Prime-based Mimic Functions for the Implementation of Covert Channels

WESLEY CONNELL w@zupy.com
DAN TAMIR dt19@txstate.edu
MINA GUIRGUIS msg@txstate.edu

Computer Science Department
Texas State University
San Marcos, TX 78666, USA

***Abstract*— A novel method for design and implementation of a covert channel using a prime-based mimic function is presented. The main idea is to map the production rules of a context-free grammar onto a set of prime numbers, and encode messages using primes as factors so that the generated data fits the statistical properties of a given language. The method's performance is assessed through different metrics such as language perplexities, secret-to-cover ratio, and transmission bit-rates. A set of experiments shows that prime-based mimic functions provide an efficient method for implementing a covert channel with a competitive secret-to-cover ratio and low perplexities.**

I. INTRODUCTION AND LITERATURE REVIEW

This paper concentrates on lexical steganography using mimic functions. That is, functions that map a message M_I from a domain A into a message M_O from a range B; such that M_O assumes the statistical properties of the elements of B [1]. In specific, we define a prime-based mimic function that encodes a secret message into the expressions of the grammar of a given formal language [2]. The proposed encoding can generate expressions that are statistically similar to typical expressions that belong to the language generated by the grammar. It can be used to embed message carrying expressions within other expressions of the language while maintaining a competitive bit-rate in a way that is not likely to be detected by a machine or a human being.

Steganography is the art and science of hiding information in plain sight such as images, audio, or text. By concealing information in an abstract medium, referred to as *cover* medium, we are effectively sending this information over a covert channel [3], [4]. Many abstract mediums can provide cover for information [1], [5], [6]. Covert channels are typically used to violate security policies. Thus, they present an ongoing threat to almost any entity that has sensitive data. In response, several studies have focused on detecting covert channels and scrambling data fields within the channel to render these covert channels useless [1], [7]–[9].

Covert channels can be classified into two categories; storage-based and timing-based. In storage-based covert channels, information is hidden in text, images, packet headers, etc. For example, the research work reported in [3], [10] have investigated the utility of different fields in the TCP/IP headers for hiding information. In timing-based covert channels, the timing of events indicates the covert message. For example, the work in [11] shows a covert channel where the hiding of information is based on whether the receiver receives a packet within a given interval.

In order to perform optimally, the cover medium is chosen and utilized in a way that debilitate the ability of humans and machines to recognize it as a cover for hidden information. Nevertheless, many steganography techniques exhibit a low secret-to-cover ratio (SCTR)[1]. Additionally, these techniques are vulnerable to automated steganalysis [8], [9].

There are numerous techniques and mediums for steganography [1], [7], [12], [13]. Lexical steganography techniques and mimic functions hide secret information within text that shares the probability distribution of a natural language or of a context free language that is likely to be transmitted over the cover medium. [1], [5], [7], [12], [14], [15]. Of specific interest is the process of hiding encrypted messages in order to hide the actual usage of encryption technology. In this case, it may be desirable to produce text that has the same probability distribution as the original information before it has been encrypted [5].

Wayner proposes mimic functions that use Huffman coding and inverse Huffman coding to preserve the probability distribution of the input stream [14], [15]. To increase the strength of this approach, Wayner developed mimic functions that map text into the production rules of a context-free grammar (CFG) and generate text that is accepted by a CFG parser [2]. The generated cover is shown to have theoretical strength, or resistance to steganalysis that is proportional to the average complexity of the CFG [15]. Although the method proposed by Wayner has potential to achieve the theoretical limits of probability preservation and produce an "innocent looking" text, its implementation is quite complex. It requires the construction of a probabilistic grammar and two Huffman coders as well as a complex mapping function that traverses Huffman trees in order to map text into production rules. In contrast, the method proposed in this paper uses a relatively simple probability preserving mapping of text to production rules [1], [4]. We are not aware of practical implementations

[1]The ratio of secret message size to the cover message size.

of Wayner's method and the actual SCTR obtained using this method in realistic scenarios.

NICETEXT [5] utilizes a collection of dictionaries and styles to construct a cover that is statistically similar to a specific language. The dictionaries, however, contain information of direct mapping of text-tokens to text-tokens rather than mapping text-tokens to production rules as done in this paper. Chapman's method yields an STCR of 0.044 [5]. Our method yields a better STCR of 0.069.

In this paper we propose a a new set of prime based mimic functions and a novel method of using these mimic functions for mapping data into production rules of a CFG and generating a set of expressions that belongs to the context free language (CFL) defined by the CFG. The method enables setting the probability of occurrence of expressions. We demonstrate the utility of the proposed method using statistical analysis of expressions generated by the mimic functions [14]. The analysis uses a language model and calculates geometric perplexities and average perplexities of expressions [12], [16]–[18]. Additionally, we calculate the average bit-rate and compare it to other steganography techniques. Our results exhibit low perplexities, and competitive secret-to-cover ratios.

Paper Organization: The next Section, Section II, describes the overall design of the covert channel. Section III, presents experiments' results. Finally, Section IV includes conclusions and directions for future work.

II. Prime-based Mimic Functions

In this section we define a prime-based mimic function and give a simple example of its usage. In addition, we discuss assessment metrics for the proposed method.

A. The Language Model

The proposed prime-based mimic function modifies data to fit the statistical properties of a language defined by a context-free grammar (CFG) G [2]. The CFG is modified by adding a function ξ which maps productions to a set of prime numbers. The grammar G augmented by the mapping ξ has the following structure:

$$G^M \quad = \quad (V, T, S, P, \xi) \tag{1}$$

where V is a finite set of variables, T is a finite set of terminal symbols, S is the starting symbol, P is a finite set of productions [2], and ξ is a function that maps productions to the prime-numbers and 1.

The Fundamental Theorem of Arithmetic states that every integer greater than one is composed of the products of a finite set of prime numbers [19]. The mapping ξ exploits this property and provides a uniquely decodable and compact encoding of information using the grammar productions rules where each production rule is mapped to a prime number, and combinations of production rules represent integers. In other words, integers are encoded using expressions that belong to the language defined by G^M. The encoded integers are obtained by interpreting fixed sized blocks of bits from the message as binary numbers.

An Illustrative Example: Consider the following simple augmented grammar (G^M) that generates palindromes of even length over the characters 'a' and 'b':

$$G^M \quad = \quad (\{S\}, \{a, b\}, \{S\}, P, \xi) \tag{2}$$

where the set P is given by the following productions:

$$p_0 : S \rightarrow aSa, \qquad p_1 : S \rightarrow bSb, \qquad p_2 : S \rightarrow \lambda$$

and the mapping function ξ maps the above productions to primes in the following manner:

$$p_0 :\rightarrow 2, \qquad p_1 :\rightarrow 3, \qquad p_2 :\rightarrow 1$$

To further illustrate, consider the valid expression 'aa'. This expression is derived from G^M by starting from S, and applying the sequence of rules: p_0 (which results in 'aSa'); followed by the rule p_2. Rule p_0 is mapped by ξ to the prime number 2, and rule p_2 is mapped by ξ to the integer 1. Hence, under this method, 'aa' encodes the integer 2. In a similar way 'bb' encodes the integer 3, while 'abba' and 'baab' are two different encodings of the integer 6. Thus, if the encoder wants to send the value 6 to the decoder it can use the production rules and generate one of these two expressions (say 'baab'), then send it to the decoder. The decoder parses the expression 'baab' [2], identifies that the only way to generate it is through the sequence of rules $\{P_1, P_0, P_2\}$. Factoring the product values associated with each production rule, the decoder obtains the value 6.

The fact that there may be more than one way to encode an integer is an advantage of this method since it increases the available options of covering an integer inside expressions and increases the difficulty of steganalysis. On the other hand, if the language is unambiguous, and the encoder / decoder consistently use the same procedure for generating / parsing expressions [2], then this encoding method is uniquely decodable and the decoding of an expression is guaranteed to yield the encoded integer. Note that rules mapped to 1 can be used an arbitrary number of times without affecting the value of expressions. This is another factor that contributes to the ability to produce more than one representations for an integer.

There are two issues to consider. First, one may question whether the expressions generated by the system would not prompt human attention. Our research shows that if the language defined by the grammar is complex enough, then it would be very hard for an interceptor to distinguish between artificial expressions and "natural" expressions transmitted over the medium. This can also be adjusted by assigning weights that reflect probabilities to production rules and selecting productions based on their weight. Another question relates to the strength (resistance to steganalysis) of the method. The method is strong since there is an exponential number of ways to map the productions onto the set of prime numbers (augmented by 1). Hence, a brute force approach to recover the original message from the set of expressions would not be

feasible as the number of productions grows. Moreover, the sender has an option to encrypt the data prior to hiding it.

B. The Mapping Algorithm

Another issue that needs to be addressed is the potential of having prime factors that do not correspond to productions. For example, the number seven - and its multiples - cannot be represented with only three productions mapped into the set $\{1, 2, 3\}$. Hence, all the prime numbers that are not included in ξ cannot be components of the encoded integer. In order to overcome this problem, we define a mapping procedure.

A few parameters govern the definition of the mapping. First, the range of integers to be mapped is determined by the block size. Second, the mapping must be uniquely invertible. Finally, a proper mapping algorithm allows the full range of input data to be mapped to an integer that can be encoded by G^M. A mapping example that fits with the previous example grammar is:

$$0 \to 2 : \{2\} \quad 3 \to 6 : \{2, 3\} \quad 6 \to 12 : \{2, 2, 3\}$$
$$1 \to 3 : \{3\} \quad 4 \to 8 : \{2, 2, 2\} \quad 7 \to 16 : \{2, 2, 2, 2\}$$
$$2 \to 4 : \{2, 2\} \quad 5 \to 9 : \{3, 3\}$$

Note that this mapping can be generated automatically using inverted gray code [20].

The example mapping allows a 3-bit integer, including the integers five and seven, to be encoded using only the factors $\{1, 2, 3\}$. To continue with the example, assume that the encoder has to decode the byte '00000110'. Under the mapping example, and with the production rules specified above, the byte '00000110' has an integer value of 6. It is mapped by the encoder to one of a few expressions including 'abaaba'. The decoder parses the expression, obtains $2 \times 2 \times 3 = 12$, and maps it back into 6. In a similar way, the number 7 is mapped by the encoder to the expression 'aaaaaaaa'. The decoder obtains $2 \times 2 \times 2 \times 2 = 16$. This is used, potentially through hashing, as a pointer to the decoder table where the number 7 is stored. This type of mapping algorithm, can be implemented statically by tables and / or dynamically. It can be expanded to include any size of input integer. In addition, many other mappings such as using powers of two can be constructed [20].

When a table is used for mapping, the table size depends on the block size. Hence, there there are several implementation trade-offs. Large blocks might require large tables. On the other hand, a dynamic mapping procedure might require extensive computation. In consideration of several alternatives to the mapping procedure including using powers of two (an additive representation of integers) as the way to map integers to expressions, we have decided that a combination of a table on the encoder side and a table along with a computation of an index to the table using factors of primes (a multiplicative representation of integers) on the decoder side, is a cost effective solution. In the future, we plan to further look at other mapping methods including additive methods such as powers of two.

C. General Usage

The process of using a prime-based mimic function as a steganography tool is shown in Figure 1. Suppose that Alice wants to send a secret message to Bob using our approach. Before transmission, Alice segments the message into blocks.[2] Sequentially, each block, interpreted as the binary representation of an integer, is mapped by the prime-based mimic function into a covert message referred to as a "valid sentence", and transmitted to Bob. Bob can processes (parse) the covert message and recover the original message using the inverse function of the same prime-based mimic function.

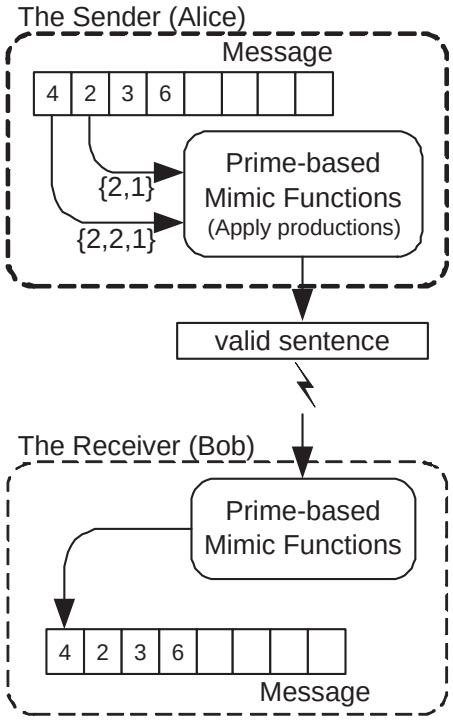

Fig. 1. The general approach of the proposed method.

D. Assessment Metrics

Perplexity: Perplexity is the most relevant metric of properties of a language model for this research. An intuitive definition of perplexity is as a measure of "surprise." In the context of language, unconventional or erroneous use of syntax and semantics in expressions carries more self information and is more "surprising" than the level of surprise associated with expressions that are generated using conventional syntax and semantics. In this context, perplexity can be interpreted as the "level of surprise" of a language analyzer (a human being or a machine)when it encounters a specific expression. Formally, perplexity is defined in the following way:

$$Perplexity = 2^{-\sum_{i=1}^{n} p(x_i) \log_2(p(x_i))} \quad (3)$$

where n is the number of events in X and $p(x_i)$ is the probability of occurrence of event x_i [16]. Low total entropy

[2]Notice that Alice may encrypt the message before using our approach.

of a sample translates into a low level of uncertainty as well as a low level of perplexity. Hence, a low perplexity artificial expression is less surprising. Raising the entropy by a power of two has a normalizing effect on the logarithm, but still maintains the proper proportionality where lower perplexity is equivalent to less surprise.

Secret-to-cover Ratio: The secret-to-cover ratio (STCR) of a steganography technique is the ratio of the secret message size to the cover message size. This ratio is a simple measurement of a steganography technique's efficiency. For prime-based mimic functions, the lower bounds of the STCR can be determined. However, since the construction of the grammar and mapping algorithm used in a prime-based mimic function greatly influences the STCR, an exact ratio must be determined individually theoretically and / or empirically.

The lower-bound of the SCTR is defined as requiring one symbol in the cover message for each prime factor of the mapped value obtained from the mapping algorithm. Since a production in the grammar must have a right-hand side variable and eventually terminate with at least one symbol, a single prime factor must be expressed by at least one symbol.

With the strictest definition of a prime-based mimic function, the upper-bound of the STCR cannot be defined since the number of possible expressions can be infinite. This lack of an upper-bound is caused by the traversal of productions having a prime-cost of 1 arranged in a loop. However, an implementation would limit the traversal of these productions and reduce the upper-bound to a finite, yet still possibly large, value. In fact, in our experiment the STCR exhibited was between 0.028 and 0.069, a value greater than most of the steganography techniques described in the literature survey.

Complexity: The overall complexity of our prime-based mimic function is quite low. Storage of the grammar is based on the number of productions used and thus linear in space complexity. The algorithm for **off-line**, automatic, construction of an automaton that accepts lists of production rules and produces expressions is given in [20]. The algorithm requires four total passes over the production list per expression [20]. Therefore, the time complexity is $4 \times p \times n$, where p is the number of productions in the grammar and n is the number of expressions to create. Since p is constant, the complexity exhibits linear growth. **On-line** the automaton generates expressions in one pass over a list of production rules. Hence, the automaton is linear in p. Note that our method does not require full factorization of large integers, hence the table generation which is done off-line requires linear time. Generating a pointer to the decoder table is done on-line by multiplication and is linear in p. Thus the entire prime-based mimic functions steganography procedure is linear in time complexity as well. More details about the procedures and their complexity can be found in [20].

Robustness: Robustness is the ability of a steganographic technique to resist modifications to the cover message. At the symbol level, a prime-based mimic function resists modification to the generated cover message if the changed symbols are part of productions with the same prime-cost. This, however, has a low probability of occurrence. On the other hand, bit errors due to channel quality or bit manipulation are likely to generate invalid expressions detected and flagged by the decoder. Furthermore, error correction bits and escape symbols can be used to reduce all possible ways of message changes, including bit manipulation.

III. Experimental Results

This section describes the implementation details, experiment methodology, and results.

A. Implementation Details and Experimental Methodology

Each prime-based mimic function requires a language definition to serve as the cover for the secret data. In this research, we implement a mimic function to produce strings within a basic arithmetic expression language that contains expressions such as x=(y+x)/y. We decided to use this example as the first testbed for our method since it is simple enough and enables thorough analysis. On the other hand, it is complex enough to enable identifying problems and issues. A context-free arithmetic expression language can be defined unambiguously and easily meets the prime-numbering criteria discussed above [2]. We define the basic arithmetic language as follows:

$$G^M = (\{E, T, F, G, K\}, \{x, y, +, -, *, /, \hat{}, \$\}, \{E\}, P, \xi) \quad (4)$$

The set P is given by the following productions:

$$\begin{array}{lll} p_0 : E \to T & p_4 : K \to x|y & p_8 : T \to T/F \\ p_1 : T \to F & p_5 : E \to E + T & p_9 : F \to F \hat{} G \\ p_2 : F \to G & p_6 : E \to E - T & p_{10} : F \to F \,\$\, G \\ p_3 : G \to K & p_7 : T \to T * F & p_{11} : G \to (E) \end{array}$$

where the exponential operation is denoted with a caret character (production p_9) and the logarithm operation is denoted by a dollar sign character (p_{10}). The format of the strings contained within this language is similar to many of the expressions found in typical algebra textbooks.

The mapping function ξ is defined by: $\{(p_0, 1), (p_1, 1), (p_2, 1), (p_3, 1), (p_4, 1), (p_5, 2), (p_6, 3), (p_7, 5), (p_8, 7), (p_9, 2), (p_{10}, 3), (p_{11}, 1)\}$. Notice that we have mapped multiple productions to 1. This is used to simplify the implementation by mapping only the productions that contain operational tokens to the set of prime numbers. Furthermore, weights related to probability of occurrence of productions in expressions can be associated with the mapping. We plan to perform future experiments using this feature.

The SRI Language Modeling toolkit (SRILM) is used to analyze five sets of data encoded by the mimic-coder program: uncompressed text, compressed text, encrypted text, random, data, and a file that contains only zeros. Each data set is analyzed using a 4-bit, 8-bit, and 16-bit blocks. The training data used is a set of one-hundred arithmetic expressions from a college algebra textbook. The default language models and

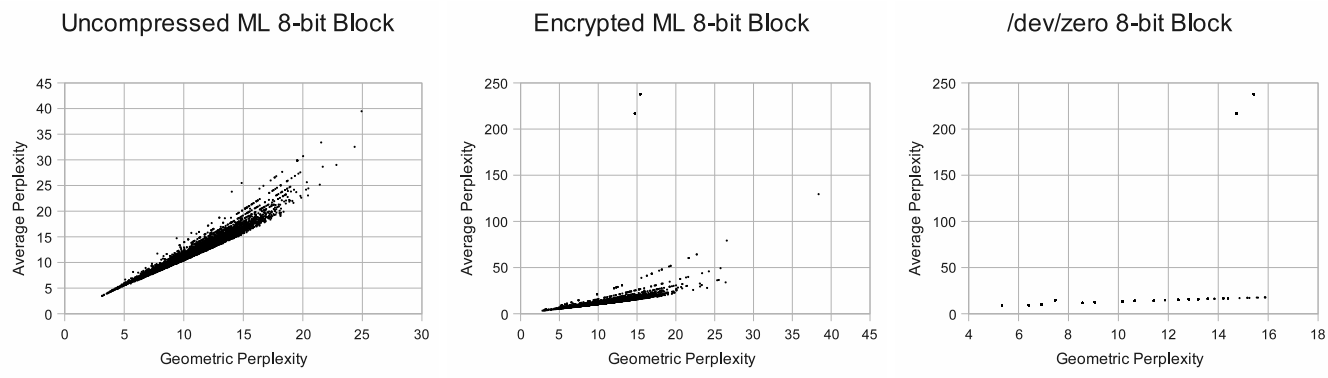

Fig. 2. Perplexities using 8-bit blocks for uncompressed data (Left), encrypted (Middle) and zeroed out data (Right).

parameters presented to the SRILM package were found to be suitable. The general input chosen for the mimic-coder program is page 270 of The Memoirs and Letters of Benjamin Franklin (ML). In this usage, the ML serves as the secret message while the arithmetic expressions generated by the program serve as the cover.

The total size of the ML, representative of the uncompressed data set, is 12,404 bytes. The range of values in the ML is limited to the printable ASCII characters. The ML is compressed using gzip compression and produces a data file of 5,149 bytes. The encrypted data set, created by processing the ML with 'openssl enc -aes-256-cbc -salt', produces a file larger than the original text at 12,432 bytes. For comparison, 12,404 bytes of random data, 12,404 and 12,404 bytes of zeros were encoded separately to be analyzed.

The analysis phase compares the geometric and average perplexities for each respective experiment. Perplexity data points which are close to the origin of the graph show low language-model surprise and thus an expected sentence. The comparison of geometric and average perplexities using SRLIM has been used by Meng [12] and Taskiran [7] to automatically and accurately identify generated cover-text within normal text. Measuring both the arithmetic and geometric averages of perplexity as is important since these features have distinct contribution to the analysis of acceptance of an expression by a language model. Finally, the data sizes before and after encoding were compared across experiments to test relationships between block sizes of the mapping algorithm, the SCTR, and the encoding bit-rate associated with each type of data.

B. Results

Figure 2 shows the perplexities results for the 8-bit block size for uncompressed data (Left), encrypted (Middle), and zeroed text (Right). The geometric perplexity is presented on the x-axis while the average perplexity on the y-axis. With compressed data, we see a tight clustering of data points within (20, 25) to the origin with no outliers. This shows that the language model is accepting the sentences with very little surprise. The encrypted file presents a tight clustering of data points within (20, 50) to the origin which shows the language model accepting the sentences with little surprise. Three outliers are present which represent mapped values of 1 and show both higher geometric and average perplexities. The zeroed out data have a sparse and roughly linear layout of data points between (16, 25) and the origin which show the language model accepting the sentences with very little surprise. Two outliers are present which represent mapped values of 1 and show both higher geometric and average perplexities. The performance of random and compressed data is close to the performance of the encrypted data and thus are not shown.

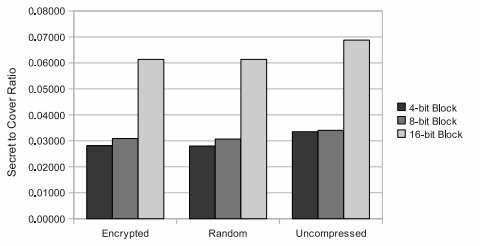

Fig. 3. The Secret-to-Cover ratio.

Figure 3 compares the STCR for the encrypted, random, and uncompressed experiments. The zero-valued byte experiments were omitted since they do not transmit any data, and the compressed file is omitted since it is not "fair" to compare the SCTR of compressed and un-compressed files. The graph shows that the 4-bit and 8-bit block sizes have close SCTR of about 0.03% while the 16-bit block presents an increasee of about twice in the SCTR (0.069). We attribute the sharp increase in the 16-bit case to overhead that is inherent in the algorithm. It affects the relative performance of small size blocks, and is less apparent in blocks of larger size. Overall, the combination of resistance to steganalysis and SCTR obtained is highly competitive (e.g., when compared to bit plans or NICETEXT [1], [5]).

Figure 4 compares the average bit-rate across a T1 telecommunications line for each experiment type and block size. The graph shows the 4-bit and 8-bit block sizes having similar bit-rates while the 16-bit block have a doubled bit-rate. The results

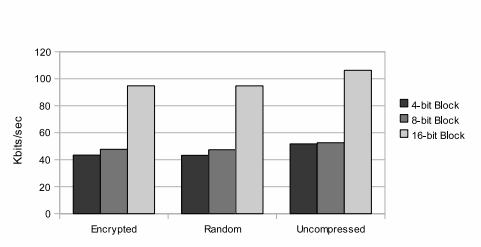

Fig. 4. The achievable bit rate (K bits/sec.

trend is similar to the trend in the SCTR with an increasee of about 50% between 8-bit and 16-bit; which attribute to algorithm overhead. Overall a competitive combination of resistance to steganalysis and high bit rate is obtained.

C. Discussion

An important property of our results is the consistent clustering exhibited. The majority of perplexity measurements lie below a geometric perplexity of 30 and an average perplexity of 50. When comparing the clusters of perplexity measurements to other external measurements, a prime-based mimic function performs quite well. When using a highly trained English-based language model [17]; Katz et al., reported perplexity measurements of 80-120 during a comparison of their language modeling technique and other common techniques. Another report by Brown [18], found that the Brown linguistic corpus exhibited a perplexity of 271 and perplexities of 244 and 236 following data interpolation.One reason however, that the English-based language models exhibit higher perplexities with test data, is that they use a grammar that is trying to emulate natural language, while our CFG is intended for a pure context free language. The analysis of the generated valid and invalid arithmetic expressions shows that all of the expressions generated in the ML experiments, with the exception of the few outliers, exhibit a similar range of perplexities. Additionally, the experiments with random data and zero-valued data also exhibit a similar range of perplexities to that of the generated valid arithmetic expressions. This shows that the expressions generated by the prime-based mimic function are statistically close to actual arithmetic grammar.

Finally, considering the strength of the method, we achieved a competitve SCTR which is higher than most reported results. Bit plans has a higher SCTR (0.125), but are more volunareable to setganalysis.

IV. Conclusions

This paper has presents a new method for steganography that is implemented using prime-based mimic functions. Secret information is encoded in expressions that are accepted by a context-free grammar in a way that would not raise suspicion when observed by a human or machine. The method is very promising and shows a low language model perplexity as well as a competitive bit-rate / STCR.

In the future we plan to investigate additional/ more comlicated grammers and additional ways to map integers to production rules and expressions. In addition, we plan to investigate the option of using our method in order to embed data in expressions that are part of “real messages” transmitted over the media.For example, we would like to use our method to embed data inside HTML expressions transmitted over the Internet. Finally, we plan to assess the utility of using probability weights for production rule selection and the likelihood of certain valid sentences in raising suspicion (for example, long sentences).

References

[1] Katzenbeisser S. and Petitcolas F. A. P., *Information hiding techniques for steganography and digital watermarking*, Artech House, 2000.
[2] J.E. Hopcroft, R. Motwani, and J.D. Ullman, *Introduction to automata theory, languages, and computation*, Addison-wesley, 2006.
[3] S. Murdoch and S. Lewis, “Embedding Covert Channels into TCP/IP,” in *Proceedings of the 7th International Workshop on Information Hiding*, 2005.
[4] M. Guirguis and J. Valdez, “Masquerading a Wired Covert Channel into a Wireless-like Channel,” *Technical Report, Comptuer Science Department, Texas State University*, December 2008.
[5] M. Chapman, “Hiding the Hidden: A Software System for Concealing Ciphertext as Innocuous Text,” *The University of Wisconsin - Milwaukee, Thesis*, 1997.
[6] T. Shon, J. Moon, S. Lee, D. Lee, and J. Lim, “Covert Channel Detection in the ICMP Payload Using Support Vector Machine,” in *Proceedings of ISCIS*, November 2003.
[7] C. Taskiran, U. Topkara, M. Topkara, and E. Delp, “Attacks on Lexical Natural Language Steganography Systems,” in *Proceedings of the SPIE International Conference on Security, Steganography, and Watermarking of Multimedia Contents*, Jan 2006.
[8] G. Fisk and M. Fisk T and C. Papadopoulos and J. Neil, “Eliminating Steganography in Internet Traffic with Active Wardens,” in *Proceedings of the fifth International Workshop on Information Hiding*, 2002.
[9] S. Gianvecchio and H. Wang, “Detecting Covert Timing Channels: An Entropy-based Approach,” in *Proceedings of the 14th ACM conference on Computer and Communications Security (CCS’07)*. October 2007, Alexandria, Virginia.
[10] B. Xu, J. Wang, and D. Peng, “Practical Protocol Steganography: Hiding Data in IP Header,” in *Proceedings of the First Asia International Conference on Modelling and Simulations*, 2007.
[11] S. Cabuk, C. Brodley, and C. Shields, “IP Covert Timing Channels: Design and Detection,” in *Proceedings of the 11th ACM conference on Computer and Communications Security (CCS’04)*. October 2004, Washington, DC.
[12] P. Meng, L. Huang, Z. Chen, W. Yang, and D. Li, “Linguistic Steganography Detection Based on Perplexity,” in *Proceedings of the International Conference on Multimedia Information Technology*, 2008.
[13] P. Moulin and J.A. O’Sullivan, “Information-theoretic analysis of information hiding,” *Information Theory, IEEE Transactions on*, vol. 49, no. 3, pp. 563–593, Mar 2003.
[14] P. Wayner, “Mimic Functions,” *Cryptologia*, vol. XVI, pp. 193–214, July 1992.
[15] P. Wayner, “Strong Theoretical Stegnography,” *Cryptologia*, vol. XIX, pp. 285–289, July 1995.
[16] C. Manning and H. Schtze, “Foundations of Statistical Natural Language Processing,” *Massachusetts Institute of Technology*, 1999.
[17] S. Katz, “Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer,” *IEEE Transactions on Acoustics, Speech and Signal Processing*, 1987.
[18] P. Brown R. Mercer V. Della Pietra and J. Lai, “Class-based N-gram Models of Natural Language,” *Computational Linguistics, vol. 18*, 1992.
[19] G. Hardy and E. Wright, “An Introduction to the Theory of Numbers,” *Oxford: Clarendon Press*, 1960.
[20] W. Connell, “Prime Based Mimic Functions,” *Texas State University - San Marcos, Thesis*, 2009.