Security in Building Automation Systems

Wolfgang Granzer, Fritz Praus, and Wolfgang Kastner, *Member, IEEE*

***Abstract*—Building automation systems are traditionally concerned with the control of heating, ventilation, and air conditioning, as well as lighting and shading, systems. They have their origin in a time where security has been considered as a side issue at best. Nowadays, with the rising desire to integrate security-critical services that were formerly provided by isolated subsystems, security must no longer be neglected. Thus, the development of a comprehensive security concept is of utmost importance. This paper starts with a security threat analysis and identifies the challenges of providing security in the building automation domain. Afterward, the security mechanisms of available standards are thoroughly analyzed. Finally, two approaches that provide both secure communication and secure execution of possibly untrusted control applications are presented.**

***Index Terms*—Building automation, embedded networks, integration, security.**

I. Introduction and Motivation

BUILDING AUTOMATION SYSTEMs (BASs) aim at improving the control and management of mechanical and electrical systems in buildings—more generally, the interaction among all kinds of devices typically found there. The core application area of BASs is environmental control handled by the traditional services—heating, ventilation, and air conditioning (HVAC) and lighting/shading [1]. Other application-specific services are often implemented by separated systems. This is particularly true for *safety-critical* (e.g., fire or social alarm systems) and *security-critical* (e.g., intrusion alarm or access control systems) services, which are typically provided by proprietary stand-alone systems [2]. If at all, only loose coupling is implemented with the core BAS for visualization and alarm management.

To activate all inherent synergies among these diverse systems, an integration of security and safety-critical systems is a major topic [3]. The promised benefits range from lower (life cycle) costs to increased and improved functionality. Consider, for example, the possibility of sharing the data originating from just one sensor in multiple application domains in parallel. This will reduce investment and maintenance costs and also facilitates management and, in particular, configuration of the integrated BAS, as now, a multitude of different management solutions can be substituted for a unified view and a single central configuration access point.

Manuscript received December 31, 2008; revised May 30, 2009; accepted August 24, 2009. Date of publication November 13, 2009; date of current version October 13, 2010. This work was supported by Österreichischer Fonds zur Förderung der Wissenschaftlichen Forschung (FWF; Austrian Science Foundation) under Project P19673.

The authors are with the Automation Systems Group, Institute of Computer Aided Automation, Vienna University of Technology, 1040 Vienna, Austria (e-mail: w@auto.tuwien.ac.at; f@auto.tuwien.ac.at; k@auto.tuwien.ac.at).

Digital Object Identifier 10.1109/TIE.2009.2036033

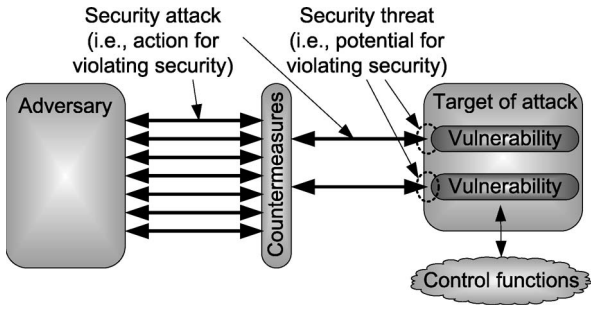

Fig. 1. Security attacks.

Integration also makes way for further improvement of the traditional services. For example, although often not regarded as security critical, incorporating security countermeasures for HVAC and lighting/shading services will prevent, among others, vandalism acts. The economic impact of a company-wide shutdown of the lighting system can be easily compared to a successful attack on the company Web server—the only difference being that, for the Web server, elaborate security measures are already common practice.

Obviously, an integration of security-critical services demands the underlying control system to be *reliable* and *robust* against malicious manipulations. Therefore, based on a sound threat analysis, the goal of this paper is to present a comprehensive security concept that fulfills the demands for security-critical building automation services.

II. Security Threats in Building Automation Systems

To be able to integrate security-critical services, the implemented *control functions*, i.e., functions that control the building automation services, have to be protected against unauthorized access and malicious interference (*security attack*). A typical example of such a security attack is the manipulation of an access control system that opens and closes an entrance door. To perform security attacks, the malicious entity (*adversary*) has to identify *vulnerabilities* of a system that can be utilized to gain unauthorized access to the control functions. The existence of vulnerabilities leads to a *security threat* which can be regarded as the potential for violation of security that may or may not be utilized. Fig. 1 shows the relation between these basic security terms.

On one hand, the protection of a system against security attacks demands that the amount of vulnerabilities is minimized by incorporating security in every stage of the system's life cycle, particularly already during design [4]. On the other hand, countermeasures that eliminate or prevent security threats and attacks in advance have to be implemented. For example, the

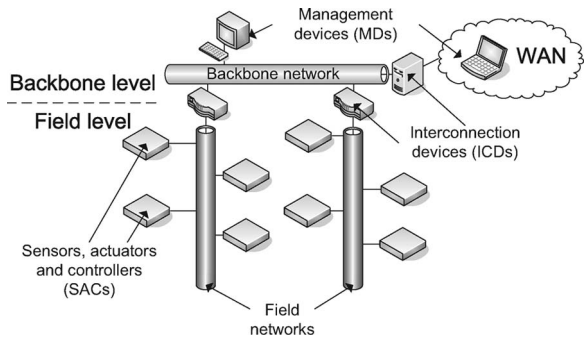

Fig. 2. Building automation network.

encryption of transmitted data can be used to avoid a disclosure of confidential information. In cases where a prevention is not possible with reasonable effort, mechanisms trying to at least detect security attacks, report them, and minimize the resulting damage have to be deployed. Consider, for example, an intrusion detection system (IDS) that reveals abnormal system behavior and tries to prevent a propagation by isolating the source of the attack.

A. Target Analysis

In today's BASs, the control functions are distributed to *control applications* being hosted on different devices interconnected by a common network. Unauthorized access by an adversary to control functions can be gained by, on one hand, directly manipulating the control applications (e.g., changing the control logic or modifying control data such as an output value) or, on the other hand, by indirectly changing control parameters or interfering with the data exchanged among the control applications. Therefore, the control applications themselves, as well as the ways to interact with them, have to be secured.

The distributed control applications of a BAS are spread over a network called *building automation network (BAN)*. While the functionality of a BAS is organized in a three-level hierarchy, BANs are typically implemented following a two-tiered model (cf. Fig. 2). *Field networks* are home for sensors, actuators, and controllers (SACs). They are interconnected by a common *backbone*, where management nodes that require a global view of the entire BAN are located.

With this topology in mind, three different device classes can be identified: *SACs* that interact with the environment and perform control functions, *management devices (MDs)* that execute configuration (e.g., set initial configuration parameters), maintenance (e.g., changing set points), and operator tasks (e.g., visualization and alarm monitoring), as well as *interconnection devices (ICDs)* providing an interconnection between network segments (e.g., routers) or remote access to foreign networks (e.g., gateway to a wide area network).

B. Attack Analysis

Based on this abstract BAN model, five potential attack targets and the following attack scenarios can be identified.

1) Field network: An adversary may try to interfere with the data being exchanged among control applications.
2) Backbone: The data transmitted across network borders are concentrated there. Thus, an adversary may gain a global view of the entire system.
3) SAC: An adversary may directly access SACs to manipulate the behavior of the hosted control applications by changing configuration parameters (e.g., set point), the control logic (e.g., algorithm), or the control data (e.g., output value).
4) ICD: An adversary may attack the application running on the ICD to get access to the data passing through the ICD. As ICDs may also provide an interconnection to foreign public networks (e.g., Internet), an ICD can also be misused as access point to launch further attacks on the BAN.
5) MD: An adversary may attack an MD by manipulating the operator software and also impersonate an MD. The privileges of the compromised device can then be misused to gain management access to SACs or ICDs.

To sum up, an adversary has two different opportunities for getting access to control functions (cf. Fig. 3). On one hand, an adversary may attack the network medium to access the exchanged data and thus interfere with the data when they are transmitted (*network attacks*). According to [5], an adversary may try to intercept, manipulate, fabricate, or interrupt the transmitted data. Access to the network medium can be achieved in two ways.

1) Medium access: The adversary gains physical access to the network medium. This can be accomplished more easily when open communication technologies (i.e., radio frequency or powerline) are used.
2) Device access: The adversary can use the network interface of another device (e.g., a compromised SAC or a Web gateway).

On the other hand, the adversary may attack a device to access control functions (*device attacks*). These attacks can be classified based on the means used to launch them [6], [7].

1) Software attacks: An adversary may use regular communication channels to exploit weaknesses in a device's software.
2) Physical or invasive attacks: An adversary may use physical intrusion or manipulation to interfere with a device.
3) Side-channel attacks: An adversary may observe external (device) parameters which are measurable during operation to collect information about internals.

While mechanisms that counteract network attacks are presented in Section IV, countermeasures against device attacks will be discussed in Section V.

C. Challenges for Providing Security in Building Automation Systems

In the information technology (IT) world, many well-established security mechanisms exist. However, they cannot be trivially mapped to the building automation domain for various reasons. Due to cost efficiency, SACs are normally *embedded devices* with limited system resources (e.g., memory and processing power) that rely on bus or battery power. Security mechanisms (particularly cryptographic algorithms) are

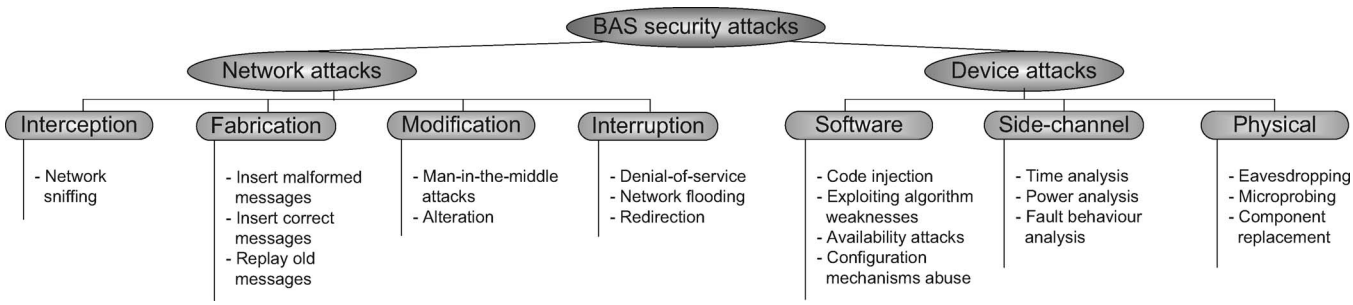

Fig. 3. Attacks in building automation systems.

computationally intensive and must not exceed the available device resources. The overhead imposed by these mechanisms needs to be reasonably small. Therefore, a suitable balance between a required level of security and available resources has to be found ("good enough security").

For many services in the IT world, the amount of devices that communicate with each other is relatively small, thus allowing the client/server model to be used in most cases where only the communication between the clients and the server has to be secured. BANs, on the other hand, usually consist of only a few MDs, some ICDs with defined applications, and thousands of manifold SACs. Communication between SACs occurs peer to peer based without a central instance. Thus, the *scalability* of the integrated security mechanisms is of major concern.

Furthermore, IT security mechanisms are geared toward different requirements regarding the used network technology. While, in the IT world, Internet Protocol (IP)-based network protocols are dominant, the use of IP networks in BANs is reserved to the backbone level. At the field level, predominantly *non-IP field buses* are used. Moreover, control data typically transmitted in BANs have a small volume (on the order of bytes) with perhaps soft real-time requirements (e.g., reaction time in a lighting system). In the IT/office domain, the data volume to be transferred is commonly high (on the order of mega- or gigabytes) with usually no real-time requirements.

Moreover, in BANs, devices often operate in *untrusted environments* where physical access (e.g., an intrusion alarm in a public building or a wireless sensor network [8]) is given. Therefore, it has to be assumed that a short time physical access to devices and networks cannot be avoided.

Finally, a BAS has to be kept operable for years or even decades. Due to the intended *long lifetime*, the possibility to update the software running on the devices has to be provided. This update mechanism also offers an additional attack point that has to be protected against unauthorized use.

III. Security in Building Automation Standards

The most important open BASs' protocol standards spanning more than one application domain are BACnet [9], LonWorks [10], KNX [11], and IEEE 802.15.4/ZigBee [12], [13].

BACnet [9] offers several services to prevent unauthorized interception, modification, and fabrication of the exchanged data. These mechanisms use the symmetric Data Encryption Standard (DES) algorithm and a trusted key server, which is responsible for generating and distributing session keys. The session keys are used to encrypt the transmitted data between two devices. To establish a secure connection to the key server, each node holds an initial secret key. However, BACnet suffers several security flaws and weaknesses [14]: The initial secret key management is not defined, reuse of old session keys is possible since the lifetime is not limited, DES is no longer secure, and, finally, the protocol is vulnerable to man-in-the-middle attacks, type flaws, parallel interleaving attacks, replay attacks, and implementation-dependent flaws. Therefore, BACnet Addendum g [15] replaces these security services to overcome the aforementioned limitations. The new services use the Advanced Encryption Standard (AES), the Keyed–Hash message authentication code (MAC) (HMAC) algorithm, and a unique message identifier in combination with a time stamp to protect the exchanged data against interception, modification, and fabrication. Furthermore, advanced security concepts like the use of different key types and key revisions have been introduced.

LonWorks [10] provides a four-step challenge-response mechanism to counteract modification and fabrication attacks. A sender which intends to authenticate a transmission asserts the authentication bit of its message. Receivers reply with a 64-b random number. The sender returns a 64-b hash value calculated over the content of the message and the random number using a shared key. The receiver performs the same calculation and compares the results. However, [16] describes the following security flaws: Interception of confidential data cannot be avoided, the identity of the receiver is not verified, authentication is limited to acknowledged unicast and multicast, the authentication protocol is inefficient in large groups due to the limited key length, the cryptographic algorithm is not secure, key management services are not defined, and each node can use only a single key. In addition to the basic LonWorks authentication mechanism, LonWorks/IP [17] defines its own security mechanism which uses MD5 together with a shared secret to protect the data against modification and fabrication. Since the transmitted data are not encrypted, an interception cannot be avoided. However, due to the fact that MD5 is not collision resistant, the used mechanism is insecure, too. Furthermore, secure mechanisms to manage and distribute the used shared secrets are missing.

KNX [11] does not offer mechanisms to avoid network attacks. It only provides a basic access control scheme which can be used to limit the management access to devices. Up to 255 different access levels can be defined, each of them associated with a different set of privileges. For each of these

access levels, a 4-B password can be specified. However, it is of limited use due to the following weaknesses [18]: The keys are transmitted in clear text, there are no mechanisms to manage keys in a secure manner, KNX's single management tool Engineering Tool Software (ETS) uses only one key for the whole installation, and an injection of messages after successful authentication cannot be avoided. In addition to this access control mechanism, KNXnet/IP specifies some rudimentary security guidelines. Since they are based on isolation (e.g., firewall; KNXnet/IP only Intranet) and "security by obscurity" (e.g., the use of nonstandard IP addresses; rely on the missing expertise of an adversary), these security guidelines are not able to provide reasonable improvements.

IEEE 802.15.4:2006 [12] offers security services at the link layer that use CCM* as combined encryption and authentication block cipher mode. It can be used in environments requiring authentication only (against fabrication and modification), encryption only (against interception), or a combination of both. *ZigBee 2007* [13] utilizes the IEEE 802.15.4:2003 transmission services of the data link layer. However, ZigBee 2007 does not use the security mechanisms provided by IEEE 802.15.4:2003—they are completely replaced by a more advanced security concept. This concept supports the use of different key types and provides advanced key management services. Again, CCM* is used as a cryptographic algorithm.

While the security mechanisms of LonWorks and KNX are not able to fully avoid network attacks, BACnet Addendum g and IEEE 802.15.4/ZigBee provide a solid base. However, there is much room for improvement. Due to performance reasons, the distribution of security primitives is based on symmetric algorithms exclusively, and thus, an entity called key server is required. However, nowadays, asymmetric algorithms based on elliptic curve cryptography (ECC) [19] are suitable for embedded devices and eliminate the need for a key server [20], [21]. Moreover, the use of a single key server introduces a single point of failure. Therefore, a scheme based on multiple key servers is desirable [22]. To support all kinds of applications, the use of different communication models shall be possible. BACnet only provides support for the client/server model—exchanging process data within groups as it is possible in LonWorks and KNX is not supported. ZigBee supports multicast but services to control the group membership, which may be required for security-critical applications, are missing. Finally, mechanisms to protect against interruption attacks [e.g., Denial-of-Service (DoS) attacks] are not supported by any of these solutions.

Mechanisms to counteract device attacks are not covered by any of these standards. Since BACnet and IEEE 802.15.4/ZigBee only specify the communication protocol and the application model to be used, details about the device implementation and methods to manage control applications (e.g., management tools to configure and upload control applications) are not covered. Therefore, appropriate security mechanisms that handle device attacks are missing, too. A variety of engineering tools is available for LonWorks, most of them based on the LonWorks Network Operating System (LNS) management middleware. However, no security countermeasures are incorporated into LNS. For KNX, a single management tool called ETS is available, which provides mechanisms to configure and upload the control applications. However, the only way to avoid unauthorized use of these management services is to use the insecure access control mechanism mentioned before. A test for malicious behavior of the uploaded control application is neither provided in LonWorks nor in KNX.

IV. Secure Communication in Building Automation Networks

Entities[1] in the building automation domain, which exchange data, are members of a so-called *communication relationship*. The communication rules within such a relationship are specified by a certain communication model. Three different models are common [22]: the *client/server model* (point to point), the *producer/consumer model* (multipoint to multipoint; loose group membership), and the *publisher/subscriber model* (multipoint to multipoint; strict group membership). In the producer/consumer model, data are made publicly available, and the entities decide on their own whether they are interested in the data or not. In the publisher/subscriber model, a group coordinator manages the group membership.

A. Requirements for Secure Communication

Entities intending to become a member of a secure communication relationship must prove their identity first (*entity authentication*). Entity authentication guarantees that the involved entities are what they claim to be, as well as that they are active at the time of authentication.

In addition, the data exchanged between authenticated members of a relationship have to be protected against security attacks. This is guaranteed by a *secured channel*. A secured channel uses physical and/or cryptographic techniques to avoid unauthorized interference of the transmitted data and provides data integrity (against modification), data freshness (against fabrication), and/or data confidentiality (against interception).

While the client/server model normally requires a point-to-point communication service, the producer/consumer, as well as the publisher/subscriber, model needs an efficient service to send data to multiple receivers. Therefore, the security countermeasures should support *unicast*, as well as *broadcast*, and, if possible, *multicast* communication services.

However, security attacks that threaten the availability of data (e.g., DoS attacks) cannot be handled using a secured channel. By wasting network or system resources (e.g., flooding the network with unsolicited messages), the adversary tries to prevent the target from performing its expected function. Therefore, in addition to the concept based on secured channels, dedicated *interrupt countermeasures* that guarantee data availability have to be integrated.

B. Available Solutions From the IT Domain

An obvious approach would be to use well-established IT security mechanisms to counteract network attacks. Within the

[1]The term entity is used as a synonym for a control application that wants to exchange data, as well as for a device that hosts control applications.

diversity of available IT security mechanisms, three of the most important ones are described here. The *IP Security (IPsec)* has been specified as an extension to the IP protocol. IPsec provides mutual entity authentication and data confidentiality, integrity, and freshness. For key exchange, the Internet Key Exchange protocol is used. Due to the design of IPsec, it is of limited use for BASs' field networks. Since it is an extension of IP, IPsec is not suitable in non-IP networks in a native way. In addition, due to its complexity (e.g., protocol overhead; used cryptographic algorithms), the available resources of embedded devices may be insufficient. Furthermore, due to problems with IPsec for multicast communication, changes in the implementation of IPsec would be necessary.

Secure sockets layer (SSL) and its successor *transport layer security (TLS)* are used to protect communication between two entities. During the initial handshake, the entities are authenticated, and a secured channel that guarantees data confidentiality, integrity, and freshness is established. The main disadvantage of SSL/TLS is its support for unicast connections only. Securing multicast or broadcast is not possible.

A *virtual private network (VPN)* is a logical network that is built upon a physical host network. A VPN is transparent to the connected devices. Secure VPNs use a tunneling protocol that provides a secured channel. A popular secure VPN implementation is OpenVPN which makes use of TLS. Most implementations are tailored for PC-based devices and, therefore, too complex to run on embedded devices. Furthermore, they use a centralized server where each client has to connect to. This is a disadvantage in larger networks due to the lack of scalability, particularly if multicast and broadcast communication services are necessary.

While these solutions provide a solid base for protecting network traffic, the management of the communication relationships and their associated security tokens (i.e., input parameters for the used cryptographic algorithms like shared keys, asymmetric keys, and certificates) is not sufficient for BASs. This is particularly true for field networks containing embedded devices and communication relationships that are based on multiple senders and receivers.

C. Approach to Secure Communication

Our approach taken is divided into four phases: First, each entity has to be configured once at installation time (*initial configuration*). Using this initial knowledge, each entity is able to prove its identity and may dynamically join one or more secure communication relationships at run time (*secure binding*). Within such a secure relationship, the entities are able to exchange data through a secured channel (*secure communication*).To close communication, the entity has to leave the relationship (*secure unbinding*).

1) Initial Configuration: To be able to securely bind to communication relationships, the entity has to be prepared by uploading initial configuration data that consist of general information about the BAN (e.g., address information), as well as the so-called *initial security token set (ISTS)*. As outlined later, an ISTS is required to join a secure relationship during the secure binding phase.

Due to the nature of *initial* configuration data, there are no security tokens available that can be used to cryptographically secure the distribution. To avoid that an adversary manipulates the initial configuration data, physical security must be guaranteed at the time of upload [18].

2) Secure Binding: After an entity has retrieved the initial configuration data, it is able to join one or more secure communication relationships. In contrast to the initial configuration phase which is performed only once, an entity is able to bind to multiple communication relationships anytime at run time.

To securely bind to a communication relationship, the joining entity (JE) sends a request to a trustworthy entity (TE). The TE proves the identity of the JE and vice versa. This process is called entity authentication. If the JE possesses the required rights, the TE distributes the so-called *dynamic security token set (DSTS)* to the JE. A DSTS is uniquely assigned to a single communication relationship and is later used in the secure communication phase to secure the exchanged data. To prevent security attacks, the authentication of the entities, as well as the retrieval of the DSTS, has to be secured. This is accomplished using cryptographic algorithms that take the previously received ISTS as input parameter.

To be able to act as a TE for a communication relationship, a TE must be in possession of the DSTS it is responsible for and the ISTS in order to securely communicate with the JEs. In addition, each JE must know where it can find the TEs for the communication relationships that the JE wants to join.

To achieve this, two different schemes that vary in the election of the TE exist. First, predefined coordinators can be used as TEs. To be able to contact them, the address information of available TEs is contained in the initial configuration data of the JE. To secure the communication between the JEs and TEs, the ISTS is used, which contains (among other security tokens) shared secret keys (for symmetric algorithms) or public/private keys and certificates (for asymmetric algorithms).[2] To avoid a single point of failure, a set of predefined redundant coordinators can be used.

Second, TEs can be dynamically elected following a democratic approach. This approach is only reasonable if asymmetric algorithms are used since the ISTS must only contain the certificate of the central authority and the entity's certificate, as well as its public/private key pair. If symmetric algorithms are used, each possible TE would have to hold the shared keys of all possible JEs. Obviously, asymmetric algorithms, like ECC, that are suitable for embedded devices [24] have to be used. Furthermore, a service to discover the TEs must be available since their election is not fixed.

3) Secure Communication: After the DSTS has been retrieved, the secured channel can be established. To save memory and processing power on embedded devices, it shall be possible to individually choose the desired level of security.

1) *Protected channel:* A protected channel guarantees data integrity and allows one to verify the data origin.
2) *Trusted channel:* In a trusted channel, data freshness and data integrity are guaranteed.

[2]Basic cryptography concepts and common algorithms are explained in detail in [23].

3) *Confidential channel:* In a confidential channel, the transmitted data are additionally encrypted.

To realize the objectives of a secured channel, symmetric cryptographic algorithms are used. A typical example would be to use an encryption algorithm like AES to guarantee data confidentiality, a MAC algorithm like HMAC or cipher-based MAC (CMAC) to guarantee data integrity, and a monotonically increasing counter that is embedded in the message to guarantee data freshness. Alternatively, authenticated encryption algorithms that use a single algorithm to guarantee all three objectives (e.g., Offset Codebook (OCB) mode used in MiniSec [25]) can be used. Regardless of the algorithms employed, input parameters like shared secret keys or other security tokens are required. These parameters are contained in the DSTS.

To guarantee that only authorized entities can process secured messages and to reduce the amount of DSTS uses, each DSTS is dedicated to a single secure communication relationship. This guarantees the logical separation of the different communication relationships. To further reduce the amount of DSTS uses, the lifetime of a DSTS shall be limited. Therefore, a DSTS *refreshing* mechanism is necessary, which periodically invalidates a DSTS (*revocation*) and distributes a new one to all members of the communication relationship.

4) Secure Unbinding: There are two reasons for secure unbinding. First, an entity may decide on its own to leave a communication relationship because it is no longer interested in the exchanged data. An example is an MD that temporarily performs some management tasks. Second, members of a relationship may decide to actively exclude an entity due to security reasons. This may, for example, be a compromised device where an abnormal behavior is detected by an IDS.

In both cases, it must be guaranteed that the device is no longer able to participate in the communication. Therefore, a *DSTS revocation* has to be performed, and a new DSTS is generated and distributed to the remaining members. The responsibility of this revocation process lies at the TEs that are assigned to the dedicated relationship.

D. Summary

The aforementioned generic approach provides entity authentication, as well as data integrity, freshness, and confidentiality, to counteract the security threat modification, fabrication, and interception. Compared to available solutions from the IT world, it is tailored to the use in BASs. The approach has already been put into practice. Reference [22] features an implementation based on predefined multiple key servers, while a democratic election based on ECC is presented in [26]. To guarantee data availability and thus avoid or detect interruption threats, additional measures are necessary. A possible solution that is based on intrusion detection and isolation of the attack sources is presented in [27].

V. Secure Devices in Building Automation Systems

While an overall device security deals with software, side-channel, and physical attacks, this section focuses on software attacks. Still, an extensive survey on side-channel and physical attacks can be found in [28].

A. Requirements for Securing Devices

Any (malicious) software, irrelevant whether it originates from trusted or nontrusted sources, being run on BASs' devices may exploit weaknesses in security schemes and system implementation intentionally or unintentionally. Accidental programming flaws shall not be present just like software being infected by Trojans. Therefore, security mechanisms need to be included, which ascertain the operational correctness of protected code and data before and at run time.

While the software architecture of MDs and ICDs is rather fixed and tailored to the device's functionality, a SAC typically comprises of a generic "template" network node with application-specific hardware. Universally designed base platforms consisting of microcontrollers and network interfaces are used in conjunction with application-specific components (e.g., switches; temperature sensors) to form a particular system. Similarly, the software is split into a generic operating system (OS) or system software providing basic functionality and (customizable) control applications dealing with the specific hardware. On one hand, such a two-level concept allows rapid innovation and implementation. On the other hand, it may impose security risks, which a security concept has to deal with: While malicious, erroneous, or compromised control applications may be uploaded long after device deployment, they shall not interfere with the concerning device software and thus violate the device security.

B. Available Software Protection Techniques

Commonly used state-of-the-art techniques can be categorized into static software, dynamic software, hardware-supported, and human-assisted methods.

Within the context of security, *static code analysis (SCA)* is used to detect programming flaws that result in vulnerabilities [29]. Using *code signing (CS)*, executables are signed by the developers to confirm both origin and nonmodification. The user can thus decide about the trustworthiness of a program with respect to its origin. *Software watermarking* can be used to embed additional nonremovable information into a piece of data, usually to assure that the rights of the creator are not violated (digital rights management). *Proof-carrying code (PCC)*[30] is a technique where a code developer provides a proof along with a program that allows one to check with certainty that the code can be securely executed (e.g., does not to contain buffer overflows).

IDS observes the behavior of a system and uses the collected information to detect malicious behavior or actions [31]. *Signature-based IDS (SIDS)* detects malicious actions by comparing observed information to a collection of signatures describing known attacks. *Anomaly-based IDS (AIDS)* uses representations of the trained normal behavior of a system to detect abnormal activities. *Software-monitoring techniques (SMTs)* observe the execution of specific programs [32]. By identifying and reacting to certain security-relevant events, they can check if programs behave according to a given security

TABLE I
COMPARISON OF DESCRIBED SOFTWARE METHODS TO IMPROVE APPLICATION SECURITY

	method	compile-/runtime	attack types	updates required	scalability	SAC	ICD	MD
static software methods	static code analysis	ct	known	yes	-	∼	+	+
	code-signing	ct	n/a	no	∼	∼	-	+
	proof-carrying code	ct	all	no	-	-	∼	-
dynamic software methods	signature based IDS	rt	known	yes	-	-	-	-
	anomaly based IDS	rt	all	no	+	+	+	∼
	software monitoring	rt	all	no	+	∼	∼	∼
	sandboxing	rt	all	no	+	∼/+	∼/+	+
	self-checking code	both	all	no	+	∼	∼	∼
	operating system	rt	all	yes	+	-	-	∼
	attack specific	both	specific	no	-	∼	∼	∼
hardware supported	coprocessor	rt	all	no	-	-	∼	+
	physical partitioning	rt	all	no	-	-	∼	+
	system architecture	rt	specific	no	∼	+	+	∼
	CPU extension	rt	specific	no	-	∼	∼	+
human	inspection and certification	ct	all	n/a	-	-	∼	∼

policy. A *sandbox (SB)* is a technique where programs are executed in a controlled way, often with restricted permissions. The essential benefit is that the executing host is protected from direct attacks of the software running in the SB. *Self-checking code (SCC)* checks itself for modifications at run time, assuming that modifications are undesirable and probably malicious [33]. In addition, a number of *attack specific countermeasures (ASCs)* which target only specific types of vulnerabilities (e.g., buffer overflows [34]) exist.

Hardware-supported methods may form an extra security barrier. A common approach is to use a *coprocessor* which performs security checks at run time [35]. The increased spreading of multicore architectures allows their use for security mechanisms. *Physical partitioning* [36] may be used to improve a system's reliability, as well as its resistance against security attacks. Processors implementing the *Harvard architecture* provide resistance against code injection attacks by design through the separation of instruction and data memory. A recent security technique is the *no-execute bit* (NX) in modern CPUs, allowing memory regions to be designated as being nonexecutable.

Manual inspection and certification performed by humans may eliminate a lot of possible attacks, but require extensive knowledge by the auditing person, and are time consuming and error prone.

C. Approach for Software Security in Building Automation Systems

Table I discusses the characteristics of the presented techniques relevant for the building automation domain and evaluates the applicability of the methods for SACs, ICDs, and MDs. Column "compile time (ct)/run time (rt)" describes whether the method is applied at compile time or before execution, at run time, or both. Column "attack types" discusses the effectiveness against known, specific, or all attack types. Column "updates required" states whether the method requires continuous updates, and column "scalability" evaluates the method with respect to its costs and maintainability in BASs.

SCA, CS, and PCC are assumed to be easily applicable with respect to resources of the target system because they are only used at compile time. A problem of SCA is that, for typical programming languages, some fundamental questions are undecidable or uncomputable [37]. CS can also be effective to prevent the installation of malicious programs by simply refusing to execute not properly signed ones and thus trusting the signee. However, CS does not prevent accidental flaws. The universal applicability of PCC is questionable since the generation and encoding of proofs for complex security policies are nontrivial tasks. A combination of the static PCC approach with dynamic execution monitoring, however, seems to be practically feasible [38]. Moreover, SCA, CS, and PCC have to be performed on every code change. SIDS is not well suited as it depends on a usually large database and requires constant updates which would be difficult in the case of SACs and ICDs. AIDS, SMT, and SCC may be efficiently implemented and could therefore be quite appropriate. SMT at least requires hardware support for context switches. ASCs can also work well but are not generally applicable due to differing processor and memory architectures. The applicability of SBs strongly depends on the overhead imposed by its feature set. While a reduced and lightweight SB could easily be deployed to SACs, an architecture like the full Java VM with its vast execution and security mechanisms imposes high overhead. The targeted hardware of SACs and ICDs does not provide the necessary hardware support for OSs. Moreover, such OSs cannot provide comparable protection or are not even designed to provide security measures. Hardware-supported methods requiring additional components cannot be cost effectively deployed to SACs. Manual inspection and certification do not scale at all and may only be applied to code which does not change frequently.

As can be seen, all methods have one aspect in common: They are not able to offer full protection against the threats discussed in Section II. Therefore, the approach is to combine the techniques to make them—along with some human preparation or interaction—useful in practice.

The software of an MD typically consists of an OS and the management software. The security of the OS has to be provided by the system administrator. The security of the management software itself can be established using SCA, sandboxing, and up-to-date development tools. Since the software of ICDs is rather fixed, SCA and hardware-supported methods may be used.

To ease the development of secure control applications hosted on SACs, a software environment being able to deal

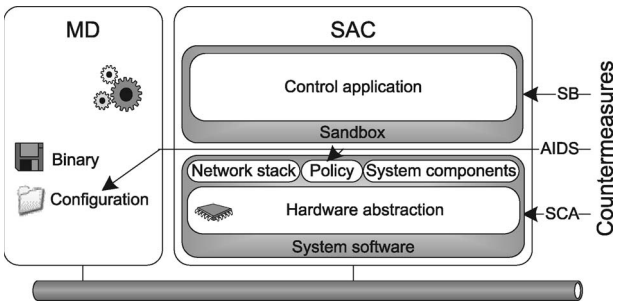

Fig. 4. Secure control applications.

with the discussed attacks is required. Protection basically against software attacks shall be provided but with other attack scenarios such as physical or side-channel attacks in mind [39].

As outlined in Fig. 4, the software of a SAC consists of three major components, each imposing an additional security barrier to the overall security and limiting possible attack points.

1) A simple, tight, and secure system software provides controlled access to system resources. Its security is analyzed using human-being-based inspection and SCA using automated tools, as well as formal verification [40].
2) An SB restricts the execution of customizable control applications. Basically, this uploadable code shall be allowed to perform any desired action. However, to prevent security flaws, it is under continuous control of the system software during run time, determining whether it is allowed to execute a desired function or not. The SB is also designed to support the rapid development of control applications. It provides a clear abstraction of the underlying hardware and offers interfaces to the system software. The developer is thus relieved of any hardware or device-specific details and can focus on the application development itself. This allows portability of applications between devices offering the same SB.
3) A configuration has to be provided to the system software during the upload of a control application that defines its basic policy (i.e., normal behavior). Any abnormal behavior can then be detected by the system software using an AIDS. Thus, limits to, for example, network or processing resources may be defined, which are enforced at run time.

To further limit the possible attack scenarios, the use of Harvard-architecture-based hardware is recommended.

D. Summary

When deployed to SACs, the proposed architecture allows the development, upload, and execution of arbitrary, noninspected, and uncertified (and possibly erroneous or malicious) software without compromising the overall device security. Note that not only attacks evolving from accidental software faults can be prevented but also attacks resulting from intentional malicious software. The solution is low cost since it does not require any additional hardware modifications, thus allowing easy and compatible integration into existing BASs. References [39] and [41] show the novelty of this approach when put into practice.

VI. Conclusion

The history of IT security can serve both as good and bad examples for the future of BASs' security. Important Internet-related protocols were developed for a virtually closed user community. Consequently, security was neglected, and services and applications that used these protocols remained unprotected against security attacks. Due to the ubiquitous use of the Internet, unprotected services and applications became attractive for adversaries. The resulting economic damage through viruses, Trojan horses, and worms is still clearly tangible today. To counteract these threats, protocol extensions and security mechanisms have been developed that are widely used today.

As has been demonstrated in this paper, a BAS is equally prone to security attacks (as the IT world years before) because existing technologies lack state-of-the-art security features. Thus, it is mandatory to identify and set up security mechanisms as it has been done in the IT domain. On the road to reach this ambitious goal, it is necessary to secure communication and to provide a secure environment for BASs' devices—otherwise, adversaries will soon single out unprotected building systems as their next target on a large scale.

References

[1] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newman, "Communication systems for building automation and control," *Proc. IEEE*, vol. 93, no. 6, pp. 1178–1203, Jun. 2005.
[2] M. Thuillard, P. Ryser, and G. Pfister, "Life safety and security systems," in *Sensors in Intelligent Buildings*, vol. 2. Weinheim, Germany: Wiley-VCH, 2001, pp. 307–397.
[3] T. Novak, A. Treytl, and P. Palensky, "Common approach to functional safety and system security in building automation and control systems," in *Proc. IEEE Int. Conf. Emerging Technol. Factory Autom.*, 2007, pp. 1141–1148.
[4] Information Technology–Security Technique–Evaluation Criteria for IT Security, IEC 15408, 2005.
[5] C. P. Pfleeger and S. L. Pfleeger, *Security in Computing,* 4th ed. Englewood Cliffs, NJ: Prentice-Hall, 2006.
[6] D. Hwang, P. Schaumont, K. Tiri, and I. Verbauwhede, "Securing embedded systems," *IEEE Security Privacy*, vol. 4, no. 2, pp. 40–49, Mar. 2006.
[7] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady, "Security in embedded systems: Design challenges," *ACM Trans. Embed. Comput. Syst.*, vol. 3, no. 3, pp. 461–491, Aug. 2004.
[8] V. Gungor and G. Hancke, "Industrial wireless sensor networks: Challenges, design principles, and technical approaches," *IEEE Trans. Ind. Electron.*, vol. 56, no. 10, pp. 4258–4265, Oct. 2009.
[9] BACnet—A Data Communication Protocol for Building Automation and Control Networks, ANSI/ASHRAE 135, 2008.
[10] Control Network Protocol Specification, ANSI/EIA/CEA 709.1, 1999.
[11] *KNX Specification,* Konnex Assoc., Diegem, Belgium, 2009, Ver. 2.0.
[12] Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs), IEEE 802.15.4, 2006.
[13] *ZigBee Specification 2007,* ZigBee Alliance, San Ramon, CA, 2007.
[14] D. G. Holmberg, "BACnet wide area network security threat assessment," Nat. Inst. Standards Technol., Gaithersburg, MD, NISTIR Tech. Rep. 7009, 2003.
[15] ANSI/ASHRAE Addendum 135–2008 g: Updating BACnet Network Security, May 2010, Status: 5th public review. [Online]. Available: http://bacnet.org/Addenda/index.html
[16] C. Schwaiger and A. Treytl, "Smart card based security for fieldbus systems," in *Proc. IEEE Int. Conf. Emerging Technol. Factory Autom.*, 2003, pp. 398–406.
[17] Tunneling Component Network Protocols Over Internet Protocol Channels, ANSI/EIA 852, 2002.
[18] W. Granzer, W. Kastner, G. Neugschwandtner, and F. Praus, "Security in networked building automation systems," in *Proc. IEEE Int. Workshop Factory Commun. Syst.*, 2006, pp. 283–292.
[19] A. Cilardo, L. Coppolino, N. Mazzocca, and L. Romano, "Elliptic curve cryptography engineering," *Proc. IEEE*, vol. 943, no. 2, pp. 395–406, Feb. 2006.

[20] V. Gupta, M. Wurm, Y. Zhu, M. Millard, S. Fung, N. Gura, H. Eberle, and S. C. Shantz, "Sizzle: A standards-based end-to-end security architecture for the embedded Internet," *Pervasive Mobile Comput.*, vol. 1, no. 4, pp. 425–445, Dec. 2005.
[21] W.-S. Juang, S.-T. Chen, and H.-T. Liaw, "Robust and efficient password-authenticated key agreement using smart cards," *IEEE Trans. Ind. Electron.*, vol. 55, no. 6, pp. 2551–2556, Jun. 2008.
[22] W. Granzer, C. Reinisch, and W. Kastner, "Key set management in networked building automation systems using multiple key servers," in *Proc. IEEE Int. Workshop Factory Commun. Syst.*, 2008, pp. 205–214.
[23] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. Boca Raton, FL: CRC, 1997.
[24] T. Wollinger, J. Pelzl, V. Wittelsberger, C. Paar, K. Saldamli, and C. K. Koc, "Elliptic and hyperelliptic curves on embedded μP," *ACM Trans. Embed. Comput. Syst.*, vol. 3, no. 3, pp. 509–533, Aug. 2004.
[25] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "MiniSec: A secure sensor network communication architecture," in *Proc. Int. Conf. Inf. Process. Sens. Netw.*, 2007, pp. 479–488.
[26] W. Granzer, D. Lechner, F. Praus, and W. Kastner, "Securing IP backbones in building automation networks," in *Proc. IEEE Int. Conf. Ind. Informat.*, 2009, pp. 410–415.
[27] W. Granzer, C. Reinisch, and W. Kastner, "Denial-of-service in automation systems," in *Proc. IEEE Int. Conf. Emerging Technol. Factory Autom.*, 2008, pp. 468–471.
[28] F. Koeune and F.-X. Standaert, "A tutorial on physical security and side-channel attacks," in *Foundations of Security Analysis and Design III*. Berlin, Germany: Springer-Verlag, Sep. 2005, pp. 78–108.
[29] B. Chess and G. McGraw, "Static analysis for security," *IEEE Security Privacy*, vol. 2, no. 6, pp. 76–79, Nov. 2004.
[30] G. C. Necula and P. Lee, "Safe, untrusted agents using proof-carrying code," in *Mobile Agents and Security*, vol. 1419, *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, Jan. 1998, pp. 61–91.
[31] Z. Li, A. Das, and J. Zhou, "Theoretical basis for intrusion detection," in *Proc. IEEE Inf. Assurance Workshop*, 2005, pp. 184–192.
[32] V. Kiriansky, D. Bruening, and S. Amarasinghe, "Secure execution via program shepherding," in *Proc. USENIX Security Symp.*, 2002, pp. 191–206.
[33] Y. Chen, R. Venkatesan, M. Cary, R. Pang, S. Sinha, and M. H. Jakubowski, "Oblivious hashing: A stealthy software integrity verification primitive," in *Proc. Int. Workshop Inf. Hiding*, 2003, pp. 400–414.
[34] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton, "StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks," in *Proc. USENIX Security Conf.*, 1998, pp. 63–78.
[35] D. Arora, S. Ravi, A. Raghunathan, and N. K. Jha, "Hardware-assisted run-time monitoring for secure program execution on embedded processors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 12, pp. 1295–1308, Dec. 2006.
[36] I. Hiroaki, M. Edahiro, and J. Sakai, "Towards scalable and secure execution platform for embedded systems," in *Proc. Des. Autom. Conf.*, 2007, pp. 350–354.
[37] W. Landi, "Undecidability of static analysis," *ACM Lett. Program. Lang. Syst.*, vol. 1, no. 4, pp. 323–337, Dec. 1992.
[38] R. Sekar, V. Venkatakrishnan, S. Basu, S. Bhatkar, and D. C. DuVarney, "Model-carrying code: A practical approach for safe execution of untrusted applications," in *Proc. ACM Symp. Oper. Syst. Principles*, 2003, pp. 15–28.
[39] F. Praus, T. F. Thomas, and W. Kastner, "Secure and customizable software applications in embedded networks," in *Proc. IEEE Int. Conf. Emerging Technol. Factory Autom.*, 2008, pp. 1473–1480.
[40] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "seL4: Formal verification of an OS kernel," in *Proc. ACM Symp. Oper. Syst. Principles*, 2009, pp. 207–220.
[41] F. Praus, W. Granzer, and W. Kastner, "Enhanced control application development in building automation," in *Proc. IEEE Int. Conf. Ind. Informat.*, 2009, pp. 390–395.

Wolfgang Granzer received the Dipl.Ing. and Dr.Techn. degrees in computer science from Vienna University of Technology, Vienna, Austria, in 2005 and 2010, respectively.

He was a Junior Scientist with the Research Unit of Integrated Sensor Systems, Austrian Academy of Sciences, Wiener Neustadt, Austria, where he was involved in the research activities on high-precision clock synchronization. Since 2006, he has been with the Automation Systems Group, Institute of Computer Aided Automation, Viennna University of Technology, where he initiated the research project "Security in Building Automation." His research interests include the field of automation systems with special focus on security, embedded networks, and home and building automation.

Fritz Praus received the Dipl.Ing. degree in computer science and the Mag.rer.soc.oec. degree in computer science management from Vienna University of Technology, Vienna, Austria, in 2005 and 2008, respectively, where he is currently working toward the Ph.D. degree. His Ph.D. dissertation is about secure control applications in building automation.

From 2005 to 2006, he was with the Research Unit of Integrated Sensor Systems, Austrian Academy of Sciences, Wiener Neustadt, Austria, where he was working on high-precision clock synchronization. Since 2006, he has been with the Automation Systems Group, Institute of Computer Aided Automation, Vienna University of Technology, where his research interest is in the area of automation systems. His focus lies on security, hardware–software codesign, and home and building automation.

Wolfgang Kastner (M'06) received the Dipl.Ing. and Dr.Techn. degrees in computer science from Vienna University of Technology, Vienna, Austria, in 1992 and 1996, respectively.

Since 1992, he has been with the Automation Systems Group, Institute of Computer Aided Automation, Vienna University of Technology, Austria, where he has been holding the position of an Associate Professor for computer engineering since 2001. From 1992 to 1996, he was working on reliable transmission protocols for distributed real-time systems. Since 1997, his research interests have been in the areas of control networks and automation systems with a special focus on home and building automation. His current research targets the field- and management-level integrations of building automation networks, concentrating on the open standards BACnet, LonWorks, KNX, and IEEE 802.15.4/ZigBee.

Dr. Kastner is a Founder Member of the IEEE Industrial Electronics Society Technical Committee for Building Automation, Control, and Management.