Towards Compression of Encrypted Images and Video Sequences

Daniel Schonberg, Member, IEEE, Stark C. Draper, Member, IEEE, Chuohao Ye, Student Member, IEEE, and Kannan Ramchandran, Fellow, IEEE

Abstract—We present a framework for compressing encrypted media such as images and videos. Encryption masks the source, rendering traditional compression algorithms ineffective. By conceiving of the problem as one of distributed source coding, it has been shown in previous work that encrypted data is as compressible as unencrypted data. However, there are two major challenges to realizing these theoretical results. First is the development of models that both capture the underlying statistical structure and are compatible with our framework. Second is that because the source is masked by encryption, the compressor cannot know what rate to target.

We tackle these issues in this work. We first develop statistical models for images before extending it to videos, where our techniques really gain traction. As illustration, we compare our results to a state-of-the-art motion-compensated loss less techniques really gain traction. As illustration, we compare our results to a state-of-the-art motion-compensated loss less techniques really gain traction. As illustration, we compare our results to a state-of-the-art motion-compensated loss less techniques really gain traction. As illustration, we compare our results to a state-of-the-art motion-compensated loss less techniques really gain traction.

I. INTRODUCTION

Encryption masks digital content so that it appears completely random, and thus renders traditional compression algorithms ineffective. Best practices therefore dictate that content must be compressed before it is encrypted. Unfortunately best practices in compression and security cannot be assumed. This motivates the search for novel compression routines that operate on uncompressed, encrypted data.

At first such a search seems quixotic. Ideally, encrypted data is independent of the source and is marginally (in the statistical sense) incompressible. However, even under these ideal assumptions, in [2] it is shown that in certain cases (when a stream-cipher is used) the compression problem can be posed as a distributed source-coding problem in the sense of Slepian and Wolf [3]. This makes post-encryption compression possible. Furthermore, theoretically, neither compression performance nor security need be compromised by swapping the order.

As a motivating application, consider the delivery of high-definition video over home wireless networks. On one hand are content providers who have strict security requirements. To meet these requirements while maintaining the highest quality video, content providers often encrypt their raw (uncompressed) content prior to distribution. For example, the High-Bandwidth Digital Content Protection (HDCP) standard [1] requires the encryption of raw video content. On the other hand are wireless infrastructure providers. The transport capacity of wireless networks is limited, and the data rate of raw high-definition video will overwhelm communication links. This is therefore an ideal setting in which to apply the framework developed in [2], satisfying the security concerns of the content providers and the bandwidth constraints of the infrastructure providers.

However, there are a number of real-world challenges facing any constructive realization of the theoretical framework of [2]. In this paper we identify two main challenges, describe the types of solutions required, and develop prototypes for image and video content. The first challenge is the development of good statistical source models that are required by the joint decryption/decompression process. The appropriateness of the model has a large impact on compression performance since the compression rates achieved cannot exceed the limits of the model. The second challenge is statistics-blind operation of the compression system. Since encryption masks the source, the compressor cannot calculate empirical statistics as is done, e.g., in universal compression. Without knowledge of source statistics, the compressor cannot know at what compression rate to operate. By leveraging a small amount of destination-source feedback we develop a protocol that asymptotically approaches the results of [2].

A. High-level system description and prior work

To illustrate the system architecture developed in [2], consider the process depicted in Fig. 1. In the left-hand plot is a frame from the “Foreman” video sequence. This unencrypted plain-text image $x^n$ is described by $n$ bits (equal to the number of pixels times the number of bits per pixel – i.e., $x^n$ is a binary-sequence of length $n$). The middle plot of the figure shows a key sequence $k^n$ of the same dimension as the plain-text, consisting of bits selected uniformly at
random (an independent and identically distributed (i.i.d.) Bernoulli-0.5 sequence). This is an example of a ‘‘one-time pad’’ stream-cipher. The image is encrypted by applying a bitwise exclusive-OR (XOR) between each key bit and the corresponding plain-text bit to produce the cipher-text \( y^n = x^n \oplus k^n \), shown in the right-hand plot. Under the key-generation assumptions, this system offers perfect security [4]. Compression of the highly structured plain-text image has been well studied. However, none of these techniques can successfully compress the marginally random cipher-text. The cipher-text is a Bernoulli-0.5 i.i.d. image and is therefore not compressible without other information about the source.

While the plain-text and cipher-text are statistically independent, the cipher-text and stream-cipher are not. This insight is crucial. Since the stream-cipher is known to the decoder, compression is possible by leveraging the dependence between cipher-text and stream-cipher. The compression and reconstruction/decryption process can be understood by viewing the cipher-text \( y^n \) as a noisy version of the stream-cipher \( k^n \) where \( y^n = x^n \oplus k^n \). The ‘‘noise’’ is the plain-text data \( x^n \).

As a simple example of the insight that leads to the results in [2], suppose the plain-text \( x^n \) has a low Hamming-weight.\(^1\) The set of most-likely cipher-texts are then clustered around the key sequence \( k^n \). The key sequence thus serves as a noisy observation of the cipher-text, and based on the realization of \( k^n \), the decoder can narrow the set of possible cipher-texts observed by the encoder. Because the decoder has a small set of possible cipher-texts, the encoder can get away with sending a non-unique description of the cipher-text that, in effect, describes a list of possible (encrypted) source sequences. The decoder identifies the correct source sequence by intersecting its list of possibilities with the encoder’s list.

A simplified block diagram of our approach is depicted in Fig. 2. The length \( n \) plain-text source \( x^n \) is encrypted with the stream-cipher \( k^n \) to produce the cipher-text \( y^n \). The cipher-text is compressed into the \( nR \) \((R < 1)\) parity constraints. This vector of bits is referred to as the syndrome (due to the connection to error-correction coding). The decoder estimates \( \hat{x}^n \) based on the syndrome and the stream-cipher \( k^n \). In standard Slepian-Wolf terminology the key sequence is the ‘‘side-information’’ available to the decoder.

In [2] the connection between compression of encrypted data and Slepian-Wolf coding is established. In addition, a graphical-code based implementation for the compression of an encrypted i.i.d. source is presented. However, for the media coding applications that we envision, an i.i.d. model does a poor job of capturing source characteristics. Furthermore, [2] assumes prior knowledge of the source statistics. In practice the source is masked by encryption, so the compressor cannot assume such knowledge. This paper takes first steps in addressing each of these two limitations.

B. Main results & outline

In this work, we develop a framework for the compression of encrypted media sources. The general framework is presented in Section II, along with the choices for encryption and compression we use. A crucial aspect of making these ideas work is the development of source models that both capture important features of the underlying source and are amenable to decoding. In Section III we consider the design of a statistical source model for image content. As previous work [2] focuses on memoryless models that ignore spatial and temporal dependencies, our contribution is the development and implementation of practical models for compressing encrypted images. We present results for both binary and gray scale images, demonstrating that these models achieve greater compression than i.i.d. models.

While for conceptual simplicity we first demonstrate improvements for images, the temporal redundancy of video is more amenable to our methods. Not only is encrypted video better matched to our techniques, but because of the much higher raw data rates it is also in greater need of a solution. Operating on a frame by frame basis, video offers the decoder the opportunity to combine intra-frame spatial statistical models with inter-frame predictive temporal models. In Section IV we present a proof-of-concept implementation and evaluate its performance on standard video sequences.

Finally, in Section V we address the lack of \textit{a priori} knowledge of source statistics. We present and analyze a protocol that exploits low-rate feedback to enable the compressor to adapt its rate to the empirical rate of the (masked) source model. Our analysis, presented in the Appendix, shows that our design converges to the model entropy-rate of a stationary source. Finally, we incorporate blind compression into the developed framework to demonstrate a complete automatic system for compressing encrypted video.

\(^1\)That is, it is mostly zeros and contains only a few ones. Although real-world media source do not necessarily have a low Hamming weight, images and video do contain sparse structure that can be exploited in a similar manner.
II. General Framework

In this section we present our framework and specialize to the scenario of interest (stream-cipher encryption and compression using linear codes). A block-diagram depicting the quantities of interest is given in Fig. 3. The left half of the factor graph depicts the generative model (the mapping from plain-text source and key to syndrome), while the right half is used by the decoder to estimate the plain-text.

The generative model contains the mapping from the source\(^2\) \(x^n\) and key \(k^n\) through encryption into cipher-text \(y^n\). The “encryption function” block constrains \(x^n\), \(k^n\) and \(y^n\) to be compatible inputs and outputs of the encryption function. The generative model also includes the function mapping between cipher-text \(y^n\) and syndrome \(s^n\). The compression rate is \(m/n\). The “code constraint” block constrains \(y^n\) and \(s^n\) to be compatible inputs and outputs of the encoding function. The right-half of the figure is used by the decoder to estimate the plain-text. It takes as inputs the key \(k^n\) and syndrome \(s^n\) and outputs the source and cipher-text estimates, \(\hat{x}^n\) and \(\hat{y}^n\), that are modeled as hidden variables.

In our work, we use the formalism of factor graphs [5] to decompose the high-level blocks of Fig. 3 into more tractable low-level functions representing statistical relationships between quantities of interest. Factor graphs not only concisely represent the statistical structure of an inference problem, but also provide the basis for efficient estimation algorithms. Factor graphs consist of two types of nodes, variable nodes (represented by circles) and function nodes (represented by rectangles or squares). Variable nodes represent constraints between subsets of variable nodes. The set of variable nodes constrained by a particular function relationship are connected to the function node by edges.

A. Encryption Function

In this paper we exclusively consider stream-cipher encryption. Such encryption is used in a variety of symmetric key encryption standards, and (under our assumption of an i.i.d. Bernoulli-0.5 key) are Shannonsense secure [4]. The structure of stream-ciphers allows us to factor the larger encryption process into a set of bit-by-bit relationships. The cipher-text is computed as \(y_i = \hat{x}_i \oplus k_i\), where \(\hat{x}_i\) is the \(i\)th bit of the plain-text, \(k_i\) is the \(i\)th bit of the key, and \(\oplus\) indicates addition mod-2 (XOR).

At the decoder, a parallel constraint must be enforced on the cipher-text estimate, plain-text estimate, and key: \(\hat{y}_i, \hat{x}_i,\) and \(k_i\) for every \(i\). In particular, \(\hat{x}_i = \hat{y}_i \oplus k_i\).

We split the large encryption factor node into \(n\) low-level factor nodes. As depicted in Fig. 4, the square node labeled \(f_i\) enforces even parity, i.e., \(\hat{x}_i \oplus \hat{y}_i \oplus k_i = 0\).

B. Code Constraint

We use a linear transform to compress the cipher-text. The syndrome is calculated as

\[
s^n = H y^n
\]

where \(H\) is an \(m \times n\) binary matrix, and addition is mod-2.

As with the encryption function, the “code constraint” function node of Fig. 3 can be factored into \(m\) local functional constraints each corresponding to a row of (1). To represent the constraints graphically, we draw an edge connecting the \(i\)th variable \(\hat{y}_i\) to the \(j\)th parity constraint \(g_j\) if there is a non-zero (i.e., unity) entry in the \(i\)th column and \(j\)th row of \(H\). During decoding the \(i\)th function node constrains the mod-2 sum of all \(\hat{y}_i\) connected to \(g_j\) to equal \(s_j\). This limits the search space of possible cipher-text sequences to the sequences that satisfy the parity constraints. An example is shown in Fig. 5.

C. Source and Source Model

Unlike the encryption and compression functions, where exact description of the imposed constraints is possible, we do not have exact knowledge of the underlying characteristics

\(\hat{m}/n\)

Fig. 4. The graphical constraints imposed by stream-cipher encryption. The plain-text source estimate \(\hat{x}_i\), the key bit \(k_i\), and the cipher-text estimate \(\hat{y}_i\) must have even parity.

Fig. 5. The graphical model of the constraints imposed by using a LDPC code for compression. All cipher-text bit estimates \(\hat{y}_i\) connected to the same parity-check \(g_j\) must sum to the syndrome value \(s_j\).

In this paper we choose the linear transform \(H\) to be parity-check matrix of a low-density parity-check (LDPC) code [6]. The \(H\) matrix corresponding to a LDPC code is a sparse matrix, i.e., it has few ones in any column or row. This sparsity makes efficient calculation of \(\hat{y}^n\) possible. Further discussion of the LDPC codes we use and how we modify them is provided in our experimental descriptions in Section III-D. See [7] for more extensive discussion of code design and the merits of using LDPC codes for compression.

In this framework, choosing the number of syndrome bits \(m\) to transmit is an issue. This choice depends on the statistics of the source according to the model in use. How to choose the least rate even when the statistics are unknown is the subject of Section V.
of the source. Instead, the decoder must use a model of the source. Thus, in Fig. 3 the left-hand side function is labeled “source” (determined by nature) while the corresponding function on the right-hand side is labeled “source model” (chosen by the system designer). The compression rate our system can achieve depends on how well our source model captures the source characteristics. In this paper we study a variety of models.

We first discuss the simple $n$-bit i.i.d. model used in [2]. We defer discussion of more sophisticated models to Sections III and IV. The factor graph corresponding to the i.i.d. model is represented in Fig. 6. This graph consists of the $n$ variable nodes corresponding to source bits estimates, labeled $\hat{x}_i$, and the $n$ function nodes corresponding to the source priors, labeled $P_i$. The source prior $P_i$ is the marginal probability that each bit is equal to one, denoted $P_i = \Pr(\hat{x}_i = 1)$. For now, assume the decoder is supplied with knowledge of $p$. We show how to relax this assumption in Section III-D.

![Fig. 6. The i.i.d. source model considered in [2]. This model consists of the source estimates $\hat{x}_i$ and their priors.](image)

D. Algorithm

The objective of the decoding algorithm is to find the maximum a-posteriori estimate of $x^n$ given $s^n$ and $k^n$. We calculate our estimate using the sum-product message passing algorithm [6], [5]. This algorithm is an inference algorithm that works on the factor graph. When the edges and nodes forming the factor graph contain no loops (i.e., the graph is a tree), the sum-product algorithm exactly computes the posterior distribution of $x^n$. When there are loops in the graph (such as in the code graph of Fig. 5) the sum-product algorithm yields only approximate estimates of the posteriors. However, for many applications it achieves good empirical performance at reasonable computational cost (the number of calculations grows polynomially with the block length).

The sum-product algorithm works by iteratively sending messages between nodes along the edges of the factor graph. Each message pertains to a single variable node and represents the algorithm’s current belief of whether its value is zero or one. (Recall that all variables in our system are binary.) For generic binary variable $x_i$, we therefore use log-likelihood ratios $\log[(1 - \Pr(x_i = 1))/\Pr(x_i = 1)]$ to represent beliefs.

By using this parametrization we only need to propagate scalar messages. Incoming messages are fused together to produce updated beliefs. Over a number of message passing iterations, statistical information is disseminated across the graph. The algorithm continues until either a stopping condition is met, e.g. if our current estimate satisfies $s^n = Hy^n$, or a maximum number of iterations, e.g. 100, is reached. The maximum number of iterations is set to insure successful decoding in most cases.

III. FROM I.I.D. MODELS TO 1-D AND 2-D MODELS WITH MEMORY TO IMAGES

As discussed earlier, matching our source model to the true nature of the source is essential to improve performance. Each source model we discuss is intended for use with the factor graph of Fig. 3. Each model is described in terms of the
decoder half (right-hand side) of Fig. 3, and is of increasing complexity with better compression performance. Since we focus on bit-wise stream ciphers in Section II-A, each model is also bit-based. As we describe each model, we illustrate them using factor graphs.

A. 1-D Markov Model

The first extension of the i.i.d. model we consider is to a Markov dependency between successive bits. The factor graph of our binary bit-plane 1-D Markov model is shown in Fig. 7. In this figure, we again consider the \( n \) bits, labeled \( \hat{x}_i \), and the source priors, labeled \( P_i \). Additionally, we now consider the correlation between consecutive bits, shown as the constraints labeled \( M_i \). The correlation constraints represent the Markov state transitions. In addition to parametrization the source priors with the marginal probability \( P_i = \Pr(\hat{x}_i = 1) \), we now also parametrize the Markov correlation constraints. Assuming these correlations to be equivalent both forwards and backwards, we denote these correlations \( h_0 = \Pr(\hat{x}_i = 1|\hat{x}_{i-1} = 0) = \Pr(\hat{x}_i = 1|\hat{x}_{i+1} = 0) \) and \( h_1 = \Pr(\hat{x}_i = 1|\hat{x}_{i-1} = 1) = \Pr(\hat{x}_i = 1|\hat{x}_{i+1} = 1) \).

```
\begin{align*}
W & \hat{x}_1 \quad \hat{x}_2 \quad \hat{x}_3 \quad \cdots \quad \hat{x}_{i-1} \quad \hat{x}_i \quad \hat{x}_{i+1} \\
& P_{i-1} \quad P_i \quad P_{i+1}
\end{align*}
```

Fig. 7. The binary bit-plane 1-D source model. This model consists of bits (circles), source priors (squares below the bits) and correlation constraints (squares between the bits). Messages, \( \mu \) and \( \nu \), are probabilistic beliefs about the source nodes which are exchanged for inference and are described (2) and in Section III-C.

B. 2-D Markov Model

In order to capture the spatial correlations of images, we introduce a binary bit-plane 2-D Markov model\(^3\). Consider \( n \) bits as arranged according to a grid, with \( N_v \) rows and \( N_h \) columns (\( N_v \times N_h = n \)). In addition to the source prior, we consider the correlation between each pixel \( \hat{x}_{i,j} \) and its 4 nearest neighbors; up & down, left & right. A section of the corresponding factor graph is illustrated in Fig. 8. Besides the circles labeled \( \hat{x}_{i,j} \) (\( i \in \{1, \ldots, N_v\} \) and \( j \in \{1, \ldots, N_h\} \)) representing the bits and the squares labeled \( P_{i,j} \) representing the source priors, the constraints labeled \( M_{i,j}^h \) represent the horizontal correlations while the those labeled \( M_{i,j}^v \) represent the vertical correlations.

As with the other models, the prior probability on each bit is denoted \( P_{i,j} = \Pr(\hat{x}_{i,j} = 1) \). For the 2-D model, we denote the horizontal correlation parameters as \( h_0 = \Pr(\hat{x}_{i,j} = 1|\hat{x}_{i,j-1} = 0) = \Pr(\hat{x}_{i,j} = 1|\hat{x}_{i,j+1} = 0) \) and \( h_1 = \Pr(\hat{x}_{i,j} = 1|\hat{x}_{i,j-1} = 1) = \Pr(\hat{x}_{i,j} = 1|\hat{x}_{i,j+1} = 1) \). We respectively denote the vertical correlation parameters as \( v_0 \) and \( v_1 \) for the 2-D model.

\(^3\)The constructions of [9], [10] are related but consider an image and a noisy version of the same image. Since neither the key nor the cipher-text are “images” here, their constructions do not directly apply.

C. Message Passing Rules

Incorporating the models above into the general framework of Fig. 3 is straightforward. In this section, we give the form of the additional message update rules necessary that were not covered in Section II-D. There are two additional types of nodes considered here; the source prior and the correlation constraints. Since each source prior is a terminal node, the node’s messages are constant across iterations and the messages are the log-likelihood ratio of the bit according to the prior, \( p \).

As a representative example, we write the update rule for the 1-D model (Fig. 7) from \( M_2 \) to \( \hat{x}_2 \) (labeled \( \mu \)) based on the message from \( \hat{x}_1 \) to \( M_2 \) (labeled \( \nu \)). All other correlation constraint message updates (messages in the alternate direction as well the messages update rules for \( M_{i,j}^h \) and \( M_{i,j}^v \)) are identical in form. Recall the parameters for \( M_2 \) here; \( h_0 = \Pr(\hat{x}_i = 1|\hat{x}_{i-1} = 0) \) and \( h_1 = \Pr(\hat{x}_i = 1|\hat{x}_{i-1} = 1) \). The update equation is given below, derived using Bayes rule, the law of total probability, and based on the definition of the messages (see Section II-D and, e.g., [5]).

\[
\mu = \log \left( \frac{(2 - h_0 - h_1) + (h_1 - h_0) \tanh(\nu/2)}{(h_0 + h_1 - 1)(h_0 - h_1) + (h_0 - h_1) \tanh(\nu/2)} \right) \tag{2}
\]

\( M_2 \) thus converts the estimate of \( \hat{x}_1 \) to an estimate of \( \hat{x}_2 \), based on the correlation between \( \hat{x}_1 \) and \( \hat{x}_2 \).

D. Bit doping

As mentioned in Section II-B, we use modified LDPC codes to compress the encrypted sources. The compressor implements a sparse linear transformation based on a LDPC code parity-check matrix. The code is modified so that a fraction of the encrypted bits are not compressed. We refer to these uncompressed bits as “doped” bits. In image compression applications the doped bits range from 30% to 50% of the total output compressed bits. We use these bits in two ways.
First, since doped bits are known unambiguously by the decoder they anchor the iterative decoder and initiate the decoding process, biasing it towards the correct solution. Without having these bits as a catalyst, decoding fails to get started. Decoding fails since marginally each source bit is roughly uniform, and thus the log likelihood ratios lack bias. This aspect of their use is similar to their use in the distributed source coding scheme of Markov sources in [11] and fountain codes [12].

Second, they provide a mechanism to estimate the statistics of the data. As mentioned in Section II-D, we assume that the decoder knows the source statistics. Using doped bits, we can eliminate this assumption. By selecting an appropriate subset of the source to send as doped bits, the decoder develops estimates of the source parameters \((p, h_0, \text{etc.})\). For example, with Markov sources we dope adjacent bits for parameter estimation. In this work, we dope bits uniformly at random. Note that the doped bits could be incorporated into the code’s degree distribution since each is equivalent to adding a degree 1 check node to the sparse linear transform. How to optimally incorporate dope bits using code design techniques [8] is an open problem.

**E. Experimental Results**

We now give example and results for the 1-D and 2-D models of Sections III-B and III-B. Consider the binary map of the world depicted in Fig. 9. We encrypt this binary map image (10, 000 bits), compress it, and then decompress it using each model. To apply the 1-D Markov model we first raster scan (north to south, than east to west) the encrypted image. Through a sequence of trials, we determine the minimum rate we can compress the encrypted image to and still recover the original source exactly. We present those numbers as the maximum compressibility rate achieved by our techniques.

For these experiments, we generate a set of 38 LDPC codes with compression rates ranging from 0.03 to 0.95, with a spacing of approximately 0.025 between codes. Most codes correspond to parameters obtained from the binary symmetric channel design section of the LTHC website [21]. The rest of the codes we use are designed using extrinsic information transfer (EXIT) charts [22]. Short cycles of length four are removed by search. Although the degree distributions we use are designed for i.i.d. sources, they perform well in practice. We note that it should be possible to improve performance by optimizing the codes to the (non-i.i.d.) source model in use.

Our first attempt to decompress the map uses the i.i.d. model. Based on the doped bits we measure this image, under an i.i.d. source model, to have parameter \(P = 0.38\) and an entropy of 0.9580 bits. In practice, we are unable recover the source no matter what code rate we use to compress the encrypted image. Using the 1-D model, we estimate \((P, h_0, h_1) = (0.3738, 0.0399, 0.9331)\), and the encrypted image could be compressed to 7,710 bits. Finally, we compress the encrypted map image to 4,299 bits using the 2-D model, of which 2,019 are doped bits. The decoder empirically estimates

\[
(P, h_0, h_1, v_0, v_1) = (0.3935, 0.0594, 0.9132, 0.0420, 0.9295)
\]

The compressed bits and the reconstruction are presented in Fig. 10.

To demonstrate some effects of the source model on decoding, we present the estimates of the 1-D and 2-D decoders at three early iterations in Fig. 11. The influence of the 1-D source model is seen in the convergence along north-south lines. In contrast, when the 2-D source model is used the estimates converge in “clumped” regions.

**F. Gray Scale Images**

We now discuss the challenges posed by gray-scale image data whose pixel values range from 0 to 255. As with the 2-D model, pixels are assumed to lie on a grid with \(N_v\) rows and \(N_h\) columns. We use a bit plane decomposition of each pixel, with 8 bits per pixel and bit planes range from most to least significant. Due to the bitwise focus, we find it convenient to consider each bit plane in sequence, modeling each with the factor graph of Fig. 8.

In contrast to this model, we could use a pixel based model, matching the image more closely. Unfortunately, this would also result in a drastic increase in decoding complexity since the computations of the sum-product algorithm is exponential in the alphabet size. Developing the algorithm to operate efficiently with pixel based operations is an open problem.

Though the 1-D and 2-D models provide significant improvements over the i.i.d. model when considering binary images, extension of the models in this section to “real world” gray-scale images proves difficult. In practice we find the only
significant exploitable correlations to be in the two most significant bit planes. Even exploiting inter-bit plane correlations (as discussed in Section IV) provides little improvement in our experiments.

The two most common techniques used for compressing unencrypted gray scale images are inapplicable in compressing encrypted images. The first method is the application of decorrelating transforms, such as the DCT (Discrete Cosine Transform). Encrypted information is already decorrelated, as can be seen in Fig. 1. The transform of the encrypted image lacks the exploitable structure of the transform of the unencrypted image. The second method is the use of localized predictors. Without access to the local unencrypted context available to compressors of unencrypted images, localized predictors fail. In contrast, due to the high temporal correlation that is present, video offers greater opportunity for compression as we discuss next.

**IV. VIDEO CONTENT**

In this section, we consider encrypted gray-scale video. Video offers a larger opportunity for compression, due to temporal as well as spatial correlations. Since we restrict ourselves to bitwise stream ciphers, we develop a bitwise source model for video. We then describe adjustments to the decoding algorithm and present experimental results.

**A. Source Model**

We base our model for video on a sequence of the 2-D Markov model considered in Section III-B, one for each bit plane. We begin by relabeling each bit from Fig. 8 (using a decoder-focused notation) as \( \tilde{x}_{i,j}[l] \) to represent the \( l \)th most significant bit of the pixel at row \( i \) and column \( j \) of frame \( t \). A dropped index implies that we are considering the entire range over that index. For example, \( \tilde{x}_t[l] \) represents the entire \( l \)th bit plane of the frame \( t \), while \( \tilde{x}_t \) represents the entire frame at time \( t \) with elements ranging from 0 to 255 (since we include every bit plane).

We consider the following approach for exploiting the temporal correlations, diagrammed in Figures 12 and 13. Figure 12 shows how information is fed to the decoder for the estimate of \( \tilde{x}_t \). Our system works frame-by-frame. It uses the two previous frames (that have already been decompressed and decrypted) to estimate motion vectors and make a prediction of the current frame. That is, given frames \( \tilde{x}_t^{-2} \) and \( \tilde{x}_t^{-1} \) we generate a prediction \( \hat{x}_t = g(\tilde{x}_t^{-1}, \tilde{x}_t^{-2}) \) of frame \( t \). This function is represented by the “Predictor” boxes in Fig. 12.

The predictor implemented at the decoder operates across bit planes and uses motion compensation together with motion extrapolation to generate \( \hat{x}_t \). This process is illustrated in Figure 13. The estimate \( \tilde{x}_t^{-1} \) is divided into non-overlapping blocks of \( 8 \times 8 \) pixels. For each block, we perform motion estimation to find the best matching block in \( \tilde{x}_t^{-2} \). We estimate the motion vector for each block in \( \tilde{x}_t \) as the motion vector of the co-located block in \( \tilde{x}_t^{-1} \). As shown in Figure 13, we then use the appropriate predictor from \( \tilde{x}_t^{-1} \) as indicated by the estimated motion vector. We use the sum-of-squares difference as the metric to find the best matching block. This approach works best when motion fields change slowly between frames. More sophisticated prediction could lead to a better \( \hat{x}_t \) and better model performance.

The prediction \( \tilde{x}_t \) is useful for understanding the actual frame \( \tilde{x}_t \). To make appropriate use of the prediction we need to quantify its accuracy. To do this we calculate the empirical distribution \( P(\tilde{x}_t^{−1}, \tilde{x}_t^{-2}) \) of the previous frame and its prediction. These distributions are calculated on a pixel level. The distribution for individual bit planes can be determined from the pixel level distribution. We estimate the joint distribution of our current frame and its predictor as \( \tilde{P}(\tilde{x}_t, \tilde{x}_t^t) \approx P(\tilde{x}_t^t, \hat{x}_t) \). In Fig. 12, we represent this function with the box labeled “Pr(Decoded, Predicted).” In practice, when calculating the histogram for the previous frame and its predictor, we add it to the histogram (with a forgetting factor of one-half) used for decoding the prior frame. This results in greater estimate stability. For clarity we omit this from Fig. 12.

We consider one final source of side information for the decoder (as mentioned in Section III-F). Since we process the frame a single bit plane at a time, we can change to processing them in order from most to least significant. Consider adjacent pixels \( x_{i,j}^t = 127 \) and \( x_{i,j+1}^t = 128 \). If we consider only the bitwise expansion of these two pixels, we see that they may differ at every bit plane. Looking at each bit plane in isolation, the strong similarity between these two pixels can be missed. Conversely, by considering each bit plane in context of the bit planes of greater significance when using \( P(\tilde{x}^t, \hat{x}^t) \), we are able to exploit more relevant correlations.

The overall model consists of a predicted frame \( \tilde{x}_t^t \) based on the two previous frames and an empirical distribution \( \tilde{P}(\tilde{x}_t^t, \hat{x}_t) \) based on the previous frame and its predictor. We use this information to locally alter the parameters \( (p, h_0, h_1, v_0, v_1) \). We no longer assume them to be stationary; instead we make them dependent upon their context. For example, we rewrite \( p = p(\tilde{x}_{i,j}^t[l]|\tilde{x}_{i,j}^t[l], \tilde{x}_{i,j}^t[l-1], \ldots, \tilde{x}_{i,j}^t[l-1]) \). Specifically, we calculate each of these parameters using the marginalized empirical distribution \( \tilde{P}(\tilde{x}_t^t, \hat{x}_t) \) conditioned as
the predictor for the block with thickened edges in \( \hat{d} \) distributions. As with the image results, through a sequence of the previous prediction frame.

Sections V-C, are selected from a group of 38 different degree longer block length codes. If considered each frame as a whole and using a measure of prediction accuracy is generated by measuring the accuracy of the previous prediction frame.

In the above example, the best predictor for the block with thickened edges in \( \hat{x}^{t-1} \) is the lightly thickened box in \( \hat{x}^{t-2} \), shown with its corresponding motion vector \((v_x, v_y)\). For the motion vector of each block in frame \( \hat{x}^t \) we use the motion vector of the co-located block in frame \( \hat{x}^{t-1} \). In this example the predictor for the block with thickened edges in \( \hat{x}^t \) is the lightly thickened box in \( \hat{x}^{t-1} \), as pointed by the motion vector \((v_x, v_y)\).

Above.

B. Message Passing Rules

When belief propagation is run for each bit plane using the graph model we develop, the operation of the constraints is similar to the operation discussed in Section III-C. Since each source constraint, labeled \( P_{i,j}[l] \), is a terminal node, it transmits its log likelihood ratio calculated using the marginal probability \( p(x_{i,j}[l]|x_{i,j}^t, x_{i,j}^t[1], \ldots, x_{i,j}^t[l-1]) \). For the correlation nodes, labeled \( M^\|^i_{i,j}[l] \) and \( M^\|^i_{i,j}[l] \), message update rules are of the form in (2) but modified for the correlation parameters above.

C. Experimental Results

We evaluate our technique on the standard “Foreman,” “Garden,” and “Football” video sequences. Respectively, these are low-, high-, and high-motion sequences. We consider 12 encrypted frames (i.e., a group of pictures (GOP) size of 12) selected from each of these test sequences. The first three frames are used to initialize the predictors. Thus, we only compress frames 4 through 12 and only report compression results for these frames. For ease of implementation, we divide each frame into 9 regions (arranged \( 3 \times 3 \)). Each region, having 1/9 of the pixels, contains about \( 10^4 \) pixels and each bit plane contains about \( 10^4 \) bits. Some performance improvement would result from considering each frame as a whole and using longer block length codes.

The codes we use to compress the bit plane sections, as in Section V-C, are selected from a group of 38 different degree distributions. As with the image results, through a sequence of trials we determine the minimum rate we can compress the encrypted sections to and still recover the original source exactly. We aggregate the total number of compressed bits and present the results here.

As in Section III, we use bit doping. In contrast to Section III we now only require 5% to 10% of the total output to be doped bits. The reduced need for doped bits is due to the use of the predictors which bias the initial log likelihood ratios (as do channel outputs in channel coding), enabling inference to get started.

We present overall results in Table I. The entries indicate the average compression ratio (number of output bits divided by number of source bits). For example, 0.67 indicates that on average a 10000 bit source can be compressed to 6700 bits. As expected higher compression rates are achieved for the low-motion Foreman sequence than for the high-motion Garden or Football sequences. In the latter two cases, the motion vectors of the previous frame differ significantly from that of the current frame and hence yield lower quality predictors of the source pixels. Of course, second-order predictors that predict the “motion” in the motion vectors themselves would lead to better performance. We do not further explore such ideas here.

An example of the operations performed for a particular frame is depicted in Fig. 14. This figure demonstrates everything that goes into the decompression of frame 4 except for the encryption key, which is omitted for clarity. The encoder has access to only the encrypted version of frame 3. In contrast, the decoder is given the predicted frame 4, the probability model estimating the reliability of the predicted frame, and the compressed bits. In this example, the frame is
In Fig. 15, we present more detailed results from compressing the encrypted “Foreman” sequence. The vertical axis of these plots represent the compression ratio presented earlier (output bits per source bit). In the plot on the left (Fig. 15(a)), we plot the rate as a function of the frame number. Recall we make no effort to compress the first three frames. This plot shows that the overall performance varies as the video progresses but each frame is compressed to at least 70% of the source rate. In the plot on the right (Fig. 15(b)), we present the rate used for each bit plane (across the 9 compressed frames). The horizontal axis ranges from the most significant bit plane (1) at left to the least significant bit plane (8) at right. For reference, we are able to compress the most significant bit plane by 78% on average. Due to variations in the magnitude of the motion, for the most significant bit plane frame 12 is most compressible (81%) while frame 5 is least compressible (76%). This gives a good indication of how much correlation our system can exploit in each bit plane. As can be seen in this plot, we are able to obtain no compression gains in the two least significant bit planes of this sequence.

We compare the compression rates of our approach to those achieved by compressors that work on the unencrypted source data. As a first point of comparison we compress the unencrypted videos losslessly using JPEG-LS [13], [14]. This is pure intra-frame video compression which ignores temporal dependence. JPEG-LS not only has low encoding complexity, but also demonstrates exceptional compression performance relative to other lossless image coding standards [15]. For our study we use a publicly available JPEG-LS coder. As a second point of comparison we use a lossless video codec that does exploit temporal dependencies. We use the Moscow State University (MSU) lossless video codec. This codec is claimed by its authors to have the among the highest lossless video compression performance [16]. Due to its proprietary nature, the details of their video codec are not known. However, its performance seem to be comparable to past results in the literature that used either fixed spatio-temporal predictor [17] or motion compensation [18].

### V. System Operation Blind To Source Statistics

In this section we discuss how to implement a system for compressing encrypted data when source statistics are unknown. We present a protocol and analyze it for the simple case of unknown but i.i.d. source statistics. We discuss application of the protocol to dependent sources and demonstrate an implementation of the protocol for compressing encrypted video.

Our approach is motivated by the idea of incremental universal Slepian-Wolf coding developed in [19]. The basic insight there is that if a very small amount of destination to source feedback is available the decoder can adaptively determine the source statistics. While that work is in an information-theoretic setting where unlimited computational resources are assumed, in this section we show how to build a simple protocol that, asymptotically, delivers the same performance as that of [19]. In this section, we focus on explaining the protocol and demonstrating its practical aspects. We defer detailed analysis to the appendix.

### A. Protocol for i.i.d. sources

Let $x^n$ be a length-$n$ i.i.d. Bernoulli-$(Q)$ binary random variables, i.e., $\Pr[x_i = 1] = 1 - \Pr[x_i = 0] = Q$. The key is a one-time pad, an i.i.d. Bernoulli-$(0.5)$ sequence. The ciphertext is calculated as $y_i = x_i \oplus k_i$. As described in [2], by using Slepian-Wolf codes, one needs only transmit at a rate equal to $H(y|k) = H(x \oplus k|k) = H(x|k) = H(x) = H(Q)$.

---


where $H(Q)$ is the entropy of a Bernoulli-$\theta(Q)$ source. Thus if $Q$ is known, no source compressibility is lost because of the encryption.

To deal with the lack of knowledge of $Q$ we divide the source sequence into a number of blocks. Blocks are indexed by $k$ and are compressed and transmitted sequentially. Between each block, the compression system uses the feedback channel to give the encoder an estimate of $Q$. Over time the transmitter acquires an increasingly accurate estimate of $Q$ and can drive is compression rate to $H_B(Q)$. The encoder maintains a state variable, $Q_k$, its current estimate of the statistics of $x_n$, and a rate-margin design parameter $\epsilon_k > 0$. We set $\hat{Q}_0 = 0.5$. The system works as follows:

1. The encoder divides the encrypted sequence into blocks of length $l_k$, indexed by $k = 1, 2, \ldots$. We use $n_k = \sum_{j=1}^{k} l_j$ to denote the cumulative block-length. The $k$-th block corresponds to symbols $y_{n_k}^{n_{k+1}}$. By choosing the cumulative block-length to double every transmission, i.e., $n_k = 2n_{k-1}$, the error analysis simplifies somewhat.

We assume this choice for the duration.

2. At step $k$, the encoder encodes $y_{n_k}^{n_{k+1}}$ using a rate-$R_k$ Slepian-Wolf code, where $R_k$ depends on the estimate $\hat{Q}_k$ and margin $\epsilon_k$ as $R_k(\hat{Q}_k, \epsilon_k) =$

\[
\begin{cases} 
\max[H(\hat{Q}_k + \epsilon_k), H(\hat{Q}_k - \epsilon_k)], & \text{if } \epsilon_k < |\hat{Q}_k - 0.5|, \\
1, & \text{else}.
\end{cases}
\]

i.e., the rate used is $H(\hat{Q}_k)$ plus a margin. The various cases in (3) come into play depending on whether $\hat{Q}_k$ is greater than or less than one-half, or within the margin $\epsilon_k$ of one-half.\footnote{If we used the alternate choice $H(\hat{Q}_k) + \epsilon_k$, we could avoid the multiplicity of cases of (3). However, by expressing the margin in the current manner, our analysis simplifies, resulting in close-form error expressions.}

3. The decoder attempts to decode. If it can, it sends an acknowledgment to the transmitter, along with the updated estimate $\hat{Q}_k$. The estimate $\hat{Q}_k$ is simply the proportion of 1s (or 0s) observed thus far. The feedback following the $k$th block can therefore be accomplished with $1 + \log_2 l_k$ bits. If the decoder cannot decode reliably, it sends a “NAK”, i.e., a request for an additional transmission for the source block. It holds off sending the updated estimate until it receives the retransmission.\footnote{We assume the code has perfect error-detection capabilities which, by using a cyclic-redundancy check (CRC), is essentially trivial to ensure in practice.}

4) If the encoder receives an acknowledgment, it moves onto the next block, using the updated estimate of $Q$. If the encoder receives a “NAK”, it sends the sequence $y_{n_k}^{n_{k+1}}$ uncomprssed.\footnote{More efficient hybrid-ARQ-type retransmission strategies can be used [19], [20], e.g. send incrementally more parity bits over several rounds with feedback. The current assumption simplifies the error analysis and, for long sequences, leads to a negligible loss in efficiency.}

Let $c_k$ denote the cost (in bits per symbol) of transmitting block $k$. When block decoding succeeds, the cost is the rate of the Slepian-Wolf code. When block decoding fails, the cost is $\ln 2$ (i.e., uncompressed transmission – 1 bit/symbol or $\ln 2$ nats/symbol). Our objective is to minimize the expected cost $E[c_k]$ of the scheme in each block. We use the binary random variable $\epsilon_k$ to denote the event of a decoding error ($\epsilon_k = 1$) on block $k$. We set our parameter to minimize

\[
E[c_k] = \Pr[\epsilon_k = 1] \ln 2 + E[R_k],
\]

where $E[R_k]$ is the expected rate of the Slepian-Wolf code, which depends on the outcome of earlier decodings and on the source realization. With some abuse of notation we use $R_k$ both to represent the rate-determination function (cf. 3) and the resulting rate (cf. 4). We express rate in nats to simplify notation.
In the Appendix, we show that
\[ E[\epsilon_k] \leq H_B\left(Q + \frac{1 + \sqrt{2}}{\sqrt{2}} \epsilon_k\right) \]
\[ + 2 \ln 2 \min \left[1, (l_k + 1)^2 \exp\left\{-l_k \frac{\epsilon_k^2}{4Q(1-Q)}\right\}\right] \]
\[ \leq H_B\left(Q + \frac{1 + \sqrt{2}}{\sqrt{2}} \epsilon_k\right) \]
\[ + 2 \ln 2 \min \left[1, (l_k + 1)^2 \exp\left\{-l_k \frac{\epsilon_k^2}{4Q(1-Q)}\right\}\right] . \]
We want to choose \( \epsilon_k \) to decrease as fast as possible so the first term converges to the source entropy. However, we must choose \( \epsilon_k \) to decrease slower than \( 1/\sqrt{l_k} \) so that the second term does not blow up. Such a choice ensures that the expected cost of transmission approaches \( H_B(Q) \), the cost in the case where the encoder knows the source statistics.

**B. Practical Considerations**

A consideration omitted from Section V-A is how to use the protocol for sources that are not i.i.d. The only needed modification is to transmit the desired rate, \( R_{k+1} \) instead of estimates of the source statistics, \( Q_k \). The random calculation of syndrome bits is inherently universal, thus there is no need for the encoder to track the source statistics. (In contrast the decoder needs to use its current model of the source when decoding.) By transmitting \( R_{k+1} \) to a constant pre-specified precision, the feedback rate remains a negligible fraction of the total as block length grows.

We now consider implementation issues. For encoding we use LDPC codes as discussed in Section II-B. As the iterative sum-product algorithm progresses, it either converges to a solution that satisfies the code’s constraints (typically the maximum a-priori estimate) or fails to converge. By presetting a maximum number of iterations before declaring a decoding failure, we use the latter event to indicate detected decoding errors for our protocol. In our experiments, we have never witnessed sum-product to converge to an incorrect estimate. Either it converges to the correct estimate or not at all. However, if incorrect convergence is a worry, a cyclic-redundancy check (CRC) can be added at negligible cost in rate to drive the probability of undetected error to negligible levels. This can be done because error-detection is essentially a much simpler problem then error-correction.

Slepian-Wolf systems based on LDPC codes approach the minimal compression rate bound only for large block lengths. As is discussed in [7], extra redundancy is needed for the short block-lengths used for the first few blocks of our protocol. As will be seen in the results presented in the next section, the effects of short block lengths are not large.

**C. Results**

In this section we present details of the implementation and performance results for our blind algorithm. In each simulation, blocks of 100,000 source symbols are generated. The initial block-length is set to 100. While quite short for LDPC codes, recall that the first block is sent uncompressed and the block length grows rapidly, each successive block doubling in length. The redundancy parameter \( \epsilon_k \) is set to \( \epsilon_k = 2/\sqrt{n_k} \). We arbitrarily chose the constant numerator 2 since it results in good empirical performance and satisfies the constraint of Section V-A. As discussed in Section III-E, we use a selection of 38 codes. While the proposed protocol requires a large number of LDPC codes to be generated and stored, this can be partly mitigated by some combination of rate-adaptive techniques and fewer rate points.

We plot the results of memoryless source simulations in Fig. 16. In these plots the average cumulative redundancy in percent (averaged over 25 simulations) is plotted versus cumulative block-length, \( n_k \). Cumulative redundancy is defined as \( \sum_{j=1}^{k} \left[ d_j - H(Q)\right]n_k \), where \( d_j \) is the cumulative sum of bits transmitted in either the forward or feedback channels. The results of the system are plotted for 2 different source entropy rates: 0.1791 and 0.1944. As an example, to transmit 100,000 bits for a source entropy \( H_B(Q) = 0.1791 \) bits required an average of 26,787 bits. This translates into a redundancy of 49%, or 8,875 bits more than the 17,912 bit minimum. In these plots, as \( n_k \) grows the overall redundancy declines. In addition, for a source of entropy 0.1791 (the other source is omitted for clarity, but is similar), a bound on the expected cumulative redundancy using (23) is also plotted, as well as the performance assuming full knowledge of the source statistics (based on the results of [7]). Despite the limitations mentioned, and the fact that our bound omits the cost of feedback, our results perform well in relation to our bound.

![Fig. 16. Averaged results of blind compression of an i.i.d. source. The horizontal axis is the number of cipher-text bits \( n_k \) in log-scale, and the vertical axis is the system redundancy (percentage of bits used above the entropy rate) in log-scale. Ideally, we would like to transmit at 0% redundancy (the entropy rate). As the number of bits transmitted grows, performance improves.]

**D. Blind Video Compression**

We incorporate the protocol for blind transmission described earlier with the encrypted video compression framework of Section IV into an end-to-end system for compressing encrypted video. To operate in a blind manner the decoder feeds rate requests (as discussed in Section V-B) for each portion of each bit plane of each frame. When a decoding error is detected, the decoder requests the encrypted data be
retransmitted uncompressed. Though inefficient, as mentioned in Section V-A, performance can be improved with a hybrid-ARQ-type retransmission strategy.

For clarity, the following discussion presumes each bit plane of each frame is compressed whole, even though as in Section IV-C, we break each bit plane into sections. We label the rate used for the most significant bit plane of frame with then apply this function to all three sequences. We select significant predictability between the value and a retransmission, as in the Appendix. Fortunately, there are implemented off-the-shelf LDPC codes. We would like to design codes better matched to the assumed statistical model of the underlying sources, and that are designed with bit-doping in mind. Finally, although we consider lossless video coding here, it is possible to extend these ideas to lossy video coding. For example, our inability to compress the least significant bit planes suggests dropping the less significant bit planes. Since these bit planes play little significance in the prediction process, their loss should not strongly inhibit algorithm performance.

APPENDIX

In this appendix we analyze the protocol presented in Sec. V. Without loss of generality, in the analysis we assume that for all terms of (4), the probability of a decoding error. In Section B we bound the second term, the expected rate used during transmission. In Section C we put the results together to choose \( c_k \).

A. Probability of Decoding Error

We assume that the scheme uses Slepian-Wolf codes that succeed (i.e., there are no decoding errors, true in the asymptote) as long as the entropy rate of the \( k \)-th block is below the transmission rate, i.e., \( H(x_{n_{k-1}+1}^{n_k}) < l_k R_k \) where \( l_k = n_k - n_{k-1} + 1 \). We use large deviation techniques along the lines of [27] to bound the probability of decoding error on the \( k \)-th block.

\[
\Pr[e_k = 1] = \Pr[H(x_{n_{k-1}+1}^{n_k}) > l_k R_k] 
= \sum_{P} \sum_{x_{n_{k-1}}^{n_k} \in T_P} p(x_{n_{k-1}}^{n_k}) \Pr[H(x_{n_{k-1}+1}^{n_k}) > l_k R_k] 
= \sum_{P} \sum_{x_{n_{k-1}}^{n_k} \in T_P} p(x_{n_{k-1}}^{n_k}) \sum_{H(P) > H_{n_{k-1}}^{n_k} \in T_P} \sum_{n_{k-1}+1}^{n_k} p(x_{n_{k-1}+1}).
\]

In (7) the rate \( R_k \) is random, depending on the empirical distribution of the first \( n_{k-1} \) source symbols and the margin \( c_k \) via (3). In (8) we use \( P \) to denote this empirical distribution, and \( T_P \) to denote the set of sequences of that empirical distribution and of appropriate length. Since \( Q_{k-1} = P \), plugging \( P \) into (3) gives the, now non-random, rate used

<table>
<thead>
<tr>
<th>Frames 4 - 12</th>
<th>Foreman</th>
<th>Garden</th>
<th>Football</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypted</td>
<td>0.7918</td>
<td>0.9039</td>
<td>0.9917</td>
</tr>
</tbody>
</table>

TABLE II: RESULTS FOR THE BLIND COMPRESSION OF THE THREE ENCRYPTED SEQUENCES. THE PROPOSED APPROACH ACHIEVES SIGNIFICANT COMPRESSION GAINS.

VI. CONCLUSIONS & FUTURE DIRECTIONS

In this work, we have presented a practical scheme for compressing encrypted image and video data. We describe a general framework and specialize this framework to LDPC-based compression and stream ciphers. We confront two main issues: appropriate source models and statistics-blind encoder operation. Regarding the first, we propose idealized i.i.d., 1-D, and 2-D Markov models for compressing encrypted images, and give an example of the resulting benefits. We further describe how to extend these models to exploit the temporal correlations present in gray scale video. We implement the system and provide performance results for the “Foreman”, “Garden”, and “Football” sequences. Regarding
\[ R_k(P, \epsilon_k). \] We continue as
\[
\Pr[\epsilon_k] \leq \sum_P \sum_{P, P_{\epsilon_k}} \exp\{-n_{k-1}D(P\|Q) - l_kD(\hat{P}\|Q)\}
\]
\[
\leq \sum_P \sum_{P, P_{\epsilon_k}} \exp\{-l_k \min_{P, P_{\epsilon_k}} [D(P\|Q) + D(\hat{P}\|Q)]\}
\]
\[
\leq (l_k + 1)^2 \exp\{-l_k[D(P^*\|Q) + D(P^* + \epsilon_k\|Q)]\}
\]
In (10) we use \( p(x^{n_{k-1}}) = \exp\{-n_{k-1}[H(P) + D(P\|Q)]\} \) for all sequences \( x^{n_{k-1}} \in T_P \), and \( |T_P| \leq \exp\{n_{k-1}H(P)\} \), see [27]. In (11) we use \( l_k = n_{k-1} \), and the minimization is over all distributions (even continuous variables), not just those that are types. In (12) \( P^* \) is the minimizing distribution noting that each summation is over \((l_k + 1)\) elements. After minimization the exponent does not depend on \( P \) or \( \hat{P} \). We sum over all binary types of length \( l_k \), of which there are \( l_k + 1 \).

The error exponent of the probability of decoding error \( d \) depends on the unknown distribution \( Q \). We study this exponent to determine a good choice for \( \epsilon_k \). To do this, we solve \( l_kP^* \) assuming a fixed \( \epsilon_k \).

\[
\frac{d}{dP}[D(P\|Q) + D(P + \epsilon_k\|Q)]
\]
\[
= \frac{d}{dP} \left[ P \ln \frac{P}{Q} + (1 - P) \ln \frac{1 - P}{Q} + (P + \epsilon_k) \ln \frac{P + \epsilon_k}{Q} \right]
\]
\[
+ (1 - P - \epsilon_k) \ln \frac{1 - P - \epsilon_k}{Q}
\]
\[
= \ln \left[ \frac{P(P + \epsilon_k)(1 - Q)^2}{(1 - P)(1 - P - \epsilon_k)Q^2} \right].
\]
Setting (14) equal to zero, and solving for \( P \) using the quadratic equation gives
\[
P^* = -\frac{\epsilon_k}{2} - \frac{2Q^2 - \sqrt{2^2(1 - 2Q)^2 + 4Q^2(1 - Q)^2}}{2(1 - 2Q)}.
\]
For any choice of \( \epsilon_k \), and any source distribution \( Q \), this value of \( P^* \) determines the dominant source of decoding error. Using (15) in (12) yields a bound on the decoding error for this protocol. Note that because \( D(P\|Q) \) is convex in its arguments, \( P^* \leq Q \leq P^* + \epsilon_k \).

The error exponent \( D(P^*\|Q) + D(P^* + \epsilon_k\|Q) \) has a particularly simple form when \( \epsilon_k \) is small. We define \( P^* = Q - \bar{\epsilon} \) and \( P^* + \epsilon_k = Q + \bar{\epsilon} \), where \( \epsilon_k = \bar{\epsilon} + \tilde{\epsilon} \). By the convexity property just discussed, \( \tilde{\epsilon}, \bar{\epsilon} > 0 \). With these definitions, we approximate the error exponent when \( \epsilon_k \) is small.

\[
D(P^*\|Q) + D(P^* + \epsilon_k\|Q) = D(Q - \tilde{\epsilon}\|Q) + D(Q + \bar{\epsilon}\|Q)
\]
\[
= (Q - \tilde{\epsilon}) \ln \left[ 1 - \frac{\tilde{\epsilon}}{Q} \right] + (Q + \bar{\epsilon}) \ln \left[ 1 + \frac{\bar{\epsilon}}{1 - Q} \right]
\]
\[
+ (Q + \bar{\epsilon}) \ln \left[ 1 + \frac{\bar{\epsilon}}{Q} \right] + (Q - \tilde{\epsilon}) \ln \left[ 1 - \frac{\tilde{\epsilon}}{1 - Q} \right]
\]
\[
\simeq \frac{\tilde{\epsilon}^2 + \bar{\epsilon}^2}{2Q(1 - Q)} + \frac{(\tilde{\epsilon}^3 - 3\tilde{\epsilon}\bar{\epsilon}^2)}{2Q^2(1 - Q)^2} \geq \frac{\epsilon_k^2}{4Q(1 - Q)} \geq \epsilon_k^2
\]
In (16) we use \( \ln(1 + x) \simeq x - x^2/2 \). Writing \( \bar{\epsilon}^2 + \tilde{\epsilon}^2 = (\bar{\epsilon} + \tilde{\epsilon})^2 - 2\tilde{\epsilon}\bar{\epsilon} = \epsilon_k^2 - 2\tilde{\epsilon}\bar{\epsilon} \), one can see that (16) is minimized under the constraint for small \( \epsilon_k \) by selecting \( \bar{\epsilon} = \tilde{\epsilon} = \epsilon_k/2 \). Choosing \( Q = 0.5 \) lower-bounds the quadratic approximation of the error exponent.

In Fig. 17 we plot the error exponent and quadratic approximation to it for \( Q = 0.05 \) and \( Q = 0.3 \). The approximation (17) is quite good, even for large values of \( \epsilon_k \). For \( Q = 0.3 \), one can barely distinguish the quadratic approximation from the exact solution. The lowest curve in Fig. 17 is the lower-bound to the quadratic approximation with \( Q = 0.5 \).

![Fig. 17. Error exponents and quadratic approximations for Q = 0.05 and Q = 0.3. We also plot the lower-bound on the quadratic approximation (Q = 0.5). Note that for Q = 0.3, the approximation is so close to the error exponent that the lines are nearly on top of each other.](image)

**B. Bounding the expected Slepian Wolf rate, \( E[R_k] \)**

In order to minimize the cost (4) we must also take into account the second term of (4), \( E[R_k] \).

\[
E[R_k] \leq \Pr[H(x^{n_{k-1}})]
\]
\[
\leq H(Q + \gamma)H(Q + \gamma + \epsilon_k)
\]
\[
+ \Pr[H(x^{n_{k-1}}) > H(Q + \gamma)] \ln 2
\]
\[
\leq H(Q + \gamma + \epsilon_k) + \ln 2 \sum_{H(P) > H(Q + \gamma + \epsilon_k)} \sum_{x^{n_{k-1}}} p(x^{n_{k-1}})
\]
\[
\leq H(Q + \gamma + \epsilon_k) + \ln 2 \sum_{H(P) > H(Q + \gamma)} \exp\{-n_{k-1}D(P\|Q)\}
\]
\[
\leq H(Q + \gamma + \epsilon_k) + \ln 2(l_k + 1) \exp\{-l_kD(Q + \gamma\|Q)\}
\]
In (19) we split the expected rate into two events. The first is a high-probability event that occurs when the realized entropy is below the source entropy plus an arbitrary positive margin \( \gamma \). The second is a low-probability event that occurs when the realized entropy is large. In the former case, the code rate is upper bounded by \( H(Q + \gamma + \epsilon_k) \), while in the latter it is upper bounded by \( \ln 2 \). In (20) we upper bound the probability of the high-probability event by one, and analyze the low-probability
event using techniques similar to those used in Section A. As in that section, we also examine the small-\(\gamma\) region which gives,

\[
D(Q + \gamma \| Q) \approx \frac{\gamma^2}{2Q(1 - Q)}.
\]  

(22)

C. Optimizing the rate margin, \(\epsilon_k\)

We can now understand how \(\epsilon_k\) should be chosen. It is easiest to see this in the small-\(\epsilon_k\) region. Indeed, the small-\(\epsilon_k\) region is of greatest interest as we want to be as efficient as possible for large data files.

Noting that the probability \(Pr[\epsilon_k = 1]\) in (7) and \(Pr[H(x^{n-1}) > H(Q + \gamma)]\) in (19) can both be upper bounded by one, then substituting (12), (15), (21), and (22) into (4) gives

\[
E[\epsilon_k] = E[R_k] + \ln 2 \Pr[\epsilon_k] \leq H(Q + \gamma + \epsilon_k) + \ln 2 \min \left[1, (l_k + 1) \exp \left\{ -l_k \frac{\gamma^2}{2Q(1 - Q)} \right\} \right] + \ln 2 \min \left[1, (l_k + 1)^2 \exp \left\{ -l_k \frac{\epsilon_k^2}{4Q(1 - Q)} \right\} \right]
\]

To give the best bound, we want to pick \(\gamma\) small. Picking \(\gamma = \epsilon_k/\sqrt{2}\) balances the exponents. Combining the exponential terms gives (5):

\[
E[\epsilon_k] \leq H \left( Q + \frac{1 + \sqrt{2}}{\sqrt{2}} \epsilon_k \right) + 2 \ln 2 \min \left[1, (l_k + 1)^2 \exp \left\{ -l_k \frac{\epsilon_k^2}{4Q(1 - Q)} \right\} \right]
\]

(24)

In the first term the difference between the expected cost and entropy grows with \(\epsilon_k\). Hence, we want to pick \(\epsilon_k\) as small as possible. However, the second term constrains this choice. The margin \(\epsilon_k\) must go to zero slower than \(1/\sqrt{l_k}\), else the \(l_k\) in the exponent will dominate. Thus the algorithm we present quickly converges to the scenario of non-blind transmission. The dependence on \(\sqrt{l_k}\) in this expression is somewhat intuitive. The standard deviation in the best estimate of \(Q\) drops as \(1/\sqrt{n-1} = 1/\sqrt{l_k}\).

REFERENCES