HEBGS: Homomorphic Encryption-based Background Subtraction Using a Fast-Converging Numerical Method

Justin Shyi and Sunwoong Kim
Division of Engineering and Mathematics, University of Washington, Bothell, WA 98011, USA
Email: {shyi123, sunwoong}@uw.edu

***Abstract*—Recent advances in cloud services provide greater computing ability to edge devices on cyber-physical systems (CPS) and internet of things (IoT) but cause security issues in cloud servers and networks. This paper applies homomorphic encryption (HE) to background subtraction (BGS) in CPS/IoT. Cheon *et al.*'s numerical methods are adopted to implement the non-linear functions of BGS in the HE domain. In particular, square- and square root-based HE-based BGS (HEBGS) designs are proposed for the input condition of the numerical comparison operation. In addition, a fast-converging method is proposed so that the numerical comparison operation outputs more accurate results with lower iterations. Although the outer loop of the numerical comparison operation is removed, the proposed square-based HEBGS with the fast-converging method shows an average peak signal-to-noise ratio value of 20dB and an average structural similarity index measure value of 0.89 compared to the non-HE-based conventional BGS. On a PC, the execution time of the proposed design for each 128$\times$128-sized frame is 0.34 seconds.**

***Index Terms*—homomorphic encryption, background subtraction, approximation, cyber-physical systems, internet of things**

I. Introduction

Recently, the use of cyber-physical systems (CPS) and internet of things (IoT) has increased significantly [1]. The microcontroller performance and memory size of edge devices are often insufficient to handle real-time processing and store large amounts of data generated by multiple sensors. Users have turned to third-party cloud services to meet their needs, but when the data is submitted to those entities, the information ultimately leaves their hands. With or without malicious intent, information in the cloud and network is constantly vulnerable. Thus, naturally, the first step in protecting sensitive information is to encrypt it before sending it to the cloud.

Through regular encryption, users of cloud services secure an additional layer of protection but miss out on one of the greatest tools the cloud services provide: cloud computing. By using homomorphic encryption (HE), which performs operations on a ciphertext (*ct*) without decryption, users gain the protection of encryption while taking advantage of the vast computing infrastructure built in those third-party services [2].

This paper moves background subtraction (BGS), which is a real-world application frequently used in CPS/IoT, to the HE domain. The prior work of Akkaya *et al.* applies HE to BGS for drone-based CPS [3]. However, BGS is performed on an unencrypted video frame, and HE is then applied separately to the generated foreground and background frames. After transmission, the server restores the original frame by homomorphically adding the encrypted foreground and background frames and analyzes it.

Unlike the previous work, our proposed work performs BGS in the HE domain. In particular, numerical methods for non-linear operations in BGS are adopted. To reduce the depth and execution time in our HE-based BGS (HEBGS), while minimizing the degradation of accuracy (or even improving the accuracy), a fast-converging method for the numerical comparison operation is proposed.

II. Background

A. Background Subtraction

BGS generates foreground masks by subtracting input frames from a static camera by a background model [4], [5]. The formula of the conventional BGS technique is as follows:

$$R(x,y) = \begin{cases} \text{foreground}, & \text{if } |I(x,y) - B(x,y)| \geq T, \\ \text{background}, & \text{otherwise.} \end{cases} \quad (1)$$

where $R(x,y)$, [1]$I(x,y)$, and $B(x,y)$ stand for the resulting mask, input frame, and background model at the coordinate (x,y), respectively. T is a threshold and a given constant. If the absolute value of the difference from a background is greater than or equal to a threshold value, an input pixel is classified as a foreground. Otherwise, it is classified as a background.

B. Homomorphic Encryption and Basic Operations

Popular HE schemes, such as BGV/BFV for integer plaintext (*pt*) messages [6]–[8] and CKKS for real number *pt* messages [9], provide the following operations [2]:

- Key generation: generates secret keys, public keys, and relinearization keys.
- Encryption: converts *pt* into *ct* using public keys.
- Decryption: converts *ct* into *pt* using secret keys.
- Arithmetic operations: *ct*-*ct* addition/subtraction/multiplication and *ct*-*pt* addition/subtraction/multiplication.

In addition to these operations, a packing encoding technique is supported. It packs multiple *pt* messages into a single *ct* and enables single instruction multiple data operations.

[1] $I(x,y)$ and $B(x,y)$ are simply denoted by I and B in the rest of this paper.

Algorithm 1 Comp$(a, b; n_c, d_c)$ [12]

```
Input: $a, b \in [0, 1]$, $n_c, d_c \in \mathbb{N}$
Output: a value between 0 and 1 (1 if $a > b$; 0 if $a < b$)
1: $w \leftarrow a - b$
2: for ($i = 1$; $i \leq d_c$; $i = i + 1$) do
3:     $w \leftarrow f_{n_c}(w)$
4: end for
5: return  $(w + 1)/2$
```

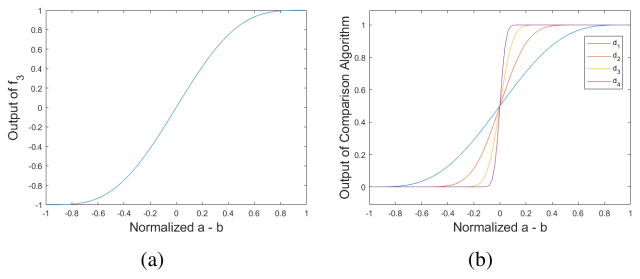

Fig. 1. Graphs of the approximate step function and Comp operation. (a) f_3 results (b) Comp results with n_c of 3 and d_c from 1 to 4.

Among the homomorphic arithmetic operations, *ct-ct* multiplication has special features. First, after each multiplication, a relinearization step is performed with relinearization keys to keep the *ct* size. Second, each multiplication significantly increases the level of noise used to hide *pt* messages in *ct*. There is a limit to the number of consecutive multiplications to obtain the desired value after decryption, which is called the (multiplicative circuit) *depth*. Lastly, *ct-ct* multiplication is very slow compared to other homomorphic operations, so custom hardware accelerators are sometimes used [10], [11].

C. Numerical Methods for Homomophically Encrypted Data

One of the most used operations in real-world applications is a comparison operation. Since only addition, subtraction, and multiplication are supported in the HE domain, a comparison operation is numerically performed [12], [13]. Algorithm 1 shows the numerical comparison algorithm. This algorithm includes the step function generating approximate -1 and 1. The formula of this approximate step function is as follows:

$$f_{n_c}(w) = \sum_{j=0}^{n_c} \frac{1}{4^j} \cdot \binom{2j}{j} \cdot w(1 - w^2)^j. \tag{2}$$

In this algorithm, the total number of iterations is $d_c \times n_c$. If these iterations are performed in a binary tree fashion, the total number of consecutive multiplications is $d_c \lceil \log_2 n_c \rceil$, which is the (ideal) depth required for one Comp operation [14].

As the values of n_c and d_c increase, the Comp operation outputs values closer to 0 and 1. Fig. 1(a) shows the graph of $f_3(\cdot)$, and Fig. 1(b) shows the graphs of the Comp operation when the n_c value of 3 is used with the d_c values from 1 to 4. As the number of iterations increases, the total depth and execution time increase, which makes an application infeasible or prevents it from working in real time.

Another non-linear operation often used in real-world applications is a square root operation. The numerical algorithm for this operation, denoted by Sqrt, was presented in Cheon *et al.*'s work [13]. This algorithm contains a single loop with d_s iterations, and its optimal depth in the HE domain is $2d_s - 1$.

III. Application of Homomorphic Encryption to Background Subtraction

In this section, we propose methods to perform BGS on homomorphically encrypted data. Specifically, the numerical methods presented in Section II-C are adopted to perform the non-linear operations in (1) in the HE domain. Note that several studies use the Comp operation for HE-based applications [14], [15]. Unlike the prior studies, this work focuses on how to perform comparison operations involving absolute value calculations in the HE domain.

The most intuitive way is to run two separate Comp operations of which the first inputs are $I - B$ and $B - I$ but the second inputs are the same as T. We call this design *Design 0*. To satisfy the input condition of the Comp operation (i.e., $a - b \in [-1, 1]$), a normalization process is performed in advance. The worst case is when the difference between I and B becomes $-(2^{bpp} - 1)$, where *bpp* stands for bits per pixel. It sets the $a - b$ value to $-(2^{bpp} - 1) - T$. Therefore, to get the minimum bound of $a - b$ to be -1, the inputs of the Comp operation are multiplied by $1/(2^{bpp} - 1 + T)$.

If $|I - B| > T$, one of the two Comp operations outputs a result close to 0, and the other outputs a result close to 1. On the other hand, if $|I - B| < T$, the results of both Comp operations are close to 0. To calculate the final pixel value, these Comp operation results are added together and multiplied by $2^{bpp} - 1$. This approach keeps the error in absolute value calculation and the total depth low. However, it requires a long execution time because of the two expensive Comp operations.

If inputs of a comparison operation are positive numbers, squaring them does not change the result. Therefore, to reduce the execution time of Design 0, normalized $I - B$ and T are squared and compared to each other. This design, called *Design 1*, requires an additional multiplication for square calculation but eliminates the need to use two Comp operations in the HE domain, which almost doubles the processing speed of Design 0. However, since the Comp operation is an approximate operation, the result changes when inputs are squared. In particular, squaring causes the input values to be closer to 0, which makes the convergence of the Comp operation slower. Consequently, it increases the number of iterations to obtain an accurate comparison result.

Comparing $\sqrt{(I - B)^2}$ and T addresses this slow convergence problem. To implement this design, called *Design 2*, in the HE domain, the Sqrt operation [13] is used. The improved convergence speed comes at the cost of increased depth as well as introducing errors from the Sqrt operation. Compared to Design 0, the processing speed of Design 2 is faster because the Comp operation has a nested loop while the Sqrt operation has a single loop.

IV. Fast-Converging Method

Due to the depth and execution time issues in the HE domain, the actual result of the Comp operation lands somewhere

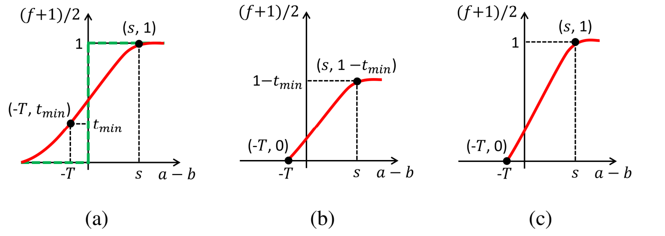

Fig. 2. Step functions in the Comp operation. (a) original ideal (in green) and approximate (in red) step functions (b) shifted approximate step function (c) shifted-and-scaled approximate step function.

within [0, 1]. Consider Design 1 for example, the value $I-B$ is squared. If $|I-B|$ is small, the input for the Comp operation after normalization is close to 0. Thus, especially in the case of lower d_c values, the outputs of Design 1 may not fully converge. Fig. 2(a) shows ideal and approximate step functions in the Comp operation. To have outputs of the Comp operation converge faster with small iterations, this section proposes a novel method based on the fact that a large number of $|I-B|-T$ values in BGS exist around -T.

The minimum output value of HEBGS occurs when I = B (i.e., $a=0$). In this case, the initial input of $f_{n_c}(\cdot)$ is just the normalized -T. The proposed fast-converging method calculates the theoretical minimum output of the Comp operation for -T, denoted by t_{min}, and subtracts it from outputs of the Comp operation for any I and B values. By this shifting, the output of HEBGS is artificially converged to 0 in the case when I = B. Fig. 2(b) shows the graph of the shifted function. In fact, the subtraction of t_{min} causes the output to shift away from 1 when $|I-B|$ is large. Thus, to ensure that the Comp operation fully converges for this scenario, the shifted function output is divided by $1-t_{min}$. Fig. 2(c) shows the result where the graph in Fig. 2(b) is scaled by $\frac{1}{1-t_{min}}$. Note that this scaling assumes that the theoretical maximum is 1.

V. Client-Device-Server Model

This section introduces a scenario where HEBGS is implemented as a client-device-server model. Fig. 3 shows the entities connected to one another, tasks, and data transmitted between them. The client is the entity generating public, secret, and relinearization keys. The public and relinearization keys are sent to the device and server, respectively. The client keeps the secret key and performs decryption upon the reception of the HEBGS results. The device is a remote entity capable of obtaining input frames. Captured frames may contain sensitive information, which must not be leaked out of the device. The device is implemented on a tiny microcontroller with a camera sensor. It encrypts captured input frames using the public keys, and the encrypted frames are sent to the server.

The server is a resource-rich third-party entity. On obtainment of an encrypted input frame, the server either stores it as a background model or performs HEBGS with the existing background model. Input, intermediate, and resulting frames are all encrypted, and therefore the server cannot acquire any sensitive information. After performing HEBGS, it sends the resulting masks to the client, and the client checks whether objects newly appear through the decrypted masks.

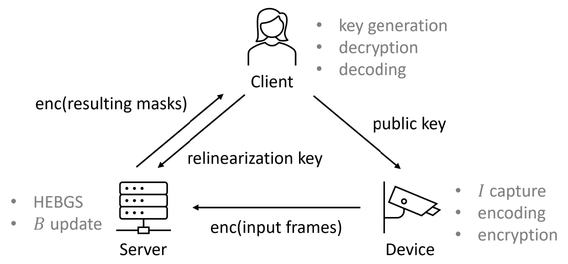

Fig. 3. A client-device-server model scenario for HEBGS.

Note that this paper focuses on how to convert the conventional BGS to HEBGS, not how to estimate B well. However, we suggest two simple methods for updating B: 1) the client side instructs the server to update after analyzing the results; 2) the server automatically updates after a set period of time.

VI. Evaluation

A. Parameters and Dataset

Microsoft SEAL open-source library (v3.6) was used for the HEBGS implementation [16]. In particular, the homomorphic operations of the CKKS scheme implemented in this library were used. HEBGS aims at a 128-bit security level, which is widely used in recent HE-based applications [14], [15], [17], [18]. Two important HE parameters are the polynomial degree N and the total bit length of q ($\log_2 q$), where q is the product of different primes. Each time multiplication of ciphertexts is performed, one prime is dropped. The N value of 2^{15} has 885 bits for $\log_2 q$ [19]. When each prime is assigned 40 bits, the number of available primes is 22. The first and last primes have special uses, and therefore the maximum depth is 20.

As the value of T, 40 was empirically chosen. A ciphertext with N of 2^{15} in the CKKS scheme has 2^{14} slots [20]. Therefore, each frame was resized to a 128×128 dimension to include all pixels in a single ciphertext. The n_c value was fixed at 3 because the depth of $f_{n_c}(\cdot)$ is proportional to the logarithm of n_c and 3 is the maximum value for a depth of 3. The d_c value changed from 1 to 4. The d_s value of the Sqrt operation was fixed at 2 because it was the maximum value that ran in conjunction with d_c of 4 and n_c of 3 of the Comp operation, not exceeding the maximum depth of 20.

We evaluated four proposed HEBGS designs: Design 1, Design 2, Design 3 (= Design 1 + fast-converging), and Design 4 (= Design 2 + fast-converging). As test videos, two simple videos — SI-01 (frames #75-300) and SI-02 (frames #70-300) [21] — and two complex videos — Lab (frames #267-400) [22] and Nominal Motion (frames #275-500) [23] — were used. The first few frames without foreground objects were excluded, and the frame before foreground objects appear was used as an initial background model for each video.

B. Accuracy

To evaluate the still frame quality of the HEBGS designs, the peak signal-to-noise ratio (PSNR) and structural similarity

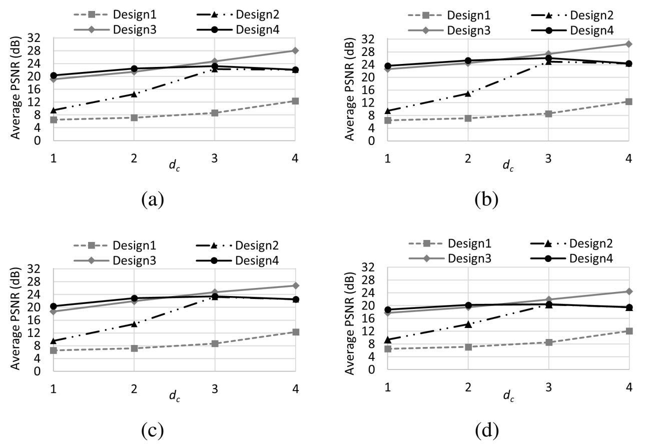

Fig. 4. Average PSNR results of the HEBGS designs compared to the conventional BGS [5]. (a) SI-01 (b) SI-02 (c) Lab (d) Nominal Motion.

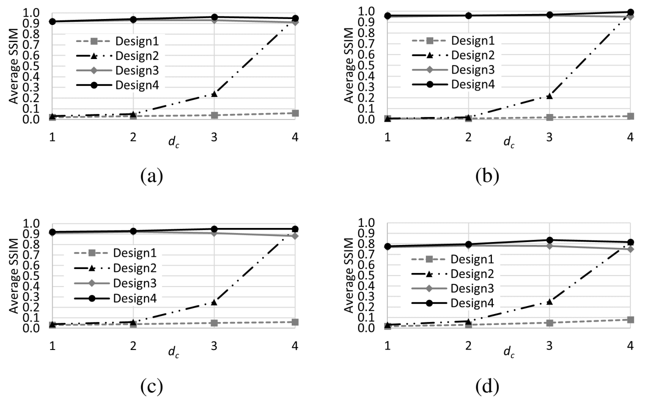

Fig. 5. Average SSIM results of the HEBGS designs compared to the conventional BGS [5]. (a) SI-01 (b) SI-02 (c) Lab (d) Nominal Motion.

index measure (SSIM) functions implemented in OpenCV were used [24]. The resulting frames of the non-HE-based conventional BGS [5] were used as reference frames of the PSNR and SSIM functions. For each video, the PSNR values and SSIM values of all frames were averaged, respectively.

Figs. 4 and 5 show the average PSNR and SSIM results depending on the d_c value, respectively. Overall, PSNR and SSIM values for all designs increase as the d_c value increases. Specifically, at higher d_c values, a larger input value for $f_{n_c}(\cdot)$ in the square root-based designs makes them outperform the square-based designs in terms of SSIM values. This is especially evident when not using the fast-converging method.

The use of the fast-converging method drastically improves the accuracy. However, as the d_c value increases, the effectiveness of the fast-converging method diminishes. At d_c of 4 for instance, Design 4 results in total convergence and no longer benefits from the fast-converging method. This is because the t_{min} value in the fast-converging method is approximately 0 at d_c of 4.

The PSNR values for square root-based designs drop slightly when the d_c value exceeds 3. It is because the `Comp` operation becomes accurate and the inaccuracies caused by the `Sqrt` operation began to show. In particular, when $|I - B|$ is close to T, the error created by approximating $\sqrt{(I-B)^2}$ may cause it to end up on the wrong side of T, making the comparison result to 1 when it should be 0.

TABLE I
DEPTH AND EXECUTION TIME (IN SECONDS) PER FRAME OF HEBGS

d_c	Design 1		Design 2		Design 3		Design 4	
	Dep.	Time	Dep.	Time	Dep.	Time	Dep.	Time
1	8	0.33	11	0.70	8	0.34	11	0.69
2	11	0.81	14	1.37	11	0.81	14	1.38
3	14	1.56	17	2.27	14	1.59	17	2.27
4	17	2.55	20	3.57	17	2.55	20	3.55

C. Total Depth and Execution Time

To evaluate the execution time of the HEBGS designs, a PC with the Intel Xeon W-2295 CPU and 128GB RAM was used. A single thread was used when running the software. Table I shows the execution time results per frame along with the total depth results. Since the execution times of the client and device functions are relatively short, only the execution times of the HEBGS designs running on the server are shown.

As the d_c value increases, the total depth and execution time increase, which is common to all the cases in this table. Designs 2 and 4 using the `Sqrt` operation use 3 more depth than Designs 1 and 3, resulting in longer execution times. However, they show shorter execution times than Designs 1 and 3 that use the same depth, and the difference increases as the d_c value increases. This is because the computational complexity of $f_{n_c}(\cdot)$ is higher than that of the `Sqrt` operation.

Our fast-converging method has little effect on the execution time. When the d_c value is 1, where the outer loop of the `Comp` operation is removed, Design 4 shows approximately 1dB higher average PSNR results compared to Design 3. However, in terms of execution time, Design 3 shows a 51% reduction, which makes Design 3 the best overall among the four designs.

VII. CONCLUSION

This paper applies HE to BGS. To perform a comparison operation in the HE domain, a numerical method is adopted. For absolute value calculation, square- and square-root-based designs are proposed. To reduce the total depth and execution time in the comparison operation of HEBGS, a novel fast-converging method is proposed, which even improves the accuracy of resulting frames. When both accuracy and execution time are considered, the square-based HEBGS design with the fast-converging method is the most superior.

ACKNOWLEDGMENT

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-00840, Development and Library Implementation of Fully Homomorphic Machine Learning Algorithms supporting Neural Network Learning over Encrypted Data). In addition, this material is based upon work supported by the National Science Foundation under Grant No. 2105373.

References

[1] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*. Berkeley, CA, USA: Lee & Seshia, 2011.

[2] J. H. Cheon *et al.*, "Introduction to Homomorphic Encryption and Schemes," in *Protecting Privacy through Homomorphic Encryption*, Springer, 2021, pp. 3–28.

[3] K. Akkaya, V. Baboolal, N. Saputro, S. Uluagac, and H. Menouar, "Privacy-Preserving Control of Video Transmissions for Drone-based Intelligent Transportation System," in *Proc. CNS*, 2019.

[4] M. Piccardi, "Background Subtraction Techniques: A Review," in *Proc. SMC*, 2004.

[5] S. E. Umbaugh, *Computer Imaging: Digital Image Analysis and Processing*, Boca Raton, FL, USA: CRC Press, 2005.

[6] Z. Brakerski, "Fully Homomorphic Encryption Without Modulus Switching From Classical GapSVP," in *Proc. CRYPTO*, 2012.

[7] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) Fully Homomorphic Encryption Without Bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 1-36, 2014.

[8] J. Fan and F. Vercauteren, "Somewhat Practical Fully Homomorphic Encryption," Cryptology ePrint Archive, Tech. Rep. 2012/144, 2012.

[9] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in *Proc. ASIACRYPT*, 2017.

[10] S. Kim, K. Lee, W. Cho, J. H. Cheon, and R. A. Rutenbar, "FPGA-based Accelerators of Fully Pipelined Modular Multipliers for Homomorphic Encryption," in *Proc. ReConFig*, 2019.

[11] S. Kim, K. Lee, W. Cho, Y. Nam, J. H. Cheon, and R. A. Rutenbar, "Hardware Architecture of a Number Theoretic Transform for a Bootstrappable RNS-based Homomorphic Encryption Scheme," in *Proc. FCCM*, 2020.

[12] J. H. Cheon, D. Kim, and D. Kim, "Efficient Homomorphic Comparison Methods with Optimal Complexity," in *Proc. ASIACRYPT*, 2020.

[13] J. H. Cheon, D. Kim, D. Kim, H. H. Lee, and K. Lee, "Numerical Method for Comparison on Homomorphically Encrypted Numbers," in *Proc. ASIACRYPT*, 2019.

[14] P. Nam, J. Shyi, and S. Kim, "HEMTH: Small Depth Multilevel Thresholding for a Homomorphically Encrypted Image," in *Proc. MMSP*, 2022.

[15] M. Cho, K. Lee, and S. Kim, "HELPSE: Homomorphic Encryption-based Lightweight Password Strength Estimation in a Virtual Keyboard System," in *Proc. GLSVLSI*, 2022.

[16] Microsoft SEAL (release 3.6), [Online]. Available: https://github.com/Microsoft/SEAL

[17] S. D. Kannivelu and S. Kim, "A Homomorphic Encryption-based Adaptive Image Filter Using Division Over Encrypted Data," in *Proc. RTCSA*, 2021.

[18] D. L. Elworth and S. Kim, "HEKWS: Privacy-Preserving Convolutional Neural Network-based Keyword Spotting with a Cipehrtext Packing Technique," in *Proc. MMSP*, 2022.

[19] H. Chen, K. Han, Z. Huang, A. Jalali, and K. Laine, "Simple Encrypted Arithmetic Library v2.3.0," [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2017/12/sealmanual.pdf

[20] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for Approximate Homomorphic Encryption," in *Proc. EUROCRYPT*, 2018.

[21] C. Cuevas, E. M. Yáñez, and N. García, "Labeled Dataset for Integral Evaluation of Moving Object Detection Algorithms: LASIESTA," *Comput. Vis. Image Underst.,* vol. 152, pp. 103-117, Nov. 2016.

[22] C. Benedek and T. Szirányi, "Bayesian Foreground and Shadow Detection in Uncertain Frame Rate Surveillance Videos," *IEEE Trans. Image Process.*, vol. 17, no. 4, pp. 608-621, Mar. 2008.

[23] Y. Sheikh and M. Shah, "Bayesian Modeling of Dynamic Scenes for Object Detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 11, pp. 1778-1792, 2005.

[24] Video Input with OpenCV and Similarity Measurement, [Online]. Available: https://docs.opencv.org/4.x/d5/dc4/tutorial_video_input_psnr_ssim.html