Hindawi Publishing Corporation
EURASIP Journal on Information Security
Volume 2007, Article ID 79048, 7 pages
doi:10.1155/2007/79048

Research Article

Joint Source, Channel Coding, and Secrecy

Enrico Magli, Marco Grangetto, and Gabriella Olmo

Dipartimento di Elettronica, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy

Correspondence should be addressed to Enrico Magli, enrico.magli@polito.it

Received 1 March 2007; Accepted 13 August 2007

Recommended by Qibin Sun

We introduce the concept of joint source coding, channel coding, and secrecy. In particular, we propose two practical joint schemes: the first one is based on error-correcting randomized arithmetic codes, while the second one employs turbo codes with compression, error protection, and securization capabilities. We provide simulation results on ideal binary data showing that the proposed schemes achieve satisfactory performance; they also eliminate the need for external compression and ciphering blocks with a significant potential computational advantage.

1. INTRODUCTION AND BACKGROUND

The design of modern multimedia communication systems is very challenging, as the system must satisfy several contrasting requirements. On one hand, because of the typically limited transmission bandwidth, compression is necessary in order to serve the highest possible number of users. On the other hand, compression typically renders the transmission very sensitive to errors or packet losses, which can very easily decrease the quality perceived by the final users. In addition to these problems, the ease of access and distribution of digital multimedia contents makes it possible to infringe the rights of the copyright owners by generating copies of the content which are exactly identical to the original. Therefore, the last years have witnessed an enormous growth of activity in the field of digital rights management.

The design issues mentioned above have received a great deal of attention, either individually or jointly. The last decades have seen a lot of technical advances in source coding (e.g., context-based coding) and channel coding (e.g., turbo codes and low-density parity-check codes). Joint source and channel coding have also been considered; while different source and channel codes can be used in a joint scheme in order to achieve the desired balance between source and channel coding (see, e.g., [1]), it is known that both source codes and channel codes can be used to perform the joint task. For example, ternary arithmetic coders (AC) with a forbidden symbol, along with their soft decoding, have been proposed for compression and error protection [2]. On the other hand, turbo codes have also been used for compression and protection; the compression is obtained by not sending the systematic bits and heavily puncturing the parity bits, while the decoder can take advantage of knowledge of the source prior probabilities [3].

The problem of joint compression and security has also been considered. In [4], a binary arithmetic coder with security capabilities has been proposed, which employs pseudorandom swapping of the intervals in order to prevent decoding by an unauthorized user without knowledge of the pseudorandom key; this scheme has been generalized in [6, 20] to arbitrary partitions of the unit interval. With arithmetic and Huffman codes, security can also be achieved through proper design of the probability model, as shown in [5]. The main problem of joint compression and security is to make sure that security does not decrease the compression performance. To do so, encryption must take place after compression, even though it has been shown [6] that in some cases the reverse is also possible by using distributed source coding ideas.

Some work has also been presented regarding the problem of joint channel coding and security. Following McEliece's pioneering work [7], several authors have investigated the possible use of channel codes for security applications. In [8], the authors propose to use punctured turbo codes and to adapt the puncturing pattern in order to obtain security. In [9], it is also proposed to exploit perfect punctured erasure-correcting codes, in such a way that correct decoding is guaranteed provided that at least as many symbols

as the input symbols are received out of the systematic and parity bits, and the puncturing pattern is known.

In this paper, we present the first attempt at designing practical algorithms that perform source coding, channel coding, and security within one single tool. This is very important in light of simplifying the system design, as well as providing advanced features. For example, if channel coding was applied to a JPEG 2000 compressed and secured image, after channel coding, the data would not be format-compliant; moreover, it would be difficult to perform compressed-domain processing such as rate adaptation or resolution-progressive transmission, because these operations would require channel decoding. On the other hand, having a single tool provides maximum flexibility in handling the compressed, secured, and protected file.

In particular, inspired by the duality between source and channel coding, we propose two algorithms based on compression and channel coding, respectively. The first algorithm employs arithmetic codes for error correction [2], improved with interval randomization as in [4]. The second algorithm is based on turbo codes, can exploit the prior source probability to improve compression, and randomizes the interleaver, the puncturing pattern and the bitstream in order to achieve secrecy. Both algorithms have the desirable feature that security is obtained without any degradation of compression performance.

This paper is organized as follows. In Sections 2 and 3, we describe the two algorithms, as well as their security issues. In Section 4, we show performance results, and in Section 5, we draw some conclusions.

2. PROPOSED ALGORITHM 1: ERROR-CORRECTING RANDOMIZED ARITHMETIC CODING

The first proposed algorithm is based on the ternary arithmetic coder with forbidden symbol described in [2].

2.1. Encoder

We consider a binary memoryless source $\mathbf{X}$ with probabilities P_0 and P_1, encoded by means of a ternary AC with alphabet "0," "1," and μ (the forbidden symbol), with probability distribution $P_0(1-\varepsilon)$, $(1-P_0)(1-\varepsilon)$, and ε. We consider the encoding of a length-L string $\mathbf{X}$, which is mapped onto a variable-length sequence $\mathbf{Y}$ of length L_Y bits. The introduction of μ produces an amount of artificial coding redundancy per encoded bit equal to $-\log_2(1-\varepsilon)$, at the expense of compression efficiency.

The AC encoding procedure is based on the classical recursive probability interval partition known as Elias coding; at each iteration, the interval is split into three subintervals that represent symbols "0", "1", and μ, and the subinterval corresponding to "0" or "1" is selected depending on the value of the input bit. Unlike a classical AC, in which the relation between interval and symbols is fixed and known by both the encoder and the decoder, in the proposed, algorithm we extend the approach of [4] to the ternary coder, by randomizing the order of the three intervals at each encoding step (i.e., at each input bit). By doing so, even in the error-free case, the decoder is going to be able to decode the received codeword only if it can replicate the sequence of pseudorandom numbers (i.e., the key) that have been used to randomize the intervals. Interestingly, this randomization does not affect the bit rate, since the length of the final interval is the same as the nonrandomized coder; the redundancy is related to the introduction of the forbidden symbol, and has the purpose of allowing the correction of transmission errors.

At each input bit, the interval ordering is selected through the sample value of a random variable S, which is used to pick one out of the six possible orderings of "0", "1", and μ; for maximum randomization, all orderings have the same occurrence probability, equal to 1/6. The sample drawing of random variable S is done using a pseudorandom number generator with seed K, which represents the encryption key. Note that, since this coder encrypts one bit at a time during the AC process, it can be functionally seen as a stream cypher.

2.2. Decoder

The coding redundancy can be exploited by the decoder for error correction. Let us consider that $\mathbf{Y}$ is transmitted, and the receiver observes the sequence $\mathbf{R}$ of length $L_R = L_Y$, through a channel with transition probability $P(\mathbf{R}|\mathbf{Y})$. The objective of the MAP decoder is to select the most likely input string

$$\hat{\mathbf{X}} = \arg\max_{\mathbf{X}} P(\mathbf{X}|\mathbf{R}) = \arg\max_{\mathbf{X}} \frac{P(\mathbf{R}|\mathbf{Y})P(\mathbf{X})}{P(\mathbf{R})}. \quad (1)$$

The MAP metric is characterized by the a priori source term $P(\mathbf{X})$, and the probability $P(\mathbf{R})$ of observing a certain sequence at the receiver. We can write $P(\mathbf{R}) = \sum_{\mathbf{Y}\in\mathbf{B}_{L_R}} P(\mathbf{R}|\mathbf{Y})P(\mathbf{X})$, where $\mathbf{B}_{L_R}$ is the subset of all coded sequences whose length is L_R bits, with L_R the length of $\mathbf{R}$.

The estimation of $\hat{\mathbf{X}}$ is carried out using a sequential decoding technique as detailed below.

2.3. Detailed algorithm implementation and setup

The AC encoding procedure maps the input sequence $\mathbf{X}$ onto the probability interval $I(\mathbf{X})$, represented by the codeword $\mathbf{Y} = [Y_0, Y_1, \ldots, Y_{L_{R-1}}]$ of length L_R. The encoding is based on the iterative selection of the probability interval $I(\mathbf{X})$, which is progressively refined according to the input source symbols X_k, $k = 1, \ldots, L$. At the first iteration, the interval is initialized to $I_0 = [0, 1)$, that is, the whole probability space. For $k = 1, \ldots, L$, the interval $I_k = [I_k^l, I_k^r)$ is progressively selected according to the following rules:

$$I_k = \begin{cases} [I_{k-1}^l, I_{k-1}^l + |I_{k-1}|P(X_{k-1}=0)), & \text{if } X_{k-1}=0, \\ [I_{k-1}^r - |I_{k-1}|P(X_{k-1}=1), I_{k-1}^r), & \text{if } X_{k-1}=1, \end{cases} \quad (2)$$

where $|I_k| = I_k^r - I_k^l$ represents the interval length at iteration k. After all source bits have been processed, the interval $I_L = I(\mathbf{X})$ is identified along with the codeword $\mathbf{Y}$,

which corresponds to the binary representation of the value $v(\mathbf{Y}) : v(\mathbf{Y}) \in I(\mathbf{X})$, chosen so as to minimize the number of bits in $\mathbf{Y}$. In order to guarantee the decoding uniqueness, either the number of input symbols L is provided to the decoder, or an end-of-file symbol is used to terminate the decoding process.

The coding procedure described above is made feasible in practice by means of finite precision arithmetic, with proper scaling of the interval I_k in order to avoid the underflow condition. Furthermore, the interval scaling makes the sequential encoding easier. For example, as soon as $|I_k^r| < 0.5$, the most significant bit of $\mathbf{Y}$ is fully determined and can be transmitted to the decoder, and the interval can be scaled up by a factor of two. The results reported in this paper are obtained with a 32-bit fixed-point implementation.

In principle, the metric (1) must be evaluated for all possible transmitted strings $\mathbf{Y} \in \mathbf{B}_{L_R}$. This is practically infeasible for reasonable values of L, and therefore suboptimal sequential search is employed. The decoding metric is decomposed into additive terms $m = \log[P(\mathbf{X}|\mathbf{R})] = \sum_{j=0}^{L_R-1} m_j$, with additive branch metric $m_j = \log[P(R_j|Y_j)] + \log[P(X_j)] - \log[P(R_j)]$; $P(X_j)$ represents the a priori probability of the source bits output by the arithmetic decoder in correspondence of the jth bit of $\mathbf{Y}$, and $P(R_j)$ the probability of the jth bit of $\mathbf{R}$. As in [2], we let $P(R_j) = 0.5$, assuming that all the received sequences in $\mathbf{B}_{L_R}$ are equally probable.

The problem then recasts as a search along the binary tree of all sequences of length L_R. In this paper, the M-algorithm [10] is used to this purpose. At each depth j, the M-algorithm limits the search space to the M nodes that exhibit the highest posterior probability; for this reason, it belongs to the so-called *breadth-first* techniques. At each iteration, all the stored nodes are extended one step forward and only the best M, out of the $2M$ possible paths, are kept for the next recursion. When the algorithm reaches the maximum depth $j = N-1$, the surviving path with the best accumulated metric is taken as the best estimate $\hat{\mathbf{Y}}$. Note that, because of the search space reduction, the correct path can be irreversibly dropped during the recursions, leading to a decoding failure. We have selected $M = 256$ as a good compromise between performance and complexity.

2.4. Security

The cryptanalysis of a randomized binary arithmetic coder has been presented in [4]; the main result is due to [11] and shows that resynchronization of the first b bits of a binary arithmetic coded string requires no less than $O(2^{b/2})$ decoding attempts. This leads to huge complexity for a brute force attack, also in case only partial resynchronization is acceptable. This bound, which is derived counting the number of existing sequences with $b/2$ ones and $b/2$ zeros, is also a lower bound for the ternary coder as used in this paper.

It is also worth investigating the robustness of randomized arithmetic coders towards plaintext attacks. While in general these coders are robust towards known-plaintext attacks, in [12], it has been shown that if the security lies in the adaptive probability model, these algorithms are susceptible to chosen-plaintext attacks; specifically, only $w+2$ symbols are required to break a probability of w bits. The security of randomized arithmetic coders as in [4] does not rely on the adaptive probability model, but rather on the random interval swapping. However, it has been pointed out in [13] that such randomized arithmetic coders are also susceptible to chosen-plaintext attacks. In fact, if an attacker has access to the randomized encoder, they would need to perform a number of trials on the order of N to determine an N-bit pseudorandom swapping sequence; this could be done by comparing N-output pairs corresponding to inputs that differ in exactly one symbol. In [13], input and output permutations are used to make the arithmetic coder secure to chosen-plaintext attacks; this could also be applied to our proposed algorithm.

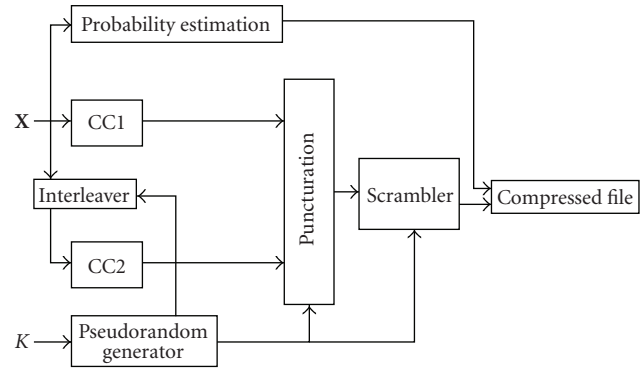

Figure 1: Block diagram of the joint encoder.

3. PROPOSED ALGORITHM 2: PUNCTURED TURBO CODING

3.1. Encoder

The block diagram of the second algorithm is shown in Figure 1. The private key K is used to initialize the pseudorandom number generator; the purpose of this generator is to create a random interleaver of suitable size for the turbo coding process as well as a pseudorandom puncturing pattern.

The core of the encoder lies in the parallel turbo code, which consists in the blocks labelled as first constituent convolutional code "CC1," "Interleaver," and second constituent convolutional code "CC2" in Figure 1. In particular, the binary source information string $\mathbf{X}$ is encoded using the (CC1), generating a first set of parity bits in addition to the systematic information bits. Since, unlike a separated source and channel coding scenario, in our setting, the turbo code also performs compression, only the parity bits from CC1 are retained, whereas the systematic bits are discarded.

A known permutation (interleaving) is applied to the input source data. The permutation is such that the position of each and every input symbol is scrambled in the interleaved sequence, according to a pseudorandom order. The purpose of the interleaver is mainly to increase the number of states of the turbo coder, and not to provide security. However, the interleaving operation does add some security if the scrambling

rule is kept secret, as is the case of the proposed scheme. The interleaved sequence is fed as input to the (CC2), which generates a second set of parity bits; as done with CC1, the systematic bits are discarded.

The parity bits from CC1 and CC2 are input to the puncturation block. As has been said, unlike other turbo coding schemes, the systematic information bits are discarded. This is necessary for two reasons. The first is that we want to achieve compression, and hence we need to transmit few bits. The second, equally important, is that the systematic bits would represent plaintext that is left in the clear; this would be a threat to the system security, as these bits would leak information on the other plaintext bits through the convolutional codes. Since in our implementation the constituent convolutional codes have rate 1/2, for an information string of L-bits we obtain $2L$ parity bits. The L systematic bits are discarded, and the parity bits are punctured as much as necessary in order to obtain the desired bit rate; a pseudorandom puncturing pattern is employed.

The punctured bits are further scrambled using a pseudorandom scrambling pattern. This has the purpose of complicating a possible attack aiming at estimating the puncturation pattern, possibly using different coded and protected versions of the same image, as it makes it impossible to carry out statistical inference on corresponding bits in two or more coded sequences.

The final compressed file contains the punctured and scrambled parity bits; moreover, it also contains the probability $P_0 = P(X_i = 0)$ of the original sequence, which can be used by the decoder to improve the turbo decoder performance.

3.2. Decoder

The interleaver, puncturation pattern, and scrambling pattern are not explicitly revealed to the decoder. However, the encoder and the decoder share common randomness; hence if the decoder knows the private key used by encoder, it will be able to generate the same interleaver, puncturation pattern, and scrambler employed at the encoder side. In this case, correct decoding is possible.

The decoding process works as follows. A schematic is given in Figure 2, while a detailed description along with pseudocode is available in [14]. The punctured parity bits $Xp1$ and $Xp2$ related to constituent code "CC1" and "CC2" are input to a maximum a posteriori decoder of the respective convolutional code. Each decoder generates soft decoding information in terms of the extrinsic information $Le12$ of decoder 1 to be passed to decoder 2, and $Le21$ of decoder 2 to be passed to decoder 1. An interleaver and a deinterleaver are inserted between decoders 1 and 2 so as to match the encoding process. The information exchange between decoder 1 and decoder 2 is iterated for a given number of times; after the last iteration, $Le12$ and $Le21$ are used to estimate the systematic bits. Note that the turbo decoder is also able to exploit the a priori source information, that is, the probability P_0. This is very important in order to achieve a good compression ratio, and is obtained by modifying the extrinsic information of each the two recursive convolutional decoders;

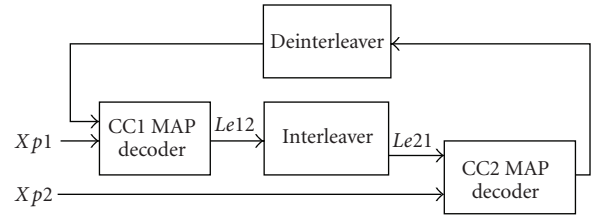

Figure 2: Block diagram of the turbo decoder.

in particular, the extrinsic information is modified by adding the term $\log((1 - P_0)/P_0)$.

3.3. Detailed algorithm implementation and setup

For the Algorithm 2, we have employed a turbo code based on two rate-1/2 16-state constituent convolutional coders, using a uniform random interleaver. The turbo code software we have used is an amended version of the software, distributed with [15] and freely available on the Internet at www.the-art-of-ecc.com, which implements the punctured parallel turbo encoder and decoder as described by the pseudocode in [14]. The constituent convolutional codes have rate-1/2
generator (31,27) octal (16 states), and the interleaver is random. The turbo decoder is a BCJR decoder with a limit of 15 iterations. This turbo code has been improved as follows.

(i) Since we need to select the bit rate with high precision, we have modified the available regular puncturing pattern implementing a pseudorandom puncturing pattern.

(ii) Since, in addition to channel coding and decoding, the turbo code also performs compression, the BCJR decoder has been improved by adding the a priori term to the extrinsic information. In particular, denoting as L_k^i the extrinsic information on X_k produced in constituent decoder D_i, with $i \in \{1, 2\}$, the modified extrinsic information is $L_k^i + \log((1 - P_0)/P_0)$.

3.4. Security

The security of the proposed scheme lies in the fact that the unauthorized decoder does not know some information which is necessary for decoding; this is akin to the system proposed in [4], where a pseudorandom interval swapping is used to desynchronize the decoder. In general, this kind of systems are potentially prone to chosen-plaintext attacks, since it is often possible to come up with specific input sequences that will bring the encoder in a known state from which it is possible to estimate the key. However, these algorithms are generally robust towards known-plaintext attacks. A typical chosen-plaintext attack to channel-coding based encryption schemes needs to estimate the key from chosen input-output pairs [16]. In [8], this is shown to be extremely difficult; in the proposed scheme, we have added the scrambler block so as to make the system secure against attacks that aim at estimating the code structure using multiple images

encoded with the same interleaver and puncturation. In [8], it is also shown that the ciphertext-only attack to turbo codes is an NP-complete problem if the puncturation matrix is not known. Obviously, the lack of knowledge of the interleaver along with the presence of the scrambler improve the security of the proposed system even further with respect to [8].

It should be noted that, in turbo coding, the interleaver is designed to improve the coding performance, and not to guarantee secrecy. The proposed scheme does not rely on the interleaver to achieve secrecy, but rather on the puncturation pattern and on the scrambler, while the interleaver simply adds another intermediate level of scrambling. This is necessary in order to prevent attacks based on estimation of the interleaver itself. If the interleaver was truly random, it would be extremely difficult for an attacker to estimate it using other than a brute force search. However, some interleavers like S-random do exhibit some structure that could potentially be exploited by an attacker. Note that the interleaver acts as a "diffusion" step, which helps improving the secrecy of the overall system.

4. RESULTS

We tested the proposed schemes using ideal binary sources transmitted over a binary symmetric channel with transition probability p. In the following, we refer to the algorithm based on arithmetic codes as Algorithm 1, and the algorithm based on turbo codes as Algorithm 2.

For both algorithms, the results have been averaged over 1000 runs with block size $L = 2000$, hence $2 \cdot 10^6$ bits have been simulated.

Table 1 reports the residual bit error rate (BER) of the decoded data block when the BER on the transmission channel is equal to $p = 5 \cdot 10^{-3}$. The rate R_C represents the equivalent rate of a channel code applied after compression. That is, for a block size equal to L, the number of bits transmitted on the channel is equal to $L \cdot H \cdot R_C$, where H is the entropy of the source. The source has prior probability $P_0 = 0.8$.

As can be seen from Table 1, a decoder with knowledge of the private key is able to reconstruct the data with a small error provided that the rate is sufficiently high.

For $R_C = 8/9$, Algorithm 2 does not work. Note that $R_C = 8/9$ means that with $P_0 = 0.8$, only 0.8122 bits per sample (bps) are transmitted on the channel, as compared to the rate of 1 bps for the original data. For $R_C = 4/5$, the bit rate is 0.9024 bps; the performance is still not good, although the decoder halves the error rate with respect to the previous case. Error-free transmission is obtained with $R_C = 2/3$, which corresponds to a bit rate of 1.0829 bps. These results reflect a typical behavior of turbo codes; when the BER is below a certain threshold, almost all errors are corrected, whereas a very large error probability is obtained in the opposite case. For Algorithm 2, even with a relatively high-channel BER ($p = 5 \cdot 10^{-3}$), a minimum overhead is required to obtain compression, error correction, and secrecy; in fact, it can be seen from the previous example that, if the probability of the least probable symbol is sufficiently small, compression based on channel codes turns out to be very effective. As can be seen from Table 1, Algorithm 1 provides better results for small levels of redundancy, and slightly worse results at higher levels, for example, $R_C = 2/3$.

Table 1: Residual BER for block size $L = 2000$ and prior probability $P_0 = 0.8$, as obtained by a decoder with knowledge of the private key.

R_C	Algorithm 1	Algorithm 2
8/9	0.038	0.191614
4/5	0.0057	0.099489
2/3	$1.6 \cdot 10^{-5}$	$<0.5 \cdot 10^{-6}$

Table 2: Residual BER for block size $L = 2000$ and prior probability $P_0 = 0.9$, as obtained by a decoder with knowledge of the private key.

R_C	Algorithm 1	Algorithm 2
8/9	0.014	0.091261
4/5	0.0014	0.043532
2/3	$2.2 \cdot 10^{-5}$	$<0.5 \cdot 10^{-6}$

Table 3: Residual BER for block size $L = 2000$ and prior probability $P_0 = 0.5$, as obtained by a decoder with knowledge of the private key.

R_C	Algorithm 1	Algorithm 2
8/9	$7.4 \cdot 10^{-2}$	0.496
4/5	$8.1 \cdot 10^{-3}$	0.496
2/3	$1.7 \cdot 10^{-4}$	0.483

Table 2 shows the results of a similar simulation, with different prior probability $P_0 = 0.9$. The results are very similar to the previous case. For $R_C = 8/9$, only 0.5276 bps are transmitted on the channel, as compared to the rate of 1 bps for the original data. For $R_C = 4/5$, the bit-rate is 0.5862 bps, and for $R_C = 2/3$, it is 0.7035 bps. As in the previous case, Algorithm 2 does not work at low redundancy, and works very well after a certain threshold, while Algorithm 1 works well at all redundancies, though slightly worse than Algorithm 2 at $R_C = 2/3$. Out of the 6000 decoding attempts whose results are reported in Tables 1 and 2, Algorithm 1 yielded 7 decoding failures.

We also tested the algorithms for a symmetric source with probability $P_0 = 0.5$. This case is representative of a source that is not compressible, but still has to be secured and transmitted. In general, the results, shown in Table 3, are worse than for $P_0 = 0.8$ and $P_0 = 0.9$; since this source is not compressible, the amount of rate left for redundancy is smaller and, as a consequence, the residual BER is higher. This is especially apparent for Algorithm 2, since even at $R_C = 2/3$, the turbo code is in the error floor and does not correct the errors. It takes a slightly more powerful code, for example, $R_C = 0.6$, for the turbo code to start correcting errors. The performance degradation for Algorithm 1 is much less dramatic, though the algorithm indeed performs worse than that with higher P_0.

The following experiments show the results obtained by a decoder that attempts to recover the original string without knowing the key.

Table 4: Residual BER for block size $L = 2000$ and different prior probabilities, as obtained by a decoder that does not have knowledge of the interleaver.

R_C	$P_0 = 0.8$	$P_0 = 0.9$
8/9	0.196	0.097
4/5	0.1915	0.0953
2/3	0.1855	0.089

Table 5: Residual BER for block size $L = 2000$ and different prior probabilities, as obtained by a decoder that does not have knowledge of the puncturation pattern.

R_C	$P_0 = 0.8$	$P_0 = 0.9$
8/9	0.203	0.1013
4/5	0.2049	0.1018
2/3	0.216	0.1038

We first present the results for Algorithm 2. Table 4 shows the effect of decoding the binary string with knowledge of the puncturation pattern and the scrambler, but without knowledge of the interleaver. The decoder generates a random interleaver different from that used by the encoder and attempts to decode the string. As can be seen, the residual BER for these attempts is always very high and typically close to $\min(P_0, 1 - P_0)$. This means that the decoder here is only exploiting the prior source information in order to limit the error probability. A possible improvement, which is left for further work, is to modify the decoder so that it estimates the prior probabilities from the data, as suggested in [3]; in this case, the estimation obtained using a wrong interleaver would be highly inaccurate, and the residual BER would likely be close to 0.5.

Table 5 shows the results of a similar test, assuming that the decoder knows the scrambling pattern and the interleaver, but does not know the puncturation pattern. The results are very similar to the previous case with the residual BER being very high.

For Algorithm 1, similar results are obtained. The decoding process of Algorithm 1 is complicated by the event of a decoding failure, which becomes very likely when the probability set is mismatched, as happens when one tries to decode a randomized AC codeword without knowing the key. The following setups have been tested, and all of them yielded 100% of decoding failures. We denote by $\hat{\varepsilon}$ and $\hat{P}_0$ the probabilities used by the decoder as follows:

(i) $\hat{\varepsilon} > \varepsilon$;
(ii) $\hat{\varepsilon} = 0.99 \cdot \varepsilon$;
(iii) $\hat{\varepsilon} = 0$;
(iv) $\hat{P}_0 = 0.95 \cdot P_0$.

In one setting, we have set P_0 to the correct value and $\hat{\varepsilon} = 0.5 \cdot \varepsilon$; this tests the security of Algorithm 1 when the intervals are multiplied by powers of two, which is a more critical case for the AC. In this simulation, 50% of the decoding attempts yielded a failure; the residual BER for the successful attempts is reported in Table 6; as can be seen, even when the decoding is successful, Algorithm 1 yields very high BER.

Table 6: Residual BER of Algorithm 1 for block size $L = 2000$, P_0 known, and $\hat{\varepsilon} = 0.5 \cdot \varepsilon$.

R_C	$P_0 = 0.8$	$P_0 = 0.9$
8/9	0.331	0.192
4/5	0.335	0.201
2/3	0.347	0.214

5. CONCLUSIONS

We have proposed two practical algorithms able to perform jointly source coding, channel coding, and secrecy. Algorithm 1 is based on an arithmetic coder with forbidden symbol and interval randomization coupled with a soft decoder; Algorithm 2 is based on turbo codes, which generate parity bits for error protection and can be used for compression by puncturing the output. Secrecy is achieved by hiding to the unauthorized decoder, the parameters of the encoding process, that is, the interleaver, the puncturation pattern, and the scrambling pattern.

It has been shown that, given a reasonable amount of redundancy, both algorithms provide good performance under quality metric for protection, distortion, and secrecy. Algorithm 2 requires a minimum level of redundancy, after which its performance is near perfect, whereas Algorithm 1 also works at lower-redundancy levels.

The proposed schemes eliminate the need for external compression and ciphering blocks, and hence reduce the system complexity, which is a clear advantage for power-limited wireless applications. In particular, the adoption of Algorithm 1 requires trivial modifications of an existing compression algorithm, for example, JPEG 2000 or H.264/AVC.

REFERENCES

[1] A. Mohr, E. Riskin, and R. Ladner, "Unequal loss protection: graceful degradation of image quality over packet erasure channels through forward error correction," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 6, pp. 819–828, 2000.

[2] M. Grangetto, P. Cosman, and G. Olmo, "Joint source/channel coding and MAP decoding of arithmetic codes," *IEEE Transactions on Communications*, vol. 53, no. 6, pp. 1007–1016, 2005.

[3] J. Garcia-Frias and Y. Zhao, "Compression of binary memoryless sources using punctured turbo codes," *IEEE Communications Letters*, vol. 6, no. 9, pp. 394–396, 2002.

[4] M. Grangetto, E. Magli, and G. Olmo, "Multimedia selective encryption by means of randomized arithmetic coding," *IEEE Transactions on Multimedia*, vol. 8, no. 5, pp. 905–917, 2006.

[5] C.-P. Wu and C.-C. J. Kuo, "Design of integrated multimedia compression and encryption systems," *IEEE Transactions on Multimedia*, vol. 7, no. 5, pp. 828–839, 2005.

[6] M. Johnson, P. Ishwar, V. Prabhakaran, D. Schonberg, and K. Ramchandran, "On compressing encrypted data," *IEEE Transactions on Signal Processing*, vol. 52, no. 10, pp. 2992–3006, 2004.

[7] R. J. McEliece, "A public-key cryptosystem based on algebraic coding theory," DSN Progress Report 42-44, pp. 114–116, Jet Propulsion Laboratory, Pasadena, Calif, USA, January-February 1978.

[8] A. Payandeh, M. Ahmadian, and M. Reza Aref, "Adaptive secure channel coding based on punctured turbo codes," *IEE Proceedings: Communications*, vol. 153, no. 2, pp. 313–316, 2006.
[9] L. Xu, "A general encryption scheme based on MDS code," in *Proceedings IEEE International Symposium on Information Theory (ISIT '03)*, p. 19, Yokohama, Japan, June-July 2003.
[10] J. B. Anderson and S. Mohan, *Source and Channel Coding*, Kluwer Academic Publishers, Norwell, Mass, USA, 1991.
[11] P. W. Moo and X. Wu, "Resynchronization properties of arithmetic coding," in *Proceedings of IEEE International Conference on Image Processing (ICIP '99)*, vol. 2, pp. 545–549, Kobe, Japan, October 1999.
[12] J. G. Cleary, S. A. Irvine, and I. Rinsma-Melchert, "On the insecurity of arithmetic coding," *Computers and Security*, vol. 14, no. 2, pp. 167–180, 1995.
[13] H. Kim, J. Wen, and J. D. Villasenor, "Secure arithmetic coding," *IEEE Transactions on Signal Processing*, vol. 55, no. 5, pp. 2263–2272, 2007.
[14] W. E. Ryan, "A turbo code tutorial," 1997, http://www.ece.arizona.edu/~ryan/.
[15] R. H. Morelos-Zaragoza, *The Art of Error Correcting Coding*, John Wiley & Sons, New York, NY, USA, 2nd edition, 2006.
[16] T. R. N. Rao and K.-H. Nam, "Private-key algebraic-code encryptions," *IEEE Transactions on Information Theory*, vol. 35, no. 4, pp. 829–833, 1989.