TrustCAM: Security and Privacy-Protection for an Embedded Smart Camera based on Trusted Computing

Thomas Winkler
Institute of Networked and Embedded Systems
Klagenfurt University
Lakeside Park B02b
9020 Klagenfurt, Austria
thomas.winkler@uni-klu.ac.at

Bernhard Rinner
Institute of Networked and Embedded Systems
Klagenfurt University
Lakeside Park B02b
9020 Klagenfurt, Austria
bernhard.rinner@uni-klu.ac.at

Abstract

Security and privacy protection are critical issues for public acceptance of camera networks. Smart cameras, with onboard image processing, can be used to identify and remove privacy sensitive image regions. Existing approaches, however, only address isolated aspects without considering the integration with established security technologies and the underlying platform. This work tries to fill this gap and presents TrustCAM, a security-enhanced smart camera. Based on Trusted Computing, we realize integrity protection, authenticity and confidentiality of image data. Multiple levels of privacy protection, together with access control, are supported. Impact on overall system performance is evaluated on a real prototype implementation.

1. Introduction and Motivation

Smart camera networks attract a lot of attention in research [13]. They have the potential for many applications which are not limited to classical surveillance but also include new fields such as elderly care, home automation or entertainment. The use of wireless interfaces is increasing and the amount of software running on cameras is growing. With these properties, cameras are likely to become attractive targets for attackers. Operators consequently will be interested in mechanisms that allow to reliably check the state of cameras and to get assurance that, e.g., a video is unmodified and comes from the intended camera. Likewise, monitored persons have the interest that their privacy is preserved and videos can only be accessed by authorized parties.

In this work, we enhance smart camera security by applying Trusted Computing (TC). TC proposes a hardware based security solution. A microchip – the Trusted Platform Module (TPM) – provides a set of well defined and well reviewed security primitives. To our knowledge, this is the first work that implements and evaluates Trusted Computing concepts in a embedded smart camera. An advantage of TPMs is that they are cheap and readily available. The drawback that comes with low price is that current implementations are relatively slow. Therefore, challenges lie in the proper integration into a camera system and the computer vision tasks. Our contribution is threefold: First, we describe our custom camera prototype with integrated TC support. Second, we discuss how TC can increase the security of a camera and how it can be used to realize a multi-level, privacy protection system. Third, we present results from our practical implementation and discuss the impact on overall system performance. The remainder of this work is organized as follows: After summarizing related work in section 2, we give an outline of basic TC concepts (section 3). Our system design is presented in section 4. Section 5 discusses the security enhancements, their implementation as well as evaluation results. Section 6 concludes the paper.

2. Related Work

The importance of privacy and security for camera networks has been recognized by many researchers. Efforts have been made to protect privacy of monitored persons, e.g., by Fleck et al. [7] in the context of elderly care applications. Serpanos et al. [12] do not limit their considerations to privacy but provide a holistic discussion of security questions. They cover fundamental topics including data confidentiality, integrity, freshness or mutual authentication and discuss their relevance for camera networks. Cavallaro [3] highlights the need for privacy protection and argues that smart cameras offer the potential for major improvements over CCTV systems. Computer vision is powerful enough to identify personal data in videos but challenges remain in making algorithms reliable and robust. Senior et al. [11] discuss the meaning of privacy in video surveillance and conclude that there is no general notion of privacy but what is acceptable depends on the individual person and cultural

attitudes. Critical aspects of a surveillance system include what data is available and in what form, who has access to data and in what form and how long it is stored. PrivacyCam by Chattopadhyay et al. [4] is a DSP based system where motion regions are encrypted before images are streamed. Dufaux et al. [6] also scramble regions of interest. They however do not rely on cryptography but do scrambling as part of MPEG-4 and MJPEG encoding. Baaziz et al. [2] not only do scrambling, but additionally embed watermarks into images to ensure data integrity. The system by Tansuriyavong et al. [14] performs face recognition and blanks silhouettes of persons. Depending on the configuration, only person's names, silhouettes of full images are displayed. Moncrieff et al. [10] apply dynamic data hiding techniques. While, during normal operation privacy sensitive data is removed, in case of, e.g., an alarm, the system is automatically adapted to reveal additional information.

Trusted Computing has attracted many researchers but little work is targeted at embedded systems. Grossmann et al. [8] describe a teletherapeutic application where vital parameters are monitored and an insulin pump is controlled from remote via a smartphone. A TPM is used to attest the system state before sensitive information is transmitted. secFleck by Hu et al. [9] is a TPM extension board for the Fleck mote. The TPM is used as random number generator, for RSA en- and decryption and digital signatures. No details about key management or key hierarchies are given. Software TPM implementations for embedded systems are explored by Dietrich and Winter [5]. Specifically, the use of existing CPU extensions like ARM TrustZone is evaluated to implement a software TPM with security guarantees similar to those of dedicated hardware. Alternatively, smart cards as found in mobile phones to could be used to implement TPM functionality. Aaraj [1] et al. also explore a software TPM solution. To improve performance, they implement critical functions on reconfigurable hardware.

The majority of existing literature on smart camera security is focused computer vision aspects such as identifying privacy relevant image regions. Little effort has been made to integrate such approaches with solutions from computer security research. We see our work as a first step towards a more holistic approach to ultimately fill this gap.

3. Trusted Computing Overview

TC is an industry initiative headed by the Trusted Computing Group (TCG). The main output of the group is a set of specifications for a hardware chip – the Trusted Platform Module (TPM) [15] – and software infrastructure like the TCG Software Stack (TSS) [16]. The TPM is typically implemented as a microcontroller (execution engine) with accelerators for RSA and SHA1. Additionally, the TPM provides a random number generator and limited amount of volatile and non-volatile memory. With an Opt-In process, users can choose if they want to make use of the TPM.

RSA keys can be generated for different purposes such as encryption or signing. Upon creation, keys can be declared migratable or not. While migratable keys can be transferred to a different TPM, non-migratable keys can not. Regardless of key type and migratability, a private TPM key can never be extracted from the chip as plaintext but only in encrypted form. By definition, every key must have a parent key that is used to encrypt the key when it has to be swapped out of the TPM due to limited internal memory. At the top of this key hierarchy is the Storage Root Key (SRK) which never leaves the TPM. TC defines three roots of trust:

Root of Trust for Measurement (RTM). In TC, measuring is the process of computing the SHA1 hash of an application binary before it is executed. Typically starting from an immutable part of the BIOS, a chain of trust is established where each component in the chain is measured before control is passed to it. The measurements are stored inside the TPM in memory regions called Platform Configuration Registers (PCRs). As available memory in the TPM is limited, a special operation called TPM_Extend is used to write to PCRs:

$$PCR[i] \leftarrow SHA1(PCR[i]||measurement).$$

TPM_Extend computes the hash of the current PCR value concatenated with the new measurement. This accumulated value is written back into the PCR.

Root of Trust for Reporting (RTR). Reporting of the platform state is called attestation and is done with the TPM_Quote command. As part of that, PCR values get signed inside the TPM using a key unique to that TPM. In theory, this key could be the Endorsement Key (EK) which is inserted into the TPM upon manufacturing. For privacy reasons however, not directly the EK but alias keys are used. They are called Attestation Identity Keys (AIKs) and are generated with the help of an external, trusted third party.

Root of Trust for Storage (RTS). The RTS allows to use the TPM to securely store data. Binding of data refers to encrypting data with a TPM key and hence guaranteeing that this data only is accessible by this specific TPM instance. Sealing of data allows to specify a set of PCR values the data is associated with. Like unbinding, unsealing can only be done by the specific TPM instance that holds the private sealing key. Additionally, the plaintext is only released if the current PCR values match those specified upon sealing.

4. Security and Privacy Protection

Before describing the TrustCAM prototype system in section 5, we discuss the design goals, underlying assumptions, setup procedures and targeted features.

4.1. Goals, Threats and Assumptions

The main class of threats we consider in this work are software attacks on camera systems. The motivation for this is that smart cameras come with a large amount of software. Often, standard operating systems such as Linux are used. Additionally, more and more systems have wireless interfaces. We believe that these properties make smart cameras an attractive target for attackers. Attacks on the camera hardware, including side channel attacks and hardware manipulation, are beyond the scope of this work. We however assume that with adequate effort, a reasonable degree of tamper resistance can be achieved by, e.g., specifically designed enclosures. One of the major advantages of TC is that private cryptographic keys are protected by the TPM and can never be exported. This allows to achieve a higher level of security than a pure software solution could. We use this property as a basis to realize the following set of security features, highly relevant for camera applications:

1. **Integrity.** Manipulation of images or videos should be detected, e.g., using checksums and digital signatures.
2. **Authenticity.** Evidence about the origin of images (i.e., which camera took an image) should be provided.
3. **Confidentiality.** To protect privacy, image regions that contain personal information should be encrypted.
4. **Multi-Level Access Control**. Different levels of abstraction for confidential data together with access control for each level should be supported.

Beyond these features, there is a wide range of security aspects that are equally important but are not considered in this work. These include *freshness* of image data and the related problem of image *timestamping*. Moreover, denial of service attacks and system *availability* are not addressed. Related work that discusses some of these aspects in the context of intelligent cameras can be found in [17].

4.2. System Setup and Camera Deployment

Figure 1 shows an example network of TrustCAM nodes operated by a central control station (CS). Each camera is equipped with an individual TPM called TPM_C. Likewise, the CS is equipped with a TPM denoted TPM_S. Furthermore, the CS runs a protected database where cryptographic keys generated during camera setup are securely stored.

For setup and deployment, we require that cameras are under full control of the operating personnel. As part of this setup, cryptographic keys are generated by the camera's TPM_C as well as by TPM_S of the control station. First, TPM_C must be activated by calling the TPM_TakeOwnership command. As part of this process, a unique owner secret is set and the Storage Root Key (K_{SRK}) is generated. K_{SRK} acts as parent key for the non-migratable, 2048 bit RSA signing key K_{SIG} which is generated in the second setup step. Being non-migratable guarantees that the private part of K_{SIG} can only be used inside TPM_C. Consequently, data signed with K_{SIG} is guaranteed to come from this specific camera. The public part of K_{SIG} is exported and stored in the CS database. Finally, in the third step, a number of non-migratable, 2048 bit RSA binding keys ($K_{BIND1...N}$) is generated by TPM_C of the CS. The public parts of these binding keys are exported and stored on the camera. During operation, they are used by the camera for encryption of privacy sensitive image data. As the private parts of the binding key can only be used inside TPM_C, this ensures that sensitive data can only be decrypted when having access to the CS.

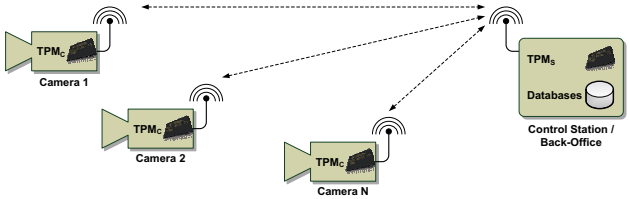

Figure 1. An network of trustworthy cameras and a control station.

4.3. Integrity and Authenticity

We achieve image integrity by cryptographically signing frames before they are streamed. For this, we use the non-migratable signing key K_{SIG} which is protected by TPM_C. The property of being non-migratable ensures that the private part of the key can only be used inside the camera's TPM_C. This solves the problem of image authentication as it guarantees that images signed with this key actually originate from the specific camera TPM_C is part of. Assume that the CS requests a video stream from camera CAM. Once the signed images arrive at the CS, the public signing key $K_{SIG_{pub}}$ from the CS database has to be retrieved. With this key, an operator can verify the image signature. In summary, the procedure works as follows:

1. CS: request video stream from camera CAM
2. CAM: acquire image data img from sensor
3. CAM: sign img: $Sig_{Res} = TPM\_Sign_{K_{SIG}}(img)$
4. CAM: send image signature Sig_{Res} and img to CS
5. CS: retrieve public signing key $K_{SIG_{pub}}$ of CAM from local database
6. CS: verify signature: $Verify_{K_{SIG_{pub}}}(Sig_{Res}, img)$
7. CS: If signature verification succeeds, one knows
   (a) that the image was not modified and
   (b) it comes from the intended camera as it was signed with K_{SIG} protected by CAM's TPM_C.

4.4. Confidentiality and Access Control

Privacy in video surveillance applications is a critical issue. To maintain privacy of monitored persons, relevant information such as faces or license plates need to be protected. While completely removing such information already on the camera would solve the privacy problem, at the same time it would significantly reduce the usefulness of a camera system for many tasks. As a consequence, our concept – in accordance with other proposals like [4, 14] – suggests to encrypt sensitive image regions to maintain confidentiality. We use the public binding keys $K_{BIND1...N_{pub}}$ which were created during camera deployment. By that, we not only achieve confidentiality but also an additional security property: Since the binding keys are non-migratable, they can not be extracted from TPM_S and hence can not be leaked to a third party. Consequently, actual access to the CS with its TPM_S is required for image decryption. We assume that CS access is limited to authorized staff.

Using multiple binding keys $K_{BIND1...N_{pub}}$ allows us to implement a multi-level security system. Suppose that sensitive image regions have been detected by the camera. In a next step, these regions are extracted from the image. The extracted regions then are bound to TPM_S using K_{BIND1} and K_{BIND2}. Furthermore, an abstracted version of the sensitive regions is generated (e.g., showing only outlines of persons) which is bound to TPM_S using K_{BIND3}. To be able to decrypt the image regions at the CS, not only access to the keys (and hence TPM_S) is required but also the keys' passwords (usage secrets) must be known. Depending on their security clearance, operators have knowledge of one of the usage secrets for K_{BIND1}, K_{BIND2}, K_{BIND3} or none. Consequently, they have access to different types of information (image without sensitive regions, abstraction of sensitive regions, ...). To reveal the full original image, in this model the two operators who know the usage secret for K_{BIND1} and K_{BIND2} respectively have to cooperate. Requiring two operators to collaborate provides a certain degree of protection against operator misuse. It however can not protect against attacks by collaborating insiders.

5. Implementation and Results

To evaluate the performance impact of the proposed security enhancements, we implemented them on our TrustCAM prototype. For the experiments, the control station is simulated with a laptop computer also equipped with a TPM. All communication between TrustCAM and the laptop is done via WiFi. In the following sections, we describe implementation details together with measurement results.

5.1. Hard- and Software Architecture

Our custom prototype system mostly uses commercially available components. Figure 2 gives an overview of the system architecture. TrustCAM is based on the BeagleBoard[1] which has a dual-core processor with an ARM Cortex A8 CPU clocked at 480 MHz and a TMS320C64x+ digital signal processor running at 360 MHz. The system is equipped with 256 MB RAM and 256 MB NAND flash. Via USB, we connect a color SVGA CMOS sensor (Logitech QuickCam Pro 9000) and an RA-Link RA-2571 802.11b/g WiFi adapter. An XBee radio provides a second, low-performance communication channel. Finally, an Atmel AT97SC3203S – the only commercial TPM designed for embedded devices – is connected to the mainboard via the I2C bus. Figure 3 shows a picture of the prototype system.

On the software side, we rely on an ARM Linux system with a customized kernel. Based on that, we have implemented a custom software framework that supports composition of applications from independent processing blocks. These blocks implement a predefined interface and perform a certain task such as motion detection or video streaming. Processing blocks are executed as individual threads that communicate via shared memory. The actual tasks performed by the system are defined by the selected processing blocks and how they are lined up. This high-level application logic, together with parameters for the processing blocks, is specified by application designers as a script. For application level TPM access, we use a modified versions of the TrouSerS[2] TCG software stack where we have replaced the trusted device driver library (TDDL) with a fully custom version that interacts with the TPM on the I2C bus.

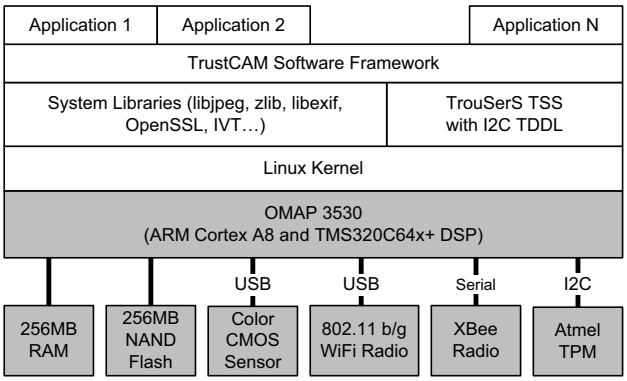

Figure 2. The TrustCAM prototype hard- (gray) and software (white) architecture. The image sensor and WiFi radio are connected via USB. The XBee low performance radio uses a serial connection and the TPM is attached via the I2C bus. The software layers consist of a custom Linux kernel, standard system libraries, a modified TrouSerS TSS and the TrustCAM software framework.

5.2. Image Encryption

As outlined in section 4, we propose to encrypt sensitive image regions. For that purpose we create two 256 bit AES

[1] BeagleBoard Website: http://www.beagleboard.org (03/2010)
[2] TrouSerS Website: http://trousers.sourceforge.net/ (03/2010)

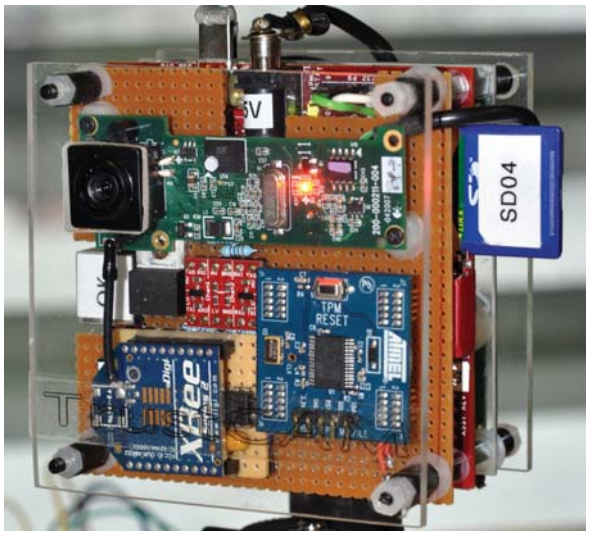

Figure 3. The TrustCAM prototype with the image sensor, the XBee radio and the Atmel I2C TPM at the top level. Behind that are the processing board and WiFi radio.

session keys, K_{AES1} and K_{AES2}, at application startup. They are bound to TPM_S of the control station. Specifically, K_{AES1} is bound using K_{BIND1} and K_{BIND2}:

$$K_{AES1_{bound}} = Bind_{K_{BIND1_{pub}}}(Bind_{K_{BIND2_{pub}}}(K_{AES1})).$$

The other symmetric key, K_{AES2}, is bound with K_{BIND3}:

$$K_{AES2_{bound}} = Bind_{K_{BIND3_{pub}}}(K_{AES2}).$$

Figure 4 depicts the processing flow of our camera. The image acquisition block reads an image from the sensor and passes it to the privacy protection block. The sensor can deliver either uncompressed YUYV or compressed JPEG images. The first operation performed by the block is detection of regions of interest (ROI). For evaluation purposes, we use ROI with fixed sizes. The ROI locations are computed randomly for every frame. In an actual implementation regions of interest could be determined using, e.g., motion detection or face detection. After the ROI have been selected, they are extracted from the input images. The remaining background image (IMG_{Back}) and the ROI image (IMG_{ROI}) are JPEG compressed using libjpeg. As intermediate level between revealing no or all sensitive image data, canny edge detection is performed on the ROI image. The resulting black and white, binary image (IMG_{Edge}) allows to, e.g., observe actions of persons in the video stream without revealing too much privacy sensitive information. IMG_{Edge} is compressed using zlib since JPEG compression is not efficient for binary images. The compressed ROI images are encrypted with the respective AES session keys:

$$IMG_{ROI_{ENC}} = Enc_{K_{AES1}}(IMG_{ROI}).$$
$$IMG_{Edge_{ENC}} = Enc_{K_{AES2}}(IMG_{Edge}).$$

Table 1 shows performance values for AES256 encryption measured on TrustCAM. Typical sizes of JPEG compressed regions of interest (200x200 pixels, JPEG quality: 80) are around 8 kB. For zlib compressed, edge-detected ROI, typical sizes are around 4 kB. Runtimes for AES encryption hence are less than 2 ms in both cases. After encryption, the background image IMG_{Back}, the encrypted ROI images $IMG_{ROI_{ENC}}$ and $IMG_{Edge_{ENC}}$ and the bound AES keys $K_{AES1_{bound}}$ and $K_{AES2_{bound}}$ are combined into a single image. For the prototype, they are embedded into IMG_{Back} as custom EXIF data.

	Data Size	Runtime	
		TrustCAM	Laptop
SHA1	15 kB	0.7 ms	0.09 ms
	40 kB	1.9 ms	0.3 ms
	80 kB	3.8 ms	2.1 ms
AES 256	8 kB	1.6 ms	0.2 ms
	15 kB	2.9 ms	0.3 ms
	40 kB	7.6 ms	0.7 ms
	80 kB	15.4 ms	1.4 ms

Table 1. Average runtimes (10 runs) for AES 256 encryption and SHA1 hashing on TrustCAM and a Core2 Duo (1.6 GHz) laptop.

5.3. Image Signing

The next step in the processing chain is the calculation of the digital signature of the combined image. Image signing is broken down into the computation of the SHA1 hash sum of the image and the digital signing of the hash value inside TPM_C. Table 1 presents typical runtimes for

	Atmel (LPC) (AT97SC3203)	Atmel (I2C) (AT97SC3203S)
TPM_OIAP	44 ms	47 ms
TPM_Sign	793 ms	804 ms
TPM_Unbind	827 ms	837 ms

Table 2. Average runtimes (10 runs) for selected TPM commands (2048 bit RSA key size) of Atmel TPMs on a PC (LPC bus) and on TrustCAM (I2C bus).

SHA1 calculation on TrustCAM. A JPEG compressed image at 640x480 (JPEG quality: 80) together with the embedded EXIF data has around 30 kB. The resulting runtime for SHA1 computation is less than 2 ms. Runtimes for the second set of operations involved in data signing, TPM_Sign and TPM_OIAP (TPM session establishment), are given in table 2. The runtimes on TrustCAM with its TPM connected to the I2C bus are about 850 ms. For comparison, we also measured the runtimes of an Atmel TPM in a desktop PC where it is connected to the LPC bus (LPC bus @33 MHz vs. I2C @50 kHz[3]). The difference of about

[3]Both, the Atmel I2C TPM and the BeagleBoard support 100 kHz I2C communication. For stability reasons, we only use 50 kHz.

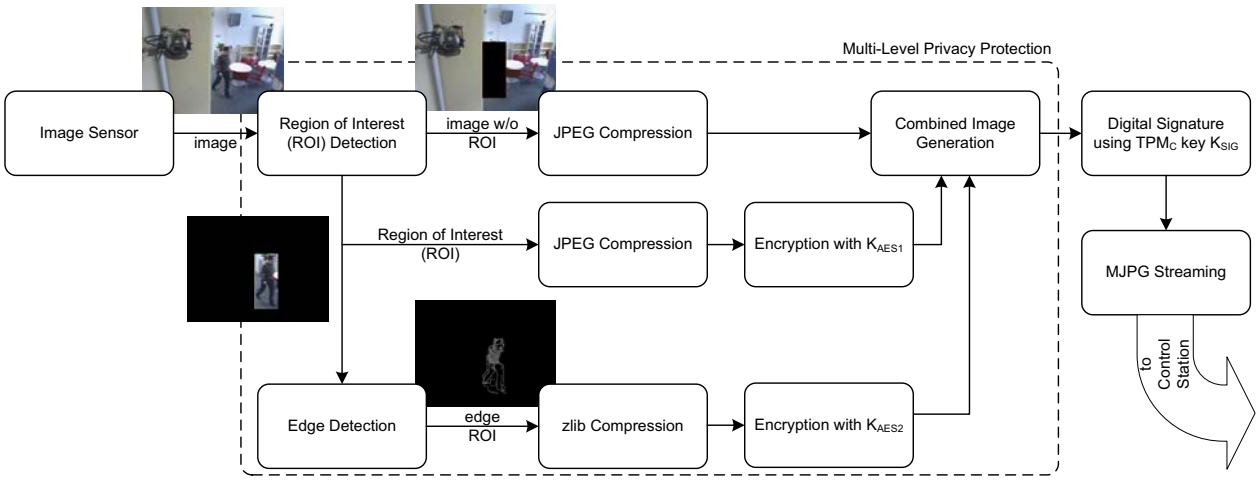

Figure 4. The TrustCAM onboard processing flow. Raw images are read from the sensor and regions of interest (ROI) are detected. Different privacy levels are achieved by, e.g., performing edge-detection for ROI. Afterwards, ROI images are JPEG compressed and encrypted with AES session keys bound to TPM_S of the CS. Combined images are signed with a key protected by the camera's TPM_C.

10 ms suggests that the major part of the time is consumed by the TPM and not by communication. Comparison measurements with TPMs from different manufacturers can be found in, e.g., [17]. For completeness, table 2 also includes the runtime for the TPM_Unbind command that is used at the control station for decryption of the AES session keys. Based on the runtimes, it clearly is impossible to, e.g., sign every frame of a video stream as this would reduce the framerate to little more than 1 fps. We consequently adapted the image signing procedure such that sequences of images instead of individual images are signed. We accumulate the hashes for a group of N frames in a way similar to the TPM's PCR_Extend operation:

$$AccSum_{Frm1\ldots x} = SHA1(AccSum_{Frm1\ldots(x-1)} || SHA1(Frm_x)).$$

The accumulated hash for N frames is signed by TPM_C:

$$Sig_{Res} = TPM\_Sign_{K_{SIG}}(AccSum_{Frm1\ldots N}).$$

As signature computation takes a significant amount of time, we do not wait for the result but continue with the processing and streaming of video data. Specifically, the accumulation of the hash sum for the next group of images (beginning at frame $N+1$) is started. This continuous operation is possible since TPM commands are executed in parallel to the main processor. Once the TPM completes the sign command, the signature is retrieved and attached to the next frame to be streamed. Note that the signature also contains the start and end indices of the group. For the prototype, signature data is embedded as custom EXIF data. At this point, the accumulated hash sum of the next image group is sent to the TPM for signing. The size of the image groups is automatically adapted to the current frame rate.

The presented approach allows to overcome the problem of low TPM performance. By signing groups of frames instead of individual images, we can deliver a video stream without interruptions from TPM operations. At the same time, the security of the system is not reduced. For videos that are stored and viewed at a later point in time, integrity of images can easily be verified. For live video, integrity guarantees for the currently displayed frame can not be given since the signature for the current image group is delivered at a later point in time. This delay is primarily determined by TPM performance and is less than 1 s for our system. Another limitation is, that if a frames of a group is lost, damaged or manipulated, the integrity and authenticity of the entire group can not be verified. We however believe, that these limitations are acceptable for most applications.

5.4. Control Station

At the control station, the streamed background images can be displayed without any further efforts. If however the ROI included in the EXIF data of the images should be displayed, they first need to be decrypted. For that purpose, the bound AES session keys $K_{AES1_{bound}}$ and $K_{AES2_{bound}}$ need to be unbound. As the private keys required for unbinding never leave TPM_S, this decryption requires actual access to the CS and its TPM_S. While for unbinding $K_{AES2_{bound}}$, the usage secret of K_{BIND3} is sufficient, for for $K_{AES1_{bound}}$ the secrets of two binding keys K_{BIND1} and K_{BIND2} are required. Assuming that these usage secrets are not known by a single system operator, this ensures that at least two persons have to cooperate to reconstruct the original image. As previously mentioned, this approach provides limited protection against insider attacks from single operators but can not prevent attacks by collaborating insiders. Once the AES session keys have been unbound, the ROI images $IMG_{ROI_{ENC}}$ and/or $IMG_{Edge_{ENC}}$ can be decrypted and inserted into the background image. For

Input Format Resol.	Type	Internal Format	Plain Streaming	Image Signing	ROI Encryption 100x100	ROI Encryption 200x200	ROI Encr. (200x200) + Image Signing
320x240	YUYV	Gray	24.4 fps	24.0 fps	23.6 fps	21.3 fps	20.6 fps
	JPEG		n/a	n/a	18.5 fps	13.8 fps	13.0 fps
	YUYV	RGB24	23.8 fps	23.2 fps	18.2 fps	12.6 fps	12.1 fps
	JPEG		25.0 fps	25.0 fps	14.5 fps	10.5 fps	10.0 fps
640x480	YUYV	Gray	12.8 fps	12.3 fps	11.8 fps	9.8 fps	9.6 fps
	JPEG		n/a	n/a	5.7 fps	5.2 fps	5.0 fps
	YUYV	RGB24	6.5 fps	6.2 fps	5.9 fps	5.2 fps	5.0 fps
	JPEG		25.0 fps	25.0 fps	4.5 fps	4.0 fps	3.8 fps

Table 3. Frame rates (avg. over 1000 frames) for different types of video streaming between TrustCAM and CS via WiFi. For plain streaming, JPEG frames are streamed or YUYV images are, after optional conversion to grayscale, JPEG compressed and streamed. For image signing, the TPM signature for an image group is computed. ROI encryption shows the frame rates if a region is extracted, encrypted and streamed together with the background. The last column shows the frames rates for combined ROI encryption and image signing.

Input Format Resol.	Type	Internal Format	JPEG Decomp.	Color Conv.	ROI Extract	JPEG Comp. Backgr.	JPEG Comp. Roi	zlib Comp. Edge	AES256 Encr. ROI	AES256 Encr. Edge	SHA1 Sign.	Total
320x240	YUYV	Gray	n/a	2.4 ms	9.7 ms	16.1 ms	9.8 ms	5.2 ms	1.6 ms	1.2 ms	1.0 ms	47.0 ms
	JPEG		24.7 ms	4.9 ms								74.2 ms
	YUYV	RGB24	n/a	6.6 ms	12.8 ms	31.8 ms	18.7 ms	5.3 ms	1.8 ms	1.2 ms	1.0 ms	79.2 ms
	JPEG		24.8 ms	n/a								97.4 ms
640x480	YUYV	Gray	n/a	8.8 ms	8.9 ms	63.9 ms	9.1 ms	3.9 ms	1.2 ms	0.9 ms	1.7 ms	98.4 ms
	JPEG		85.3 ms	18.7 ms								193.6 ms
	YUYV	RGB24	n/a	27.6 ms	11.5 ms	125.9 ms	17.8 ms	3.8 ms	1.3 ms	0.9 ms	1.9 ms	190.7 ms
	JPEG		85.6 ms	n/a								248.7 ms

Table 4. Average Runtimes (1000 frames) for the individual processing steps for a single frame. Included are JPEG decompression, color conversion, extraction, compression and encryption of the ROI (200x200 pixels) and finally SHA1 computation for the combined image.

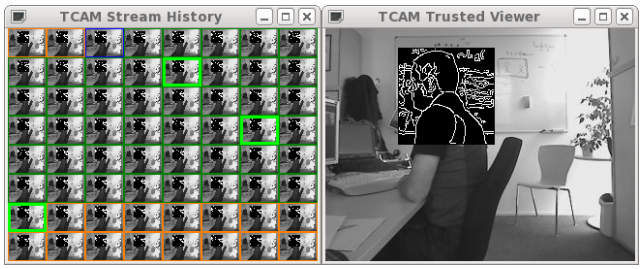

Figure 5. The live viewer at the CS. On the right is the current frame with the decrypted, edge-detected ROI. The left window shows the content of a circular buffer with the last 64 frames. Yet unverified frames have an orange border, verified ones have a dark green border. The last frame of a group has a bright green border.

the verification of the image signature, the control station computes the hash sum Sum_{Frm_x} for every incoming frame and stores it in a local cache. Additionally, the EXIF data of every incoming frame is checked for an image group signature Sig_{Res}. As the signature data also contains the start and end indices of the image group, the control station now computes the expected accumulated hash sum $ExpAccSum_{Frm1...N}$ for the N frames indicated by the start and end indices. It then loads the public signing key K_{SIG} that belongs to the expected camera from its local database and verifies the signature of the image group: $Verify_{K_{SIG_{pub}}}(Sig_{Res}, ExpAccSum_{Frm1...N})$. If verification is successful, one has assurance that (1) the images of the group were not modified and (2) the images of the group come from the expected camera. The live viewer of the prototype is shown in figure 5. It not only displays the current image, but also a history of recent frames together with their verification status. If authenticity and integrity of an image group could not be verified, a warning message is shown and playback is interrupted.

5.5. Performance Analysis

Table 3 shows the achieved frame rates for different streaming modes. For plain streaming, JPEG frames are read from the sensor and are directly streamed at 25 fps. Alternatively, YUYV frames are read, optionally converted to grayscale, JPEG compressed and then streamed. Compared to native JPEG images where no compression in software is required, frame rates are between 24.4 and 6.5 fps depending on resolution and color depth. If images are digitally signed, frame rates decrease by only about 0.5 fps compared to plain streaming. This relatively small impact is achieved by signing image groups and the fact that signature computation runs on the TPM in parallel to the ARM CPU.

The *ROI encryption* column presents the frame rates if regions of interest (100x100 or 200x200 pixels) are extracted and encrypted. If YUYV images are read from the sensor which then are converted to grayscale, frame rates

are reduced between 1 and 3 fps compared to plain streaming depending on resolution and ROI size. If no conversion to grayscale is done, frame rates are reduced by up to 11 fps (resolution: 320x240, ROI size: 200x200). Table 4 explains this observation by giving detailed performance numbers for the individual processing steps for a single frame. JPEG compression of the ROI in this case takes 18.7 ms with additional 5.3 ms for zlib compression of the binary ROI. This time is relatively high compared to the runtime for JPEG compression of the full-size background image (31.8 ms).

In comparison, AES encryption of ROI images and SHA1 computation of output images have runtimes of less than 2 ms each. While the direct performance impact by these security relevant functions is acceptable, the overall performance still degrades considerably. This primarily results from the additional effort required for extraction and compression of ROI images as clearly illustrated by table 4. Improvements could be made by, e.g., deriving regions of interest directly from the JPEG images delivered by the sensor as proposed in [4]. If however uncompressed images are required for further processing on the camera, an alternative would be to read YUYV images from the sensor and use the DSP for image compression.

6. Conclusions and Future Work

In this work we presented TrustCAM, an embedded smart camera platform equipped with a TPM. We implemented and evaluated digital signing of video streams to achieve image integrity and authenticity. Additionally, to preserve privacy of monitored people, we encrypt regions of interest using cryptographic keys protected by a TPM only accessible by camera operators. The performance impact of cryptographic operations is relatively small. The additional overhead for ROI processing and compression however reduces overall system performance. Future work therefore will include the integration of the Digital Signal Processor for JPEG compression and image processing. As part of that, more advanced codecs for video streaming than Motion-JPEG could be integrated as well. The potential use of a TPM in a camera system is not limited to image integrity, authenticity and privacy protection. In ongoing work we will extend our prototype to support integrity reporting of the full software base of the camera. Moreover, image timestamping is an important application, e.g., in traffic monitoring, where the TPM can be used to provide evidence when an image was taken.

Based on our results and experience, we believe that enhancing smart camera security with Trusted Computing is a viable and promising approach. Readily available Trusted Platform Modules are a cheap, solid and widely reviewed basis for custom security solutions. Issues like low TPM performance can be handled if the TPM computations are properly integrated into the processing chain.

References

[1] N. Aaraj, A. Raghunathan, and N. K. Jha. Analysis and Design of a Hardware/Software Trusted Platform Module for Embedded Systems. *ACM Transactions Embedded Computing Systems*, 8(1):1–31, 2008. 2

[2] N. Baaziz, N. Lolo, O. Padilla, and F. Petngang. Security and Privacy Protection for Automated Video Surveillance. In *Proceedings of the IEEE Int. Symposium on Signal Processing and Information Technology*, pages 17–22, 2007. 2

[3] A. Cavallaro. Privacy in Video Surveillance. *IEEE Signal Processing Magazine*, 24(2):168–166, March 2007. 1

[4] A. Chattopadhyay and T. Boult. PrivacyCam: A Privacy Preserving Camera Using uCLinux on the Blackfin DSP. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2007. 2, 4, 8

[5] K. Dietrich and J. Winter. Implementation Aspects of Mobile and Embedded Trusted Computing. In *Proceedings of the Conference on Trusted Computing*, pages 29–44, 2009. 2

[6] F. Dufaux and T. Ebrahimi. Scrambling for Video Surveillance with Privacy. In *Proceedings of the Computer Vision and Pattern Recognition Workshop*, pages 160–166, 2006. 2

[7] S. Fleck and W. Strasser. Smart Camera Based Monitoring System and Its Application to Assisted Living. *Proceedings of the IEEE*, 96(10):1698–1714, 2008. 1

[8] U. Grossmann, E. Berkhan, L. C. Jatoba, J. Ottenbacher, W. Stork, and K. D. Mueller-Glaser. Security for Mobile Low Power Nodes in a Personal Area Network by Means of Trusted Platform Modules. In *Proceedings of the Workshop on Security and Privacy in Ad hoc and Sensor Networks*, pages 172–186, 2007. 2

[9] W. Hu, P. Corke, W. C. Shih, and L. Overs. secFleck: A Public Key Technology Platform for Wireless Sensor Networks. In *Proceedings of the 6th European Conference on Wireless Sensor Networks*, pages 296–311, 2009. 2

[10] S. Moncrieff, S. Venkatesh, and G. A. W. West. Dynamic Privacy in Public Surveillance. *IEEE Computer*, 42(9):22–28, Sept. 2009. 2

[11] A. Senior, S. Pankanti, A. Hampapur, L. Brown, Y.-L. Tian, A. Ekin, J. Connell, C. F. Shu, and M. Lu. Enabling Video Privacy through Computer Vision. *IEEE Security & Privacy Magazine*, 3(3):50–57, May/June 2005. 1

[12] D. N. Serpanos and A. Papalambrou. Security and Privacy in Distributed Smart Cameras. *Proceedings of the IEEE*, 96(10):1678–1687, October 2008. 1

[13] S. Soro and W. Heinzelman. A Survey of Visual Sensor Networks. *Advances in Multimedia*, 2009:1–21, May 2009. 1

[14] S. Tansuriyavong and S. Hanaki. Privacy Protection by concealing Persons in circumstantial Video Image. In *Proceedings of the Workshop on Perceptive User Interfaces*, pages 1–4, 2001. 2, 4

[15] Trusted Computing Group. TCG Software Stack Specification (TSS) Version 1.2, Level 1, Errata A, March 2007. 2

[16] Trusted Computing Group. TPM Main Specification Version 1.2, Level 2, Revision 103, July 2007. 2

[17] T. Winkler and B. Rinner. Applications of Trusted Computing in Pervasive Smart Camera Networks. In *Proceedings of the Workshop on Embedded System Security*, 2009. 3, 6