

On Scrambling the Burrows–Wheeler Transform to Provide Privacy in Lossless Compression[☆]

M. Oğuzhan Külekci

National Research Institute of Electronics & Cryptology, Turkey.

Abstract

The usual way of ensuring the confidentiality of the compressed data is to encrypt it with a standard encryption algorithm. Although the computational cost of encryption is practically tolerable in most cases, the lack of flexibility to perform pattern matching on the compressed data due to the encryption level is the main disadvantage. Another alternative to provide privacy in compression is to alter the compression algorithms in such a way that the decompression requires the knowledge of some secret parameters. Securing the arithmetic and Huffman coders along with the dictionary based schemes have been previously studied, where Burrows–Wheeler transform (BWT) has not been addressed before in that sense. On BWT of an input data it is not possible to perform a successful search nor construct any part of it without the proper knowledge of the lexicographical ordering used in the construction. Based upon this observation, this study investigates methods to provide privacy in BWT by using a randomly selected permutation of the input symbols as the lexicographical order. The proposed technique aims to support pattern matching on compressed data, while still retaining the confidentiality. Unifying compression and security in a single step is also considered instead of the two–level compress–then–encrypt paradigm.

Keywords:

secure compression, Burrows–Wheeler transform (BWT), compressed pattern matching, secure BWT, scrambled BWT

[☆]A preliminary version of this study has been presented in 1st International Conference on Data Compression, Communication and Processing (CCP'11).

Email address: kulekci@uekae.tubitak.gov.tr (M. Oğuzhan Külekci)

1. Introduction

Compression aims to represent an input data with least number of bits. The reduced size of the input yields efficient storage. Another benefit is the better use of the available bandwidth in a network while transmitting the data. As an example, it would be really difficult to store/share/exchange movies or songs without the contemporary multimedia compression techniques.

On the other hand, security and privacy are vital in today's digital life as data is vulnerable to attacks while being transmitted over a network or stored in a multi-user environment. Hence, the users demand for compression combined with security, which gives rise to secure compression paradigm.

In practice, applying a standard encryption algorithm following the compression is widely preferred. Most of the compression software used today include an option to encrypt the output with a password specified by the user. As an example WinZip¹ and WinRAR² packages provide encrypting the compressed data with the AES (Daemen and Rijmen, 2002) algorithm. That compress-then-encrypt approach provides high level of secrecy in case of proper usage, such as selecting strong passwords. Several attacks (Biham and Kocher, 1995; Stay, 2002; Kohno, 2004; Yeo and Phan, 2006) on encrypted compressed data has been published, which mainly benefited from improper injection of the cryptographic primitives into the compression software.

The main disadvantage of encryption is the lack of the opportunity to perform text processing on the compressed data, which means any operation doable on the underlying text without decompressing the whole file (Kim et al., 2007). As an example, seeking keywords in compressed data is a frequent daily practice, where compressed pattern matching (Amir, 1996; Shibata et al., 1999; Tao, 2005) methods may be used. When encryption level is appended over compression, the compressed string search algorithms may only be applied after decryption.

A second approach in securing the compressed data is to unify compression and encryption, which seems more elegant than a pure encryption scheme in a sense that both compression and security requirements are fulfilled in a single step. Creating multiple statistical models and hopping between them

¹www.winzip.com

²www.rarlab.com

according to a bit stream generated via a stream cipher during the compression is one of the ways for such an integration. It might be preferred to encrypt the coefficients used in the compression algorithm (Zeng and Lei, 2003), when the target is a multimedia file such as video. Some alterations in the entropy coders like the interval splitting in arithmetic coding (Kim et al., 2007) or using multiple code-trees in Huffman coding (Wu and Kuo, 2005) have been also studied to provide a level of secrecy. Using encrypted dictionaries (Govindan, 2004) in dictionary-based compression is yet another alternative. The alterations performed on compression algorithms to provide security might decrease the compression ratio, where the absence of efficient methods to perform compressed pattern matching on concurrently compressed and secured data is again a major disadvantage.

Although securing the arithmetic and Huffman coders along with the dictionary based schemes have been previously studied, Burrows-Wheeler transform (BWT) (Burrows and Wheeler, 1994) has not been addressed before in that sense. This study investigates scrambling the BWT to secure the compressed files. The main idea is based on the observation that the reconstruction of the original data from its BWT or the backwards search (Ferragina and Manzini, 2000) of a pattern require the knowledge of the lexicographical ordering used in the construction of the BWT. When a random permutation of the symbols is used as the lexicographic ordering in BWT, it is hard to deduce this ordering. We refer the BWT of an input data with a secret character ordering as *scrambled-BWT* or *sBWT* in short.

The sBWT followed by the classical move-to-front (MTF) and run-length encoding can unify compression and security simultaneously. Despite the garbling achieved by sBWT via the randomly chosen lexicographical order, the MTF procedure initialized with that secret ordering introduces a secondary scrambling akin to double encryption schemes. Considering the efficient pattern matching methods previously developed on BWT+MTF compressed data (Bell et al., 2002), we can ensure the confidentiality of the compressed data while preserving the ability of efficient searching. As oppose to other techniques of this genre, removing the necessity of running a concurrent stream cipher while compressing/decompressing is yet another advantage.

The sBWT method can also be used in place of the encryption step in compress-then-encrypt approaches. When sBWT replaces the encryption, it is possible to retrieve any part of the compressed text or even search a specific pattern via the backwards searching. This option is not available in

a standard ciphering system, since decoding of t_i requires the processing of all previous characters t_1 to t_{i-1} in general. Thus, sBWT may be useful if one needs to retrieve an arbitrary part of the compressed text or the compressed pattern matching technique of the underlying algorithm performs shifts over the data as in the study of Shibata et al. (2000).

The outline of the paper is as follows. We survey the previous studies in section 2 followed by the review of the BWT and backwards searching in section 3. The proposed method is introduced and discussed in sections 4 and 5. We conclude with the summary and further opportunities.

2. Previous Work on Securing the Compression

The two general methodologies used in practice to secure the compressed data are sketched in Figure 1. The first approach, which we refer as compress-then-encrypt, is mostly preferred by commercial compression solutions such as the WinRAR and WinZip, where both use AES algorithm since 2002 and 2004 respectively. Prior to AES usage, the security features of WinZip were investigated and some early attacks were proposed by Biham&Kohen (1995) followed by the study of Stay (2002). Kohno (2004) showed that there were still vulnerabilities after the inclusion of the AES on version 9.0 of the WinZip software. It worths here to note that those attacks were not attempting to break the AES cipher, but benefiting from some weaknesses originating from the improper integration of compression software and the encryption. Similar attacks on WinRAR were also studied by Yeo&Phan (2006).

The compress-then-encrypt schemes provide high level secrecy with the encryption algorithms that are proved to be secure as a second layer. Although the additional computational cost of encryption is affordable on today's powerful processors, the major drawback is the lack of opportunities to perform search on the compressed data. There exists methods (Amir, 1996; Shibata et al., 1999; Shapira and Klein, 2006; Bell et al., 2002) to run pattern matching on compressed data without decompressing it. However, the encryption level inhibits this opportunity. It might be questioned that there are also some studies aiming searchable encryption (Abdalla et al., 2008). Unfortunately, the cryptographic primitives used in those searchable encryption algorithms are far more slow and resource demanding than any standard algorithm, which makes them practically infeasible for today.

The second approach in securing the compressed data is to integrate compression and encryption in a single framework. One of the earliest studies on

this area by Witten&Cleary (1988) proposed keeping the underlying probability model hidden in an adaptive arithmetic coder for security. Bergen&Hogan (1992; 1993) showed that this was not appropriate by introducing an attack capturing the secret statistical model of the coder, where Liu *et al.* (1997) introduced an alternative method to defer the Bergen&Hogan attack. Cleary *et al.* (1995) described that arithmetic coding is not secure and Lim *et al.* (Lim et al., 1997) gave a complete cryptanalysis. Barbir (1997) proposed an alteration in arithmetic coding to make it secure. The idea was to update the statistical model not after the compression of each symbol, but only at some specific points defined by a concurrently running stream cipher. This semi-adaptive time-variant modeling scheme degraded the compression efficiency for security. Randomized arithmetic coding by Grangetto *et al.* (2004; 2006) again incorporated a stream cipher such that the probability intervals of characters were shuffled during the compression according to the bits generated by the cipher. A similar study by Kim *et al.* (2007) combined shuffling with interval splitting, where the probability interval of a symbol was defined no more to be contiguous, but split up into several pieces. Recently, Zhou *et al.* (2009) described a chosen-cipher-text attack on that secure arithmetic coding. Despite numerous efforts to make arithmetic coding secure, there has been also some attempts focusing on other compression techniques. For example, Wu&Kuo (2005) incorporated multiple code-trees in Huffman coding that are based on multiple statistical models used in a secret ordering. Zhou *et al.* (2008) concentrated on dictionary based compression and proposed a method to secure Lempel-Ziv-Welch (Welch, 1984) compression.

Unified compression and encryption methods generally include a stream cipher that is run concurrently with the compression algorithm. The entropy coders are tuned to execute on multiple models in a secret ordering that is specified by the output of that stream cipher. Altering a compression scheme to provide security usually causes a decrease in compression performance and speed. The disadvantage of missing flexibility to perform pattern matching on the compressed data is still a problem of this genre as is the case for the compress-then-encrypt approach.

3. The Burrows–Wheeler Transform and the Backwards Searching Paradigm

3.1. The Burrows–Wheeler Transform

Let the input text be denoted by $T = t_1t_2\dots t_n$, where $t_i \in \Sigma$ for $1 \leq i \leq n$ and Σ represents the alphabet of characters occurring in T as $\Sigma = \{\epsilon_1, \epsilon_2, \dots, \epsilon_k\}$. The cardinality of Σ is $|\Sigma| = k$. While computing the BWT of T , a hypothetical end-of-sequence character $\$,$ which is not in Σ and lexicographically smaller than any other symbol $\epsilon_i \in \Sigma$, $1 \leq i \leq k$, is assumed to be inserted at the end of T .

The cyclic–right–shift s of T is $CRS_s(T) = t_s t_{s+1} \dots t_n \$ t_1 t_2 \dots t_{s-1}$. We lexicographically sort all $CRS_s(T)$ strings for $1 \leq s \leq (n + 1)$. The BWT of T is formed by taking the corresponding rightmost characters of each $CRS_s(T)$ in the sorted order, which is shown in Figure 2 on a sample input text $T = \text{mississippi}$. The last column $L = \text{ipssm\$pissii}$ depicted in Figure 2 is the resulting $BWT(T)$.

3.2. Reconstructing Original Text from Its BWT

The procedure to reconstruct the original text T from its BWT is given in Algorithm 1. The initial step is to compute the first column F from the input $L = BWT(T)$. Column F is simply the lexicographical ordering of the characters appearing in L as can be observed at figure 2 c). Once the ordering among the symbols of Σ is known, F can be calculated easily by sorting the characters of L . We need to define the functions $rangeF(\epsilon_i)$ and $rankL(pos, ch)$ to continue with the reconstruction.

- $[begin, end] \leftarrow rangeF(\epsilon_i)$: $begin$ and end are the beginning and ending indices of ϵ_i on F , e.g., $rangeF(p) = [7, 8]$ in Figure 2.
- $q \leftarrow rankL(pos, ch)$: q is the number of characters that are equal to ch up to position pos in L , e.g., $rankL(8, i) = 1$ in Figure 2.

Since $CRS_{n+1} = \$t_1t_2\dots t_n$ appears at top of the sorted list regarding to the assumption that $\$$ is the least significant symbol, the first character of L is always the last character of T as $t_n = \ell_1$. Thus, we set the beginning position on L as $j = 1$. We continue by finding the range of ℓ_j on the F column and move forward from the beginning of this range equal to the number of ℓ_j characters observed up to point j on L . We update the value of j equal to this position's index. The next character in reverse order of the text is the

Input: $BWT(T) = L = \ell_1\ell_2 \dots \ell_{n+1}$
Output: $T = t_1t_2 \dots t_n$

```

1  $F \leftarrow \text{construct}F(L)$ ;
2  $j \leftarrow 1$ ;
3 for  $i = n$  down to 1 do
4    $t_i \leftarrow \ell_j$ ;
5    $[begin, end] \leftarrow \text{range}_F(\ell_j)$ ;
6    $q = \text{rank}_L(j, \ell_j)$ ;
7    $j \leftarrow begin + q$ ;
8 end

```

Algorithm 1: ReconstructText

corresponding symbol ℓ_j on L column. Same procedure is repeated until the complete text is reconstructed.

3.3. Searching a Pattern via BWT

Input: Pattern $P = p_1p_2 \dots p_m$, $BWT(T) = L = \ell_1\ell_2 \dots \ell_{n+1}$
Output: How many times P appears in T ?

```

1  $F \leftarrow \text{construct}F(L)$ ;
2  $i \leftarrow m$ ;  $[begin, end] \leftarrow \text{range}_F(p_m)$ ;
3 while  $i > 1$  do
4   if  $(end - begin) < 1$  then return 0;
5    $q_1 \leftarrow \text{rank}_L(begin, p_{i-1})$ ;
6    $Pq_2 \leftarrow \text{rank}_L(end, p_{i-1})$ ;
7   if  $\ell_{end} = p_{i-1}$  then  $q_2 \leftarrow q_2 + 1$ ;
8    $[begin, end] \leftarrow \text{range}_F(p_{i-1})$ ;
9    $end \leftarrow begin + q_2 - 1$ ;
10   $begin \leftarrow begin + q_1$ ;
11   $i = i - 1$ ;
12 end
13 return  $end - begin + 1$ ;

```

Algorithm 2: FindPattern

We use the same rank_L and range_F primitives in searching for a given pattern $P = p_1p_2 \dots p_m$ as indicated in Algorithm 2. After computing the F column from the given L column, we proceed on P in reverse order by setting

the initial value i to m , and the initial range to $rangeF(p_m)$. The initial range on F is narrowed down according to the number of p_{i-1} characters observed on L up to the *begin* and *end* positions of this range. The main reason is the property of BWT that the h^{th} occurrence of a symbol on the L column corresponds to its h^{th} appearance on F column. We continue this process until the range is empty, which indicates P is not observed in T , or we reach to the beginning of P with this traversal, which means number of occurrences of P is equal to the length of the range. More details on backwards searching can be obtained from (Ferragina and Manzini, 2000) and (Navarro and Mäkinen, 2007).

4. Proposed Method

Assume that a permutation of the symbols among the alphabet of the input file \mathcal{F} is randomly selected and used as the lexicographical order while sorting the cyclic-right-shifts in BWT computation. The F column, which is simply the sorted sequence of symbols appearing on the L column, can not be calculated correctly in absence of the proper lexicographical ordering. Once F is miscalculated, it becomes not possible to reconstruct file \mathcal{F} nor perform backwards search since both require construction of F from L as an initial step as shown in Algorithms 1 and 2.

Based on this observation we propose to use Burrows-Wheeler transform with a secret lexicographical ordering to support security in data compression. Assume user \mathcal{U} wants to compress file \mathcal{F} and keep it secure from others. With this aim, \mathcal{U} selects a random permutation $\mathcal{O} = o_1 < o_2 < \dots < o_k$, where for all $1 < i \leq k$, $o_i \in \Sigma$ and $o_i \neq o_j$ while $i \neq j$, and keeps it secret. We will investigate the possible solutions of achieving compression and security by using the BWT computed according to the secret ordering under two sections as *scrambled-BWT* (sBWT) and *compress-then-sBWT*. Both approaches are depicted in Figure 3.

4.1. sBWT: Injecting security into BWT-based compression schemes

\mathcal{U} first computes the BWT of file \mathcal{F} according to \mathcal{O} , followed by the classical move-to-front and run-length encoding to achieve the compression. It is possible to perform fast search on such BWT compressed sequences (Bell et al., 2002). Thus, \mathcal{U} preserves the ability to perform efficient pattern matching on compressed data in such a scheme. Keeping the randomly selected

lexicographical ordering \mathcal{O} secret provides confidentiality of the compressed data as it is not possible to open the archive without \mathcal{O} .

The only way of a malicious adversary to reach the information content of \mathcal{F} is to capture \mathcal{O} . There exists $|\Sigma|!$ possible distinct permutations of symbols in Σ . The brute-force attack requires $|\Sigma|!$ decompression and verification according to an intelligibility criteria. For sufficiently large $|\Sigma|$ (e.g. $20! > 2^{61}$, $25! > 2^{83}$, $100! > 2^{526}$) this becomes impractical. If the alphabet size is not large enough to avoid the brute-force attack, say less than 20, it is always possible to represent the data with a super alphabet by combining consecutive symbols. In such a case the search procedure of geometric Burrows–Wheeler transform (Chien et al., 2008) may be followed for efficient pattern matching.

If there exists a statistical relationship between the characters of the input file, as is the case for natural languages, there exists a more convenient attack. It is possible to get the number of occurrences, $freq_i$, of each character $\epsilon_i \in \Sigma$ by simply counting on the L column. The problem is than to correctly determine the F column, which also means capturing the secret \mathcal{O} since the characters are sorted according to their lexicographic order on F . For any $\epsilon_i \in \Sigma$, assume r_i indicates its starting position on F . Thus, the range of ϵ_i on F is $[r_i, r_i + freq_i - 1]$. Remembering from BWT that l_i and f_i symbols are consecutive characters in file \mathcal{F} , and assuming that the adversary has the knowledge of the digram probability distribution of \mathcal{F} , it is easy to guess the range of ϵ_i by traversing the possible ranges and calculating the digram distributions. The value of r_i can be any value between 1 and $n - freq_i + 1$. On each possible interval $[r_i, r_i + freq_i - 1]$ the digram statistics of $\{l_{r_i}\epsilon_i, l_{r_i+1}\epsilon_i, \dots, l_{r_i+freq_i-1}\epsilon_i\}$ are computed. The interval that has the closest match to the original digram probabilities is the correct one. This procedure requires $n - freq_i + 1$ digram frequency calculations. If we sum over all symbols in Σ , $\sum_{i=1..k} (n - freq_i + 1) = n \cdot (k - 1) + k$, the complexity of constructing correct F is $O(n \cdot |\Sigma|)$.

In general, following the BWT, a BWT-based compressor performs a move-to-front (MTF) coding (Bentley et al., 1986; Gagie and Manzini, 2007), where the individual symbols are coded with numbers corresponding to their relative positions in the alphabet (Kaplan and Verbin, 2007). Main mechanism of MTF is to change the ordering of the alphabet such that the the most recently coded character is moved to the first position after each encoding. As it is expected to have local runs of the identical characters in BWT of an input text T , MTF results in a sequence of small numbers with skewed probabilities, and thus, improves the compression ratio.

Considering a secure BWT-based compression system as sBWT+MTF followed by an entropy encoding, one needs to reverse the MTF coding to obtain the actual sBWT sequence in order to apply the statistical attack mentioned above. Bringing back the correct sequence in MTF requires the knowledge of the correct alphabet ordering used at the first step. Thus, initializing the alphabet ordering in MTF with the secret lexicographical order used in sBWT provides protection against that statistical attack.

At first sight, it might be argued that using a secret ordering solely in MTF would be enough to secure the BWT-based compression and replacing BWT with sBWT is not so meaningful in that sense. However, with a careful analysis, once the opponent has the statistical properties of the raw data and standard BWT is used before MTF with a secret ordering, it is again possible to run the same idea to capture the initial ordering used in MTF.

As an example assume we are compressing an English text, and computed BWT with the standard lexicographical ordering followed by the MTF with a secret initial symbol ranking. Since letter *e* is the most frequent symbol, it is not difficult to guess the first interval of the *F* column, and hence, the corresponding range in *L* column. With the statistics of the **e* digrams, the adversary can run a procedure accordingly to capture the initial symbol ranking used in MTF. On the other hand, if we follow the path of sBWT+MTF with the secret ordering, it is not trivial for the attacker to reverse this chain as he/she will not have any idea about the input statistics of MTF, and thus, will not be able to bring back the sBWT data from MTF. This is something like a double encryption scheme. Note that since the MTF step alters the alphabet at each symbol coding, the resulting sequence is no more a mono-alphabetic simple substitution, but rather a poly-alphabetic scheme that provides strength against statistical attacks.

4.2. Compress-then-sBWT: Using sBWT on securing the compressed data

The statistical attack on sBWT can be applied if and only if the digram statistics of \mathcal{F} is known or easy to guess. When the input data is homophonic meaning that the frequency distribution of symbols is flat, the adversary cannot benefit from that procedure. Compression removes the redundancy in the data and produces homophonic output. There is no way to apply statistical attacks on homophonic sequences. Therefore, the encryption algorithms are generally deployed after compression in practical usage, even if the aim does not require to compress.

If the user wants to keep the ability to perform pattern matching on the compressed data, it would be useful to select a compression scheme on which compressed pattern matching is defined, e.g. (Bell et al., 2002; Shapira and Klein, 2006; Tao, 2005). When sBWT is applied after such an algorithm, \mathcal{U} has the capability of reconstructing any part of the compressed data that gives the flexibility to run the related search procedure of the preferred algorithm. This type of secure compression solution is depicted in Figure 3.

Replacing encryption with sBWT provides user the ability to reconstruct any part of the compressed data (assuming the underlying compression scheme lets partial decompression, e.g., static Huffman coding) or even search for a specific pattern. When \mathcal{U} would like to retrieve a specific part of the data, he/she should run the related decryption till that position in case of a standard ciphering system, which is not mandatory in sBWT scheme. Thus, for long archives it might be more advantageous to prefer compress-then-sBWT than a usual compress-then-encrypt solution. The availability of this option also helps when the underlying compressed pattern matching algorithm is sub-linear and performs arbitrary jumps while traversing the text (Shibata et al., 2000). On the other hand, it should also be considered that computing the sBWT of the compressed data might require more computational resource during the construction than a simple encryption scheme. Therefore, cons and pros of either solution should be carefully analyzed in practice to decide which way to follow according to the requirements of the actual system.

5. Discussion

The scrambled-BWT proposal aims to integrate compression and security in a single framework for BWT-based schemes, where compress-then-sBWT provides an alternative replacement of encryption in compress-then-encrypt approaches. When we compare sBWT with the works such as the interval splitting or randomized arithmetic coding or multiple Huffman coding, we do not need to run a stream cipher along with the compression and preserve the capability to perform compressed pattern matching with methods allowing efficient search on BWT-based compression (Bell et al., 2002).

Despite removing the necessity of a concurrent cipher, the sBWT method does not introduce any additional computational complexity into the standard BWT calculation. Once the secret lexicographical ordering is specified, the rest of the process is the classical BWT compression without any sacri-

face from compression ratio for the security. It can easily be integrated into existing software and provides an elegant way of preserving confidentiality.

Besides preserving privacy in text compression, sBWT may also serve for secure text indexing within the frame work of FM-index (Ferragina and Manzini, 2000). Last decade witnessed great developments with the advent of compressed self-indexes (Hon et al., 2010) that are succinct structures with space complexities proportional to the compressed size of the input data and enable fast search functionalities. FM-index is a pioneer work of such based on the BWT and backwards searching, where sBWT might serve as a simple solution to ensure confidentiality.

The compress-then-sBWT method replaces the encryption step in compress-then-encrypt approaches with the sBWT. Unlike compress-then-encrypt methods, compress-then-sBWT preserves the opportunity to run compressed pattern matching as long as the selected compression scheme supports. When compared with a standard encryption algorithm such as the AES, sBWT seems more elegant by removing the necessity to deal with cryptographic primitives. Assuming that the alphabet of a compressed file would compose of 256 bytes, there are $256!$ distinct orderings to be used by sBWT. Remembering that the key space of AES is 2^{256} , the proposed sBWT+MTF type compression provides much larger key space as $256! \gg 2^{256}$.

However, we need to state that a standard encryption using cryptographic primitives provides provable security. On the other hand, the security in sBWT with a secret ordering is more in the practical domain than theory from a cryptographic point of view. Note that this is actually the case for most of the secure compression schemes we investigated in section 2. Thus, for highly confidential data, it might still be more appropriate to consider a standard encryption.

6. Conclusions

A major amount of literature on secure compression had focused on providing secrecy on arithmetic coding with some limited attempts targeting Huffman or dictionary methods. This study proposed an alternative way to preserve the confidentiality of the BWT-based compression with the key observation that it is not possible to reconstruct the original text nor perform any search without the correct lexicographical order used in the construction. A randomly selected permutation of characters from the input alphabet is assumed to be the lexicographical order in BWT, and this permutation is

kept as the secret key of the system. Alphabet reordering was previously applied to enhance the compression ratio (Chapin and Tate, 1998), but to the best of our knowledge it is the first time used to ensure the security.

Two schemes as *scrambled-BWT* (*sBWT*) and *compress-then-sBWT* are discussed. Compress-then-sBWT is especially useful in keeping the content of the compressed files hidden such as establishing the privacy of the JPEG images. Scrambled-BWT unifies compression and security in a single step. Retaining the possibility to perform compressed pattern matching is the major advantage, where prior studies did not pay much attention. Further investigations of provably secure techniques in Burrows-Wheeler transform is an open problem, which may lead to significant contributions in practice on privacy preserving text indexing.

An immediate opportunity of the idea presented in the study appears for BWT-based compression software like the *bzip2*³. Those programs might consider to include an option that would let the users specify their own private lexicographical ordering to be used in BWT of the input file, and thus, ensure confidentiality of their data along with the compression. It is also possible to integrate the idea into the FM-index to provide privacy in compressed text indexing.

References

- Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P., Shi, H.. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. *Journal of Cryptology* 2008;21:350–391.
- Amir, A.. Let sleeping files lie: Pattern matching in Z-compressed files. *Journal of Computer and System Sciences* 1996;52:299–307.
- Barbir, A.. A methodology for performing secure data compression. In: *Proceedings of the 29th Southeastern Symposium on System Theory*. 1997. p. 266.
- Bell, T., Powell, M., Mukherjee, A., Adjeroh, D.. Searching BWT compressed text with the Boyer-Moore algorithm and binary search. In: *Proceedings of the Data Compression Conference*. 2002. p. 112–121.

³www.bzip2.org

- Bentley, J.L., Sleator, D.D., Tarjan, R.E., Wei, V.K.. A locally adaptive text compression scheme. *Communications of the ACM* 1986;29:11.
- Bergen, H., Hogan, J.. Data security in a fixed-model arithmetic coding compression algorithm. *Computers & Security* 1992;11(5):445–461.
- Bergen, H., Hogan, J.. A chosen plaintext attack on an adaptive arithmetic coding compression algorithm. *Computers & Security* 1993;12(2):157–167.
- Biham, E., Kocher, P.. A known plaintext attack on the PKZIP stream cipher. In: *Proceedings of Fast Software Encryption'94*. 1995. p. 144–153.
- Burrows, M., Wheeler, D.J.. A block-sorting lossless data compression algorithm. Technical Report 124; DEC SRC; 1994.
- Chapin, B., Tate, S.R.. Higher compression from the BWT by modified sorting. In: *Proceedings of the Data Compression Conference*. 1998. p. 532.
- Chien, Y.F., Hon, W.K., Shah, R., Vitter, J.S.. Geometric Burrows-Wheeler transform: Linking range searching and text indexing. In: *Proceedings of the Data Compression Conference*. 2008. p. 252–261.
- Cleary, J., Irvine, S., Rinsma-Melchert, I.. On the insecurity of arithmetic coding. *Computers & Security* 1995;14(2):167–180.
- Daemen, J., Rijmen, V.. *The Design of Rijndael: AES—the Advanced Encryption Standard*. Springer, 2002.
- Ferragina, P., Manzini, G.. Opportunistic data structures with applications. In: *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*. 2000. p. 390–398.
- Gagie, T., Manzini, G.. Move-to-front, distance coding, and inversion frequencies revisited. In: *Combinatorial Pattern Matching*. volume 4580 of *Lecture Notes in Computer Science*; 2007. p. 71–82.
- Govindan, V.. An intelligent text data encryption and compression for high speed and secure data transmission over internet. In: *Proceedings of IIT Kanpur Hackers Workshop*. 2004. .

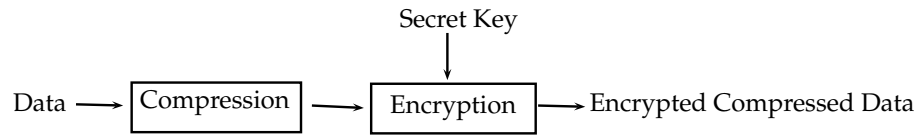
- Grangetto, M., Grosso, A., Olmo, G.. Selective encryption of JPEG2000 images by means of randomized arithmetic coding. In: Proceedings of IEEE 6th Workshop on Multimedia Signal Processing. 2004. p. 347–350.
- Grangetto, M., Magli, E., Olmo, G.. Multimedia selective encryption by means of randomized arithmetic coding. IEEE Transactions on Multimedia 2006;8(5):905–917.
- Hon, W.K., Shah, R., Vitter, J.S.. Compression, indexing, and retrieval for massive string data. In: Combinatorial Pattern Matching. volume 6129 of *Lecture Notes in Computer Science*; 2010. p. 260–274.
- Kaplan, H., Verbin, E.. Most Burrows–Wheeler based compressors are not optimal. In: Combinatorial Pattern Matching. volume 4580 of *Lecture Notes in Computer Science*; 2007. p. 107–118.
- Kim, H., Wen, J., Villasenor, J.. Secure arithmetic coding. IEEE Transactions on Signal Processing 2007;55(5):2263–2272.
- Kohno, T.. Attacking and repairing the WinZip encryption scheme. In: Proceedings of the 11th ACM Conference on Computer and Communications Security. 2004. p. 72–81.
- Lim, J., Boyd, C., Dawson, E.. Cryptanalysis of adaptive arithmetic coding encryption scheme. In: Proceedings of the 2nd Australasian Conference on Information Security and Privacy. 1997. p. 216–227.
- Liu, X., Farrell, P., Boyd, C.. Resisting the Bergen-Hogan attack on adaptive arithmetic coding. In: Proceedings of the 6th IMA International Conference on Cryptography and Coding. 1997. p. 199–208.
- Navarro, G., Mäkinen, V.. Compressed full-text indexes. ACM Computing Surveys 2007;39(1).
- Shapira, D., Klein, S.. Compressed pattern matching in JPEG images. International Journal of Foundations of Computer Science 2006;17(6):1297–1306.
- Shibata, Y., Kida, T., Fukamachi, S., Takeda, M., Shinohara, A., Shinohara, T., Arikawa, S.. Byte Pair encoding: A text compression scheme that accelerates pattern matching. Technical Report; Department of Informatics, Kyushu University, Japan; 1999.

- Shibata, Y., Matsumoto, T., Takeda, M., Shinohara, A., Arikawa, S.. A Boyer-Moore type algorithm for compressed pattern matching. In: Combinatorial Pattern Matching. volume 1848 of *Lecture Notes in Computer Science*; 2000. p. 181–194.
- Stay, M.. ZIP attacks with reduced known plaintext. In: Fast Software Encryption. 2002. p. 411–429.
- Tao, T.. Compressed pattern matching for text and images. Ph.D. thesis; Orlando, FL, USA; 2005. AAI3178974.
- Welch, T.A.. A technique for high-performance data compression. *IEEE Computer* 1984;17(6):8–19.
- Witten, I., Cleary, J.. On the privacy afforded by adaptive text compression. *Computers & Security* 1988;7(4):397–408.
- Wu, C., Kuo, C.. Design of integrated multimedia compression and encryption systems. *IEEE Transactions on Multimedia* 2005;7(5):828–839.
- Yeo, G., Phan, R.. On the security of the WinRAR encryption feature. *International Journal of Information Security* 2006;5(2):115–123.
- Zeng, W., Lei, S.. Efficient frequency domain selective scrambling of digital video. In: *IEEE Transactions on Multimedia*. volume 5; 2003. p. 118.
- Zhou, J., Au, O., Fan, X., Wong, P.. Secure Lempel-Ziv-Welch algorithm with random dictionary insertion and permutation. In: *IEEE International Conference on Multimedia and Expo*. 2008. p. 245–248.
- Zhou, J., Au, O., Wong, P.. Adaptive chosen-ciphertext attack on secure arithmetic coding. *IEEE Transactions on Signal Processing* 2009;57(5):1825–1838.

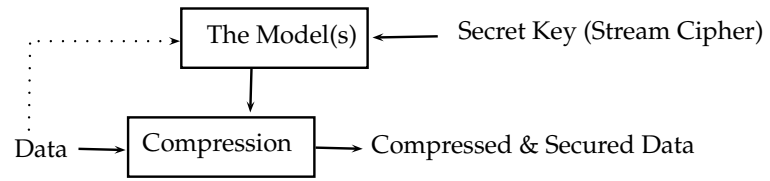
Figure 1. General methods to secure compressed data.

Figure 2. The BWT of sample text `mississippi`.

Figure 3. Proposed methods to ensure confidentiality of the compressed data.



a) Approach 1 : Compress-then-encrypt



b) Approach 2: Unifying compression and security

Figure 1: General methods to secure compressed data

s	$CRS_s(T)$	s	$CRS_s(T)$	i	F		L	i
1	mississippi\$	12	\$mississippi	1	\$	mississipp	i	1
2	ississippi\$m	11	i\$mississippi	2	i	\$mississip	p	2
3	ssissippi\$mi	8	ippi\$mississ	3	i	ppi\$missis	s	3
4	sissippi\$mis	5	issippi\$miss	4	i	ssippi\$mis	s	4
5	issippi\$miss	2	issippi\$m	5	i	ssissippi\$	m	5
6	ssippi\$missi	1	mississippi\$	6	m	ississippi	\$	6
7	sippi\$missis	10	pi\$mississip	7	p	i\$mississi	p	7
8	ippi\$mississ	9	ppi\$mississi	8	p	pi\$mississ	i	8
9	ppi\$mississi	7	sippi\$missis	9	s	ippi\$missi	s	9
10	pi\$mississip	4	sissippi\$mis	10	s	issippi\$mi	s	10
11	i\$mississipp	6	ssippi\$missi	11	s	sippi\$miss	i	11
12	\$mississippi	3	ssissippi\$mi	12	s	sissippi\$m	i	12

a) Cyclic-right-shifts

b) Sorted CRS

c) $BWT(T) = L$

Figure 2: The BWT of sample text mississippi.

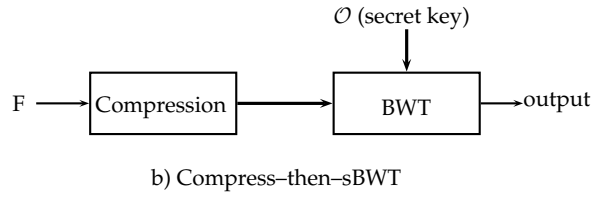
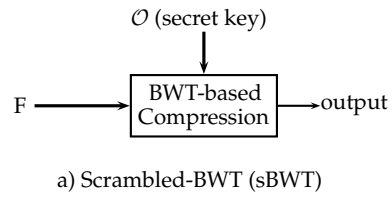


Figure 3: Proposed methods to ensure confidentiality of the compressed data.