Property-Based Taming of Lying Mobile Nodes

Mark Manulis* Ahmad-Reza Sadeghi

Horst-Görtz Institute, Ruhr-University of Bochum
D-44801 Bochum, Germany
mark.manulis@rub.de, sadeghi@crypto.rub.de

Abstract

Intelligent security protocols can verify whether the involved principals have properties that are defined based on certain functional and security policies. The property we focus on is the performance of mobile devices participating in a security protocol. In this context, the protocol should distribute the computation, communication and storage costs fairly among all devices. However, the protocol should foresee against cheating participants who may lie about their properties to gain advantage.

1. Introduction

Many security applications in practice are based on interactive computation among mistrusting parties. Basically, each party inputs own secrets to the protocol and receives the (secret) output at the end. The protocol has to guarantee the security requirements which can vary depending on the application. In this context it is important to ensure the trustworthiness of the involved principals with respect to certain (security) policies. More concretely, we want to enforce that a mobile device taking part in the security protocol has the desired *property*, otherwise the principal should not be granted access to the output of the protocol.

In this paper, we consider a group key agreement for heterogeneous mobile ad-hoc groups [6]. These are groups where principals are equipped with different kinds of mobile devices such as cell phones, PDAs, laptops. Obviously, performance capabilities vary among devices due to different hardware and software configurations. Taking this into account we are interested in a fair distribution of the computational load among the devices. The protocol in [6] distributes computation costs non-uniformly, i.e., more powerful devices have to bear higher costs for computation, communication and memory. Imaginable attacks in this scenario are that principals may cheat on the performance of their devices in order to save own resources at the expense of those devices that are less powerful, e.g. a kind of a DoS attack. In order to resist such attacks [6] introduces a new security requirement, called *performance honesty*. However, the protocol in [6] does not provide this requirement and is therefore vulnerable to this kind of attacks.

* Both authors were supported by the European Commission through IST-2002-507932 ECRYPT.

In this paper we propose a system model for mobile devices based on Trusted Computing (TCG specification [8]) which can be used to enforce the performance honesty in [6]. The idea is to distribute protocol operations within a device between trusted and non-trusted components. Note that as for PCs it is expected that in near future a wide range of mobile devices will be equipped with trusted hardware features.

2. Related Work

The solution in [6] assumes that every user is honest concerning his device' performance. Although [6] mentions two possible approaches against cheating, it does not provide any concrete realization.

The first approach is based on "*incentives*" and aims to discourage selfish behaviour of participants by making cooperation more attractive, e.g., [1] and [7]. Two flavours of such approaches have been introduced for packet forwarding: *reputation-based* and *pricing-based* schemes. In reputation-based schemes a misbehaving device looses its reputation that depends on the rate of the forwarded packets. Devices with lower reputation receive less services (punishment). In pricing-based schemes packet forwarding is treated as a service that may be priced, e.g., using virtual currency as in [2] or specific micro-payment systems as in [3]. Incentive-based approaches require high communication overhead to compute and update reputations of devices, or to handle the accounting. Besides that some pricing-based schemes still require tamper-resistant hardware components or virtual banks (trust authorities). Another problem with incentive-based approaches is that the performance can only be measured by the device itself (to the contrary, the packet forwarding ratio is measured by other devices during the communication with the challenged device).

Therefore, the second suggestion based on the trusted computing hardware, e.g., Trusted Platform Module (TPM) [8] seems more suitable and is also more efficient and effective.

3. Model

3.1. Basic Definitions and Conventions

We denote the set of devices (users) with $\mathcal{M} := \{M_1, \ldots, M_n\}$. The *performance ratio* μ_i quantifies the performance capability of a mobile device M_i,

Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA'06)
1550-445X/06 $20.00 © 2006 IEEE

and is the output of a benchmarking mapping that performs protocol specific network and cryptographic operations. A mobile device M_i is considered to be more powerful than M_j if $\mu_i > \mu_j$. A *performance ratio order*, denoted by P, of a mobile ad-hoc group is a sorted list of devices such that $\mu_i \geq \mu_{i+1}$ holds for any M_i, M_{i+1} with $1 \leq i < n$. A signature scheme is denoted by a tuple $(\mathsf{Sign}, \mathsf{Verify})$ (the key generation algorithm is omitted). With $\sigma \leftarrow \mathsf{Sign}_{SK_X}(m)$ we denote the signature of a party X on a message m using the signing key SK_X. The verification algorithm $ind \leftarrow \mathsf{Verify}_{PK_X}(\sigma)$ returns $ind \in \{true, false\}$. By $\mathsf{Hash}()$ we denote a cryptographic hash function. The deployed cryptographic primitives are assumed to be secure.

3.2. Requirements

The underlying security protocol should fulfill the specified security requirements stated for that protocol and the *performance honesty* requirement, i.e., that no principal is able to cheat on the performance ratio of its device. The following protocol is considered as a use case for our system model.

4. Short Review of μSTR-H Protocols

μSTR-H [6] is a contributory group key agreement protocol based on [4]. It computes the group key as a function of public contributions of all participants. μSTR-H consists of five main protocols: setup, join, leave, merge, and partition, that can be used to initialize the entire group and to handle dynamic events, respectively. These protocols provide semantic security against *passive* adversaries, i.e. it is computationally infeasible to distinguish between bits of the group key computed by μSTR-H and random bits. Additionally, assuming that all participants do not cheat on the performance of their devices μSTR-H protocols achieve *cost fairness* in heterogeneous environment, i.e., protocol costs are distributed between mobile devices according to their performance ratios.

μSTR-H protocols assume a public, reliable and authenticated broadcast channel within the group. Reliability can be achieved, for example, using reliable multicast protocols for ad-hoc networks, like RDG [5]. Authentication can be achieved using public key certificates, i.e., every participant M_i has its own private/public key pair $(skey_i, pkey_i)$ which is used for message authentication. The public key $pkey_i$ should be certified. In the following description we omit the indication of the authentication procedure, i.e., of signature generation and verification.

μSTR-H sorts all group members (devices) according to the performance ratio order P. For a better efficiency all operations in μSTR-H are done in a cyclic group $<G>$ of prime order t of points of an elliptic curve E over a prime or binary finite field $\mathbb{F}_q$. Every group member M_i has its own secret session random $r_i \in_{\mathcal{R}} \{1, \ldots, t-1\}$. The corresponding blinded value $R_i = r_i G$ (scalar-point multiplication equivalent to modular exponentiation in $\mathbb{Z}_q^*$) is public. Every M_i saves a list $\mathbf{R}_i = \{R_i, \ldots, R_n\}$. During the protocol every member M_i computes a set $\{k_i, \ldots, k_n\}$ where k_n is the desired shared group key. For each k_i there exists a public key $K_i = k_i G$. Note that $k_1 = r_1$ and $K_1 = R_1$. Each secret key k_i $(i > 1)$ can be computed in two different ways: $k_i = map(r_i K_{i-1})$ or $k_i = map(k_{i-1} R_i)$ where map is a point-to-integer mapping function. Every member M_i computes and saves a list $\mathbf{k}_i = \{k_i, \ldots, k_n\}$ (note M_1 saves $\{k_2, \ldots, k_n\}$). In order to handle dynamic events every member M_i $(i > 1)$ has also to save K_{i-1}.

Initialization: Figure 1 provides a full description of the setup protocol of μSTR-H based on the above preliminaries. For simplicity i corresponds to member's M_i position in P, and μ_i can be obtained from P.

- M_i selects r_i, computes R_i, and broadcasts (R_i, μ_i)
- M_i computes $P = (M_1, \ldots, M_n)$, finds own index i, and saves $\mathbf{R}_i = (R_i, \ldots, R_n)$. Additionally, M_1 computes $\mathbf{k}_1 = (k_2, \ldots, k_n)$, and broadcasts $(K_2, \ldots, K_{n-1})$.
- M_i $(i > 1)$ computes $\mathbf{k}_i = (k_i, \ldots, k_n)$, and saves K_{i-1}.

Figure 1. μSTR-H Setup

Due to the cost fairness requirement, less powerful devices have to compute less secret keys and save less blinded session randoms than more powerful devices that are located in the head of P. Figure 2 shows an example of a group of 5 members. Member M_3 uses its session random r_3, public key K_2 and blinded session randoms R_4, R_5 to compute secret keys k_3, k_4, and the group key k_5.

	K_2	K_3	K_4	
	k_2	k_3	k_4	k_5
R_1	R_2	R_3	R_4	R_5
r_1	r_2	r_3	r_4	r_5
M_1	M_2	M_3	M_4	M_5

Figure 2. Example of μSTR-H

Dynamic Events: A sponsor is used in μSTR-H to handle dynamic events. The sponsor M_s is temporarily chosen from the group according to the actual state of P. M_s changes own session random r_s to achieve the freshness of the updated group key and broadcasts public keys that are required by other members to update the group key. There always exists at least one member who can verify information sent by the sponsor. In case of join the new member is inserted in P. The sponsor of join is the highest-numbered member below the position j of the new member. If $j = 1$ then the sponsor is M_2. In case of leave the leaving member is excluded from P. The sponsor of leave is the highest-numbered member below the position d of the leaving member. If $d = 1$ then the sponsor is M_2. For more details on μSTR-H protocols for join, leave, merge and partition, as well as its security analysis we refer to [6].

4.1. Attacks against Perfomance Honesty

In the following we consider attack scenarios against performance honesty in μSTR-H. Members trying to cheat on performance ratios of their devices are called *lying members*.

Individual Attacks: The adversary M_i pretends that its device has a lower μ_i than it really does in order to get a better position in P and save own costs during the protocol. Note that pretending to have a higher value for μ_i makes no sense, since it comes at adversary's expense. Another thinkable variation is that M_i has several mobile devices with different performance ratios. M_i's goal is to use a higher-performance device for the protocol computations but obtain a better position in P by submitting the performance ratio of its lower-performance device.

Collusion Attacks: Collusion may consist of different lying members so that at least one of them is able to obtain an advantageous position in P. It is also imaginable that some members of the collusion submit correct performance ratio parameters whereas other members cheat. In case of additive dynamic events (e.g., join) current members may collude in order to place new members on disadvantageous positions in P.

We require from μSTR-H to be resistant against all kinds of individual and collusion attacks.

5. Trusted Components

In this section, we propose an abstract system model based on TCG's (*Trusted Computing Group*) specifications ([9], [8]) that provide a practical solution to the missing link between the properties offered by a device platform and its configuration. Note that our system model uses trusted computing functionalities as black box.

Figure 3 illustrates this abstract model for a device M.

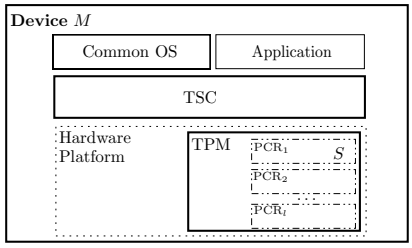

Figure 3. Abstract Model of M

The Trusted Computing Base (TCB) of a device consists of a Trusted Platform Module (TPM) and Trusted Software Component (TSC) described in the following.

Trusted Platform Module (TPM): The TPM provides a secure random number generator, non-volatile tamper-resistant storage, key generation algorithms, several cryptographic functions, amongst others a cryptographic hash function. A set of *Platform Configuration Registers* (PCR) are used in TPM to store hash values. The value of a PCR can only be modified as follows: $PCR_{i+1} \leftarrow \mathsf{Hash}(PCR_i|I)$, with the old value PCR_i, the new value PCR_{i+1}, and the input I (e.g., a SHA-1 hash value). This process is called *extending* a PCR. The main task of the TPM is to bind relevant secrets used in the protocol to the properties of the device. For this purpose, the following set of the TCG security mechanisms can be applied.

Integrity Measurement and Platform Configuration: Integrity measurement is done during the boot process by computing a hash value of the initial platform state. For this purpose TPM computes a hash of (measures) the code and parameters of the BIOS and extends the first PCR register by this result. A chain of trust can be established if an enhanced BIOS and bootloader also measure the code they are transferring control to as next, e.g., to the operating system (OS). The security of the chain relies strongly on explicit security assumptions about the TPM. Thus, the PCR values $PCR_0, \ldots, PCR_l$ provide evidence of the system's state after boot. We call this state the platform's *configuration*, denoted by $S := (PCR_0, \ldots, PCR_l)$.

Attestation: The TCG attestation protocol is used to give assurance about the platform configuration S_i to a remote party. To guarantee integrity and freshness, this value and a fresh nonce provided by the remote party are digitally signed with an asymmetric key pair (SK_{AIK}, PK_{AIK}) called *Attestation Identity Key* (AIK) that is under the sole control of the TPM.

Sealing: Data D can be cryptographically bound to a certain platform configuration S_i using an asymmetric key pair (SK, PK) generated by the TPM. By sealing D under S we mean the encryption of D using PK and S. The unseal command releases the decrypted data D only if for the current configuration S' holds $S' = S$.

Trusted Software Component (TSC): The TSC includes elementary properties of a security kernel such as protecting its memory from being accessed and manipulated by unauthorized processes/programs and isolation of processes (e.g., common OS is isolated from the application). Since TSC is a software component it can be used to process operations that are too expensive for the TPM.

In our system model the operating system (OS) of the device and the application software such as μSTR-H protocol run on top of the TSC. Note that OS and the application software are not trusted components.

6. TCG Meets μSTR-H

6.1. A Concrete System Model

Based on the abstract model in Section 5 we propose a concrete model of a device in the context of the μSTR-H protocols in Figure 4. Each device M_i is the host of a

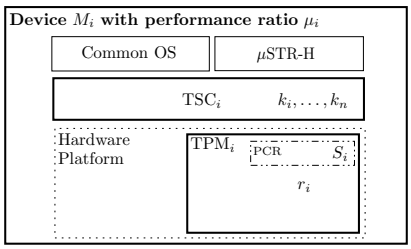

Figure 4. A Concrete System Model of M_i

TPM_i. TPM_i is supposed to bind the secret session random r_i to the performance ratio μ_i using the sealing function. Hence, the host can only access r_i if μ_i is valid, i.e., if M_i generates r_i using one device he cannot use r_i for the

computations on another device. The computation of secret keys k_j, $i \leq j \leq n$ and corresponding public keys K_j, $i \leq j \leq n-1$ are performed by TSC_i. However, TSC_i should not be able to perform certain security critical actions without involving the TPM_i, or simply break the underlying security protocol by leaking the secret values k_i. Since TSC_i is a part of TCB, its measurement S_i (e.g., a hash of TSC_i's code which is computed at the boot time) will be included in the system configuration stored in PCRs, and hence the sealing functionality can be used to bind the secrets also to S_i. Since TCB represents a "constant" part of the device, any attempt to manipulate it by patches changes PCR values making unsealing impossible.

Note that the trust assumption on TSC_i concerns only software attacks and not attacks that attempt to read out the memory content by using hardware devices. The untrusted component of μSTR-H which runs on top of TSC_i and is isolated from other processes is responsible for the communication with other devices and performs operations which do not explicitly require the knowledge of device's secrets, e.g., verification of received signatures.

6.2. Modified Protocols

Every TPM_i chooses $r_i \in_{\mathcal{R}} \{1, \ldots, t-1\}$ and computes $R_i = r_i G$. To enforce performance honesty TPM_i seals r_i under μ_i and the measurement S_i of TSC_i. TPM_i computes also the signature $\sigma_i \leftarrow \mathsf{Sign}_{SK_{AIK_i}}(R_i, \mu_i)$ testifying that R_i is generated in a way that it is bound to μ_i and S_i. Signatures of TPM_i can be verified $\mathsf{Verify}_{PK_{AIK_i}}(R_i, \mu_i, \sigma_i)$ using its certificate $\mathit{Cert}_{\mathrm{TPM}_i}$ that confirms the validity of TPM_i and the validity of its AIK, and is attached to the message. Note that $\mathit{Cert}_{\mathrm{TPM}_i}$ can be used instead of public key certificates of users, i.e., device authentication instead of member authentication. r_i remains inside TPM_i, and is used (unsealed) by TPM_i only if μ_i and S_i are correct. The user M_i (untrusted μSTR-H component on top of TSC_i) gains access to R_i, μ_i and σ_i.

Given K_{i-1} TSC_i computes $k_i = map(r_i K_{i-1})$ and $(k_{i+1}, \ldots, k_n)$ and $(K_i, \ldots, K_{n-1})$ after it obtains the value $r_i K_{i-1}$ from TPM_i. M_i gains access to the public keys while the secret keys (including k_n) remain in the memory of TSC_i. This is an important observation for the performance honesty. For simplicity we omit explicit notation of the interaction between M_i, TPM_i, and TSC_i according to Figure 4 in the following description of our protocols assuming that required data is transmitted over defined interfaces. Further, we assume that it is possible to obtain μ_i, σ_i and $\mathit{Cert}_{\mathrm{TPM}_i}$ using i and P.

If in μSTR-H a sponsor M_s has to change r_s then TPM_s chooses a new value for r_s, computes R_s, seals r_s under the configuration (μ_s, S_s) and generates a signature on R_s, μ_s using AIK.

μSTR-H Setup: Let $M_1, \ldots, M_n$ be participants wishing to initialize a mobile ad-hoc group.

- TPM_i selects r_i, computes R_i and σ_i. M_i broadcasts $(R_i, \mu_i, \sigma_i, \mathit{Cert}_{\mathrm{TPM}_i})$.
- M_i verifies σ_j for all $1 \leq j \leq n$, computes P, and stores $\mathbf{R}_i$.

  Additionally: TSC_1 computes $\mathbf{k}_1 = (k_2,\ldots,k_n)$, $(K_2,\ldots,K_{n-1})$. M_1 broadcasts $(K_2,\ldots,K_{n-1})$.
- For all $i > 1$: M_i stores K_{i-1}. TSC_i computes $\mathbf{k}_i = (k_i,\ldots,k_n)$.

Note that after the completion of the protocol the performance ratio order P contains only devices with verifiable TPM's signatures.

μSTR-H Join: A new member M_j is inserted in P according to its performance ratio μ_j. The new group key is denoted k_{n+1}.

- New member's TPM_j selects r_j, computes R_j and σ_j. M_j broadcasts $(R_j, \mu_j, \sigma_j, \mathit{Cert}_{\mathrm{TPM}_j})$.
- After successful verification of σ_j every M_i inserts M_j in P, renumbers all M_i $(i > j)$ to M_{i+1}, and adds R_j to $\mathbf{R}_i$. (note if verification of σ_j fails then the protocol stops).
  Additionally: TPM_s selects new r_s, computes R_s and σ_s. TSC_s recomputes $\mathbf{k}_s = (k_s,\ldots,k_{n+1})$ and $(K_s,\ldots,K_n)$. M_s broadcasts $(P, \sigma_s, R_s, (R_{j+1}, \ldots, R_{n+1}), (K_s,\ldots,K_n), \mathit{Cert}_{\mathrm{TPM}_s})$.
- M_j after successful verification of σ_i for all $1 \leq i \leq n+1$ stores P, finds own index j, stores K_s and $\mathbf{R}_j = (R_j,\ldots,R_{n+1})$. TSC_j computes $\mathbf{k}_j = (k_j,\ldots,k_{n+1})$.
  For all $i < s$: After successful verification of σ_s M_i updates R_s in $\mathbf{R}_i$. TSC_i recomputes $(k_s,\ldots,k_{n+1})$ in $\mathbf{k}_i$.
  For all $i > s$: M_i updates K_{i-1}. TSC_i recomputes $(k_i,\ldots,k_{n+1})$.

μSTR-H Leave: A leaving member M_d is removed from P. The new group key is denoted k_{n-1}.

- M_i deletes M_d from P, if $i < d$ also R_d from $\mathbf{R}_i$, and k_d from $\mathbf{k}_i$, and renumbers all M_i $(i > d)$ to M_{i-1}.
  Additionally: TPM_s selects new r_s, computes R_s and σ_s. TSC_s recomputes $\mathbf{k}_s = (k_s,\ldots,k_{n-1})$ and $(K_s,\ldots,K_{n-2})$. M_s broadcasts $(R_s, \sigma_s, \mathit{Cert}_{\mathrm{TPM}_s}, (K_s,\ldots,K_{n-2})$.
- For all $i < s$: M_i after successful verification of σ_s updates R_s in $\mathbf{R}_i$. TSC_i recomputes $(k_s,\ldots,k_{n-1})$ in $\mathbf{k}_i$.
  For all $i > s$: M_i updates K_{i-1}. TSC_i recomputes $\mathbf{k}_i = (k_i,\ldots,k_{n-1})$.

μSTR-H Merge: Two groups, G′ of size n' and G″ of size n'', are merging to a common group G. The resulting P is computed by merging P' and P''. The new group key is denoted $k_{n'+n''}$.

- M'_1 and M''_1 broadcast $(P', (R'_1, \ldots, R'_{n'}))$ and $(P'', (R''_1, \ldots, R''_{n''}))$, respectively.
- Every member M_i verifies signatures σ_j for all $1 \leq j \leq n' + n''$, merges P' and P'' to P, and stores $\mathbf{R}_i = (R_i,\ldots,R_{n'+n''})$.
  Additionally: TPM_s selects new r_s, computes R_s and σ_s. TSC_s recomputes $\mathbf{k}_s = (k_s,\ldots,k_{n'+n''})$ and $(K_s,\ldots,K_{n'+n''-1})$. M_s broadcasts $(R_s, \sigma_s, (K_s,\ldots,K_{n'+n''-1}))$.
- For all $i < s$: M_i after successful verification of σ_s updates R_s in $\mathbf{R}_i$. TSC_i recomputes $(k_s,\ldots,k_{n'+n''})$ in $\mathbf{k}_i$.
  For all $i > s$: M_i updates K_{i-1}. TSC_i recomputes $\mathbf{k}_i = (k_i,\ldots,k_{n'+n''})$.

μSTR-H Partition: A subgroup G′ of size v leaves group G of size n. The new group key is denoted k_{n-v}.

- M_i deletes all M'_j from P, if $i < j$ also R'_j from $\mathbf{R}_i$ and k'_j from $\mathbf{k}_i$, and renumbers all survived members M_i accordingly. Additionally: TPM$_s$ selects new r_s, computes R_s and σ_s. TSC$_s$ recomputes $\mathbf{k}_s = (k_s,\ldots,k_{n-v})$ and $(K_s,\ldots,K_{n-v-1})$. M_s broadcasts $(R_s,(K_s,\ldots,K_{n-v-1}))$.
- For all $i < s$: M_i after successful verification of σ_s updates R_s in $\mathbf{R}_i$. TSC$_i$ recomputes $(k_s,\ldots,k_{n-v})$ in $\mathbf{k}_i$.
  For all $i > s$: M_i updates K_{i-1}. TSC$_i$ recomputes $\mathbf{k}_i = (k_i,\ldots,k_{n-v})$.

6.3. Security Analysis

In this section we informally analyse the performance honesty of our modifications to μSTR-H. Note that other security issues of μSTR-H are still valid since our modifications concern only internal computations within the device.

In the modified protocols every M_i is able to compute P and check own position in P by verifying the TPMs' signatures σ_j, $1 \leq j \leq n$ of other devices. σ_i ensures that R_i has been generated within the device those performance ratio is μ_i. Since TPM$_i$ is trusted r_i never leaves its memory. The secret keys $\mathbf{k}_i = (k_i, \ldots, k_n)$ are computed by TSC$_i$. Since $k_i = map(r_i K_{i-1})$ and TSC$_i$ does not know r_i TPM$_i$ must be involved in the computation. Since TSC$_i$ is trusted secret keys $\mathbf{k}_i = (k_i, \ldots, k_n)$ never leave its memory. Also, r_i is bound to TSC$_i$, i.e., any adversarial modification of TSC$_i$ would make unsealing impossible. Hence, M_i does not get any direct access to the group key k_n or to any other secrets that needed to compute it. Therefore, M_i must use the same device throughout the protocol and ask TSC$_i$ to perform application specific operations using k_n. Since the signature scheme is assumed to be secure M_i cannot forge σ_i and cheat on μ_i. Thus, modifed μSTR-H provides performance-honesty. For completeness, we consider the security against individual and collusion attacks from Section 4.1.

Security against individual attacks: User M_i cannot lie on its performance ratio μ_i because μ_i and R_i are signed by TPM$_i$. Since the signature scheme is secure M_i cannot forge σ_i for another value of μ_i. If M_i submits a signature that cannot be verified by other users then M_i's contribution R_i is dropped and M_i cannot compute the group key. If M_i replays a valid signature σ_i then it is assigned to a position in P, but cannot compute the group key because it does not know r_i. Similarly, if M_i submits σ_i of its lower-performance device but wishes to use its higher-performance device to compute the group key it fails because r_i remains inside TPM$_i$ of the lower-performance device.

Security against collusion attacks: Consider a collusion $\mathcal{C} = \{M_{i_1}, \ldots, M_{i_k}\}$. Assume that all members in $\mathcal{C}$ lie on their performance ratios. Since they cannot forge TPMs' signatures there should be at least one honest member $M_j \notin \mathcal{C}$. M_j is able to identify all lying members, because M_j verifies the order of P by proving the TPMs' signatures. Assume that all members in $\mathcal{C}$ submit correct performance ratios. Let $\mathcal{H} \subset \mathcal{C}$ be a set of collusion members who wish to save own costs and obtain the group key from members in $\mathcal{C} - \mathcal{H}$. As described above r_i remains inside TPM$_i$ and all $\mathbf{k}_i = (k_i, \ldots, k_n)$ in the memory of TSC$_i$. Therefore, even if members in $\mathcal{C} - \mathcal{H}$ compute k_n they do not get any direct access to it or to any other secret value (e.g., r_i or $\mathbf{k}_i$). Hence, they cannot provide members in $\mathcal{H}$ with the required secret information. Assume that during the communication session all group members collude in order to assign joining members to disadvantageous positions in P. Every new member checks own position in the updated P by verifying the TPMs' signatures of all devices, and is, therefore, able to identify all lying members.

7. Conclusion

In this paper we have considered the deployment of trusted computing technology to control the access of mobile devices to the input/output of a security protocol depending on the devices' properties. In the context of secure group communication (μSTR-H protocol) we have designed a system model for mobile devices to achieve this control considering the performance of every involved mobile device as a property.

References

[1] S. Basagni, M. Conti, S. Giordano, and I. Stojmenović, editors. *Mobile Ad Hoc Networking*. Wiley-IEEE Press, 2004.

[2] L. Buttyan and J.-P. Hubaux. Enforcing service availability in mobile ad-hoc wans. In *MobiHoc '00: Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing*, pages 87–96, Piscataway, NJ, USA, 2000. IEEE Press.

[3] M. Jakobsson, J.-P. Hubaux, and L. Buttyan. A micropayment scheme encouraging collaboration in multi-hop cellular networks. In *Financial Cryptography 2003*, volume 2742 of *Lecture Notes in Computer Science*, pages 15–33. Berlin: Springer-Verlag, 2003.

[4] Y. Kim, A. Perrig, and G. Tsudik. Communication-efficient group key agreement. In *Information Systems Security, Proc. of the 17th International Information Security Conference, IFIP SEC'01*, 2001.

[5] J. Luo, P. T. Eugster, and J.-P. Hubaux. Route driven gossip: Probabilistic reliable multicast in ad hoc networks. In *INFOCOM*, 2003.

[6] M. Manulis. Key Agreement for Heterogeneous Mobile Ad-Hoc Groups. In *Proceedings of 11th International Conference on Parallel and Distributed Systems Volume 2 International Workshop on Security in Networks and Distributed Systems*, pages 290–294. IEEE Computer Society, 2005.

[7] P. Michiardi and R. Molva. A game theoretical approach to evaluate cooperation enforcement mechanisms in mobile ad hoc networks. In *Proceedings of Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt'03)*, 2003.

[8] Trusted Computing Group (TCG). TPM main specification, Version 1.2, `http://www.trustedcomputinggroup.org`, November 2003.

[9] Trusted Computing Platform Alliance (TCPA). TCPA main specification, Version 1.1.b, February 2002.