Puzzle - A Novel Video Encryption Algorithm

Fuwen Liu, Hartmut Koenig

Brandenburg University of Technology Cottbus,
Department of Computer Science
PF 10 33 44, 03013 Cottbus, Germany
{lfw,koenig}@informatik.tu-cottbus.de

Abstract. Networked multimedia applications have matured in recent years to be deployed in a larger scale in the Internet. Confidentiality is one of the primary concerns of these services for their commercial usages, e.g. in video on demand services or in video conferences. In particular, video encryption algorithms are strongly required that fulfill real-time requirements. In this paper we present the video encryption algorithm *Puzzle* to encrypting video streams in software. It is fast enough to fulfill real-time constraints and to provide a sufficient security. *Puzzle* is a video compression independent algorithm which can be easily incorporated into existing multimedia systems.

1 Introduction

Due to significant advances in video compression and networking technologies, networked multimedia applications, e.g. video on demand or video conferences, are becoming increasingly popular. Confidentiality is one of the primary concerns for their commercial use. This issue is usually addressed by encryption. Only authorized parties who possess the decryption keys are able to access to the clear multimedia contents. While for text and audio encryption applicable algorithms are available, there is still a lack of appropriate video encryption algorithms. In particular, video encryption algorithms are strongly required that fulfill real-time requirements. The most straightforward approach is to encrypt the entire compressed video stream with conventional cryptographic algorithms such as AES [1]. This is called a *naive algorithm* approach [2]. This approach is simple to implement and easy to integrate into existing multimedia systems, since it is independent of certain video compression algorithms.

Nowadays, advanced computers are fast enough to encrypt a single channel MPEG2 video stream with a bit rate between 4 and 9 Mbps in real-time using the naive algorithm approach [3]. However, this evolution of the computer power does not completely eliminate the need to develop faster encryption algorithms for video data. Many multimedia applications such as video on demand and multiparty P2P video conferences always require specific algorithms for the video encryption because they usually support multi-channel video communication. The simultaneous encryption or decryption, respectively, of all streams causes a huge processing burden at the end systems. Appropriate encryption algorithms allow to alleviate these burdens and to enroll more users to the service.

Since mid 90's many research efforts have been devoted to designing specific video encryption algorithms. Several algorithms were proposed. These algorithms, however, are characterized by a considerable unbalance between security and efficiency. Some of them are efficient to fulfill the real-time requirements but with a limited security level, whilst others are vice versa strong enough to meet the security demands but with a limited encryption efficiency. Moreover, most of these algorithms are related to a certain video compression algorithm and implemented together in software. This makes them less practicable, because today video compression algorithms are standardized and mostly implemented in hardware.

In this paper we propose the video encryption algorithm *Puzzle* which is not only efficient but also sufficiently secure. *Puzzle* can be easily integrated into existing video systems regardless of their implementation (i.e. software or hardware). The paper is organized as follows. After addressing related work in Section 2 we describe the principle of the *Puzzle* algorithm in Section 3. Next in Section 4, we evaluate its performance and compare it with the standard cipher AES. In Section 5 we give a security analysis of *Puzzle*. Some final remarks conclude the paper.

2 Related Work

Existing video encryption methods have been comprehensively surveyed in [4], [5], where they are called *selective encryption* algorithms. This underlines the essence of these methods. They only partially encrypt relevant video information to reduce the computational complexity by exploiting compression and perceptual characteristics. The relationship between selective encryption algorithms and video compression algorithms is a key factor to decide whether an encryption algorithm can easily be integrated into a multimedia system. In this paper we therefore further classify the selective encryption algorithms into two categories according to their association with video compression algorithms: joint compression and encryption algorithms and compression- independent encryption algorithms.

The main idea of the *joint compression and encryption algorithms* is that encryption is applied to a certain step of the compression algorithm so that the output is significantly different from a video stream using a standard compression algorithm. The receivers cannot re-establish the original video without having the encryption key. Figure 1 illustrates the paradigm.

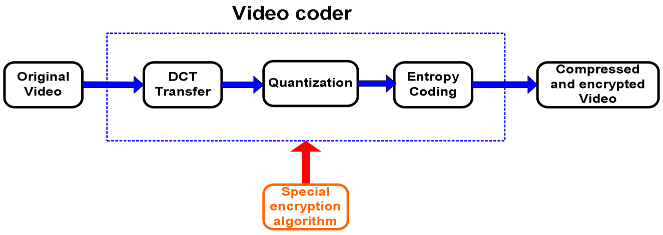


Fig. 1. Principle of *joint compression and encryption* algorithms

The approaches [6], [7], [8], [9] are well known examples of such kind of algorithms. The weakness of the joint compression and encryption techniques is that they cannot be integrated into multimedia systems whose video codecs are implemented in hardware. Certainly, these approaches can be combined with multimedia systems implemented in software, but they completely destroy the modular design of the original codec. Appropriate modifications of the standard video codecs must be made to accommodate these schemes. Therefore, *joint compression and encryption* algorithms preclude the use of standard video codecs in multimedia systems.

The basic idea of *compression-independent encryption algorithms* (see Fig.2) is that compression and encryption are carried out separately. Only parts of the compressed video streams are encrypted with conventional algorithms taking the particular characteristics of the compressed video streams into account.

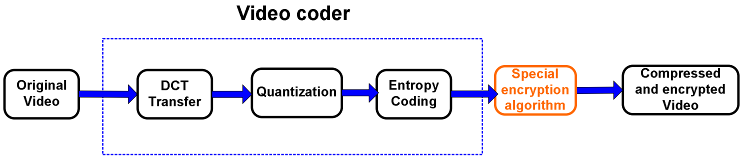


Fig. 2. Principle of *compression- independent encryption* algorithms

So Spanos and Maples [10] and Li [11] exploit the fact that B- and P-frames are predicted from I-frames in interframe compression algorithms. Both proposed encryption approaches in which only the I-frames are encrypted. In theory it should prevent an eavesdropper without encryption key from the reconstruction of the original video. However, Agi and Gong [2] demonstrated that some scene contents are still discernible by directly playing back the selectively encrypted video stream on a standard decoder, since the unencrypted I-macro blocks in the B- and P-frames can be fully decoded without any information from the I-frames. Moreover, this approach did not achieve a significant computational reduction with respect to the total encryption, because the I-frames make about 30~ 60 per cent of an MPEG video [2]. Qiao and Nahrstedt [12] introduced the *video encryption algorithm* VEA in which half of the bit stream is encrypted with a standard encryption algorithm. This half stream is exclusive-ORed with the other half stream. The statistical analysis shows that MPEG video streams are almost uniformly distributed. VEA takes advantage of this special statistical behaviour of MPEG video streams to achieve the sufficient security level. However, the algorithm reduces the encryption load only by 47 per cent, since a half bit stream has to be encrypted with conventional algorithms.

The inflexibility and confinement to deploying joint compression and encryption algorithms in current multimedia systems make them less practicable. Compression-independent encryption algorithms, in contrast, do not suffer from this weakness. They can be easily integrated into existing multimedia systems. Although several such kinds of algorithms are available, they do not achieve a noticeable encryption speed improvement compared to naive algorithms (only about a double speed-up). Moreover, some of them are not resistant to the simple perceptual attack (playing back an encrypted video stream on a standard video player).

In the sequel, we present an efficient and sufficiently secure video encryption algorithm. The outstanding benefit of this scheme is the drastic reduction of encryption overhead for the high resolution video. The algorithm we present here has been improved compared to the first sketch in [13] which appeared not strong enough to resist against sophisticated differential attacks [15]. The new version performs the encryption in reverse order to remove the suspected vulnerability. This reordering in part changed the encryption steps.

3 Principle of the *Puzzle* Algorithm

In this section we first give an overview on the basic idea of the *Puzzle* algorithm. After that the steps of the algorithm are described in detail.

3.1 Principle

The *Puzzle* algorithm is inspired by the children game puzzle which splits an entire picture into many small pieces and places them in disorder so that children cannot recognize the entire picture. When children play the game, they have to spend much time to put these pieces together to re-establish the original picture (see Figure 3).

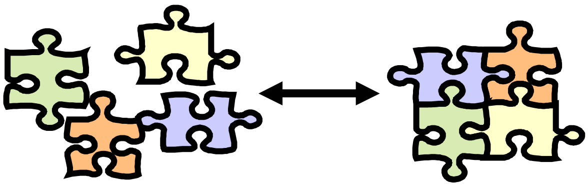


Fig. 3. Puzzle game

Children usually reconstruct the original picture using or comparing known fragments of the picture and referring them to the accompanied original picture. We cannot therefore straightforwardly apply the game to encrypt a video frame. If we, however, modify the rules of the game in the following way, it will be nearly impossible for children to recover the original picture. The children should be only allowed to view the reverse side of the pieces, so that they have to re-establish the picture without any hints to the original picture. It is manifested that *n!* trials are required to re-establish the original, where *n* is the number of pieces. Basically, *n* needs not necessarily to be large. Assume that a picture is divided into 64 pieces, then the number of possible permutations is 64! = 1.27x10^{89}. It is unlikely that children reconstruct the original picture when having so many permutations. With this rule in mind we designed our *Puzzle* algorithm.

3.2 Encryption steps

Puzzle consists of two steps: (1) *Puzzling* the compressed video data of each frame and (2) *Obscuring* the puzzled video data. In step (1) the video data are partitioned into many blocks which are randomly shuffled afterwards. Step (2) corresponds to the turning over the pieces to the reverse side.

3.2.1 Puzzling

A compressed video frame is puzzled by *partitioning* the frame into n blocks of same length b and *disordering* these blocks according to a random permutation list.

3.2.1.1 Partitioning

Given a L bytes long frame (excluding the frame header) of compressed video data $\boldsymbol{V}$ ($v_1v_2...v_L$). The partitioning of the compressed video data $\boldsymbol{V}$ of length L into n blocks of the same length b is a typical factoring problem, i.e. $L = n \times b$. This problem is easy to solve if one of two variables (n,b) is assumed as constant. Unfortunately, we cannot solve this problem this way. If we fix the value of b, the value of n may become very large in some frames or very small in other ones, since the length L varies for each frame. On the other hand, a too large value of n causes a larger computation overhead when exchanging the blocks. If the value of n is too small the scheme can be easily be broken. To solve the problem we put some constraints on the variables n,b. The length of a block b should be $b=2^m$, where m is an integer. The value of n is only allowed to vary in the range from mb to $2mb$, whereby mb is a predefined constant number. It indicates that the compressed video data $\boldsymbol{V}$ should be at least split into mb blocks.

Using these constraints, the value of m can be uniquely determined by the following formula:

$$mb \le L / 2^m < 2mb \ . \tag{1}$$

The length of a block is given through $b=2^m$. The actual block number n can be calculated by the following formula:

$$n = \begin{cases} pn & \text{if } pn \text{ is even} \\ pn-1 & \text{if } pn \text{ is odd} \end{cases} \tag{2}$$

Where pn is the quotient of L/b. Formula (2) makes the value of n always an even number. This operation is necessary to disorder the blocks in the next step. With formula (1) and (2), the product of n and b might be unequal to the video frame length L when pn is odd or the remainder of L/b is unequal to zero. The difference between both is determined using the following formula:

$$d = L - n \times b \tag{3}$$

Formula (3) implies that the d bytes video data at the beginning of the video frame will be excluded from the disordering procedure.

3.2.1.2 Disordering

The basic idea for the disordering of the blocks is that the n blocks of compressed video data $\boldsymbol{V}(v_{d+1}v_{d+2}\ldots v_L)$ are divided into two equal parts: an upper and the lower one. Each consists of $n/2$ blocks. Both parts are interchanged in accordance with a permutation list $\boldsymbol{P}=p_1p_2\ldots p_{n/2}$. This permutation list should be derived from a random sequence to resist an attacker to guess the original position of the blocks. We exploit a stream cipher with an key $\boldsymbol{K}$, such as SEAL [16] or AES-CTR[17], to generate l bytes of random sequence, called key stream ***S*** ($s_1s_2\ldots s_l$), for each video frame. Since the values of the key stream ***S*** vary for each video frame, the permutation lists of different frames are distinct. The algorithm to generate the permutation list is described in the *Appendix*. After deriving the permutation list we can generate the temporary cipher text ***T***$=t_1t_2\ldots t_{L-d}$ from the video data ***V***$=v_{d+1}v_{d+2}\ldots v_L$ by swapping the i^{th} block of the upper part with the p_i^{th} block of the lower part of ***V.*** Figure 4 shows an example of this disordering process. It is assumed that a frame ***V*** is split into 256 blocks $B_1B_2\ldots B_{256}$. The permutation list derived from the key stream ***S*** is ***P***= {256, 213, 216 … 130}.

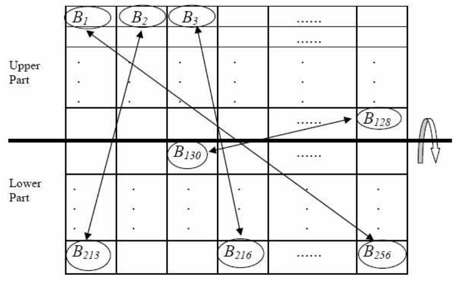


Fig. 4. A *Puzzle* scenario

3.2.2 Obscuring

The temporary cipher text ***T*** is obscured using a light-weight encryption. Its basic idea is to encrypt only a small portion of ***T*** (first *l* bytes) with a stream cipher. Every *l* bytes are grouped into a portion for the rest data of ***T***. Each portion is encrypted by simply exclsive-ORing its preceding. The procedure is as follows. The first *d* bytes of the compressed video data ***V***($v_1v_2...v_d$) that are not involved in the puzzling procedure are exclusive-ORed with *d* bytes of key stream ***A***($a_1\,a_2\ldots a_d$) generated by a stream cipher with the encryption key ***K***. The first *l* ($l<L$) bytes of ***T*** ($t_1t_2...t_l$) are exclusive-ORed with *l* bytes of the key stream ***S*** ($s_1s_2\ldots s_l$) generated in the puzzling step. The sense of the reuse of key stream ***S*** is to make the algorithm more efficient. After that the first *l* bytes of ***T*** are used as key stream and exclusive-ORed with the second *l* bytes. Then the second *l* bytes of ***T*** are exclusive-ORed with the third *l* byte and so on until the end of the frame is reached. As output we receive the *L* bytes long cipher text ***C*** ($c_1c_2\ldots t_L$). Figure 5 shows the principle. Note that the frame header remains unencrypted, because it usually contains standard information.

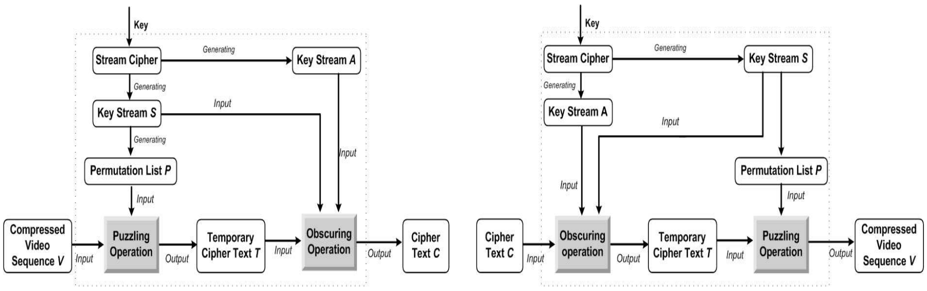
Input text	$v_1v_2...v_d$	$t_1\quad t_2\ \ldots\ t_l$	$t_{l+1}\quad t_{l+2}\ \ldots\ t_{2l}$	$t_{2l+1}\quad t_{2l+2}\ \ldots\ t_{3l}$	$\ldots\ldots t_{L-d}$
$\oplus$					
Key stream	$a_1a_2\ldots a_d$	$s_1\quad s_2\ \ldots\ s_l$	$t_1\quad t_2\ \ldots\ t_l$	$t_{l+1}\quad t_{l+2}\ \ldots\ t_{2l}$	$\ldots\ldots t_{L-d-l}$
Cipher text	$c_1\,c_2\ldots c_d$	$c_{d+1}\,c_{d+2}\ldots c_{d+l}$	$c_{d+1+1}\,c_{d+l+2}\ldots c_{d+2l}$	$c_{d+21+1}\,c_{d+2l+2}\ldots\,c_{d+3l}$	$\ldots\ldots\,c_L$

Note: v_i, s_i, a_i and t_i denote a data byte. The input text contains the temporary cipher text *T* and the first *d* bytes of the compressed video data.

Fig. 5. Obscuring algorithm

3.3 Encoding and decoding procedures

The components of the *Puzzle* encoding procedure described above are summarized in Figure 6a). The original compressed video sequence can be recovered from the cipher text at the receiver's side by executing the encoding operations in reverse order (see Figure 6b)).



a) Encoding

b) Decoding

Fig. 6. Encoding and decoding procedures of *Puzzle*

4 Measurements

We run a series of experiments for different resolution videos to measure the speed of *Puzzle* in comparison with the standard cipher AES on a SUN ULTRA 10 platform. To make a trade-off between security and efficiency, the values of the variable *l* and *mb* in our algorithm are both set to 128. Further we used AES-CTR [17] as stream cipher. The measurement results are depicted in Figure 7.

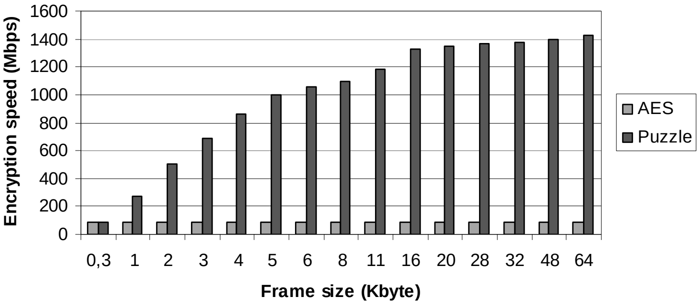


Fig. 7. Comparison of the encryption speed of AES and *Puzzle*

Figure 7 shows that the encryption speed of AES basically does not change for different frame sizes, whereas that of *Puzzle* noticeably increases for larger frames. AES and *Puzzle* only have a nearly equivalent encryption speed at a frame size of 300 byte. With increasing frame sizes the encryption speed of *Puzzle* becomes considerably higher than that of AES. This indicates that *Puzzle* is better suited for high resolution video streams which usually have a large frame size.

5 Security Analysis of the *Puzzle* Algorithm

In this section we evaluate the security properties of the *Puzzle* algorithm. A good crypto system should withstand the following most important attack classes [14]: *ciphertext-only attacks*, *known-plaintext attack*s, and *chosen-plaintext attacks*. Furthermore, the measure to defend against *differential attack* should be taken into account.

Ciphertext-only attacks: Based on the cipher text an adversary has two possibilities for trying to re-establish the original frame encrypted with *Puzzle* (see Figure 6b)). He/she can either attempt to break the puzzling and then the obscuring operation or to crack these steps in the reverse order. The first attack corresponds to the situation when the child first tries to put the disordered pieces to their correct position only looking at their backside and then turns the whole correctly reordered picture to the right side. In *Puzzle* each frame is split in at least 128 blocks in the puzzling operation, i.e. more than 64! = $1.27 \text{x} 10^{89}$ trials to reconstruct a single original frame. This is obviously computationally infeasible to be broken, especially as a 128 bit key length standard block cipher is believed to be computationally secure enough today. The number of possible permutation for such standard block cipher is, however, only 2^{128} = $3.4 \text{x} 10^{38}$.

The second attack resembles the situation when the child first turns the disordered pieces to the right side one by one and then re-establishes the picture using content information. In *Puzzle* the cipher text ***C*** is generated by connecting two puzzled plaintexts fragments using the exclusive OR operation except the first $l+d$ bytes of the obscuring operation. As shown in [12] the computation complexity to obtain a 10 bytes MPEG compressed video sequence by separating two 5 bytes long exclusive-ORed plaintext is equivalent to that of breaking a 64-bit key length block cipher which has 2^{64} combinations. As mentioned in Section 4, we recommend to applying *Puzzle* on video sequences with a frame length larger than 300 bytes. Accordingly the attacker has at least to try all $30 \text{x} 2^{64} \approx 2^{69}$ combinations to obtain the plain texts of the disordered blocks for a single frame.

Known- plaintext attacks: Given a compressed video frame and its corresponding cipher text, as show in Figures 5 and 6b), it is not difficult for an attacker to get the l bytes long key stream ***S*** and the d bytes long key stream ***A***. However, the attacker is unable to decrypt the other video frames using this key stream, since we utilize AES-CTR as the stream cipher to generate the distinct key streams ***S*** and ***A*** with encryption key ***K*** for each frame. It is further impossible to derive the encryption key ***K*** from a known key stream ***S*,** since AES-CTR is a confidentiality mode [17] which is secure against known-plaintext attacks. Therefore our algorithm can withstand these attacks.

Chosen-plaintext attacks: Our scheme is resistant against chosen-plaintext attacks for the same reasons given for known- plaintext attacks.

Differential attacks: The original goal of the differential cryptanalysis is to attempt to reconstruct the encryption key by studying the differences between the plaintext and the respective ciphertext pairs. The differential cryptanalysis can reduce the complexity of attacking a cipher by half. Generally speaking, it is a specific kind of a *chosen-plaintext attack*. As discussed above, such attack is not effective to our scheme. On the other hand, attackers might apply the basic idea of differential cryptanalysis to launch a *specific* ciphertext-only attack by analyzing the ciphertext of our scheme without the knowledge of the respective plaintext for the specific structure of our scheme. The order of encryption procedure in our scheme decides, whether our scheme is strong enough to withstand such a specific ciphertext-only attack. In [13] we first obscured the original frame and then puzzled the obscured one. This encryption order is suspect to be too weak for specific ciphertext-only attacks, because the edges values of the blocks of the original video frame tend to be very close. These close values are inherited to the obscured frame in this encryption order. The attacker might determine which blocks might be neighbors using this information. For that reason, we have changed the encryption order, i.e. first puzzling then obscuring. The edges of neighbor blocks will now have significantly different values so that such an attack is avoided.

6 Final Remarks

In this paper we presented the improved video encryption algorithm *Puzzle* for encrypting video communication in real-time. *Puzzle* is a compression-independent encryption algorithm which can be easily integrated into available multimedia applications. It provides a good trade-off between security demands and encryption efficiency. *Puzzle* achieves a sufficiently fast encryption speed to meet the real-time requirements of most used multimedia applications, especially for high resolution video streams. *Puzzle* withstands most important cryptanalysis attacks. By changing the order of the encryption steps the algorithm has become resistant against differential attacks. We use *Puzzle* as part of the security architecture of our multiparty P2P video conference system BRAVIS [18] to ensuring confidential talks over the Internet.

References

1. NIST: Advanced Encryption Standard. FIPS 197, 2001.
2. I. Agi and L. Gong: An Empirical Study of MPEG Video Transmission. NDSS’96, pp.137-144, 1996.
3. B. G. Haskell, A. Puri, and A. N. Netravali: Digital Video: An Introduction to MPEG-2. Kluwer Academic Publishers, 1996.
4. B. Fuhrt and D. Kirovski: Eds. Multimedia Security Handbook, CRC Press, 2004.
5. X. Liu and A. M. Eskicioglu: Selective Encryption of Multimedia Content in Distribution Networks: Challenges and New Directions. IASTED International Conference on Communications, Internet and Information Technology (CIIT), 2003.
6. L. Tang: Methods for Encrypting and Decrypting MPEG Video Data Efficiently. In Proceedings ACM International Conference on Multimedia, pp.219-229, 1996.
7. C. Shi, S. Y. Wang, and B. Bhargava: MPEG Video Encryption in Real-Time Using Secret Key Cryptography. PDATA’99, 1999.

8. W. Zeng and S. Lei: Efficient Frequency Domain Selective Scrambling of Digital Video. IEEE Transactions on Multimedia, 2002.
9. C.–P. Wu and C.-C. J. Kuo: Efficient Multimedia Encryption via Entropy Codec Design. SPIE Int. Symposium on Electronic Imaging 2001, Vol. 4314, 2001.
10. G. A. Spanos and T. B. Maples: Performance Study of a Selective Encryption Scheme for the Security of Networked Real-time Video. ICCCN, pp. 2-10, 1995.
11. Y. Li, Z. Chen, S. –M. Tan, and R. H. Campbell: Security Enhanced MPEG Player. IEEE 1st International Workshop on Multimedia Software, 1996.
12. L.Qiao and K. Nahrstedt: Comparison of MPEG Encryption Algorithms. Computer and Graphics 22 (1998) 4, pp.437-448.
13. F. Liu and H. Koenig: A Novel Encryption Algorithm for High Resolution Video. In Proceeding of ACM NOSSDAV'05, pp.69-74, 2005.
14. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone: Handbook of Applied Cryptography. CRC Press Series on Discrete Mathematics and its Applications. CRC Press, 1997.
15. E.Biham and A.Shamir: Differential Cryptanalysis of the Full 16-round DES. Advances in Cryptology-CRYPTO'92. Springer. pp.487-496, 1992.
16. P. Rogaway and D. Coppersmith: A software-optimized encryption algorithm. Proceedings of the 1st International Workshop on Fast Software Encryption, Springer, pp.56-63, 1993.
17. M. Dworkin: Recommendation for Block Cipher Modes of Operation, Methods, and Techniques. NIST Special Publication 800-38A, Dec.2001.
18. The BRAVIS video conference system. http://www.bravis.tu-cottbus.de.

Appendix: Generation of the Permutation List

```
Algorithm Permutation list generation
    Input: A key stream $S=s_1s_2\ldots s_l$, $n$ --  number of blocks in the compressed video data $V$
    Output: A permutation list $P=p_1p_2..p_{n/2}$.

    begin
      Let $A$ be an auxiliary sequence $A=a_1a_2\ldots a_{n/2}$, its value of an element is
          $a_i = i + n/2,\ 1 \le i \le n/2;$
      Define $D$ as another auxiliary sequence which is used to temporarily save the value
      selected from the key stream $S$;
        for $i=1$ to $l$ do
            /* Make the value of every element in S ranging from 1+n/2 to n. */
          if (($s_i$ mod n)≤n/2)  $s_i$= ($s_i$ mod n)+n/2;
              else $s_i$= $s_i$ mod n;
          end if
          Put $s_i$ in the auxiliary sequence $D$ without repetition;
          Extract $s_i$ from the sequence $A$ and build sequence {$A$-$D$};
        end for;
         /* Get the permutation list P, || denotes the append operation. */
        $P$=$D$||{$A$-$D$};
     end
```