Attacking Scrambled Burrows-Wheeler Transform

Martin Stanek

Department of Computer Science, Comenius University
Mlynská dolina, 842 48 Bratislava, Slovak Republic
stanek@dcs.fmph.uniba.sk

Abstract

Scrambled Burrows-Wheeler transform [6] is an attempt to combine privacy (encryption) and data compression. We show that the proposed approach is insecure. We present chosen plaintext and known plaintext attacks and estimate their complexity in various scenarios.

1 Introduction

The Burrows-Wheeler transform (BWT) [2] is a commonly used transform in lossless compression algorithms. The BWT does not compress the data itself, instead it is usually the first step in a sequence of algorithms transforming an input data into compressed data. The most prominent example of BWT-based compression is bzip2 program [8], which uses basically the following sequence of algorithms: the BWT, the move-to-front transform (MTF), and Huffman coding.

In practice, there are many situations with the simultaneous requirements of data compression and privacy (encryption). A common approach is compress-then-encrypt paradigm with independent algorithms used for compression and encryption. While widely preferred, the algorithms should be properly combined in order to avoid possible attacks [1, 5]. Another approach is to unify compression and encryption. However, designing a secure, "encrypting" variant of compression method is not an easy task. Many attempts were broken successfully [3, 7, 9].

Recently, Oğuzhan Külekci [6] proposed a novel approach – scrambled BWT – to combine data compression with privacy requirement. The scrambled BWT uses a secret lexicographic order as a secret key. In order to thwart some weaknesses, the author proposed to accompany the scrambled BWT with modified MTF that uses the secret lexicographic order as well.

Our contribution We show that the proposed scrambled BWT with MTF is completely insecure and can be attacked easily (in the sense of chosen plaintext as well as known plaintext attacks). In case of known plaintext attacks we estimate experimentally the bit security of the scrambled BWT with MTF in various scenarios.

We briefly introduce the "standard" BWT, the MTF, and the scrambled BWT in Section 2. Section 3 contains our analysis of the proposal, and shows chosen-plaintext and known-plaintext attacks on the scrambled BWT with MTF.

2 Preliminaries

Let $\mathcal{A}$ be an alphabet with size $L = |\mathcal{A}|$. Let N denotes a block length. A cyclic rotation of string/block $x = x_0x_1 \dots x_{N-1} \in \mathcal{A}^L$ with offset k is string/block $x^{(k)} = x_kx_{k+1} \dots x_{k+N}$ where all the indices are computed modulo N. The w-th symbol of $x^{(k)}$ is denoted as $x_w^{(k)}$, for $1 \leq w < N$.

For real-world scenarios one can expect $L = 256$ (using bytes as an alphabet) and the block size N several hundreds kilobytes (e.g. bzip2 uses block of size 900kB as default).

2.1 "Standard" BWT

Let $x = x_0x_1 \dots x_{N-1} \in \mathcal{A}^L$ be an input block. The BWT sorts all cyclic rotations $x^{(0)}, x^{(1)}, \dots, x^{(N-1)}$ of input block x in lexicographic order. Let $j_0, j_1, \dots, j_{N-1}$ be a permutation of $\{0, 1, \dots, N-1\}$ such that

$$x^{(j_0)} < x^{(j_1)} < \dots < x^{(j_{N-1})}.$$

Then the result of the BWT is a string consisting of the last symbols from each of the sorted cyclic rotations:

$$y = x_{N-1}^{(j_0)}, x_{N-1}^{(j_1)}, \dots, x_{N-1}^{(j_{N-1})}.$$

In order to facilitate the inversion transformation, an additional pointer is used to remember the position of the original string $x = x^{(0)}$, i.e. t such that $j_t = 0$.

2.2 MTF

The MTF transforms an input string $x = x_0x_1 \dots x_{N-1} \in \mathcal{A}^L$ into sequence of N numbers $\{p_i\}_{i=0}^{N-1}$, where $p_i \in \{0, 1, \dots, L-1\}$. The algorithm maintains a table $T[0, \dots, N-1]$ of all symbols from $\mathcal{A}$, initially sorted in lexicographic order. For each $i = 0, 1, \dots, N-1$ the symbol x_i is processed as follows:

1. p_i is the position of x_i in table T;
2. T is modified: x_i is moved to the front/top of the table

It is easy to see that the MTF is invertible. The main idea of MTF is that the recently used symbols are encoded as small integers. This makes its output a suitable data for subsequent compression by simple entropy coders such as Huffman or arithmetic coding.

2.3 Scrambled BWT with MTF

Oğuzhan Külekci [6] proposed scrambled BWT, where a secret (encryption) key is a secret lexicographic order of symbols in $\mathcal{A}$. One of the claimed advantages is a large key space, for $L = 256$ it is 256! keys. The author observed, that using scrambled BWT is not secure enough, and can be attacked by exploiting known statistical relationships among plaintext symbols (e.g. digram frequencies). Therefore he accompanied the scrambled BWT (sBWT) with MTF, where the secret lexicographic order is applied as well (i.e. the initialization of T depends on this order):

> "...Thus, initializing the alphabet ordering in MTF with the secret lexicographic order used in sBWT provides protection against that statistical attack."

Remark 1. The paper [6] uses a variant of the BWT with special symbol denoting the end of block. It that case, there is no need to remember the position of the original block among sorted cyclic rotations. The analysis done in Section 3 is valid for this modification as well.

3 Security analysis

Let us denote the secret lexicographic order as k and the corresponding scrambled BWT as sBWT_k. Similarly, the MTF with secret lexicographic order is denoted as MTF_k. Let x be an input. The author proposes [6] the same secret key (secret lexicographic order) for both transformations: $\mathrm{MTF}_k(\mathrm{sBWT}_k(x))$.

Let us note that attacks described in this sections will work even for situation with two independent secret keys: $\mathrm{MTF}_{k_2}(\mathrm{sBWT}_{k_1}(x))$. The attack will "undo" the MTF (revealing the value of $\mathrm{sBWT}_{k_1}(x)$), and the sBWT part can be attacked by exploiting the statistical properties of plaintext as suggested in [6].

Remark 2. We can ignore other pre- and post-processing steps in the compression algorithm, since they do not depend on the key.

The main observations are the following:

1. The scrambled BWT, with secret lexicographic order as a key, keeps the frequencies of symbols intact, i.e. when symbol '`a`' appears l times

in an input block, then '`a`' will appear exactly l times in the output block.

2. For any input x and any two lexicographic orders k_1, k_2, performing $y = \mathrm{MTF}_{k_2}^{-1}(\mathrm{MTF}_{k_1}(x))$ can permute the symbols but it does not change the frequencies. For example, '`a`' can become '`w`', '`b`' can become '`f`' ... but in such case the number of `a`'s in x is the same as the number of `w`'s in y, the number of `b`'s in x is the same as the number of `f`'s in y etc.

For the rest of the section we denote an input block x and the resulting block $y = \mathrm{MTF}_k(\mathrm{sBWT}_k(x))$. Our analysis is done primarily with single data block (it is sufficient for most scenarios). However, it can be extended to multiple blocks in a straightforward manner, see Section 3.3.

3.1 Chosen plaintext attack

The goal of the attack is to identify a secret key (lexicographic order):

1. Construct input block x, where symbols from $\mathcal{A}$ have unique frequencies.
2. Compute $z^* = \mathrm{MTF}_{k'}^{-1}(y)$ for an arbitrary lexicographic order k'.
3. Pair symbols in x and z^* according their frequencies, and recover the correct "middle" value $z = \mathrm{sBWT}_k(x) = \mathrm{MTF}_k^{-1}(y)$.
4. Reconstruct the key from z and y.

For example, let $L = 256$. Then taking frequencies $1, 2, \ldots, 256$ for symbols in $\mathcal{A}$, allow us to reconstruct the key from a single block as long as the block size is (at least) 32 896.

Example 1. Let us illustrate the attack with the following toy example. Let $\mathcal{A} = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$ and $N = 10$. We choose the input block $x =$ `abbcccdddd`, and observe the output $y =$ `3133002202`. We apply inverse MTF with natural lexicographic order ($\mathtt{a} < \mathtt{b} < \mathtt{c} < \mathtt{d}$): $z^* =$ `dacbbbdccb`. Pairing symbols with equal frequencies yields $z =$ `abcdddbccd`. Knowing "middle" value z and the result of $\mathrm{MTF}_k(z)$, we can easily reconstruct the secret lexicographic order: $\mathtt{b} < \mathtt{d} < \mathtt{c} < \mathtt{a}$.

3.2 Known plaintext attack (single block)

Known plaintext attack extend the previous attack assuming that the attacker cannot control the frequencies of particular symbols in the input data. However, considering the block sizes used in practice (e.g. default 900kB block size in bzip2), the probability of equal frequencies of symbols is rather low.

Remark 3. Moreover, for short blocks, one can assume that the attacker will known the input data for multiple blocks. Therefore he can combine results from these blocks and significantly reduce the complexity of the attack further, see Section 3.3.

Let us denote by $\#_v(x)$ the number of symbols $v \in \mathcal{A}$ in the string x. Let $C(x) = \{\#_v(x) \mid v \in \mathcal{A}\}$ be the set of all distinct values of $\#_v(x)$. For each value $r \in C(x)$ we define $\hat{r}$ to be the number of symbols having exactly r occurrences in x, i.e. $\hat{r} = |\{v \in \mathcal{A} \mid \#_v(x) = r\}|$.

The attacker proceed similarly to the chosen plaintext attack, computing $z^* = \mathrm{MTF}_{k'}^{-1}(y)$ for an arbitrary lexicographic order k'. Then he tries to pair symbols in x and z^* according their frequencies to recover the correct "middle" value $z = \mathrm{sBWT}_k(x) = \mathrm{MTF}_k^{-1}(y)$. However, this time there in no guarantee of unique frequencies, therefore the attacker can perform an exhaustive search for all assignments of symbols in groups with equal frequencies. For each assignment, the attacker computes a corresponding lexicographic order and performs an inverse BWT with this order. Comparing the result with the original plaintext gives a confirmation/rejection of particular assignment. The size of the search space is $\prod_{r \in C(x)} \hat{r}!$.

In order to estimate the complexity of this known plaintext attack, we performed the following experiments:

RandBytes We generate the input block as a stream of randomly and independently generated bytes (i.e. $L = 256$) with uniform distribution. Since in real-world one can expect much more "compressible" input block (with greater variation of symbols frequencies), our model makes the situation for the attacker much worse. Therefore, the estimations can be viewed as an upper bound for attacker's complexity.

RandReduced This is a similar experiment as the previous one, but this time we restrict possible symbols to the set of $L = 100$ symbols (generated with uniform distribution). The rest of the symbols are not generated.

RandText This and the last experiment (RandKernel) are probably the most realistic of our experiments. In RandText we model the input block as a stream of independently generated bytes, where the probabilities of individual symbols correspond to the probabilities of symbols in a novel *Crime and Punishment* [4].

RandKernel Similar experiment to RandText. However, this time the probabilities of individual symbols (bytes) correspond to the probabilities of symbols in `kernel32.dll` file in Windows 7.

We gradually increased N in each experiment and computed the average "bit security/complexity" of the attack: $\log_2(\prod_{r \in C(x)} \hat{r}!)$. The average value was computed from 1000 samples. It is interesting to see the level of bit security degradation from the theoretical level: $\log_2(256!) \sim 1684$ bits of key length. The values for RandReduced, RandText and RandKernel experiments show that the scrambled BWT with MTF offers practically no security. The results are shown in Table 1.

Table 1: Bit security for single block KPA (with block size N)

N	RandBytes	RandReduced	RandText	RandKernel
50 000	378.7	28.6	10.1	65.3
100 000	304.2	20.9	8.4	49.0
150 000	263.7	17.2	8.8	41.4
200 000	236.4	15.2	9.2	36.5

3.3 Known plaintext attack (multiple blocks)

The known plaintext attack from previous section can be easily extended for multiple blocks, with improved performance. The majority of "ties" (groups of symbols with equal frequencies) in one block can be broken by considering the frequencies in other blocks. In order to illustrate this effect, we performed experiments similar to those in the previous section (RandBytes, RandReduced, RandText, and RandKernel). However, this time we used two blocks of known plaintext and 10 times smaller block sizes. The results are shown in Table 2 (again, 1000 samples were used to estimate individual values). Even with such artificially small block sizes one can observe further decrease in the bit security of the scrambled BWT with MTF.

Table 2: Bit security for two blocks KPA (with block size N)

N	RandBytes	RandReduced	RandText	RandKernel
5 000	113.3	4.0	8.4	44.6
10 000	61.0	1.9	5.0	21.4
15 000	41.8	1.3	3.6	13.9
20 000	31.6	1.0	3.2	10.4

4 Conclusion

Usually, providing privacy (encryption) by modifying the data compression techniques is not a good idea in practice. We demonstrated the security problems of the scrambled BWT with MTF. Moreover, it seems that these problems cannot be easily fixed.

Acknowledgment This work was supported by P12/1.

References

[1] Eli Biham and Paul C. Kocher. A known plaintext attack on the PKZIP stream cipher. In Bart Preneel, editor, *FSE*, volume 1008 of *Lecture Notes in Computer Science*, pages 144–153. Springer, 1994.

[2] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

[3] John G. Cleary, Sean A. Irvine, and Ingrid Rinsma-Melchert. On the insecurity of arithmetic coding. *Computers & Security*, 14(2):167–180, 1995.

[4] Fyodor M. Dostoevsky. *Crime and Punishment*, volume 2554. Project Guttenberg, English translation by C. Garnett, 2006.

[5] Tadayoshi Kohno. Attacking and repairing the winZip encryption scheme. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick Drew McDaniel, editors, *ACM Conference on Computer and Communications Security*, pages 72–81. ACM, 2004.

[6] M. Oğuzhan Külekci. On scrambling the Burrows-Wheeler transform to provide privacy in lossless compression. *Computers & Security*, 31(1):26–32, 2012.

[7] Jen Lim, Colin Boyd, and Ed Dawson. Cryptanalysis of adaptive arithmetic coding encryption schemes. In Vijay Varadharajan, Josef Pieprzyk, and Yi Mu, editors, *ACISP*, volume 1270 of *Lecture Notes in Computer Science*, pages 216–227. Springer, 1997.

[8] J. Seward. bzip2, 1996-2012.

[9] Jiantao Zhou, Oscar C. Au, and Peter Hon-Wah Wong. Adaptive chosen-ciphertext attack on secure arithmetic coding. *IEEE Transactions on Signal Processing*, 57(5):1825–1838, 2009.