New Paradigm of Inference Control with Trusted Computing

Yanjiang Yang, Yingjiu Li, and Robert H. Deng

School of Information Systems, Singapore Management University
80 Stamford Road, Singapore 178902
{yjyang,yjli,robertdeng}@smu.edu.sg

Abstract. The database server is a crucial bottleneck in traditional inference control architecture, as it enforces highly computation-intensive auditing for all users who query the protected database. As a result, most auditing methods, though rigorously studied, can never be implemented in practice for protecting largescale real-world database systems. To shift this paradigm, we propose a new inference control architecture that will entrust inference control to each users platform, provided that the platform is equipped with trusted computing technology. The trusted computing technology is designed to attest the state of a users platform to the database server, so as to assure the server that inference control could be enforced as expected. A generic protocol is proposed to formalize the interactions between the users platform and database server. Any existing inference control technique can work with our protocol, for which the security properties are formally proven. Since each user's platform enforces inference control for its own queries, our solution avoids the bottleneck.

Key words: Inference control, trusted computing, auditing, security protocol

1 Introduction

Inference problem The inference problem has been a long standing issue in database security that was first studied in statistical databases [2, 13] and then extended to multilevel databases and general-purpose databases [21]. The inference problem can be referred to as the concern that one can infer (sensitive) information beyond one's privileges from the data to which one is granted access. The inference problem cannot be solved by traditional access control, as the disclosure of sensitive information is not derived from unauthorized accesses but from authorized ones. The existence of various inference vulnerabilities is due to the inevitable interconnections between sensitive data that are protected from and non-sensitive data that are provided in users' accesses.

Figure 1 shows a simple example that helps to illustrate the inference problem. The employee table contains age and salary information for a group of employees. To protect individuals' salary information, the following access rule is enforced: *while the database server can answer queries about sums of salaries over multiple employees, any query about a single employee's salary is illegitimate, and thus should be denied.* With this access control enforced, however, employee A's salary can still be easily de-

NAME	AGE	SALARY
A	28	2800
B	29	3100
C	30	3200
D	31	3600
E	32	3000
F	33	3200

Fig. 1. Employee table

rived from the following legitimate queries q_1 and q_2 provided that A is the only employee whose age is 28:

q_1 : select sum(SALARY) from EMPLOYEE where AGE ≥ 28 and AGE ≤ 32

q_2 : select sum(SALARY) from EMPLOYEE where AGE ≥ 29 and AGE ≤ 32

Inference control Inference poses a serious threat to the confidentiality of database systems. Extensive research has been conducted on inference control to mitigate the threat. Inference control techniques can be classified into four categories [2]: conceptual-modeling, data perturbation, output perturbation, and query restriction. The *conceptual-modeling approach* (e.g., [8,15,17]) investigates the inference problem from a high level perspective, presenting frameworks for inference control. The proposed frameworks are sometimes too general for practical implementation. The *data perturbation approach* (e.g., [23, 29, 32, 38, 53, 56, 57]) typically replicates the original database and generates a perturbed database with noise for users to access. This approach suffers from a severe bias problem due to the noise that is added into data; therefore, it is not suitable for dynamic databases. In contrast, the *output perturbation approach* (e.g., [1, 3, 14, 22] does not add noise, 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 sub-categories: query-set-size control (e.g., [16, 44]), query-set-overlap control (e.g., [18]), auditing (e.g., [6, 9, 24]), partitioning (e.g., [7, 45, 60]), and cell suppression (e.g., [10, 36, 41]). A comparison of these methods is given in [2] from various aspects including degree of security, query processing overhead, and suitability for dynamic databases.

Auditing Auditing is an important query restriction-based approach. Traditional auditing works on the server side. The server keeps a log of all users' queries and, whenever a new query arrives on the server side, checks for possible inference vulnerabilities against the new 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 vulnerability is detected. Due to these features, auditing has triggered intensive research in database security from the 1970s [24,43] through the 1980s [4–6, 9] and 1990s [11, 34, 58] to the 21st century [27, 30, 31, 33, 54, 55, 59].

Unfortunately, auditing faces enormous difficulty in practical deployment, mainly due to the excessive computational overhead it requires to check for inference vulnera-

bilities from the accumulated query log. Audit Expert[1] [9] is a typical example. It was shown that it takes Audit Expert $\mathcal{O}(n^2)$ time to process a new SUM query [9], 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 [6]), the auditing complexity is significantly higher in more general cases. Noting that Audit Expert protects only real-valued attributes from being inferred exactly, Kleinberg et al. [27] studied the auditing of boolean attributes and proved intractable results. Li et al. [31] concluded that the problem of protecting bounded real or integer attributes from being inferred within accurate-enough intervals has much higher complexities than that of Audit Expert.

The high complexity in auditing results in low system scalability. The database server can only afford a small number of users querying simultaneously. For this reason, auditing is not deemed to be a practical inference control method for real world database systems [2].

Inference control with trusted computing To resolve the impracticality problem, we propose a new architecture for inference control (especially auditing) with trusted computing. The new architecture entrusts the enforcement of inference control to individual users' computer platforms. In this new architecture, the database server is responsible for the enforcement of traditional access control, while each user's platform is empowered to handle inference control based on their own query logs in a decentralized manner. 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.

Since the inference control is enforced on the user side, it is crucial to ensure that the enforcement is conducted exactly as expected by the database server, without any interference or manipulation. This requires that each user's platform is in a trusted state when the inference control is enforced. A typical solution to attain this is to equip each user's machine with a TCG-compliant trusted platform module (TPM) [52] 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 inference control and attest the trusted state of a user's platform to the remote database server when inference control is enforced. A generic protocol is proposed to formalize the interactions between the user's platform and the database server. Any existing inference control technique can work with our protocol, for which the security properties are formally proven.

Paper organization The rest of this paper is organized as follows. In Section 2, we propose a new architecture for shifting the inference control paradigm. In Section 3, we present a protocol to enable inference control to be executed on the users' side using standard TPM commands. In Section 4, we discuss some extensions to our solution. Finally, Section 5 concludes this paper.

[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.

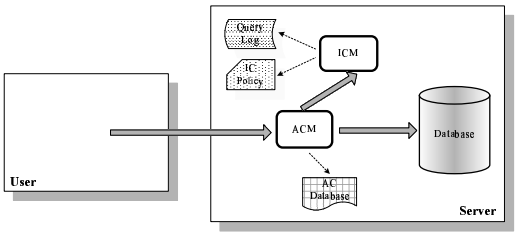
2 Architecture

Traditional architecture The traditional architecture for inference control is illustrated in Figure 2(a), where both access control and inference control are enforced at the database server side. In this architecture, the access control module (ACM) implements access control functionality, while the inference control module (ICM) executes a designated inference control 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 whether the user is a legitimate user with respect to the queried data. This can be done by checking an access control database (AC database), which contains access control rules and policy. If the user is legitimate, the database server further checks with ICM to determine whether the query will lead to any information disclosure. ICM assesses the query against the inference control (IC) policy as well as the user's past queries (collected in the query log) by executing the designated inference control algorithm. The response to the query is returned to the user only if ACM decides that the user has the proper access right and if ICM concludes that no information disclosure would occur under the inference control policy. In this architecture, the IC policy is an essential component that stipulates what is necessary for the execution of the inference control algorithm, e.g., protection attributes, objectives, and constraints. The query log is maintained by the server, which accumulates all queries asked by each user.

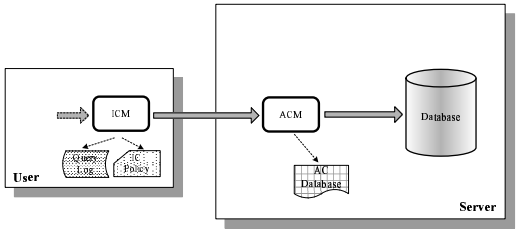
New architecture Since the enforcement of inference control is computationally intensive, it may bottleneck the database server in the traditional architecture. To solve this problem, we propose a new architecture, shown in Figure 2(b), for inference control. The basic idea is to offload the inference control function to individual users. More specifically, ICM resides at the user side instead of on the server side. ICM maintains a query log 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], to ICM. Then, ICM executes the inference control algorithm by checking the query against its query log and IC policy. ICM releases the response to the user only if the query would lead to no information disclosure under the IC policy. In this architecture, ICM on the user side acts as an extension of the database server in inference control.

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 inference control. To address this issue, certain kind of assurance must be given to the database server that ICM will be executed as expected, free of user's interference and manipulation. This kind of assurance is achieved by virtue of trusted computing. In Figure 3, a user's machine is equipped with a TCG-compliant TPM [52] and possibly other trusted hardware. A trusted platform can be built based on TPM at the hardware layer, as well as a secure kernel in the OS kernel space and ICM in the application space.

[2] The IC policy can be delivered to the user each time it is modified by the server; otherwise, it can be kept at the user's platform safely (protected by TPM).



(a) Traditional inference control architecture



(b) New inference control architecture

Fig. 2. Inference control architectures

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 protects the integrity of the components in the platform, including the secure kernel and ICM, through its integrity measuring, storing, and reporting mechanisms. More importantly, the running state of the protected platform can be conveyed to the remote 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 performs inference control as expected, free of user's interference or manipulation. 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 inference control can be considered yet another application of trusted computing [35]. Other security applications that have been rigorously studied in recent years include digital rights management [20], remote access control enforcement [39, 42], server-side user privacy protection [26], server privacy protection [48], secure auction [37], and integrity measurement [40], to name a few. The objective of these applications is to enable a server to extend its control over data distributed to client sides, or protect users' privacy on the server side, while our major

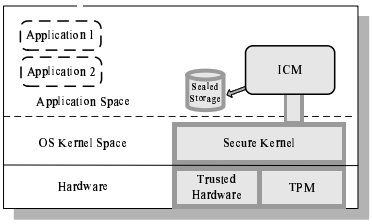


Fig. 3. TPM-enabled user host

concern is to securely decentralize the enforcement of inference control so as to resolve the efficiency and scalability problems inherent in inference control. For simplicity reasons, we assume prior knowledge about TPM. Interested readers are referred to [35] for a brief review of TPM.

Interactions between ACM and ICM In our new architecture, ACM enforces the access control mechanism over the database, and it represents the database server by interacting with all users. On the other hand, ICM is responsible for enforcing inference control 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.

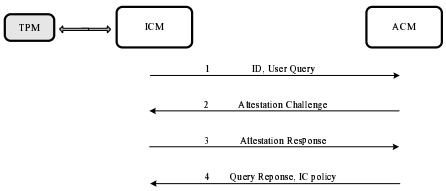


Fig. 4. Interactions between ACM and ICM

The interactions between ICM (having access to TPM) and ACM are illustrated in Figure 4. It is assumed that the user has certain identification information (e.g., user password) to identify itself to ACM. When ICM sends the user's identification information together with a user query to ACM (database server), ACM enforces access control and formulates a response to the user's 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 to ICM for the enforcement of inference control. A detailed protocol is given in the next section to formalize the interactions between ACM and ICM.

3 Protocol

In this section, we present a protocol for the interactions between ACM and ICM. The protocol enables ICM at the user side to enforce the inference control prescribed by the database server. The designing of the protocol assumes the use of version 1.2 TPM command set [51] and data structure [50].

3.1 Overview

Shifting inference control 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 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 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 [39] is that the server frequently challenges the user's platform so as to detect and thwart these types of attacks.

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 inference control. 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 log Since the enforcement of inference control depends on the query log (which is maintained by ICM in the user's host), any unauthorized modification including deletion of the query log would render inference control baseless. 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 method by associating the integrity digest of the query log with the key for protecting the confidentiality of query responses.

Authenticity of IC policy The IC policy regulates how inference control 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.

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. 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 a 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 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 this content digest 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's host may access the code and data within the ICM application domain. Though a TPM-enabled platform can be configured as a restricted system (in which only a small set of protected applications such as ICM can run) 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) [28] incorporates an additional set of hardware and software components around the TCG-compliant TPM, which provides a protected execution environment that is sufficient for our solution. Without further complicating our presentation, it is reasonable to assume in our protocol that TPM (possibly together with other trusted hardware) enables ICM to run in isolation, free from interference by 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 Figure 4, and the last step represents the enforcement of inference control over the query response and the IC policy that ICM receives from ACM in the last stage of interaction. The following notation will be used in our presentation. Let $E_{pk}(.)$ and $D_{sk}(.)$ denote the encryption operation with public key pk and the decryption operation with private key sk, respectively. Let $enc(k,.)$ and $dec(k,.)$ denote encryption and decryption with symmetric key k, respectively. Let $S_{sk}(.)$ denote a message-aware digital signature scheme with private key sk. Let $SHA1(.)$ denote SHA-1 hash function. Let $A \rightarrow B : m$ represent A sending message m to B.

Step 1 ICM $\rightarrow$ ACM: id_U, q

To issue query q, the user invokes ICM to send user identification information id_U 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 access control.

Step 2 Upon receiving 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 sub-steps.

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 access control 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. The nonce is used to thwart replay attacks in the following platform attestation.

Step 3 The platform attestation is performed in this step. Before the start of this step, ICM 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].

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. The externally supplied data is calculated from $SHA1(n_{ACM})$, and the PCRs contain the integrity measurement metrics for the protected platform including the booting procedure, the OS, and ICM.

[3] In the case that the user queries the database server for the first time, there will be two extra sub-steps prior to step 3.1 that enable ICM to generate pk_{ICM}. The two extra sub-steps 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 3.2 TPM → ICM: TPM_Certify_Info, $\sigma_{TPM} = S_{sk_{TPM}}(SHA1(pk_{ICM}) ||SHA1(n_{ACM})||im)$

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}, 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} (resp. pk_{TPM}) denotes the private (resp. public) AIK of TPM. 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 σ_{TPM} serves as not only certification of pk_{ICM}, but also platform integrity reporting of im.

Step 3.3 ICM → 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 to ICM for the enforcement of inference control.

Step 4.1 ACM: $verify(\sigma_{TPM})$

Upon receiving 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 $validate(.)$ to denote this process.

Step 4.3 ACM → ICM: $\varepsilon_1 = E_{pk_{ICM}}(k), \varepsilon_2 = enc_k(d), \sigma_{ACM} = S_{sk_{ACM}}(\varepsilon_1||\varepsilon_2||$ IC policy $||q||pk_{ICM}), pk_{ACM}$

If step 4.1 or 4.2 fails, the protocol aborts. Otherwise, ACM generates a secret key k for symmetric key encryption. It encrypts k using the public key pk_{ICM}, yielding ε_1. Then, it encrypts the query response d using k, yielding ε_2. After formulating the IC policy that is to be enforced by the user, ACM signs the IC policy, ε_1, ε_2, query q, and pk_{ICM} using its private key, yielding digital signature σ_{ACM}. Finally, ACM sends ε_1, ε_2, σ_{ACM}, and pk_{ACM} (including this public key's certificate) to ICM.

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

Step 5.1 ICM: $verify(\sigma_{ACM})$

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

Step 5.2 ICM → 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 is computed from the user's password and the digest of the query log: $SHA1$(password||digest-of-query-log). Please refer to steps 5.7 and 5.8 for the exact composition of TPM_KEY,

why the authorization data is computed in such way, and how digest-of-query-log is obtained.

Step 5.3 TPM → ICM: $k = D_{sk_{ICM}}(\varepsilon_1)$

Once TPM decides that the protected user platform is in a trusted state, and that the authorization data matches 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 to ICM.

Step 5.4 ICM: $d = dec_k(\varepsilon_2)$

ICM decrypts ε_2 using key k to get the query response d.

Step 5.5 ICM: $infcon(q_d, Q,$ IC policy)

ICM enforces inference control based on q_d, Q and the IC policy, where q_d denotes the current query q as well as its response d, and Q denotes the set of past queries as well as their responses (obtained from the query log). For reasons of generality, we assume that the query responses are used in inference control, though they are not absolutely necessary for data independent algorithms such as Audit Expert. ICM reveals d to the user if $infcon(.)$ arbitrates that q is safe, leading to no information disclosure through inference; otherwise, ICM refuses to release d and proceeds to step 5.7.

Step 5.6 ICM: $Q = Q \cup \{q_d\}$

ICM updates the query log Q by adding q_d. Note that Q remains unchanged if q_d causes inference.

Step 5.7 ICM → 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 (i) the handle of a wrapping key that can perform key wrapping, (ii) the authorization data necessary to access the wrapping key, (iii) a set of PCRs whose contents are bound to the wrapping operation, and (iv) the information about the key to be generated (e.g., key length, key algorithm, key usage).

The piece of authorization data is a 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 user's query log; that is, $SHA1$(password||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 user's query log. If the user's query log 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 → 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:

(i) TPM_Auth_Never, (ii) TPM_Auth_Always, and (iii) 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 indicate TPM to set TPM_Auth_Data_Usage to TPM_Auth_Priv_Use_Only.

3.3 Security

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 does not seem difficult to verify that our protocol meets the security requirements as posed by the security threats listed in Section 3.1. While mitigating these threats is a focus in our protocol design, the security of our protocol demands a formal analysis.

Under the assumption that the underlying TPM is perfect, our protocol can essentially be considered as an authentication protocol between ICM and ACM, aiming to satisfy the requirement that *ACM validates the security state of ICM before sending out any query response*. The security of our protocol can be formally proven using the rank function [46], a specialized theorem-proving method for establishing the correctness of authentication protocols based on communicating sequential processes (CSP) [47]. This will eliminate typical attacks such as replay attacks and masquerade attacks that are targeted at our authentication protocol.

To perform the proof, the rank function approach requires modeling the following three components: (1) the protocol, (2) the environment (attacker), and (3) the security requirements on the protocol. In particular, first, the protocol is captured as a CSP process in terms of the behavior of each system party; second, 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; third, 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. For limit of space, we omit the detailed proof.

4 Extensions

Defending against collusion attack A typical attack against inference control 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 inference control [2]. We must point out that there seems no technique can prevent a user from purposely recording his/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 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 must be changed to $SHA1$(password||digest of encrypted query log) in our protocol.

In certain cases, the database server may be able to "blacklist" some collusion groups of suspicious users who may collude using certain out-of-band information (e.g., the users from the same network domain). To further mitigate the collusion attack, inference control should be enforced based on queries from all users in a collusion group rather than from each individual user. While such control can be easily enforced with a central query log in the traditional architecture, it is not as easy to combine many users' query logs in our new architecture. A possible solution is to extend the new architecture such that the database server manages a central query log as in the traditional architecture. When any user in a collusion group issues a query, the server sends to the user's ICM the queries from all other users in the same collusion group. ICM can then enforce inference control based on the combination of its own queries and the queries it receives. To maintain the confidentiality of the query log, the queries should be sent from ACM to ICM in an encrypted form, which can be as easily done in our protocol as encrypting the query responses. Upon receiving them, ICM may keep these queries in its execution domain and delete them after use. Alternatively, ICM can add these queries to its query log such that the query log contains queries from a collusion group instead of from an individual user (this would substantially decrease the number of queries sent each time by the database server).

User using multiple hosts Our protocol is essentially designed for the scenario in which a user is bound to a single host. This can be seen that each user's query log is maintained at the user's host and the user's host accumulates the queries issued from that particular host. If a user is allowed to use multiple hosts to query the database, the query logs maintained at different hosts may not be readily available to the current host where inference control is enforced. A convenient solution is to let the database server maintain a central query log that collects user queries at the discretion of user identity in conjunction with host identity[4]. When a user issues some queries from a host, the queries previously issued by the user from all other hosts are passed by ACM to the current user host for the enforcement of inference control.

Database update Database update (e.g., deletion of some records) may necessitate updating the user's query log on the user's side. To facilitate updating, the database server may manage a central query log as discussed above. 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 inference control will be enforced upon the latest query log.

[4] The TPM bound to a user's host can be used to unambiguously identify the host.

5 Conclusions

This paper proposed a new inference control architecture and an inference control protocol upon it. While traditional inference control is enforced by a database server for all its users, our solution entrusts each user's host to enforce inference control for itself, provided that the user's host is equipped with trusted computing technology. By decentralizing the highly computation-intensive task of inference control, 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 inference control configuration can only support a small number of users due to the bottleneck of enforcing inference control for all users on the server side. In this sense, our solution removes the crucial impediment in traditional inference control configuration and identifies a new paradigm for the practical implementation of inference control.

Our solution relies on the adoption of widely available trusted computing technology, which is envisioned to be ubiquitous in several years. This trusted computing technology is utilized by the database server to attest users' platforms so that the inference control can be enforced on the user side as expected by the database server. The security properties of our solution are formally proven with the rank function approach. Our solution can work with any existing inference control technique; even a hybrid system of mixing our solution (for some users whose platforms are TPM equipped) with traditional inference control (for those users who may not implement trusted computing) can be easily set up. An interesting future direction is to implement our solution with various existing inference control techniques in some real world settings.

References

1. J. O. Achugbue, F. Y. Chin. *The Effectiveness of Output Modification by Rounding for Protection of Statistical Databases*, INFOR, Vol. 17(3), pp. 209-218, 1979.
2. N. R. Adam, J. C. Wortmann. *Security-Control Methods for Statistical Databases: A Comparative Study*, ACM Computing Surveys, Vol. 21(4), pp. 516-556, 1989.
3. L. L. Beck. *A Security Mechanism for Statistical Databases*, ACM Trans. Database Systems, Vol. 5(3), pp. 5(3), pp. 316-338, 1980.
4. M. Chen, L. McNamee, M. A. Melkanoff. *A Model of Summary Data and Its Applications to Statistical Databases*, Proc. 4th International Conference on Statistical and Scientific Database Management, LNCS 339, pp. 354-372, 1989.
5. F. Y. Chin. *Security Problems on Inference Control for SUM, MAX, and MIN queries*, J. ACM, Vol. (33), pp. 451-464, 1986.
6. F. Y. Chin, P. Kossowski, S. C. Loh. *Efficient Inference Control for Range Sum Queries*, Theor. Comput. Sci., Vol. 32, pp. 77-86, 1984.
7. F. Y. Chin, G. Özsoyoglu. *Security in Partitioned Dynamic Statistical Databases*, Proc. IEEE COMPSAC, pp. 594-601, 1979.
8. F. Y. Chin, G. Özsoyoglu. *Statistical Database Design*, ACM Trans. Dababase Systems, Vol. 6(1), pp. 113-139, 1981.
9. F. Y. Chin, G. Özsoyoglu. *Auditing and Inference Control in Statistical Databases*, IEEE Trans. Softw. Eng. Vol. 6, pp. 574-582, 1982.
10. L. H. Cox. *Suppression Methodology and Statistical Disclosure Control*, J. Am. Stat. Assoc. Vol. 75(370), pp. 377-385, 1980.

11. L. H. Cox, L. V. Zayatz. *An Agenda for Research on Statistical Disclosure Limitation*, J. Official Statistics, Vol. 75, pp. 205-220, 1995.
12. R. Delicata. *An Analysis of Two Protocols for Conditional Access in Mobile Systems*, Technical Report CS-04-13, Department of Computing, University of Surrey, 2005.
13. D. E. Denning. *Cryptography and Data Security*, Addison Wesley, 1982.
14. D. E. Denning. *Secure Statistical Databases with Random Sample Queries*, ACM Trans. Database Systems, Vol. 5(3), pp. 88 - 102, 1980.
15. D. E. Denning. *A Security Model for the Statistical Database Problem*, Proc. 2nd International Workshop on Management, pp. 1 - 16, 1983.
16. D. E. Denning, P. J. Denning, M. D. Schwartz. *The Tracker: A threat to Statistical Database Security*, ACM Trans. Database Systems, Vol. 4(1), pp. 76 - 96, 1979.
17. D. E. Denning, J. Schlörer. *Inference Control for Statistical Databases*, Computer, Vol. 16(7), pp. 69-82, 1983.
18. D. Dobkin, A. K. Jones, R. J. Lipton. *Secure Databases: Protection Against User Influence*, ACM Trans. Database Systems, Vol. 4(1), pp. 97-106, 1979.
19. D. Dolve, A. C. Yao. *On the Security of Public Key Protocols*, IEEE Transactions on Information Technology, Vol. 29(2), pp. 198-208, 1983.
20. J. S. Erickson. *Fair use, DRM, and trusted computing*, Communications of ACM, Vol. 46(4), pp. 34-39, 2003.
21. C. Farkas, S. Jajodia. *The Inference Problem: A Survey*, SIGKDD Explorations, 4(2). pp. 6 - 11, 2002.
22. I. P. Fellegi, J. L. Phillips. *Statistical Confidentiality: Some Theory and Applications to Data Dissemination*, Ann. Ec. Soc. Meas., Vol. 3(2), pp. 399-409, 1974.
23. B. G. Greenberg, J. R. Abernathy, D. G. Horvitz. *Application of Randomized Response Technique in Obtaining Quantitative Data*, Proc. Social Statistics Section, America, Statistical Association, pp. 40-43, 1969.
24. L. J. Hoffman. *Modern Methods for Computer Security and Privacy*, Prentice-Hall, 1977.
25. M. L. Hui, G. Lowe. *Safe Simplifying Transformations for Security Protocols*, Proc. 12th Computer Security Foundations Workshop, pp. 32-43, 1999.
26. A. Iliev, S. W. Smith. *Protecting User Privacy via Trusted Computing at the Server*, IEEE Security and Privacy, 3(2): 20-28. March/April 2005.
27. J. Kleinberg, C. Papadimitriou, P. Raghavan. *Auditing Boolean Attributes*, Proc. 9th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 86-91, 2000.
28. LaGrande technology architecture. Intel Developer Forum, 2003.
29. D. Lefons, A. Silvestri, F. Tangorra. *An Analytic Approach to Statistical Databases*, Proc. 9th Very Large Databases, pp. 260-273, 1983.
30. Y. Li, H. Lu, R. H. Deng. *Practical Inference Control for Data*, Proc. IEEE Symposium on Security and Privacy, pp.115-120, 2006.
31. Y. Li, L. Wang, X. S. Wang, S. Jajodia. *Auditing Interval-based Inference*, Proc. 14th Conference on Advanced Information Systems Engineering, pp. 553-567, 2002.
32. C. K. Liew, W. J. Choi, C. J. Liew. *A Data Distortion by Probability Distribution*, ACM Trans. Database Systems, Vol. 10(3), pp. 395-411, 1985.
33. F. M. Malvestuto, M. Mezzini. *Auditing Sum-Queries*, Proc. International Conference on Database Theory, pp. 504-509, 2003.
34. F. M. Malvestuto, M. Moscarini. *An Audit Expert for Large Statistical Databases*, Statistical Data Protection, EUROSTAT, pp. 29-43, 1999.
35. C. Mitchell. *Trusted Computing*, The Institution of Electrical Engineers, London, UK, 2005.
36. G. Özsoyoglu, J. Chung. *Information Loss in the Lattice Model of Summary Tables Due To Suppression*, Proc. IEEE Symposium on Security and Privacy, pp. 75 - 83, 1986.

37. A. Perrig, S. W. Smith, D. Song, J. D. Tygar. *SAM: A Flexible and Secure Auction Architecture using Tusted Hardware*, eJETA.org: The Electronic Journal for E-Commerce Tools and Applications. 1(1). 2002.
38. J. P. Reiss. *Practical Data Swapping: The First Step*, Proc. IEEE Symposium on Security and Privacy, pp. 36-44, 1980.
39. R. Sailer, T. Jaeger, X. Zhang, L. van Doorn. *Attestation-Based Policy Enforcement for Remote Access*, Proc. ACM Conference on Computer and Communications Security, pp. 308-317, 2004.
40. R. Sailer, X. Zhang, T. Jaeger, L. van Doorn. *Design and Implementation of a TCG-based Integrity Measurement Architecture*, USENIX Security Symposium, pp. 223-238, USENIX, 2004.
41. G. Sande. *Automated Cell Supperssion to Reserve Confidentiality of Business Statistics*, Proc. 2nd Workshop on Statistical Database Management, pp. 346-353, 1983.
42. R. Sandhu, X. Zhang. *Peer-to-Peer Access Control Architecture Using Trusted Computing Technology*, Proc. ACM Symposium on Access Control Models and Technologies, pp. 147-158, 2005.
43. J. Schlörer. *Confidentiality of Statistical Records: A Threat Monitoring Scheme of On-line Dialogue*, Methods Inform. Med., Vol. 15(1), pp. 36-42, 1976.
44. J. Schlörer. *Disclosure from Statistical Databases: Quantitative Aspects of Trackers*, ACM Trans. Database Systems, Vol 5(4), pp. 467-492, 1980.
45. J. Schlörer. *Information Loss in Partitioned Statistical Databases*, Comput. J. Vol. 26(3), pp. 218-223, 1983.
46. S. Schneider. *Verifying Authentication Protocols with CSP*, Proc. 10th Computer Security Foundation Workshop, pp. 3-17, 1997.
47. S. Schneider. *Concurrent and Real-time Systems: the CSP Approach*, Addison-Wesley, 1999.
48. S. W. Smith, D. Safford. *Practical Server Privacy Using Secure Coprocessors*, IBM Systems Journal (special issue on End-to-End Security) 40:683-695. 2001.
49. TCG. TPM Main, Part 1 Design Principles, TCG Specification Ver. 1.2, Revision 62, www.trustedcomputinggroup.org, 2003.
50. TCG. TPM Main, Part 2 TPM Data Structure, TCG Specification Ver. 1.2, Revision 62, www.trustedcomputinggroup.org, 2003.
51. TCG. TPM Main, Part 3 Commands, TCG Specification Ver. 1.2, Revision 62, www.trustedcomputinggroup.org, 2003.
52. Trusted Computing Group. www.trustedcomputinggroup.org. 2006.
53. J. F. Traub, Y. Yemini, H. Wozniakowski. *The Statistical Security of A Statistical Database*, ACM Trans. Database Systems, Vol. 9(4), pp. 672-679, 1984.
54. L. Wang, Y. Li, D. Wijesekera, S. Jajodia. *Precisely Answering Multi-dimensional Range Queries without Privacy Breaches*, Proc. Europ. Symp. Computer Security, ESORICS, pp. 100-115, 2003.
55. L. Wang, D. Wijesekera, S. Jajodia. *Cardinality-based Inference Control in Sum-only Data Cubes*, Proc. Europ. Symp. Computer Security, ERORICS, LNCS 2502, pp. 55-71, 2002.
56. S. L. Warner. *Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias*, J. Am. Stat. Asso. Vol. 60(309), pp. 63-69, 1965.
57. S. L. Warner. *The Linear Randomized Response Model*, J. Am. Stat. Asso., Vol. 66(336), pp. 884-888, 1971.
58. L. Willenborg, T. Waal. *Statistical Discolure Control in Practice*, Lecture Notes in Statistics, Vol. 111, Springer-Verlag, 1996.
59. L. Willenborg, T. Waal. *Elements of Statistical Discolure*, Lecture Notes in Statistics, Vol. 155, Springer-Verlag, 2000.
60. C. T. Yu, F. Y. Chin. *A Study on the Protection of Statistical Databases*, Proc. ACM SIGMOD, pp. 169-181, 1977.