou, O. C. Au, P. H. Wong, X. Fan, Cryptanalysis of secure arithmetic coding, in: ICASSP, 2008, pp. 1769–1772.
[18] H.-M. Sun, K.-H. Wang, W.-C. Ting, On the security of the secure arithmetic code, IEEE Trans. Information Forensics and Security 4 (4) (2009) 781–789. doi:http://dx.doi.org/10.1109/TIFS.2009.2031944.
[19] I. Witten, R. Neal, J. Cleary, Arithmetic coding for data compression, Communications of the ACM 30 (6) (1987) 520–540.
[20] J. Rissanen, G. Langdon, Arithmetic coding, IBM Journal of research and development 23 (2) (1979) 149–162.
[21] G. Langdon, An introduction to arithmetic coding, IBM Journal of Research and Development 28 (2) (1984) 135–149.
[22] A. Moffat, R. Neal, I. Witten, Arithmetic coding revisited, ACM Transactions on Information Systems (TOIS) 16 (3) (1998) 256–294.
[23] D. Marpe, G. Marten, H. L. Cycon, S. I. W. Kolloquium, T. U. Ilmenau, A fast renormalization technique for h.264/mpeg4-avc arithmetic coding (2006).
[24] Y. Jia, E. hui Yang, D. ke He, S. Chan, A greedy renormalization method for arithmetic coding, Communications, IEEE Transactions on 55 (8) (2007) 1494 –1503. doi:10.1109/TCOMM.2007.902534.
[25] A. Pande, P. Mohapatra, J. Zambreno, Using chaotic maps for encrypting image/ video content, in: IEEE International Symposium of Multimedia, 2011.
[26] A. Pande, J. Zambreno, P. Mohapatra, Hardware architecture for simultaneous arithmetic coding and encryption, in: International Conference on Engineering of Reconfigurable Systems and Algorithms, 2011.
[27] J. Zambreno, D. Nguyen, A. N. Choudhary, Exploring area/delay tradeoffs in an AES FPGA implementation, in: Proc. IEEE Intl. Conf. Field Programmable Logic and Applications, FPL 2004, 2004, pp. 575–585.