TrustEYE.M4: Protecting the Sensor – not the Camera

Thomas Winkler[1], Ádám Erdélyi[1], and Bernhard Rinner[1]

[1]Institute of Networked and Embedded Systems and Lakeside Labs
Alpen-Adria-Universität Klagenfurt
Lakeside Park B02b, 9020 Klagenfurt, Austria
{firstname.lastname}@aau.at

Abstract

Images captured in camera networks are potentially privacy sensitive and therefore need protection. A critical aspect is where protection is applied – after transmission at the data center or preferably already on the camera. In this work we take on-camera protection a step further and propose to make privacy protection and security inherent features of the image sensing unit. Already within the camera we realize strong separation between components that have access to raw image data and those that do not need raw data access. Our approach is based on the custom-designed TrustEYE.M4 prototype of a secure sensing unit. We demonstrate the feasibility of sensor-level privacy protection with a cartoon-like effect based on mean shift filtering.

1. Motivation and Approach

In home monitoring and assisted living applications cameras are deployed in private environments where captured data is highly privacy sensitive. This is addressed by integrating privacy protection techniques [2, 8], combined with IT-security features [3, 16, 19], directly into cameras.

With our TrustEYE.M4 prototype [18] we contribute to the state of the art by moving privacy protection and security into the image sensing unit. We demonstrate the feasibility of this approach by applying real-time privacy protection via a cartoon-like effect at the sensor level. Furthermore, we advance the state of the art by addressing the following two major limitations of existing secure camera solutions:

Implicitly trusted software components. Large software components of a camera such as the operating system, the network stack and system libraries are part of the implicitly trusted software base. Even with, e.g., secure boot which ensures that only genuine software is executed, it is impossible to guarantee the absence of, yet undisclosed, security flaws which can be exploited by attackers. It is therefore a 'best practice' to reduce the amount trusted components as much as possible.

Lack of separation. On-camera privacy protection and security measures are performed typically at the application level as part of the computer vision tasks. Consequentially, security and privacy protection are tightly interwoven with the application logic and they are left in the responsibility of application developers.

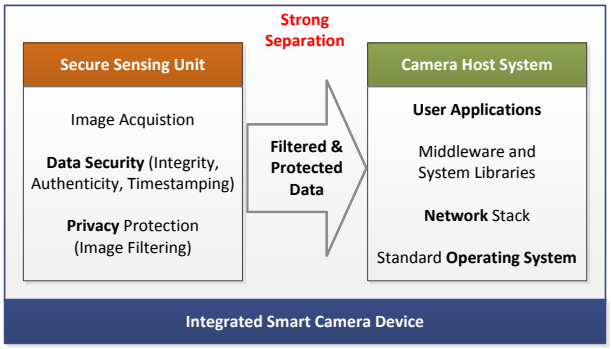

Figure 1. The camera is divided into a secure sensing unit with exclusive access to raw images and an untrusted camera host system which runs user applications, middleware and networking tasks.

Figure 1 presents an overview of our secure sensing unit approach which overcomes these two major limitations. The camera device is divided internally into a secure sensing unit and a camera host system. The secure sensing unit has exclusive access to raw image data and it applies data security techniques and image pre-filtering for privacy protection. Data security typically provides non-repudiation guarantees [12, 20] (i.e., authenticity, integrity and freshness/timestamping) for captured images. This can be achieved via cryptographic techniques [19] or via image watermarking [11] and steganography. Privacy protection

is implemented by filtering captured images before they are forwarded to the camera host system. Filtering can be performed for regions of interest (object-based) or for the entire image (global). Object-based filtering [14] could target human bodies or face [10]. In this case the achieved privacy protection depends on the reliability of the underlying object detection. Global techniques [6, 15] do not depend on detection performance and therefore provide higher privacy guarantees. Adaptive global protection combines both approaches by incorporating the results of unreliable detectors [15] to determine the strength of global protection.

For a secure sensing unit several approaches exists. High security against manipulation is achieved by fabricating the secure sensing unit as an ASIC. This promises also high performance but is rather inflexible in case of changing system requirements. In contrast to that are software solutions which use virtualization to realize two separated system domains. This achieves high flexibility but might not be on par with the security of an ASIC. In-between these two approaches we see SoC-based solutions. Critical security features are provided by hardware while controlled flexibility is achieved via the firmware running on the SoC.

Regardless of the chosen approach, there are major challenges which have to be considered during design:

Privacy protection vs. utility. A central aspect of camera networks is to be able to observe behavior of monitored individuals. Privacy protection inside the secure sensing unit and the need for behavior monitoring must be balanced such that the utility of the camera system does not degrade severely. Due to application requirements, different regional laws and different cultural attitudes there is no single solution but a continuum.

Flexibility and adaptation. To be able to choose a scenario-specific privacy vs. utility tradeoff, also the underlying platform must be flexible and support the adaptation of the implemented privacy protection solutions. It is critical that access to this adaptation functionality is limited strictly to eligible parties.

Resource limitations. The secure sensing unit is closely integrated with the image sensor. Due to cost, space and resource constraints the protection features in the secure sensing unit have to be relatively light-weight.

The remainder of this paper is organized as follows. Section 2 presents related work in the field of sensor-level security. Thereafter, Section 3 describes the hard- and software architecture of TrusteEYE.M4 – our custom-designed SoC-based secure sensing solution. Section 4 demonstrates the feasibility of the approach with a cartoon-like privacy filter applied in the secure sensing unit before images are forwarded to the camera host system. Finally, Section 5 concludes the paper and gives an outlook to future work.

2. Related Work

Several related approaches for camera and sensor security are presented in this section.

PrivacyCam [3] is a camera system based on a Blackfin DSP clocked at 400 MHz, 32 MB of SDRAM and an Omnivision OV7660 color CMOS sensor. Regions of interest are identified based on background subtraction and resulting regions are encrypted using AES.

TrustCAM [19] exploits the capabilities of a Trusted Platform Module (TPM) to implement a secure camera. The TPM is used to monitor and record the software state of the camera, to implement authenticity and integrity guarantees via digital signatures and to provide secure timestamping. The camera ensures confidentiality of all captured data.

Most work on securing image data at the sensor level has focused so far on providing integrity protection and authenticity guarantees. The most common implementation form is by embedding watermarks into the captured data [9].

Mohanty and Adamo [12, 1] describe a secure digital camera system that provides integrity, authenticity and ownership guarantees for digital video via a combination of watermarking and encryption. A binary watermark image is encrypted with a user-supplied key before it is embedded into the image. An FPGA-based prototype demonstrates the approach under real-time conditions. Karthigaikumar and Baskaran [7] focus on real-time performance and as well as low power consumption with their ASIC implementation of a custom watermarking algorithm.

De Strycker et al. [5] use a digital signal processor to embed an invisible, digital watermark into video frames in real-time. The watermark consists of a pseudo-noise pattern that depends on a secret key. The system is evaluated in the context of a video broadcasting application where it provides authenticity guarantees for delivered video streams.

Nelson et al. [13] propose an image sensor with built-in watermarking. In their concept, every image sensor is equipped with a unique, secret key used to generate pseudo-random noise serving as watermark. To verify image authenticity, a recipient has to know the sensor's secret key.

Stifter et al. [17] suggest to integrate a secure storage for a symmetric, cryptographic key into the image sensor. This key is used as part of message authentication code (MAC) computations. The system provides integrity and authenticity guarantees for delivered data. Furthermore, image freshness is guaranteed via a 'non-repeating' sequence number.

3. TrustEYE.M4 System Architecture

TrustEYE.M4 is entirely custom-designed at our lab including both hard- as well as software. The main design goal is its application as a secure sensing unit together with an off-the-shelf camera host system. For that application, hardware components have been selected which provide

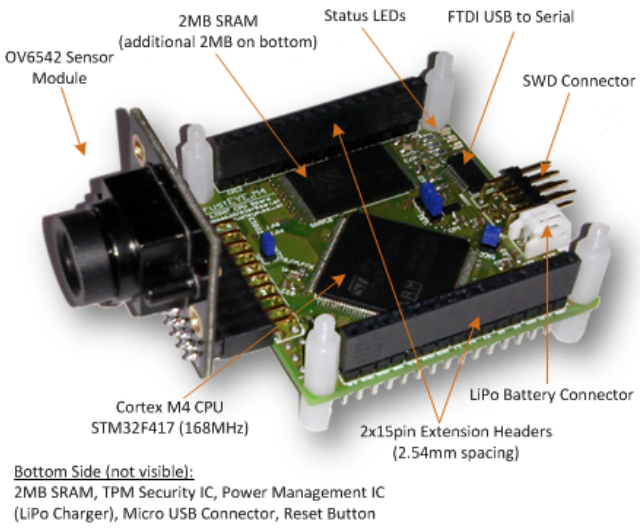

Figure 2. The 50×50 mm TrustEYE.M4 CPU board with an OmniVision OV5642 image sensor module.

a dedicated high-performance image sensor interface and hardware security features for firmware protection.

The TrustEYE.M4 CPU board shown in Figure 2 is based on a two layer 50×50 mm printed circuit board and is using an STM32F417 ARM Cortex M4 microcontroller. The CPU provides 192 kB on-chip SRAM and 1 MB on-chip program Flash memory. Since the on-chip SRAM is insufficient to hold multiple images for processing and since typical computer vision algorithms require additional storage for intermediate results, an additional 2×2 MB of external SRAM are included on the circuit board. Data transfers from the image sensor module to SRAM and from SRAM to the camera host system are implemented via the microcontroller's DMA engines such that the CPU itself is available for image processing. The system is powered either via a Micro-USB connector, a single-cell lithium polymer battery or directly via the camera host system.

Currently two image sensor modules are supported – one with an OmniVision OV7725 (640×480) and one with an OmniVision OV5642 sensor (5 megapixels). The sensors are configured via the I2C bus, deliver their data via a parallel 8-bit interface and can be configured for various data formats including YUV422, YUV420 or RGB.

Programming and debugging support is provided via the Serial Wire Debug (SWD) interface or the controller's serial bootloader. An dedicated connector (cp. Figure 5) attaches TrustEYE.M4 via SPI to a RaspberryPI[1] single-board computer running Linux which serves as camera host system.

The TrustEYE.M4 CPU provides hardware accelerators for cryptographic algorithms including AES256, SHA1, SHA256 and HMAC. Furthermore the SoC provides a true random number generator and a 96-bit unique ID. The

[1]RaspberryPI SBC: http://www.raspberrypi.org (visited: 03/2014)

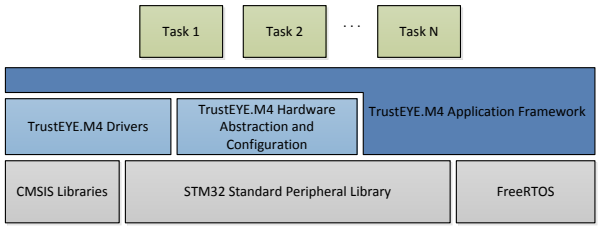

Figure 3. TrustEYE.M4's software stack includes standard libraries and FreeRTOS (gray), custom drivers and an application framework (blue). On top are actual application tasks (green).

chip's program Flash memory can be both permanently read- and write protected. The on-board ST33TPM12SPI TPM chip provides RSA key generation (2048 bits), RSA signature creation and encryption, secure monotonic counters, remote attestation capabilities and comes with an endorsement key certificate. The TPM is the basis for providing non-repudiation guarantees (integrity, authenticity and timestamping) for captured images based on TPM-protected, non-migratable 2048 bit RSA keys.

Figure 3 presents an overview of the software architecture of TrustEYE.M4. The lowest layer consist of three components: (1) The CMSIS (Cortex Microcontroller Software Interface Standard)[2] library, (2) the STM32 Standard Peripheral Library[3] which provides an access layer for on-chip peripherals and (3) the FreeRTOS[4] real-time operating system. The second layer contains custom hardware drivers for, e.g., the image sensors and the TPM. The TrustEYE.M4 hardware abstraction provides a flexible mechanism for the configuration of hardware functions and pin mappings. The TrustEYE.M4 application framework supports the de-composition of applications into tasks which are scheduled by FreeRTOS. As a single-core processor system, TrustEYE.M4 does not support parallel execution. However, the DMA engines of the CPU allow to implement bulk data transfers without blocking the CPU. To ensure that tasks waiting for DMA completion do not block other tasks, a synchronized, double-buffering mechanism is included in the software framework for efficient data handover.

4. Secure Sensing Unit Demonstrator

Figure 4 shows a setup where the TrustEYE.M4 CPU board is connected to a RaspberryPI Embedded Linux system. TrustEYE.M4 acts as the image sensing unit which pre-processes captured images before forwarding them to the RaspberryPI camera host system. To demonstrate the proposed concept of a secure sensing unit we implement the application outlined in Figure 5. It shows clearly the strict separation of the sensing unit from the camera host system

[2]CMSIS Library: http://www.arm.com/cmsis/ (visited: 03/2014)
[3]STM Standard Periph. Library: http://www.st.com (visited: 03/2014)
[4]FreeRTOS website: http://www.freertos.org (visited: 03/2014)

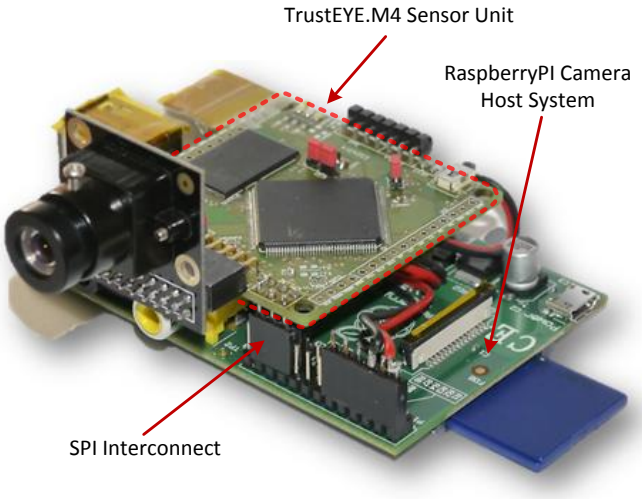

Figure 4. TrustEYE.M4 used as a secure sensing unit on top of a RaspberryPI Linux system serving as camera host system.

where the Linux operating system, the streaming application and potential user applications are executed. Security flaws in, e.g., the network stack of the Linux operating system may result in a security breach of the camera host system. The isolated sensing unit, which is the only entity that has access to the raw image data, is not affected. On TrustEYE.M4, YUV422 images are read from the sensor which are then filtered by the privacy task with a global cartoon-like effect as described subsequently in section 4.1. The resulting, pre-filtered image data is then forwarded to the camera host system for further processing.

4.1. Cartoon-like Privacy Filter

We designed and implemented a cartoon-like filtering effect for TrustEYE.M4 which is applied globally to the captured frames. This makes privacy protection independent of unreliable region of interest detection. Cartooning effects, as proposed in [6], achieve good privacy protection while maintaining high levels of intelligibility. A central component of cartooning is color-based segmentation where the mean shift algorithm was shown to deliver good results. Established mean shift filtering implementations as, e.g., *pyrMeanShiftFilter()* from OpenCV, are too complex to be ported to the resource-constraint TrustEYE.M4. Therefore, we implemented a customized filter inspired by mean shift which operates on YUV422 images delivered by the sensor.

In mean shift filtering, for every pixel location (X, Y) of an input image I a surrounding spatial region (Eq. 1) defined by radius r is examined.

$$R = \{(x, y) \in I \mid (X - r < x < X + r) \wedge (Y - r < y < Y + r)\}. \quad (1)$$

For every location (x, y) the color distance between to region's center (X, Y) is computed (Eq. 2). Note that the concept of mean shift is independent of a particular color model. If $colorDist$ is smaller than a defined color radius c then $color(x, y)$ contributes to the mean color of the region $Color_{R_{mean}}$ and (x, y) contributes to the spatial mean of the region $(X_{R_{mean}}, Y_{R_{mean}})$.

$$colorDist = dist(color((X, Y)), color(x, y)). \quad (2)$$

The mean shift procedure is then repeated with $(X_{R_{mean}}, Y_{R_{mean}})$ as the new region center. This procedure is continued until a termination criterion (e.g., number of iterations or minimal step size) is met. Finally, the pixel value at $I(X, Y)$ is set to $Color_{R_{mean}}$ of the last iteration.

4.1.1 Cartoon Effect for Low-Resource Devices

For a mean shift filtering variant that is computationally feasible on TrustEYE.M4 we limit the number of iterations to one. Furthermore, the processing effort for examining the individual regions R around each pixel position (X, Y) is very high, especially for larger spatial radii which achieve good visual results. To overcome this limitation we have re-designed the mean shift procedure and introduced a processing step where N integral images [4] are computed per frame. Each integral image $IntImg[n]$ is computed for the original input image I but only for the n^{th} sub-range of the color space ($colorRange[n]$). The $update()$ function in the $compIntImg()$ function shown in Listing 1 only incorporates the current value $I(X, Y)$ into the $IntImg[n]$ if it is within within $colorRange[n]$. Once complete, each integral image holds at every position the integrals of the color values belonging to the respective $colorRange[n]$. Additionally, the $update()$ function stores at each position the count integral of how many elements fall into $colorRange[n]$. It must be noted that depending on the underlying color model, the individual channels can not be treated separately when computing $IntImg[1...N]$. In our case of YUV we treat the Y channel independently whereas the U and V channel are considered jointly.

Listing 1. Integral image computation.

```
1  func compIntImg(I, n)
2    var IntImg
3    for each (X, Y):
4      if (I(X,Y) in colorRange[n]):
5        update(IntImg(X, Y), I(X, Y))
6      else
7        update(IntImg(X, Y), null)
8    return IntImg
9
10 for (n = 1 to N):
11   IntImg[n]=compIntImg(I, colorRange[n])
```

The actual cartooning procedure now iterates over all image points $I(X, Y)$, computes the corresponding color range n and extracts the four corner points (A, B, C, D) [5] of

[5] A = top left; D = bottom right point of R

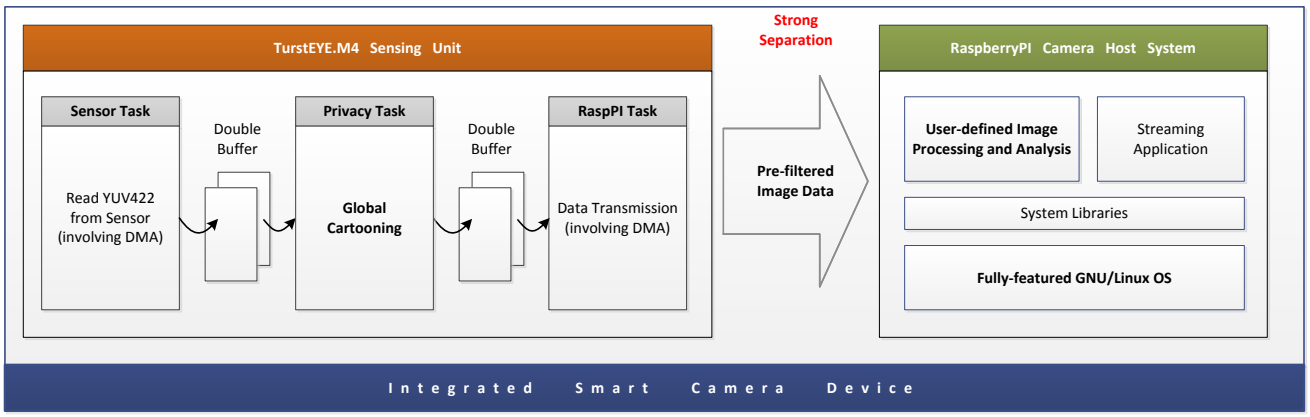

Figure 5. In this example TrustEYE.M4 is attached to a RaspberryPI camera host system (cp. Figure 4). TrustEYE.M4 has exclusive access to the image sensor and applies globally a cartoon-like filter to raw images before forwarding them to the camera host system. On the camera host system standard operating systems, software libraries and user-defined applications (e.g., video streaming) are executed.

the spatial region R around (X, Y) from $IntImg[n]$. Each corner point consists of the color integral and the count integral, e.g., $A = (Color_{INT\_A}, Count_{INT\_A})$. The region's color and count integrals are computed as

$$(Color_{INT}, Count_{INT}) = D - B - C + A. \quad (3)$$

and subsequently, the mean color can be computed and set:

$$I(X, Y) = Color_{INT}/Count_{INT}. \quad (4)$$

4.1.2 Performance Considerations

A rough runtime comparison of our mean shift variant with the standard approach reveals a substantial difference. For the standard mean shift an $r \times r$ region is examined for every location (X, Y) of an image with a spatial resolution of $hRes \times vRes$ resulting in the following runtime estimate:

$$t_{mshift} = O(hRes \times vRes \times r^2). \quad (5)$$

For the mean shift variant implemented on TrustEYE.M4 we first computed the N integral images where N times the $hRes \times vRes$ image is traversed followed by an additional traversal for the computation of the final output values:

$$t_{mshift.M4} = O((hRes \times vRes) \times (N + 1)). \quad (6)$$

The runtime estimates show that our mean shift variant is independent of the spatial radius r and depends instead on the number of color regions N. N is typically smaller than r and affects runtime not in squared form as r does. Since the spatial radius does not affect the runtime and region R is solely defined by the corner points (A, B, C, D) it involves no extra computational effort to shrink or enlarge the region.

Memory requirements of our mean shift variant are substantially higher than those of standard versions because N integral images have to be stored. Furthermore, the elements of the integral images require typically 32 bits to accommodate the integral values. To reduce the memory requirements we have introduced an iterative computation of the integral images where a window of $hRes \times r$ is moved from top to bottom and N integral images of $hRes \times r$ instead of $hRes * vRes$ are kept in memory.

In terms of delivered results our modified mean shift version is not on par with standard implementations. This is due to the fact that we do not support a color radius centered around a pixels value $I(X, Y)$ but colors are sorted into n color regions. Color separation in the output is less pronounced and accurate than in standard mean shift.

	ext. SRAM	int. SRAM
Mean Shift	89 ms	62 ms
Roberts Cross	11 ms	n/a

Table 1. Execution times on TrustEYE.M4.

4.2. Evaluation

Figure 6 shows an example of a cartooned image from TrustEYE.M4. Sample videos are available on the TrustEYE website [18]. The spatial radius r is 32 and the number of regions N for the Y channel and the interdependent U/V channels are set to 4. To enhance the cartoon-like effect we additionally apply Roberts cross filtering for edge enhancement. For an image resolution of 320×240 we achieve 11 fps including transmission of uncompressed images to the RaspberryPI and subsequent streaming via Ethernet. Table 1 presents the runtimes for the customized mean shift function and the Roberts cross edge detection. With the iterative computation approach, the integral images fit into internal SRAM which notably speeds up data access. This results in a mean shift runtime of 62 ms per frame. Including edge-enhancement, a runtime of 73 ms per frame is achieved resulting in a theoretical frame rate of 13.7 fps.

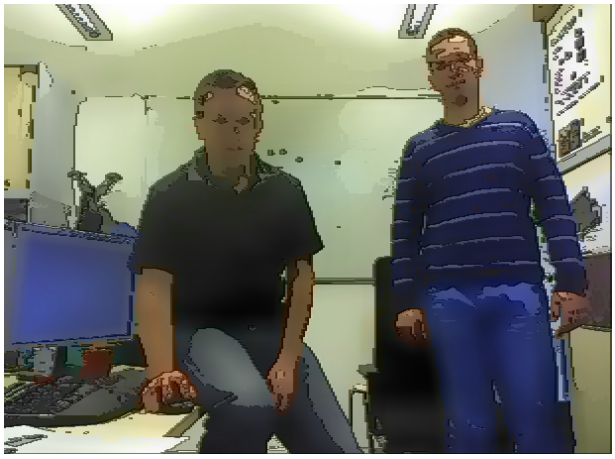

Figure 6. Cartooning image from the TrustEYE.M4 sensing unit.

Due to SPI bus limitations and network overheads this is reduced in practice to the already mentioned 11 fps.

5. Conclusion and Outlook

In this work we presented TrustEYE.M4 – a novel approach towards moving security and privacy protection as close to the sensor as possible and thereby reducing the number of implicitly trusted components of a smart camera. With a customized version of mean shift filtering we demonstrated that even on severely resource-limited devices global cartooning effects for privacy protection are feasible. Ongoing work includes the thorough evaluation of the utility vs. privacy tradeoff achieved with global filtering such as cartooning, the exploration of approaches that allow to recover identities under controlled conditions and the integration of underlying hardware-based security features to guarantee strong protection of TrustEYE.M4's firmware.

Acknowledgment

This work was performed in the project *TrustEYE: Trustworthy Sensing and Cooperation in Visual Sensor Networks* [18]. It was funded by the European Regional Development Fund (ERDF) and the Carinthian Economic Promotion Fund (KWF) under grant KWF-3520/23312/35521.

References

[1] O. Adamo, S. P. Mohanty, E. Kougianos, and M. Varanasi. VLSI Architecture for Encryption and Watermarking Units Towards the Making of a Secure Camera. In *Proceedings of the Int. System-on-Chip Conference*, pages 141–144, 2006.

[2] C. Araimo and J.-l. Dugelay. Scrambling Faces for Flexible Privacy Preservation using Image Background Self-similarities. In *Proceedings of the MediaEval Workshop*, 2 pages, 2012.

[3] A. Chattopadhyay and T. E. Boult. PrivacyCam: A Privacy Preserving Camera Using uClinux on the Blackfin DSP. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[4] F. C. Crow. Summed-area tables for texture mapping. *ACM SIGGRAPH Computer Graphics*, 18(3):207–212, July 1984.

[5] L. De Strycker, P. Termont, J. Vandewege, J. Haitsma, A. Kalker, M. Maes, and G. Depovere. Implementation of a Real-time Digital Watermarking Process for Broadcast Monitoring on a TriMedia VLIW Processor. *IEE Proceedings - Vision, Image, and Signal Processing*, 147(4):371, 2000.

[6] A. Erdélyi, T. Winkler, and B. Rinner. Serious Fun: Cartooning for Privacy Protection. In *Proceedings of the MediaEval Workshop*, 2 pages, 2013.

[7] P. Karthigaikumar and K. Baskaran. FPGA and ASIC Implementation of Robust Invisible Binary Image Watermarking Algorithm Using Connectivity Preserving Criteria. *Microelectronics Journal*, 42(1):82–88, Jan. 2011.

[8] P. Korshunov and T. Ebrahimi. Using Warping for Privacy Protection in Video Surveillance. In *Proceedings of the Internat. Conference on Digital Signal Processing*, 6 pages, 2013.

[9] E. Kougianos, S. P. Mohanty, and R. N. Mahapatra. Hardware Assisted Watermarking for Multimedia. *Computers & Electrical Engineering*, 35(2):339–358, Mar. 2009.

[10] I. Martínez-Ponte, X. Desurmont, J. Meessen, and J.-F. Delaigle. Robust Human Face Hiding Ensuring Privacy. In *Proceedings of the International Workshop on Image Analysis for Multimedia Interactive Services*, 4 pages, 2005.

[11] P. Meerwald and A. Uhl. Watermarking of Raw Digital Images in Camera Firmware. *IPSJ Transactions on Computer Vision and Applications*, 2:16–24, 2009.

[12] S. P. Mohanty. A Secure Digital Camera Architecture for Integrated Real-Time Digital Rights Management. *Journal of Systems Architecture*, 55(10-12):468–480, Oct. 2009.

[13] G. Nelson, G. Jullien, and O. Yadid-Pecht. CMOS Image Sensor with Watermarking Capabilities. In *Internat. Symposium on Circuits and Systems*, pages 5326–5329, 2005.

[14] F. Z. Qureshi. Object-Video Streams for Preserving Privacy in Video Surveillance. In *Proceedings of the International Conference on Advanced Video and Signal-Based Surveillance*, pages 442–447, 2009.

[15] M. Saini, P. K. Atrey, S. Mehrotra, and M. S. Kankanhalli. Adaptive Transformation for Robust Privacy Protection in Video Surveillance. *Advances in Multimedia*, 14 pages, 2012.

[16] D. N. Serpanos and A. Papalambrou. Security and Privacy in Distributed Smart Cameras. *Proceedings of the IEEE*, 96(10):1678–1687, Oct. 2008.

[17] P. Stifter, K. Eberhardt, A. Erni, and K. Hoffmann. Image Sensor for Security Applications with On-chip Data Authentication. *Proceedings of the Society of Photo-Optical Instrumentation Engineers*, 6241:8, 2006.

[18] T. Winkler. Website of Project TrustEYE: Trustworthy Sensing and Collaboration in Visual Sensor Networks. `http://trusteye.aau.at`, 2012. last visited: June 2014.

[19] T. Winkler and B. Rinner. Securing Embedded Smart Cameras with Trusted Computing. *EURASIP Journal on Wireless Communications and Networking*, 2011:20, 2011.

[20] T. Winkler and B. Rinner. Security and Privacy Protection in Visual Sensor Networks: A Survey. *ACM Computing Surveys*, 47(1):42, 2014. (in print).