Applications of Trusted Computing in Pervasive Smart Camera Networks

Thomas Winkler
Pervasive Computing / Institute of Networked and Embedded Systems (NES)
Lakeside Park B02b
9020 Klagenfurt, Austria
thomas.winkler@uni-klu.ac.at

Bernhard Rinner
Pervasive Computing / Institute of Networked and Embedded Systems (NES)
Lakeside Park B02b
9020 Klagenfurt, Austria
bernhard.rinner@uni-klu.ac.at

ABSTRACT

Pervasive Smart Cameras are embedded computer vision systems bringing together the research areas of smart cameras and wireless sensor networks. Besides traditional areas like video surveillance or traffic monitoring, small, cheap and powerful camera systems open a wide range of potential new applications including assisted living, home automation or entertainment. At the same time, widespread deployment of cameras introduces several security challenges. Using wireless networking and being mounted at remote locations, smart cameras are an attractive target for attackers. Another issue of crucial importance when it comes to the acceptance of camera systems is user privacy. In this work we explore the use of Trusted Computing concepts to enhance security of an experimental smart camera system. Additionally, we discuss required and achievable performance based on evaluations on our prototype platform.

1. INTRODUCTION

Cameras are present in a variety of situations of our life. In traffic monitoring applications they are used to detect traffic jams or accidents [3]. Video surveillance is performed in public places like airports or train stations. Even in private environments, applications like assisted living [9] are emerging where cameras are used to monitor the daily life of individuals.

With advances in technology, cameras have become smart and ubiquitous [14, 13]. By definition, a smart camera is an embedded system that consists of an image sensor, a processing and a communication unit. Captured image data is analyzed on the camera and only the results of this analysis are delivered to a control station. This can be statistical information like the number of persons in a room or an alarm if some unusual event like a fallen person or unattended luggage is detected. While smart cameras are designed to typically deliver events instead of video streams, there still is the requirement to occasionally transmit images. This way, operating personnel can e.g. more accurately evaluate a reported event before taking further action.

When it comes to system security and privacy of monitored individuals, smart camera systems raise new challenges. Contrary to conventional systems, smart cameras come with a relatively large software stack. This typically includes a flexible and powerful embedded operating system like uClinux as well as a variety of applications running on top of it. Additionally, more and more cameras are equipped with wireless networking interfaces. These facts make them attractive targets for potential attackers. The operator of a camera system has high interest to gain assurance that the software stack of the camera was not modified. Additionally, in applications like law enforcement, proof might be required that certain information actually originates from a specific camera and was captured at a certain point in time. In situations where the camera is tasked to store or deliver images, monitored individuals have a high interest that their privacy is protected.

In this work we address selected security and privacy issues in the context of smart camera systems by exploring the possibility of adding a hardware security chip to our prototype system. Specifically, we discuss how Trusted Computing (TC) and a Trusted Platform Module (TPM) could be used to record and report the state of a platform, provide evidence that data is coming from a certain camera and ensure that privacy relevant information only is accessible by authorized persons. Additionally, we evaluate the suitability of existing TPM solutions in the context of our application domain.

The remainder of this paper is organized as follows: Section 2 presents related work on security and privacy aspects in video surveillance as well as approaches to use TPMs in the domain of embedded systems. It is followed by a brief overview of Trusted Computing fundamentals in section 3. Subsequently, section 4 outlines the architecture of our prototype system including the camera hard- and software as well as the monitoring station. Section 5 presents how selected TC concepts can be applied to smart cameras. A discussion how they can be implemented in an embedded system together with evaluation results is presented in section 6. Section 7 concludes the paper and outlines potential future work.

2. RELATED WORK

Aspects of privacy and security in camera applications have been addressed in a number of related publications

WESS'09, October 15, 2009, Grenoble, France.

which are covered in section 2.1. Subsequently, section 2.2 concentrates on related work that explores the applicability of Trusted Computing in embedded systems.

2.1 Security and Privacy in Camera Networks

Serpanos et al. [18] provide an extensive discussion of security and privacy related issues in smart camera networks. They discuss the need for confidentiality, integrity and freshness of data transmitted between nodes. In cases where images are transmitted, privacy of observed persons is a critical issue as it not only involves protection of sensitive information against external attackers but also against legitimate system operators. To achieve this goal, relevant parts of the images need to be recognized and appropriately encrypted.

Senior et al.[17] discuss the meaning of privacy in the context of video surveillance and conclude that there is no general notion of privacy but what is acceptable depends on the individual person and also on cultural attitudes. They continue with a discussion of critical aspects of a surveillance system including what data is available and in what form (e.g. raw images vs. metadata), who has access to data and in what form (e.g. plain vs. encrypted) and how long it is stored. Finally, the authors present a concept for a system that preserves user privacy by pre-processing videos on the camera and a layered approach for granting access to the different types of information produced by the camera.

PrivacyCam [4] by Chattopadhyay et al. is a camera system based on a Blackfin DSP clocked at 400 MHz, 32 MB of SDRAM and an Omnivision OV7660 color CMOS sensor. uClinux is used as operating system. Regions of interest are identified based on a background subtraction model and resulting regions are encrypted using an AES session key.

Dufaux et al. [8] also follow the approach of scrambling regions of interest. They however do not rely on conventional cryptographic algorithms but integrate the content scrambling into the MPEG-4 and MJPEG encoding processes.

A similar approach is discussed by Baaziz et al. [2] where in a first step motion detection is performed followed by content scrambling. To ensure data integrity, an additional watermark is embedded in the image which allows to detect manipulation of image data. Limited reconstruction of manipulated image regions is possible due to redundancy introduced by the watermark.

Tansuriyavong et al. [21] present a system used in an office scenario that blanks the silhouettes of persons. Additionally, the system integrates face recognition to identify previously registered persons. The system can be configured what information should be disclosed - full images, silhouettes, names or any combination of that.

2.2 Embedded Trusted Computing

The Low Pin Count (LPC) bus that is used to connect TPMs to the PC platform, typically is not available on embedded systems. Some manufacturers additionally equip their TPMs with a serial, two-wire interface making them suitable for embedded systems. Grossmann et al. [10] demonstrate the use of an Atmel AT97SC3203S TPM together with an MSP430 microcontroller from Texas Instruments in the context of a teletherapeutic application. In this scenario, the software state of the embedded device is attested using the TPM before sensitive information is transmitted. The authors also give measurement results for runtime and energy consumption for selected TPM commands. The TPM Quote operation e.g. is measured to take 800 ms during which a current of 39 mA (@ 3.3 V) is drawn.

For secFleck, Shih et al. [11] mount an Atmel TPM on an extension board for the Fleck mote platform which is powered by an Atmel Atmega128 running at 8 MHz. Apparently, the TPM is not used for platform attestation but only as random number generator, for RSA en- and decryption and signature creation and verification. The authors claim that *the* public TPM key is used in these operations which however is not directly possible with the TPM's Endorsement Key (EK). No details are provided on key management or key hierarchy. secFleck is also used by Dua et al. [6] to enhance security of a participatory sensing application where users sense their local environment and make these measurements available to other users. The TPM is used to attest the integrity of the users platforms. In the proposed protocol the PCRs are signed directly using the EK. Again, this violates the TPM specification and therefore should not be possible with a compliant TPM chip. In another work by the same authors [7], a similar approach for trustworthy sensing is discussed. A peripheral platform equipped with a TPM, sensors and bluetooth is used for sensing. A second platform, e.g. a mobile phone, can query the device via bluetooth and in turn receives sensed data signed by the peripherals TPM. Details on how data is verified or how a verifier gains the assurance that data is coming from a TPM protected platform are not discussed in this work.

Aaraj et al. [1] evaluate the performance of a pure software TPM on an embedded platform (Xscale PXA-250 at 400 MHz and 32 MB RAM). They present runtime measurements for a wide range of TPM commands including TPM Quote (1239 ms with a 2048 bit RSA key) and TPM Sign (902 ms for an RSA key length of 2048 bits; 343 ms for an RSA key length of 1024 bits). Based on these results, the authors replaced RSA with elliptic curve cryptography (ECC) which reduced the time for TPM Quote to 381 ms (key length 224 bits) and TPM Sign to 191 ms (data size 20 Bytes, key length 224 bits). On average, execution time was reduced by a factor of 6.5. Note however that ECC is not supported by the current TPM specification but may be adopted in future versions. Based on detailed performance analysis, the authors implemented special hardware instructions to accelerate ECC. For this, they used the Xtensa platform from Tensilica with the base processor operating at 320 MHz. On this platform the unoptimized TPM Quote operation using a 224 bit ECC key takes 169.81 ms. With hardware optimizations this time was reduced to 84.154 ms on an uni-core system and to 30.70 ms on a six-core system.

Other researchers like Dietrich and Winter [25, 5] also evaluate the possibility of using software based TPM implementations for embedded systems. Many embedded systems already come with integrated security functionality like ARM TrustZone that can be used to develop software TPM solutions. The same authors explore the use of Smart Cards or SIM cards as found in mobile phones to implement TPM functionality. Research on software TPM implementations is still in early stages but preliminary results suggest that they might be able to provide security levels similar to those of hardware TPMs.

3. TRUSTED COMPUTING OVERVIEW

Trusted Computing (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) [22] – and surrounding software infrastructure like the TCG Software Stack (TSS) [23]. The TPM, as shown in figure 1, typically is implemented as a microcontroller (execution engine) with accelerators for RSA and SHA1. Additionally, the TPM provides a random number generator as well as 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 chip.

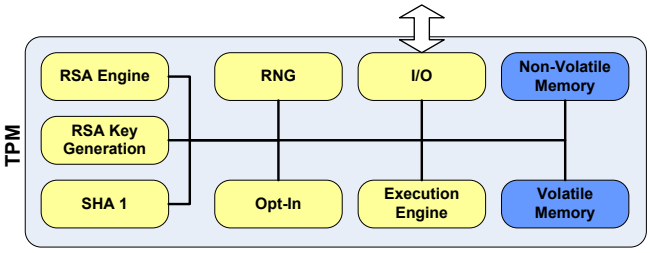

Figure 1: A Trusted Platform Module (TPM) consists of shielded locations (memory) and protected capabilities which are functions that operate on shielded locations.

RSA keys can be generated for different purposes like data 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 is required to 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 root 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. 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 the amount of memory inside the TPM is limited, a special operation called TPM Extend is used when writing to PCRs:

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

With the extend operation, the current PCR value is not overwritten but the new measurement is accumulated with the current PCR value.

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 the data only is accessible by this specific TPM instance. Sealing of data allows to specify a set of PCR values the data is sealed to. As with binding, the 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. SYSTEM ARCHITECTURE

Before discussing how Trusted Computing concepts can be applied in smart camera applications, we are going to outline the building blocks and architecture of a prototypical camera network. In the context of this work, we differentiate two major components: (1) The individual camera nodes and (2) the back office where the camera network is controlled and the information from the cameras is collected and evaluated.

4.1 Camera Prototype

For experimentation and evaluation purposes we have built an embedded, low-power camera system based on off-the-shelf components. To simplify application development, we implemented a custom middleware system that allows to compose image processing applications from simple, re-usable components.

4.1.1 Hardware Architecture

Figure 2 shows our pervasive smart camera platform. The platform is based on the BeagleBoard[1] equipped with an OMAP 3530 processor from Texas Instruments. The processor is based on an ARM Cortex-A8 clocked at 480 MHz and an additional TMS320C64x+ digital signal processor running at 430 MHz.

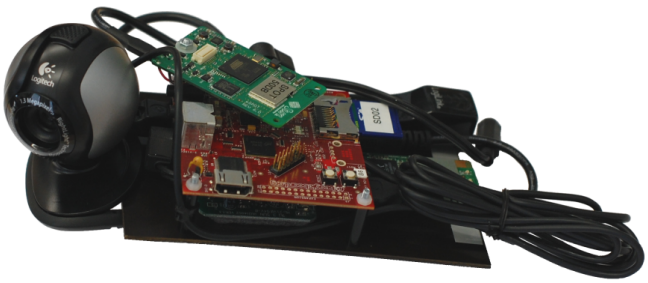

Figure 2: A Pervasive Smart Camera prototype with an embedded processing board, a webcam, an 802.11 radio and a SunSPOT for 802.15.4 connectivity.

The system provides 128 MB RAM and 256 MB NAND flash. Peripherals can be attached via USB, I2C, SPI, DVI as well as stereo in/out. In our setup, USB is used to connect a Logitech QuickCam S5500 (color, VGA), an RA-Link RA-2571 802.11b/g WiFi adapter as well as a SunSPOT [19] mote used for 802.15.4 wireless connectivity. For development and debugging purposes, the nodes additionally are equipped with USB to Ethernet adapters. As operating system a Debian GNU/Linux distribution compiled for the ARM platform and an OMAP specific kernel are used.

[1]BeagleBoard Website: http://www.beagleboard.org (Aug. 2009)

In visual sensor networks, power consumption is an important aspect. WiFi allows us to occasionally transmit video streams which can be useful to inspect and evaluate events reported by the camera network. During normal operation however, where only small amounts of data are exchanged between cameras for control and coordination, WiFi is too power intensive. For that reason, in [24] we have proposed the approach of equipping our cameras with an additional 802.15.4 radio. The resulting dual radio network allows us to trade communication performance for power consumption based on the actual requirements of the application.

Our prototype system currently is not equipped with a hardware TPM. To still be able to evaluate the concepts proposed in this paper, we are using the TPM emulator by Strasser et al. [20]. The camera's TPM subsequently is called TPM_C.

4.1.2 Camera Software Framework

To simplify application development and to allow re-use of components, a software framework has been designed that supports composition of applications from individual blocks which are instantiated and interconnected at runtime. The selected approach for the middleware framework follows the concept of modeling the dataflow between the individual components.

Conceptually, every block has an output memory where its results can be accessed by subsequent blocks. To maintain consistency of stored data, access to the memory is guarded by a lock that is passed between the producing and consuming block similar to a token. Blocks can form chains of arbitrary length where each pair of blocks is connected by shared memory and a lock. In our implementation a processing block is realized as an individual operating system process expecting well-defined input data and generating output consumable by subsequent blocks. The shared memories are implemented as POSIX shared memory synchronized by an inter-process locking mechanism.

Using separate processes instead of threads for the processing blocks offers a number of benefits. Blocks can be implemented in any programming language as long as there exists shared memory and locking support. This allows to e.g. use native code in places where performance is critical such as low-level data processing and to rely on higher level languages for e.g. statistics generation, system configuration and networking aspects. Moreover, separate processes allow to more easily implement watchdog functionality that monitors individual parts of the processing chain and restarts blocks as required.

Processing blocks do not directly support multiple consumers for their output memory. To overcome this limitation, a central entity running on every camera node called the *NodeManager* is introduced. Once a block has written its data to its output memory, it passes the lock to the *NodeManager* who in turn gives the lock to all registered consumer blocks which then can perform parallel read access to the memory. Once all consumer blocks have returned the lock, the *NodeManager* passes the lock for the shared memory to the producer block which can now fill it with new data. Note that the producer block does not necessarily have to be idle while not holding the memory lock but it can continue with internal processing as required. As shown in figure 3, there exists exactly one *NodeManager* per camera. Per definition, the *NodeManager* not only is responsible for lock management but also is the only entity that creates new block instances. This allows it to keep track of running blocks and their connections. Additionally, the *NodeManager* monitors the available system resources and can decide whether creation of additional blocks is allowed or not.

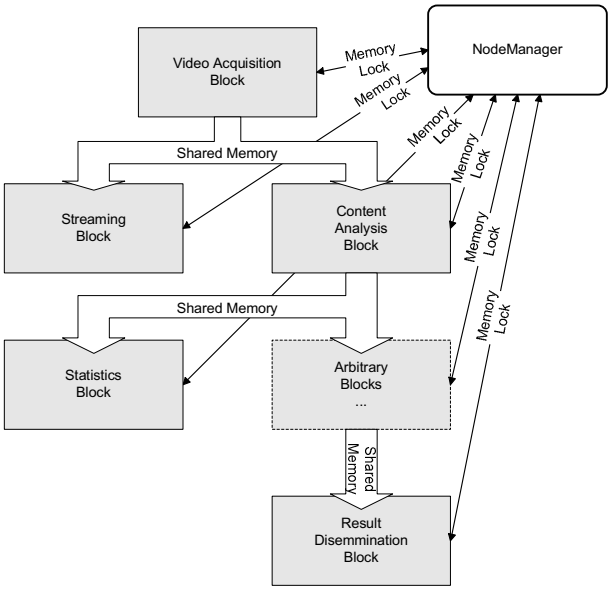

Figure 3: The *NodeManager* is responsible for creating processing chains and lock management. The output of individual blocks is stored in shared memory that can be accessed by one or more consumers. Final results of processing chains typically are made available as a subscribable service consumed by other nodes or client applications.

Additional details and performance evaluations for the camera software framework can be found in [16].

4.2 Back-Office

The back-office is the location from where the camera network is operated and controlled. It is only accessible by operating personnel with adequate security clearance. In the back-office, there is dedicated computing infrastructure that hosts a database storing key material generated during the camera setup procedure (see section 5.1) as well as data received as periodic trusted lifebeats from the cameras (described in section 5.3). To associate trusted lifebeat events with UTC time, the control station is assumed to have a reliable time source. The control station is also equipped with a TPM called TPM_S. This allows to implement functionality that ensures that images delivered by a camera can only be accessed at the control station.

Beyond the mentioned requirements, the back-office and its infrastructure is not further discussed. In the context of this work it is considered as a secure, trusted facility.

5. TRUSTED COMPUTING INTEGRATION

Before presenting details on how TC can be integrated into a smart camera network, we are going to discuss the scope of the presented concepts as well as listing certain assumptions we have made. The overall system is run by an operating agency that controls the setup process of cameras as well as the central control station infrastructure. For

asserting the state of a camera, the operators can rely on information gathered during the setup process of the camera. In the current version of the system we do not include the option for external users to assert the state of a camera. We therefore currently can also omit a trusted third party or similar concepts that would be needed in such a case. We however intend to address this in future work.

Subsequently we are now going to describe the process how new cameras are prepared for deployment. In section 5.2, we outline a concept for establishment of a chain of trust on an embedded system. In section 5.3 we describe a trusted lifebeat process that allows system operators to periodically check the cameras software state. Thereafter, we describe in section 5.4 how data delivered by a camera can be checked for its origin and how privacy sensitive image regions can be protected ensuring they only are accessible at the control station (section 5.5).

5.1 Camera Setup

Before camera nodes are deployed, they need to be properly set up. It is assumed that this setup is done when the cameras are under full control of the operating personnel. The main part of the setup procedure is the generation of TPM keys on the camera and at the control station. All keys are generated as RSA keys with a length of 2048 bits. The setup procedure involves the following steps:

TPM Ownership. Calling the TPM_TakeOwnership operation of the cameras TPM_C sets an owner secret and generates the Storage Root Key K_{SRK}. The owner secret is not required during normal operation of the camera and is set to a random value unique to every camera. For maintenance operations, the cameras owner secret is stored in the database of the control station. Contrary to a desktop PC, there are not multiple users per camera. Hence, no user storage keys are generated but all signing, binding and sealing keys are created with K_{SRK} as their parent.

Identity Key Creation. An Attestation Identity Key serves as an alias for the Endorsement Key (K_{EK}) and is used during platform attestation. In the application context of a visual sensor network, contrary to a conventional PC, there are not multiple users per camera. In fact there is only the system software running on each camera taking the role of a single system user. Moreover, all cameras in the network are uniquely identified and well known by the operators. Consequently, there is no need for the anonymity gained by using multiple AIKs in conjunction with a PrivacyCA. Therefore, only a single Attestation Identity Key K_{AIK} is generated during setup that serves for platform attestation and optionally for certifying other TPM keys. The public porting $K_{AIK_{pub}}$ is stored in the back office database together with $K_{EK_{pub}}$.

Compared to usage scenarios on a PC with multiple users, our application context is a lot less generic. Cameras deployed in the network are known in advance and there are no human users actively using the cameras. As user anonymity is not an issue, even K_{EK} which uniquely identifies a TPM and hence the system it belongs to, could be used as signature key during Quote operations. This however is not supported by the TPM specification.

Note that user anonymity in this context must not be mistaken with privacy of persons monitored by the cameras. This aspect is discussed in section 5.5.

Signature Key Creation. For signing data coming from a camera like events or images, a non-migratable singing key K_{SIG} is created with K_{SRK} as its parent. Being non-migratable ensures that the private key can not leave the cameras TPM_C and can only be used inside this specific TPM_C. This provides assurance that data signed with this particular key really originates from this specific camera.

Encryption Key Creation. To ensure privacy of monitored persons, images delivered by the camera to the control station have to be encrypted. This encryption can be done for full images or special regions of interest where e.g. motion or faces have been detected.
A non-migratable binding key K_{BIND1} is created by the control station's TPM_S. The public portion of this key, $K_{BIND1_{pub}}$, is exported from TPM_S and stored on the camera. Note that the private part of K_{BIND1} can not be exported from TPM_S and therefore, encrypted data delivered by the camera can only be decrypted at the control station and not e.g. by an intermediate attacker who interferes with the transmission. To decrypt data bound with $K_{BIND1_{pub}}$, the usage secret of the key has to be supplied by the operator. To avoid that a single person with access to the control station and knowledge of this usage secret can decrypt data, additional binding keys can be introduced. This opens the possibility to use two or more keys to encrypt privacy sensitive data and hence two or more operators have to cooperate to decrypt the data. Moreover, different security levels can be associated with different keys. Depending on the security level, only certain data is accessible. This is explored in section 5.5.

Table 1 summarizes the cryptographic keys generated as part of the camera setup procedure.

	Control Station	**Camera**
Endorsement Key	$K_{EK_{pub}}$	K_{EK}
Storage Root Key	-	K_{SRK}
Attestation Identity Key	$K_{AIK_{pub}}$	K_{AIK}
Signature Key	$K_{SIG_{pub}}$	K_{SIG}
Binding Key	K_{BIND1}	$K_{BIND1_{pub}}$
Add. Binding Keys	$K_{BIND2...N}$	$K_{BIND2...N_{pub}}$

Table 1: The key material generated as part of the camera setup procedure. All keys are non-migratable, 2048 bit RSA keys. The binding keys are generated by the control station's TPM_S while all other keys are generated by the cameras TPM_C. The *pub* subscript denotes the public RSA key.

5.2 Chain of Trust on Cameras

For this work, we assume there is a hardware TPM or a software TPM with similar security and usage characteristics available on the cameras. However, as mentioned in section 4.1.1 our camera prototype is not equipped with such a TPM solution. For the implementation of application level TPM usage we therefore use the TPM emulator [20]. With

this setup, boot up measurements and the establishment of a chain of trust can not be fully implemented. In this section we nevertheless outline the concepts and individual steps required for trusted boot of a camera node.
It is assumed that an immutable Root of Trust for Measurement (RTM) is available on the system. This RTM measures the uBoot bootloader which then measures (1) the Linux kernel, (2) the kernel command line and (3) the basic software image. The basic software image, a Linux root file system, is implemented as read-only, compressed squashfs file system. This file-system includes system libraries, the basic camera software framework including the NodeManager, required image processing libraries and the TrouSerS [12] TCG software stack. After measuring these three components, the bootloader passes control to the Linux kernel.
Contrary to our approach, on a generic PC platform measuring the entire root file-system typically is not an option for two reasons: (1) The root file-system is too large to be completely measured and (2) the measurement is of little value because the file-system is not read-only and binaries can get modified after being measured. With the Integrity Measurement Architecture [15], a Linux kernel extension is available that adds functionality to measure every binary executed by the system. In case of an embedded system however, root file-systems often are small in size (a few to a few dozens of MB) and write access to the file-system is not required. These characteristics allow us to measure all major components while only generating a very small set of PCR values. Contrary to a general system, this small set can easily be validated.

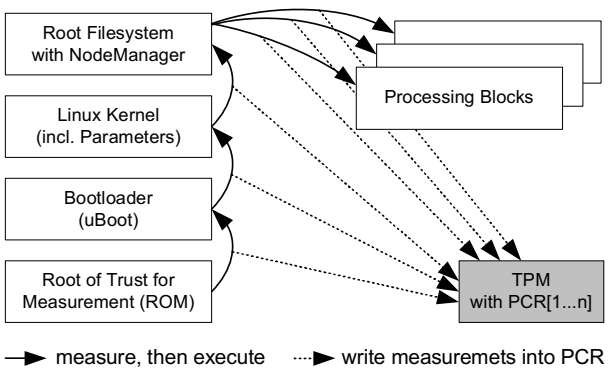

Figure 4: The camera's chain of trust starts from a static Root of Trust for Measurement and is extended to the bootloader, the Linux kernel and its command line and the read-only root file-system.

As outlined in section 4.1.2, the NodeManager is a central component on every camera. It allows to start applications consisting of multiple independent blocks forming an image processing chain. Measuring the root file-system allows a verifier to get information about the software base available on the platform but it does not provide information which image processing tasks are running on the platform. To facilitate that, the NodeManager measures started processing blocks into predefined PCRs. The description that defines which blocks are started, what their parameters are and how they are interconnected is also measured. This configuration, together with the processing blocks defines the actual behavior of the camera. By measuring these components separately, a verifier can not only get information on the software base but also gain an insight which image processing tasks actually are executed by the camera.

5.3 Trusted Lifebeat

The main purpose of a lifebeat is to determine the state of a system based on the periodic transmission of status messages. If a status message is not received for a predefined amount of time, then it has to be assumed that the system is no longer operational. The trusted lifebeat mechanism proposed for our visual sensor network is based on this concept and extends it such that information about the current state of a camera is included in the lifebeat message. At its core, this trusted lifebeat is based on TCG's remote attestation concept. Contrary to a conventional lifebeat, in our architecture the lifebeat is not automatically sent by a client (i.e. a camera) but periodically requested by the control station. This is done to supply a nonce to ensure freshness of the platform attestation information contained in the lifebeat.

The information returned by the camera is not limited to the state of the platform but also includes the value of the TPM's internal tick counter. This tick counter is initialized upon TPM startup with a session nonce TSN that uniquely identifies the tick session. The tick value TSV is reset to 0 upon TPM startup. The number of μs per tick is given by the tick rate $TRATE$. Inclusion of signed tick values in the periodic lifebeat messages is the foundation for associating timestamped images with UTC time (see section 5.4).

In detail, the lifebeat mechanism is defined as follows:

1. The control station sends a nonce n and the list of requested PCRs to a camera. Additionally, the control station records the current UTC time t_0.

   If the camera does not respond within a predefined amount of time, it is considered to be out of service and should be retrieved for inspection.

2. The camera performs a TPM TickStampBlob operation resulting in:

   $$TickStamp_{Res} \leftarrow TPM\_TickStampBlob_{K_{SIG}}(n||TSN_{LB}||TSV_{LB}||TRATE_{LB})$$

   TSV_{LB} is the current tick value, TSN_{LB} identifies the tick session with a unique nonce and $TRATE_{LB}$ is the number of microseconds per tick.

3. Then, the camera performs a TPM Quote operation and generates:

   $$Quote_{Res} \leftarrow TPM\_Quote_{K_{AIK}}(PCRs||TickStamp_{Res}).$$

   Note that $TickStamp_{Res}$ is included in the signature instead of n supplied by the control station. n however is implicitly included as it is part of $TickStamp_{Res}$. Including $TickStamp_{Res}$ associates tick count and platform state. This provides the verifier with information about the platform state at the time the TickStamp operation was performed.

4. $Quote_{Res}$, $TickStamp_{Res}$, the PCR values, the timer values (TSV_{LB}, TSN_{LB}, $TRATE_{LB}$) and the stored measurement log are returned to the control station.

5. When the response from the camera is received, the control station stores the current UTC time as t_1.

6. The control station verifies the provided data as follows:

   (a) Verify the signature of $TickStamp_{Res}$ using key $K_{SIG_{pub}}$ from the control station database and check the nonce n.

   (b) Verify the signature of $Quote_{Res}$ using $K_{AIK_{pub}}$ from the control station database.

   (c) Check the returned PCR values together with the measurement log to verify that the system is in a state that is compliant with a predefined policy (i.e. the system image and the executed applications are known and categorized as uncritical).

7. If the verification is not successful, the camera should be taken out of service for closer inspection.

8. The time values t_0, t_1, TSV_{LB}, TSN_{LB}, $TRATE_{LB}$ are stored in the database of the control station. This associates the tick value TSV_{LB} of the current tick session TSN_{LB} with the UTC time interval t_0 to t_1. For a more accurate association, average runtimes of individual steps of the lifebeat can be taken into account.

The described, remotely triggered trusted lifebeat procedure not necessarily has to be executed in a strict time interval but the control station can send requests at random time intervals. It however should be ensured that these intervals do not exceed a previously configured maximum time.

One of the major problems of TC on a generic PC platform is the verification of PCR values reported by TPM Quote because there are no limitations on what applications users are allowed to run. This results in a huge number of possible PCR configurations. In the context of the proposed system, this is not an issue because the software base is relatively small and measurements are only performed for a limited number of components like the bootloader, the firmware base image and application blocks.

5.4 Image Signing and Timestamping

An important issue when using cameras e.g. in traffic surveillance or law enforcement applications is to get assurance where (i.e. by which camera) and when an image was taken. Having TPMs on the cameras provides basic functionality required for this task. In our setup the first issue is addressed by signing image data with a non-migratable signing key that is protected by the TPM of the camera. As this private key can not be extracted from the TPM, this signature proves that an image actually originates from the camera the TPM belongs to. Signing of images on the cameras is done as follows:

1. Acquire image data img from the camera sensor.

2. Sign image data: $Sig_{Res} \leftarrow TPM\_Sign_{K_{SIG}}(img)$

3. The image signature Sig_{Res} and the original image img are transmitted to the control station. Optionally, they can be stored in local memory for later use. This is useful in applications where e.g. frequent radio transmissions have to be avoided to save power.

4. To verify the origin of image data, at the control station $K_{SIG_{pub}}$ associated to the camera in question has to be looked up.

5. If signature verification $Verify_{K_{SIG_{pub}}}(Sig_{Res}, img)$ succeeds, one has assurance that the data (1) was not altered and (2) comes from a specific camera as it was signed with a key protected by the camera's TPM.

To address the issue when an image was taken, we make use of the TPM TickStampBlob function. The procedure is similar to that for image signing. At the control station, the tick values have to be associated to the universal time with the help to the tick values obtained from the periodic lifebeat. Image timestamping is done as follows:

1. Acquire image data img from the camera sensor.

2. Call the TPM TickStampBlob function that signs the current TPM tick value and the image:

$$TickStamp_{Res} \leftarrow TPM\_TickStampBlob_{K_{SIG}}(TSN_{img}||TSV_{img}||TRATE_{img}||img)$$

3. $TickStamp_{Res}$, TSN_{img}, TSV_{img}, $TRATE_{img}$ as well as img are transferred to the control station or are stored on the camera for later use.

4. At the control station, $K_{SIG_{pub}}$ associated with the camera is retrieved from the database.

5. Verify the timestamp data:

$$Verify_{K_{SIG_{pub}}}(TickStamp_{Res}, TSN_{img}||TSV_{img}||TRATE_{img}||img)$$

6. From the database, retrieve the most recent lifebeat that took place before the timestamping of the image. This data includes t_0, t_1, TSN_{LB}, TSV_{LB} and $TRATE_{LB}$ as described in section 5.3. Lookup is done with (TSN_{img}, TSV_{img}) as key such that $TSN_{img} = TSN_{LB}$, $TSV_{img} > TSV_{LB}$. If available, check that $TSN_{img} = TSN_{LB+1}$ and $TSV_{img} < TSV_{LB+1}$.

7. Associate the TSV_{img} time value with UTC time:

$$t_0 + \vec{x} < UTC_{img} < t_1 + \vec{x}$$
$$\text{with } \vec{x} = (TSV_{img} - TSV_{LB}) * TRATE\ [\mu s]$$

   Assuming that the time $\vec{t_Q}$ the TPM requires for the Quote and the minimal transmission time $\vec{t_{tx}}$ for the response back to the control station is known, the interval for UTC_{img} can be narrowed further:

$$t_0 + \vec{x} < UTC_{img} < t_1 + \vec{x} - \vec{t_Q} - \vec{t_{tx}}$$

8. If none of the aforementioned steps failed, one now knows that the image data (1) was not modified, (2) comes from a specific camera (signed with a TPM key) and (3) was taken within a certain, narrow timeframe.

The accuracy of the time UTC_{img} associated with the image depends on two major aspects. The first one is the time required for transmission of the lifebeat request over the network to the camera and the time for transmitting

the result. The second aspect is the time required for processing the lifebeat request on the camera. Processing time is dominated by the time the TPM requires for the TPM Quote and TPM TickStampBlob operations. For a given TPM model, those times can be measured in advance and taken into account for calculation of the image timestamp. Network latency, especially in wireless networks, can not easily be predicted and therefore remains as an uncertainty factor. Corresponding evaluations of our prototype system are presented in section 6.

5.5 Encryption of Sensitive Image Regions

Even in case of smart cameras, there are situations where full images get stored for later use or are sent to the control station. To achieve privacy of monitored individuals we not only propose to encrypt selected regions of an image but to specifically do that with a key protected by the control station's TPM_S. For that purpose the binding keys $K_{BIND1...N_{pub}}$ which have been created during camera setup are used. This ensures that privacy relevant information can only be decrypted by individuals that have access to the control station and its TPM_S. In our prototype, we have chosen a rather simple model to identify privacy relevant information: Any motion detected by the camera is classified as privacy relevant. After identifying the motion regions,

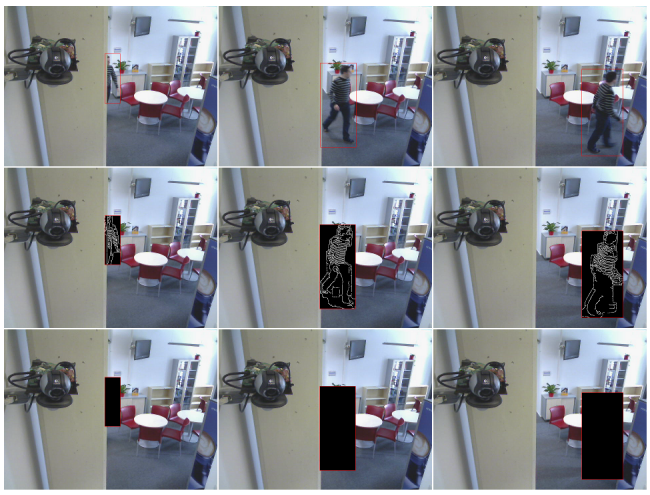

Figure 5: In the upper row, the full image is disclosed while in the middle row the motion regions are replaced by the results of an edge detection. The last row shows a series of images where the moving person is completely removed.

we replace the detected regions with a black box as shown in the bottom row of figure 5. The original regions (bounding boxes in upper row of figure 5) are encrypted with an AES session key that gets bound to TPM_S using K_{BIND1}. To provide an intermediate level between revealing all or no information contained in motion regions, we perform edge detection. The result of this process (boxes in center row of figure 5) also is encrypted with an AES session key which is bound to TPM_S with K_{BIND2}. The images where edge detection was performed, provide operating personnel with more details about the activity within the motion regions without revealing too many details. Depending on their security clearance, operators have knowledge of the usage secrets for K_{BIND1}, K_{BIND2} or none and hence have access to images with different information. To avoid misuse of secrets, the session keys could be encrypted using two different binding keys, e.g. K_{BIND1} and K_{BIND3} such that at least two operators have to cooperate to reveal the encrypted data. Regardless of the policy, using binding keys of TPM_S to protect the privacy sensitive data guarantees that this data can only be accessed at the control station.

6. EVALUATION AND DISCUSSION

As embedded computer vision systems have limited computing power, the performance impact of the proposed security concepts is of high interest. In our system, the two main aspects are the performance of the TPM and the performance of AES and SHA1 computations which are always done by the ARM CPU no matter if a hardware or software TPM is used. In table 2 we compare runtimes for selected TPM commands executed on hardware and software TPM implementations. The measurements have been performed on the parameter block generation layer of the TSS which means that overhead for communication with higher layers is not included. For TrouSerS, this overhead was measured to be 5 to 15 ms depending on the command. Table 2 shows that performance of hardware TPMs strongly depends on the manufacturer. The Infineon implementation has the lowest runtimes followed by the Intel TPM that is part of recent chipsets. In all cases, the software TPM emulator running on the camera prototype outperforms the hardware TPMs. For completeness, we not only present runtime measurements for 2048 bit RSA keys but also for 1024 bit keys showing a clear performance advantage. In our prototype we however chose to work with 2048 bit keys only. AES and SHA1 performance is presented in table 3. For AES and SHA1, data sizes have been chosen to approximately match those of a JPEG compressed region of interest (8 kB) and JPEG compressed color images at resolutions of 320x240 (15 kB) and 640x480 (40 kB). Additionally, times are given for raw RGB images at 320x240 (80 kB).

The motion detection application operates on color images at a resolution of 320x240 pixels. The procedure is based on frame differencing followed by binarization, morphological operations and region growing. With this simple, not optimized implementation 12.5 fps are achieved resulting in a processing time of 80 ms per frame. Based on runtimes for TPM commands and the vision component, we now are going to discuss the performance of the proposed TC application scenarios.

For the Trusted Lifebeat, the relevant TPM operations are TPM_TickStampBlob, TPM_Quote and TPM_OIAP for authorization sessions. Assuming 2048 bit RSA keys, this results in a runtime of about 165 ms using TPM Emulator on our camera platform. Including TSS overheads, runtime increases to 180 ms. With the fastest hardware TPM (Infineon) runtime would go up to 755 ms. As in this case the TPM commands are executed in parallel to the main processor, this would be acceptable. The amount of data transmitted in lifebeat requests and responses is very small. On our prototype, WiFi transmission time $\vec{t}_{tx}$ for 5 kB of data is less than 2.5 ms. Consequently, the time difference $t_1 - t_0$ in the lifebeat is dominated by the TPM operations. Knowing the times for TPM Quote ($\vec{t}_Q$) and TPM TickStampBlob ($\vec{t}_{TS}$) allows to give the minimal achievable time between t_0 and t_1 which for our prototype is $2 * \vec{t}_{tx} + \vec{t}_{TS} + \vec{t}_Q \approxeq 165\,ms$.

	Atmel (AT97SC3203)	Broadcom (BCM5755)	Infineon (SLB9635TT)	Intel (ICH10)	STMicro-electronics (ST19NP18)	TPM Emulator (Laptop)	TPM Emulator (Camera)
TPM_OIAP	44 ms	19.0 ms	28.6 ms	23.4 ms	15.1 ms	0.4 ms	2.9 ms
RSA key size: 2048 bits							
TPM_Quote	827.1 ms	910.6 ms	353.5 ms	fail	948.3 ms	15.6 ms	78.6 ms
TPM_TickStampBlob	799.7 ms	912.4 ms	340.9 ms	491.2 ms	914.8 ms	15.8 ms	79.4 ms
TPM_Sign	792.6 ms	911.2 ms	340.0 ms	474.6 ms	901.4 ms	15.8 ms	77.5 ms
TPM_Seal	126.0 ms	21.9 ms	181.9 ms	145.2 ms	318.1 ms	1.2 ms	8.2 ms
TPM_Unseal	855.1 ms	909.3 ms	344.1 ms	559.2 ms	1069.7 ms	16.0 ms	82.5 ms
TPM_Unbind	827.5 ms	880.7 ms	335.2 ms	516.4 ms	996.9 ms	15.6 ms	80.4 ms
RSA key size: 1024 bits							
TPM_Quote	207.3 ms	391.5 ms	105.1 ms	fail	253.1 ms	2.8 ms	15.8 ms
TPM_TickStampBlob	190.2 ms	390.2 ms	96.9 ms	146.4 ms	218.5 ms	2.7 ms	16.6 ms
TPM_Sign	185.5 ms	390.3 ms	95.5 ms	132.8 ms	206.6 ms	2.6 ms	14.7 ms
TPM_Unbind	208.1 ms	390.4 ms	88.5 ms	135.8 ms	257.6 ms	2.9 ms	17.1 ms

Table 2: Runtime measurements for selected TPM commands on different TPM 1.2 chips and the TPM Emulator (Core2 Duo, 1.6 GHz and PSC). Where applicable, RSA key sizes of 2048 and 1024 bits were evaluated. Values are averaged over 10 runs. TPM_Quote could not successfully be performed on the Intel TPM.

	Data Size	Runtime PSC Prototype	Runtime Laptop
SHA1	8 kB	0.4 ms	0.05 ms
	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 128	8 kB	1.2 ms	0.1 ms
	15 kB	2.2 ms	0.2 ms
	40 kB	6.0 ms	0.5 ms
	80 kB	11.8 ms	1.1 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 3: Average runtimes (10 runs) for SHA1 and AES (key lengths 128 and 256 bits) on the camera prototype. For comparison, measurements on a laptop (Core2 Duo, 1.6 GHz) are given.

The UTC time assigned to a timestamped image is between t_0 and $t_1 - \vec{t}_{tx} - \vec{t}_Q \approxeq t_1 - 82\,ms$.

Image signing requires the computation of the SHA1 hash of the image which takes less than 2 ms for sizes of 40 kB and below. The more relevant parts are the TPM_Sign / TPM_TickStampBlob and TPM_OIAP operations which together require 82 ms (2048 bit RSA key). Combined with SHA1 computation and TSS overhead, image signing (or timestamping) requires about 90 ms. Signing or timestamping every frame would result in more than halving the maximum achievable framerate. With the fastest hardware TPM (Infineon) the framerate would even decrease to 2 fps. Depending on the application, it might however not be required to sign or timestamp every frame but only selected frames where a special event was detected. Alternatively, signing or timestamping could be performed e.g. once per second for the past n frames where the individual SHA1 sums are aggregated similar to the PCR extend operation. Both approaches can help to reduce the performance impact of image signing.

For encryption of regions of interest, an AES 256 session key is generated with the TPM's random number generator. This key is then protected with one of the public, 2048 bit RSA binding keys belonging to TPM_S. Binding the AES session key requires 5 ms. AES encryption of JPEG compressed regions of interest (8 kB) takes less than 2 ms which is an acceptable performance impact. Session key management (e.g. validity periods) depends on a predefined policy.

7. CONCLUSION AND FUTURE WORK

In this work we have explored potential applications of Trusted Computing in the context of an embedded computer vision system. We have limited our approach to selected aspects including a trusted lifebeat to attest the state of cameras, signing and timestamping of images as well as protecting privacy sensitive image data. Compared to existing security solutions, the addition of a TPM has the advantage of secure storage for keys used for signing, timestamping, lifebeat and platform attestation. In addition to specifying the involved components and procedures, we did a prototype implementation on our camera platform. The conducted performance measurements show that the impact on overall performance is acceptable for the lifebeat and image encryption. For image signing, the overhead can be reduced by limiting it to important events or signing groups of images.

Based on the presented results, we identified a number of directions for future research. Hardware TPM solutions do not provide adequate performance (e.g. signing and timestamping) to be used in computer vision applications where timing is critical. A software TPM is much more usable in this context but still does not provide ideal performance. To increase performance, one approach is to replace RSA with ECC as proposed in related work. Alternatively, the DSP co-processor of our prototype could be used to perform cryptographic or computer vision applications in parallel to the main processor thereby increasing overall performance. As a pure software solution, our TPM does not provide security characteristics similar to an actual hardware implementation. Approaches how a software TPM can provide security equivalent to a hardware TPM are under active research. One alternative that provides both – the security of a hardware TPM and the performance of a software TPM – could be a hybrid approach where both solutions are combined in one platform.

In our current work we have only considered trust and security aspects that involve the control station/system operators and the cameras. Persons monitored by the system are currently outside this loop. In a future extension we intend to address this issue by adding mechanisms that allow

users to gain insight what applications are running on the cameras using a mobile handset. This way, users can check if e.g. their privacy is appropriately protected.

8. ACKNOWLEDGEMENTS

We thank Martin Pirker, Ronald Tögl and Daniel Hein from the IAIK Trusted Computing Labs/Graz University of Technology for their assistance with the runtime measurements on Intel, Infineon and STMicroelectronics TPMs.

9. 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] N. Baaziz, N. Lolo, O. Padilla, and F. Petngang. Security and privacy protection for automated video surveillance. In *Proceedings of the IEEE International Symposium on Signal Processing and Information Technology*, pages 17–22, 2007.

[3] M. Bramberger, J. Brunner, B. Rinner, and H. Schwabach. Real-Time Video Analysis on an Embedded Smart Camera for Traffic Surveillance. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 174–181, 2004.

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

[5] K. Dietrich and J. Winter. Implementation Aspects of Mobile and Embedded Trusted Computing. In *Trusted Computing*, Lecture Notes in Computer Science, pages 29–44. Springer, 2009.

[6] A. Dua, N. Bulusu, W. C. Feng, and W. Hu. Towards Trustworthy Participatory Sensing. In *Proceedings of the Usenix Workshop on Hot Topics in Security (HotSec)*, August 2009.

[7] A. Dua, W. Hu, and N. Bulusu. A Trusted Platform Based Framework for Participatory Sensing. In *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2009.

[8] F. Dufaux and T. Ebrahimi. Scrambling for Video Surveillance with Privacy. In *Proceedings Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*, pages 160–166, 2006.

[9] 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.

[10] 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 *Security and Privacy in Ad-hoc and Sensor Networks*, Lecture Notes in Computer Science, pages 172–186. Springer, 2007.

[11] W. Hu, P. Corke, W. C. Shih, and L. Overs. secFleck: A Public Key Technology Platform for Wireless Sensor Networks. In *Conference On Embedded Networked Sensor Systems*, Lecture Notes in Computer Science, pages 296–311. Springer, 2009.

[12] IBM. TrouSerS TCG Software Stack. `http://sourceforge.net/projects/trousers/`. last visited: Aug. 2009.

[13] B. Rinner, T. Winkler, W. Schriebl, M. Quaritsch, and W. Wolf. The Evolution from Single to Pervasive Smart Cameras. In *Proceedings of the Int. Conference on Distributed Smart Cameras (ICDSC)*, 2008.

[14] B. Rinner and W. Wolf, editors. *Proceedings of the IEEE: Special Issue on Distributed Smart Cameras*, volume 96, 2008.

[15] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and Implementation of a TCG-Based Integrity Measurement Architecture. In *Proceedings of the 13th USENIX Security Symposium*, pages 223–238, 2004.

[16] W. Schriebl, T. Winkler, A. Starzacher, and B. Rinner. A Pervasive Smart Camera Network Architecture applied for Multi-Camera Object Classification. In *Proceedings of the ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, 2009. (to appear).

[17] 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, 2005.

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

[19] D. Simon, C. Cifuentes, D. Cleal, J. Daniels, and D. White. Java on the Bare Metal of Wireless Sensor Devices. In *Proceedings of the International Conference on Virtual Execution Environments*, pages 78 – 88, 2006.

[20] M. Strasser and H. Stamer. A Software-Based Trusted Platform Module Emulator. In *Proceedings of the International Conference on Trusted Computing and Trust in Information Technologies (TRUST)*, pages 33–47. Springer, 2008.

[21] 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.

[22] Trusted Computing Group. TCG Software Stack (TSS) Specification Version 1.2 Level 1 Errata A. `http://www.trustedcomputinggroup.org/resources/tcg_software_stack_tss_specification`, March 2007. last visited: Aug. 2009.

[23] Trusted Computing Group. TPM Main Specification 1.2, Lelvel 2, Revision 103. `http://www.trustedcomputinggroup.org/resources/tpm_main_specification`, July 2007. last visited: Aug. 2009.

[24] T. Winkler and B. Rinner. Pervasive Smart Camera Networks exploiting heterogeneous wireless Channels. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 296 – 299, March 2009.

[25] J. Winter. Trusted Computing Building Blocks for Embedded Linux-based ARM TrustZone Platforms. In *Proceedings of the ACM Workshop on Scalable Trusted Computing (STC)*, pages 21–30. ACM, 2008.