Efficient Privacy Preserving Video Surveillance

Maneesh Upmanyu, Anoop M. Namboodiri, Kannan Srinathan and C.V. Jawahar
International Institute of Information Technology
Gachibowli, Hyderabad, A.P. 500032, INDIA
{upmanyu@research., anoop@, srinathan@, jawahar@}iiit.ac.in

Abstract

Widespread use of surveillance cameras in offices and other business establishments, pose a significant threat to the privacy of the employees and visitors. The challenge of introducing privacy and security in such a practical surveillance system has been stifled by the enormous computational and communication overhead required by the solutions. In this paper, we propose an efficient framework to carry out privacy preserving surveillance. We split each frame into a set of random images. Each image by itself does not convey any meaningful information about the original frame, while collectively, they retain all the information. Our solution is derived from a secret sharing scheme based on the Chinese Remainder Theorem, suitably adapted to image data. Our method enables distributed secure processing and storage, while retaining the ability to reconstruct the original data in case of a legal requirement. The system installed in an office like environment can effectively detect and track people, or solve similar surveillance tasks. Our proposed paradigm is highly efficient compared to Secure Multiparty Computation, making privacy preserving surveillance, practical.

1. Introduction

Video surveillance is a critical tool for a variety of tasks such as law enforcement, personal safety, traffic control, resource planning, and security of assets, to name a few. However, the proliferation in the use of cameras for surveillance purposes has introduced severe concerns of privacy. Everyone is constantly being watched on the roads, offices, supermarkets, parking lots, airports, or any other commercial establishment. This raises concerns such as, watching you in your private moments, locating you at a specific place and time or with a person, spying on your everyday activities, or even implicitly controlling some of your actions. *Privacy preserving video surveillance* addresses these contrasting requirements of confidentiality and utility.

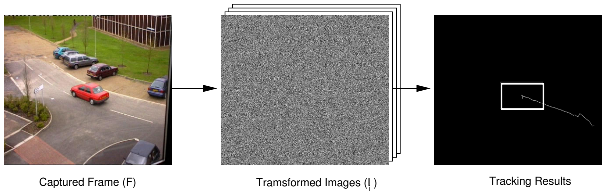


Figure 1. Privacy in surveillance: Tracking in obfuscated videos.

Ideally, one would like surveillance to be carried out on obfuscated videos, without revealing any personal information. For instance, in Figure 1 a frame, F, of the surveillance video is transformed into a set of seemingly random images, I_i, on which a tracking operation is successfully carried out. In this paper, we propose an elegant distributed framework to carryout provably private surveillance while significantly reducing the overheads of introducing privacy.

A practical and an efficient solution fits into the business model of 'surveillance as a service'. A company can monitor houses, streets, stores, etc and alert the clients of suspicious incidents, without intruding into their privacy. The client just installs a shatter-cam and sends the feeds for surveillance. Our proposed solution works well with current trends of computing/storing on remote server clouds.

Traditionally, confidentiality of the data is achieved through encryption. However, by definition, encryption destroys any structure present in the data, thus negating the ability to perform any meaningful video processing task. Realizing this, recent privacy preserving computer vision algorithms build on cryptographic protocols such as secure multiparty computation(SMC) [2, 12]. SMC uses interactions between multiple parties to achieve a specific task, while keeping everyone oblivious of others data.

Introducing privacy and security in visual data processing was attempted with considerable success in different domains. Blind vision [2, 3] allows someone to run their classifier on another person's data without revealing the algorithm or gaining knowledge of the data. Shashank *et al.* [12] exploited the clustered nature of image databases to improve the efficiency of SMC for example based image retrieval by processing multiple queries together.

Smart cameras [6] do surveillance in the camera itself or try to mask sensitive information in the videos. The former requires expensive programmable cameras and are restricted to single camera algorithms. Changing the algorithms is tedious and costly. The second approach addresses problem specific concerns in surveillance videos. Dufaux *et al.* [7] proposed a selective scrambling based solution to preserve privacy. Face swapping [5] and face de-identification [10, 13] try to modify face images such that they can be automatically detected, but cannot be correctly recognized. Boult *et al.* [6] uses programmable cameras to carry out detection and masking of the regions of interest. All the above approaches rely on the success of detection of interest regions and do not provide any guarantee of privacy. Moreover, the original video is lost in all.

We note that most of the current privacy preserving algorithms are based on the generic framework of *SMC* [2, 12, 15] which requires heavy communication to achieve secure computation. For instance, a single multiplication is carried out via complex distributed protocol involving *oblivious transfer (OT)* [8], which is a highly communication intensive subroutine in SMC. Factually, the round-trip time in a LAN is of the order of a few milliseconds, whereas several floating operations take no more than few nanoseconds. Clearly, these delays are too high, while dealing with voluminous data like surveillance videos. Hence, solutions based on SMC are impractical for our application.

In this work, we use the paradigm of *secret sharing* [11] to achieve private and efficient computation of surveillance algorithms. Secret sharing (SS) methods [4, 9, 11] try to split any data into multiple shares such that no share by itself has any useful information, but together, they retain all the information of the original data. However, the standard SS methods, which were invented to address secure storage of data, results in significant data expansion (each share is at least the size of the data). Computing on the shares [11] is inefficient as it would require some sort of SMC. In this work, we exploit certain desirable properties of visual data such as fixed range and insensitivity to data-scale, to achieve distributed, efficient and secure computation of surveillance algorithms in the above framework. Using this approach, one can do change detection on a typical surveillance video at 10 frames/second on generic hardware. Our approach also addresses the concerns related to video surveillance, presented in Table 1.

2. The Proposed Approach

The privacy of our surveillance system is based on splitting the information present in an image into multiple parts or shares. Each share is sent to an independent server for processing. The protocol in a nutshell is as follows:

1. The camera splits each captured frame, F, into k (> 2), shares using a shatter function: $\phi(F) = [I_1, I_2, \ldots, I_k]$. Each share is then sent to an independent computation server for processing. Note that no share by itself reveals any useful information about the original image.

a) Preserves Privacy: Ensure that the person doing surveillance learns only the final output, and nothing else.
b) Computationally Efficient: The image representation should allow efficient computations so as to realize surveillance algorithms in the encrypted domain itself. Moreover, the encoding process at the sensor should be light weight.
c) Efficient to Transmit: The encoding process should not blow up the size of the video data.
d) Secure Storage: One should be able to store the surveillance data in a secure fashion, so that breaking into a storage server do not compromise the privacy of the data.
e) Addresses Legal Issues: For legal or investigative purposes, someone with authorization should be able to recover the plain video without the client's help.

Table 1. Mandatory and desirable characteristics of a secure, privacy-preserving surveillance system.

2. To carry out a basic operation f on the input image, each computation server blindly carries out the equivalent basic operations, f' on its own share. This is equivalent to the corresponding basic operation being carried out on the original image: $f'(I_j) \equiv \phi(f(F)))$.

3. The results of operations on the shares are then integrated by the observer using a merge function, to obtain the final result: $f(F) = \mu(f'(I_1), f'(I_2), \ldots, f'(I_k))$.

Figure 2 shows a schematic diagram of the complete process. The privacy of the overall system relies on the fact that neither the independent shares, nor the results of computations on them, reveal any information about the original image. The integration of results from the independent servers reveal only the final result of the algorithm to the observer.

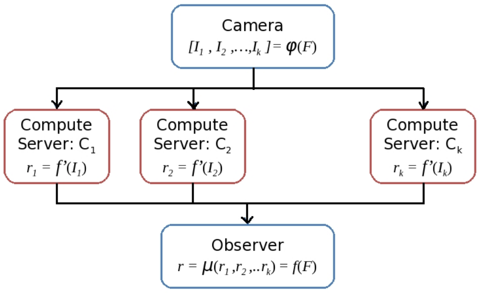


Figure 2. Secure computation of $f(d)$ by a set of compute servers.

In order to analyze the security and privacy of the system, we will formalize the notion of security as follows:

1. *Information Revealed:* We use the term information in the strictest information theoretic sense. That is, an observable quantity I is said not to reveal any information about another quantity F (in our case, the original image), if: $\forall_a Pr(F = a|I)$ is same as $Pr(F = a)$.

2. *Preservation of Privacy:* A surveillance system is said to preserve privacy, if it reveals nothing more than the final output of the surveillance algorithm to any party in the system, outside the camera.

3. *Security Model:* The servers are assumed to be *honest, but curious* in nature. i.e., they will carry out the expected computations faithfully, but may try to learn about the original image from their view of the data. They are independent in the sense that they will not collude to extract any additional information.

The functions $\phi()$ and $\mu()$ that form the basis of our protocol are adapted from the popular secret sharing scheme using the *Chinese Remainder Theorem* (CRT) [1, 9].

2.1. The Modular Transformations: $\phi()$ and $\mu()$

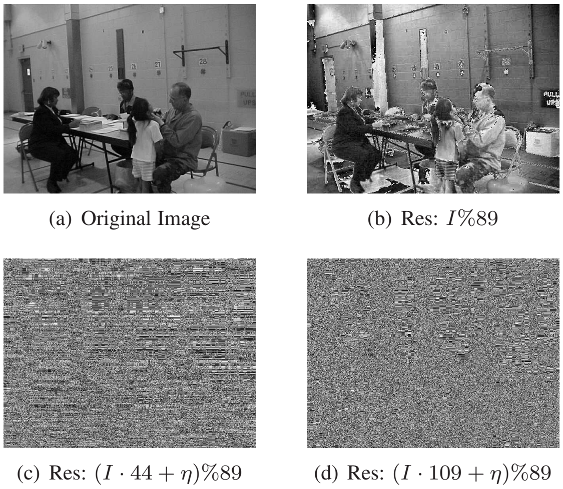
In our problem, we secure each pixel, d of an image, F, independently using a pixel level shatter: $\phi_p()$. The direct CRT based transformation would compute each share as $d\ mod\ p_i$ for different primes (p_i). However, given the correlation between the neighboring pixels in an image, the modulo remainder reveals significant information about the secret (image) (see Figure 3(b)). To overcome this, we introduce the following modification to obtain the shatter function $\phi_p()$:

$$d_i = \phi_p(d, p_i) = (d \cdot s + \eta) \bmod p_i, \qquad (1)$$

where d is a single pixel in the image, s is a constant, positive scale factor, and η is a uniform random number: $U(0, r_{max}), r_{max} \leq s$. Note that the first part of $\phi_p()$, effectively makes the LSBs of the resulting number, random. For example, if $s = 2^k$ and $r_{max} = s$, then k random bits are appended to the right of d. Intuitively, if $p_i < s$, then d_i would essentially be random, and would not reveal any information about d (see Sec.2.4 for analysis).

Given, $d_i = \phi_p(d, p_i)$ for different relatively-prime p_is, the secret d can be recovered by CRT by solving a system of congruence. The solution is unique if all the intermediate values $(d_i, f'(d_i))$ are less than the product of the primes ($p_i s$). Note that η, which was randomly chosen for each pixel is not used for recovering the secret. The CRT hence forms our recovery transformation $\mu_p()$, at the pixel level.

To shatter an image, we apply the above transformation to each pixel in the image independently, while keeping the set of p_is and s constant. However, we vary the random number, η for every pixel and every image that we encode, and hence the result of modular division is essentially random. Note that without scaling and randomization, the modulo image will reveal considerable amount of information about the original image (see Figure 3(b)). Choosing an arbitrary range of randomization results in partially secure shares (see Figure 3(c)). As explained in section 2.3, it is possible to choose a range for η that will generate secret shares that are completely obfuscated (see Figure 3(d)). In this example, the second-order entropy of image 3(d) is equivalent to the entropy of a pure random noise image.


(a) Original Image

(b) Res: $I\%89$

(c) Res: $(I \cdot 44 + \eta)\%89$

(d) Res: $(I \cdot 109 + \eta)\%89$

Figure 3. Data Obfuscation: Information retained in the residue image for various scale factors and corresponding η distributions.

The standard CRT based secret sharing scheme by itself is not ideal for our application as it leads to heavy communication overheads. Each pixel share is of size $|p_i|$ bits, the total share size is therefore given by $\sum_{i=1}^{i<l} |p_i| > l.|p_o|$, where l is the number of servers/shares. Choosing an optimal p_o becomes crucial since it determines both the range of numbers we can correctly represent as well as the total size of the shares. Our solution, based on a SS scheme designed for visual data, reduces the data expansion by at least a factor of l. Such reduction is prominent for huge data such as live-video feeds, while ensuring acceptable levels of security. In the process, we also utilize the homomorphic properties of the modular domain to circumvent SMC, thus gaining on efficiency.

2.2. Computing on the Shares

The operations of addition and multiplication are well defined in modular arithmetic, hence making the transformed data appropriate for computations. For example, if f is defined as: $f(x, y) = x + y$, then one can compute $x_i + y_i$ at each compute server and recover $x + y$ at the observer using CRT. In other words, modular arithmetic is homomorphic to both addition and multiplication, within the modulo base. i.e.,

$$\begin{aligned} (a+b) \bmod p &= (\ (a \bmod p)\ +\ (b \bmod p)\ ) \bmod p \\ (a \cdot b) \bmod p &= (\ (a \bmod p)\ \cdot\ (b \bmod p)\ ) \bmod p \end{aligned}$$

As mentioned before, given the value of the rhs of the above equations for multiple values of p, one can exactly recover $(a + b)$ or $(a \cdot b)$ using CRT.

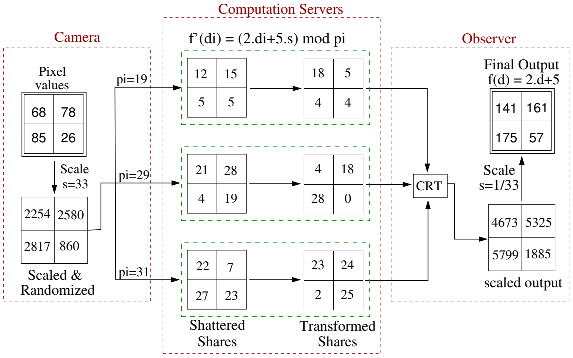


Figure 4. Example: Affine intensity transform in modular domain.

To clarify the process, let us consider the processing of the image patch shown in Figure 4. As each pixel is processed independently, consider the pixel with value 68. Let the scale factor s be 33 and the random number η, be 10, the resulting $d \cdot s + \eta$ would be 2254. If the image is shattered into three shares, with primes 19, 29, and 31, the corresponding shattered shares would be: $\{12, 21, 22\}$.

If the algorithm applies the affine transformation $f(d)$ = $2 \cdot d + 5$ to each pixel d, then each compute server will carry out $f'(d_i)$ = $(2 \cdot d_i + 5 \cdot s)\%p_i$ independently on its share, and obtain $\{18, 4, 23\}$. The observer will integrate these numbers using CRT to obtain 4673, which when divided by the scale factor 33 gives 141 $(2 \cdot 68 + 5 = 141)$.

The above computations are valid only for integer values of operands and results. As the pixel values of an image has a limited and finite range, these conditions are easily satisfied. Moreover, any non-integer operation can be correctly simulated by appropriately scaling the entire data/algorithm before the shattering operation. One can exactly reconstruct the ideal output by scaling down the results if all the intermediate values are correctly represented in the Residue Number System, i.e., the product of the primes p_i is greater than any intermediate value in the entire computation.

2.3. Selection of Primes and Scaling Factor

Given the value of k, we select $p_1, \ldots, p_k$ such that their product, P, is greater than any intermediate value in the algorithm. One could choose the smallest k consecutive primes satisfying the above property. The scaling factor, s, should be higher than the largest prime, to obfuscate each share (see Figure 3). The range of the intermediate values is a function of the scale factor. So the process may have to be iterated a couple of times during the design phase.

Let M denote the maximum intermediate value that is to be represented in the surveillance algorithm, when run in the plain domain. Assume that we require a scale factor of s to achieve complete privacy. Let c be the constant such that $M.s^c$ is the maximum intermediate value to be represented after scaling. Note that c depends on the algorithm and is usually small (≈ 2) in most vision algorithms. For example, if the operations being performed are linear then c is 1, for quadratic functions c would be 2 and so on. In other words, the primes $p_1, \ldots, p_k$ are chosen such that:

$$s \geq \max_i p_i, \text{ and } s < \left(\frac{\prod_i p_i}{M} \right)^{\frac{1}{c}} \tag{2}$$

Simplifying the above, we find that, if $M < (\max_i p_i)^{k-c}$, then the original image is hidden from the individual servers. At the same time, we can guarantee that the reconstruction by CRT is unique. The above inequality is a sufficient condition and our experiments show that it is usually not necessary. We now take a closer look at the amount and nature of information that is revealed by the residue image and the related privacy concerns.

2.4. Analysis of Privacy

As noted before, the privacy of the system is based on splitting the information present in an image into multiple shares. The parameters used for shattering i.e., the primes p_i and scale factor s are constant for each shattering operation and are in general assumed to be public. The only possible information leakage of the secret is that retained by each share. We now analytically show that with an optimal parameter selection, the information retained by a share is negligible. Consider a pixel with value $d \in [0, 255]$, which is scaled by s, followed by addition of a random number, η. Note that, if η follows the uniform distribution, $U(0, r_{max} - 1)$, the distribution of $r_i = \eta\%p_i$ would be:

$$Pr(r_i = x) = \begin{cases} (k+1)/r_{max}, & x < r_{max}\%p_i \\ k/r_{max}, & otherwise, \end{cases} \tag{3}$$

where k is $\lfloor r_{max}/p_i \rfloor$. The above distribution will be uniform, leading to perfect security if r_{max} is a multiple of p_i. On the other hand, if r_{max} is not a multiple of p_i, there is a slight step of size $1/r_{max}$ (see Equation 3) in the resulting distribution at $x_k = r_{max} \ mod \ p_i$, where r_{max} known only to the camera. That is, for a particular pixel value d, the share d_i, is marginally more likely to be within a certain range of pixel values. Taking the most paranoid point of view, we assume that in surveillance, a large number of frames are likely to be identical, and one might learn this distribution, over an extremely long period of time. This is statistically impossible if we choose a sufficiently large k, and we can choose a value of k to make the step arbitrarily small. Even in this case what someone would learn is: 'the background is slightly more likely to be dark than bright'.

In short, we see that the method of shattering is statistically impossible to break for images, and unlike encryption based methods, even if it is broken, nothing useful can be learned from the information that is revealed. The advantage of our method is that such a high level of security can be achieved, while allowing computations to be carried out efficiently on the shattered videos.

2.5. Implementation Challenges

As described above, carrying out integer addition and multiplication in the modulo domain is relatively straight forward. We might feel that one can achieve any operation that can be modeled as a combinatorial network of *AND* and *XOR* gates, which makes it equal in power to a general purpose computer. However, there are many practical challenges to be overcome to implement the functionalities.

Modulo arithmetic is carried out using positive integers only. Hence one has to map the range of numbers used in an algorithm to an appropriate range of positive integers. Signed numbers in the residue form are represented with an implicit sign. If the range of numbers used is (0, M), we used the numbers in the range (0, M/2) to represent positive numbers, and for the remaining numbers it is negative. The change in sign of $|Z|_M$ is performed by the operation of additive inversion of Z, i.e. -Z = M - Z, which is equivalent to $(m_1 - z_1, m_2 - z_2, ...m_k - z_k)$.

The employed RNS, correctly represents integers in the range (-M/2, M/2). Use of any number beyond this range will result in errors in the recovered results using CRT, which we refer to as overflow or underflow. Overflow and underflow are safely avoided by working in a domain large enough to correctly represent all intermediate values encountered. However, the net data size of the shattered video streams is directly proportional to the domain-size that we work with. An efficient domain can be chosen by precomputing the upper bound on the possible intermediate values, and then appropriately deciding on the RNS.

Operations such as divisions and thresholding are difficult to achieve in RNS. Division of an integer A by B is defined as $A/B = (a_i.{b_i}^{-1})$ mod m_i in the RNS. This is valid if B is co-prime with M and B divides A. For this to always hold, one would have to take into account B, in choosing the RNS. Though this looks practical (since the original algorithm is known beforehand), it might not always be efficient since the shattered data size (# of bits) is directly proportional to the chosen domain-size (consider the case where division has to be performed for multiple divisors. Validating the division for every divisor can sometimes result in choosing a larger, than required, domain size, thus affecting the efficiency).

An alternate solution for division and thresholding can be designed using an additional untrusted server. Every independent computation server (ICS) sends over their respective residues to the additional server, where the merge function is applied and the division/comparison is performed in plain domain. However, simply doing this would end up revealing the intermediate results to the additional server. To secure against such information leakage, every ICS does a reversible randomization of their respective residues before sending them over to the additional server. The server does the division/comparison on the randomized data(post merging) it received. The computed result is shattered by it and sent back to the respective ICS where it is de-randomized to retrieve the actual(expected) modulo result. Note that the randomization should be done in a way, so as to avoid possible factorization and GCD based attacks.

3. Experimental Results and Discussions

We now provide a detailed account of the implementation and analysis of a common surveillance task, tracking of moving objects, using the proposed framework. We describe the mapping of this problem to the framework and show the steps involved in carrying out the computations. We also describe in brief, the results of face detection in the proposed framework. The process of tracking is carried out in two steps, that of change detection by background subtraction, followed by tracking of points of change.

Background Image Subtraction: We first consider the problem of background removal. Specifically, our problem is: given a static background image (in shattered form), subtract it from each captured image, such that at any point of time, the original image or the background image is not revealed to anyone. At the end of the protocol, all that is learned by the observer is the final output (the difference image). The complete process can be sub-divided into:

1: Deciding the RNS: Every pixel in the image has a value in the range $[0 - 255]$. In background subtraction, the range of numbers in the result is $Y = [-255, 255]$, or $Y = [0, 511]$ (using an implicit sign). Shattering involves scaling every pixel by s. Therefore in the RNS we need to correctly represent the range $R = [0, s.Y]$. The optimal number of servers and the prime numbers defining the RNS is chosen as described in section 2.3. In our example, we have number of servers $k = 3$, the scale factor $s = 33$, and the primes are $19, 29$, and 31. These set of parameters form our RNS, which is made public.

2: Defining the Algorithm: Next step is to map the original algorithm into modulo domain. In our case it is pixelwise subtraction of a fixed background image (B) from the captured image frame (F), both in the shattered form. As the only operation that needs to be performed is subtraction between two scaled quantities, the equivalent operation in the modulo domain would be the subtraction of the corresponding shattered pixel values as described in section 2.2.

3: Capture Image and Shatter: The image captured by the camera (Figure 5(a)) is shattered using the modular transformation described in section 2.1. In our example, the parameters used in the shatter function are the ones as obtained above. Input frame F is scaled by $s = 33$ and its lower order bits randomized to obtain an image I'. The image I' is then shattered using the primes $p_i s$ and the shattered shares are sent to the corresponding servers.

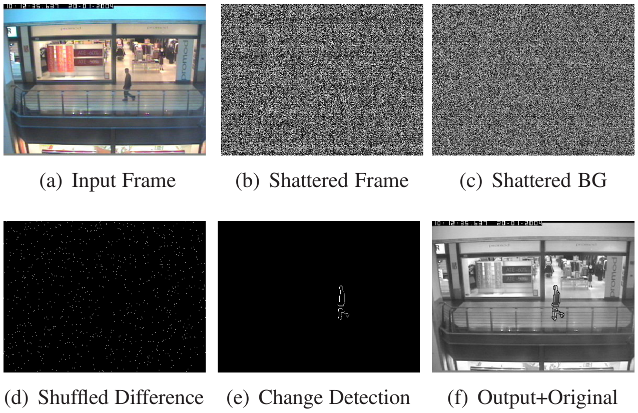
4: Apply the algorithm on shattered components: The shattered components received by the servers are now independently processed at each of the servers. As defined in step 2, the background image B' is subtracted from the each of the input frames. At the server i, the arithmetic operations are done modulo prime p_i. For example if the difference of two pixels is computed as $D = 100$, and the corresponding prime for the server is $p_i = 29$, then the difference is stored as $100\%29 = 13$.

5: Merge the outputs at the observer: The computed results are sent over to the observer who uses the merge function (see section 2.1) to obtain the final output. In our example, the observer obtains the 3 shattered images. Now the observer uses the Chinese remainder theorem(CRT) to reconstruct every pixel of the output image from the corresponding pixel values of the shattered images it receives. For example if the components of a particular pixel after subtraction are $\{12, 0, 11\}$ corresponding to the primes $\{19, 29, 31\}$, CRT would reconstruct -1508 from these values. The result is then scaled down by the initial scale factor 33 to obtain the final result as -45.

Change Detection: The detection of change involves subtraction of a frame from a background frame, which is carried out as explained before. We also update the background image by replacing pixels in the background where change is detected with the corresponding ones from the foreground. This is done directly in the RNS. The difference values are integrated by a thresholding server, using CRT, which then compares the result against the pre-defined threshold to detect motion.

However, sending the shattered differences to a threshold server reveals the difference image. To avoid this, we apply a reversible pixel shuffle that would remove any structure in the image, before sending it to the threshold server as explained in section 2.5.

The additional (untrusted) server thresholds the received image to get a shuffled binary image. This is sent back to each of the ICS, where it is de-shuffled to obtained the final binary image. As the result is now in plain domain, one can also apply any post-processing operations such as erosion, merging, etc. to remove any noise. Moreover, the background learning can work in the transformed domain as the pixels with no change are known to the compute servers in each frame. We also note that the accuracy of the algorithm is not affected by the obfuscation process, as indicated by the comparison with plain domain result. Figure 5 shows a sample frame that is being processed in the framework.


(a) Input Frame (b) Shattered Frame (c) Shattered BG

(d) Shuffled Difference (e) Change Detection (f) Output+Original

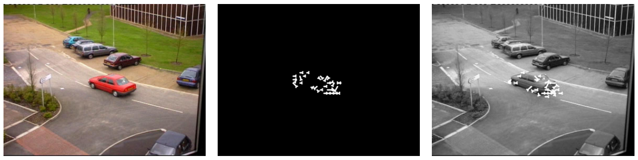
Figure 5. Change Detection: (b,c) are the shattered shares seen by one of the compute servers, (d) is the obfuscated difference image obtained after a pixel shuffle, (e) is the output as available with the observer, and (f) comparison with original image.

Table 2 shows the exact time (in seconds) spent by the individual compute servers as well as the thresholding server (usually the observer). We note that even with a non-optimized implementation on a desktop class machine, one can achieve a computation speed of upto 14 (QVGA) frames per second at each server. The total data to be transmitted in the process (in Kilobytes) and the corresponding communication time, assuming a 100Mbps connection between the two servers, are also given. We note that most operations are carried out in sub-second times.

Image	# of	Comp. Time		Commn.	
Resol.	Servers	Serv.	Merge	Data	Time
PITS'00 768x576	3	0.367	1.294	324	0.025
	5	0.362	1.433	270	0.017
	7	0.377	1.316	162	0.013
CAVIAR 384x288	3	0.110	0.292	81	0.006
	5	0.122	0.310	67.5	0.005
	7	0.137	0.338	40.5	0.003
Towers 320x240	3	0.071	0.189	56.25	0.004
	5	0.074	0.201	46.87	0.004
	7	0.073	0.217	28.12	0.002

Table 2. Avg. computation and communication times for change detection.

Optical Flow and Tracking: The above algorithm is extended to compute the optical flow. We use the change detection results as guidelines, and compare a patch around each motion pixel against its neighbors. The comparison is done using correlation, which is similar to the affine transformation operation described in example in Figure 4. The optical flow estimates thus derived, along with the motion segmentation results from the previous section can be used to build a complete system that detects and tracks people/objects. Figure 6 shows the results of optical flow being computed from pairs of images. The shattered images available with the computation servers are omitted here due to space constraints, and they look very similar to those in the previous experiment. Once again, the help of an additional server is used in finding the maximal correlation windows after a shuffle, and the complete task is carried out without revealing any additional information about the image. One such result of tracking is shown in Figure 1.



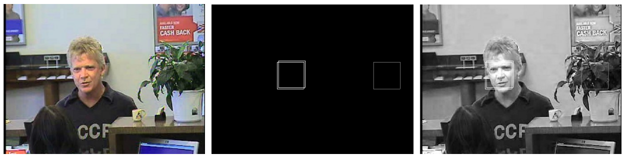
(a) Original Frame (b) Tracking (c) Tracking in Plain

Figure 6. Computation of Optical Flow: (a) Frame from input sequence, (b) optical flow computed, and (d) results superimposed on original frame. Note that the shattered images are omitted here.

Face Detection: For the next experiment we implement a more complex classifier, the popular face detection algorithm by Viola and Jones(VJ) [14] that uses a cascade of classifiers. Each classifier needs a set of Haar-like features. Note that all the features can be computed by addition or subtraction of pixel values within a rectangular neighborhood, which can be implemented directly in the RNS.

Note that the classifiers are trained on plain images and applied on the shattered ones. In a nutshell, VJ adopts a rejection cascade, where every image-window is passed through the cascade to detect faces. An integral image representation(which is summation of pixels) is computed for the input image. For every stage of the cascade, the rectangular features are computed from the integral image (this involves addition and subtraction operations). The computed feature values are securely merged and compared against the cascade threshold to decide upon the acceptance/rejection of the window.

Considering that the only operations involved in VJ are addition, subtraction and thresholding, it is fairly straight forward to define the equivalent functions for modular domain. Each computation server computes the integral representation of its own secret share (this is equivalent to 'shatter' of the integral image computed in plain domain). The cascade (which is trained in plain domain) is then applied independently at each server. Every window of the image is then passed through the cascade, for which the feature value is computed for every weak classifier. In our setup, every server can independently compute its share of the feature value (computed from the integral representation on its secret share). Thresholding is done (as described before) with the help of a pixel shuffle and an additional server. The location of the windows that pass through the complete cascade are made known to the observer as final output. Figure 7 shows the result of face detection on an input image as obtained by the observer as well as the plain domain result for comparison. Once again we note that the outputs are identical. One can also notice that the plain image as well as the feature values computed are hidden from all parties involved, thus securing against any possible information leakage, and in the process only knowledge gained by the observer is the final output.



(a) (b) (c)

Figure 7. Face detection (a): Captured input image, (b): Result as received by observer, and (c) Detection result, if run on the plain image. The detected faces are shown in white boxes and the current window being processed is shown in gray.

Table 3 shows the amount of communication between the compute servers and the observer with varying number of servers and image sizes (in kilobytes). The data can be communicated over high speed connections to the observer for thresholding.

Image Resolution	Parallelization				
	3	5	7	10	12
200x 200	463.0	289.4	231.5	173.6	173.6
320x240	994.8	621.7	497.4	373.0	373.0
400x320	1777.1	1110.7	888.5	666.4	666.4
512x512	3908.4	2442.7	1954.2	1465.6	1465.6

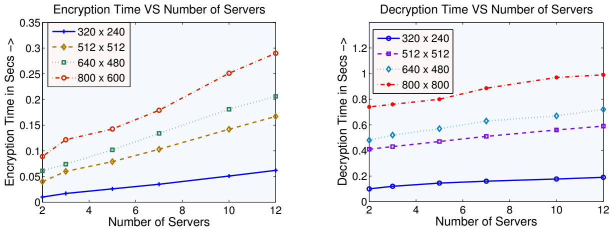
Table 3. Data transferred between thresholder and servers (in KB).

Overheads of Parallelization: To estimate the effects of encoding in terms of computation and communication overheads as well as accuracy, we study three different aspects. In the first experiment, we compute the average number of bits in an encoded frame and the total image size. An interesting observation from Table 4 is that as the number of compute servers (or primes) increases, the average bits in the resultant image first decreases, and then increases. The increase in the later part is due to the need of using larger primes, which drives up the size of the resulting image. One can always choose an optimal number of servers, as already explained in section 2.3.

			Avg Data Size	
# primes	Scale	Avg bits	Size/Frame	Total Size
3	17	6	56.25	168.75
4	31	5	46.87	187.48
5	19	4	37.50	187.50
10	13	3	28.13	281.30
20	11	5	46.87	937.40
50	31	6	56.25	2812.5

Table 4. Avg. data size (without compression) vs. parallelization.

Figure 8 shows the time required for each shattering and merging operation for a frame. We note that the time required for merging is considerably higher due to the use of large-number arithmetic when dealing with scaled numbers. Even then, the system is able to do these operations in well under a second.



(a) Shattering Time (b) Merging Time

Figure 8. Time required to shatter/merge a frame with increasing number of servers.

Comparison with SMC: Clearly, any secure solution needs more processing over an insecure one (eg. http vs https). For an application, if overheads are within acceptable limits, a provably secure method is always preferred. Our method compares with crypto solutions (SMC based) in terms of privacy, which are extremely inefficient. Our main achievement is an approach that achieves information theoretic privacy, while being extremely efficient over SMC. Our cameras can be inexpensive as the in-camera operations are simple and fixed. For example, VJ based FD on a QVGA frame requires evaluation of 92636 windows ($W_o = 24$, $H_o = 24$, $S = 1.25$). As per [2], each window takes a few seconds for a non-face and several minutes for a face. Even at 2 secs per window, this translates to 185272 secs (51 Hrs) per frame, which we reduce to 2-3 secs per frame. The processing time can further be improved by processing several windows simultaneously.

Data Transformation and Accuracy: Image encoding was achieved by taking the residue images of a noisy version of the scaled input image. To study the quality of the restored image, we conducted an experiment to encode a set of varied images over a range of parameters and computed the PSNR scores of the recomputed image and is shown in Table 5. We see that all the images have PSNR in the fifties. Note that for image compression purposes, a PSNR value of above 35 is considered very good, and our transformation is practically loss-less.

Image Resolution	Scaling Factor			
	11	31	80	120
320×240	51.552	51.309	51.138	51.134
512×512	51.598	51.345	51.176	51.161
640×480	51.568	51.301	51.141	51.138
800×600	51.567	51.307	51.142	51.134

Table 5. Peak Signal-to-Noise Ratio(PSNR), for k = 5.

As the images are represented faithfully by the transformation, and the algorithms are exactly mapped from the plain domain, the performance of the algorithms in the proposed framework would be the same as that of their plain domain equivalents. Furthermore, we note that the noise η, that we add to a scaled pixel is conditioned to be always less than the scale factor s. This is equivalent to adding a noise of less than 1 unit to the original image. This noise is often far less than that present naturally in surveillance videos and does not affect the results of the algorithms.

4. Conclusions

We have presented an efficient, practical and highly secure framework for implementing visual surveillance on untrusted remote computers. To achieve this we demonstrate that the properties of visual data can be exploited to break the bottleneck of computational and communication overheads. The issues in practical implementation of certain algorithms including change detection, optical flow, and face detection are addressed. This work opens up a new avenue for practical and provably secure implementations of vision algorithms, that are based on distribution of data over multiple computers.

References

[1] C. Asmuth and J. Bloom. A modular approach to key safeguarding. *IEEE Transactions on Information Theory*, 29:208–210, 1983.

[2] S. Avidan and M. Butman. Blind vision. In *ECCV*, pages 1–13, 2006.

[3] S. Avidan and M. Butman. Efficient methods for privacy preserving face detection. In *NIPS*, pages 57–64, 2006.

[4] J. C. Benaloh. Secret sharing homomorphisms: keeping shares of a secret secret. *CRYPTO*, 283:251–260, 1986.

[5] D. Bitouk, N. Kumar, S. Dhillon, P. Belhumeur, and S. K. Nayar. Face swapping: automatically replacing faces in photographs. *ACM Trans. Graph.*, 27(3):1–8, 2008.

[6] A. Chattopadhyay and T. Boult. Privacycam: a privacy preserving camera using uclinux on the blackfin dsp. *CVPR*, pages 1–8, June 2007.

[7] F. Dufaux and T. Ebrahimi. Scrambling for privacy protection in video surveillance systems. *Circuits and Systems for Video Tech., IEEE Trans.*, 18(8):1168–1174, Aug. 2008.

[8] O. Goldreich. *The Foundations of Cryptography - Volume 2*. Cambridge University Press, May 2004.

[9] M. Mignotte. How to share a secret. *CRYPTO*, 1983.

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

[11] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[12] J. Shashank, P. Kowshik, K. Srinathan, and C. V. Jawahar. Private content based image retrieval. In *CVPR*, 2008.

[13] T. Spindler, C. Wartmann, L. Hovestadt, D. Roth, L. VanGool, and A. Steffen. Privacy in video surveilled spaces. *Journal of Computer Security*, 16(2):199–222, Jan. 2008.

[14] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *CVPR*, 1:511–518, 2001.

[15] L. Wan, W. K. Ng, S. Han, and V. Lee. Privacy-preservation for gradient descent methods. In *KDD*, pages 775–783, San Jose, CA, Aug. 2007.