Can Matrix Codes be Broken with a Cipher-Text Only Attack?

by

William Gasarch and Jonathan Katz

1 Matrix Codes

Alice wants to sent a message to Bob such that if Eve intercepts it, she cannot decode it. The alphabet is of size s and the message is viewed as a sequence of numbers in the set $\{0, \ldots, s-1\}$.

One way to encrypt is to take a $n \times n$ matrix M of entries from $\{0, \ldots, s-1\}$ with determinant relatively prime to s (so it has an inverse mod s). Then encrypt the first n characters x with Mx, the next n characters y with My, etc.

Eve can clearly find M with a known-plaintext attack. Is there a ciphertext-only attack that works? Throughout this paper we assume the following.

1. Eve knows the alphabet has size s, the dimension of the matrix n, and plaintext frequencies for all unigrams, bigrams, ..., whatever -grams she needs.

2. Given a proposed matrix M Eve can determine if it is the correct matrix or not, e.g., by checking whether M^{-1} applied to the entire ciphertext yields valid English text.

3. Eve has access to a very long ciphertext that we denote $c_1, c_2, \ldots,$ where each c_i is an n-gram.

We will present several attacks and, for each one, say what n has to be to make it infeasible. We assume that any attack that requires more than 2^{128} operations is infeasible. Since the notion of *operation* is informal this should be considered as a guideline rather than a strict rule. As a starting point, note that a brute-force attack that tries all s^{n^2} matrices takes roughly s^{n^2} steps and, for $s = 26$, is only feasible when $n \leq 6$.

2 An Attack Based on n^2-Grams

Let $NUM(n)$ be the set of n-grams that occur with "high" probability. (The probability could be tailored, and the set can be generated by analyzing an existing corpus.) Note that $|NUM(n)| \ll s^n$.

Theorem 2.1 *There is an attack that takes $O(n^2|NUM(n^2)|)$ steps.*

Proof: Let $N = |NUM(n^2)|$, and $NUM(n^2) = \{t_1, \ldots, t_N\}$.. Let $d_1, \ldots, d_n$ be the first n^2 characters in the ciphertext, broken into n-grams.

For each $1 \leq i \leq N$

1. Let $t_i = u_1 \cdots u_n$ where each u is an n-gram.
2. Solve for the entries of the matrix by solving the simultaneous equations $Mu_i = d_i$.
3. Check if the resulting matrix works.

For each $1 \leq i \leq N$ this takes $O(n^2)$ steps. Hence the total time is $O(n^2 N) = O(n^2|NUM(n^2)|)$.

The running time can likely be improved by considering the elements of $NUM(n^2)$ in decreasing order of probability. ∎

3 A Row-by-Row Attack

Theorem 3.1 *There is an attack that takes roughly $n \cdot s^n$ steps.*

Proof: The idea is to determine M^{-1} row-by-row, using exhaustive search. This takes time s^n per row, so time $n \cdot s^n$ overall.

We describe the approach for determining the first row of M^{-1}. For each possible value r of this row, compute $r \cdot c_1$, $r \cdot c_2$, $r \cdot c_3$, ... If the guess r is correct then this yields the initial letter in each n-gram of the plaintext. Those initial letters are expected to follow (known) letter frequencies, and this fact can be used to identify r.

It is interesting to note that r is not unique: in particular, *each* row of M^{-1} is expected to yield the correct letter frequencies. This may actually be a good thing. We can hope that the best n matches yield the n rows of M^{-1}. Then identifying these n matches would take *total* time s^n. To determine the ordering among those rows, we can use bigram analysis on a smaller number of ciphertext blocks. ∎

For $s = 26$ the above attack is feasible for $n \leq 30$. However, we note several ways the attack can (potentially) be improved.

First of all, we can apply the above attack modulo *factors* of s. Let p be any factor of s (it need not be a prime factor). Then we can learn $M^{-1} \bmod p$ by reducing the ciphertext modulo p and using known letter frequencies modulo p. This takes time p^n. There is a tradeoff here: for small p the attack is faster but we learn less information about M^{-1}; more problematic is that letter frequencies may become more smooth as p decreases (though this depends on the initial letter frequencies). For English $n = 26 = 13 \cdot 2$ and, based on known letter frequencies for English text, this approach seems to work for both $p = 2$ and $p = 13$. This would give a total complexity for the attack of $n \cdot (2^n + 13^n)$, which is feasible for $n \leq 30$ or so.

Another idea (which may be combined with the previous one) is to use a lattice-based attack to find a given row r. Let C denote the matrix in which the ith row is c_i. Then $C \cdot r^T$ should be a vector in which each character occurs according to the known letter frequencies. Unfortunately we don't know the permuted order in which the characters will occur. Nevertheless, we expect $C \cdot r^T - c$ to be "short" if c is an appropriately chosen constant vector, and this gives hope that lattice-based algorithms can be applied. It remains to be seen how this plays out in practice.