Shifting inference control to user side: Architecture and protocol

Yanjiang YANG

Yingjiu LI
Singapore Management University, yjli@smu.edu.sg

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

Feng BAO

Shifting Inference Control to User Side: Architecture and Protocol

Yanjiang Yang, *Member*, *IEEE*, Yingjiu Li, *Member*, *IEEE*, Robert H. Deng, and Feng Bao

Abstract—Inference has been a longstanding issue in database security, and inference control, aiming to curb inference, provides an extra line of defense to the confidentiality of databases by complementing access control. However, in traditional inference control architecture, database server is a crucial bottleneck, as it enforces highly computation-intensive auditing for all users who query the protected database. As a result, most auditing methods, though rigorously studied, are not practical for protecting large-scale real-world database systems. In this paper, we shift this paradigm by proposing a new inference control architecture, entrusting inference control to each user's platform that is equipped with trusted computing technology. The trusted computing technology is designed to attest the state of a user's platform to the database server, so as to assure the server that inference control could be enforced as prescribed. A generic protocol is proposed to formalize the interactions between the user's platform and database server. The authentication property of the protocol is formally proven. Since inference control is enforced in a distributed manner, our solution avoids the bottleneck in the traditional architecture, thus can potentially support a large number of users making queries.

Index Terms—Inference control, trusted computing, database, auditing, security protocol.

1 Introduction

The inference problem has been a longstanding issue in database security that was first studied in statistical databases [3], [16] and then extended to multilevel databases and general-purpose databases [24]. The inference problem can be referred to as the concern that one can infer (sensitive) information beyond one's privileges from the unsensitive data to which one is granted access. The set of unsensitive data leading to reference is said to constitute an *inference channel*. The inference problem cannot be solved by traditional access control (AC), as the disclosure of sensitive information does not result from unauthorized accesses but from authorized ones. The existence of various inference channels is due to the inevitable interconnections between sensitive data that are protected from and nonsensitive data that are granted to users' accesses.

For better understanding, Table 1 shows a simple example that helps illustrate the inference problem. The EMPLOYEE table contains AGE and SALARY information for the employees in a company. To protect individuals' salary information, the following access rule is enforced: *the database server can answer queries about sums of salaries over multiple employees but rejects any query on a single employee's salary*. With this AC policy enforced, however, employee C's salary can still be easily derived from the following legitimate queries Q_1, Q_2, Q_3, and Q_4, provided that the reply to Q_3 is 1 and that to Q_4 is 0 (i.e., the four queries form an inference channel):

- Q_1: select sum(SALARY) from EMPLOYEE, where AGE $\geq$ 30;
- Q_2: select sum(SALARY) from EMPLOYEE, where AGE $\geq$ 32;
- Q_3: select count(NAME) from EMPLOYEE, where AGE $=$ 30; and
- Q_4: select count(NAME) from EMPLOYEE, where AGE $=$ 31.

1.1 Related Work

The existence of inference channels poses a serious threat to the confidentiality of database systems. Extensive research has been conducted on *inference control (IC)* to mitigate the threat. IC techniques that aim to remove inference channels can be classified into four general categories [3]: conceptual approach, data perturbation, output perturbation, and query restriction. There is also effort considering IC in a specific context [8], [43], [44], or exploring a particular aspect of IC [64]. The *conceptual approach* includes two well-known models, the conceptual model [11] and the lattice model [18], [20]. The former provides a general framework dealing with security from the development of the conceptual schema to implementation; the latter presents a framework describing statistical database information in tabular form at different levels of aggregation. The conceptual approach has turned out to be quite useful for further study and understanding of the inference problem, but the proposed frameworks are sometimes too general for practical implementation.

The *data perturbation approach* (e.g., [27], [33], [36], [42], [61], [65], and [66]) typically replicates the original database and generates a perturbed database with noises for users to access. This approach includes two subcategories: probability distribution method and fixed data perturbation. In the

- *Y. Yang is with the Institute for Infocomm Research, A*STAR, 1 Fusionopolis Way, #21-01 Connexis, South Tower, Singapore 138632. E-mail: yyang@i2r.a-star.edu.sg.*
- *Y. Li and R.H. Deng are with the School of Information Systems, Singapore Management University, 80 Stamford Road, Singapore 178902. E-mail: {yjli, robertdeng}@smu.edu.sg.*
- *F. Bao is with the Institute for Infocomm Research, 1 Fusionopolis Way, #19-01 Connexis, South Tower, Singapore 138632. E-mail: baofeng@i2r.a-star.edu.sg.*

TABLE 1
Employee Table

NAME	AGE	SALARY
A	24	2800
B	26	3100
C	30	3200
D	32	3600
E	35	3000
F	36	3200

former, the perturbed database is a sample drawn from the same distribution as the original database; in the latter, the perturbed database is generated by adding independent noises to each entry of the original database. By and large, the data perturbation approach suffers from a severe bias problem due to the noises that are added into data; therefore, it is not suitable for dynamic databases. In contrast, the *output perturbation approach* (e.g., [1], [5], [17], and [25]) does not add noises but performs certain manipulations over database queries such as rounding the query replies up or down. Though the output-perturbation-based approach is immune to the bias problem, it may suffer from having null query sets, in which case useful information is disclosed.

The last category is the *query restriction approach*, which can be further classified into five subcategories: query-set-size control (e.g., [19] and [50]), query-set-overlap control (e.g., [21]), partitioning (e.g., [10], [51], and [70]), cell suppression (e.g., [13], [40], and [47]), and auditing (e.g., [9], [12], and [28]). A comparison of these methods is given in [3] from various aspects including degree of security, query processing overhead, and suitability for dynamic databases.

Among the query restriction-based approach, auditing is an important IC method, having many desirable properties. To enforce auditing, the database server keeps a log of all users' queries; when replying to a user query, it checks for possible inference channels against the current query as well as the past queries asked by the same user. Since the control decision is made based upon a user's whole access history, auditing has the potential to achieve better security. Furthermore, auditing provides users with unperturbed query results as long as no inference channel is detected. Due to these features, auditing has triggered intensive research in database security from the 1970s [28], [49] through the 1980s [6], [7], [9], [12] and 1990s [14], [38], [67] to the 21st century [31], [34], [35], [37], [62], [63], [68], [71].

Unfortunately, auditing faces enormous difficulty in practical deployment, mainly due to the excessive computational overhead it requires to check for inference channels from the accumulated query log. Audit Expert[1] [12] is a typical example. It was shown that it takes Audit Expert $\mathcal{O}(n^2)$ time to process a new SUM query [12], where n is the number of database entities or records, and $\mathcal{O}(mn^2)$ time to process a set of m queries. While this workload could be improved to some extent in certain specific situations (e.g., for range queries [9]), the complexity is impractically high in the general case for auditing large databases.

1. Audit Expert is a classic auditing method. It maintains a binary matrix whose columns represent specific linear combinations of database entities (records) and whose rows represent user queries that have already been answered. Audit Expert transforms the matrix by elementary row operations to a standard form and concludes that exact inference exists if at least one row contains all zeros except in one column.

The obvious drawback of auditing due to high computational complexity is low system scalability. The database server that enforces auditing can only afford a small number of users asking queries simultaneously. For this reason, auditing has not been deemed to be a practical IC method for real-world database systems [3].

1.2 Our Contributions

To resolve the impracticality problem, we propose a new architecture for IC (especially auditing) with trusted computing, shifting IC from the server side to the user side. More specifically, the new architecture works by entrusting the enforcement of IC to individual users' computer platforms, in contrast to the traditional architecture in which IC is enforced by the database server. In this new architecture, the database server is still responsible for the enforcement of AC, but each user's platform is empowered to handle IC over the queries issued by its own users. Since the computation-intensive task of auditing is amortized to all users, the database server is no longer a bottleneck. As a result, our architecture can potentially be used for protecting large-scale database systems.

A crucial issue associated with this shift of paradigm is to ensure that IC is enforced at the user side exactly as prescribed by the database server without any interference or manipulation. This requires that each user's platform is in a trusted state when IC is enforced. A typical solution to attain such a trusted state is to equip each user's machine with a TCG-compliant trusted platform module (TPM) [55], [60] that establishes a hardware-based chain of trust from booting to OS loading to application execution. In our architecture, TPM is used to protect the execution environment of IC so as to attest the trusted state of a user's platform to the remote database server, convincing the server that the enforcement of IC will not deviate. We propose a generic protocol to formalize the interactions between the user's platform and the database server. Any existing IC technique can work with our protocol in the new architecture, and the authentication property of our protocol can be formally proven.

1.3 Organization

The rest of this paper is organized as follows: In Section 2, we propose a new architecture for shifting the IC paradigm. In Section 3, we present a protocol to enable IC to be executed on the users' side using standard TPM commands, followed by the security analysis in Section 4. In Section 5, we further discuss some extensions to our solution. Finally, we conclude this paper in Section 6.

2 Architecture

Traditional architecture. The traditional architecture for IC is illustrated in Fig. 1a, where both AC and IC are enforced at the database server side. In this architecture, the AC module (ACM) implements AC functionality, while the IC module (ICM) executes a designated IC algorithm (e.g., Audit Expert) and acts as an extra line of defense in protecting the database. Upon receiving a new query from a user, ACM first decides

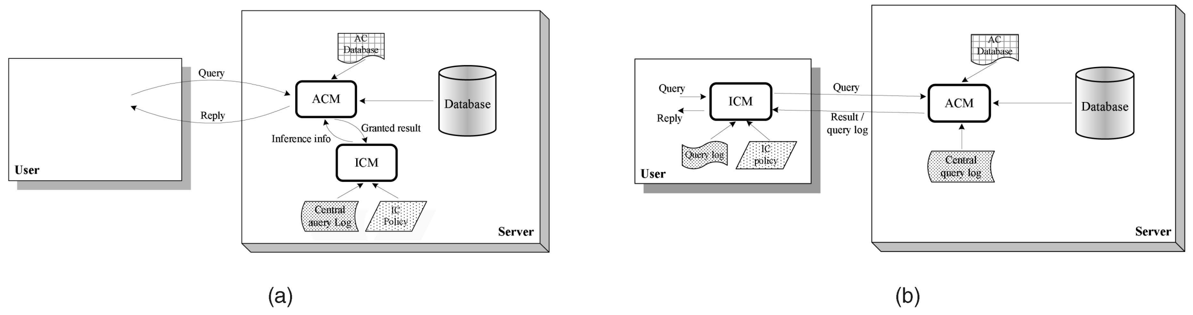

Fig. 1. IC architectures. (a) Traditional IC architecture. (b) New IC architecture.

whether the user is a legitimate user with respect to the queried data. This can be done by checking an AC database, which contains AC rules and policy. If the user is a legitimate user, the database server further checks with ICM to determine whether the query will lead to any inference. To achieve this, ACM passes the query result granted to the user to ICM, who assesses the query against the IC policy as well as the user's past queries (collected in the central query log that accumulates all users' queries) by executing the designated IC algorithm. ACM bases the reply to the user's query on both the AC decision made by ACM and the IC decision returned by ICM: the query result is returned to the user only if ACM decides that the user has the proper access right and ICM evaluates that no inference would occur under the IC policy; otherwise, ACM rejects the user's query.

New architecture. Since the enforcement of IC is computationally intensive, it may bottleneck the database server in the traditional architecture. To solve this problem, we propose a new architecture, shown in Fig. 1b, to enforce IC in a distributed manner. The basic idea is to offload the IC function to individual users. More specifically, ICM resides at the user side instead of on the server side. ICM maintains a *local* query log at the user's host by accumulating the queries issued by the user. To query the database, the user contacts ICM by issuing a query, then ICM checks with ACM at the server side to see whether the user has the right to access the data. ACM checks the user's request against the access rules and policy. If the user is granted access, ACM returns the query response, together with the IC policy[2] and the user's latest past queries issued from other hosts, to ICM. Then, ICM executes the IC algorithm by checking the query against the IC policy based on all the user's past queries, which comprise the local query log as well as the queries just passed from the server. ICM releases the query response to the user only if the query would lead to no inference under the IC policy. In this architecture, ICM on the user side acts as an extension of the database server in IC.

In the process of IC, it is critical to support a user using multiple hosts to query the database. This can be readily achieved in the traditional architecture, because a user's queries, regardless of being issued from which host, are collected in the central query log maintained by the server. However, in the new architecture, queries that are maintained at different hosts may not be readily available to the current host where IC is enforced. Our solution is that the database server still maintains a central query log as in the traditional architecture, but it collects user queries at the discretion of user identity in conjunction with host identity. When a user issues a query from a particular host, the queries previously issued by the user from all other hosts are passed by ACM to the current host for the enforcement of IC. For better communication efficiency, the host updates its local query log by including all the queries it receives from the server. As such, the server only needs to send the *latest* past queries from other hosts, i.e., the queries issued at other hosts by the user since when the last query was issued at the current host. For convenience, these queries are referred to as *complementary queries* in the sequel. It should be clear that our new architecture dispenses fatally expensive IC functionality to individual users at the price of minimum overheads of local storage (to store local query log) and network communication (to transfer complementary queries and attestation information), which are expected to be tolerable without bottlenecking the current system of broadband networks and powerful user computers. The communication overhead of sending complementary queries is not incurred if the user chooses to use a single host.

Example of IC policy. It should be noted that in either the traditional architectures or the new architecture, IC policy is an essential component in the enforcement of IC. It stipulates what is necessary for the execution of the IC algorithm, e.g., the attributes to be protected (e.g., Boolean, integer, or real), protection objectives (e.g., partial disclosure control or complete disclosure control), constraints (e.g., database dependencies, integrity constraints, and domain knowledge), and so on. For illustration purposes, an example of IC policy for Audit Expert specified in XML is given in Fig. 2. While a more formal language (e.g., XACML [69]) can be defined, we are concerned with demonstrating how an

```
<?xml version="1.0" encoding="UTF-8"?>
<policy ID='urn.www.example.employee'>
    <owner HREF='http://www.authentication.com'
        signKey='2ABG64897HJ' signAlg='RSA' signValue='♯◇†||┌‡⅃♮♭ℓ'/>
    <subject> Senior User </subject>
    <object>
        <attributeName> Salary </attributeType>
        <attributeType> Numerical </attributeType>
    </object>
    <queryType> Ad Hoc Query </queryType>
    <auditType> Exact Audit </auditType>
</policy>
```

Fig. 2. A sample policy for Audit Expert.

2. The IC policy needs to be delivered to the user only when it has been modified by the server recently; otherwise, it can be kept at the user's platform safely (protected by TPM).

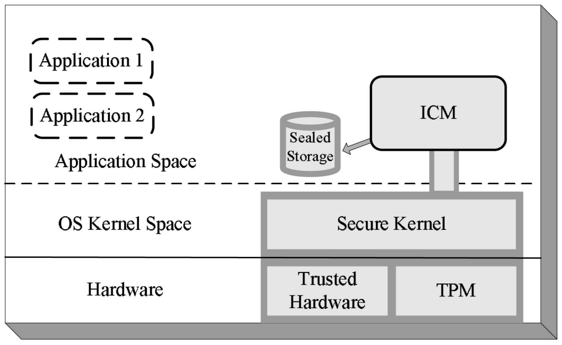

Fig. 3. TPM-enabled user host.

IC policy regulates the enforcement of IC in our proposed new architecture. The IC policy defines the *subject* who accesses the database, the *object* that consists of sensitive attribute(s) to be protected, the *type of queries* imposed by the subject, and the *type of audit* to be enforced. In Audit Expert, there are two types of queries: range (sum) queries and ad hoc (sum) queries, and two types of audit: exact audit [9], [12] and interval audit [35]. For our example, a subject is defined to be "senior user," "junior user," or "suspicious user," and the object is defined to be the *SALARY* attribute in the *EMPLOYEE* table, which is shown in Table 1. The policy states that a senior user can launch any ad hoc query and obtain the query result as long as no exact value of the salary attribute can be inferred from his cumulative queries.

Role of trusted computing. A challenging issue in our new architecture is that the database server may lose its control over ICM and that the user may compromise ICM so as to bypass IC. To address this issue, assurance must be given to the database server that ICM is executed as expected, free of user's interference and manipulation. This kind of assurance is achieved by virtue of trusted computing. In Fig. 3, a user's machine is equipped with a TCG-compliant TPM [60] and possibly other trusted hardware. A trusted platform can be built from TPM at the hardware layer to a secure kernel in the OS kernel space and to ICM in the application space.

The hardware, underpinning and cooperating with the secure kernel, provides necessary security functions to ICM, from basic cryptographic functions to sealed storage, platform attestation, and a protected running environment. TPM can assure the remote database server of the integrity of the components in the platform, including the secure kernel and ICM, through its integrity measuring, storing, and reporting mechanisms. In particular, the running state of the protected platform can be conveyed to the database server by virtue of the platform attestation mechanism of TPM, so that the server can decide whether the protected platform runs in a sufficiently trusted state. The protected platform running in a trusted state ensures that ICM enforces IC as expected. This platform architecture can be considered as an open system in the sense that the host accommodates both protected applications and unprotected applications.

The involvement of TPM in IC can be considered yet another application of trusted computing [39], [55]. Other trusted computing applications that have been rigorously studied in recent years include digital rights management [23], remote AC enforcement [45], [48], server-side user privacy protection [30], server privacy protection [56], secure auction [41], and integrity measurement [46], to name a few. The objective of these applications is to either enable a server to extend its control over data distributed to client side or protect users' privacy on the server side, while our major concern is to securely decentralize the enforcement of IC so as to resolve the efficiency and scalability problems inherent in IC. In the sequel, we assume prior knowledge about TPM, and interested readers are referred to Appendix for a brief review of TPM.

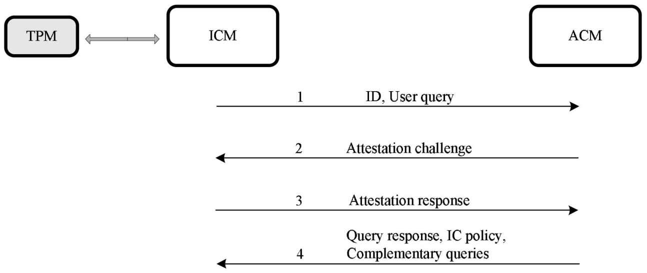

Fig. 4. Interactions between ACM and ICM.

Interactions between ACM and ICM. In our new architecture, ACM enforces the AC mechanism over an AC database, and it represents the database server by interacting with all users. On the other hand, ICM is responsible for enforcing IC according to the IC policy specified by the database server, and it also acts as an interface of the user side interacting with the database server. ICM is an application protected by TPM, which is inextricably bound to the user host.

The main interactions between ICM (having access to TPM) and ACM are illustrated in Fig. 4. It is assumed that the user has certain identification information (e.g., user password) to identify itself to ACM. ICM first sends the identification information (including user's identification information and host's identity) together with a user query to ACM (database server). Upon reception of the user query, ACM enforces AC and formulates a response to the query if the user query is authorized. Before ACM delivers the query's response, it first sends an attestation challenge to ICM. Based on the attestation response from ICM, ACM can decide whether the user's platform is in a trusted state. If so, it releases the query's response as well as the IC policy and the set of complementary queries to ICM for the enforcement of IC. A detailed protocol is given in Section 3 to formalize the interactions between ACM and ICM.

3 Protocol

We present a protocol for the interactions between ACM and ICM. The protocol enables ICM at the user side to enforce the IC prescribed by the database server. The designing of the protocol assumes the use of version 1.2 TPM command set [59] and data structure [58].

3.1 Overview

Shifting IC from the database server to users' hosts incurs new security threats that do not exist in the traditional architecture. We enumerate the new security threats and explain the basic ideas in our protocol design to mitigate these threats. The fundamental assumption is that the TPM is perfectly secure in the sense that the functions of TPM cannot be compromised. We note that there may exist some attacks that modify or crash the protected applications at user side after they have been attested by the server. Attacks of these kinds are not specific to our system but generic to all applications of trusted computing technology. A simple solution suggested by Sailer et al. [45] is that the server frequently challenges the user's platform so as to detect the attacks as soon as possible. Shi et al. [52] proposed fine-grained attestation, which attests only to the piece of code that is concerned, instead of the entire memory content. Further, it measures code immediately before it is executed and uses a sand-boxing mechanism to protect the execution of the attested code. This greatly mitigates the concerns on the discrepancy between the time of use and time of attestation.

Integrity of ICM. Since ICM resides in the user's host, a malicious user clearly has a motivation to alter the designated function of the protected platform, especially ICM, so as to bypass the IC. Since TPM is inextricably bound to the user's host, we can use its integrity measuring, storage, and reporting mechanisms to detect any compromise of the integrity of the user's platform, including ICM.

Integrity of query history. This includes integrity of the local query log and authenticity of the set of complementary queries sent by the server. The latter can be addressed in the same way as the IC policy (see below), and we here focus on the former. We know that accurate IC depends on the complete query history. However, as the local query log is maintained in the user's host, it easily suffers malicious alteration including complete erasion. To thwart this threat, a straightforward solution is to let ICM hold a secret key for either MACing or signing the query log, with the secret key stored in the sealed storage of TPM. However, this introduces an extra key for ICM to manage. We instead use a different yet novel method by associating the integrity digest of the query log with the key for protecting the confidentiality of query responses (see the details in *confidentiality of query responses*).

Authenticity of IC policy. The IC policy regulates how IC is enforced. While in transit or in storage, the IC policy is subject to malicious alteration. It is extremely important to ensure that the IC policy enforced by ICM in the user's host is indeed dispatched by the database server and has not been tampered with. This is achieved by assuming that ACM holds a digital signature key pair (pk_{ACM}, sk_{ACM}); before disseminating the IC policy to the user, ACM signs the IC policy so that ICM can check whether the policy has been compromised either in transit or in storage. This also enables ICM to verify the source of the IC policy. Apparently, the same can be applied to protect the authenticity of the complementary queries, which also need to be transmitted from the server to the user host.

Confidentiality of secrets maintained by ICM. In some cases, ICM needs to maintain some secret keys so as to protect the database server's data on the user side. To prevent malicious users from reading the secrets, ICM needs help from TPM to store these secrets in the sealed storage provided by TPM.

Confidentiality of query responses. Before ICM determines whether it is safe to release query responses, the user should be kept from reading the responses, whether they are in transit or in store on the user host. While in store, the query responses can be protected using secret keys maintained by ICM (as described above). To achieve the confidentiality of query responses in transit, a secure channel between ICM and ACM is established. More specifically, ICM asks TPM to generate an ephemeral asymmetric encryption key pair, where the public key is certified by TPM and the private key is stored in its protected storage. The public key can be used by ACM to encrypt the query responses, which will be sent to ICM. Upon receiving the encrypted message, ICM asks TPM for decryption operation in a secure software environment. Note that, in this solution, TPM acts as a certifying party; there is no need to resort to external certification mechanisms.

As stated earlier, we novelly integrate the integrity protection of the local query log into the confidentiality protection of query responses. This is explained as follows: When requesting that TPM perform the decryption operation, the invoking entity (i.e., ICM) is required to provide a piece of authorization data, which is normally derived from a password that is provided by the user who invokes ICM. In our solution, however, the piece of authorization data is derived by ICM not only from the user's password but also from a content digest of the local query log. As a result, if the integrity of the query log is compromised, the authorization data will be refuted by TPM so that the private key cannot be accessed for the decryption operation. We must point out that the content digest of the query log is not intended to enhance the secrecy of the authorization data, which depends totally on the strength of the user's password.

Protected execution environment. A protected execution environment is needed for the running of ICM; otherwise, the OS kernel or other applications running in parallel on the user host may access the code and data within the ICM application domain. Though a TPM-enabled platform can be configured as a restricted system (which only runs a small set of protected applications, i.e., ICM in our case) or an open system but with all applications being protected by TPM, neither of the systems is practical. While the impracticality of the restricted system is obvious, it is challenging for TPM to perform platform attestation in an open system. The reason is that the attestation would involve a large set of application integrity metrics and that the database server must know in advance all the applications that run on each user's platform.

A more practical solution is that the user host remains open, but it is partitioned into a protected domain and an unprotected domain. The protected domain consists of a restricted set of protected applications such as ICM, while the unprotected domain includes other application softwares that do not need to be protected. Although the current TPM functionalities specified by the Trusted Computing Group (TCG) do not suffice to support this solution, more efforts have been made to establish the protected environment as desired. For example, the Intel's LaGrande Technology (LT) [32] incorporates an additional set of hardware and software components around the TCG-compliant TPM, enabling software domain separation and protection; as such, a process without permission is not able to access another process's memory space, keystroke information, and display information. This provides a protected execution environment that is sufficient for our solution. Without further complicating our presentation, it is reasonable to

TABLE 2
Notations

Notation	*Description*
$E_{pk}(.)$, $D_{sk}(.)$	public key encryption with public key pk, and decryption with private key sk
$enc(k,.)$, $dec(k,.)$	symmetric key encryption and decryption with secret key k, respectively
$S_{sk}(.)$, $V_{pk}(.)$	digital signature signing with private key sk, and verification with public key pk. Signature is assumed to be message aware, i.e., from a message the message signed can be derived
$SHA1(.)$	SHA-1 hash function
$A \rightarrow B : m$	entity A sending message m to entity B

assume in our protocol that TPM (possibly together with other trusted hardware) enables ICM to run in isolation, free from interferences from other applications running in parallel. Moreover, the application data that ICM uses in its execution domain will be automatically erased as long as ICM exits its execution.

3.2 Steps

We present our protocol in five steps, where the first four steps correspond to the four stages of interactions shown in Fig. 4, and the last step represents the enforcement of IC locally by ICM. Main notations used in the following presentation are listed in Table 2.

Step 1. ICM $\rightarrow$ ACM: id_U, id_H, q.
To issue query q, the user invokes ICM to send user identification information id_U, the host identity id_H, together with q to ACM on the database server side. Without specifying the composition of the identification information, we simply assume that id_U suffices to enable ACM to identify the user and enforce AC.

Step 2. Upon reception of a query request from the user, ACM checks whether the user has the requested right to access the data in the query. If so, ACM challenges the user's platform for remote attestation. This step consists of three substeps.

Step 2.1. ACM: $identify(id_U)$.

ACM identifies the user by executing $identify(id_U)$, the deployed identification function.

Step 2.2. ACM: $ac(id_U)$.

ACM executes the AC algorithm $ac(id_U)$ to determine whether the user has the permission to access the data in the query. If the user is not authorized, ACM aborts the protocol; otherwise, it continues with step 2.3.

Step 2.3. ACM $\rightarrow$ ICM: n_{ACM}.

ACM generates a random nonce n_{ACM} and sends it to ICM as the challenge for platform attestation. The nonce is used to thwart replay attacks in the following platform attestation.

Step 3. The platform attestation is performed in this step. Be aware that before the start of this step, ICM already has in possession a public key pk_{ICM} generated by TPM in the last query session. This will be clear shortly in steps 5.7 and 5.8.[3] The details of step 3 are given as follows:

3. In the case that the user queries the database server for the first time, there will be two extra substeps prior to step 3.1 that enable ICM to generate pk_{ICM}. The two extra substeps are the same as steps 5.7 and 5.8, with the only exception that the user's query log is empty at this point.

Step 3.1. ICM $\rightarrow$ TPM: TPM_CertifyKey.

ICM first invokes a standard TPM command TPM_CertifyKey for TPM to certify pk_{ICM}. The TPM_CertifyKey command instructs TPM to generate a signature on a public key using its attestation identity key (AIK). The operation of key certification can be bound to a specific state of the underlying platform. The input parameters of TPM_CertifyKey include the key to be certified, externally supplied data of 20 bytes, and the Platform Configuration Registers (PCRs), whose contents are bound to the certification operation. In our case, the externally supplied data are calculated as $SHA1(n_{ACM}, id_U, id_H)$, and the PCRs contain the integrity measurement metrics for the protected platform including the booting procedure, the OS, and ICM.

Step 3.2. TPM $\rightarrow$ ICM: TPM_Certify_Info, $\sigma_{TPM} = S_{sk_{TPM}}(SHA1(pk_{ICM}) \| SHA1(n_{ACM}, id_U, id_H) \| im)$.

In response, TPM outputs a TPM_Certify_Info data structure, as well as a signature signed on the public key pk_{ICM}, the nonce n_{ACM}, the identification information id_U, id_H associated with the query, and the integrity measurement metrics im of the platform. Here, TPM_Certify_Info contains information regarding the usage of the public key pk_{ICM}, the PCRs involved in signing, and a digest of the public key. Note that sk_{TPM} denotes the private AIK of TPM (accordingly, pk_{TPM} denotes the public AIK). For the sake of simplicity, an atomic quantity im is used to represent the integrity measurement metrics of the protected platform. It is interesting to note that the platform integrity reporting is achieved through certification of the public key in this step.

Step 3.3. ICM $\rightarrow$ ACM: TPM_Key, TPM_Certify_Info, σ_{TPM}, TPM.AIK credential.

In response to the attestation challenge, ICM sends TPM_Key, TPM_Certify_Info, σ_{TPM}, and the relevant TPM AIK credential to ACM on the server side. Here, TPM_Key is a data structure that is generated in the last query session; it contains the public key pk_{ICM} and other related information, as will be explained in step 5.8.

Step 4. ACM verifies the attestation response and sends a query response as well as the IC policy and the complementary queries to ICM for the enforcement of IC.

Step 4.1. ACM: $V_{pk_{TPM}}(\sigma_{TPM})$.

Upon reception of the attestation response, ACM first verifies the signature σ_{TPM} using public key pk_{TPM} and the corresponding certificate information.

Step 4.2. ACM: $validate(im)$.

Then, ACM verifies whether im (contained in TPM_Certify_Info) represents a trusted state of the user's platform as expected. In particular, it verifies whether ICM is running as expected. We use an atomic function $validate(\cdot)$ to denote this process.

Step 4.3. ACM $\rightarrow$ ICM: $\varepsilon_1 = E_{pk_{ICM}}(k_1 \| k_2)$, $\varepsilon_2 = enc_{k_1}(d)$, $\varepsilon_3 = enc_{k_2}(Com\_Q)$, $\sigma_{ACM} = S_{sk_{ACM}}(\varepsilon_1 \| \varepsilon_2 \| \varepsilon_3 \| IC\_Policy \| pk_{ICM} \| q)$, pk_{ACM}.

If step 4.1 or 4.2 fails, the protocol aborts. If both step 4.1 and step 4.2 succeed, ACM generates two secret keys k_1 and k_2 for symmetric key encryption. It encrypts k_1 and k_2 using public key pk_{ICM}, yielding ε_1. Then, it encrypts the query response d using k_1, yielding ε_2, determines the set of complementary queries (denoted as Com_Q) among the central query log at the discretion of user identity and host identity, and encrypts the set using k_2, yielding ε_3. After formulating the IC policy (denoted as IC_Policy) that is to be enforced by the user, ACM signs the IC policy, the complementary queries, ε_1, ε_2, and ε_3 using its private key, yielding digital signature σ_{ACM}. Finally, ACM sends ε_1, ε_2, ε_3, σ_{ACM}, and pk_{ACM} (including this public key's certificate) to ICM and updates the central query log by including q. Note that we actually adopt a rollback mechanism that if q is rejected (i.e., q leads to inference) later by ICM, ACM will be informed to remove q from the central query log. Note also that the reason for including pk_{ICM} and q in σ_{ACM} is to prevent replay attack; moreover, since a complementary query set contains the responses to these queries, so confidentiality should be provided against outside adversary taping on the communication channel.

Step 5. ICM enforces IC over the query response and IC policy in a protected execution environment supported by TPM.

Step 5.1. ICM: $V_{pk_{ACM}}(\sigma_{ACM})$.

Upon reception of the query response, ICM verifies the signature σ_{ACM}. If the signature is genuine, it proceeds to the next step.

Step 5.2. ICM $\rightarrow$ TPM: TPM_LoadKey2.

ICM issues TPM command TPM_LoadKey2 to TPM so as to load the private key sk_{ICM} to TPM. The input parameters taken by TPM_LoadKey2 include a TPM_KEY structure and authorization data. The TPM_KEY structure specifies the clear public key pk_{ICM} and the wrapped private key sk_{ICM} (which can be unwrapped by TPM), as well as information on PCR values bound to the key pair. The authorization data are computed from the user's password and the digest of the local query log: $SHA1(\text{password}\|\text{digest-of-query-log})$. Refer to steps 5.7 and 5.8 for the exact composition of TPM_KEY, why the authorization data is computed in such a way, and how digest-of-query-log is obtained.

Step 5.3. TPM $\rightarrow$ ICM: $(k_1, k_2) = D_{sk_{ICM}}(\varepsilon_1)$.

Once TPM decides that the protected user platform is in a trusted state and that the authorization data match that specified when the ICM key pair was generated (see step 5.7), TPM unwraps sk_{ICM}, uses it to decrypt ε_1, and returns k_1 and k_2 to ICM.

Step 5.4. ICM: $d = dec_{k_1}(\varepsilon_2)$, $Com\_Q = dec_{k_2}(\varepsilon_3)$.

ICM decrypts ε_2 using key k_1 to get the query response d, and ε_3 using key k_2 to obtain the set of complementary queries.

Step 5.5. ICM: $infcon(q_d, Q, IC\_Policy)$.

ICM enforces IC based on q_d and Q and the IC policy, where q_d denotes the current query q together with its response d, and Q denotes the union of the local query log and Com_Q. For reasons of generality, we assume that the query responses are used in IC, though they are not absolutely necessary for data independent algorithms such as Audit Expert.

Step 5.6. ICM: $\mathcal{Q} = \mathcal{Q} \cup \{q_d\} \cup Com\_Q$.

If $infcon(\cdot)$ arbitrates that q is safe, leading to no inference, ICM accepts the query: ICM updates the local query log $\mathcal{Q}$ by including q_d and the set of complementary queries and then reveals the query response d to the user. Otherwise, if $infcon(\cdot)$ decides that q_d causes inference, ICM rejects the query: ICM refuses to release d and updates $\mathcal{Q}$ by including only Com_Q. ICM finally sends a REJECT acknowledgement to the database server, so that the server updates the central query log by deleting q.[4] Note that inconsistency between the user's local query log and the server's central query log will occur if the server does not receive the REJECT acknowledgement sent by ICM. For clear presentation, we defer the discussion to this problem to Section 5.

4. To provide authenticity, the acknowledgement can be signed by TPM.

The above has completed the main task of IC. However, as we mentioned earlier ICM still needs to prepare a public-key encryption key pair for the next query session. The reason for this is that we design to bind the integrity of the local query log $\mathcal{Q}$ to the use of the private key. It is thus necessary to generate the key pair immediately following the update of $\mathcal{Q}$. The procedure of key generation is given as follows:

Step 5.7. ICM $\rightarrow$ TPM: TPM_CreatWrapKey.

In this step, ICM invokes TPM command TPM_CreatWrapKey to instruct TPM to generate an asymmetric key pair (pk_{ICM}, sk_{ICM}) and to wrap the private key sk_{ICM}. The input parameters of this command include

1. the handle of a wrapping key that can perform key wrapping,
2. the authorization data necessary to access the wrapping key,
3. a set of PCRs whose contents are bound to the wrapping operation, and
4. the information about the key to be generated (e.g., key length, key algorithm, key usage).

The piece of authorization data is an SHA-1 hash value (20 bytes) that is required for unwrapping the wrapped data. In our scenario, the piece of authorization data is derived from the user's password and the content digest of the local query log; that is, $SHA1(\text{password}\|\text{digest-of- query-log})$, where digest-of-query-log is obtained by applying a one-way function to the whole set of the user's queries accumulated in the local query log. If the local query log $\mathcal{Q}$ is maliciously modified later, the authorization data calculated for unwrapping operation in the next query session will be refuted by TPM, and as a result, sk_{ICM} cannot be accessed by ICM.

The key pair generated by TPM_CreatWrapKey is bound to a state of the platform. The binding is achieved by specifying a set of PCRs whose contents are bound to the wrapping operation. In our case, the PCRs record the integrity measurement metrics of the protected platform. This binding ensures that sk_{ICM} cannot be unwrapped unless the user's platform is in a trusted state.

Step 5.8. TPM $\rightarrow$ ICM: TPM_Key.

Finally, TPM returns to ICM a TPM_Key data structure, which contains public key pk_{ICM} and the corresponding private key sk_{ICM} encrypted by a wrapping key. TPM_Key also contains a field TPM_Auth_Data_Usage, which can take one of the following three values: 1) TPM_Auth_Never, 2) TPM_Auth_Always, and 3) TPM_Auth_Priv_Use_Only. The first case allows the invoking party to load the private key without submission of any authorization data, while the second and third cases associate authorization data with the public/private key pair and the private key only, respectively. In our case, it suffices to instruct TPM to set TPM_Auth_Data_Usage to TPM_Auth_Priv_Use_Only.

1. ICM $\rightarrow$ ACM: id, q
2. ACM $\rightarrow$ ICM: n_{ACM}
3. ICM $\rightarrow$ ACM: $\sigma_{TPM} = S_{sk_{TPM}}(pk_{ICM}||n_{ACM}||im)$
4. ACM $\rightarrow$ ICM: $\varepsilon_1 = E_{pk_{ICM}}(k_1, k_2)$, $\varepsilon_2 = enc_{k_1}(d)$, $\varepsilon_3 = enc_{k_2}(Com\_Q)$, $\sigma_{ACM} = S_{sk_{ACM}}(\varepsilon_1||\varepsilon_2||\varepsilon_3||IC\_Policy||pk_{ICM}||q)$

Fig. 5. Simplified protocol.

4 Security Analysis

In this section, we formally prove the security, in particular, the authentication property, of our protocol. Given that the security services provided by TPM, the protected execution environment on top of TPM, and the cryptographic primitives we employed are secure (in a sense that these services are not compromised), it is not difficult to verify that our protocol meets the security requirements listed in Section 3.1 if the authentication property of our protocol can be formally proven.

Our protocol involves ACM on the server side and ICM on the user side. According to Fig. 3, ICM takes root in a trusted subsystem, which consists of secure OS kernel, TPM, and other trusted hardware. The trusted subsystem restricts ICM's behavior from accessing the secrets and operations of TPM, unless it is authorized, as specified in the protocol. In the presence of "perfect" TPM (see Section 4.2 for our assumptions on TPM), ICM and TPM can be considered as a single entity (represented by ICM) in our protocol analysis. Our protocol is essentially an authentication protocol between ICM and ACM, with ICM having access to relevant TPM commands as local operations. The main objective of authentication is to ensure that *ACM validates the security state of ICM before sending out any query response*. We formally prove this property using the rank function [53], a specialized theorem-proving method for establishing the correctness of authentication protocols based on communicating sequential processes (CSPs) [54]. This will eliminate typical attacks such as replay attacks and masquerade attacks that are targeted at our authentication protocol. It is cautioned that our proof assumes prior knowledge of CSP [54].

To perform the proof, the rank function approach requires modeling of the following three components: 1) the protocol, 2) the environment (attacker), and 3) the security requirements on the protocol:

- The protocol is captured as a CSP process in terms of the behavior of each system party.
- The environment is also described as a CSP process. It is considered to be an unreliable medium that can lose, reorder, and duplicate messages. The particular behavior captured within the medium is precisely the behavior that the protocol is designed to overcome.
- The security requirements on the protocol are expressed as **sat** specifications on the observable behaviors of the overall system.

When these components are modeled, one can use well-established proof techniques to verify whether the protocol satisfies its security requirements. We next elaborate on the proof.

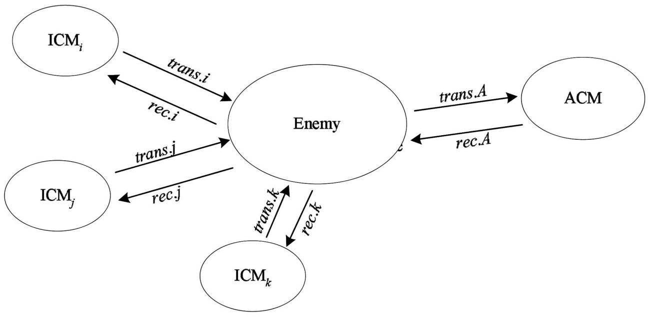

Fig. 6. Network model.

4.1 Protocol Simplification

Our protocol can be simplified as follows for proof clarity: 1) The public keys and corresponding certificate information are removed from the protocol messages. This information is used for the verification of digital signatures and in principle can be obtained from a public directory. 2) The use of the hash function is removed in the computation of σ_{TPM}, yielding $\sigma_{TPM} = S_{sk_{TPM}}(pk_{ICM}\|n_{ACM}, id\|im)$, where id is the concatenation of id_U and id_H. In the sequel, for simplicity, we omit id as we can view n_{ACM} as a compound of itself and id. The removal of the hash function does not affect the correctness of our analysis, as there is no secret involved in hashing [29]. Fig. 5 illustrates the simplified version of our protocol, which captures only the essential messages exchanged between ICM and ACM without revealing the details of local operations.

4.2 Assumptions on TPM

Several assumptions are made to model the involvement of TPM [15]. First, it is assumed that a user's host has access to a single TPM and is running a single instance of ICM. An attacker may also have access to a single TPM; however, the attacker's platform is not trusted by the database server, which distinguishes it from honest users. Second, only TPM commands that are used in the protocol are modeled. This means that the attacker cannot use other TPM commands to gain advantages. This assumption is strong but reasonable, as we are in a position to analyze the protocol in the presence of "perfect" TPM. The resulting analysis can be considered as a first step toward further analysis on a richer set of TPM behaviors.

4.3 Protocol Modeling

Attacker. The model of the network is standard, based on the Dolev-Yao model [22]. As shown in Fig. 6, the network is composed of a set of honest users' ICMs and a single server's ACM. The attacker is in full control of the network, and all communication passes through it. The attacker may choose to block or redirect messages, send messages of its choice, and masquerade as other users. The users and the server transmit messages on channel $trans$ and receive messages on channel rec. The event $trans.i.j.m$ represents the transmission of message m to j from i. Likewise, $rec.j.i.m$ indicates the reception of m by j from i.

The capabilities of the attacker are bounded by a finite set of deductions, by which it can generate new messages from the set of messages it already knows. Let $X \vdash m$ denote that new message m is deduced from a set X of messages. In our context, the attacker is capable of the following three deductions related to the TPM commands used in the protocol, in addition

$$\frac{\text{TPM\_CreateWrapKey}}{X \vdash pk \| E_{TPM_E}(sk)} ((pk, sk) \in AK_E)$$

$$\text{TPM\_CertifyKey} \quad \frac{X \vdash n \quad X \vdash pk}{X \vdash S_{sk_{TPM_E}}(pk \| n \| im)} (-validate(im))$$

$$\text{TPM\_LoadKey2} \quad \frac{X \vdash E_{TPM}(sk) \quad X \vdash E_{pk}(m)}{X \vdash m} ((pk, sk) \in AK_E)$$

to the standard deductions77 as pointed out in [53] (recall that the attacker is allowed access to TPM). The first deduction results from the attacker's access to TPM_CreateWrapKey command, where AK_E denotes the set of key pairs available to the attacker, and $E_{TPM_E}(sk)$ represents that TPM stores sk in its protected storage. The second deduction states that given a nonce n and a public key pk, the attacker can deduce $S_{sk_{TPM_E}}(pk \| n \| im)$ (the attestation response) by invoking TPM_CertifyKey, where the side condition $\neg validate(im)$ ensures that im represents a platform state untrusted to the database server. The third deduction states that given a ciphertext $E_{pk}(m)$ and the corresponding private sk stays in the protected storage of the TPM, the attacker can get the cleartext m by invoking TPM command TPM_LoadKey2.

The following process $ENEMY$ characterizes the behavior of the attacker in possession of a set X of messages:

$$\begin{aligned} ENEMY(X) = {} & trans?i?j?m \rightarrow ENEMY(X \cup \{m\}) \\ & \Box \\ & \Box_{\substack{i,j:U \\ X \vdash m}} \quad rec!i!j!m \rightarrow ENEMY(X). \end{aligned}$$

The first branch of choice models the attacker intercepting and adding any new message m (sent from any user to any other user) to the set of messages X the attacker knows. The second branch states that the attacker can send any message m that can be generated from X to any user i, pretending it coming from any other user j.

The attacker is initially parameterized by a set IK representing the attacker's initial knowledge, such as the nonces that the attacker can use, usernames, public keys, and compromised old session keys:

$$ENEMY = ENEMY(IK).$$

Users. Each user is identified by its identity. We thus model each ICM i by CSP process $ICM_i(id_i, AK_i)$, where id_i is user identity and AK_i is the set of asymmetric key pairs that ICM i can use. As a result, $USERS$ models the behavior of all ICMs as an interleaving of ICM_i (U is the set of all users):

$$USERS = |||_{i:U} ICM_i(id_i, AK_i).$$

Each ICM_i models the messages engaged by ICM i:

$$\begin{aligned} ICM_i(id_i, AK_i) = {} & \Box_{(pk_i, sk_i):AK_i,:q:Q_i}\ trans.i.A.q \\ & \rightarrow \Box_{n_A:N}\ rec.i.A.n_A \\ & \rightarrow \sqcap_{im:M}\ trans.i.A.\Big(S_{sk_{TPM_i}}(pk_i \| n_A \| im)\Big) \\ & \rightarrow \Box_{k:K}\ rec.i.A.\big(E_{pk_i}(k), enc_k(d), \\ & \quad S_{sk_A}\big(E_{pk_i}(k) \| enc_k(d) \| \text{IC\_Policy} \| n_A\big)\big) \\ & \rightarrow ICM_i(id_i, AK_i \setminus \{(pk_i, sk_i)\}), \end{aligned}$$

where A is the identity of ACM, Q_i is the set of queries, N is the set of nonces, M is the set of integrity metrics, and K is the set of symmetric keys. The recursive definition of ICM_i ensures that only one instance of ICM will be running at a time.

Server. The CSP process $SERVER$ models the behavior of interleaving instances of a single ACM:

$$SERVER = |||_{n_A:N} ACM(n_A).$$

Each instance of ACM is modeled by process $ACM(n_A)$, parameterized by a nonce n_A (see Fig. 5):

$$\begin{aligned} ACM(n_A) = {} & \Box_{i:U, q:Q_i}\ rec.A.i.q \\ & \rightarrow trans.A.i.n_A \\ & \rightarrow \Box_{im:M, pk_i:AK_i}\ rec.A.i.\Big(S_{sk_{TPM_i}}(pk_i \| n_A \| im)\Big) \\ & \rightarrow validate(im) \& signal.Trust.i.n_A.app(q) \\ & \rightarrow \quad trans.A.i.\big(E_{pk_i}(k), enc_k(d), \\ & \quad S_{sk_A}\big(E_{pk_i}(k) \| enc_k(d) \| \text{IC\_Policy} \| pk_{ICM} \| q\big)\big), \\ & \quad \text{where } k = key(n_A, i) \text{ and } d = app(p). \end{aligned}$$

Note that in the definition of $ACM(n_A)$, upon reception of $S_{sk_{TPM_i}}(pk_i \| n_A \| im)$, ACM performs a signal event $signal.Trust.i.n_A.app(q)$ if im is validated as representing a trusted state of the platform that issues q in a certain protocol run. The secret key k is derived by $key(n_A, i)$ and d is derived by $app(p)$.

Entire Network. The entire network is thus modeled by process NET, a parallel composition of $USERS$, $SERVER$, and $ENEMY$. The process is synchronized on $trans$ and rec:

$$NET = (USERS|||SERVER)|[\{|trans, rec|\}]|ENEMY.$$

4.4 Authentication Objective

The authentication objective of the protocol is that ACM responds to the query from ICM only after it has validated the state of ICM. In other words, ACM will not send out the query response unless ICM has already attested its state as expected. For a particular ICM I, message $trans.I.A.S_{sk_{TPM_I}}(pk_I \| n_{ACM} \| im)$ indicates that ICM has already attested its state in corresponding to nonce n_{ACM}. For ACM, message $signal.Trust.I.n_{ACM}.d_I$ indicates that ACM has already validated the state of I, signaling that I is trusted to receive the response value d_I in a run involving n_{ACM}. The authentication objective can thus be formalized as a **sat** predicate:

$$NET \ \textbf{sat} \ R \ \textbf{precedes} \ T,$$

where $R = \{trans.I.A.S_{sk_{TPM_I}}(pk_I \| n_{ACM} \| im) | validate(im)\}$, and $T = \{signal.Trust.I.n_{ACN}.d_I\}$.

4.5 Rank Function

The rank function approach [53] assigns a rank *sec* to the messages that should remain secret during the protocol and

Users and host:	$\rho(i) = pub$
Nonces:	$\rho(n) = pub$
Integrity metrics:	$\rho(im) = pub$
Queries:	$\rho(q) = pub$
Response values:	$\rho(d) = \begin{cases} \text{sec} & d = d_I \\ pub & \text{otherwise} \end{cases}$
Complement. queries:	$\rho(Com\_Q_i) = \begin{cases} \text{sec} & i = I \\ pub & \text{otherwise} \end{cases}$
Symmetric keys:	$\rho(key(n_A, i)) = \begin{cases} \text{sec} & n_A = n_{ACM} \wedge i = I \\ pub & \text{otherwise} \end{cases}$
Sym. key encryption:	$\rho(enc_k(m)) = \rho(k)$
TPM keys:	$\begin{cases} \rho(pk_{TPM_i}) = pub \\ \rho(sk_{TPM_i}) = \text{sec} \end{cases}$
TPM Protected Stor.:	$\rho(E_{TPM_i}(m)) = \rho(m)$
TPM signatures :	$\rho(S_{sk_{TPM_i}}(m)) = \begin{cases} \text{sec} & i = I \wedge m \in \{(pk_i \Vert n_{ACM} \Vert im) \mid validate(im)\} \\ pub & \text{otherwise} \end{cases}$
Signals:	$\rho(signal.Trust.i.n_A.d) = \begin{cases} \text{sec} & i = I \wedge n_A = n_{ACM} \wedge d = d_I \\ pub & \text{otherwise} \end{cases}$
Server keys:	$\begin{cases} \rho(pk_s) = pub \\ \rho(sk_s) = \begin{cases} \text{sec} & s = A \\ pub & \text{otherwise} \end{cases} \end{cases}$
Server signature :	$\rho(S_{sk_S}(m)) = \rho(m)$
Ephemeral keys:	$\begin{cases} \rho(pk_i) = pub \\ \rho(sk_i) = \begin{cases} \text{sec} & i = I \\ pub & \text{otherwise} \end{cases} \end{cases}$
Eph. Key encryption:	$\rho(E_{pk_i}(m) = \rho(m)$
Multiple messages (or concatenations):	$\rho(m_1, m_2) = \min(\rho(m_1), \rho(m_2))$
Events:	$\begin{cases} \rho(trans.i.j.m) = \rho(m) \\ \rho(rec.j.i.m) = \rho(m) \end{cases}$

Fig. 7. Rank function.

a rank *pub* to the messages that should be public ($sec < pub$). The central rank theorem [53] gives four conditions the rank function must satisfy in order to conclude that the protocol indeed upholds the **sat** predicate. The interpretation of the four conditions within our context is given below:

1. The messages in the initial set IK known to the $ENEMY$ process are of rank *pub*.
2. Any message generated by $ENEMY$ from X under the deductions rules has a rank *pub* whenever messages in X have a rank *pub*.
3. $signal.Trust.I.n_{ACM}.d_I$ in R has a rank *sec*.
4. An ICM, when it is blocked from sending $trans.I.A.S_{sk_{TPM_I}}(pk_I \| n_{ACM} \| im)$, can never give out a message of rank *sec* unless it has previously received a message of rank *sec*.

Fig. 7 shows the definition of our rank function ρ. The security of our protocol is proven by verifying that the rank function ρ satisfies the four conditions listed above, thus the authentication objective is achieved. The first three conditions can be checked independently of any specific CSP process (with a slight exception in our case that the attacker is capable of three extra deductions related to TPM commands), and it is not difficult to verify them against ρ. The fourth condition, however, requires verifying $USERS|||SERVER$ (CSP modeling of the honest agents) against R **precedes** T. In particular, all agents that are blocked on events from R should preserve the rank function in a sense that if an agent receives messages of rank *pub* only, then it should send messages of rank *pub* only:

$$(USERS|||SERVER|[R]|Stop) \textbf{ sat preserves } \rho.$$

This can be interpreted as follows: If message $trans.I.A.S_{sk_{TPM_I}}(pk_I \| n_{ACM} \| im)$, which is the only message in R, is blocked, then $signal.Trust.I.n_{ACM}.d_I$ does not occur. According to the laws of CSP, this can be proven if all of the following four predicates hold:

1. $ICM_I(id_I, AK_I)|[R]|Stop)$ **sat preserves** ρ;
2. $ICM_i(id_i, AK_i)|[R]|Stop)$ **sat preserves** ρ, where $i \in U \setminus \{I\}$;
3. $ACM(n_{ACM})|[R]|Stop)$ **sat preserves** ρ; and
4. $ACM(n_A)|[R]|Stop)$ **sat preserves** ρ, where $n_A \in N \setminus \{n_{ACM}\}$.

Checking these four predicates is a routine task. The complete proof that these four predicates hold can be found very similar to that given in [15].

5 Discussions

In this section, we discuss several extensions to the above system.

Defending against collusion attack. A typical attack against IC systems is *collusion attack*. A collusion attack involves several users forming a *collusion group* and combining their query logs so as to infer some sensitive information that cannot be derived from any individual query log. The collusion attack is inherently difficult to mitigate and presents as a serious inhibitor in the practical use of IC [3]. We must point out that there seems no technique can prevent a user from purposely writing down her queries (as well as query responses) and using them in collusion with other users. What we can achieve is to restrict malevolent users from directly using the query logs that are maintained by ICMs for collusion. This requires the *confidentiality of query log* to be maintained against any programs other than ICM. To attain this requirement, the local query log can be encrypted by ICM using a secret key, which can be stored in and retrieved from the sealed storage of TPM by ICM only (using TPM_Seal and TPM_Unseal commands). Note that in this case, the authorization data for unwrapping operations (i.e., wrapping sk_{ICM}) must be changed to $SHA1$(password | | digest of encrypted query log) in the above protocol.

In certain cases, the database server may be able to "blacklist" some collusion groups of suspicious users who may collude, with the help of certain out-of-band information (e.g., the users from the same network domain). To further mitigate the collusion attack, IC should be enforced based on queries from all users in a collusion group rather than from each individual user. This can be easily achieved in our new architecture, with the database server managing the central query log. More specifically, when any user in a collusion group issues a query, the server sends to ICM the complementary queries, which include not only the user's latest past queries issued at other hosts but also the queries from all other users in the same collusion group; ICM can then enforce IC based on the combination of its local query log and the complementary queries it receives.

By and large, we can argue that our new architecture in no way adversely affects the implementation of any technique against collusion attack that can be applied in the traditional architecture.

Casual damage of local query log. We have associated the integrity of the query log with the access to the ephemeral private key used to obtain the query responses in order to deter malignant compromise to the local query log by the user. However, this may cause problems in certain cases that the local query log is damaged casually, e.g., by the user's misoperations or by break-ins: the user is unable to query the database permanently, although she does not purposely damage the local query log. This problem can be solved by establishing an extra protocol that allows the user to retrieve her queries kept in the central query log, under the condition that she successfully identifies herself to the database server.

Possible inconsistency between the local and central query logs. In our protocol, we adopted a rollback mechanism that by default ACM will include a query in the central query log regardless of whether the query is accepted or rejected by ICM; and ACM roll backs by deleting the query if the query is later rejected by ICM. An inconsistency problem occurs if the acknowledgement gets lost en route so that ACM does not get it, e.g., due to the unreliability of the network or because an attacker blocks the acknowledgement: the local query log does not contain the query, but the central query log has it. There is no adverse effect if the user uses a single host, but it indeed has serious consequences in the case of a user using multiple hosts. In such a case, the query will be sent to other hosts as a part of the complementary query set. Since the query has already caused inference with the previous queries, no matter what the new queries are, they are always rejected. This problem is essentially a "fairness" issue that involves updating to the two distinct query logs. While we can adapt the techniques of fair exchange protocols, e.g., [2] and [4], for a remedy, the resulting solution would be quite expensive. We can alleviate this problem as follows: The user should be careful if her query is rejected and keeps a copy of the REJECT acknowledgement. When the user queries the database next time, she must present the acknowledgement to the server. If ACM has already received the REJECT acknowledgement, it ignores it; otherwise, it updates the central query log accordingly before processing the user's query.

Effect of user's host crash. In the security analysis in Section 4, we did not consider an exceptional scenario that the user's host is crashed prematurely before it completes a protocol execution. Note that the user has no motive to crash her machine, since she wants to get the query response. However, host crashes could be caused by break-ins or hardware/software failure. We next discuss the effect of such incidents. Referring to Fig. 4 and the protocol in Section 3, we consider three possible points where the user host could crash. The first is at a point that ICM has completed step 1 but before its engaging in step 3. The effect is that ACM does not receive attestation response, and thus, it aborts the protocol. As neither party updates its query log, no adverse effect is incurred. The second possible crash point is at step 5.6, where ICM has updated its local query log but before the query response is presented to the user. This causes an unfair situation to the user. A possible solution is that the user reissues the query after the host is recovered from crash. The third possible crash point is after step 4.3 but before ICM sends out the REJECT acknowledgement in case the query is rejected. Since the user does not even get a copy of the REJECT acknowledgement, she is not able to convince the server to update the central query log next time. A solution to this problem is that the user reissues the query so as to get a copy of the REJECT acknowledgement after the crash is recovered; this acknowledgement can be used by the user as an evidence to roll back the rejected query on the server side next time when she queries the database.

Database update. Database update (e.g., deletion of some records) may necessitate updating the user's local query log on the user's side. The central query log helps the database server solve this problem. In case of database update, the database server can determine the set of queries that are affected by the update process. When a user queries the database, the database server first checks for the affected queries that belong to the user; it then informs the user to update its query log so that the IC will be enforced upon the latest query log.

6 Conclusions

We have proposed a new IC architecture and an IC protocol upon it. While traditional IC is enforced by a database server for all its users, our solution entrusts each user's host with the enforcement of IC for itself, provided that the user's host is equipped with trusted computing technology. By decentralizing the highly computation-intensive task of IC, our solution enjoys much better system scalability and is thus suitable for supporting a large number of users in real-world database systems. In comparison, the traditional IC configuration can only support a small number of users due to the bottleneck of enforcing IC for all users on the server side. In this sense, our solution removes the crucial impediment in traditional IC configuration and identifies a new paradigm for the practical implementation of IC. Our solution can work with any existing IC technique; even a hybrid system of mixing our solution (for some users whose platforms are TPM equipped) with traditional IC (for those users who may not implement trusted computing) can be easily set up. As a price to pay, the new architecture incurs tolerable local storage and communication overheads.

Our solution relies on the use of trusting computing technology, which is envisioned to be ubiquitous in several years. Among other functionalities, the trusted computing technology enables platform attestation for convincing the remote database server that a user platform is in a trusted state so that IC can be enforced on the user side as expected. Our IC protocol is proposed to formalize the interactions between the user platform and the database server based on remote attestation. The authentication property of the protocol is formally proven with the rank function approach. We note that the security of our solution is proven based on the "perfect TMP" assumption and the Dolev-Yao adversary model. It remains unaddressed if the "perfect TMP" assumption does not hold, or the adversarial behaviors are not captured by the Dolev-Yao model. As an example, the vulnerabilities caused by host crashes are not covered in our security proof. In addition, we did not prove all security properties (e.g., integrity of ICM and authenticity of IC policy), as they can be straightforwardly derived and enforced once the authentication property is established.

Appendix

A Brief Introduction to TPM

The TCG [60] defines a set of trusted computing specifications [57], [58], [59] aiming to provide a hardware-based root of trust and a set of mechanisms to propagate trust to applications as well as across platforms. The root of trust in TCG is a tamper resistant hardware engine, called TPM. TPM is assumed robust against both hardware and software attacks from either the underlying host or external sources. It is a self-contained coprocessor with specialized functions such as random number generation, RSA key generation, RSA public key algorithms, SHA-1 hash function, HMAC function, and volatile and nonvolatile memories. TPM may have an identity, with an AIK pair associated it. The AIK is issued by the Privacy Certification Authority (P-CA), together with a certificate that binds the public AIK to TPM. The private AIK can only be used by TPM to generate signatures. TPM is also installed with a unique Endorsement Key (EK) pair by the manufacturer before shipping, which is exclusively used for data encryption purposes. The private EK is securely held in TPM for decryption operations, and the public EK is associated with an endorsement credential and accessible to any application for encrypting data to TPM.

TPM facilitates storage of integrity measurement metrics of the underlying platform to the internal registers and reporting of the metrics. In particular, TPM contains a set of PCRs, which is used to record the integrity and configuration metrics of a running platform from booting to OS loading to application software loading. Each PCR value is the SHA-1 hash value of its current value concatenated with the new measured value of the protected objects, i.e., $\mathrm{PCR}[n] \leftarrow \mathrm{SHA1}(\mathrm{PCR}[n] \| \text{latest measured value})$. Measurement of a platform's integrity results in the generation of measurement events, which comprises two classes of data: Measured values, which are representations of the data or program code to be measured; and measurement digests, which are hashes of the measured values. The measurement digests are stored to PCRs in TPM, while the measured values are stored to the Stored Measurement Log (SML) outside TPM. With integrity measurement in place, TPM (attestator) can attest to a remote challenging platform (challenger) the integrity state of its underlying platform through *platform attestation*. In particular, attestation works as follows: the challenger sends a challenge message to the attestator, who then returns the related PCR values signed by its AIK, together with the relevant SML entries and the corresponding credentials. The challenger validates the response and decides whether the attestator platform is trusted for its intended purpose.

TPM provides hardware-based secure storage for secrets by storing a *storage root key* (SRK) inside the chip and never exposing it outside. *Sealed storage* is an essential security mechanism offered by TPM. Sealed storage protects sensitive data with integrity values. In particular, besides applying an encryption key (RSA public key encryption) to encrypt the data, one or more PCR values are stored together with protected data during the encryption. Consequently, TPM releases protected data only if the current PCR values match those stored during encryption. The encryption key is protected either directly by SRK, or by a key protected by the SRK. Hence, the SRK acts as the root of *trust for storage*, and all encryption keys can actually form a key hierarchy. SRK is the only storage key permanently residing within TPM.

Acknowledgments

A preliminary version of this paper has been published in the Proceedings of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC '07), LNCS 4602, pp. 243-258, 2007. This research is partly supported by the Office of Research, Singapore Management University. The authors would like to thank the reviewers for their valuable comments and suggestions, which help them improve the quality of this paper.

References

[1] J.O. Achugbue and F.Y. Chin, "The Effectiveness of Output Modification by Rounding for Protection of Statistical Databases," *INFOR*, vol. 17, no. 3, pp. 209-218, 1979.

[2] N. Asokan, M. Schunter, and M. Waidner, "Optimistic Protocols for Fair Exchange," *Proc. ACM Conf. Computer and Comm. Security (CCS '97)*, pp. 7-17, 1997.

[3] N.R. Adam and J.C. Wortmann, "Security-Control Methods for Statistical Databases: A Comparative Study," *ACM Computing Surveys*, vol. 21, no. 4, pp. 516-556, 1989.

[4] F. Bao, R. Deng, and W.B. Mao, "Efficient and Practical Fair Exchange with Offline TTP," *Proc. IEEE Symp. Security and Privacy (S&P '98)*, pp. 77-85, 1998.

[5] L.L. Beck, "A Security Mechanism for Statistical Databases," *ACM Trans. Database Systems*, vol. 5, no. 3, pp. 316-338, 1980.

[6] M. Chen, L. McNamee, and M.A. Melkanoff, "A Model of Summary Data and Its Applications to Statistical Databases," *Proc. Fourth Int'l Conf. Statistical and Scientific Database Management (SSDBM '89)*, pp. 354-372, 1989.

[7] F.Y. Chin, "Security Problems on Inference Control for SUM, MAX, and MIN Queries," *J. ACM*, vol. 33, pp. 451-464, 1986.

[8] R. Chow, P. Goll, and J. Staddon, "Inference Detection Technology for Web 2.0," *Proc. Web 2.0 Security and Privacy (W2SP)*, 2007.

[9] F.Y. Chin, P. Kossowski, and S.C. Loh, "Efficient Inference Control for Range Sum Queries," *Theoretical Computer Science,* vol. 32, pp. 77-86, 1984.

[10] F.Y. Chin and G. Özsoyoglu, "Security in Partitioned Dynamic Statistical Databases," *Proc. IEEE Int'l Computer Software and Applications Conf. (COMPSAC '79),* pp. 594-601, 1979.

[11] F.Y. Chin and G. Özsoyoglu, "Statistical Database Design," *ACM Trans. Database Systems,* vol. 6, no. 1, pp. 113-139, 1981.

[12] F.Y. Chin and G. Özsoyoglu, "Auditing and Inference Control in Statistical Databases," *IEEE Trans. Software Eng.,* vol. 6, pp. 574-582, 1982.

[13] L.H. Cox, "Suppression Methodology and Statistical Disclosure Control," *J. Am. Statistical Assoc.,* vol. 75, no. 370, pp. 377-385, 1980.

[14] L.H. Cox and L.V. Zayatz, "An Agenda for Research on Statistical Disclosure Limitation," *J. Official Statistics,* vol. 75, pp. 205-220, 1995.

[15] R. Delicata, "An Analysis of Two Protocols for Conditional Access in Mobile Systems," Technical Report CS-04-13, Dept. Computing, Univ. of Surrey, 2005.

[16] D.E. Denning, *Cryptography and Data Security.* Addison Wesley, 1982.

[17] D.E. Denning, "Secure Statistical Databases with Random Sample Queries," *ACM Trans. Database Systems,* vol. 5, no. 3, pp. 88-102, 1980.

[18] D.E. Denning, "A Security Model for the Statistical Database Problem," *Proc. Second Int'l Workshop Management,* pp. 1-16, 1983.

[19] D.E. Denning, P.J. Denning, and M.D. Schwartz, "The Tracker: A Threat to Statistical Database Security," *ACM Trans. Database Systems,* vol. 4, no. 1, pp. 76-96, 1979.

[20] D.E. Denning and J. Schlörer, "Inference Control for Statistical Databases," *Computer,* vol. 16, no. 7, pp. 69-82, 1983.

[21] D. Dobkin, A.K. Jones, and R.J. Lipton, "Secure Databases: Protection against User Influence," *ACM Trans. Database Systems,* vol. 4, no. 1, pp. 97-106, 1979.

[22] D. Dolev and A.C. Yao, "On the Security of Public Key Protocols," *IEEE Trans. Information Technology,* vol. 29, no. 2, pp. 198-208, 1983.

[23] J.S. Erickson, "Fair Use, DRM, and Trusted Computing," *Comm. ACM,* vol. 46, no. 4, pp. 34-39, 2003.

[24] C. Farkas and S. Jajodia, "The Inference Problem: A Survey," *SIGKDD Explorations,* vol. 4, no. 2, pp. 6-11, 2002.

[25] I.P. Fellegi and J.L. Phillips, "Statistical Confidentiality: Some Theory and Applications to Data Dissemination," *Annals of Economic and Social Measurement,* vol. 3, no. 2, pp. 399-409, 1974.

[26] E. Gallery, "An Overview of Trusted Computing Technology," *Trusted Computing,* C. Mitchell, ed., pp. 29-114, 2005.

[27] B.G. Greenberg, J.R. Abernathy, and D.G. Horvitz, "Application of Randomized Response Technique in Obtaining Quantitative Data," *Proc. Social Statistics Section, American Statistical Assoc.,* pp. 40-43, 1969.

[28] L.J. Hoffman, *Modern Methods for Computer Security and Privacy.* Prentice-Hall, 1977.

[29] M.L. Hui and G. Lowe, "Safe Simplifying Transformations for Security Protocols," *Proc. 12th Computer Security Foundations Workshop,* pp. 32-43, 1999.

[30] A. Iliev and S.W. Smith, "Protecting User Privacy via Trusted Computing at the Server," *IEEE Security and Privacy,* vol. 3, no. 2, Mar./Apr. 2005.

[31] J. Kleinberg, C. Papadimitriou, and P. Raghavan, "Auditing Boolean Attributes," *Proc. Ninth ACM Sigmod-SIGACT-SIGART Symp. Principles of Database Systems (PODS '00),* pp. 86-91, 2000.

[32] LaGrande Technology Architecture. Intel Developer Forum, 2003.

[33] D. Lefons, A. Silvestri, and F. Tangorra, "An Analytic Approach to Statistical Databases," *Proc. Ninth Int'l Conf. Very Large Data Bases (VLDB '83),* pp. 260-273, 1983.

[34] Y. Li, H. Lu, and R.H. Deng, "Practical Inference Control for Data," *Proc. IEEE Symp. Security and Privacy (S&P '06),* pp. 115-120, 2006.

[35] Y. Li, L. Wang, X.S. Wang, and S. Jajodia, "Auditing Interval-Based Inference," *Proc. 14th Int'l Conf. Advanced Information Systems Eng. (CAiSE '02),* pp. 553-567, 2002.

[36] C.K. Liew, W.J. Choi, and C.J. Liew, "A Data Distortion by Probability Distribution," *ACM Trans. Database Systems,* vol. 10, no. 3, pp. 395-411, 1985.

[37] F.M. Malvestuto and M. Mezzini, "Auditing Sum-Queries," *Proc. Ninth Int'l Conf. Database Theory (ICDT '03),* pp. 504-509, 2003.

[38] F.M. Malvestuto and M. Moscarini, "An Audit Expert for Large Statistical Databases," *Statistical Data Protection, EUROSTAT '99,* pp. 29-43, 1999.

[39] C. Mitchell, *Trusted Computing.* IEE, 2005.

[40] G. Özsoyoglu and J. Chung, "Information Loss in the Lattice Model of Summary Tables Due to Suppression," *Proc. IEEE Symp. Security and Privacy (S&P '86),* pp. 75-83, 1986.

[41] A. Perrig, S.W. Smith, D. Song, and J.D. Tygar, "SAM: A Flexible and Secure Auction Architecture Using Trusted Hardware," *eJETA.org: The Electronic J. E-Commerce Tools and Applications,* vol. 1, no. 1, 2002.

[42] J.P. Reiss, "Practical Data Swapping: The First Step," *Proc. IEEE Symp. Security and Privacy (S&P '80),* pp. 36-44, 1980.

[43] J. Staddon, P. Goll, and B. Zimny, "Web-Based Inference Detection," *Proc. USENIX Security Symp.,* pp. 71-86, 2007.

[44] J. Staddon, "Dynamic Inference Control," *Proc. ACM Sigmod Workshop Research Issues in Data Mining and Knowledge Discovery (DMKD '03),* pp. 94-100, 2003.

[45] R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn, "Attestation-Based Policy Enforcement for Remote Access," *Proc. ACM Conf. Computer and Comm. Security (CCS '04),* pp. 308-317, 2004.

[46] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and Implementation of a TCG-Based Integrity Measurement Architecture," *Proc. USENIX Security Symp.,* pp. 223-238, 2004.

[47] G. Sande, "Automated Cell Suppression to Reserve Confidentiality of Business Statistics," *Proc. Second Workshop Statistical Database Management,* pp. 346-353, 1983.

[48] R. Sandhu and X. Zhang, "Peer-to-Peer Access Control Architecture Using Trusted Computing Technology," *Proc. 10th ACM Symp. Access Control Models and Technologies (SACMAT '05),* pp. 147-158, 2005.

[49] J. Schlörer, "Confidentiality of Statistical Records: A Threat Monitoring Scheme of On-line Dialogue," *Methods of Information in Medicine,* vol. 15, no. 1, pp. 36-42, 1976.

[50] J. Schlörer, "Disclosure from Statistical Databases: Quantitative Aspects of Trackers," *ACM Trans. Database Systems,* vol. 5, no. 4, pp. 467-492, 1980.

[51] J. Schlörer, "Information Loss in Partitioned Statistical Databases," *Computer J.,* vol. 26, no. 3, pp. 218-223, 1983.

[52] E. Shi, A. Perrig, and L. Van Doorn, "BIND: A Fine-grained Attestation Service for Secure Distributed Systems," *Proc. IEEE Symp. Security and Privacy (S&P '05),* pp. 154-168, 2005.

[53] S. Schneider, "Verifying Authentication Protocols with CSP," *Proc. 10th Computer Security Foundations Workshop,* pp. 3-17, 1997.

[54] S. Schneider, *Concurrent and Real-Time Systems: The CSP Approach.* Addison-Wesley, 1999.

[55] S.W. Smith, *Trusted Computing Platforms: Design and Applications.* Springer, 2005.

[56] S.W. Smith and D. Safford, "Practical Server Privacy Using Secure Coprocessors," *IBM Systems J.* (special issue on End-to-End Security), vol. 40, pp. 683-695, 2001.

[57] *TCG. TPM Main, Part 1 Design Principles, TCG Specification Ver.1.2,* Revision 62, http://www.trustedcomputinggroup.org, 2003.

[58] *TCG. TPM Main, Part 2 TPM Data Structure, TCG Specification Ver.1.2,* Revision 62, http://www.trustedcomputinggroup.org, 2003.

[59] *TCG. TPM Main, Part 3 Commands, TCG Specification Ver.1.2,* Revision 62, www.trustedcomputinggroup.org, 2003.

[60] Trusted Computing Group, http://www.trustedcomputinggroup.org, 2006.

[61] J.F. Traub, Y. Yemini, and H. Wozniakowski, "The Statistical Security of a Statistical Database," *ACM Trans. Database Systems,* vol. 9, no. 4, pp. 672-679, 1984.

[62] L. Wang, Y. Li, D. Wijesekera, and S. Jajodia, "Precisely Answering Multi-Dimensional Range Queries without Privacy Breaches," *Proc. European Symp. Research in Computer Security (ESORICS '03),* pp. 100-115, 2003.

[63] L. Wang, D. Wijesekera, and S. Jajodia, "Cardinality-Based Inference Control in Sum-Only Data Cubes," *Proc. European Symp. Research in Computer Security (ESORICS '02),* pp. 55-71, 2002.

[64] D. Woodruff and J. Staddon, "Private Inference Control," *Proc. ACM Conf. Computer and Comm. Security (CCS '04),* pp. 188-197, 2004.

[65] S.L. Warner, "Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias," *J. Am. Statistical Assoc.,* vol. 60, no. 309, pp. 63-69, 1965.

[66] S.L. Warner, "The Linear Randomized Response Model," *J. Am. Statistical Assoc.,* vol. 66, no. 336, pp. 884-888, 1971.

[67] L. Willenborg and T. Waal, *Statistical Disclosure Control in Practice,* vol. 111, Springer-Verlag, 1996.

[68] L. Willenborg and T. Waal, *Elements of Statistical Disclosure,* vol. 155, Springer-Verlag, 2000.

[69] XACML and OASIS Security Services Technical Committee, *eXtensible Access Control Markup Language (XACML) Committee Specification 2.0,* 2005.

[70] C.T. Yu and F.Y. Chin, "A Study on the Protection of Statistical Databases," *Proc. ACM SIGMOD '77,* pp. 169-181, 1977.

[71] N. Zhang, W. Zhao, and J. Chen, "Cardinality-based Inference Control in OLAP Systems: An Information Theoretic Approach," *Proc. ACM Int'l Workshop Data Warehousing and OLAP (DOLAP '04),* pp. 59-64, 2004.

Yanjiang Yang received the BEng and MEng degrees in computer science and engineering from Nanjing University of Aeronautics and Astronautics, China, in 1995 and 1998, respectively, and the MSc degree in biomedical imaging and the PhD degree in security and cryptography from the National University of Singapore in 2001 and 2005, respectively. He is currently a research fellow in the Institute for Infocomm Research, A*STAR, Singapore. His research areas include information security and biomedical imaging. He is a member of the IEEE.

Yingjiu Li received the PhD degree in information technology from George Mason University in 2003. He is currently an assistant professor in the School of Information Systems, Singapore Management University. His research interests include applications security, privacy protection, and data rights management. He has published more than 40 technical papers in the refereed journals and conference proceedings. He is a member of the ACM and the IEEE.

Robert H. Deng received the BEng degree from the National University of Defense Technology, Changsha, China and the MSc and PhD degrees from Illinois Institute of Technology, Chicago. He has been with the Singapore Management University since 2004 and is currently a professor, an associate dean for Faculty and Research, and the director of SIS Research Center, School of Information Systems. Prior to this, he was a principal scientist and the manager of Infocomm Security Department, Institute for Infocomm Research, Singapore. He has 26 patents and more than 200 technical publications in international conference proceedings and journals in the areas of computer networks, network security, and information security. He served as a general chair, a program committee chair, and a member of numerous international conferences. He received the University Outstanding Researcher Award from the National University of Singapore in 1999 and the Lee Kuan Yew Fellow for Research Excellence from the Singapore Management University in 2006.

Feng Bao received the BS degree in mathematics and the MS degree in computer science from Peking University in 1984 and 1986, respectively, and the PhD degree in computer science from Gunma University in 1996. He is currently the principal scientist and the department head of the Cryptography and Security Department, Institute for Infocomm Research, Singapore. His research areas include algorithm, automata theory, complexity, cryptography, distributed computing, fault tolerance, and information security. He has published more than 180 international journal/conference papers and holds 16 patents.