Bitstream-based JPEG Encryption in Real-time

Stefan Auer
Salzburg University of Applied Sciences, Austria
Alexander Bliem
Salzburg University of Applied Sciences, Austria
Dominik Engel
Salzburg University of Applied Sciences, Austria
Andreas Uhl
University of Salzburg, Austria
Andreas Unterweger*
University of Salzburg, Austria

ABSTRACT

We propose a framework to encrypt Baseline JPEG files directly at bitstream level, i.e., without the need to recompress them. Our approach enables encrypting more than 25 pictures per second in VGA resolution, allowing real-time operation in typical video surveillance applications. In addition, our approach preserves the length of the bitstream while being completely format-compliant. Furthermore, we show that an attack on the encryption process, which partly relies on AES, is practically infeasible.

Keywords: JPEG, Encryption, Real-time, Length-Preserving, Framework

INTRODUCTION

The encryption of compressed images to ensure privacy is an active research topic for a variety of different compressed image and video formats. For JPEG-compressed images (International Telecommunication Union, 1992) in particular, several approaches exist due to the widespread use of this image format. Most of them require recompressing the original data to some extent. The method proposed in this paper operates on bitstream level and uses only swap and scramble operations, which makes it very fast.

A number of approaches have been proposed which do either not preserve the length of the original file or break format compliance. These include techniques such as zig zag permutation (Kailasanathan, 2002; Tang, 1996), which significantly increases the file size, and the use of permuted Huffman tables (Wu & Kuo, 2000). Similarly, DC bit plane scrambling as proposed for example by Khan, Jeoti, & Kahn (2010) increases the file size and is therefore not length preserving as opposed to our approach.

In terms of simple length-preserving encryption algorithms on DCT-based images and videos, pseudo-randomly toggling DC and/or AC coefficient signs as proposed for example by Potdar, Talele, & Gandhe (2009), or Bhargava, Shi, & Wang (2003) is frequently used.

However, the attack complexity for breaking such themes is significantly lower than in our approach as we encrypt multiple bits per coefficient instead of only one (the sign bit).

Similarly, encrypting a limited number of bits on bitstream level starting from the DC coefficient (Puech & Rodrigues, 2005) or the high-frequency AC coefficients (Puech & Rodrigues, 2007), respectively, is of lower security as compared to the proposed approach which encrypts all coefficients and additionally increases the complexity by reordering blocks. The technique of reordering all blocks within a picture, which is used as part of our approach in a spatially limited fashion, has already been proposed in Ye, Zhengquan, and Wei (2006), Lian, Sun, and Wang (2004) and Niu, Zhou, Ding, and Yang (2008) and analyzed in Wen, Severa, Zeng, Luttrell, and Jin (2002) and others. Although it increases the total attack complexity depending on the picture size, it does not allow for RoI encryption without a significant decrease in attack complexity, as opposed to our approach.

In terms of code-word-based techniques such as the one we propose, only a small number approaches have been published. Besides swapping code words of equal length between blocks for AC value histogram spreading as proposed in Yang, Zhou, Busch, and Niu (2009), a method to shuffle code words with the same in-block position between blocks exists for MPEG-4 (Wen et al., 2002) which could also be applied to JPEG pictures. However, the latter approach may yield non-format-compliant bitstreams and both methods are not intended to be used for RoI encryption as opposed to our approach.

Another method described in Wen et al. (2002) encrypts multiple concatenated VLC (Variable Length Code) symbols and maps them to another string of valid VLC symbols so that the total length is preserved. Note that this approach, which has been applied to MPEG-4 bitstreams, cannot be used for JPEG as, in the latter, each Huffman code word is followed by a signed coefficient residual represented by a number of bits which is encoded in the Huffman code word. Changing the code words in a length-preserving way changes the number of coefficient bits encoded in the Huffman code word as opposed to the actual subsequent bits in the bitstream, making the bitstream parser get out of sync and thus breaking format compliance.

Note that our bitstream-based approach is designed for encryption without the need for recompression, which is useful when there is no possibility to intercept the encoding process. One practical use case is the encryption of pictures from surveillance cameras, most of which send streams of JPEG pictures which are already encoded. Although real-time recompression is possible with state-of-the-art hardware, omitting this step may save equipment and costs.

This paper is structured as follows: In the “Bitstream Encryption” section we describe our encryption approach. Subsequently, in the “Security Analysis” section, we give an estimation of the effort required for a successful attack on a picture encrypted with our approach. Next, we present our framework which implements our encryption approach in the “Real-time Encryption Framework” section. Finally, we evaluate the performance of the proposed encryption framework in the “Performance evaluation” section before concluding the paper.

This paper is an extension of our previous work (Unterweger, 2012) which introduced our length-preserving JPEG encryption approach. In this paper, we add an additional DC coefficient encryption step and introduce a practical implementation which is capable of encrypting JPEG images using our approach. Furthermore, we evaluate the performance of our implementation thoroughly, showing that it is suitable for encoding typical JPEG-compressed surveillance camera output in real-time, which makes additional storage facilities unnecessary.

Bitstream Encryption

In baseline JPEG, 8x8 blocks of cosine-transformed quantized AC coefficients are zig-zag scanned and run-length coded before being Huffman coded. Hereby, the Huffman code words only encode run-length pairs as symbols, whereas the actual coefficient values are written directly to the bitstream as signed residue from 0 or -2^s+1, respectively, where *s* denotes the length part of the run-length symbol.

Our approach consists of four different operations. First, the order of the run-length coded symbols together with their corresponding coefficient values (referred to as code-word-value pairs henceforth) is permuted. Second, the coefficient value bits are scrambled and third, the order of all blocks within an iMCU (interleaved Minimum Coding Unit) which use the same Huffman table is permuted. Finally, the DC coefficients are scrambled using the approach proposed by Niu et al. (2008).

As DC coefficient scrambling is known to be easy to break, it is performed as a separate step after the first three using a different key. The purpose of it is purely to make the image appear “more” encrypted to a human viewer. Note that the remaining three steps are already relatively secure, as shown in the “Security Analysis” section.

The fourth step, Niu et al.'s (2008) approach, is not described in detail herein, noting only that it is format-compliant and length-preserving. Both, the second and third step, are described in detail at the end of this section, while the subsequent paragraphs describe the first operation, i.e., the permutation of the order of code-word-value pairs.

Permutations of this order lead to a change of the order of the zero runs in each block, thereby altering the positions of the coefficient values within the 8x8 block. Figure 1 depicts this by example.

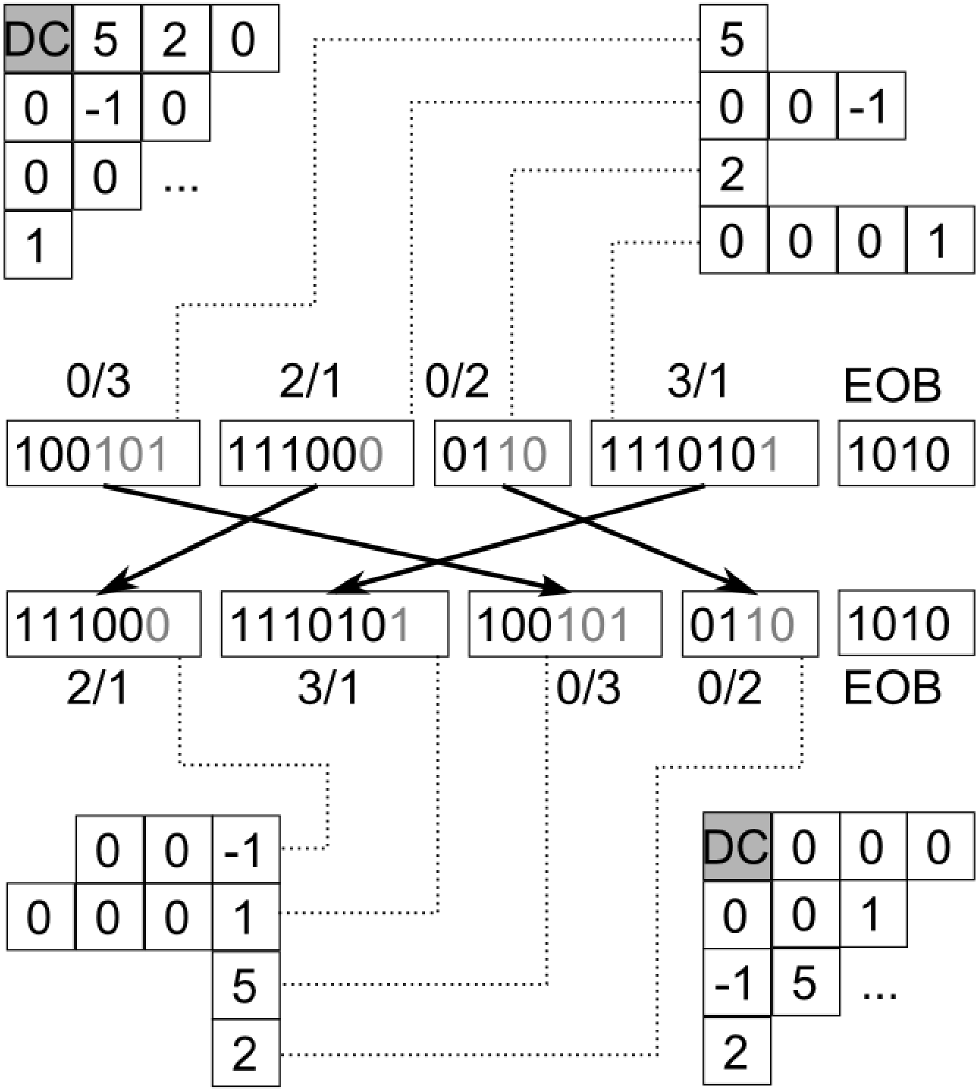

Figure 1. Example of run-length permutation: The order of Huffman coded run-length symbols and their corresponding coefficient values is permuted, thereby changing the position of the values in the coefficient matrix.

On the top left, an exemplary block with four non-zero coefficient values is shown. The dots denote that the rest of the coefficients are zero. The DC coefficient is neither changed nor considered in this step. On the top right, the zig-zag scanned values of the exemplary block are depicted and grouped with their preceding zeros. Each of these groups is coded as a Huffman code word (black) of the run-length symbol (depicted on top of each Huffman code word) and the coefficient value (gray). For example, the first coefficient (5), which is preceded by no (i.e., 0) zeros, requires three bits (101) to be represented, thus leading to a run-length of 0/3. As the rest of the blocks' coefficients apart from the four ones depicted are zero, an EOB (End Of Block) is signaled.

By swapping the groups of Huffman code words and coefficient values (if there is more than one group), the zero runs and therefore the position of the coefficient values within the block change, as depicted at the bottom of Figure 1. However, the bitstream remains format-compliant as the exchange of code words does neither change the Huffman codes themselves nor does it change the total number of coefficients. In addition, it preserves the length of the JPEG file.

The code-word-value pair order permutation through element-wise reordering is derived as follows. Before processing a JPEG file, an AES (National Institute of Standards and Technology, 2001) implementation in OFB (Output Feedback) mode is initialized with a given initialization vector and key, which can be file- or user-dependent. It then serves as a PRNG (Pseudo-Random Number Generator) by using n AES-encrypted output bits, where 2^n is the desired range of the PRNG. Note that any cryptographic PRNG could be used here.

Code-word-value pair order permutation is then performed by swapping the current code word and its corresponding coefficient value at position i with the code-word-value pair at position *rand*(n) where *rand* denotes a call to the AES-based PRNG with an upper value bound of n and n is equal to the number of total code-word-value pairs. For the example in Figure 1, n = 4, yielding the consumption of two encrypted output bits of the AES encoder per possible swap operation.

In addition to code-word-value pair order permutation, our approach changes the coefficients' values in the bitstream (gray bits in Figure 1). This is done by toggling each of the n value bits depending on whether or not the AES-based PRNG described above returns a binary zero or one when using one bit. Similar to the run-length order permutation, this does not change the length of the JPEG file as the value bits actually represent a signed residual of fixed size per code word (see above).

Furthermore, the order of all blocks using the same Huffman table within an iMCU is permuted. Figure 2 shows an example with 4:2:0 sub-sampling (Kerr, 2013) where the U and the V block use the same Huffman table (marked gray). After the permutation, the order of U and V is switched, with the bitstream still being format-compliant. The permutation itself is derived as described for run-length permutations above. Note that no code-word-value pairs are exchanged among the blocks as this would break format compliance – only whole blocks with all their code-word-value pairs are exchanged. Again, this does not change the length of the JPEG file.

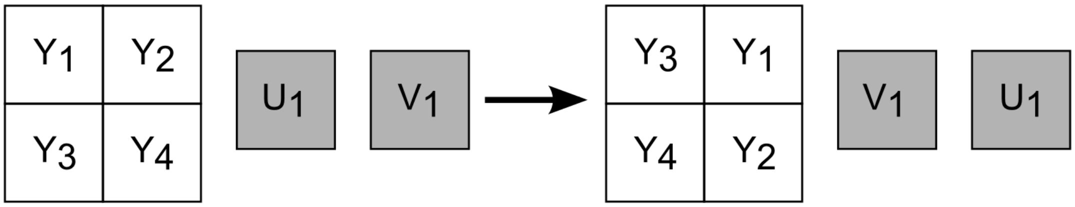

Figure 2. Example of block order permutation: The order of blocks using the same Huffman code words within an iMCU is permuted. In this example, the Y blocks use one set of Huffman code words (white), while both, the U and the V block, use another (gray).

Figure 3 (c) shows an example of a picture with a JPEG quality of 75% that is encrypted with our approach. Note that the number of possible permutations in blocks with few code-word-value pairs and/or small coefficient values (e.g., in the area above the woman's head) is small, making those blocks appear less noisy due to the lack of high frequency components. Conversely, the other blocks exhibit significant distortion due to the reordering and scrambling, revealing that the local encryption strength of our approach depends on the amount of information contained in a block as explained in more detail in the next section.

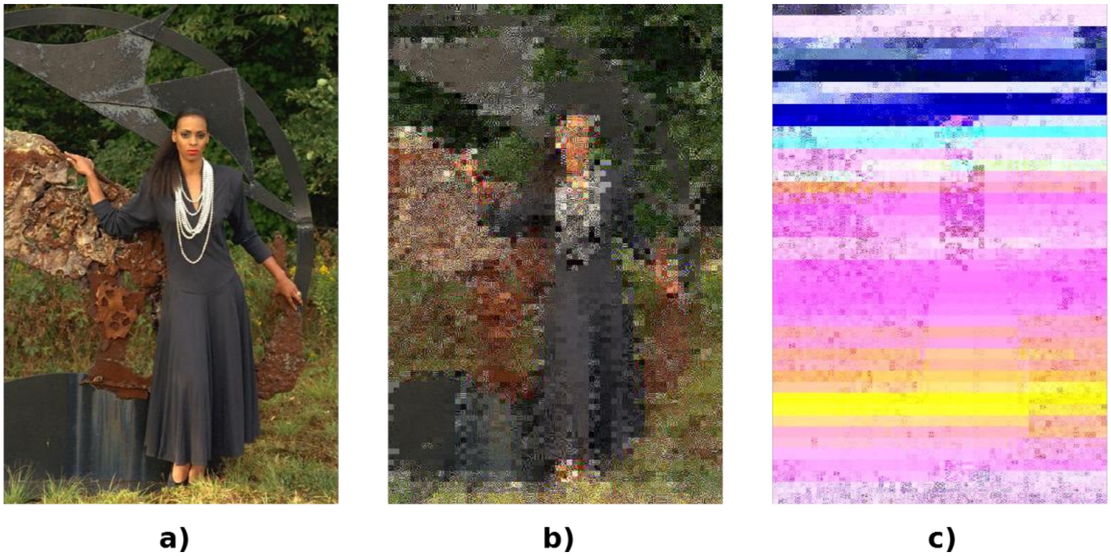

Figure 3. Example of a picture from the LIVE reference picture set (Seshadrinathan, Soundararajan, & Bovik, 2010) before (a) and after (c) encryption with the proposed method. (b) depicts a variant of our approach where DC coefficient scrambling is omitted.

Note that although the woman's silhouette is recognizable in the encrypted picture when DC coefficient scrambling is omitted (Figure 3 (b)), no more details (like facial characteristics) can be extracted from it. All information contained in high frequency coefficients is lost without proper decryption and therefore does not compromise the security of our approach on small scales, even if there is a successful attack on the scrambled DC coefficients.

Security Analysis

In order to assess the cryptographic security of our approach, its three main components are analyzed in terms of attack complexity, i.e., the number of possible combinations per iMCU and the probability of success for key extraction in a known-plaintext attack. The key space depends on the AES key size and can be up to $2^{256} > 10^{77}$ for AES with 256 bit keys (National Institute of Standards and Technology, 2001).

The approach described in the previous section relies on three independent scrambling mechanisms, disregarding the DC coefficient scrambling which can easily be circumvented: permuting the order of code-word-value pairs, toggling value bits and permuting the order of blocks within an iMCU. Due to their independence, the number of possible combinations can be analyzed separately and eventually multiplied to yield the overall number of possible combinations.

Let m denote the total number of blocks in an iMCU and let n_i denote the total number of code words in the i^{th} block of an iMCU. Let $l_{i,j}$ denote the length of the j^{th} code-word-value pair of the i^{th} block of an iMCU in bits. The number of possible values (i.e., bit combinations) $c_v(i, j)$ for each value is $2^{l_{i,j}}$, thus being

$$N_v = \prod_{i=1}^{m} \prod_{j=1}^{n_i} c_v(i,j) = \prod_{i=1}^{m} \prod_{j=1}^{n_j} 2^{l_{i,j}} \tag{1}$$

for all blocks within an iMCU.

The number of permutations $p_{rlv}(i)$ of code-word-value pairs of each block is $n_i!$, where $x!$ denotes the factorial of x. Thus, the total number of code-word-value pair permutations is

$$N_{rlv} = \prod_{i=1}^{m} p_{rlv}(i) = \prod_{i=1}^{m} n_i! \tag{2}$$

for all blocks within an iMCU. Similarly, the number of block permutations $p_b(x)$ for x blocks which use the same Huffman code words is $x!$, thus being

$$N_b = \prod_{k=1}^{h} p_b(n_h(k)) = \prod_{k=1}^{h} n_h(k)! \tag{3}$$

for all blocks within an iMCU, where h denotes the total number of different AC Huffman tables and $n_h(k)$ is the number of blocks using the k^{th} Huffman table so that

$$\sum_{k=1}^{h} n_h(k) = m \tag{4}$$

In total, this yields an overall number N of possible combinations per iMCU of

$$N = N_v \cdot N_{rlv} \cdot N_b = \prod_{i=1}^{m} \prod_{j=1}^{n_i} 2^{l_{i,j}} \cdot \prod_{i=1}^{m} n_i! \cdot \prod_{k=1}^{h} n_h(k)! \tag{5}$$

In order to estimate the values for $l_{i,j}$ and n_i for typical natural JPEG pictures, the reference pictures of the LIVE data base presented in Seshadrinathan et al. (2010) are encoded with different quality settings (between 0 and 100% with 5% step size) using the JPEG reference encoder. The encoded files are analyzed in terms of the average number of runs per block and the average length of coefficient values.

We consider the average values to be an appropriate measure for the following reason: The attack complexity of our algorithm depends on the number of code-word-value pairs within a block, i.e., it varies with its number of non-zero coefficients. The distribution of the latter (not depicted) reveals that the information content of a block with a high number of code-word-value pairs is greater than that of a block with a small number thereof.

We assume that the semantic information content, i.e., the amount of information a human can extract, of a block roughly correlates with its information content in terms of the number of code-word-value pairs. This assumption is supported by the fact that blocks with a small number of code-word-value pairs (e.g., one or two) are very unlikely to compromise the content on a small scale, thus having a low amount of semantic information content.

Note that semantic information content is hard to measure and therefore requires a justifiable approximation. Thus, we use the average number of code-word-value pairs to represent a block with a medium to high amount of information content as a practical approximation of a possibly critical block of the picture.

Figure 4 depicts the average number of runs per block and the average length of coefficient values as functions of JPEG quality for the reference pictures of the LIVE data base. Both functions increase monotonically with quality, showing that pictures with finer quantization contain a higher number of runs per block and longer coefficient values. This allows for a higher number of combinations, making an attack on an iMCU harder.

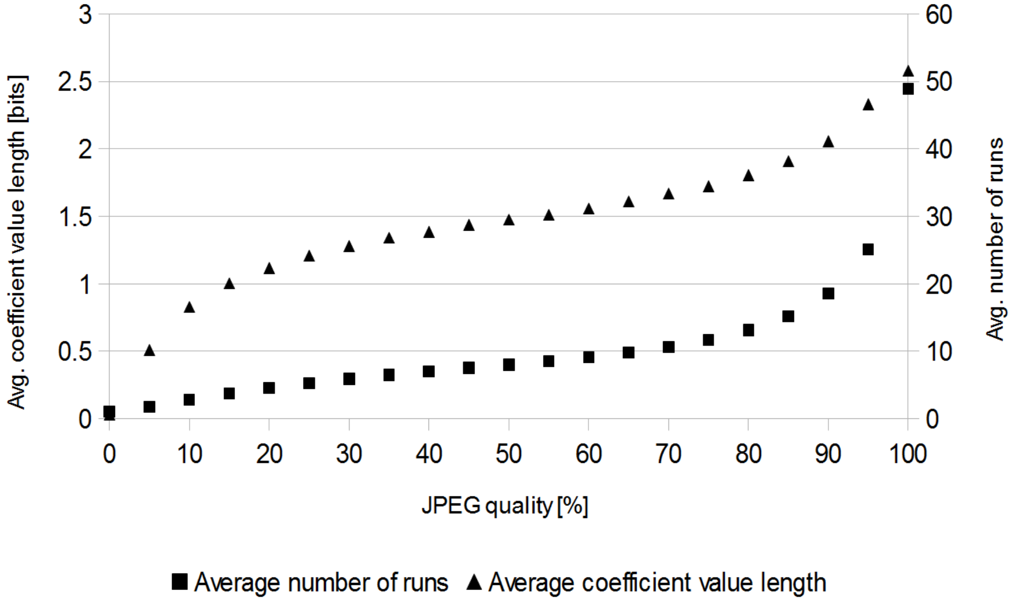

Figure 4. Average number of runs per block (squares) and average coefficient value bit length (triangles) over JPEG quality for the JPEG-compressed LIVE reference picture set (Seshadrinathan et al., 2010).

Using the average values $n(q)$ and $l(q)$ at quality q instead of all n_i and $l_{i,j}$, respectively, yields a simplified equation for the overall number $N(q)$ of combinations dependent on the JPEG quality q:

$$N(q)=2^{l(q)\cdot m\cdot n(q)}\cdot(n(q)!)^{m}\cdot\prod_{k=1}^{h} n_h(k)! \tag{6}$$

Using the Gamma function as an extension of the factorial function which is only defined for natural numbered arguments, $N(q)$ can be expressed as

$$N(q)=2^{l(q)\cdot m\cdot n(q)}\cdot(\Gamma(n(q)+1))^{m}\cdot\prod_{k=1}^{h} n_h(k)! \tag{7}$$

Thus, an attack on an iMCU composed of 4:2:0 sub-sampled (i.e., m = 6 as entailed by the JPEG standard (International Telecommunication Union, 1992)) average blocks compressed with JPEG quality q requires trying

$$N(q)=2^{6l(q)\cdot n(q)}\cdot(\Gamma(n(q)+1))^{6}\cdot 4!\cdot 2!=48\cdot 2^{6l(q)\cdot n(q)}\cdot(\Gamma(n(q)+1))^{6} \tag{8}$$

combinations, if the AC coefficients of both chroma components use the same Huffman table. For a JPEG quality of 75% (which is the default value of the JPEG reference encoder), this yields $N(75) > 10^{87}$, which is greater than the number of possible 256 bit keys, thus making a brute-force attack on the AES key more efficient than trying to reorder and descramble the iMCU. Figure 5 illustrates this and a comparison to the attack complexity for AES for different JPEG quality values.

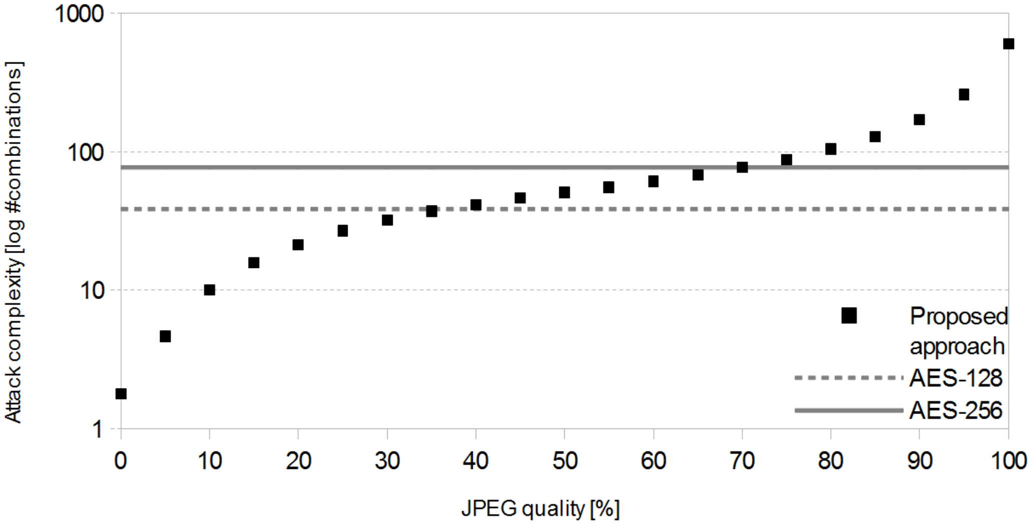

Figure 5. Average attack complexity over JPEG quality for the JPEG-compressed LIVE reference picture set (Seshadrinathan et al., 2010) for the proposed approach and AES for comparison.

Note that each iMCU can be attacked separately, thereby possibly revealing enough information about the picture that the rest of the picture's iMCUs do not need to be decrypted. This way, the total number of combinations for a full picture does not represent a valid metric for the number of combinations to try for an attack.

Note that an attacker may eliminate some orderings of code-word-value pairs as high values of high frequency AC coefficients (most of all chrominance) are very unlikely to appear in natural images (Pennebaker, & Mitchell, 1993). This reduces the effective values of n_i and $n(q)$, respectively. However, it is hard to quantify the actual reduction as it depends on the picture content, potentially known signal characteristics and the coefficient distribution of the blocks in the attacked iMCU.

Regarding known-plaintext attacks, AES is considered to be not vulnerable (Daemen & Rijmen, 2002). If an attacker has both, the original and the encrypted JPEG picture, deriving the key from the permutations and scrambled bits is nearly as hard as a brute-force attack on the AES key itself (Bogdanov, Khovratovich, & Rechberger, 2011), which is considered infeasible for 256 bit keys by today's standards.

Real-time Encryption Framework

To evaluate our encryption approach, we created a framework which implements it. Our framework consists of two parts: The first part is a DLL (Dynamic-Link Library) written in C, which is theoretically platform-independent and performs the actual encryption and decryption. The second part is a .NET Windows Forms GUI (Graphical User Interface) implemented in C#

which enables easy selection of JPEG live streams or stored JPEG pictures. The GUI calls the DLL using PInvoke on Windows (Microsoft, 2013a).

The C DLL is based on the open source project NanoJPEG (Fielder, 2013) and uses the entropy coding implementation of Barrett (2013). Additionally, the AES implementation of Sevilla (2013) is used for the AES-based PRNG as described in the “Bitstream Encryption” section.

The encryption is performed as follows after the bitstream to be encoded is provided to the DLL: First, a conservative approximation of the required amount of memory to store both, the original and the encrypted bitstreams, is made based on the size of the original bitstream. Second, the original bitstream in memory is processed and encrypted on-the-fly to yield the encrypted bitstream. Finally, the encrypted bitstream is written to a memory location provided by the DLL's caller.

The on-the-fly encryption step distinguishes between data which needs to be encrypted and the remainder which can be copied unmodified. Processing the input bitstream from start to end, markers are parsed and evaluated. SOF (Start of Frame), DHT (Define Huffman Table) and SOS (Start Of Scan) markers are evaluated and simultaneously written to the memory location of the output bitstream.

For the actual scan data, decoding is performed at Huffman-code level in order to distinguish the Huffman code words from one another to subsequently apply the encryption algorithm described in the “Bitstream Encryption” section. Once the code-word-value pairs of one iMCU are decoded and stored in memory, the encryption algorithm is applied. Subsequently, encrypted image data is written to the corresponding memory location and the next iMCU is processed, until the end of the scan is reached.

Performance Evaluation

In this section, we assess the performance of our implementation. We consider real-time encoding of JPEG-compressed input streams from surveillance cameras to be one of the main applications of our implementation. Thus, our evaluation focuses solely on execution time and real-time constraints.

As this use case does usually not require a GUI, but exhibits heavily use-case-dependent I/O (Input/Output) performance, we limit our measurements to the net execution time of our DLL's encryption routine which is described in the “Real-time Encryption Framework” section, thus disregarding execution time spent for I/O tasks.

All measurements were performed on an Intel Core 2 Duo T9600 CPU running at 2.8 GHz. In order to minimize measurement errors due to background processes, we booted Windows 7 32-bit in “Safe Mode with Command Prompt” and executed the encryption process which invoked our C DLL so that it was bound to one fixed CPU core with the highest possible process priority. All execution times were measured using QueryPerformanceCounter (Microsoft, 2013b) calls as recommended by Microsoft (2013c) for accurate measurements on multi-core systems.

We distinguish between measurements of on-line and off-line encryption. In order to compensate for caching effects and fluctuations, each picture to be encrypted off-line was in fact encrypted five times for cache warming, i.e., without considering execution time, and subsequently encrypted 20 times, yielding a final average execution time. In contrast, all pictures to be encrypted on-line were encrypted only once, i.e., without cache warming and without averaging execution times. This effectively simulates practical execution conditions, in which pictures are encrypted one after another without any potential benefits from caching.

To simulate different working conditions, we use different sets of input pictures: For a practical on-line field test with actual surveillance camera pictures, we use the BEHAVEDATA (Laghaee, 2013) picture sets, courtesy of EPSRC project GR/S98146, consisting of 11200 and 62366 JPEG images corresponding to 95 and 100% quality, respectively, each with a spatial resolution of 640x480 pixels (VGA).

To evaluate the effect of JPEG quality on off-line execution time, we use the 29 images of the LIVE reference picture set (Seshadrinathan et al., 2010), ranging between spatial resolutions of 634x438 and 768x512 pixels. We encode them using the standard JPEG encoder with default settings and quality values between 0 and 100% with a step size of 5%.

To evaluate the effect of spatial resolution on off-line execution time, we use the 24 Kodak high-resolution images from SCIEN (2013) with a spatial resolution of 3072x2048 pixels each. Starting at this resolution, we derive smaller versions of the images by symmetrically cropping them in steps of 96/64 pixels in width/height using ImageMagick 6.6.0-4 (ImageMagick Studio, 2013), maintaining both, the original aspect ratio of 3:2 and the images' centers without the need to perform interpolation of any kind. The cropped images are then encoded using the standard JPEG encoder with default settings, i.e., a quality of 75%.

The results of the practical on-line field test in terms of maximum encryption time per picture are 13.86 ms for the 95% quality picture set and 29.80 ms for the 100% quality picture set, respectively. This demonstrates the capability of our implementation to perform hard real-time encoding at a frame rate of at least 25 frames per second, which would allow for a maximum execution time of 40 ms per frame.

Note that the average execution times are 12.11 ms with a standard deviation of 0.87 and 26.34 ms with a standard deviation of 1.16, respectively, which allows for even higher performance under soft real-time conditions, i.e., when buffering makes fluctuations in execution acceptable. In conclusion, real-time processing at 25 frames per second at VGA (640x480) resolution with up to 100% JPEG quality is possible using our implementation.

The remainder of this section is dedicated to the evaluation of off-line encoding performance. Figure 6 shows the effect of JPEG quality on execution time. Except for a quality value of 100%, all pictures can be encrypted in soft real-time at 25 frames per second. Note that this does not contradict the on-line measurements as the picture sizes used in this experiment (see above) are mostly larger than VGA and exhibit stronger fluctuations.

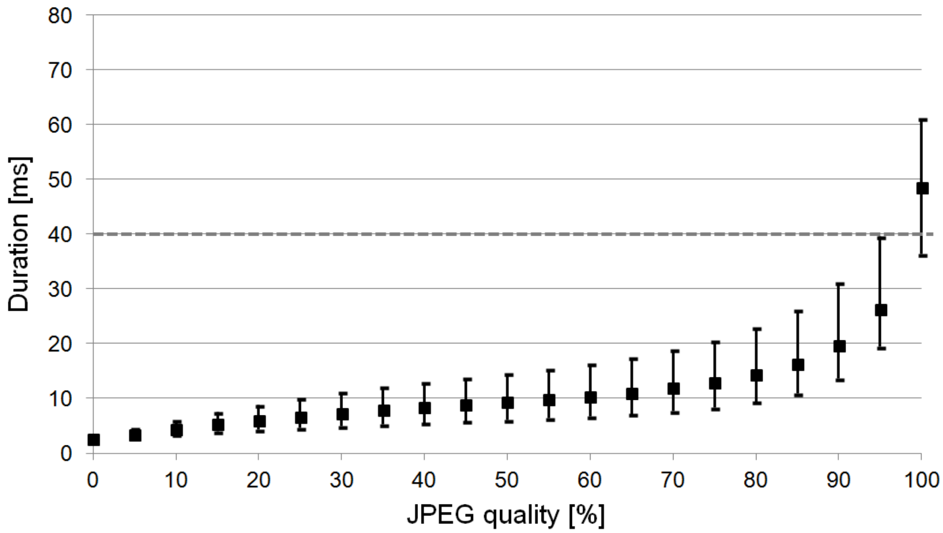

Figure 6. The effect of JPEG quality on net execution time. Squares depict the average execution time of all measurements per quality value. The dashed gray line indicates the threshold for soft real-time encryption at 25 frames per second.

Although execution time increases monotonically with JPEG quality, the increase is larger for higher quality values. This is due to the exponential increase in the number of non-zero coefficients at high quality values, which results in a large number of run-length-value pairs to be swapped and scrambled. Conversely, the number of non-zero coefficients is relatively low and changes sub-linearly for most low quality values.

Figure 7 shows the effect of spatial resolution on execution time. All pictures of the data set show a nearly linear dependency between the pictures' spatial resolution and the net execution time required for their encryption. The main difference between the pictures is the number of their non-zero coefficients induced by their content, which reflects in the slope of the quasi-linear increase in execution time. Thus, pictures with similar image characteristics exhibit similar slopes.

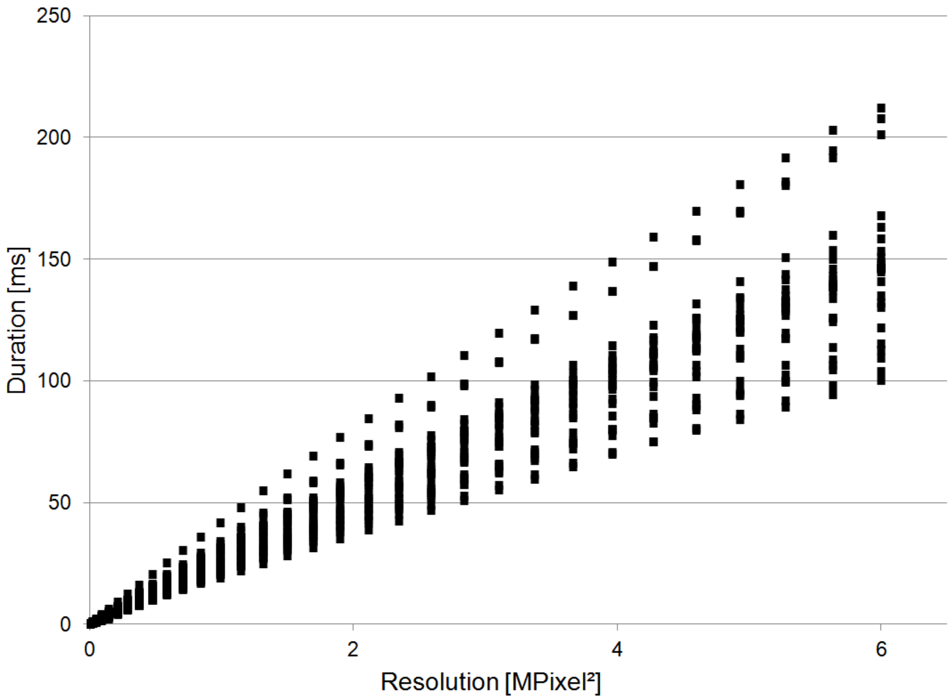

Figure 7. The effect of spatial resolution on net execution time. The different curves correspond to different images, revealing that the image content influences the slope of the quasi-linear increase in execution time as the spatial resolution increases.

As the number of non-zero coefficients directly affects the file size of the corresponding JPEG images, Figure 8 illustrates the results of Figure 7 in terms of file size instead of spatial resolution. It is not surprising that the execution time is, again, nearly linearly dependent on the file size. As our encryption approach is bitstream based, this is a desirable property, as the picture size and content can be used to estimate the required encryption time.

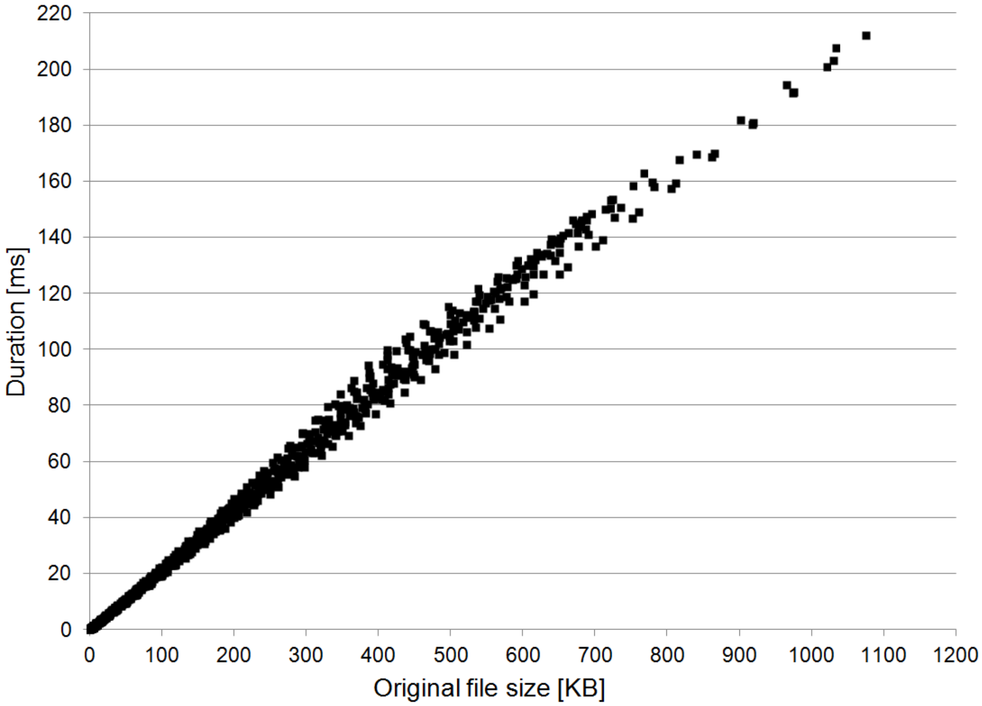

Figure 8. The effect of JPEG file size on net execution time. The different curves correspond to the different images of Figure 7, showing that there is a high correlation between the spatial dimension of a picture and its JPEG file size.

Future Work

Multiple extensions of our approach are possible and remain future work: First, blocks between different iMCUs could be swapped as long as the corresponding blocks in the iMCUs use the same Huffman tables, yet increasing the total number of possible combinations. This extension is easy to implement and preserves the length of the bitstream.

Second, it is possible to use our proposed approach for RoI encryption. iMCUs containing the RoIs can be encrypted while the rest of the picture stays intact. If DC coefficient encryption is not used at all, this form of RoI encryption is length-preserving. However, if DC coefficient encryption is applied, the DC differences around each RoI need to be corrected in order to keep the non-encrypted picture areas undistorted. As the corrected values may differ in length, this approach would not be length-preserving anymore.

Although limiting the encryption to a set of iMCUs is trivial, signaling them is not, if the length is to be preserved. However, if this limitation is lifted, embedding the RoI information can be done by inserting a comment segment into the bitstream which contains for example a bitmap of all iMCUs where a one denotes that the iMCU is encrypted, while a zero denotes that it is not. Such a comment segment increases the file size by the marker size (two bytes) plus its length field (two bytes) plus the size of the bitmap or any other desired form of RoI position/size encoding.

CONCLUSION

We proposed a framework which encrypts JPEG files by performing swap and scramble operations on their respective bitstreams in a format-compliant and length-preserving way. As our encryption approach operates at bitstream level, it does not require recompression, thus enabling real-time encryption for at least 25 pictures per second in VGA resolution. Furthermore, we showed the practical infeasibility of both, brute-force and known-plaintext attacks as well as the extensibility of our apprpach to support RoI encryption, which makes it suitable for surveillance applications.

ACKNOWLEDGEMENTS

This work is supported by FFG Bridge project 832082.

REFERENCES

Barrett, S. (2013). *stbi-1.33 - public domain JPEG/PNG reader*. Retrieved February 6, 2013, from http://nothings.org/stb_image.c.

Bhargava, B., Shi, C., & Wang, Y. (2004). MPEG video encryption algorithms. *Multimedia Tools and Applications, 24*(1), 57-79.

Bogdanov, A., Khovratovich, D., & Rechberger, C. (2011). Biclique Cryptanalysis of the Full AES. In D. Lee, & X. Wang (Eds.), *Advances in Cryptology (ASIACRYPT 2011): Vol. 7073 of Lecture Notes in Computer Science* (pp. 344-371). Berlin / Heidelberg: Springer.

Daemen, J., & Rijmen, V. (2002). *The Design of Rijndael: AES – The Advanced Encryption Standard*. Springer Verlag.

Fiedler, M. (2013). *NanoJPEG: a compact JPEG decoder*. Retrieved February 6, 2013, from http://keyj.emphy.de/nanojpeg/.

ImageMagick Studio (2013). *Convert, Edit, and Compose Images*. Retrieved February 9, 2013, from http://www.imagemagick.org/script/index.php.

International Telecommunication Union (1992). *ITU-T T.81. Digital compression and coding of continuous-tone still images | Requirements and guidelines.* Geneva, Switzerland: International Telecommunication Union.

Kailasanathan, C. (2002). Compression performance of JPEG encryption scheme. In A. N. Skodrus, & A. G. Constuntinides (Eds.), *Proceedings of the 14th International IEEE Conference on Digital Signal Processing (DSP '02)*. IEEE.

Kerr, D. A. (2013). *Chrominance Subsampling in Digital Images*. Retrieved February 2, 2013, from http://dougkerr.net/pumpkin/articles/Subsampling.pdf.

Khan, M., Jeoti, V., & Khan, M. (2010). Perceptual encryption of JPEG compressed images using DCT coefficients and splitting of DC coefficients into bitplanes. In *2010 International Conference on Intelligent and Advanced Systems (ICIAS)* (pp. 1-6). IEEE.

Laghaee, A. (2013). *BEHAVE Interactions Test Case Scenarios*. Retrieved February 9, 2013, from http://groups.inf.ed.ac.uk/vision/BEHAVEDATA/INTERACTIONS/.

Lian, S., Sun, J., & Wang, Z. (2004). A novel image encryption scheme based-on JPEG encoding. In *Proceedings of the Eighth International Conference on Information Visualisation 2004 (IV 2004)* (pp. 217-220). IEEE.

Microsoft (2013a). *Calling Native Functions from Managed Code*. Retrieved February 6, 2013, from http://msdn.microsoft.com/en-us/library/ms235282.aspx.

Microsoft (2013b). *QueryPerformanceCounter function (Windows)*. Retrieved February 9, 2013, from http://msdn.microsoft.com/en-us/library/windows/desktop/ms644904%28v=vs.85%29.aspx.

Microsoft (2013c). *Game Timing and Multicore Processors (Windows)*. Retrieved February 9, 2013, from http://msdn.microsoft.com/en-us/library/windows/desktop/ee417693%28v=vs.85%29.aspx.

National Institute of Standards and Technology (2001). *FIPS-197 – Advanced Encryption Standard (AES)*. Gaithersburg, MD: National Institute of Standards and Technology.

Niu, X., Zhou, C. Ding, J., & Yang, B. (2008). JPEG Encryption with File Size Preservation. In J.-S. Pan, X. M. Niu, H.-C. Huang, & L. C. Jain (Eds.), *International Conference on Intelligent Information Hiding and Multimedia Signal Processing 2008 (IIHMSP '08)* (pp. 308-311). IEEE.

Pennebaker, W., & Mitchell, J. (1993). *JPEG – Still image compression standard*. New York, New York: Van Nostrand Reinhold.

Potdar, U., Talele, K. T., & Gandhe, S. T. (2009). Comparison of MPEG video encryption algorithms. In *ICAC3 '09: Proceedings of the International Conference on Advances in Computing, Communication and Control* (pp. 289-294). New York, New York: ACM.

Puech, W., & Rodrigues, J. M. (2005). Crypto-Compression of Medical Images by Selective Encryption of DCT. In *European Signal Processing Conference 2005 (EUSIPCO'05)*. EURASIP.

Puech, W., & Rodrigues, J. M. (2007). Analysis and cryptanalysis of a selective encryption method for JPEG images. In L. O'Conner (Ed.) *WIAMIS '07: Proceedings of the Eight International Workshop on Image Analysis for Multimedia Interactive Services*. Los Alamitos, California: IEEE Computer Society.

SCIEN (2013). *Test Images and Videos*. Retrieved February 9, 2013, from http://scien.stanford.edu/index.php/test-images-and-videos/.

Seshadrinathan, K., Soundararajan, R., Bovik, A., & Cormack, L. (2010). Study of Subjective and Objective Quality Assessment of Video. *IEEE Transactions on Image Processing, 19*(6), 1427-1441.

Sevilla, R. R. (2013). *rijndael - An implementation of the Rijndael cipher*. Retrieved February 6, 2013, from http://www.opensource.apple.com/source/CPANInternal/CPANInternal-62/Crypt-Rijndael/rijndael.h.

Tang, L. (1996). Methods for encrypting and decrypting MPEG video data efficiently. In P. Aigrain, W. Hall, T. D. C. Little, & V. M. Bove Jr. (Eds.) *Proceedings of the ACM Multimedia 1996* (pp. 219-229). ACM Press.

Unterweger, A., & Uhl, A. (2012) Length-preserving Bit-stream-based JPEG Encryption. In *MM&Sec'12: Proceedings of the 14th ACM Multimedia and Security Workshop* (pp. 85-89). New York, New York: ACM.

Wen, J., Severa, M., Zeng, W., Luttrell, M., & Jin, W. (2002). A format-compliant configurable encryption framework for access control of video. *IEEE Transactions on Circuits and Systems for Video Technology, 12*(6), 545-557.

Wu, C.-P., & Kuo, C.-C. J. (2000). Fast encryption methods for audiovisual data confidentiality. In *SPIE International Symposium on Voice, Video, and Data Communications, Vol. 4209* (pp. 284-295). SPIE.

Yang, B., Zhou, C.-Q., Busch, C., & and Niu, X.-M. (2009). Transparent and perceptually enhanced JPEG image encryption. In *16th International Conference on Digital Signal Processing* (pp. 1-6). IEEE.

Ye, Y., Zhengquan, X., & Wei, L. (2006). A Compressed Video Encryption Approach Based on Spatial Shuffling. In *8th International Conference on Signal Processing, Vol. 4* (pp. 16-20). IEEE.