Now You See Me, Now You Don't: Secure Cloud-based Video Editing

Odd-Arild Kristensen
State university of New York - Albany
1400 Washington Ave, Albany, NY 12222
Albany, NY USA
kristensen.odd@gmail.com

Manoranjan Mohanty
National University of Singapore
21 Lower Kent Ridge Road
Singapore, Singapore
manoranjan.jnu@gmail.com

Dr. Pradeep K. Atrey
State University of New York - Albany
1400 Washington Ave, Albany, NY 12222
Albany, NY USA
patrey@albany.edu

ABSTRACT

TO BE FILLED IN LATER

1. INTRODUCTION

Cloud computing is rapidly becoming ubiquitous. Users are moving their data to cloud data centers (CDCs) for storage, and businesses are using cloud resources to do their processing. Computationally expensive operations are ideally suited for cloud computing, and as such, off-loading heavy-lifting video editing operations to CDCs is a cost-effective alternative to local processing.

Moving data to the cloud has many benefits but it comes at the cost of security and privacy. A lot of data being stored and processed by cloud services is in plaintext, which can be intercepted by attackers as the data is being transported to and from CDCs. Several recent data breaches indicate that users data is not always secure even when stored by a third party. Companies could still be the victims of a breach and the attacks could steal or tamper with the data.

These insecurities become increasingly significant when the video data in question is of a sensitive nature. With a huge amount of video data, emerging from different applications such as camera surveillance and smart phones, and being outsourced to cloud, it is important that video editing operations performed by cloud services are secure. While traditional schemes provide computationally secure data, those that lack homomorphic properties forces users to decrypt the video before processing it, thus exposing the data to increased risk.

The challenge then becomes to construct a method that allows users to encrypt their video before uploading it to cloud services where it can be safely manipulated. By utilizing a homomorphic cryptographic scheme we can achieve this goal. Using this scheme also allows us to perform basic arithmetic operations on the ciphertext, and we can still retrieve the correct result. Thus, as long as video-editing operations can be performed without comparisons and only using the four fundamental arithmetic operations (addition, subtraction, multiplication and division), we can reliably edit videos in an encrypted domain if the following holds $E(v_1 \, o \, v_2) = E(v_1) \, o \, E(v_2)$, where v_1 and v_2 represent the two data units in a video (e.g. two frames or two pixels in a frame), E denotes the encryption function, and o represents the one of the four fundamental operations.

We propose a solution to this problem by using Shamir's Secret Sharing (SSS) scheme. For each frame in a video, we process it byte-by-byte and encrypt each byte, thus creating shares, or shadow copies, of the video. Our solution allows for cropping and scaling on the encrypted shadow copies to achieve zooming. To the best of our knowledge, this is the first paper which performs cropping and scaling of a video in encrypted domain. We achieve bicubic scaling by way of the additive and multiplicative properties of homomorphism [2]. Zooming into a frame is done by first cropping the frame and then upscaling it.

The rest of this paper is outlined as follows: Section 2 presents prior work on this topic. Section 3 describes or proposed work along with its assumptions, with Section 4 presenting our security analysis and our results. Conclusion and future improvements are discussed in Section 5.

2. RELATED WORK

Khiem et al. [9] proposed two different methods of streaming video data that allowed for dynamic zooming. By dividing a frame into macroblocks of 16x16 pixels, the macroblocks needed to display the Region of Interest — RoI — can be requested and the server will only decode the necessary macroblocks and thus achieve zooming. Tiled streaming was achieved by pre-selecting grid tiles that are encoded separately, which would be requested as the RoI changes. Then the excess data would be removed during decoding to result in a zoomed frame. Monolithing streams on the other hand use dependency graphs to calculate the necessary macroblocks to decode the requested RoI.

The SSS algorithm creates polynomials of degree $n-1$ such we need n points to reconstruct the polynomial $F(x)$.

$$F(x) = \sum_{i=0}^{n} a_i x^i \mod p$$

where a_0 is the secret we want to obscure, and any other a is a random number such that $a_i < p$. x is the number of the share being created, much like an identified, such that $x_i < p$ and p is a prime number.

To reconstruct the secret value we use Lagrange Interpolation, $L(x)$, to find the points in $L(x)$ that can recreate the polynomial $F(x)$.

$$L(x) = \sum_{j=1}^{k} y_j \prod_{i=1, i\neq j}^{k} \frac{x - x_i}{x_j - x_i} \mod p$$

$L(x)$ where $x = 0$, x_i is the share value from $F(x)$ and y_j is the number of the applied shares, their identifier, will reveal the secret when the polynomial is of degree $n = k$.

Mohanty et al. [7] demonstrated how cropping and scaling can be performed on encrypted image data using Ramp Secret Sharing. This was done by creating shadow copies of the image and then performing the cropping and scaling operations on the shares. Reconstructing these shares then resulted in an image that had been zoomed in at the specified RoI.

There have been many applications of Secret Sharing. Examples including encryption of images etc. There have been efforts to limit the storage requirements of SSS, i.e. Ramp Secret Sharing. While these schemes reduce how much space is needed to store the shares, the scheme loses its information theoretically secure property, thus making it vulnerable to attacks. Alharthy et al. [1] showed how SSS scheme could be applied to video while attaining perfect secrecy. The proposed work is an extension of Mohanty and Alharthy work.

Lu et al. [6] discussed various applications of video processing in the encrypted domain, the challenges and potential techniques, focusing on video search, classification and summarization. Brown et al. demonstrated how to automatically detect different events in a video stream. Saini showed that image forgery is more reliably implemented in hardware, not in software. Similarly, Saini et al [10] found that image enhancement can be implemented better using VHDL for in-chip image processing. Bitouk et al. [3] proposed a system to replace faces in photographs. Avidan and Butman [11] presented a method to securely perform face detection without leaking vital information about the image. Chen et al. [4] proposed a method to securely calculate linear equations in CDCs. Newton et al. [8] presented an algorithm, k-same, to de-identify faces in images, thus rendering face detection impossible. Defaux and Ebrahimi [5] showed how a region of interest could be scrambled to hide private information.

3. PROPOSED WORK

3.1 Threat Model and Assumptions

Cloud-based video editing give host to a slew of security threats. The following entities are involved in the process: the owner of the video, video transmission channels, CDCs, users of the video. Here transmission channels and CDCs are assumed to be vulnerable to attacks and they are considered untrustworthy, while the owner and the user of the video can have full access to the video, and hence, are regarded as trustworthy.

With these vulnerabilities we need a solution that assumes the data is at risk as soon as it is not in the possession of the user. To achieve the best security possible, all CDCs should be unrelated and any k $(\leq n)$ must not collude. We assume that both the input and output will be in the mpeg compression format. For convenience and to maintain data integrity, all video editing operations should be nondestructive so not to change the input file. For simplicity we further assume that all operations will be performed on the entire video and not for selected frames.

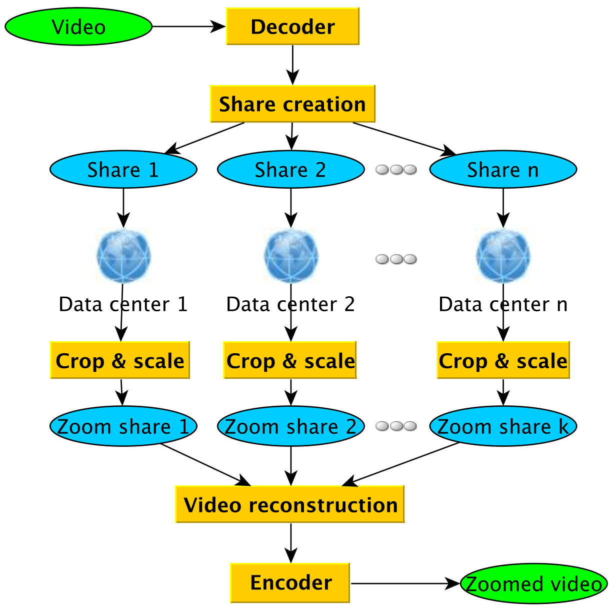

Figure 1: System architecture

3.2 Architecture

Figure 1 shows the architecture of our system. The secret video is inputted into the decoder, which gives us access to all the frames in the presentation order. Decoding is done by the FFmpeg decoder. With each frame, we input the frame, byte-by-byte to the SSS algorithm to create the shares. Once all the frames have been processed, and all shares created, the shadow copies are uploaded to the CDCs and the original secret video is deleted.

With shadow copies stored in CDCs they can now be processed. This is where shadow copies will be cropped and scaled. Shares are cropped and scaled on an individual basis. To reconstruct the secret video, edited or otherwise, k shares are downloaded and interpolated using $L(0)$ where x_i is the number of that particular share and y_i is the value of the share being used in reconstruction. Reconstruction leads to either the original secret being recovered, or the result of cropping and scaling of the shadow copies, but in plaintext.

3.3 System Implementation

We utilized FFmpeg‘s libav library for decoding and encoding. This also gave us access to the frame data, which we could then encrypt. We originally intended to use libav to re-encode each shadow copy into the same format as the secret video, but this proved difficult to achieve with codecs that use lossy compression techniques. As a result, by re-encoding a share in the same format as the original video we lose information thus making reconstruction impossible.

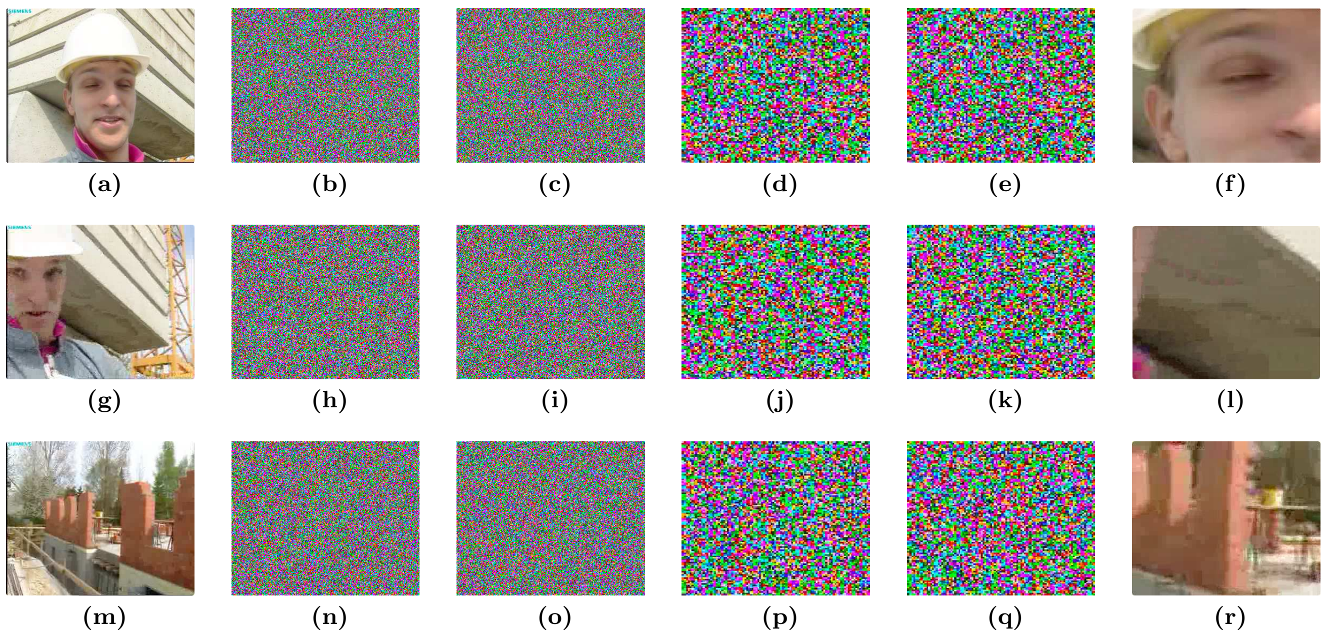

Figure 2: Cropping and scaling of the Foreman where (a) , (g) and (m) are the 1st, 190th and 300th frames, respectively, from the original video. (b), (c), (h), (i), (n) and (o) are the 1st and 2nd unedited shares for their respective frame. (d), (e), (j), (k), (p) and (q) are the 1st and 2nd cropped and scaled shadow copy for their respective frame (f), (l) and (r) show the reconstructed frames.

We could not reliably re-encode shares using lossy codecs so we opted to develop out own file format to store each share.

Our file format is reflects our naïve encryption of the video where all I, P and B frames are stored completely, and in presentation order. This is a less than ideal solution and gives cause for future improvements. As a result, we store a lot of redundant information for each frame, and as such the space required for each share is dramatically higher than that of the original video because we do not employ any compression techniques.

Furthermore we chose to store the header information in plaintext, while all the frame data is stored as ciphertext. The headers contain important information that it is needed to properly reconstruct the video. Even with headers in plaintext, the content of the video is still secure. An attacker does not gain any insight into the content of the secret video by knowing the headers. Our headers store timestamps, the resolution of the video, and the size of each frame in bytes.

Our system consists of three components, the video source, the n shareholders and the user, as illustrated by Fig. 1. A user would first create shares of the secret video before distributing all n ($n = 5$ in our case) shadow copies to its shareholders. Then cropping and scaling is done on all n shares. Any k ($k = 3$in our case) shareholders bring their shadow copies together to reconstruct the video.

After decoding the input video, we process all P, B and I frames in presentation order. All frames contain Y, Cb and Cr components with 4:2:0 sub-sampling. Each component is processed in its entirety and the decoder interleaves the components to produce a fully realized frame. Byte-by-byte we input each component into $F(x)$ such that a_0 is the current sub-pixel. When all three components are processed, we will have fully generated the shadow copies for the frame. Each shadow frame is now written to their respective shares, where all frames are stored in presentation order. This makes reconstruction easier.

Once the shadow videos have been created, they are uploaded to different cloud data centers, and the original video deleted. The cloud data centers does not possess the means to reconstruct the video, nor analyze the contents of videos.

To achieve zooming in the video, the user submits the dimensions for the RoI along with the starting coordinate of the region. The system proceeds to remove all data that lies outside the RoI for all frames. When the frame is fully cropped, the system scales the cropped frame to the same dimensions as the original frame. This is done by calculating $P(x, y)$ for each sub-pixel in the scaled output frame. Now that the RoI has been scaled the frame is written to a new file that stores the zoomed shadow copy.

To reconstruct the original video, 3 shareholders bring their shares together by downloading their shadow copy from their respective data centers, and inputting their share into the system. The system reads the frames from the shares, and inputs three current bytes into the Lagrange interpolation $L(x)$ where $x = 0$. The results from this is the reconstructed byte. After reconstructing a frame, it is re-encoded using FFmpeg‘s encoding functionality. With all frames reconstructed the original video has been retrieved.

4. ANALYSIS

As can be seen in Fig. 2 , our solution maintains visual security of the video. An attacker cannot infer the content of the video by viewing the shadow copy. Furthermore, because our system uses the SSS scheme, along with naïve video processing, we know that our system is information theoretically secure. Assuming that an adversary does not

	Average	Minimum	Maximum
Encryption	6.8646	6.029	7.829
Reconstruction	31.474875	5.419	57.53075

Table 1: Processing time in seconds

have access to at least k shadow copies, the adversary cannot infer the value of a specific byte in the frame data. If the adversary needs only 1 additional share to reconstruct the secret video, he will not be able to do so. In our system, if an attacker gains access to 2 shares, for any particular value there are $p-1$ possible values for the secret byte in the finite field $\mathcal{Z}p$.

An additional benefit of using SSS is that it is resistant to data tampering. SSS provides data integrity by making reconstruction of a secret impossible if the shadow value does not coincide with the original polynomial. Thus if the adversary modifies a share, the polynomial being reconstructed will be incorrect, resulting in an unintelligible video.

The data loss mentioned earlier does not produce dramatically different output as can be seen in Fig. 2. The data loss is only affected by the larger values a byte can have. Practically, this means that the maximum brightness of a pixel is somewhat reduced. As can be seen in Fig. 2 there is no significant reduction in video quality.

The proposed system suffer from major limitations in its design. By storing video shares in their entirety the storage requirements of our system is rather strict. foreman.yuv requires a total of 436 KB in its compressed MPEG form, whereas each shadow copy take up 47 MB. The storage requirements of the shadow copies depend on the resolution of the video as well as its length.Thus, long high resolution videos will require large amounts of free space. In our tests, a 01:58 minute video with 720x480 resolution require only 90 MB of storage in its compressed form, but approximately 2 GB for each of its shadow copies. The exact compressed-to-share storage ratio is difficult to pinpoint as it is a function of the video resolution, length and overall change from frame to frame. Videos with many new visual elements cannot be as efficiently compressed as scenes that introduce fewer elements.

Further limitations come from share generation, cropping, scaling and reconstruction. Because each byte in the video data must be considered a secret, any operation must be done on each individual byte. This means that there is a lot of processing overhead, especially with share creation, bicubic interpolation and Lagrange interpolation. If the user does not request scaling, the time to reconstruct the secret using Lagrange interpolation can be significantly reduced due to the smaller video resolution. There are fewer secrets in each frame to reconstruct. Table 1 shows the run time of our system using the Foreman as our secret. Share creation is a relatively quick process and is largely bound by decoding the input video. Reconstruction can take a lot longer because of Lagrange interpolation where we have to calculate $\frac{x-x_i}{x_j-x_i} \equiv (x-x_i)(x_j-x_i)^{-1} = 1 \mod p$ which can be computationally expensive.

5. CONCLUSION AND FUTURE WORK

As previously stated, our implementation of the encrypted videos uses a naïve way of representing video data. This leads to increased storage requirements, and as such a form of compression should be used to reduce this requirement. Adding additional video format properties can make for further improvements. This would include would include calculating the different between frames, so not to store redundant data. By applying Discrete Cosine Transform, motion vectors and possibly macroblocks, the storage requirements would decrease drastically. The benefit would be two-fold: i) reducing storage overhead, and ii) reducing processing overhead.

Implementing macrobocks for encrypted videos allow for additional benefits by making the video better suited to be streamed to a user. Currently our method is only working on a local machine, and as such extensions should be made to work on distributed systems. This would involve streaming encrypted frames to the user for reconstruction, whereupon zooming factors are provided and sent back to the servers, which then apply the operation and produce a zoomed video.

These major limitations prevent our system from being adopted in real world scenarios. Future work should attempt to identify a more efficient way of encrypting videos while maintaining its security.

6. REFERENCES

[1] S. Alharthi, P. K. Atrey, and M. S. Kankanhalli. Secret video sharing. In *Proc. of the APSIPA annual summit and conference*, 2010.

[2] J. C. Benaloh. Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret (Extended Abstract). In A. M. Odlyzko, editor, *Advances in Cryptology âĂŤ CRYPTOâĂŹ 86*, number 263 in Lecture Notes in Computer Science, pages 251–260. Springer Berlin Heidelberg, 1987.

[3] D. Bitouk, N. Kumar, S. Dhillon, P. Belhumeur, and S. K. Nayar. Face Swapping: Automatically Replacing Faces in Photographs. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, pages 39:1–39:8, New York, NY, USA, 2008. ACM.

[4] X. Chen, X. Huang, J. Li, J. Ma, W. Lou, and D. Wong. New Algorithms for Secure Outsourcing of Large-Scale Systems of Linear Equations. *IEEE Transactions on Information Forensics and Security*, 10(1):69–78, Jan. 2015.

[5] F. Dufaux and T. Ebrahimi. Scrambling for Privacy Protection in Video Surveillance Systems. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(8):1168–1174, Aug. 2008.

[6] W. Lu, A. Varna, and M. Wu. Secure video processing: Problems and challenges. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5856–5859, May 2011.

[7] M. Mohanty, W. T. Ooi, and P. Atrey. Scale me, crop me, know me not: Supporting scaling and cropping in secret image sharing. In *2013 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, July 2013.

[8] E. Newton, L. Sweeney, and B. Malin. Preserving privacy by de-identifying face images. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):232–243, Feb. 2005.

[9] N. Quang Minh Khiem, G. Ravindra, A. Carlier, and W. T. Ooi. Supporting Zoomable Video Streams with

Dynamic Region-of-interest Cropping. In *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems*, MMSys '10, pages 259–270, New York, NY, USA, 2010. ACM.

[10] P. Saini, A. Kumar, and N. Singh. Article: Fpga implementation of 2d and 3d image enhancement chip in hdl environment. *International Journal of Computer Applications*, 62(21):24–31, January 2013. Full text available.

[11] B. SchÃűlkopf, J. Platt, and T. Hofmann. *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*. MIT Press, 2007.