On Improving the Lightweight Video Encryption Algorithms for Real-time Video Transmission

Simon Fong, Yang Hang
Faculty of Science and Technology
University of Macau
Macau SAR
ccfong@umac.mo

Abstract—**Fast lightweight encryption algorithms must be developed to satisfy the level of security and the real time constraints. However the recently proposed lightweight MPEG video encryption algorithms that have been proposed in the past suffer from certain drawbacks. While some of them require hardware support, the others are weak or reduce the MPEG compression ratio. The lightweight encryption algorithm called VEA is fast, satisfies the real time constraint and does not reduce the MPEG compression ratio. However, it relies on the key generator to generate a good encryption key and it cannot withstand the known-plaintext attack. In this paper, improvements to the VEA are proposed namely the Rotation algorithm, the XOR algorithm and one that combines VEA with IDEA (I-VEA). They are able to secure MPEG video with minimal computational overhead, which do not reduce the MPEG compression ratio, do not rely upon the key generator to generate an effective key and can better resist the known-plaintext attack. The Rotation algorithm is the fastest of the three, but is relatively weak. I-VEA is the most secure but it adds the maximum computational overhead. The XOR algorithm is a good compromise between the two**

Keywords Multimedia data security, MPEG video encryption, IDEA

I. INTRODUCTION

Multimedia data security issues are vital for video-on-demand, video broadcast, multimedia e-mail and video conferencing. Encrypting algorithms that are good for textual data might not be suitable for multimedia data, primarily due to the relatively huge size of the multimedia data and the real time constraints [1]. Dedicated hardware set top boxes can be used; however they add costs to both the transmitter and the receiver. An alternative is to apply software encryption and decryption. The challenge is to design lightweight encryption algorithms that do not aggravate the heavy processing involved with multimedia data and at the same time provides satisfactory level of security. Ideally, these lightweight algorithms for MPEG encryption should be fast enough to satisfy real time constraints, should be able to resist the well-known cryptographic attacks and should not reduce the MPEG compression ratio.

The lightweight MPEG video encryption algorithms proposed in the past suffer from certain drawbacks [2]. While some of them require hardware support, the others are weak or reduce the MPEG compression ratio. An MPEG encryption algorithm called VEA [3] that was proposed recently could satisfy most of the real time encryption requirements. However, it relies on the key generator to generate a good encryption key. Besides, it cannot withstand the known plain-text attack [4]. In this paper, improvements to the VEA are proposed namely the Rotation algorithm, the XOR algorithm and an algorithm that combines VEA with IDEA called I-VEA. The improved algorithms aim to solve the problems inherent with the encryption key and the known plain-text attacks.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 describes the proposed algorithms. Section 4 presents the experiment results. Finally a conclusion is given in Section 5.

II. BACKGROUND AND RELATED WORK

Multimedia data has high information rate and hence low information value according to Shannon's theorem. Tang [5] proposed the ideal properties of a video encryption algorithm as follow.

- The encryption algorithm should have zero-one semantics.
- The computational overhead should minimal.
- The encryption procedure should not decrease the compression ratio.
- The algorithm should be robust against common image processing techniques.
- The algorithm should not affect the quality of the video.

In addition, the algorithm should be able to withstand the common attacks on cryptostreams (Ciphertext-only Attack, Known-plaintext Attack, Chosen-plaintext Attack, Man-in-the-Middle Attack and Timing Attack) [6]. The Man-in-the-Middle Attack and the Timing Attack are of importance in only public key encryption systems. It was shown in [6] that if encryption was done before the compression stage, and furthermore compression was achievable, then the encryption algorithm

would have security flaws. Encryption thus must be done after the MPEG-1 compression stage.

The simplest way to secure MPEG video would be to encrypt the entire bit-stream by using e.g. DES [7]. This encryption method guarantees the strongest possible security at the cost of computation time and it does not take advantage of the MPEG-1 structure. The AEGIS encryption system [8] tries to take advantage MPEG-1 structure. The AEGIS system acts as a stream filter between the MPEG encoder/decoder and the transmission mechanism. It employs the DES encryption algorithm for the encryption process in the cipher-block-chaining mode (CBC) which is very secure and is least affected by bit errors. The DES in AEGIS to be used with dedicated hardware, that is slow if it were to be implemented in software. AEGIS encryption also reduces the MPEG compression ratio, which is undesirable. It was shown in [9] that even though a single P-frame or a B-frame on its own is meaningless without the corresponding I-frame, a series of P-frames and B-frames carry much information if their base frames are correlated.

Tang [5] proposed a zigzag permutation algorithm. It uses a random permutation list to order the 8 x 8 block into a 1 x 64 vector. Tang's zigzag permutation algorithm splits the 8 bit binary number (DC value) into two 4 bit binary numbers which are both in the range of [0, 15] to disguise the DC component and then performs the random mapping. However, Tang's algorithm (even with the 2 key algorithm) cannot resist the known-plaintext attack [6]. Also, Tang's algorithm does not use the zigzag ordering and hence the compression ratio achievable by MPEG encoding reduces. Qiao and Nahrstedt proposed a video encryption algorithm in which all the I-frames are encrypted with a different key [6]. The key of length 1/8 of the frame is used to do the permutation in a byte by byte fashion on MPEG compressed video streams. The secret key used for encrypting the I-frame is encrypted with DES and sent along with the I-frame. This method does withstand the known-plaintext attack as different keys are used. However, the pure permutation algorithm increases the total size of the transmitted stream in general by approximately 12.5%, that results decrease in the compression ratio which is undesirable. Furthermore, only the I-frames are encrypted. Although I-frames are the most important, a sequence of P and B frames can still convey much information.

Shi and Bhargava developed a lightweight MPEG video algorithm that does not suffer from some of the drawbacks of the previous algorithms [3]. Their lightweight algorithm uses a key to randomly change the sign of the DCT coefficients during the encryption process. The reverse is done during the decryption procedure. The encryption/decryption achieved by VEA is symmetric because the encryption and the decryption procedures use the same key. Assume that, S is the plaintext, C is the ciphertext, k is the encryption key, E is the encryption algorithm, D is the decryption algorithm and m is the decryption key. Then $C = E_k(S)$ and $S = D_m(C)$. Hence $S = D_m(E_k(S))$. But $S=E_k^{-1}(E_k(S))$. Comparing the two we obtain, $D_m = E_k^{-1}$. A possible solution to this is m = k, that is the encryption and the decryption keys are the same and $D = E^{-1}$, that is the encryption algorithm is the decryption algorithm are mathematical inverses of each other. There are several advantages of this algorithm [3], like changing of keys, minimal computational overhead. VEA however, is weak for known-plaintext attack. Also, if there is any long consecutive occurrence of the same bit (e.g. a long string of 0's or a long string of 1's) in the VEA key, the security of the algorithm reduces. A long run of zeros in the VEA key may leave the many of the DCT coefficients unchanged, and a long run of ones in the VEA key makes it easy for an adversary to attack.

III. PROPOSED SOLUTIONS

A. Long strings of ones and zeros

The current VEA does not perform well when the key has a long string of 0's or 1's. It also cannot resist the known-plaintext attack. A good encryption system should not rely on the secrecy of the algorithm. The strength of the algorithm should only depend on the strength of the key. A modification of the VEA is proposed which can overcome the problem of long string of 0's and 1's as well as resist the known-plaintext attack. The algorithm solely depends on the secrecy of the key and the algorithm is assumed to be common knowledge. A long string of zeros or ones in the VEA causes a problem, as $x \oplus 0 = x$, and $x \oplus 1 = \sim x$. With a long string of zeros in the VEA key, the DCT coefficients will remain unchanged and not much encryption would be achieved. The effect of using all 1's as the VEA key is to invert all bits. A long string of 1's in the VEA key is as bad as having a long string of 0's and should be avoided. The obvious solution is to make sure that keys do not have long strings of zeros or ones [3]. However this reduces the key space drastically and also allows the adversary to launch a probability based attack to obtain the key. Let the length of the key be 'n', and the maximum number consecutive 0's or 1's allowed in the key be 'k'. The original key space is given by the permutation of the 'n' bits (2^n). The key space however reduces due to the above restriction. The equivalent circular permutation problem can be stated as: in a circle, how many ways, can 'n' 1's and 0's be arranged such that no more than k zeros or k ones occur one after the other.

Table 1 below shows the results for k=n/2. It can be seen that the security of the algorithm is considerably lowered. An adversary could launch a powerful attack based on bit sequence probability, reducing the time to crack the key.

TABLE 1. Number of 'n' bit sequences with k consecutive 0's or 1's

(n, k)	(8,4)	(10,5)	(12,6)	(14,7)	(16,8)
Number of possibilities	122	312	758	1780	4082

A better idea is to allow the key generator to generate all 2n keys. After key generation, the key can be altered to remove long sequence of zeros, using some encoding technique. Run length limited (RLL) could be used but that has the disadvantage because RLL is typically used for limiting the run length of 0's. A similar table would have to be developed for 1's. Ideally RLL (0,2) could be used with the maximum number of 2 consecutive zeros and a similar table for ones. However RLL implementation is time consuming as it involves parsing of streams and comparing with the standard tables to find a match.

A technique we used in our project is to employ Manchester coding [10] that is often used in communication to add clocking information to the data to be transmitted. Informally, the Manchester code can be described by representing a 0 by 01 and a 1 by 10. Using Manchester code it is thus possible to remove long strings of 1's as well as 0's. The overhead due to the extra computation is minimal although some substitution mechanisms need to be implemented at the transmitter and receiver.

B. Known Plain-text Attack

There is a solution to the problem of known plain-text attack provided in [6], but at the cost of extra information (keys) that needs to be embedded into the MPEG stream. Also encrypting just the I-frames is not very reliable as the base frames of a sequence of P and B frames are correlated. To solve the problem of the known-plaintext attack, it may be sufficient to encode the MPEG frames using different keys [6]. The problem then remains as to how these keys can be generated, and how they can be sent to the receiver. We have proposed three algorithms to attempt overcoming the known plain-text attack.

1) *Rotation Key Algorithm*: In this algorithm, a small second key is used. The second key is a string of characters, which are used to rotate the bits of the first key depending upon the ASCII value of the characters in the second key. The rotation of the first key (a sequence of bits) by a random amount would generate a random sequence of bits. This new sequence is going to be used as the new key. This process of generating new keys could be continued for all the characters in the second key, returning to the first character and starting all over again when all the characters are exhausted. Thus the video could be encrypted relatively more securely. The pseudo-code for the above procedure is described in Figure 1 and Figure 2.

```
/* Pseudo code for key rotation method */
  #define KEY_LENGTH 56
  begin
  decryp_key: bit[KEY_LENGTH];
  rotation_key: character[20];
  temp_store: decryp_key[20];
  key_store: decryp_key[20];
  temp: character;

  read (decryp_key);
  read (rotation_key);
    loop
      temp: = readchar (rotation_key);
      temp_store[ ]: =
      rotate_r(decryp_key[],ascii(temp));
      key_store[ ]: = manchester_enc(temp_store[ ]);
    end loop;
  end;
```

Figure 1. Pseudo-code for the rotation key algorithm

```
If (rotation value < n)
     Rotate by rotation value;
If (rotation value > n) {
     Rotate by rotation value;
     Invert the bit at the location given by
⌊rotation value / n⌋;
}
```

Figure 2. Pseudo-code for rotate_r (argument).

Let n = length of encryption key and k = length of the rotation key. An adversary can launch an exhaustive attack in the following way. Keeping the first character of the rotation key fixed to an arbitrary value φ, it is possible to obtain first frame of the video sequence in 2^{n-1} attempts in the worst case. The decryption key, for which the first frame can be seen, is the original key rotated by φ. Each of the next frames can be seen clearly if the correct rotation key is used. For each frame, 256 attempts are needed in the worst case. But the decryption key length is n. Therefore the worst case is min (n, 256) for each frame. The overall worst case complexity reduces to 2^{n-1} + (min (n, 256)).k. However, using the known plain-text attack, the adversary can easily obtain the rotated version of the original key. With the knowledge of the rotated version of the key, the adversary can completely decrypt the video using brute force in min(n, 256).k tries. Another problem associated with the basic rotation of the key is that for n < 256, rotation by a value of p and a value of (n + p) has the same effect, due to the modular nature of rotation. This in turn would mean that it is possible to obtain the decrypted video by using wrong keys. To overcome this problem, the following was incorporated into the algorithm. The LSB is assumed to be bit 0 (Figure 2). Now, the rotation of the decryption key by p is different from the rotation of the decryption key by (n + p). However, there is a possibility that two wrong keys may result in video decryption. For using two wrong keys and being able to decrypt the video, one must get the decryption key wrong by only 1 bit. The probability of this happening is $1/2^n$. However any rotated version of this key is equally useful. This gives 'n' keys out of 2^n. Also the bit should be the one that gets inverted due to wrong rotation key. For this also the probability is 1 in 256. The number of possible rounds or the number of bits that may be inverted as a result of the algorithm is $\lfloor 256/n \rfloor$. Thus the probability that the video can be decrypted using two wrong keys is $\{n/(2^n.256)\} \cdot \lfloor 256/n \rfloor$.

For n = 56, the probability is sufficiently low for all practical purposes. The method still remains weak for the known-plaintext attack. The number of rounds possible is $\lfloor 256/n \rfloor$. As the algorithm is a common knowledge, it will be known which are the bits inverted by the algorithm. If the known plain-text attack is used, a rotated version of the original key with a bit possibly inverted is obtained. Using the obtained information, the keys for the successive frames can be obtained by a simple rotation of the obtained key. The number of possible rotations is min (n, 256). For each of these rotations the bits from 1 to $\lfloor 256/n \rfloor$ should be inverted and the key should be tested. Therefore, within (min (n, 256).$\lfloor 256/n \rfloor$.k) tries, the entire video can be hacked. It was easy to hack if rotation of the decryption key is used. This was because of the fact that the original key can be obtained from the key obtained by the known plain-text in O(n) time.

2) *XOR Key Algorithm*: In this algorithm, a small second key is used. The second key is a string of characters, which are used to rotate the bits of the first key depending upon the ASCII value of the characters in the second key. The rotation of the first key (a sequence of bits) by a random amount would generate a random sequence of bits. This new sequence could be used.

The XOR algorithm uses the XOR operation to change the decryption key. The XOR operation has the masking or complementing property depending on whether 0 or 1 is used to do the XOR operation. 0 is used for masking, and 1 is used for complementing. The XOR operation has the property that if $z = x \oplus y$, from z itself it is very difficult to obtain x and y. This property of the XOR operation is used to change the decryption key with a second key. The possibility of getting a correct desired result from 2 wrong values is also remote. Consider two n-bit numbers X and Y. The XOR operation, $X \oplus Y$ gives Z. The number of different possibilities for X and Y is $2^n.2^n = 2^{2n}$. Out of these, there are (2^n-1) possibilities when two wrong values of X and Y give us the desired value of Z. Thus the probability of the two wrong values giving us the desired value is less than $1/2^n$. For n = 56, the probability is very low. If the known plain-text attack is used, a particular frame can be grabbed and the corresponding encryption key can be obtained. However this encryption key is the XOR of two other keys. With only the resultant key, it is extremely difficult to guess the two other keys. Without those keys, the rest of the video cannot be decrypted. Thus the known plain-text attack can be resisted using this algorithm. The pseudo-code for the algorithm is given in Figure 3.

```
/* Pseudo code for XOR key method */
#define KEY_LENGTH 56
begin
decryp_key: bit[KEY_LENGTH];
xor_key: character[20];
temp_store: decryp_key[20];
key_store: decryp_key[20];
temp: character;

read (decryp_key);
read (xor_key);
loop
  temp: = readchar (xor_key);
  temp_store[ ]: =
  xor (decryp_key[ ],ascii(temp));
  key_store[ ]: = manchester_enc(temp_store);
end loop;
end;
```

Figure 3. Pseudo-code for the XOR key algorithm.

The XOR operation provides reasonable security as far as the possibilities of obtaining the keys are concerned. The brute force search for the key upon a plaintext attack also does not appear very feasible.

3) *IDEA Key Algorithm*: The security of the algorithm can be enhanced by using the International Data Encryption Algorithm (IDEA) [11]. IDEA is a block cipher, with blocks of 64 bits used as plaintext and ciphertext and with a 128-bit secret key. IDEA has the advantage over DES that it is also well suited for software implementations unlike DES. IDEA is also able to resist differential cryptanalysis. The block diagram for IDEA is shown in Figure 4. In our project, IDEA is incorporated into the encryption/decryption algorithm to resist the plaintext attack. A predetermined number of keys are to be generated which would be a constant. The necessary and sufficient condition to overcome the plaintext attack is that: Each of the frames should be encrypted with different keys. The keys should have a minimum correlation, so that one key cannot be generated from another. Rule 2 was violated for the rotating key case, which made the algorithm weak. IDEA could be used to generate the set of keys. The cipher-text generated by any good encryption algorithm should have minimum redundancy and maximum randomness. The cipher-text generated by IDEA could be used as keys and the ciphertext could be fed back as the plaintext for the next key to be generated. If the number of keys to be generated is predetermined or determined based on the number of GOP's in the MPEG-1 stream, then this could be used both at the encoder and the decoder in a symmetric fashion. The process is illustrated in Figure 5. However, the operation involved in IDEA is computationally more expensive than the trivial XOR operation. Hence, a computation overhead is added. Also because of this overhead, the key set must be generated one time, before the MPEG encryption and encryption process can start. The computational complexity of breaking this algorithm is the same as the complexity of breaking the IDEA encryption, which is known to be very difficult.

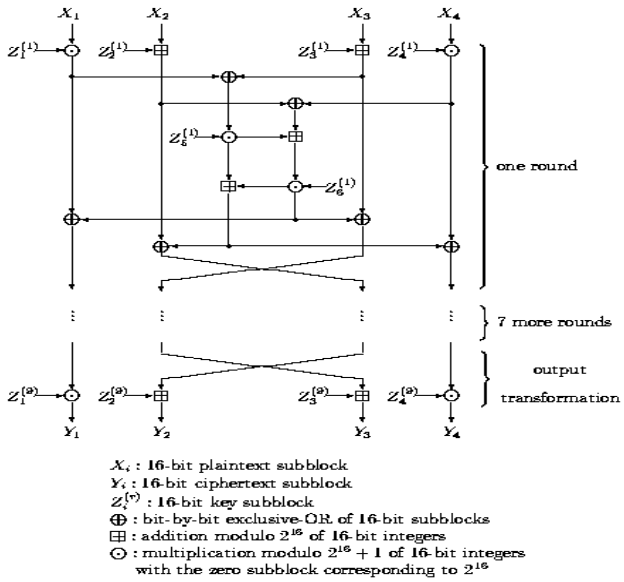


Figure 4. IDEA Computational process.

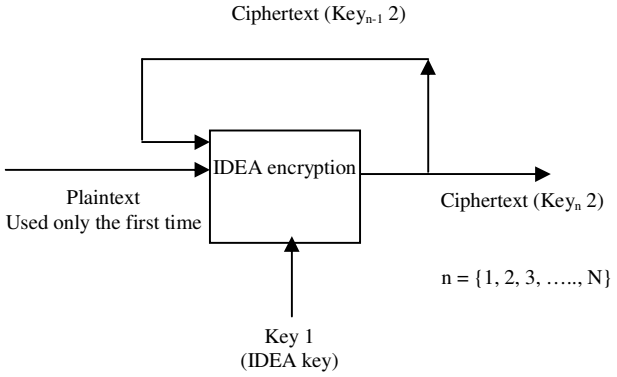


Figure 5. IDEA model to resist plaintext attack

IV. EXPERIMENTS AND RESULTS

In our experiments, the algorithms were incorporated into the MPEG-1 Berkeley encoder (mpeg_encode) [12] and decoder (mpeg_play) [13]. As mentioned in the previous section, a long string of 0's or 1's in the VEA key often allows some of the video frames to be comprehensible. In Figure 6, a frame from movie1.mpg was grabbed. The differences are clearly visible.

The movies encrypted with the Manchester encoded key gives better results as it can be seen from the following figure. However the Manchester encoding method needs slightly more time, which is still quite acceptable. The times for decryption with and without Manchester encoding are given in Table 2. The MPEG-1 file has 240 frames. The key used was a 16-bit key, 1011010100111010. The time recorded was in seconds. It can be seen that the percentage increase in time is quite negligible compared to the performance improvement that can be achieved by Manchester encoding of the key.

Three methods were developed to overcome the plaintext attack, namely the rotation key method, the XOR key method and the IDEA method. Though the rotation key method is weak, yet it is stronger than the simple video encryption algorithm. A comparison of the decryption times for the different algorithms is given in Table 3 and Table 4. A MPEG-1 movie with 240 frames was used. The decryption key of 1011010100111010 (16-bit key) was used. The second key, used as the rotation key in the first case, used as the XOR key in the second and used as the preliminary plaintext in IDEA was just an arbitrary 8-character text string: PASSWORD.

It is well known that there is a fundamental tradeoff between speed and security. Taking into account the percentage increase in time and also the amount of security provided by the algorithms, the XOR algorithm seems to be an appropriate choice. It is a good compromise between speed and security. The Manchester encoded key does perform better than the normal key. The overhead is also minimal. Thus we can combine the Manchester encoding of the key1 and use a second key, key2, for the XOR operation on key1 to prevent the known-plaintext attack.

Consider an 8 x 8 block, and name it X. The corresponding 8 x 8 block Y, is obtained after the Discrete Fourier Transform of X. Let Y be represented by the coordinates (k, l) and X by the coordinates (i, j). If the XOR operation is used to change key_1, for a given frame the key will consist of a random sequence of bits. The sequence of bits will depend on both key_1 and key_2. If the MPEG quantization, zigzag ordering, and Huffman coding are not considered for the sake of simplicity, it can be said that the sign of the DCT coefficients are randomly changed according to the key for the given frame. For each coefficient in the 8 x 8 block, $z_{kl} = e_{kl}. y_{kl}$, where k, l = 0, 1, 2, 7, where, $e_{kl} = 1$ if the sign of y_{kl} is not changed and $e_{kl} = -1$ if the sign of y_{kl} is changed. The coefficients z_{kl} are seen at the decoder and it is z_{kl} on which the IDCT is performed. The recovered pixel values at the decoder can be written as: $\hat{x}_{ij} = x'_{ij} + n_{ij}$, where $x' \approx x$ but not exactly equal due to the lossy nature of MPEG encoding. n_{ij} is the noise added to the signal [3]. It can be seen from the above that, even if only the signs of a few coefficients are changed, the IDCT operation will change most of the image pixel values. The security of a given frame depends on the key used to encrypt the frame. If a 56-bit key is used for encryption, then the number of possible combinations of 0's and 1's are 2^{56}. This means that a hacker using the brute force approach must try 2^{56} options in the worst case before he cracks the key. However, it can be seen that if the hacker gets all the bits of the key wrong, then the image is comprehensible as shown in Figure 18. In such a case, inverting all the bits of the key will give the hacker the right key. Thus the security of algorithm in general drops to 2^{55} or 2^{n-1} in the general case, for an n-bit key. If an 8-character key is used as the second key (key_2), then the number of possible variations for the second key are $(256)^8$. The overall complexity of decrypting the entire video is $2^{55}.2^{64} = 2^{119}$.

Assuming that the adversary employs the known plain-text attack, and gets the key used to encrypt the first frame. To demonstrate that the other frames are still secure, the correct key_1 can be used with an arbitrary 8-character key_2, which has its first character correct. Actually the hacker using the known-plaintext attack can only obtain the key used to encode a frame, and not entire key_1. By granting the knowledge of key_1, a worse case is being considered here.

The results of the XOR algorithm are shown below and compared with the VEA which cannot resist the plaintext attack. The sequence used was *football.mpg*. The sequence consisted of 30 frames. The GOP size was set to 10. The number of GOP's was thus 3. The GOP format is IBBPBBPBBP, and the I-frames in an MPEG sequence are most important. Thus when the first frame has been hacked, the rest of the GOP is moderately comprehensible. For VEA the key is 1011010111001001. For the XOR algorithm, key_1 = 1011 0101 1100 1001 and key_2=PASSWORD. This can be seen in Figures 7 and 8 below.

CONCLUSION

In this paper, we presented three methods to improve the lightweight video encryption algorithms for real-time transmission. In particular, the problem of the known-plaintext attack has been addressed in each case. The algorithm (XOR key algorithm) adds minimal overhead to the MPEG-1 codec and at the same time does not suffer from the drawbacks of some of the existing algorithms. While the Rotation key algorithm is relatively weak, it is faster and can be used in applications where very high security is not important. IDEA with VEA is very strong but at the same time adds overhead in terms of computational time. The experimental results on various MPEG video streams verify the improvements over the existing VEA algorithms.

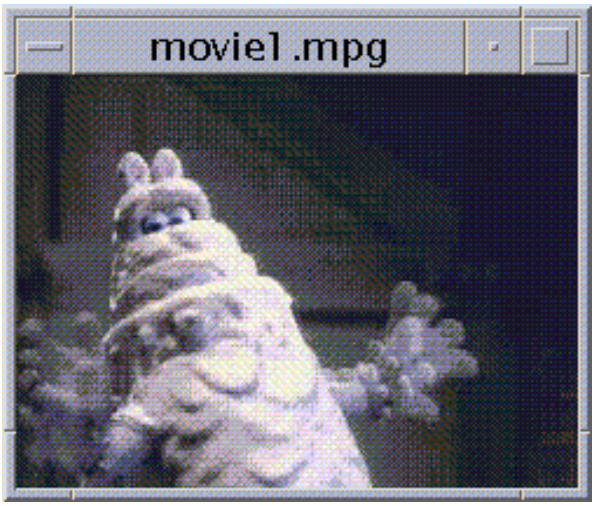


Figure 6a. Original video

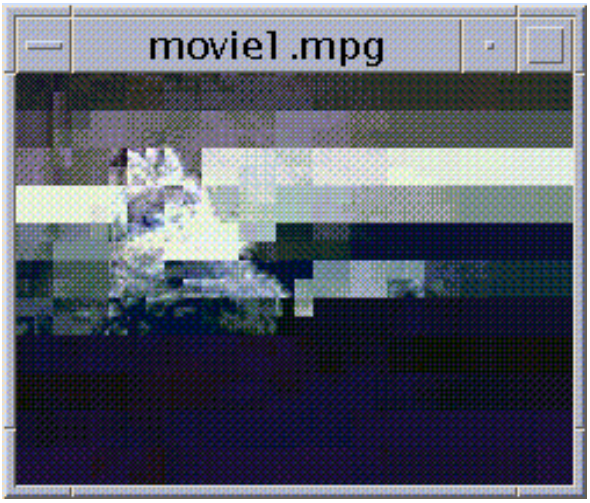


Figure 6b. Video encrypted with VEA

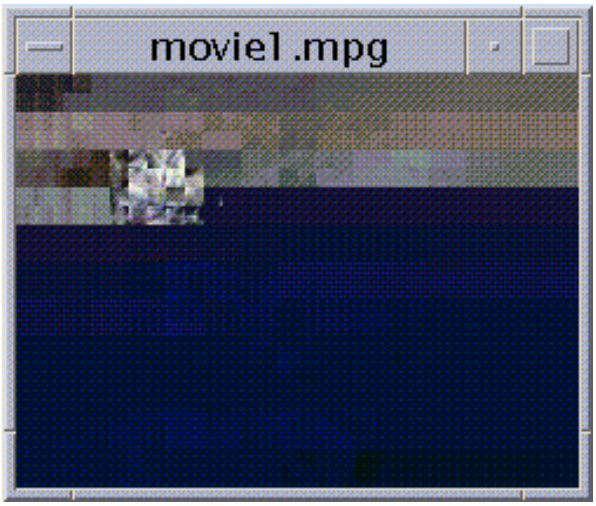


Figure 6c. Video encrypted with modified VEA

REFERENCES

[1] Xun Yi; Chik How Tan; Chee Kheong Slew; Rahman Syed, M, "Fast encryption for multimedia", IEEE Transactions on Consumer Electronics, Volume 47, Issue 1, Feb. 2001, pp.101-107.

[2] Tosun, A.S.; Wu-Chi Feng, "Lightweight security mechanisms for wireless video transmission", International Conference on Information Technology: Coding and Computing, 2001, 2-4 April 2001 pp.157–161.

[3] Shi C., Bhargava B., "A Fast MPEG Video Encryption Algorithm", ACM Multimedia, 1998, pp.81-88.

[4] "Some facets of complexity theory and cryptography: A five-lecture tutorial", ACM Computing Surveys (CSUR), Volume 34 , Issue 4, December 2002, pp. 504-549.

[5] Tang L., "Methods for Encrypting and Decrypting MPEG Video Data Efficiently", Proceedings of the fourth ACM international conference on Multimedia, 1997, pp.219-229.

[6] Qiao L., Nahrstedt K., Tam M. C., "Is MPEG Encryption by Using Random List instead of Zigzag Order Secure", Proceedings of International Symposium on Consumer Electronics, 1997, pp. 226 –229.

[7] NIST. Data Encryption Standard. FIPS Publication 46-2, 1993.

[8] Spanos G.A., Maples T.B., "Security for Real-Time MPEG Compressed Video in Distributed Multimedia Applications", Conference Proceedings of the Fifteenth Annual International Phoenix Conference, 1996, pp.72-78, 1996.

[9] Agi I., Gong L., "An Empirical Study of Secure MPEG Video Transmissions", Proceedings of The Internet Society Symposium on Network and Distributed System Security, 22-23 Feb. 1996, pp. 137-144.

[10] Forster, R, "Manchester encoding: opposing definitions resolved", Journal of Engineering Science and Education, Volume 9, Issue 6, pp. 278-280.

[11] J. Daemen, R. Govaerts, and J. Vandewalle, "Weak keys for IDEA", CRYPTO '93. pp224–231.

[12] Gong K.L., Rowe L.A., "Parallel MPEG-1 Video Encoding", In Proceedings of the 1994 Picture Coding Symposium, 1994. http://www.bmrc.berkeley.edu/research/publications/1994/120/120.html

[13] Patel K., Smith B., Rowe L., "Performance of a Software MPEG Video Decoder", In Proceedings of ACM Multimedia 93, August 1993, pp.75-82.

TABLE 2. Decoding time comparison - VEA, modified VEA and unencrypted MPEG

Experiment Number	Time to decode No encryption	Time to decrypt Key not Manchester encoded (VEA)	Time to decrypt Manchester encoded key
1	8.339493	8.394888	8.450614
2	8.341628	8.441529	8.442086
3	8.326366	8.420917	8.439929
4	8.366644	8.429125	8.408908
5	8.354606	8.394352	8.437265

Average time to decode (File not encrypted) = 8.3457474 s
Average time to decrypt (Key not encoded) = 8.4161622 s
% Increase in time wrt. no encryption/decryption = 0.8437 %

Average time to decrypt (Manchester encoded key) = 8.4357604 s
% Increase in time wrt. no encryption/decryption = 1.07855 %
% Increase in time wrt. VEA = 0.233 %

TABLE 3. Comparison of Rotation key, XOR key and IDEA

Experiment Number	Rotation of key	XOR of key	IDEA (30 keys) Key generation	+ Decryption
1	8.429115	8.421623	2.725072	8.424360
2	8.398307	8.407659	2.716831	8.432481
3	8.407394	8.442607	2.766879	8.451433
4	8.430594	8.411961	2.734230	8.440368
5	8.421079	8.429691	2.755404	8.437871

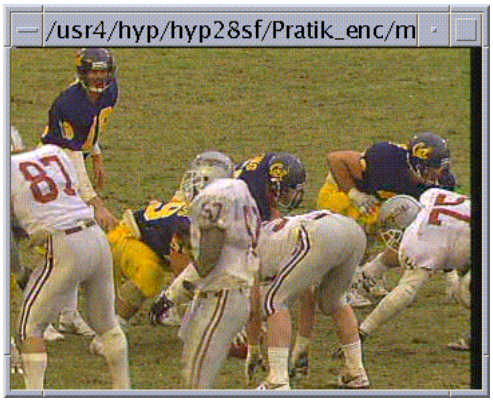
Average time for IDEA key generation = 2.7396832 s

Average time for decryption = 8.4373026 s
Total time using IDEA = 11.1769858 s.

TABLE 4. Decoding time comparison - Rotation key, XOR key, IDEA with VEA and unencrypted MPEG

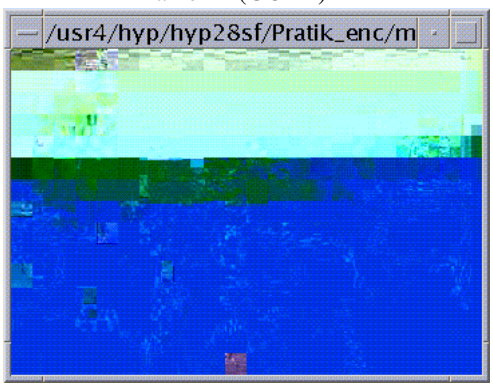
Algorithm	Average time	% Increase No encryption/decryption	% Increase compared to VEA
Rotation	8.4172978	0.8573 %	0.0135 %
XOR	8.4227082	0.9222 %	0.0778 %
IDEA	11.1769858	33.924 %	32.8 %

Frame 1 (GOP 1)



Frame 10 (GOP 1)

Frame 12 (GOP 2)



Frame 22 (GOP 3)

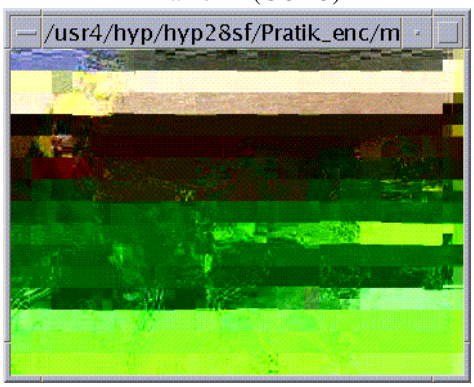


Figure 7. The VEA key obtained using known plain-text attack. Frames in all the GOP's are clearly visible