SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2014; **7:**1887–1899
Published online 18 November 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.903

RESEARCH ARTICLE

Protecting mobile agents communications in pervasive networks with a trusted distributed mediator for ID-based RSA

L. Maccari[1]*, R. Fantacci[2], T. Pecorella[2], G. Ghettini[3] and F. Chiti[2]

[1] Department of Information Engineering and Computer Science, University of Trento, Trento, Italy
[2] Department of Information Engineering, University of Florence, Florence, Italy
[3] ART - Advanced Researches and Technologies, Perugia, Italy

ABSTRACT

This paper proposes a new method of data authentication and encryption for distributed networks supporting mobile software agents. Software agents are a valuable instrument in wireless distributed monitoring networks, because they can be used to concentrate monitoring efforts in certain areas where an event is taking place. In this way, events can be tracked in a dynamic and efficient way. Mobile agents have to send messages to each other in order to coordinate their actions, and those messages need to be secured by crypto credentials. However, when agents are moved over wireless networks, how can credentials be protected from sniffing by an attacker, besides layer II encryption? Moreover, if a rogue agent is injected in the network, is it possible to limit the damages it can produce? The proposed approach bridges mediated RSA with the trusted platform modules, in order to provide an efficient and secure communication between agents. The communication is secured using indeed Identity based cryptography, while maintaing the compatibility with standard RSA and eliminating the mediator introduced by mediated RSA. We will show that this approach is convenient in terms of traffic overhead, perfectly applicable to existing trusted platform modules specifications and able to limit damages that both external and internal attackers can produce to the network. Copyright © 2013 John Wiley & Sons, Ltd.

KEYWORDS
TPM; RSA; distributed mediator; distributed network

***Correspondence**
L. Maccari, Department of Information Engineering and Computer Science, University of Trento, Via Sommarive, 14, Povo, Trento.
E-mail: maccari@disi.unitn.it

1. INTRODUCTION

A wireless pervasive network is a collection of technologies that enables the user to seamlessly communicate with devices embedded in the environment and offers enhanced possibilities of interaction.

In this model, the network consists of a mesh of heterogeneous devices. The devices perform machine-to-machine communications and interact with the users' terminals to create a distributed ambient intelligence.

In order of growing complexity, some of the components of the network can be sensors to monitor the environment, actuators to react to environmental changes, smart furnitures, surveillance, and security systems.

The user interacts with the network *explicitly*, in order to receive alarms and control the devices, or *implicitly*, as the devices he/she wears may change the behavior of smart objects in the environment. Moreover, the network could allow external users or entities to communicate with the devices, for example, the security personnel or other similar networks, in order to enforce the so-called Internet of Things paradigm. For a very comprehensive survey on pervasive networking and its challenges, refer to [1].

A programming model that, in our opinion, should be used in this context is represented by software agents [2] and, specifically, by mobile agents [3,4]. Mobile agents are programs that roam from one platform to another and interact with a software middleware present in the nodes; they perform their assigned tasks and report the results. Because the hardware platforms used are generally resource limited and the tasks to be performed are dynamic in time and space, the creation and distribution of dedicated software can make the network more flexible. As a consequence, mobile agents in distributed networks are an active field of

research. In ad-hoc networks, they are studied to support service discovery [5], routing [6], and intrusion detection [7]; while in wireless sensor networks (WSN), they prove to be useful because they allow to achieve a better monitoring of events [8]. Recently, software implementations of mobile agents have been made [9] and have been tested on real devices to show their potential [10].

One of the main challenges in these networks is to make sure that only the allowed entities can gather data from the network and run specific tasks on the various networked entities. Security aspects of mobile-agents are very challenging as in such an exposed network; the code can be stolen and analyzed, the credentials it carries can be re-used, rogue agents can be injected in the network through the gateway or a compromised host, and so on. Therefore, whenever a control center or a device receives data from an agent, it must be able to verify that it comes from a well-behaving agent and from a specific node. The source agent and the source node of the data are two key factors to be verified.

In order to enhance the security of the network, we propose to combine standard trusted platform module (TPM) chips, present in many commercial devices, and identity-based cryptography (IBC) concepts. The former will provide robustness, while the latter allows to use public-key cryptography without the burden of a public-key infrastructure (PKI).

Using these components, we design a novel approach that provides assurance on the integrity of the agent as well of the hosting node, with a much lower impact on network performance in comparison with standard encryption techniques.

2. SCENARIO AND STATE OF THE ART

We are considering a network of devices interconnected by a wireless mesh network. The devices are sensors or actuators, and they carry either sensing devices or actuators in order to perform their specific tasks. In such a scenario, the wireless links are inherently insecure, and the devices could be stolen or modified (i.e., fall in attacker's hands). Their own duties could be modified, possibly affecting the behavior of the whole network.

Typically, in such networks, the device behavior is pre-determined at deployment phase by flashing their firmware. Such an approach is not flexible, as a reprogramming (possibly over-the-air) is needed to modify the device software. Furthermore, an attacker can use reverse-engineering to acquire the knowledge of the whole network.

In order to solve the previously mentioned issues, the mobile agent paradigm can be used. Mobile agents allow a node to quickly change its behavior while limiting the capability of inspection of node programmed functions by an attacher. Therefore, we consider a system where the devices are either *sleeping*, when they have little or no functionalities in their firmware, or *active*, when they have one or more active mobile agents running. As soon as a mobile agent is no more needed in a particular device, it should roam to a new device or delete itself.

The selected scenario can be easily mapped into different real-world situations—for example, outdoor events monitoring, home automation, tactical battlefield, and so on. In an outdoor event, for example in a stadium, number of sensors or actuators might be deployed in the area, each of them being networked through a mesh wireless network. The mobile agents will be spawned in the network in order to perform a 'background' monitoring, with specialized agents spawned only on-demand (for example, if there is a particular event to monitor more accurately).

Regardless of the specific scenario, mobile agents will have to communicate between each other, in order to increase their event awareness and with external data centers. Moreover, they will need to validate the device they are roaming to, in order to avoid potentially compromised devices.

This kind of network carries many sensitive information and, consequently, can be a valuable target for attacks aimed to modify the behavior of the nodes or to access the data that are carried. Unlike traditional wired networks or home wireless networks, this kind of network has a much larger and undefined perimeter. An attacker can access it through the gateways exploiting insecurities on the remote connections or directly attacking the wireless devices. Once a device has been successfully attacked, the mobile code, which will be run on that host, will be compromised as well. If the code of a mobile agent is analyzed, the credentials it carries can be re-used by the attacker. In this way, the attacker could inhibit the triggering of reactions, inject rogue agents and, thus, finally bypass the security measures.

For this reason, the authentication of the source agent and the source node of the messages are two critical factors. Whenever the control center or any other device receives data from an agent, it must be able to verify, on one hand, that the information comes from a well-behaving agent and, on the other, that it comes exactly from the declared source node.

The solution we propose in this paper has the goal of securing the communication between all the entities of the network (devices, agents, and control center) by authenticating their messages and assuring that data come from non-modified agents. We take into consideration a very challenging scenario in which the lifetime of the agents is short. Therefore, new agents with fresh keys are often created and, for this reason, key caching is not considered. Our effort is focused on reducing the amount of traffic needed to move the keys, while preserving the security level.

The proposed design is intended to resist to two levels of attacks. Firstly, if a malicious agent is introduced in the network, it must be impossible for it to send valid information to other agents and to receive messages from them. Secondly, if an attacker gains administration rights on a node, even if he could be able to access the

DOI: 10.1002/sec

information elaborated by the agents on that node, he would not be able to re-use this information in other hosts that he does not control. This will slow the penetration speed of a successful attack.

Summarizing, in this paper, we propose a method that will

(1) Perform agent and host authentication, with host cryptography operations performed by the local node using the TPM devices.
(2) Use IBC in order to avoid the overhead due to distribution and usage of RSA certificates.
(3) Use the TPM to distribute the mediator introduced by the mediated-RSA scheme (see next subsection).

We designed our solution to be compliant with another requirement; it must be compatible with existing state of the art software and hardware, namely RSA primitives and standard trusted computing hardware.

In the following part of the section, we will outline those technologies and why they are not suitable for our goal.

2.1. Certificate-based security

Using two distinct RSA keys is a straight solution to certify both the mobile agent that is the generator of a message and the node where the agent is running. In this case, each network node and each agent must be equipped with a valid certificate; each message is first signed with the agent private key then encapsulated in another message signed with the node private key. This strategy has a negative impact on network performance, because it introduces four encryption/decryption operations and largely increases the size of the message due to the initial exchange of certificates.

Using RSA with only one encryption could be a simpler strategy. This approach provides the authentication of only the source agent or the source node, and it is the one with the least impact on network performance. We will use this approach as a benchmark for the simulations of Section 5 to show that our solution gives a comparable impact on network resources while assuring a higher security level.

These approaches have two main drawbacks: on one side, valid certificates must be exchanged whenever a message is sent for the first time, thus increasing the delay; on the other, whenever the attacker gains possession of a host node or of the agent's mobile code, he can access the secret part of the public/private key couple.

Our solution approaches these issues relying on IBC, with no need of any key or certificate exchange, and using the TPM to seal private keys.

2.2. Identity-based cryptography and mediated RSA

The problem of the exchange of public keys between hosts has been approached in the last decade with identity-based algorithms. IBC has been introduced in [11] and analyzed in [12] and [13]. It substitutes the public key of a user with the expansion of a unique ID directly related to the user. For example, when sending a ciphered email with IBC, the sender will use the destination email address as one of the inputs to a one-way function and use the result as a public key. This approach eliminates the need for certificates and for the protocols necessary to exchange them, such as transport layer security (TLS). The adoption of IBC has several side-effects widely analyzed in literature. The main one is the fact that the user can not create its own keys. A dealer will have to generate the public and private keys for all the users. Another limitation of IBC is that the cryptographic primitives used are incompatible with widely used RSA algorithm. Mediated RSA (mRSA) is an attempt to adapt the IBC architecture to standard RSA cryptographic primitives.

2.2.1. Mediated RSA.

An mRSA is presented in [14], using well known RSA primitives, in order to achieve IBC (refer to Table I for the definitions of function's names and variables used in this paper).

The mediated RSA concept is based on splitting the keys between the user and a trusted *SEcurity Mediator* (SEM). This basic approach has several drawbacks. For example, it introduces a single point-of-failure in the system and high delays in wireless mesh networks because of the fact that the mediator must be involved in all message exchanges. There are several works (Section 3) that tried to alleviate this issue. In our approach, the SEM duties are split up into local functions performed by the TPM, thus, eliminating both issues mentioned previously.

In mRSA, the public key e_i of user U_i is generated as a hash function h applied to the user ID, $e_i = h(U_i)$. Given the global parameter n (the size of the RSA modulus), the private key d_i is generated inverting e_i (similarly to standard RSA) subject to $e_i d_i = 1 \bmod \phi(n)$ where $\phi()$ is the Euler's totient function. Note that a user U_i owns a key that

Table I. Reference table for the parameters and functions.

Symbol	Meaning
A_i	Agent i (or ID of agent i)
H_k	Host k (or ID of host k)
$h()$, $g()$	hash functions
d,e,n,p,q	RSA parameters
d^u, d^{sem}	mRSA private keys
$m, m\|0$	plaintext message, padded message
c	ciphertext message
$e_{i,k}, d_{i,k}$	public, private key of A_i on H_k
$OS2IP()$	string to integer conversion
$P(), P^{-1}()$	padding functions
e_k^{tpm}, d_k^{tpm}	a valid public and private key for the TPM of host k
$h(P_i)$	a hash performed on the executable code of process i (i.e., agent i)

ID, identity; TPM, trusted platform module.

refers to the same n of any other user U_j, and that user U_i can generate the public key of U_j just by the knowledge of the user ID of U_j.

Nevertheless, the security of the RSA algorithm resides also in the fact that each public/private key must be based on a distinct n value. It can be easily shown that, given the knowledge of a couple d_i, e_i and n, the prime numbers p and q that generated n can be recovered (the proof is reported in Appendix A.1). Given this, the knowledge of d_i, e_i allows any user to invert any key d_k or e_k that has been generated using the same modulo n. This problem is solved by mRSA by splitting d_i in two parts, $d_i = d_i^u + d_i^{sem} (\mathrm{mod} \phi(n))$. The first half is given to the user, while the second is kept by a mediator. A cipher-text c derived from the encryption of message m with public key e_i will be decrypted both by the user and the mediator. They will generate two values $c^u = c^{d_i^u}$ and $c^{sem} = c^{d_i^{sem}}$. The multiplication of these two values yields the original message because $c^u c^{sem} = c^{d_i^u} c^{d_i^{sem}} = c^{d_i^u + d_i^{sem}} = c_i^d = m^{ed_i} = m(\mathrm{mod}\ n)$.

Thus, in the mRSA algorithm, the user only knows half of the private key d_i, and can not derive p, q. The drawback of this approach is represented by the mediator, as it becomes the bottleneck for the protocol. Each time a private key must be used for signing or for decrypting, the mediator must participate in the data exchange.

This has a heavy impact in the considered distributed scenario, because the communication overhead would lead to battery depletion and shorter lifetime of the whole network. Moreover, because large delays would be introduced by multi-hop transmissions, the end-to-end delay would be severely affected.

2.3. Trusted platform module

In order to describe the details of the proposed solution, we need to introduce another building block, the TPM. The TPM architecture was defined by the Trusted Computing Group and published as a standard in [15] (see [16] for an overview and [17] for details on its internals). TPM chips are often used in commercial devices (estimated in mid-2010 to 250 000 000 installed units, [18]) and are the basis of complex professional security systems.

The TPM chip is useful for our purposes because of the following:

(1) It has the ability to store encrypted data that can not be accessed by the user without using the application programming interfaces (APIs) provided by the chip.
(2) It can perform cryptographic functions such as RSA encryption and signature, Secure Hash Algorithm (SHA) hashes and Advanced Encryption Standard (AES) encryption.
(3) It includes a unique RSA public and private key not readable by the user but usable with the APIs (the key is called endorsement key (EK)). From this key, when the chip is initialized by the user, a new RSA key pair is generated, the *storage root key*.
(4) It can generate new RSA keys organized as a hierarchical tree, where each father key is used to securely store the child key. The storage root key is used to securely store the root of the tree.
(5) A user can import a key as a blob encrypted with one of the valid TPM public keys. The user might be asked an optional password to import and use the key.
(6) Because the internal memory of the chip is limited, an external memory can be *sealed* using crypto keys that works as a secure storage for the chip data.
(7) Other TPM-aware devices or software can communicate with the TPM. The BIOS of the computer can perform an integrity hash of the operative system kernel and verify it using the TPM before booting. The same can be performed by the operating system to run another software and, in turn, by any program spawning new child processes. With a certain level of complexity, this creates a chain of trust ensuring that all the software running on the host has not been tampered by an attacker.

In Figure 1, a simplified graphical representation of a TPM is reported. For our purposes, the most interesting feature of the chip is that a user can store crypto-blobs into the TPM. Crypto-blobs contain RSA public/private keys encrypted with one of the valid keys previously stored in the TPM. Being ciphered, a crypto-blob can be carried even over an insecure channel. Once the key is stored in the TPM, the user can access the TPM with the appropri-

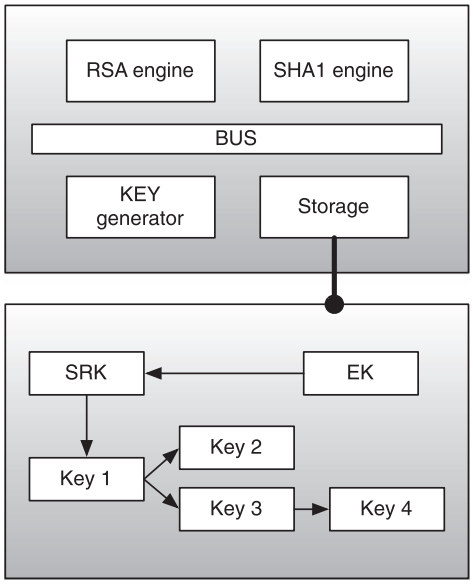

Figure 1. Simplified trusted platform module structure and key management.

ate APIs. APIs do not reveal the private part of the key; however, they give the possibility of using it to sign and decrypt data.

The knowledge of the top-level EK allows the recovery of all the key-tree stored on the chip. This is a prerogative of the manager of the device, that is the one who initially enables the chip and sets a master password. For this, operation physical access to the device is necessary. The manager can also remotely perform operations on the TPM securely, because the standard provides secure communication protocols. In our scenario, the chips are enabled off-line before network deployment.

In this paper, we do not take into consideration the case of a manager that discloses the master password or the EK key.

2.3.1. Trusted platform module performance.

To verify its real applicability, TPmRSA (described in the following) has been implemented using the Trousers open source library and a software TPM chip emulator largely used in most of the compatibility tests in literature. The implementation of TPmRSA has confirmed the applicability of the mRSA algorithm to the TPM primitives.

The time needed for encryption or signature using a TPM device depends on the device chosen. In [19], benchmarks on various TPM chips show that RSA signature with 2048-bit keys takes a time between 300 and 500 ms, while for 1024-bit keys it takes between 100 and 206 ms. Specifically, the AT97SC3203 chip shows a performance of less than 100 ms computation time for a 1024 bits RSA signature (in line with what reported by its data sheet). It is interesting to note that the same chip is the one that has been mounted on a sensor board by the authors of Shih *et al*. [20], showing computation time and energy consumption compatible with sensor requirements. This performance is generally outperformed by encryption operated on the main processor of a standard PC. Nevertheless, nodes that form our target scenario (wireless routers, mobile terminals, sensors. . .) are not equipped with standard desktop hardware, and their performance is comparable to performance of a TPM chip. It is also worth noticing that on the target scenario, the security requirements can generally be relaxed in comparison to other more common applications. To break a 768-bit RSA key, the authors of Kleinjung *et al*. [21] spent two and a half years on hundreds of machines[†]. Because the time to break such a key can be longer than the lifetime of the network itself, using short RSA keys may be an affordable risk.

[†] The authors of **[21]** report that "We spent half a year on 80 processors [. . .] This was about 3% of the main task, the *sieving,* which was performed on many hundreds of machines and took almost two years. On a single core 2.2 GHz AMD Opteron processor with 2 GB RAM, sieving would have taken about fifteen hundred years."

3. RELATED WORKS

The security of mobile agents is an active field of research as analyzed in [22]. In some cases, it has been coupled with trusted computing, as in [23]. Trusted computing has been proposed also to leverage the security of wireless mobile ad hoc networks [24–26]. Recently, it has been proposed for WSN mixing trusted and untrusted hosts [27]. It has even been ported to WSN hardware showing sustainable performances [20]. IBC has been proposed for wireless mesh networks [28]. However, only two works, to our knowledge, try to mix trusted computing and IBC. In [29], the authors present a scheme that implements an ID-based encryption scheme coupled with the use of TPM chips. The scheme is aimed to implement the secure extraction of IBC keys by a user using the trusted infrastructure given by TPM chips. The authors do not use standard TCG hardware, because the standard hardware does not expose the needed primitives to correctly implement IBC. Our solution relies on the IBC concept and correctly runs on standard hardware using just plain RSA functions. We also overcome the difficulties that lead the authors of Gaun *et al*. [29] to use a custom hardware-accelerated encryption and storage platform. In [30] is described a platform for implementing a distributed security-mediated PKI using a peer-to-peer system. The work includes the use of a trusted computing platform in order to secure the communications between couples of *islands* and form a distributed PKI. The authors implemented a trusted chain of software outside the TPM chip to overcome the limitations of the hardware. Moreover, their specific application has different constraints. As a consequence, it focuses on different security aspects (such as distribution of the PKI and replication and migration of SEM capabilities) that do not apply to our work.

None of these works is focused on distributed wireless networks. On the contrary, our paper presents and evaluates a framework that is usable in such a context.

At the *blackhats* conference, in 2010, it has been shown that with a 200K$ microscope, it is possible to extract data from an Infineon TPM chip[‡]. Such an attack requires the physical extraction of the TPM chip and its analysis with a special hardware. We do not take into consideration this possibility for two reasons: firstly, because the attacker must physically steal and extract a network node and, secondly, because the total cost of the attack may be higher than the whole cost of the network we describe in our scenario.

4. TRUSTED PLATFORM MODULE-BASED MEDIATED RSA

The core of our proposal consists in using IBC concept as in mRSA, in order to avoid the exchange of certificates.

[‡] http://www.flylogic.net/blog/

We also substitute the centralized mRSA mediator with a distributed mediator represented by TPM devices. In this way, the mRSA approach is preserved, and the problem of the centralized mediator is avoided. Each agent will carry only half of the secret needed to authenticate a message and will use the TPM in its host device to access the other half.

We will show that a plain application of mRSA to TPM devices is impossible. Because our goal is to be compliant with current TPM specifications, we will propose two modifications that will achieve the same goal.

In the first solution, named TPmRSA, we maintain the mRSA idea, but we use classical encryption functions that do not need key splitting. In the second solution, named TP*RSA, we effectively use mRSA algorithm adapting it to the TPM context.

4.1. Mediated RSA on trusted platform module devices

As a first step, we designed the implementation of mRSA on TPM compliant devices as follows:

(1) At network setup, for each agent A_i and host H_k, a set of key couples $(e_{i,k}, d_{i,k})$ is created:

- $e_{i,k} = h(A_i, H_k)$ is generated using IBC;
- $d_{i,k}$ is generated inverting $e_{i,k}$ by using the secret parameter $\phi(n)$.

The n parameter must be public and fixed. Consequently, to each agent, A_i will correspond a set of public keys $e_{i,k}$, each one to be used on host k. For every host H_k, $d_{i,k}$ is split with mRSA and distributed for its first half $d^a_{i,k}$ to A_i and for its second half $d^{tpm}_{i,k}$ to the corresponding TPM in H_k.

(2) $d^{tpm}_{i,k}$ is moved to the corresponding host in the network, while every agent will be equipped with the $d^a_{i,k}$ for each host it is allowed to move.

(3) Any agent A_j on host H_l, when sending a message to A_i on host H_k, will derive the corresponding public key as $h(A_i, H_k)$ and send the ciphered message to H_k where A_i resides.

(4) A_i on host H_k will decrypt the message with $d^{tpm}_{i,k}$ using the TPM API and in parallel with $d^a_{i,k}$ that it owns. Similarly, the message can be also authenticated using the two halves of the private key $d^a_{j,l}$ and $d^{tpm}_{j,l}$.

(5) Optionally, the $d^{tpm}_{i,k}$ could be reduced to a d^{tpm}_{k} removing the dependence over the agent. This would simplify the key management on the hosts but would downgrade the host-based authentication to a simple network-based authentication shared for any host.

Such a solution has many advantages over classical RSA or mRSA. Firstly, it does not rely on certificates, so that any agent is able to communicate with any other agent on any other host using its public key that can be easily derived by the IDs. Secondly, it does not rely on a centralized mediator, thus it does not introduce bottlenecks. It is important to notice that with mRSA, the mediator could be contacted twice during the communication, for signing the message before it is sent and for decryption when it is received.

If the $d^{tpm}_{i,k}$ is agent-dependent, it is necessary to distribute this key to H_k whenever a new agent is generated. This can be performed by the agent itself carrying keys in crypto-blobs, as explained in Section 4.2, or with keys pre-distribution, as explained in Section 4.3.

The security features of the crypto functions correspond to the underlying mRSA scheme. Specifically, no single agent is in possession of a complete $(e_{i,k}, d_{i,k})$ couple, thus, can not factor n. Consequently, $\phi(n)$ remains secret and the generation of key-couples is possible only for the network manager.

The main obstacle to this approach is that the TPM does not export to software libraries, a function that performs plain RSA mathematical functions, forcing the agent to apply standard procedures for encryption and decryption, i.e., public-key cryptography standards (PKCS). PKCS is a suite of standards used to design a formal protocol from the mathematical primitives used in public key encryption. One of the features introduced in PKCS is the use of optimal asymmetric encryption padding (OAEP), a padding and randomization function.

Given a plain-text m, a random number r and two hash functions $h()$ and $g()$, OAEP introduces the following procedure[§]:

(1) Add to m a zero sequence padding to reach the length necessary for integer exponentiation.
(2) Expand r to the length of padded m using g, combine it with padded m.
(3) Reduce the result of the combination using h and recombine it with r. More formally

- $m \rightarrow m|0$
- $X = g(r) \oplus (m|0)$
- $Y = h(X) \oplus r$

(4) encrypt and transmit X,Y

For decryption, do

(1) decrypt X,Y
(2) recover $r = Y \oplus h(X)$
(3) recover $m|0 = X \oplus g(r)$
(4) remove padding: $m|0 \rightarrow m$

[§] For the sake of clarity we omit some details of OAEP irrelevant to our context

DOI: 10.1002/sec

This procedure ensures that encrypting twice the same content will produce a different cipher-text in order to avoid reply attacks.

Using the TPM, all the steps of the decryption are implemented in a single function in the TPM API that returns the decrypted block. This last step makes it impossible to use mRSA on TPM devices. Note that the padding is added by the sender to m, encrypted with $e_{i,k}$ key and transmitted to H_k where A_i resides. When the decryption is performed by the TPM device, the $d_{i,k}^{tpm}$ key is used, but the decryption will not produce the cleartext. Instead it will produce a partially decrypted message to be combined with the other partial decryption performed by the agent.

Consequently, the decryption operated by steps 2 and 3 will lead to a non-padded decrypted message, so that step 4 in the decryption fails and the API returns an error state. It is not possible to mangle X,Y at the receiver, before the decryption, to produce a valid padded message, because the decryption is performed with $d_{i,k}^{tpm}$, which is a quantity stored only in the TPM. The possibility of blindly modifying a cipher text in order to selectively alter the corresponding cleartext would be contrary to the security of the RSA scheme itself. Similarly, it is impossible, starting from m, to craft a cipher text that, once partially decrypted with $d_{i,k}^{tpm}$, is OAEP compliant.

Concluding, this solution may be applicable to other hardware crypto devices giving a more fine-grained control on the mathematics involved in RSA. However, it is not directly portable to the TPM. Our two alternative solutions are described in the rest of this section, we named them TPmRSA and TP*RSA.

TPmRSA does not use key-splitting. The agents can not use the crypto-blobs they are equipped with and they must rely on the mediator. Therefore, it can be classified as an extension of the concepts introduced by the original mRSA. TP*RSA, on the contrary, uses key-splitting in a similar way to the original mRSA. For the sake of simplicity, in this paper, we will use the term *mediated* in both cases .

4.2. TPmRSA

TPmRSA combines IBC and the basic ideas of mRSA in a perfectly compatible way with TPM current hardware. Briefly, this approach consists in using IBC to generate keys for (A_i, H_k) that agent i carries in a ciphered blob. Each blob can be decrypted only in the TPM of H_k. In detail, the procedure works as follows:

(1) Each host H_k in the network is equipped with one public and private key e_k^{tpm}/d_k^{tpm}. Both keys are stored in the TPM. This procedure is performed only once at network set-up.
(2) For each agent A_i and host H_k, a set of key couples $(e_{i,k}, d_{i,k})$ is generated, $e_{i,k} = h(A_i, H_k)$ is generated using IBC and $d_{i,k}$ is generated inverting $e_{i,k}$ using the secret parameter $\phi(n)$. The n parameter must be public and fixed.
(3) Each $d_{i,k}$ key is encrypted in a blob using e_k^{tpm}, each A_i is equipped with the blobs corresponding to the hosts it is allowed to roam to. When A_i moves to H_k, it will carry the corresponding $d_{i,k}$ and store it into the TPM. Once stored in the TPM, the access to the key is conditioned by the possession of a password. The password is set to $h(P_i)$, where P_i is the executable code for agent A_i. The password is stored inside the blob itself and the blobs are contained in the data code of A_i.
(4) Whenever a second agent A_j will need to communicate with A_i residing on H_k, it will compute the ID-based public key $e_{i,k} = h(A_i, H_k)$ and use it to cipher the messages directed to A_i on host H_k.
(5) Agent A_i will be able to decrypt the message with $d_{i,k}$ stored in the TPM using the password set to $h(P_i)$.

Any message sent by an agent can only be decrypted by the intended receiver on the specified host and by nobody else. If A_i roams to another host H_l, it will load the corresponding key without needing to contact the network manager. With trivial modifications, the messages can be authenticated too.

To reduce the size of the keys carried by the agent, keys can be left on the corresponding host after the first visit. They will be purged by the host after a fixed maximum lifetime of the agents.

The usage of $h(P_i)$ to unlock $d_{i,k}$ binds the agent to the key. The blobs can be generated only by the network manager that is in possession of $\phi(n)$ and can create valid public/private keys. When the blob is generated, also, $h(P_i)$ is stored inside the blob, so that the blob contains a fingerprint of the original code of A_i. When A_i wants to use $d_{i,k}$, the middleware that manages the mobile agents can perform a hash on the executable code of A_i and verify that the $h(P_i)$ unlocks the key in the TPM for A_i. In this way, the agent and the crypto blob are bound to each other, and it is not possible for an attacker to steal a valid crypto blob and use it on a malicious agent.

As in the previous case, certificates are not needed because IBC is used. Consequently, the communications are set-up without the initial overhead needed, for instance, for the TLS protocol.

4.3. TP*RSA

In [31], another variation of the classical RSA algorithm is proposed, multiplicative RSA or *RSA. In *RSA, a private key d is split in two halves using multiplication instead of sum so that $d = d^a d^{tpm}$ and consequently $ed^a d^{tpm} = 1 \bmod \phi(n)$. *RSA can be used instead of mRSA to produce a mediated RSA compatible with TPM devices, which we called TP*RSA. The detailed procedure is as follows:

(1) Each host H_k in the network is equipped with one public and private key e_k^{tpm}/d_k^{tpm}. Both keys are

stored in the TPM. This procedure is performed only once at network set-up. Each H_k is equipped also with a new key couple e^{tpm}/d^{tpm} that is the same for each host.

(2) This step is the same as TPmRSA

(3) $d_{i,k}$ is divided by d_k^{tpm} and $d_{i,k}^a$ is generated (thus $d_{i,k} = d_{i,k}^a d_k^{tpm}$). $d_{i,k}^a$ key is encrypted in a blob using e^{tpm}. Each A_i is equipped with the blobs corresponding to the hosts it is allowed to roam to. When A_i moves to H_k it will carry the corresponding $d_{i,k}^a$ and store it into the TPM. Again, a password set to $h(P_i)$ is used to access the key. In this case, though, A_i will be authorized not only to use but also to recover $d_{i,k}^a$ in clear.

(4) This step is the same as in TPmRSA.

(5) Agent A_i will receive the message m, decrypt it in software with $d_{i,k}^a$, thus obtaining a partial decryption $\hat{c}$ to be decrypted again inside the TPM with d_k^{tpm}.

In our opinion, there are two important aspects that should be taken in consideration. The first one is that the problem given by OAEP identified in Section 4.2 is avoided. In fact, the decryption performed by the TPM device comes after the first decryption performed by the agent. The first partial decryption is performed in software so OAEP checks can be avoided. As a result, the TPM device decrypts $\hat{c}$, and the output is a valid padded data. The second issue is that the encryption with a common key e^{tpm} is necessary to avoid that, whoever is able to sniff the wireless traffic, can steal a valid $d_{i,k}^a$. It is a measure needed not to hide $d_{i,k}^a$ from A_i, but to hide it from potential passive attackers. A_i indeed needs to know $d_{i,k}^a$, because the first decryption must be performed out of the TPM. The important point is that A_i, once leaving H_k, must not carry an unencrypted key on the wireless media.

4.4. Security analysis

To analyze the security features of TPmRSA and TP*RSA, four kinds of attackers are introduced:

(1) An external passive attacker, that is, a human attacker sniffing the traffic on the wireless channel;

(2) An external active attacker, that is, an agent that is able to enter the network and roam between hosts but is not equipped with any valid credentials;

(3) An internal attacker agent, that is, an agent equipped with valid keys, modified in order to change its original behavior;

(4) An internal human attacker that is able to take full control of the operative system of a host, that is, to gain root access on host k.

The first attacker will not be able to intercept any valid traffic because it has been encrypted with public keys. All the communications are sent to a specific host and agent ID, so the attacker has no valid key to decrypt messages. A valid ID and the corresponding keys (both in TPmRSA and TP*RSA) are needed to authenticate any message to be sent to another agent. Also in this case, the attacker is not able to interact with the network. For the same reason, an external attacker is not able to forge valid messages to be injected in the network. Because we do not target any specific application, it is out of the scope of this paper to detail the transport protocol used to convey the information. With well known techniques (counters, nonces, etc.), it is possible to avoid reply attacks from the external attackers.

An internal attacker agent will not be able to interact with the rest of the network, because the password needed to unlock its own $d_{i,k}$ key on host k is verified by the host node using the hash of the executable code of the process. The agent will not be able to store its private key on any host or use any key already stored on any host. Therefore, it will be isolated from other agents in the network.

An internal human attacker that can take full control of a host is generally impossible to stop. In a typical networking scenario, if one of the core servers is owned by the attacker, the whole network is at risk. In our scenario, the attacker can not extract the private keys from the TPM. He will be able to analyze the code of any agent running on that host and steal their ciphered blobs. The attacker may also bypass the checks on the integrity of the running code that is performed in software on the host. Nevertheless, the attacker can not use the stolen credentials on other hosts he does not control, so the penetration speed and potential damage of its attack are reduced.

If the kernel of the operative system is TPM-verified, the attacker will not be able to use the blobs even on the compromised host, because he can not execute a non-verified code.

As for the cryptographic properties, we are using standard RSA as implemented in TPM; therefore, we rely on its security. As with mRSA, the n modulus is the same for any public/private key couple, but there is no agent in possession of a complete public and private key. This prevents the agents from factorize the values p and q.

When we introduced TPmRSA, we introduced the possibility of using only a network-wide key d_k^{tpm} instead of a network-based and host-based keys. With TP*RSA, such a solution would result into two important security weaknesses allowing an attacker to factor n. The first one can be mitigated using a correct TPM configuration. However, the second one can not be avoided. The resulting attack requires at least two cooperating malicious users.

The use of a network-based key should be considered unsafe in the following cases:

(1) When the users are able to collude and are in possession of valid keys.

(2) When the attacker is able to collect more than one valid secret key.

The proofs are reported in appendices A.2 and A.3.

5. SIMULATION RESULTS

5.1. Simulation scenario

A network simulator, based on the Omnet++ platform with the Inet module, has been made to compare TPmRSA with the security equivalent approach consisting in the use of two X509 certificates, one embedded in the process and another one embedded in the node sending the information. The solution with two RSA certificates has been chosen for comparison, because it is perfectly compatible with common hardware and software (contrary to pure IBC). It also fits better in a distributed scenario than mRSA (that requires an intermediary). It should be noticed, however, that any of the alternative solutions offer a lower security level than the one given by TPmRSA, because the agents carry the private keys with themselves. Thus, those keys can be stolen and intercepted on the wireless media, which is impossible in the case of TPmRSA.

The network is a simple 6×6 grid network, with geographical routing. The physical and media access control layers are modeled using IEEE 802.11 with free space propagation. The bitrate has been lowered to 0.5 Mbps in order to simulate a low-throughput network with mixed hardware, mesh nodes, and sensor/actuators nodes. The simulation environment represents a generic pervasive network as described in the introduction.

In the simulations, 60 agents are generated, one every 40 s and are distributed randomly in the network. They have a fixed lifetime (3600 s) and are allowed to roam on a limited number of nodes. The number of nodes in which they are allowed to roam is determined by a configuration parameter "r" that ranges from 1 to 12. The nodes are chosen randomly. An agent roams every 360 s. Each time, it moves a payload corresponding to a fixed execution code plus an overhead o_r, where o_r corresponds to one 1024 RSA key multiplied by r, in the case of TPmRSA, and to zero when using certificates.

Each node generates random alarms. When a process is present in a node and receives one of these events, it sends a message to another process in the network and receives an answer (message size is 128 B). Events are generated at time intervals defined by the parameter "m" ranging from 10 s to 600 s.

When using certificates, the inter-process communication is made of a 4-way handshake, because the two ends must exchange their certificates before they transmit any data. We call o_m the overhead for each handshake that is computed as the size of the four certificates. The size of a X509 certificate is variable, depending on the kind of attributes chosen. Using 1024 bit RSA keys and minimal attributes, we stripped down the size of two X509 certificates (including the signatures by the network Certification Authority (CA)) to a total overhead of 770 B. Note that we did not use standard TLS, which would require a TCP connection to be performed. Instead, we used a very simple UDP exchange with equivalent overhead, which is an underestimation of a real TLS exchange. When using TPmRSA, just two 128 B packets are sent so o_m equals zero.

Varying r and m, we compared the ratio $R_o = O_r/O_m$, that is, the sum of o_r computed on every time an agent roams, divided by the sum of o_m computed on every inter-process communication. Moreover, we compared the average set-up time of inter-process communication. Each configuration is repeated 20 times with different random seeds and ends after the expiration of all the lifetimes of the agents.

5.2. Numerical results

In Figure 2, we report R_o for all the scenarios considered. For the large majority of the chosen cases, the graph shows that there is an important gain using TPmRSA, with a decrease in total overhead up to 67%. In the bottom right corner, the gain in inter-process communication does not compensate the overhead because of the crypto blobs. This happens when the event generated by the nodes that triggers inter-process communication is rarely generated

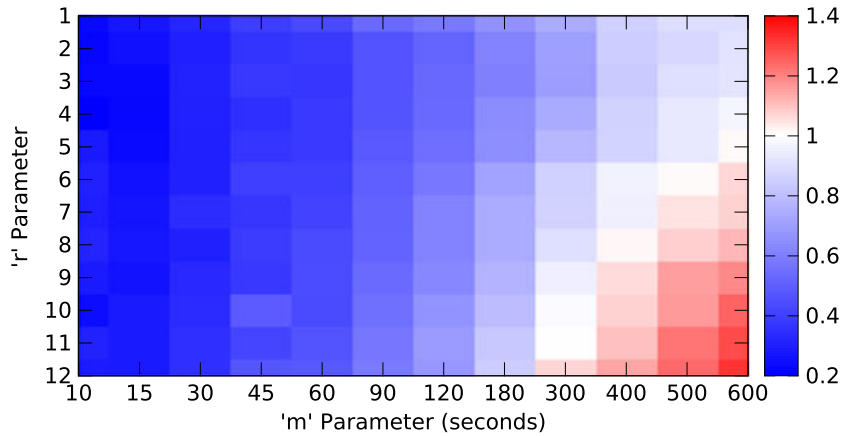

Figure 2. The R_o ratio measured varying r and m.

(more than once every 5 min) and the agents keep roaming in the network during their lifetime.

The ranges for r and m are set to show the limit of convenience of the proposed technique. Nevertheless, we believe that most of the real use-cases fall on the blue part of the map shown in Figure 2. This is because, in the simulated scenario, the traffic patterns are uniformly distributed and the endpoints of the communication are random as well. This is the worst case scenario. In realistic situations (which we plan to implement as future work), we imagine that a swarm of agents will concentrate in a limited zone in order to monitor a certain event that is happening in that area. In such a scenario, the nodes will eventually roam with less frequency and communicate more frequently to coordinate their action. We expect that, in this case, our security scheme will be even more efficient.

In Figures 3 and 4, we report R_o varying either r or m. It is also reported that the value of $R_o = 1$. These plots confirm that most of the considered simulations are able to work in the area below $R_o = 1$. They also show that the variation of the performance is very smooth. This indicates that the performance of the proposed algorithm shows a stable trend, enabling a network manager to fine-tune the parameters, depending on its specific application scenario.

While r is easy to predict (for instance, the number of nodes equipped with a specific sensor), m is much harder, because it represents the average time between two sensing events. For this reason, Figure 4 shows a very interesting behavior, even when m is large and the performance degrades compared to TLS, this degradation tends to increase with a sub-linear trend. As a consequence, if m changes in time and pushes the network state to the red area of Figure 2, the performances of TPmRSA do not sharply degrade.

Independently of the size of the network and of the traffic patterns, the inter-process communication with TPmRSA needs only a fraction of the time (in average the 17%) needed using certificates. This is due to the increased number and size of packets, and, consequently, increased probability of retransmission of the packets along the path

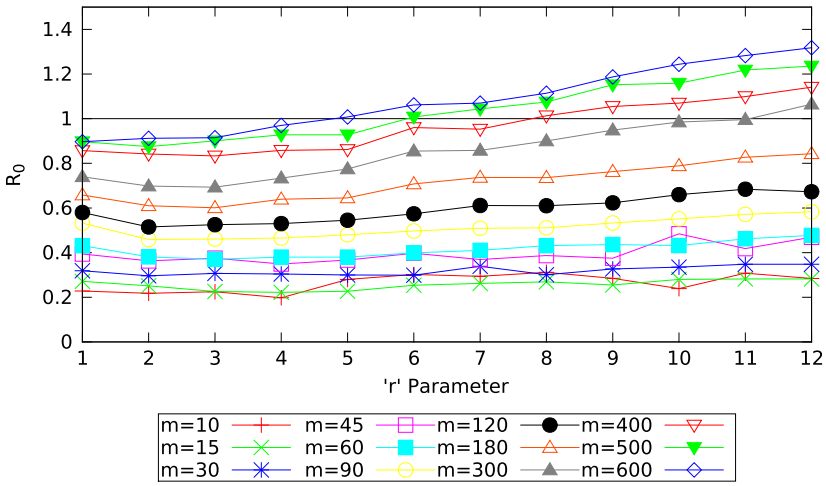

Figure 3. R_o plotted against m.

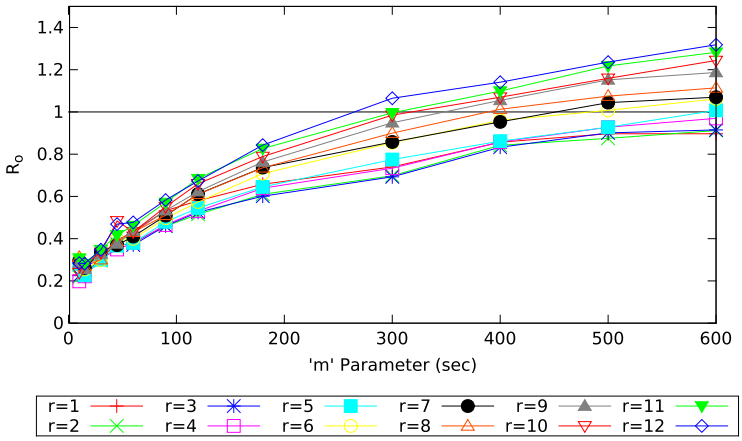

Figure 4. R_o plotted against r.

due to collisions at MAC layer. The reduction of the inter-process communication overhead guarantees a faster set-up and, thus, a faster reaction to alarms.

6. CONCLUSIONS

This paper presents a new method for data authentication and encryption suitable for wireless pervasive networks. The proposed methodology is based on TPM and is aimed at improving the communication performances and energy consumption by reducing the number of messages to be exchanged for a successful message decryption. The envisaged method can be applied to WSN as well as to other kinds of pervasive networks, without increasing significantly the hardware's cost. The simulation results demonstrated that the proposed scheme outperforms RSA certificate-based solutions in a wide range of scenarios. As further enhancements, we plan to integrate the TPM chipset in our WSN testbed in order to provide further measurements on energy consumption and communication delays.

APPENDIX A: PROOF OF THEOREMS

As an alternative to having a single private key per host, we considered to use a single private key per network d^{tpm} in TP*RSA. This would simplify the management of the crypto keys, although it introduces significant vulnerabilities to the system: if the credentials of two agents are stolen the network key can be recovered. In other scenarios, this limitations may be bearable (for instance with human users carrying keys into hardware tokens that can not reveal their credentials). We report the proofs here as reference. The first is well-known (see [32] for details), while the other ones are novel.

A.1. Factorisation of n

Given an RSA set e, d, n, it is possible to factor n and recover primes p, q.

Proof. Given d, e calculate $k = de - 1$, by construction k is a multiple of $\phi(n)$. Because $\phi(n) = (p-1)(q-1)$ it is even so exist r odd and $t \geq 1$ such that $k = 2^t r$. For any $g \in Z_n^*$, it is true that $g^k = 1 \bmod n$ so that $g^{k/2}$ is a square root of unity modulo n. By the Chinese Remainder Theorem, 1 has four square roots modulo $n = pq$. Two of these square roots are +1 and –1. The other two are $+x$ and $-x$ where x satisfies $x = 1 \bmod p$ and $x = -1 \bmod q$. If g is chosen at random from Z_n^* then with probability 1/2 (over the choice of g), one of the elements in the sequence $g^{k/2}; g^{k/4} \ldots g^{k/2^t} \bmod n$ is a square root of unity. Given x, n can be factored calculating $\gcd((x\text{-}1), n)$. □

Once p, q are revealed, given any e_i calculated using the same modulus n, it can be inverted on $\phi(n)$ to find the corresponding d_i.

A.2. Single user attack

Given e, d^u, d^{tpm}, and e^{tpm} (user and host indexes have been omitted for clarity) such that:

$$ed^u d^{tpm} \equiv 1 \bmod \phi(n) \quad \text{(A1)}$$

$$e^{tpm} d^{tpm} \equiv 1 \bmod \phi(n) \quad \text{(A2)}$$

the knowledge of e, d^u, e^{tpm} allows to factor n.

Proof. From Equation A1 comes that if a is the multiplicative inverse modulo n of d^{tpm}, then $ed^u = a \bmod \phi(n)$ and $e^{tpm} = a \bmod \phi(n)$. Then ed^u and e^{tpm} can be expressed as $ed^u = a + k_1\phi(n)$ and $e^{tpm} = a + k_2\phi(n)$. Consequently, $ed^u - e^{tpm} = (k_1 - k_2)\phi(n)$ so that the quantity $ed^u - e^{tpm}$ is a multiple of $\phi(n)$. Proof in A.1 can be applied with $k = (ed^u - e^{tpm})$. □

A.3. Collusion attack

A user does not need to know e^{tpm} for the protocol to work. It is embedded in the TPM that can be instructed not to reveal it. Nevertheless, even without the knowledge of e^{tpm}, two colluding users may perform the same attack.

Given two sets of keys $e_1, d^{1,u}$ and $e_2, d^{2,u}$ and d^{tpm}:

$$e_1 d^{1,u} d^{tpm} \equiv 1 \bmod \phi(n) \quad \text{(A3)}$$

$$e_2 d^{2,u} d^{tpm} \equiv 1 \bmod \phi(n) \quad \text{(A4)}$$

two colluding users sharing their private keys can factor n even without the knowledge of e^{tpm}.

Proof. As for A.2, it is true that $e^1 d^{1,u} = 1 - d^{tpm} + k_1\phi(n)$ and $e^2 d^{2,u} = 1 - d^{tpm} + k_2\phi(n)$. Again, $e^1 d^{1,u} - e^2 d^{2,u} = (k_1 - k_2)\phi(n)$ so that $e^1 d^{1,u} - e^2 d^{2,u}$ is a multiple of $\phi(n)$. □

ACKNOWLEDGEMENT

This work has been partially financed by The Trentino program of research, training and mobility of post-doctoral researchers, incoming Post-docs 2010, grant #40101857.

REFERENCES

1. Poslad S. *Ubiquitous Computing: Smart Devices, Environments and Interactions*, 1st edn. John Wiley & Sons, Ltd: London, UK, 2009.
2. Vinyals M, Rodriguez-Aguilar JA, Cerquides J. A survey on sensor networks from a multiagent perspective. *Computer Journal* March 2011; **54**(3): 455–470.

3. Kwon Y, Sundresh S, Mechitov K, Agha G. Actornet: an actor platform for wireless sensor networks, *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, Hakodate, Japan, 2006; 1297–1300.
4. Fok CL, Roman G, Lu C. Mobile agent middleware for sensor networks: an application case study, *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, UCLA, Los Angeles, USA, 2005; 382–387. DOI:10.1109/IPSN.2005.1440953.
5. Meier RT, Dunkel J, Kakuda Y, Ohta T. Mobile agents for service discovery in ad hoc networks, *Advanced Information Networking and Applications, 2008. AINA 2008. 22nd International Conference on*, Okinawa, Japan, 2008; 114–121. DOI:10.1109/AINA.2008.78.
6. Marwaha S, Tham CK, Srinivasan D. Mobile agents based routing protocol for mobile ad hoc networks, *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, 2002; 163–167 vol.1, DOI 10.1109/GLOCOM.2002.1188062.
7. Kachirski O, Guha R. Intrusion detection using mobile agents in wireless ad hoc networks, *Knowledge Media Networking, 2002. Proceedings. IEEE Workshop on*, Kyoto, Japan, 2002; 153–158. DOI:10.1109/KMN.2002.1115178.
8. Chen M, Gonzalez S, Leung VCM. Applications and design issues for mobile agents in wireless sensor networks. *IEEE Wireless Communications* 2007; **14**(6): 20–26.
9. Fok CL, Roman GC, Lu C. Agilla: a mobile agent middleware for self-adaptive wireless sensor networks. *ACM Transactions on Autonomous and Adaptive Systems* July 2009; **4**(3).
10. Mercadal E, Robles S, Martí R, Sreenan C, Borrell J. Double multiagent architecture for dynamic triage of victims in emergency scenarios. *Progress in Artificial Intelligence* 2012; **1**(2): 183–191.
11. Shamir A. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in Cryptology*. Springer-Verlag New York, Inc.: New York, NY, USA, 1985; 47–53.
12. Boneh D, Franklin M. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing* 2003; **32**(3): 586–615, Extended abstract in Crypto'01.
13. Cocks C. An identity based encryption scheme based on quadratic residues. In *Proceedings of the 8th IMA International Conference on Cryptography and Coding*. Springer-Verlag: London, UK, UK, 2001; 360–363.
14. Boneh D, Ding X, Tsudik G. Identity-based mediated RSA, *Proceedings of the Third International Workshop on Information and Security Applications (WISA) 2002)*, Jeju Island, Korea, 2002; 12 pages.
15. Available from: http://www.trustedcomputinggroup.org/ [accessed on 26 September, 2013] trusted Computing Group.
16. Bajikar S. Trusted platform module (TPM) based security on notebook PCs-white paper. *Mobile Platforms Group, Intel Corporation* 2002; **20**: 1–20.
17. Kinney SL. *Trusted Platform Module Basics: Using TPM in Embedded Systems*. Newnes: Boston, USA, 2006.
18. Fisher DA, McCune JM, Andrews AD. Trust and trusted computing platforms. *Technical Report*, Software Engineering Institute, Carnegie Mellon, January 2011. Available from: www.cert.org/archive/pdf/11tn005.pdf [accessed on 26 September, 2013].
19. Huanguo Z, Zhongping Q, Qi Y. Design and implementation of the TPM chip j3210, *Trusted Infrastructure Technologies Conference, 2008. APTC '08. Third Asia-Pacific*, Washington, DC, USA, 2008; 72–78. DOI:10.1109/APTC.2008.8.
20. Shih WC, Hu W, Corke P, Overs L. A public key technology platform for wireless sensor networks. In *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*, SenSys '08. ACM: New York, NY, USA, 2008; 447–448, DOI 10.1145/1460412.1460496, (to appear in print).
21. Kleinjung T, Aoki K, Franke J, Lenstra AK, Thomé E, Bos JW, Gaudry P, Kruppa A, Montgomery PL, Osvik DA, Te Riele H, Timofeev A, Zimmermann P. Factorization of a 768-bit RSA modulus. In *Proceedings of the 30th Annual Conference on Advances in Cryptology*, CRYPTO'10. Springer-Verlag: Berlin, Heidelberg, 2010; 333–350.
22. Bürkle A, Hertel A, Müller W, Wieser M. Evaluating the security of mobile agent platforms. *Autonomous Agents and Multi-Agent Systems* April 2009; **18**(2): 295–311.
23. Hein DM, Toegl R. An autonomous attestation token to secure mobile agents in disaster response. In *Security and privacy in mobile information and communication systems*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer: Berlin Heidelberg, Germany, 2009.
24. Kuntze N, Fuchs A, Rudolph C. Reliable identities using off-the-shelf hardware security in manets, *Computational Science and Engineering, 2009. CSE '09. International Conference on,* vol. 2, Miami, Florida, USA, 2009; 781–786. DOI:10.1109/CSE.2009.30.
25. Jarrett M, Ward P. Trusted computing for protecting ad-hoc routing. In *Proceedings of the 4th Annual Communication Networks and Services*

Research Conference, CNSR '06. IEEE Computer Society: Washington, DC, USA, 2006; 61–68. DOI 10.1109/CNSR.2006.56.

26. Xu G, Borcea C, Iftode L. A policy enforcing mechanism for trusted ad hoc networks. *IEEE Transactions on Dependable and Secure Computing* 2011; **8**(3): 321–336, DOI:10.1109/TDSC.2010.11.
27. Yang Y, Zhou J, Deng RH, Bao F. Better security enforcement in trusted computing enabled heterogeneous wireless sensor networks. *Security and Communication Networks* 2011; **4**(1): 11–22.
28. Deng H, Agrawal DP. Tids: threshold and identity-based security scheme for wireless ad hoc networks. *Ad Hoc Networks* 2004; **2**(3): 291 –307.
29. Guan Z, Sun H, Chen Z, Nan X. Efficient identity-based key issue with TPM, *Young Computer Scientists,2008. ICYCS 2008. The 9th International Conference for*, Moncton, New Brunswick, Canada, 2008; 2354–2359. DOI:10.1109/ICYCS.2008.523.
30. Vanrenen G, Smith S. Distributing security-mediated PKI. In *Public Key Infrastructure*, Lecture Notes in Computer Science. Springer: Berlin Heidelberg, Germany, 2004.
31. Tsudik G. Weak forward security in mediated RSA. In *Proceedings of the 3rd International Conference on Security in Communication Networks*, SCN'02. Springer-Verlag: Berlin, Heidelberg, 2003; 45–54.
32. Boneh D. Twenty years of attacks on the RSA cryptosystem. *NOTICES OF THE AMS* 1999; **46**: 203–213.