Securing Bulk Content Almost for Free*

John Byers Mei Chin Cheng Jeffrey Considine Gene Itkis
Alex Yeung
{*byers, meicheng, jconsidi, itkis, ayeung*}*@cs.bu.edu*
Computer Science Dept.
Boston University
111 Cummington St.
Boston MA 02215

January 21, 2002

Abstract

Content providers often consider the costs of security to be greater than the losses they might incur without it; many view "casual piracy" as their main concern. Our goal is to provide a low cost defense against such attacks while maintaining rigorous security guarantees.

Our defense is integrated with and leverages fast forward error correcting codes, such as Tornado codes, which are widely used to facilitate reliable delivery of rich content. We tune one such family of codes – while preserving their original desirable properties – to guarantee that none of the original content can be recovered whenever a key subset of encoded packets is missing. Ultimately we encrypt only these key codewords (only 4% of all transmissions), making the security overhead negligible.

1 Introduction

The cost of securing bulk content for delivery is often viewed as prohibitively expensive. Many content providers prefer to sidestep security issues in favor of optimizing transfer speed and client capacity so as to maximize performance to the paying customers to which content providers must cater. We propose a new method of securing content that significantly lowers the costs of security for both the server and its clients while still maintaining rigorous security guarantees.

*The latest version of the paper and our ongoing experimental work is available at http://www.cs.bu.edu/groups/aces/codes/

Our techniques are based on leveraging well known all-or-nothing like properties of certain fast forward error correction codes (FEC) such as Tornado codes [12]. These codes can be used to facilitate the delivery of bulk content, particularly in the multicast case, as advocated using the digital fountain paradigm [2]. By making minor modifications to the codes and encrypting about 4% of all transmissions, we ensure that an adversary intercepting the remaining plain-text transmissions can not determine *any* information about a particular block of the file – individual input blocks are completely undetermined by the packets sent as plain-text. This is essentially the same security guarantee as that of [18, 23] (see Sec. 3.2 for more detailed discussion, including limitations of these definitions).

Similar ideas of using preprocessing (some special scrambling or an all-or-nothing transform) with the subsequent encryption of only a small part of the content has been proposed before, see e.g. [10, 1]. The distinguishing feature of our work is that the pre-processing step we use is really motivated by other considerations (namely, forward error-correction requirements); we need only to slightly modify it, preserving the originally desired features. Thus, the preprocessing step can be considered as already "paid for", and adding confidentiality becomes almost free.

Since only 4% of the information sent is encrypted, there are clear advantages on both the server and client sides. Reducing the amount of computational work on the server side is particularly important for busy servers delivering popular content, but the savings are also significant for thin clients, such as embedded devices or mobile palmtops. Moreover, while speeding up off-line encryption in group key settings is an important performance optimization, reducing the work encrypting packets on a per-client basis by a factor of 25 can be a truly significant reduction.

These techniques motivate a study of specific issues such as integration of security, compression and other issues. For example, since our techniques send most packets as plain-text and expose significant partial information, it is important for the entropy of the input to be high to avoid basic statistical attacks. Fortunately, compression is already used for most bulk content delivery and most schemes are geared to provide the highest entropy output. In particular, our work makes explicit the amount of information exposed when plain-text is leaked. As a result, it becomes possible to rigorously address the security provided and begin to examine the tradeoffs as less encryption is used.

In this paper, we demonstrate lightweight techniques for securing content against eavesdroppers which leverage sparse parity check codes, if these are already used. We show a clear tradeoff between expensive per-packet encryption and our techniques which can be provided nearly for free once the content is appropriately encoded without losing rigorous security guarantees. Additionally, our techniques are modular enough that they can be implemented as black box "filters" on the outgoing packets of a server and the incoming packets of a client thus providing security with minimum knowledge of and interaction with encoding functionality. Our solution gives a novel, cheap and easily applicable technique for adding encryption to encoded content and opens a number of important research directions in providing rigorous security guarantees at lower cost than full per-packet encryption. We also introduce the concepts and definitions intended to make the study of light security more formal and precise.

The rest of the paper is organized as follows. We begin in Section 2 by describing the general

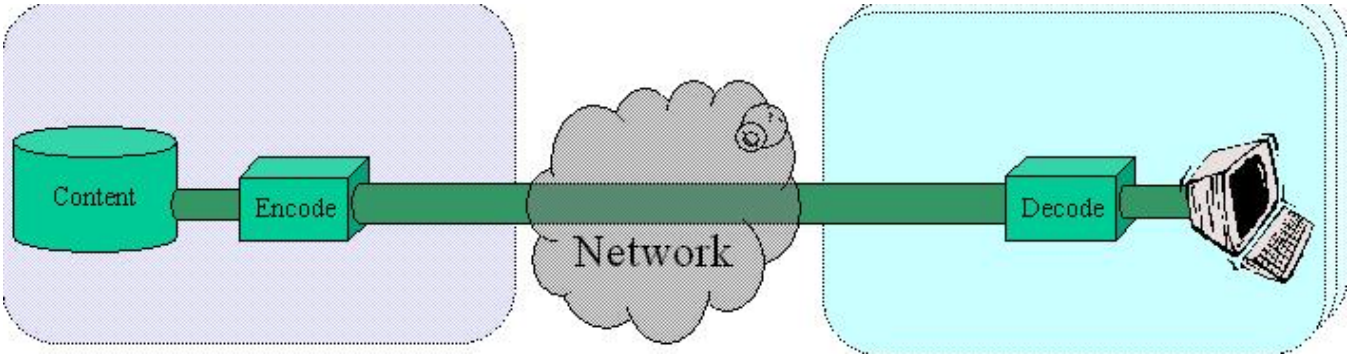

Figure 1: Typical system using FEC encoding: Server subsystem is on the left and Client subsystem(s) on the right.

implications of our approach in content delivery scenarios, and then develop three specific scenarios which stand to benefit. Next, in Section 3, we give a technical presentation of the sparse parity-check codes and the encryption methods which we use in our work. We demonstrate how to build provably secure codes, and provide several key new definitions and theorems regarding the level of security which can be obtained by our methods as well as future similar techniques. In Section 4, we present the performance of our experimental system which integrates the components presented in Section 3, focusing in on the performance of our modified codes and the performance of dovetailing encoding and encryption. We conclude with future directions and open problems in Section 5.

2 Bulk Content Delivery Scenarios and Requirements

In the scenarios we wish to secure, the goal is to reliably transmit rich (bulky) content to large audiences with high concurrency. The primary challenge associated with delivering content to large audiences which we consider is scalability. As session sizes grow, server side resources are strained, often leaving little time for separate encryption of each packet across each session.

A canonical approach to deal with issues of scale is to use a reliable multicast paradigm, whereby each of the server's transmissions are replicated inside the network and delivered to the entire multicast group. Within such a paradigm, using FEC codes (see Fig. 1) is a natural solution for dealing with widely varying packet losses across space and time [25] while avoiding the well-known feedback implosion problem that arises when all members in a set of receivers must acknowledge each transmission [13, 2]. As for securing content under the multicast paradigm, the most common approach is to use a shared group key (i.e. one known to all clients) to encrypt the data. The group key is initially distributed in some out-of-band fashion, and then it can be maintained using secure in-band key updates (efficient group key management techniques have been proposed e.g. [24] and efficiently implemented [9]). Note that when a fixed (time and

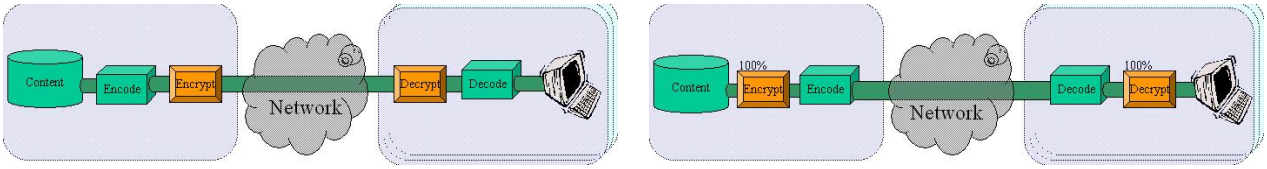

Figure 2: Two "naive" ways to secure data streams in combination with FEC encoding.

connection invariant) group key is used, it is possible to encrypt content off-line. Such a system can be implemented as either architecture of Fig. 2.

One potential issue with using the multicast channel for all encrypted packets is that a content provider may prefer to have finer-grained control over who receives content. Thus, providers may prefer to employ connections which provide a more personalized service than multicast connections for ease of accounting (e.g. for billing purpose). Such personalized connections can use any generic encrypted point-to-point channel, such as SSL [22]. But this preference defeats the benefits of multicast when every transmission must be individually encrypted for each client. However, this approach is much more realistic when only 4% of the content must be encrypted and transmitted over a dedicated connection, while the rest may be multicast in the clear – without compromising security. We note that personalized connections can also be used as a simple means of distributing the group key while the encrypted content is still sent over the shared multicast channel.

Actual content delivery implementations which approximately realize the benefits of the multicast paradigm can be divided into three categories. The first, IP multicast [6], provides a best effort IP multicast service, on top of which reliability can be supplied. IP multicast facilitates packet replication inside the network to reach the set of multicast clients so it substantially reduces both network traffic and the load at the server. Unfortunately, IP multicast has not yet seen wide adoption over much of the Internet.

An alternative to IP multicast which has the potential to reduce server load, but does not provide any reduction in network traffic is the use of individual unicast connections to deliver popular content. Of course, the server must then perform the extra work of sending packets to each client individually. One way to offset some of the associated processing costs is to use congestion-controlled UDP drawing FEC encoding packet payloads from a shared pool [20] (in this case the data flow of Fig. 1 is flowing over UDP). This method can also achieve asynchronous reliability while minimizing I/O and computation costs at the server. In [20], the authors also describe how these mechanisms can be applied to efficiently deliver popular, encoded content over TCP.

A hybrid approach is to use an overlay network [3, 19] and simulate a multicast network using a collection of unicast tunnels. Each node, including the origin server, maintains low to moderate fanout in the virtual multicast topology, thus each sends data to only a few hosts, greatly distributing the delivery load across the system. However, these nodes are heterogeneous and may not be computationally powerful, so minimizing use of resources continues to be crucial.

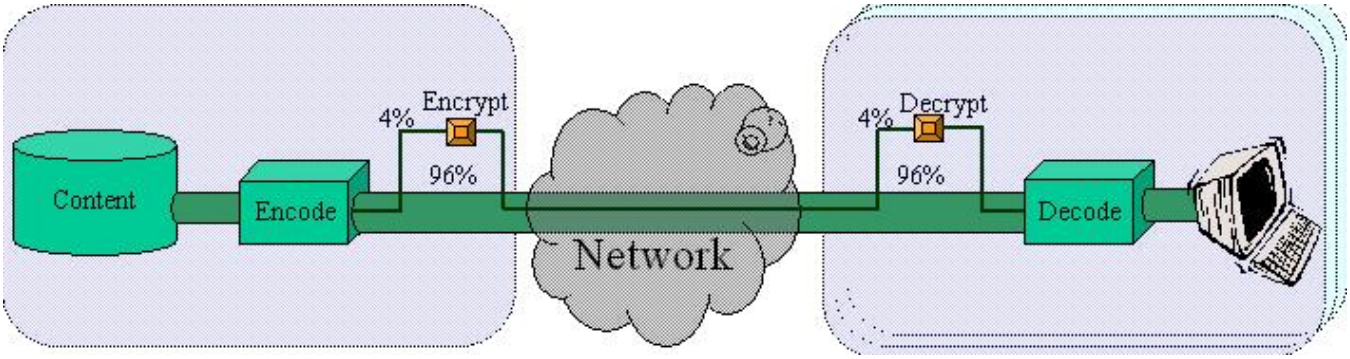

Figure 3: Our system architecture: Most data flows as in the original system, with only 4% fraction of the packets pipelinded through a secure channel.

In the ShapeShifter architecture [21], FEC codes are used over an overlay network to efficiently distribute content with better network utilization and fault tolerance.

For the first two of these approaches (true multicast and true unicast), encryption is handled in essentially the same fashion. FEC encoded is generated (possibly off-line) at the origin server and passed to the encryption filter. This filter decides whether encryption is necessary and sends it through the appropriate channel (noting there may be just one underlying channel with optional encryption). On the other side of the network, the client filter decrypts any encrypted packets received and sends both sets of packet to the FEC decoder which eventually outputs the data. In the third approach (overlay), the possibilities are richer, first since intermediate nodes may be able to forward encrypted content to other nodes when personalized encryption from the origin server is not a requirement. Also, in some scenarios, it is beneficial (for more efficient delivery) for overlay nodes to re-encode content. This particular enhancement can be used completely transparently with our proposed encryption methods provided only un-encrypted packets are used in re-encoding.

3 Secure FEC Codes and Cryptographic Primitives

The modules which will be used as building blocks in our secure content-delivery architecture (see Figure 3) are FEC encoding and decoding modules as well as encryption and decryption modules. While we are able to use any of a variety of encryption algorithms (we will use them in a purely block-box fashion), the FEC codes which we employ must be specifically customized for our integrated security approach. We describe the class of codes we employ, the properties we need from them, and the customizations we apply to achieve these properties first.

3.1 Background: Sparse Parity-Check Codes

Rabin [16] defined an Information Dispersal Algorithm (IDA) which disperses a file F of length ℓ into n pieces F_i, $1 \leq i \leq n$ each of length ℓ/m such that the original file F can be reconstructed from *any* m pieces, where $n > m$. Rabin cited many applications for this procedure, ranging from dispersity routing, to storing files on disks which are prone to failure, to fault-tolerant broadcast. While the general approach proposed by Rabin is very powerful, a limitation of the approach is the computational complexity of the encoding and decoding operations he used. Most IDA approaches use Reed-Solomon codes, which rely on cumbersome finite field operations and have $O(n^2)$ encoding and decoding times. Thus, while IDA works well for reconstructing a file stored as n pieces distributed over n remote sites when n is 14 and m is 10, its decoding inefficiency is prohibitive when multicasting an encoded file spanning n packets when n is 10,000 and m is 5,000. An alternative approach relies on the use of sparse codes based on parity-check principles, which can be encoded and decoded in nearly linear time, at the expense of relaxing the optimal decoding guarantee provided in IDA [12, 2, 11].

Sparse parity-check codes facilitate fast encoding and decoding and are well suited to use with large files and highly dispersed encodings. A document is divided into a collection of m fixed-length *input symbols*, or data packets, $x_1, \ldots, x_m$. An *encoder* produces a potentially unbounded sequence of *output symbols*, or encoding packets[1], $y_1, y_2, \ldots$ from the set of data packets. With parity-check codes, each output symbol y_i is simply the bitwise exclusive-or of a subset of the input symbols, prepended with a list of the corresponding input symbols from which y_i is produced. Previous work on packetizing these output symbols [12, 2] describes how to represent this list in compressed form into a small, fixed-width field (4 bytes is typical). For a given output symbol, we refer to the cardinality of the subset of corresponding input symbols used to produce it as its *degree*, i.e. $y_3 = x_3 \oplus x_4$ has degree 2. We say that an encoding is *on-the-fly* or *memoryless* if the random subset of input symbols used to produce each output symbol is drawn identically and independently from the same distribution $\mathcal{D}$ [11]. The time to produce an encoding packet from a set of data packets is proportional to the degree of an output symbol, while decoding from a sequence of output symbols takes time proportional to the sum of the degrees of the output symbols in the sequence. Thus, while variable degree distributions $\mathcal{D}$ may introduce some packets of large degree, it is important to note that encoding (and decoding) times are a function of the *average* degree, not the maximum. It is therefore essential that the codes used are sparse, i.e. the average degree is low.

A *decoder* attempts to recover the entire set of input symbols from a subsequence of the output symbols. Sparse codes typically relax the ideal decoding guarantees provided with IDA and Reed-Solomon codes. Such an erasure code is said to have average *decoding inefficiency* ϵ if on average, a randomly chosen subsequence of length $(1 + \epsilon)m$ output symbols is sufficient to reconstruct the original file F. To minimize decoding inefficiency, the distribution $\mathcal{D}$ of the size of the subsets is of crucial importance. High performance can be achieved using variable degree graphs, i.e. graphs in which degrees of output symbols are non-uniform [12]. The same paper

[1] The packets in this sequence are not necessarily distinct.

shows that a certain family of heavy-tailed degree distributions is a particularly good choice.

To summarize the ideas introduced so far, the use of sparse erasure codes provides an algorithm which disperses a large file F of length ℓ into packets F_i, $1 \leq i \leq n$, each of length $p = \ell/m$ such that the the original file F can be encoded in time $O(dn)$, where d is the average degree and can be reconstructed from a randomly chosen subsequence of $(1+\epsilon)m$ (independent of n) packets with high probability.[2]

A set of n output symbols can be interpreted as a system of linear equations in m unknowns (input symbols) with 0-1 coefficients. A canonical representation of which output symbols are in the set would be in the form of an $n \times m$ 0-1 matrix. The ith row of the matrix represents the input symbols combined to form the ith output symbol where 0's represent the input symbols not used and 1's represent the input symbols that were exclusive-ored together. In this representation, encoding a set of output symbols $\vec{y}$ from a set of input symbols $\vec{x}$ and matrix Y can be described by $\vec{y} = Y\vec{x}$ where multiplication is multiplication by a scalar and addition is bit-wise exclusive-or. We call Y a *sparse parity-check encoding scheme*, or *spc-encoding scheme* for short and say that $\vec{y}$ is an *spc-encoding* of $\vec{x}$.

Given this representation as a set of linear equations, Gaussian elimination can be employed as a quadratic time decoding algorithm. However, the advantage of sparse codes in decoding is that the following (and computationally inexpensive) recovery rule can be employed instead: Find any equation with exactly one variable, recover the value of the variable by setting it equal to the value of the equation, then remove the newly recovered variable from any other equations that it appears in by exclusive-or'ing its value into each of these equations. With this approach, one can prove [12] that the total number of exclusive-or operations used to decode a set of output symbols is equal to the sum of the degrees of the output symbols, i.e. achieving linear decoding time with high probability, given a suitable degree distribution.

3.2 Securing Sparse Codes

Definition 1 *Function f is* packet-independent secure *(*pi-secure *for short) if for all $\vec{x} = \langle x_1, \ldots, x_m \rangle$, $\Pr(\mathrm{x_i} = \mathrm{v} | \mathrm{f}(\vec{\mathrm{x}})) = \Pr(\mathrm{x_i} = \mathrm{v})$ for all v.*

This definition is essentially equivalent to the security definition of [23] which deals with all-or-nothing transforms [18] in the unconditional security model. In this case, $f(\vec{x})$ represents information sent as plaintext but no information about an individual x_i is known without additional information (which we send encrypted). Note that later work with all-or-nothing transforms such as [1] add semantic security guarantees, thus avoiding information leaks about $\vec{x}$ present in [23] and our scheme.

In contrast to the schemes which had to do their own preprocessing [1, 10], we achieve pi-security in the context of spc-encoding without additional overhead. One way to do this is to make sure that Y has rank less than m and does not allow recovery of any x_i. In general these

[2]The codes we employ in this paper have $\epsilon = .04$ and $d = 11$ when applied to a 32 MB file with symbol size 1400B.

requirements may be hard to combine with the on-the-fly property of the encoding. However, a simple special case fulfils both requirements without any expensive verification: just assure that all rows of Y have even number of 1s. Namely, call an spc-encoding scheme Y an *spc2-encoding* if Y has an even number of 1s in each row.

Lemma 1 *Spc2-encoding is pi-secure.*

Proof: Let Y be an spc2-encoding scheme. Then the vector space spanned by the rows of Y (under bit-wise exclusive-or) contains only even degree vectors (i.e. those with an even number of 1's). Indeed, the sum (coordinate-wise exclusive-or) of any two vectors with even number of 1's, has an even degree of 1's. Thus any odd degree vector is independent of the row-vectors of Y. In particular, a vector containing 0s everywhere except i-th position (which contains 1) is independent from row-vectors of Y. Let us add this vector as a row vector (say as the 0th row) to Y and call the resulting matrix Y'. And let us add an output symbol v as an output symbol $y_0 = v$, $\vec{y'} = \langle y_0 = v, y_1, y_2, \ldots \rangle$. Since, for the given $\vec{y}$, the equation $Y\vec{u} = \vec{y}$ had a solution for $\vec{u}$, and Y' was obtained by adding an independent row vector, then $Y'\vec{u} = \vec{y'}$ also has a solution for $\vec{u}$. Moreover, this solution is guaranteed to have $u_i = v$. ■

If an encoding is pi-secure, it protects the content, but it also prevents its decoding. In order to enable decoding we must augment the pi-secure encoding with some additional output symbols. Such additional output symbols must be communicated over a separate secure channel, in order to preserve security. We now specify one natural way to provide pi-security, beginning with a definition.

Definition 2 *A* repeated random pad encryption*, or rp-encryption for short, of $\vec{x}$ with a random pad p of fixed-length is $\vec{z} = \langle p \oplus x_1, \ldots, p \oplus x_m \rangle$.*

Corollary 1 *Repeated random pad encryption is pi-secure.*

Indeed, rp-encryption is equivalent to adding $x_0 = p$ to the vector $\vec{x}$ to obtain vector $\vec{v'}$, and computing the rp-encryption $z = Y\vec{x'}$, where Y is $(m+1) \times m$ 0-1 matrix with i-th row containing exactly two 1s: in position 0 and i. ■

This corollary highlights the relative weakness of pi-security in comparison to that of strong encryption – it can be provided by "weak" encryption schemes. Nonetheless, this paper explores how to achieve pi-security by restricting all encryption ("weak" or "strong") to apply only to a small fraction of all the data. It is our hope that new techniques can and will be developed to provide at least incremental improvements of our approach as compared to the "weak" encryption schemes such as rp-encryption.

Specifically, we propose to achieve pi-security by modifying the distributions discussed in Section 3.1, so that all the output symbols are either degree-1 or are even-degree, while keeping decoding inefficiency low, and keeping the proportion of singleton output symbols around 4% or less. Thus, the content can be protected by a pi-secure spc2-encoding plus encrypting of some singletons. If we assume that the singletons are properly secured by the encryption, then the

security of the content is reduced to the security of the spc2-encoding. Thus, we can at least guarantee pi-security with our scheme. In fact, rp-encryption can be viewed as a special case of such a construction, but missing the FEC properties of the codes in Section 3.1. Our preferred method is to derive spc2-encoding which is very close to the distributions in Section 3.1.

The next two lemmas show that rp-encryption and spc2-encoding are polynomially equivalent in their security in a strong way: namely, whatever can be learned about $\vec{x}$ from a repeated pad encoding $\vec{z}$ can be learned from the even degree encoding $\vec{y}$, and vice versa. In fact, we make the polynomial equivalences more precise below. First we show that the encoding $\vec{y}$ itself can be computed from the rp-encryption $\vec{z}$ in time $O(md)$, where d is the average packet degree. Since this is the same running time used to rpc-encode the content, if the spc2-encoding can be broken (in any sense) efficiently, than breaking the rp-encryption will incur at most additive overhead $O(md)$, i.e. the rp-encryption is not much more secure than the spc2-encoding.

Lemma 2 *Let $\vec{z}$ be an rp-encryption of $\vec{x}$ and let matrix Y be an spc2-encoding scheme. Then $\vec{y} = Y\vec{x}$ can be computed from $\vec{z}$ and Y in $O(md)$ steps.*

Proof: From $\vec{z}$ generate $\vec{y} = Y\vec{z} = Y\vec{x}$. The second equality holds because the pads cancel each other out, as there are an even number of 1s in each row of Y. ∎

The relationship in the other direction is slightly more complex since the correspondence between rp-encryption and spc2-encoding is one-to-many: a given rp-encoding determines spc2-encoding, while there could be many rp-encryptions corresponding to a given spc2-encoding. The next Lemma shows that an rp-encryption of any $\vec{x}$ with a uniformly distributed random pad can be computed from an spc-2 encoding of $\vec{x}$ in $O(m^2)$ steps. We assume that the number of output symbols $n = O(m)$.

Lemma 3 *Let Y be a 0-1 matrix of rank $m-1$, corresponding to an spc2-encoding scheme. Then given an spc2-encoding $\vec{y} = Y\vec{x}$ of some x, rp-encryption of $\vec{x}$ with a uniformly distributed random pad can be computed in time $O(m^2)$.*

Proof: The space of all even degree vectors has rank $(m-1)$. Thus, Y contains a basis for the space of all even degree vectors. In particular, the vectors corresponding to output symbols $\tilde{x_i} = x_1 \oplus x_i$ for $1 \leq i \leq m$ would be in that space. Thus, these output symbols could be obtained by Gaussian elimination in $O(m^2)$ steps.

Now, pick any output packet value r uniformly at random, and assume that $r = z_1 = x_1 \oplus p$ for some unknown, but also uniformly distributed pad p. The rp-encryption $\vec{z}$ of $\vec{x}$ with this p can now be computed as $z_i = z_1 \oplus \tilde{x_i}$ in $m-1$ steps. ∎

The above two lemmas imply that the security of even degree encoding is polynomially equivalent to the security of repeated pad encryption. While the two schemes are polynomially equivalent in their security, it is still possible that spc2-encoding is more secure than the rp-encryption. For example, by Lemma 2, whatever can be extracted from the encoding in linear time can also be extracted from the rp-encryption, also in linear time. But if some information can be computed from the rp-encryption in linear time, it is not clear how to extract this

information from the encoding faster than in quadratic time (due to Gaussian elimination) as given by Lemma 3.

So, in the worst case, by resorting to Gaussian elimination, spc2-codes can be reduced to repeated pad. If Gaussian elimination is necessary, this could provide some additional security – since for large files Gaussian elimination could be prohibitively expensive. In this specific case, it appears that a faster reduction might be possible (e.g. if more output symbols are collected), but other methods of increasing security might be possible.

Our final definition quantifies precisely how much encoded content we must encrypt in order to enable the remainder to be pi-secure:

Definition 3 *An encoding is called* α-securable, $0 \leq \alpha \leq 1$, *if for a randomly selected set of* n *output symbols, there exists a pi-secure subset of* $(1-\alpha)n$ *output symbols (i.e. the encoding can be secured by securing only an* α *fraction of output symbols).*

How can we achieve pi-security of the code in Section 3.1? Encrypting all odd degree symbols is one possibility, but it still requires that a large fraction of codewords be encrypted. The degree distributions we start with contain approximately 37% odd degree symbols so they are 0.37-securable. The approach that we propose and ultimately adopt is to modify or adapt the degree distribution so that the overwhelming majority of output symbols have even degree, while the rest are of degree one. To secure such a code, only the degree one symbols need be encrypted; all the rest may be sent as plaintext. Our experimental results in Section 4 show that we can modify our output symbol distributions so that 1) all outcomes of a given odd degree can be rounded up or down to the nearest even degree, 2) a low frequency of degree one packets can be maintained and 3) the other desirable properties of the original encoding, such as low decoding inefficency, can all be maintained. The upshot is a significantly lowered cost of securing a code against Gaussian elimination, since only approximately 4% of all packets need to be encrypted to achieve this goal, which we state this as our main result.

Theorem 1 *The content delivery scheme we have described is* .04*-securable.*

3.3 Encryption Schemes

In this paper we consider three efficient symmetric key encryption schemes: the two block ciphers Data Encryption Standard (DES) [14] and AES/Rijndael [5] cipher, and the stream cipher RC-4 [17]. The latter two are among the fastest modern cipher algorithms. AES is the the recently adopted Advanced Encryption Standard block cipher, which is intended to replace the veteran DES cipher. RC-4 is one of the fastest and most commonly used stream ciphers. We include the experiments using DES primarily for completeness and because it is probably the most widely used cipher at present. As we will demonstrate, using slower cipher algorithms like DES only strengthen the benefits of our results.

AES/Rijndael. Rijndael was recently selected as the Advanced Encryption Standard, approved by the National Institute of Standards and Technology, after extensive scrutiny and open analysis by experts, based both on its strength and efficiency. In our experiments we use CBC (Cipher Block Chaining) mode with key size 128 bits and 128 bit block-size.

RC4. RC4 is a fast stream cipher. It requires a key with any size between 1 and 256 bytes to initialize a 256-byte state table. Then a pseudo-random stream is generated by the state table to be exclusive-or'ed with the plain-text for encrypting and with the cipher-text for decrypting (such use of a pseudo-random one time pad is typical of stream ciphers). In this work, we use 128 bit (16 byte) key to initialize the state table.

DES. DES is one of the most used ciphers. It is widely deployed in applications ranging from Internet (SSL) to pay-TV, implemented in hardware (for which it was originally designed) and software. It uses 56-bit keys and 64-bit blocks. In its nearly 35 years of existence, it has demonstrated extraordinary resistance to cryptanalysis. Only in recent years have the limits set by its key size proven somewhat problematic, as successful attacks based largely on exhaustive search are now feasible. However, due to their high cost and other technical requirements, such attacks may not pose an immediate threat for many applications we focus on.

4 Experimental Results

Our experimental results are divided into two categories. The first set of results describes our methods for converting arbitrary degree distributions of sparse parity-check codes into distributions which can be secured by encrypting degree one output symbols and quantifies the performance cost of doing so. The second set describes our methods for combining erasure codes and encryption and quantifies the performance costs of using secure codes.

All of our experiments were run on an IBM Pentium III 930MHz machine with 384MB of RAM running the Red Hat Linux version 6.2. Measurements were taken for file sizes ranging from 1MB to 64MB. The time used for I/O operations was not included; only the time performing encoding and decoding or encrypting and decrypting was measured. A symbol size of 1400 bytes was used to make output symbols small enough to fit into the MTU of our LANs (Ethernet) while leaving space for packet headers and symbol identifiers.

4.1 Building Good Sparse Codes

The degree distributions we adapt to our "one or even degree" requirement are based on a heuristic using oracles, which we do not detail here, but which is fully described in [4]. This approach enables a server to compute the output symbol degree most likely to advance a client's decoding process given that the client has received k symbols from an initial distribution. The server then iteratively refines the original degree distribution based on these computed values.

m	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}
Average Inefficiency	0.175	0.101	0.060	0.040	0.045	0.024	0.024	0.017	0.017	0.014
Standard Deviation	0.066	0.029	0.014	0.008	0.042	0.011	0.007	0.003	0.013	0.009
Average Degree	10.7	15.5	13.6	13.7	11.6	14.3	18.8	19.0	15.4	16.6

Table 1: Statistics for Oracle Distributions (100 trials)

This approach can generate degree distributions with empirical decoding inefficiencies comparable to provable decoding guarantees in the literature such as Tornado Codes [12] though not quite as low as some proprietary codes [7]. These degree distributions are easily adapted to our degree restrictions and our experimental results show that the desirable properties such as low decoding inefficiency and average degree are preserved.

4.2 Picking a Good Graph

Although the codes we use are capable of generating an infinite sequence of encoding packets, the applications in this paper generally produce a finite number of output symbols whose degrees are randomly chosen from a prefix of this sequence. Therefore, we borrow an optimization also used with Tornado codes to make sure that the beginning of the random out degree sequence behaves well (i.e. it is not an outlier). A seed is randomly picked and $2m$ output symbols are generated from the first $2m$ degrees in the pseudo-random sequence produced by the seed. The utility of the random seed is evaluated by the average decoding inefficiency of 100 trials drawing from the $2m$ output symbols already generated. The seed with smallest average decoding inefficiency is then used for all trials. Note that this optimization should be done offline and only needs to be done once for a particular m and a particular prefix length.

4.3 Making Sparse Codes Securable

Our first set of experiments deals with adapting existing degree distributions to meet the "one or even" requirement while measuring the decoding inefficiency (network usage) and time elapsed (CPU usage). While the oracle based distributions can be directly created using only degree one and even degrees, we focus on general transformations that could be applied to any degree distribution (tuning the oracle directly gives similar results).

Our general approach is to generate output symbol degrees using the baseline distribution and leave them unchanged if they are one or even. If the degree generated from the baseline distribution is odd, we pick one of the neighboring even numbers to use in its place. This approach is intended to minimize changes in the behavior of the decoding process (the degrees are similar, particularly at the large scale) and in the average degree (it can change by at most one). Since

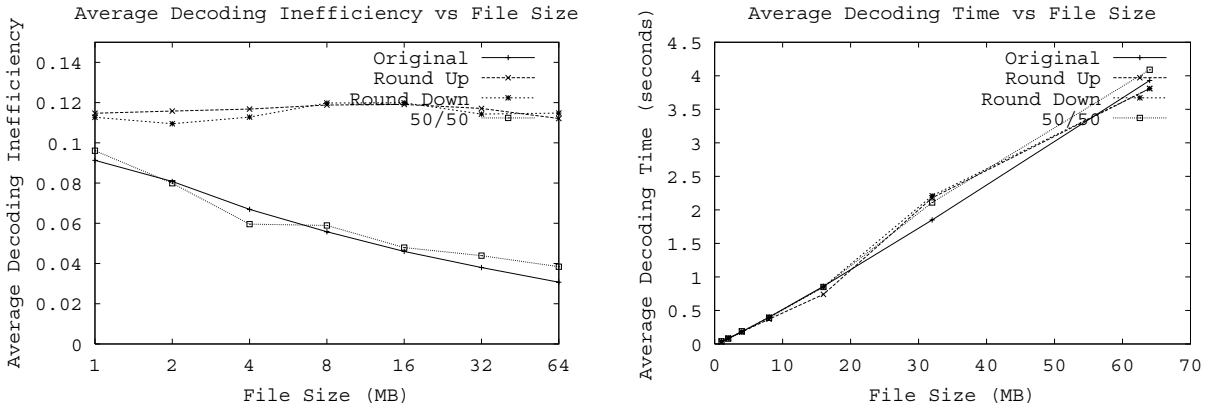

Figure 4: Experimental Data for Securing Codes. (left) Average Decoding Inefficiency vs File Size, (right) Average Decoding Time vs File Size

typical codes only have 4% output symbols of degree one, this approach also generates degree distributions that are 0.04-securable.

We experiment with three ways to map odd degrees greater than one into neighboring even degrees. When we say we "round" a degree, we mean we round it to the nearest even degree.

Round Up: Round up all odd degrees above one.

Round Down: Always round down odd degrees above one.

50/50: Round odd degrees above one up and down with equal probability.

Figure 4 shows our experimental results from modifying the degree distributions to make them securable. Both **Round Up** and **Round Down** have significantly higher decoding inefficiencies which do not improve as the file size increases. Both of these algorithms incur a decoding inefficiency penalty of roughly 8% over the original distribution. On the other hand, the decoding inefficiency of **50/50** is comparable to that of the original codes and incurs a decoding inefficiency penalty which is less than 1% for large files. For this reason, we employ **50/50** for the remainder of our experiments.

In spite of the disparity between decoding inefficiencies, the decoding times of all methods are in surprisingly close correspondence. This is an artifact of the patient decoding implementation we employ, which waits for additional output symbols before attempting to decode. As a result, redundant, high-degree symbols which contribute to the decoding inefficiency are typically discarded and do not contribute significantly to the overall running time.

4.4 Adding Encryption to Securable Codes

Our second set of experiments deals with incorporating encryption into securable codes, in scenarios where both the encoding and encryption are done online. We consider the following ways

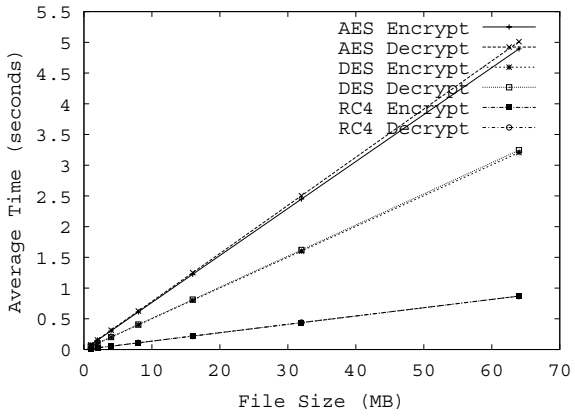

Figure 5: Cipher Speed Comparisons

of adding encryption to the transmission pipeline.

No Encoding: File is encrypted but not encoded. [3]

Encode Decode Only: File is encoded but not encrypted.

Secure Code/100%: File is encoded with a securable code and *all* output symbols are encrypted.

Secure Code/4%: File is encoded with a securable code and all degree one output symbols are encrypted.

We compare the costs of each of these methods of adding encryption using RC4, DES and AES. We use the RC4 and DES implementations of the OpenSSL Project [15] and the AES implementation at [8]. Both RC4 and AES use 128 bit keys and DES uses a 56 bit key in our experiments. There are three noteworthy points depicted in Figure 5. First, as is to be expected, the encryption and decryption times scale linearly with the file size, since the ciphers operate on a per-packet basis. Second, for each of these ciphers, encryption and decryption times are very similar; therefore, we only consider encryption times in our subsequent plots. And finally, each of three ciphers provide rather different performance, with the fastest (RC4) running about four times as fast as the slowest (AES).

Next we consider the relative costs of adding encryption and FEC in an integrated scheme. Figure 6 summarizes the experiments adding encryption to securable codes using AES and RC4. For large files, the slowest cipher (AES) requires an additional 103% time overhead to encrypt all packets, but only 6% overhead to secure the code by encrypting just the degree one symbols, while

[3] By encoding, we specifically mean applying an erasure code.

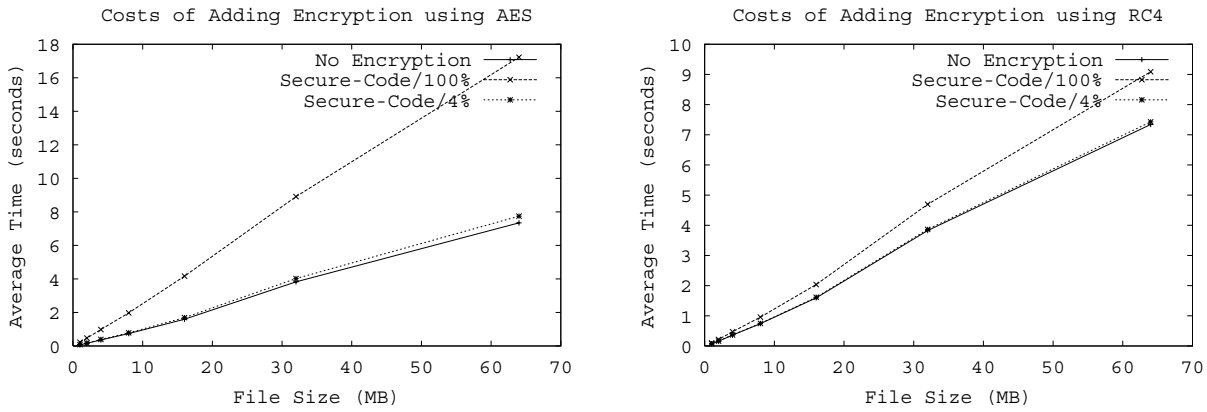

Figure 6: Experimental Data for Adding Encryption to Encoding

the fastest cipher (RC4) requires an additional 23% overhead vs. a 1% overhead, respectively. DES (not shown) requires 88% and 4%. Clearly, there is a significant advantage to using **Secure Code/4%** for the more expensive ciphers and still a 20% speedup for the least expensive. Also, we note that the slowest cipher with **Secure Code/4%** is faster than the fastest cipher with **Secure Code/100%**. We emphasize that these experiments are conducted when both the FEC encoding and the encryption is performed online, although as noted earlier, for many applications it is natural to conduct encoding offline, in which case the speedups are even more dramatic. The gains in all cases also intimately depend upon the relative speeds of the encoding and encryption algorithms used.

5 Conclusions and Future Work

In this paper, we consider a problem which has not yet received much attention in either the networking or security literature, namely, providing a level of security for content delivery applications which is strong enough to thwart casual piracy, yet does not require the full computational horsepower of packet-level encryption. Such a method is especially useful for securing popular content, when server resources are truly at a premium. Using conventional transmission of content, it is not clear how to specify such a middle ground whereby a tradeoff between security guarantees and computational resources is possible. However, when fast forward error correcting codes are employed as part of the content delivery scheme, we demonstrate that certain encoding schemes lend themselves naturally to "partial encryption," whereby only a small fraction of the codewords generated are encrypted, while the rest are transmitted in the clear. Such an approach has many advantages, notably 1) the encoding- encryption pipeline is modular and easily configurable, 2) the encryption and decryption costs are minimal and 3) reconstructing any given piece of the plain-text is provably hard (albeit not as hard as when full encryption is employed).

We view our current work as only a first step in this direction, noting that there are a number of potential improvements that can be made to both our definitions and our general approach. For example, the current security guarantees we provide are comparable to those available when a one-time pad is reused; ideally the hardness of reconstructing a piece of plain-text would scale linearly with the number of encrypted bits not revealed to the adversary.

One of our goals was to demonstrate a possibility of achieving provable light security improvements by introducing quadratic or greater overhead. Ideally, we would like to be able define and prove stronger notion of comparative security. For example, we had introduced one way to relate encodings E_1, E_2: $E_1 \preceq_{c,\sigma} E_2$. However, this relationship does not provide separation. It would be useful to prove results of the form $E_1 \prec_{c,\sigma} E_2$ (i.e., $E_1 \preceq_{c,\sigma} E_2$ and $E_1 \preceq_{c',\sigma} E_2$ for no $c' = o(c)$). While in this paper we have succeeded proving that rp-encryption $\preceq_{m^2,(1-1/2^{O(m\delta)})}$ spc2-encoding, we did not prove rp-encryption $\prec_{m^2,(1-1/2^{O(m\delta)})}$ spc2-encoding.

Our future work will also consider securing content delivery applications when fast forward error correcting codes are not immediately applicable. For example, is there a general principle that can render an arbitrary byte stream amenable to "partial encryption" of the form described earlier? Such a mechanism could be useful in providing security guarantees for live streams, among other applications.

6 Acnowledgements

The fourth author would like to thank Leonid Reyzin for useful discussions and suggestions, esp. for Section 3.2, and simplifications of Lemma 3 in particular.

References

[1] Victor Boyko. On the security properties of OAEP as an all-or-nothing transform. In *CRYPTO*, pages 503–518, 1999.

[2] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *Proceedings of ACM SIGCOMM '98*, Vancouver, September 1998.

[3] Y.-H. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *ACM SIGMETRICS 2000*, Santa Clara, CA, June 2000.

[4] Jeffrey Considine. Generating good degree distributions for erasure codes using oracles. Technical Report BUCS-TR-2001-019, Boston University, CS Dept, Boston, MA 02215, October 2001.

[5] J. Daemen and V. Rijmen. AES proposal: Rijndael, 1999. `http://www.esat.kuleuven.ac.be/~rijmen/rijndael/`.

[6] S. Deering and D. Cheriton. Multicast routing in datagram inter-networks and extended LANs. *ACM Transactions on Computer Systems*, 8:85–110, May 1990.

[7] Digital Fountain, Inc. Digital fountain technology overview. `http://www.digitalfountain.com/technology/`.

[8] B. Gladman. Cryptography technology. `http://fp.gladman.plus.com/cryptography_technology/index.htm`.

[9] H. Hamandi and G. Itkis. Group key manager on a smart-card. Preprint.

[10] Markus Jakobsson, Julien P. Stern, and Moti Yung. Scramble all, encrypt small. In *Fast Software Encryption*, pages 95–111, 1999.

[11] M. Luby, J. Gemmell, L. Vicisano, L. Rizzo, and J. Crowcroft. Asynchronous Layered Coding: A massively scalable reliable content delivery protocol. IETF Internet Draft draft-ietf-rmt-pi-alc-02.txt, July 2001, expires January 2002.

[12] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proceedings of the* 29^{th} *ACM Symposium on Theory of Computing*, April 1997.

[13] J. Nonnenmacher, E. Biersack, and D. Towsley. Parity-based loss recovery for reliable multicast transmission. In *Proceedings of ACM SIGCOMM '97*, September 1997.

[14] National Bureau of Standards. Data Encryption Standard. Federal Information Processing Standards Pub. 46, Washington, D.C., 1977.

[15] The OpenSSL Project. `http://www.openssl.org`.

[16] M. Rabin. Efficient dispersal of information for security, load balancing and fault tolerance. *Journal of the ACM*, 38:335–348, 1989.

[17] R. Rivest. The RC4 encryption algorithm. RSA Data Security, 1992.

[18] Ronald L. Rivest. All-or-nothing encryption. In *Fast Software Encryption*, number 1267 in Springer Lecture Notes in Computer Science, pages 210–218, 1997.

[19] Resilient Overlay Networks. `http://nms.lcs.mit.edu/projects/ron/`.

[20] S. Rost, J. Byers, and A. Bestavros. The cyclone server architecture: Streamlining delivery of popular content. In *Proceedings of the 6th International Web Caching and Content Delivery Workshop (WCW)*, Boston, MA, June 2001.

[21] The Shapeshifter project. `http://www.cs.bu.edu/groups/shapeshifter`.

[22] SSL 3.0 Specification. http://www.netscape.com/eng/ssl3/.

[23] D. Stinson. Some observations on all-or-nothing transforms, 1998. Available from http://cacr.math.uwaterloo.ca/~dstinson/papers/AON.ps.

[24] D. Wallner, E. Harder, and R. Agee. Key management for multicast: Issues and architectures. internet request for comments 2627, 1999. Available: ftp.ietf.org/rfc/rfc2627.txt.

[25] Maya Yajnik, Sue Moon, Jim Kurose, and Don Towsley. Measurement and modeling of temporal dependence in packet loss. In *Proceedings of INFOCOM '99*, New York, NY, March 1999.