9-2006

Three Architectures for Trusted Data Dissemination in Edge Computing

Shen-Tat GOH
Institute for Infocomm Research

Hwee Hwa PANG
Singapore Management University, hhpang@smu.edu.sg

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

Feng BAO
Singapore Management University, fbao@smu.edu.sg

DOI: https://doi.org/10.1016/j.datak.2005.05.003

Citation

GOH, Shen-Tat; PANG, Hwee Hwa; DENG, Robert H.; and BAO, Feng. Three Architectures for Trusted Data Dissemination in Edge Computing. (2006). *Data and Knowledge Engineering*. 58, (3), 381-409. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/1202

Three architectures for trusted data dissemination in edge computing

Shen-Tat Goh [a,*], HweeHwa Pang [a], Robert H. Deng [b], Feng Bao [a]

[a] *Institute for Infocomm Research, 21 Heng Mui Keng Terrace, Singapore 119613, Singapore*
[b] *Singapore Management University, 469 Bukit Timah Road, Singapore 259756, Singapore*

Abstract

Edge computing pushes application logic and the underlying data to the edge of the network, with the aim of improving availability and scalability. As the edge servers are not necessarily secure, there must be provisions for users to validate the results—that values in the result tuples are not tampered with, that no qualifying data are left out, that no spurious tuples are introduced, and that a query result is not actually the output from a different query. This paper aims to address the challenges of ensuring data integrity in edge computing. We study three schemes that enable users to check the correctness of query results produced by the edge servers. Two of the schemes are our original contributions, while the third is an adaptation of existing work. Our study shows that each scheme offers different security features, and imposes different demands on the edge servers, user machines, and interconnecting network. In other words, all three schemes are useful for different application requirements and resource configurations.

Keywords: Data integrity; Edge computing; Data dissemination; Replication

* Corresponding author. Tel.: +65 6874 8457; fax: +65 6776 8109.
E-mail addresses: stgoh@i2r.a-star.edu.sg (S.-T. Goh), hhpang@i2r.a-star.edu.sg (H. Pang), robertdeng@smu.edu.sg (R.H. Deng), baofeng@i2r.a-star.edu.sg (F. Bao).

1. Introduction

Many Web services are served from central locations, and could suffer from a number of bottlenecks ranging from Web and database server loads, to network delays. Server overloads can usually be alleviated through load balancing on a server farm. In contrast, network latencies are usually beyond the control of the Web service operators, as traffic to and from remote users has to pass through long-haul networks operated by multiple network providers that are often congested. Although aggressive build-up in recent years by telecommunication companies has expanded the capacity of the long-haul networks, new technologies like Gigabit ethernet are making bandwidth much more affordable in the Metropolitan Area Networks (MAN). Given the relative price-performance of Wide Area Networks (WAN) versus MAN, the logical approach to reduce network latency is to push the Web services to the users, into the MANs.

Edge computing is being promoted as a strategy to achieve scalable and highly available Web services (e.g., [1]). Fig. 1 shows the generic architecture of an edge computing platform. It pushes business logic and data processing from central data centers, out to proxy servers at the ‘‘edge’’ of the network and within the MANs. There are several potential advantages: Running applications at the edge cuts down network latency and produces faster responses to user applications and partners’ Web services. Adding edge servers near user clusters is also likely to be a cheaper way to achieve scalability than fortifying the servers in the central data center and provisioning more network bandwidth for every user. Finally, by lowering the dependency on a central data center, edge computing removes the single point of failure in the infrastructure, reducing its susceptibility to denial of service attacks and improving service availability.

In theory, edge computing is a natural extension of the Content Delivery Network (CDN) architecture [2,3]. In practice, pushing application logic to edge servers introduces a number of

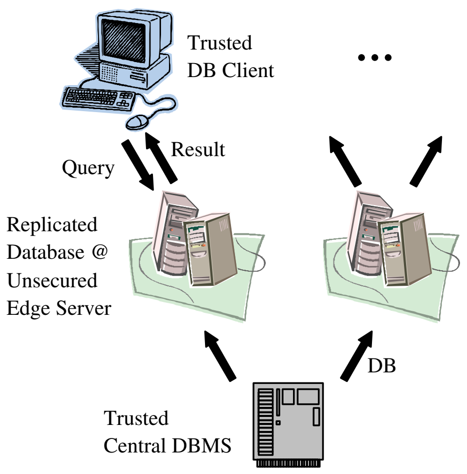

Fig. 1. Edge computing set-up.

technical challenges, one of which is data security: For applications that run on a database, edge computing entails the replication of (parts of) the database, to edge servers that perform query processing on behalf of the central DBMS. Since the edge servers are not necessarily as secure as the central data center, there must be provisions to check the integrity of query results produced by the edge servers; specifically,

- Authenticity—The query results originate from a server or cluster authorized by the master DBMS, and have not been tampered with.
- Accuracy—No spurious tuples are introduced and no qualifying data are left out. Moreover, no user is intentionally given the output from a different query.

To illustrate, a financial information provider could push historical stock prices, together with analytics software, to edge servers operated by partner ISPs (Internet Service Provider). Such an arrangement enables users to run different pricing and risk models off the edge servers directly instead of depending on a central data center that might be situated thousands of miles away, thus achieving superior responses by reducing communication latency and processing bottlenecks. To keep out unauthorized users, the data and analytics software could be protected via encryption and access control. At the same time, legitimate users who act on the results produced by the analytics software for investment decisions or recommendations would also demand assurance of the integrity of the results.

Contributions: This paper aims to address the challenges of ensuring the integrity of relational database query results generated by edge servers in an edge computing platform. We introduce two novel schemes where the master DBMS can empower a group of edge servers to collectively certify query results, on the premise that (if necessary) the edge servers can be running different operating systems and protected by different security products (i.e., firewall, intrusion detection, etc.), thus increasing the difficulty for attackers to breach all the edge servers concurrently without being detected. One of the schemes requires the edge servers to execute each query and affix partial signatures separately; the partial signatures are then combined into a final signature for the query result. In the other scheme, only one edge server needs to process each query; while producing the query result, the edge server also compiles a set of digests from a pool that has been pre-generated by the master DBMS, such that the assembled digests are adequate for a group of edge servers to check and certify the query result.

To profile the security properties and resource requirements of the proposed schemes, we compare them against a third scheme that is adapted from existing work. Our study shows that the three schemes present different security and resource tradeoffs, and are useful for different application scenarios and resource configurations.

The remainder of this paper is organized as follows: Section 2 introduces our target deployment model and the associated security threats, in addition to related work on data authentication, with particular emphasis on authentication in database systems. Our proposed authentication mechanisms are presented in Section 3. Following that, Section 4 analyzes the relative cost-performance characteristics of the alternative mechanisms. Finally, Section 5 concludes the paper and discusses avenues for future work.

2. Background

This section begins by defining the target system deployment model, and the associated security threats. Following that, we define a couple of cryptographic primitives that will be used in our authentication mechanisms, before summarizing related work.

2.1. System model

Fig. 1 shows the set-up of a generic edge computing environment. The central server that hosts the master database is located within a professionally managed data center. Thus we assume that the central server is secure and always available, and that attacks and other risks to the central server are beyond the scope of this work. The database (or parts of it) is replicated to servers situated at the edge of the network, i.e., edge servers, near the users. Updates to the replica can be achieved using techniques such as those studied in [4,5]. The users and their application clients can be (though not necessarily are) firewalled within a trusted LAN and hence protected from external attackers. Database/SQL queries issued by applications are processed at the edge servers; the result for each query is returned together with a verification object or signature that the client can use to verify the integrity of the query result. Finally, the edge servers could be operated by third-parties or they could even reside in a peer-to-peer platform, so it is possible for a hacker to tamper with the data on the edge servers.

Given that the edge servers are not secured, one security concern is controlling access to the data. Obviously, an adversary who gains access to the operating system or hardware of the storage server may be able to browse the stored data, or make illegal copies of the data. Solutions to mitigate this concern include encryption (e.g., [6]) and steganographic storage (e.g., [7]), and are complementary to the authentication schemes that we propose here.

Our primary concern in this paper is the threat to the integrity of query results that arises from the exposed edge servers. For example, an adversary who is cognizant of the data organization in the storage server may attempt to make *logical alterations* to the data; an example is to illegally effect fund transfers between two accounts. Even if the data organization is hidden, for example through data encryption or steganographic schemes [8,7], the adversary may still sabotage the database by overwriting physical pages within the storage volume. Therefore it is essential to provide mechanisms for users to verify the integrity of the query results returned.

2.2. Cryptographic primitives

Our proposed solutions to achieve trusted data dissemination in the face of the threats described above, and also many of the related work, are based on the following two cryptographic primitives:

Hash function: A *one-way* hash function, denoted as $h(\cdot)$, works in one direction: it is easy to compute a hash value $h(m)$ from a pre-image m; however, it is hard to find a pre-image that hashes to a particular hash value. A hash function is collision-resistant if it is hard to find two different pre-images m_1 and m_2 such that $h(m_1) = h(m_2)$. SHA-256 [9] is an example of a one-way, collision-resistant hash function. We will use the terms hash, hash value and message digest interchangeably.

Digital signature: A digital signature algorithm is a cryptographic tool for authenticating the integrity of the signed message as well as its origin. In the algorithm, a signer keeps a private key secret and publishes the corresponding public key. The private key is used by the signer to generate digital signatures on messages, while the public key is used by anyone to verify the signatures on messages. RSA [10] and DSA [11] are two commonly used signature algorithms.

(k, n) Threshold signature scheme: The (k,n) threshold RSA signature scheme as proposed in [12] is defined as follows:

- *Key share generation*: During system set-up, a dealer generates a RSA public key pk along with the corresponding private key share sk_i and public verification key vk_i for party i, $i = 1, 2, \ldots, n$. In practice, the dealer may be a computer or a tamper-resistant device trusted by all parties. The dealer distributes the private key shares as well as the verification keys among the parties, and subsequently erases its copy of the private key shares.
- *Signature share generation*: Given a message m, party i, $i = 1, 2, \ldots, n$, generates a signature share $\sigma_i(m)$ using its private key share sk_i, along with a "proof-of-correctness" ρ_i of the signature share based on its verification key vk_i.
- *Signature share combination and signature verification*: Given the message m, any k signature shares $\sigma_j(m)$ and their proof-of-correctness ρ_j, for $j = 1, 2, \ldots, k$, anyone can verify the correctness of the individual signature shares. If the k shares are all verified to be correct, they are combined to form the final signature $\sigma_C(m)$, which is a standard RSA signature and can be verified by the RSA public key pk.

Delegation-by-certificate proxy signature scheme: Let (pk_O, sk_O) and (pk_P, sk_P) denote the public and private key pairs of party O and party P under standard signature schemes S_O and S_P, respectively. The delegation-by-certificate proxy signature scheme is defined as follows [13]:

- *Delegation*: In order to designate P as its proxy signer, the original signer O issues a delegation certificate to P:

  $$d\_cert = (f_1|pk_P|\omega|\sigma_O)$$

  where ω is a "warrant" specifying restrictions (e.g., validity period) on the messages that the proxy signer is allowed to sign, f_1 is a flag used to signal that d_cert is a delegation certificate, and σ_O is O's signature on $f_1|pk_P|\omega$ using its private key sk_O under the signature scheme S_O. Verification of this signature is then performed by first prepending f_1 to $pk_P|\omega$.
- *Proxy signature generation*: To sign a message m on behalf of O, P computes a signature $\sigma_P(f_2|pk_O|m)$ on $f_2|pk_O|m$ using its private key sk_P under the standard signature scheme S_P, where f_2 is a flag indicating that the signature is a proxy signature. We call the combination of d_cert and $\sigma_P(f_2|pk_O|m)$ the proxy signature.
- *Proxy signature verification*: Given m, d_cert, $\sigma_P(f_2|pk_O|m)$ and authentic copies of pk_O and pk_P, a verifier first checks the validity of d_cert relative to pk_O, and then the validity of the proxy signature $\sigma_P(f_2|pk_O|m)$ on the message m relative to pk_P. Only when both verifications succeed, will the verifier consider the message m as having been authenticated/signed by O.

2.3. Related work

This paper builds upon the work by Merkle in [14]. We shall explain the Merkle hash tree with the example in Fig. 2, which is intended for authenticating data values $d_1, \ldots, d_4$. Each leaf node N_i is assigned a digest $h(d_i)$, where h is a one-way hash function. The value of each internal node is derived from its child nodes, e.g., $N_{12} = h(N_1|N_2)$ where | denotes concatenation. In addition, the value of the root node is signed. The tree can be used to authenticate any subset of the data values, in conjunction with a verification object (VO). For example, to authenticate d_1, the VO contains N_2, N_{34} and the signed N_{1234}. The recipient first computes $h(d_1)$ and $h(h(h(d_1)|N_2)|N_{34})$, then checks if the latter is the same as the signed N_{1234}. If so, d_1 is accepted; otherwise, d_1 has been tampered with.

Merkle hash trees have inspired many proposals on data authentication, including certifying selective retrievals of XML documents [15,16], for proving the presence or absence of public key certificates on revocation lists [17,18], and for authenticating JPEG2000 images [19]. The proposal that is most closely related to our work is [20], which describes a scheme for verifying query results produced by untrusted third-party publishers. The scheme calls for the data owner to periodically distribute signed digests directly to users. The digests are hashes computed recursively over tree indices on the owner's database. To prove that the answer to a query is correct, the publisher constructs a VO using the same tree index that the owner used to compute the signed digest. The VO provides a hard-to-forge proof that links the answer to the signed digest.

This work by Devenbu et al. [20] is among the first few papers to address the authentication of query results in database systems. However, when applied directly in our context of edge computing, their scheme poses a number of limitations that we aim to overcome: First, a Merkle tree is needed for each of the 2^a sort-orders on a table with a attributes; this incurs large storage overheads in the database implementation, and makes updates very expensive. Second, a VO needs to contain links all the way to the digest for the root of the tree index. This means that the VO grows linearly to the query result and logarithmically to the base table, which can be quite sizable for large databases. Another potential problem is that projections have to be performed by the clients, which leads to wasteful data transfers especially if the filtered attributes are BLOBs. Finally, the approach is weak in terms of access control—even attributes that are supposed to be filtered out through projection must be offered to users for verification. Moreover, to check for completeness, tuples beyond the left and right boundaries of the query result must be exposed to the user; this would undermine any tuple-based access control on the database.

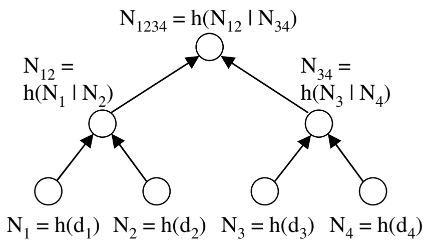

Fig. 2. Example of a Merkle hash tree.

A more recent work by Roos et al. [21] also employs the Merkle hash tree to authenticate range queries. However, the focus of that paper is on encoding the VO in a more compact form to minimize communication overhead; it has the same limitations as the scheme in [20].

Finally, in [22], Barbara et al. proposed a different mechanism for detecting unauthorized changes to a database. The mechanism derives a checksum for every record, followed by an overall database checksum from the record checksums. As long as the database checksum is secured, the database cannot be tampered with without being detected. This mechanism is designed to support data authentication by the DBMS itself, not by the recipient of query results. Hence it is not directly applicable here.

3. Trusted data dissemination networks

In this section, we present three data dissemination solutions that enable query results produced by edge servers to be checked by users for authenticity and accuracy (as defined in Section 1).

The first solution requires no security guarantees from the edge servers. The second mechanism recognizes that while no edge server by itself is secure enough, some minimum number of edge servers can be trusted to process a query, then collectively certify the result. This is based on the observation that if necessary the edge servers can be running different operating systems and protected by different security products, thus increasing the difficulty for attackers to compromise all the edge servers concurrently without being detected. The third scheme combines the first two, in such a way that a single edge server produces the query results and forwards them to a cluster of verifiers for certification.

Of the three alternatives, the first is adapted from [20]: Instead of building a separate Merkle tree, we integrate it with the B+-tree to avoid incurring extra I/Os in retrieving and searching separate index structures. To make the scheme more practical, we also change the verification procedure to eliminate the need to maintain a Merkle tree for every sort order on a table. The other two schemes are our original contributions.

3.1. Untrusted edge processor (UEP)

The first scheme, Untrusted Edge Processor (UEP), imposes no security requirement on the edge servers. Instead, the master DBMS distributes to the edge servers a hierarchy of digests with each database table, in which the root digest is signed with the private key of the master DBMS using a digital signature scheme. Based on these digests, an edge server generates an appropriate verification object (VO) to accompany each query result. The recipient then check the integrity of the query result, by deriving from it and the VO a digest to match against the signed root digest. Without the master DBMS' private key, it is computationally infeasible for the edge server to introduce spurious tuples or to alter the result while still producing a VO that matches the signed root digest. UEP is representative of the various Merkle hash tree-based solutions, like [20,23,24], etc.

The UEP scheme is depicted in Fig. 3. As shown in the figure, the master DBMS distributes the database and certified B-trees (CB-tree) (adapted from [24]) to the edge servers. The CB-tree is essentially a combination of the B+-tree and the Merkle tree [14]. Specifically, each child pointer

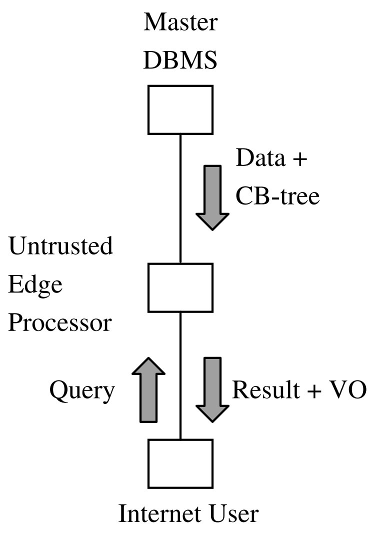

Fig. 3. Untrusted edge processor.

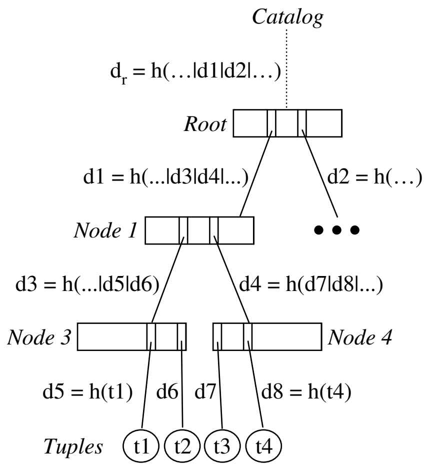

Fig. 4. Certified B-tree.

in the tree index is augmented with a digest of all the tuple values in the underlying subtree, as shown in Fig. 4. The root digest, d_r, is certified with the master DBMS' private key. Details of the construction of the CB-tree are as follows:

- For each tuple t in a database table, compute digest d_t:

$$d_t = h(\text{DBname}|\text{tablename}|\text{recordvalue})$$

where h is a one-way hash function on the concatenation of the database name, the table name, and the value of the tuple. d_t is stored with the corresponding tuple pointer in the leaf node of the B+-tree, as in nodes 3 and 4 in Fig. 4.

- For each leaf node N, a digest d_N is derived from the tuple digests within N, i.e.,

  $$d_N = h(d_{t1}|\cdots|d_{ti}|\cdots)$$

  d_N is stored with the corresponding child pointer in N's parent. An example is node 1 in Fig. 4.
- Similarly, for each internal node N, a digest d_N is computed from the digests d_{Ni}'s associated with the node pointers within N:

  $$d_N = h(d_{N1}|\cdots|d_{Ni}|\cdots)$$

  Again, d_N in turn is stored with the corresponding child pointer in N's parent.
- Finally, a root digest d_r is computed for the root node, and signed with the master DBMS' private key so that it cannot be altered by unauthorized parties.

Whenever a user query arrives, the edge server performs query evaluation as in a conventional DBMS. Each table is accessed through its CB-tree, and the digests that are needed for authentication are collected into the VO while the edge server traverses down the CB-tree. Starting from the root node, the digests for all the branches other than the child node in the query path are included in the VO. This continues until the edge server reaches the smallest subtree that covers all the result tuples, also known as the enveloping subtree. For each node along both boundaries of the enveloping subtree, the digests for all the branches that do not lead to any result tuple are added to the VO. If the result tuples do not occupy one contiguous key range, for example in the case of a multi-point query, the "gaps" in between the result tuples also have to be accounted for. This is done by inserting into the VO, for each such gap, the digest for the top node of the largest subtree that covers the gap but not any result tuple. (By catering for such gaps in the key range, we eliminate the need to maintain a CB-tree for every sort order on a table as in [20].)

To illustrate, suppose the query result comprises tuples t1, t2, t3 and t4 in Fig. 4. The VO would include:

- digests for all the tuples under node 3 other than t1 and t2, so that the verifier can compute d3;
- digests for all the tuples under node 4 other than t3 and t4, so that the verifier can compute d4;
- digests for all the nodes under node 1, other than nodes 3 and 4, so that the verifier can compute d1;
- digests for all the nodes under the root, other than node 1, so that the verifier can compute d_r; and
- the certified d_r for the verifier to match against its computed d_r.

Moreover, the tree structure is also recorded in the VO to enable the verifier to combine the digests in the correct order. Since the VO contains digests for the nodes along the path from the root to the leaf node(s), the size of the VO grows logarithmically to the table size [24].

Unlike tuples that are filtered out by selection operations, attributes that are excluded through projection operations cannot simply be dropped from the result. This is because all attribute values in the selected tuples are needed to compute the tuple digests during verification. Therefore, for each attribute of a selected tuple, either the data value or its digest must be returned as part of the VO, depending on the relative size of the attribute versus the digest.

In the case of a join operation, the edge server will not return the final join result. Instead, for each table targeted by the join, the edge server sends back the qualifying tuples with an accompanying VO. This allows the user client to verify each contributing sub-table, before performing the final join operation(s) on the sub-tables.

Security features: Suppose an adversary tampers with the database by altering the existing tuples. There are conceptually two ways that he can attempt to avoid detection:

- He can try to ensure that every altered tuple still produces the same digest. By the definition of one-way hash function, it is computationally infeasible to determine the input argument that would lead to a given digest, hence this is not a feasible option.
- He can modify the digests leading from each tampered tuple, all the way up to the root node. However, the digest of the root node is signed with a digital signature, and without the private key of the DBMS the adversary is not able to produce another valid, signed root digest.

Similarly, inserting or deleting a tuple would trigger a change to a leaf node, and possibly more changes up the CB-tree if there are node splits or merges. These changes cannot escape detection as the adversary is not able to compensate for the mismatch between the new node content and its previous digest.

Therefore, we conclude that the UEP scheme is effective in detecting any spurious tuples or tampered data values introduced by a compromised edge server.

Another desirable data authentication feature would have been to ascertain that no qualifying tuples are left out of a query answer. To achieve that, all the result tuples must occupy one contiguous range under a CB-tree (see [20] for detailed explanation). This requires a CB-tree to be created on every attribute that may be queried, which entails significant space and maintenance overheads. Moreover, for a query that involves selections on multiple attributes in a table, only one selection can be carried out by the edge server; the remaining selections have to be pushed to the recipient. (Constructing a CB-tree with a search key that concatenates those attributes does not work, because the qualifying tuples may not occupy one contiguous range under such a CB-tree.) More importantly, boundary tuples (tuples to the immediate left and right of the key range) must be offered to the user for inspection. This would undermine any tuple-based access control. With these limitations, we decided that it is impractical to build into UEP the ability to verify that all qualifying tuples are included in the query answers.

Finally, UEP is vulnerable to query result substitution, where a compromised edge processor passes off output from a different query as result for a user query.

3.2. Trusted cluster processors (TCP)

While the edge servers are not as secure as the master DBMS and hence may be compromised individually, in practice it is less likely for an intruder to gain control of several edge servers simultaneously without being detected. The reason is that attacks typically exploit weaknesses in particular system implementations, so they are not likely to succeed against all the edge servers if they are installed with different operating systems and protected by different security products (i.e., firewall, intrusion detection, etc.). Although in practice there are limited choices of different operating systems and proven security products, there are still enough combinations to set up small clusters of, say, a dozen edge servers each, that offer sufficient robustness and availability.

Based on this observation, the trusted cluster processors (TCP) solution permits a pre-specified minimum number of edge servers to independently process a query and collectively endorse the authenticity of the result on behalf of the master DBMS. This is achieved using a novel combination of a (k,n) threshold signature scheme and a proxy signature scheme.

A (k,n) threshold signature scheme is a cryptographic tool that allows any subset of k out of n parties to generate a signature on a message, but prevents the creation of a valid signature by fewer than k parties. Since it was first proposed in [25], threshold signature has been studied extensively. However, in most of the schemes, either the signature share generation or verification is interactive, thus requiring a synchronous communication network, or the size of each individual signature share grows linearly in the number of parties. In this paper, we employ the (k,n) threshold RSA signature scheme proposed in [12] as it enjoys the following desirable properties:

T1 *Unforgeability*—Any subset of k out of n parties can generate a signature, but fewer than k parties cannot generate a valid signature.
T2 *Robustness*—Invalid signature shares from corrupted parties can be detected so as to prevent them from disrupting signature generation by uncorrupted parties.
T3 *Non-interactivity*—Signature share generation and verification can be completely non-interactive.
T4 *Compact share size*—The size of an individual signature share is bounded by a small constant times the size of the RSA modulus.
T5 The resulting signature is a standard RSA signature.

T1 and T2 are proved in the random oracle model [12], T3 and T4 are essential in maintaining good system performance in practical applications, and T5 allows any client to verify the signature as long as it supports the standard RSA signature scheme. The definition of (k,n) threshold RSA signature scheme is given in Section 2.2.

A straightforward implementation of TCP is for the master DBMS to hand over its private key to each cluster, and have the n processors within the cluster share the private key based on the (k,n) threshold RSA scheme. Any k processors in the cluster can then collectively produce the master DBMS's signature. However, this approach is not a prudent security practice since it gives the cluster unlimited signing power. We overcome the problem by employing a proxy signature scheme that permits the master DBMS to delegate its signing rights to the cluster in a tightly controlled manner.

Since its introduction in [26], many proxy signature schemes have been proposed and broken. In this paper, we use the delegation-by-certificate proxy signature scheme proposed in [13] which is summarized in Section 2.2. We choose this scheme for three reasons: First, its conceptual simplicity makes it easy to implement. Second, it works with standard signature schemes (e.g., RSA or DSA) that are widely supported by existing computing platforms. Third, the scheme is proven to be secure in [13]. Informally, this proxy signature scheme has the following features:

P1 *Verifiability*—From a proxy signature, a verifier can be convinced of the original signer's (i.e., the master DBMS's) agreement on the signed message.

P2 *Unforgeability*—The original signer and third parties who are not designated as the proxy signer (i.e., the cluster) cannot create a valid proxy signature.
P3 *Identifiability*—Anyone can determine the identity of the proxy signer from a proxy signature.
P4 *Undeniability*—The proxy signer cannot repudiate a proxy signature it created.
P5 *Prevention of misuse*—The proxy signing key cannot be used for purposes other than generating valid proxy signatures.

Fig. 5 depicts the TCP scheme: For each query, one server within the cluster would serve as the scheduler, while k of the remaining n servers are picked for query processing. For better availability and load balancing, the servers could take turns to be scheduler or query processor for different queries. Here, we extend the delegation-by-certificate proxy signature scheme by replacing the proxy signer's standard signature scheme with the (k,n) threshold RSA signature scheme. In the following, we describe the TCP system set-up, runtime operation, and its security features.

System set-up: Let (pk_M, sk_M) denote the public and private key pair of the master DBMS under a standard signature scheme. Let (pk_C, sk_C) be the public and private key pair of the cluster under the (k,n) threshold RSA signature scheme. We assume that pk_M and pk_C are disseminated to the users through authenticated channels, e.g., via public key certificates issued by a certificate authority. Furthermore, we assume that there is an authenticated channel such as SSL between each query processor and the scheduler, in order to prevent interception and replay attacks in their midst. The set-up of the TCP scheme involves the following steps:

(1) *Distribution of private key share*: Based on (pk_C, sk_C), a dealer (e.g., a trusted computer or a tamper-resistant hardware) generates private key share sk_i and public verification key vk_i for

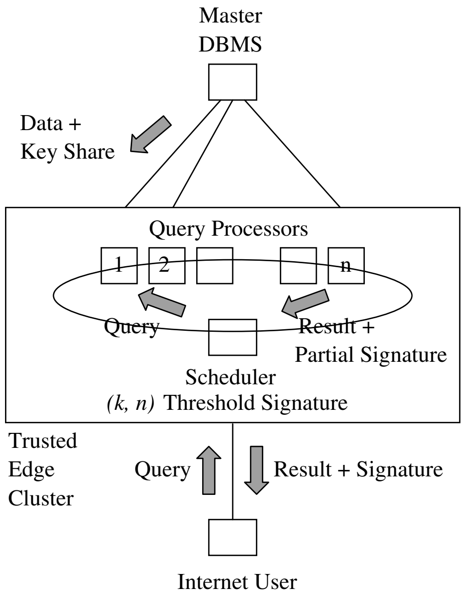

Fig. 5. Trusted cluster processors.

query processor i, $i = 1, 2, \ldots, n$. The dealer distributes the private key shares and the verification keys to the corresponding processors over secure channels and then erases its copy of the private key and shares.[1]

(2) *Delegation of signing rights*: The master DBMS issues a delegation certificate to the cluster of n processors, $d\_cert = (f_1|pk_C|\omega|\sigma_M)$, where f_1 is a flag used to indicate that d_cert is a delegation certificate, ω is a warrant specifying the validity period of the delegation as well as the type of query results the cluster is allowed to sign, and σ_M is the master DBMS's signature on "$f_1|pk_C|\omega$".

Runtime operation: The database and all subsequent updates are disseminated to the n query processors in the cluster; see [4,5] for example for update propagation techniques. This is done over a secure communication channel such as a virtual private network. Upon receiving a user query, the TCP proceeds as follows:

(1) The user query is directed to the cluster's scheduler. Taking into account the load level of the n query processors, the scheduler selects k of them for the query.
(2) Let a_j denotes the result of executing the query by selected processor j, $j = 1, 2, \ldots, k$, and f_2 be a flag used to indicate a proxy signature. Processor j generates a signature share $\sigma_j(f_2|pk_M|a_j)$ using its private key share sk_j, along with a "proof-of-correctness" ρ_j of the signature share based on its verification key vk_j. (Note that pk_M in $\sigma_j(f_2|pk_M|a_j)$ can be replaced by the unique identifier of the master DBMS without affecting the security property of the signatures.) Processor j sends a_j, $\sigma_j(f_2|pk_M|a_j)$ and ρ_j to the scheduler.
(3) With a_j, $\sigma_j(f_2|pk_M|a_j)$ and ρ_j, $\forall j = 1, 2, \ldots, k$
   - The scheduler checks if the k query results a_j are identical; if so, it proceeds to the next step. Denote the identical result as a.
   - The scheduler next verifies the correctness of the signature share $\sigma_j(f_2|pk_M|a)$ based on the proof-of-correctness ρ_j. If the k signature shares are all correct, they are combined to form the final signature $\sigma_C(f_2|pk_M|a)$. The scheduler then returns a and the proxy signature "$\sigma_C(f_2|pk_M|a)$, d_cert" to the user. Note that $\sigma_C(f_2|pk_M|a)$ is a standard RSA signature.

   If some of the query results disagree or some of the signature shares cannot be verified to be correct, the scheduler will solicit additional results and signature shares from the previously unused processors. This process continues until k identical results and k correct signature shares are obtained. Also, the scheduler raises an alarm on those processors that failed to provide correct result or signature share.
(4) Upon receiving a, $\sigma_C(f_2|pk_M|a)$ and d_cert, the user checks the validity of d_cert and $\sigma_C(f_2|pk_M|a)$, respectively. The user accepts the query result a as having been authenticated by the master DBMS only if the verification is successful.

Security features: A query processor can either be intact or corrupted. An intact processor follows its specified protocol and returns accurate query results and correct signature shares. In contrast, a corrupted processor may exhibit Byzantine failures, i.e., it deviates arbitrarily

[1] If a trusted dealer is not available, the master DBMS can take on the task of key generation. In that case, the unforgeability property (P2) applies only to third parties, not to the master DBMS/original signer.

from its specified protocol. Byzantine failures are the most severe and the most difficult to deal with [27].

Assuming that less than k query processors out of n in the cluster are corrupted at any given time, we claim that the query result a which successfully passes user verifications on "$\sigma_C(f_2|pk_M|a)$, *d_cert*" is both authentic and accurate, according to the definitions at the beginning of this section.

First we note that "$\sigma_C(f_2|pk_M|a)$, *d_cert*" is a proxy signature of the delegation-by-certificate proxy signature scheme. From property P2 we know that it is unforgeable. From property P1 we know that the user can be convinced of the master DBMS's agreement on the signed query result. This proves the authenticity aspect of the query result.

Next, we note that $\sigma_C(f_2|pk_M|a)$ is also a threshold signature of the (k,n) threshold RSA signature scheme. Based on property T1 and Step 3 of the Runtime Operation, a valid threshold signature $\sigma_C(f_2|pk_M|a)$ on the query result can be generated only if k signature shares are verified to be correct and the k associated query results are all identical. Consequently, if a valid signature is generated on an inaccurate query result, then at least k corrupted query processors must have colluded, i.e., they all provided the same inaccurate query result but correct signature shares on the result. However, this contradicts our assumption of having less than k corrupted processors and proves the accuracy aspect of the claim.

Finally, TCP is not based on Merkle tree or CB-tree, thus it does not share the limitations of [20] and UEP. In particular, TCP allows a user to confirm that no qualifying tuples are dropped from the query result, without compromising access control as there is no need to release projected attribute values or boundary tuples for user inspection here. Moreover, TCP is robust against query result substitution, as an adversary would need to simultaneously induce the scheduler and k of the query processors to return the identical wrong result to escape detection.

3.3. Trusted cluster verifiers (TCV)

While the TCP system described above requires several edge servers within a cluster to process each query concurrently, the trusted cluster verifiers (TCV) solution requires only one query processor and multiple light-weight verifiers per query. The idea is for the query processor to generate a verification object (VO) with each query result, as in UEP, then forward them to some pre-specified minimum number of verifiers for checking. The signature shares returned by those verifiers can then be combined to form a proxy signature for the query result. This eliminates the need for the user to perform projection, join and VO checking, which could require substantial computation and storage resources (see Appendix A.1). For better availability and load balancing, the edge servers in a cluster could take turns to be the query processor or verifier for different queries.

The TCV scheme is illustrated in Fig. 6. Here each cluster contains a query processor and n verifiers. This system employs the same delegation-by-certificate proxy signature scheme and (k,n) threshold RSA signature scheme as in TCP.

System set-up: The same as in the TCP system, with the only exception that the private key shares and the public verification keys are issued for the n verifiers, instead of the n query processors as in the TCP system.

Runtime operation: The database, CB-trees, and any updates to them are disseminated only to the query processor in the cluster. When a user query arrives at the cluster, the TCV scheme proceeds as follows:

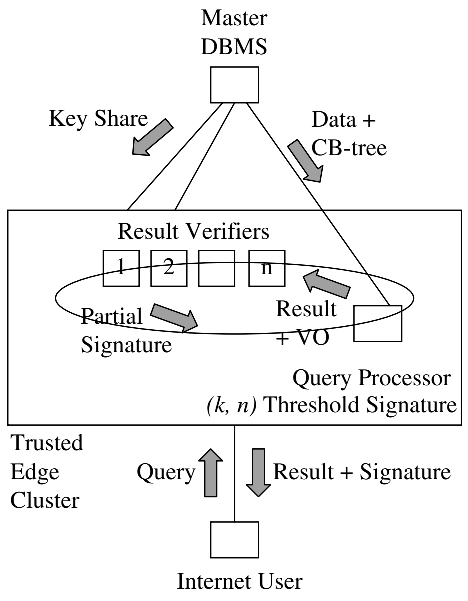

Fig. 6. Trusted cluster verifiers.

(1) The query processor generates query result a and the accompanying VO (see Section 3.1). Instead of returning a and VO directly to the user, however, the query processor channels them to k of the n verifiers.
(2) Each selected verifier $j, j = 1,2,\ldots,k$, first checks that a and VO match the signed root digest and then generates a signature share $\sigma_j(f_2|pk_M|a)$ using its private key share sk_j, along with a "proof-of-correctness" ρ_j of the signature share based on its verification key vk_j. Verifier j returns $\sigma_j(f_2|pk_M|a)$ and ρ_j to the query processor.
(3) Given $\sigma_j(f_2|pk_M|a)$ and ρ_j, $\forall j = 1,2,\ldots,k$, the query processor verifies the correctness of each signature share based on the corresponding proof-of-correctness. If some of the signature shares are found to be incorrect, the query processor contacts additional verifiers to obtain the missing signature shares. As soon as k correct signature shares are obtained, they are combined to form the final signature $\sigma_C(f_2|pk_M|a)$. The scheduler then returns a, $\sigma_C(f_2|pk_M|a)$ and $d\_cert = (f_1|pk_C|\omega|\sigma_M)$ to the user.
(4) Upon receiving a, $\sigma_C(f_2|pk_M|a)$ and *d_cert*, the user checks the validity of *d_cert* and $\sigma_C(f_2|pk_M|a)$, respectively. The user accepts the query result a as having been authenticated by the master DBMS only if the verification is successful.

In the case of a join operation, the query processor will transmit qualifying tuples in each targeted table, together with an accompanying VO, to the verifiers. The verifiers will then check the sub-tables, generate the join result, and finally sign it for the query processor.

Security features: According to the proof for UEP, it is computationally infeasible to produce a VO that matches an incorrect query result so as to escape detection by the verifiers. Since the verifiers cannot be deceived, they will certify an incorrect query answer only if they have been compromised. Assuming that there are less than k corrupted verifiers at any time, we can show that the query result a as certified by $\sigma_C(f_2|pk_M|a)$ and *d_cert* contains no spurious tuples or tampered data values. The proof follows a similar reasoning as that for the TCP system and is omitted here.

Although TCV too relies on CB-trees for result verification, it is able to let users detect whether any qualifying tuples have been dropped from the query results, without sacrificing access control like UEP. This is because here the query processor only needs to offer the projected attribute values and boundary tuples for inspection by the verifiers, not the users. Unfortunately, the price of having to maintain one CB-tree per database table still applies. In addition, TCV inherits UEP vulnerability to query result substitution, where a compromised query processor passes off output from a different query as result for a user query.

4. Experimental study

Having assessed the trusted data dissemination schemes from a security perspective, we now evaluate their performance trade-offs. We have conducted a series of experiments using discrete-event simulation models of the three architectures depicted in Figs. 3, 5 and 6. The workload is modelled after a moderate database size of *DBSize* = 1 GBytes, consisting of #*Rec* = 2 million tuples at *RecSize* = 512 Bytes each. With a standard block size of $|B|$ = 4 KBytes, the fan-out and height of the B-tree and CB-tree can be derived. The query generation follows a Poisson process with a mean inter-arrival time of *QueryArr* μs. Each query introduced into the system has a unique ID for tracking the query flow and response time. These resource and workload parameters are summarized in Table 1.

We assume a closed system, i.e., the number of *EdgeServ* and *Users* are fixed at the beginning of each experiment. Each user and edge server contains a set of processors where the speed of each can be stepped up or down by *XSpeed* times. The in/out buffer queue length of each node depends on the amount of available in/out link bandwidth and the processor speed. We assume an infinite buffer length for the queues, so there are no packet drops. Furthermore, the intranet and internet links have *IntraBW* and *InterBW* Mbps bandwidths, respectively. The signature/digest sizes and

Table 1
Resource and workload models

Parameter	Meaning	Default
$\|B\|$	Size of block/node	4 KBytes
$\|K\|$	Length of search key	16 Bytes
$\|\text{ptr}\|$	Size of node pointer	4 Bytes
DBSize	Size of database table	1 GBytes
RecSize	Length of each tuple	512 Bytes
#*Rec*	Number of tuples	2 mil
#*Col*	Number of attributes	32
f_{Btree}	Fan-out factor of B-tree	205
h_{Btree}	Height of B-tree	2
f_{CBtree}	Fan-out factor of CB-tree	114
h_{CBtree}	Height of CB-tree	3
QueryArr	Mean query inter-arrival time	1000 s
sf_{select}	Selectivity factor of selection	1 tuple

Table 2
System parameters

Parameter	Description	Default
SimTime	Total simulation time	100,000 s
WarmUp	Warm up time	5000 s
EdgeServ	Number of edge servers	20
Users	Number of users	20
Xspeed	Node speed up factor	1
IntraBW	Intranet bandwidth	1 Mbps
InterBW	Internet bandwidth	1 Mbps
SignSz	RSA signature size	128 bytes
SignGen	RSA signature generation	4630 μs
SignVeri	RSA signature verification	180 μs
DigestSz	MD5 digest size	16 bytes
HashRate	Digest hashing rate	214.491 bytes/μs
Seek	Disk average seek time	9000 μs
RotDelay	Disk average rotational delay	4700 μs
Transfer	Disk average transfer rate	7800 μs/blk

computation speeds are taken from the Crypto++ benchmark published at [28], while the disk parameters are set to published figures for the IBM 160 GB SATA hard disk. The simulator is run for a total of *SimTime* seconds, with observations in the initial *WarmUp* period being discarded while the link traffic and node usage status stabilize. Table 2 summarizes the system resource parameters.

The performance metrics include the average query response time, average link utilization, and average node utilization. Each query is timed from the moment it is issued by the user, until the returned result is fully processed by the user. The link utilization is calculated as the total amount of traffic (Mbytes) per link, divided by the simulation time and link bandwidth. Lastly, the utilization of a node is its cumulative busy periods over the simulation time.

4.1. Baseline experiment

The first experiment is intended to establish a baseline for the three schemes, with the default parameter settings in Tables 1 and 2. Fig. 7(a)–(e) plot the results against the query inter-arrival time. As expected, UEPs verification objects cause it to impose the highest load on the internet link between the user and edge server, TCP is the most demanding on the edge servers due to its parallel query execution strategy, while TCV shares the query results among the edge servers and hence induces the heaviest traffic on the intranet links. As for user node utilization in Fig. 7(e), UEP performs only hashing operations there, which is significantly cheaper than the signature verifications carried out by TCP and TCV. The utilization rate of the various resources, the highest of which reaches only 30%, confirm that this experiment models a resource-rich environment, hence the response times are limited only by typical processor and network speeds. In such an environment, UEP outperforms the other two schemes because users connect directly to the edge servers, without involvement of any intermediaries. However, UEP is not always the most efficient as we will see in the next experiment.

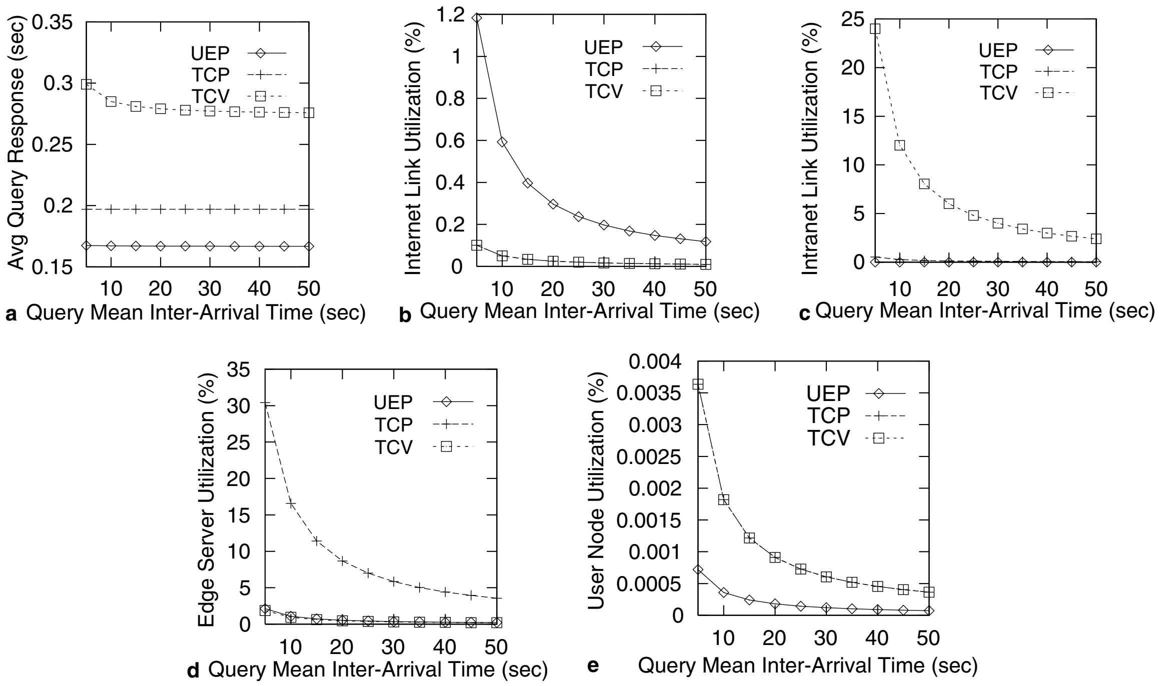

Fig. 7. Baseline experiment. (a) Response time; (b) Internet link; (c) Intranet link; (d) Edge server; (e) User.

4.2. Sensitivity to internet bandwidth

In practical deployment scenarios, the internet bandwidth is likely to be the resource that undergoes the highest load fluctuations. The next experiment is designed to profile the sensitivity of the various schemes to the available internet bandwidth *InterBW*. Fig. 8 shows that UEP experiences the worst deterioration in response time with a reduction in *InterBW*. This is because the verification objects that UEP sends to the users are a lot larger than the compact signatures in TCP and TCV, causing UEP to be highly sensitive to the internet bandwidth. In contrast, TCP and TCV are dependent on the capacity of the edge server and the interconnecting network between them, which are easier to provision. Among the two, TCP is the superior choice for the current resource settings.

4.3. Sensitivity to query selectivity

Next, we turn our attention to the performance impact of query selectivity, by varying it from 0.1% to 0.5% of *DBSize*. This leads to a corresponding growth in the size of the query results. As TCV needs to distribute each query result to several edge servers for verification, it experiences a super-linear slow-down in response time as the intranet links quickly become saturated with traffic, as shown in Fig. 9(a) and (c). UEP and TCP, in comparison, degrade gracefully with larger query results.

Another interesting observation is that, while UEP clearly has the lowest user node utilization in the first experiment, its utilization now exceeds that of TCP and TCV. The reason is that the

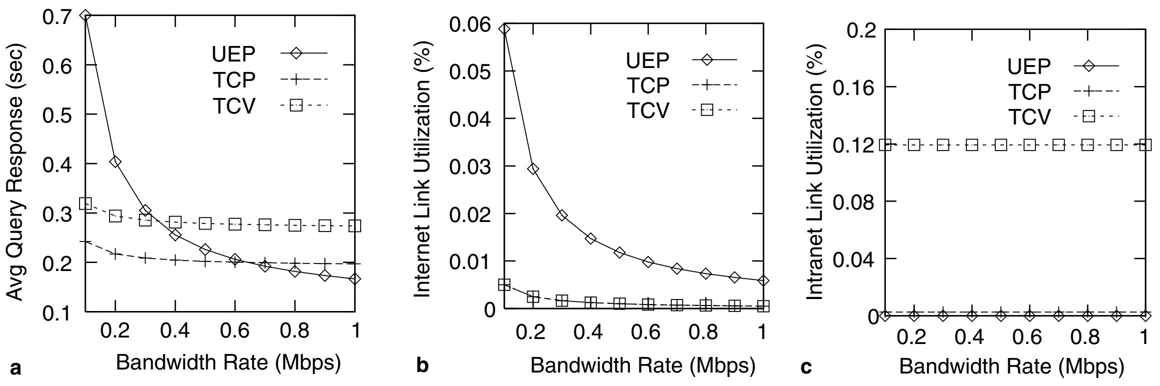

Fig. 8. Sensitivity to Internet bandwidth. (a) Response time; (b) Internet link; (c) Intranet link.

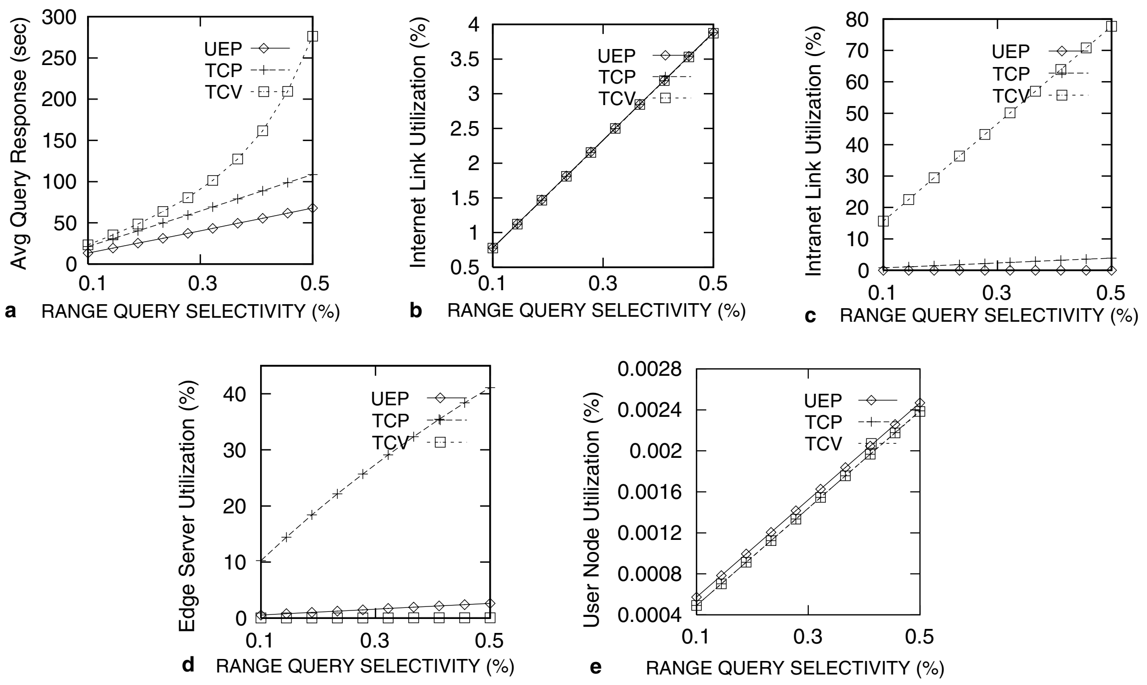

Fig. 9. Sensitivity to query selectivity. (a) Response time; (b) Internet link; (c) Intranet link; (d) Edge server; (e) User.

verification object is a tree structure with an integer height that is logarithmic in the query result size (see Appendix A for the derivation). This height has increased from 1 in the first experiment, to 2 here. In fact, although the tree height remains the same from selectivity of 0.1% through to 0.5%, it does grow further at higher selectivity settings. With each increase in tree height, the user node utilization of UEP would surpass TCP and TCV more and more. Therefore, UEP requires the user clients to possess adequate processing capacity.

4.4. Sensitivity to edge server speed

For the next experiment, we consider a system configuration comprising a few fast processors and many slower processors, which is common in practice. Fig. 10 shows the query performance as the speed of the slower processors is varied with the *XSpeed* parameter. We observe that UEP and TCP perform badly when the processor speed is low, because they depend heavily on the availability of a large group of edge servers for query processing. Between the two, UEP is more expensive as its query processing involves traversing a CB-tree with a height of 3, compared to TCP that parses a B-tree of height 2. In contrast, TCV is clearly able to adapt to the resource variation by executing most of the queries at the faster processors. This confirms that TCV could be a superior choice for systems that are configured with slow edge processors.

4.5. Sensitivity to k of n servers

Lastly, we examine the effect of k in the (k,n) threshold RSA signature scheme, with different resource configurations. The results are shown in Fig. 11(a)–(c), where the resources considered are the speed of the edge processors *XSpeed* and the internet bandwidth *InterBW*.

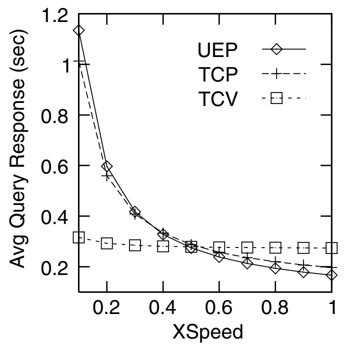

Fig. 10. Sensitivity to edge server speed.

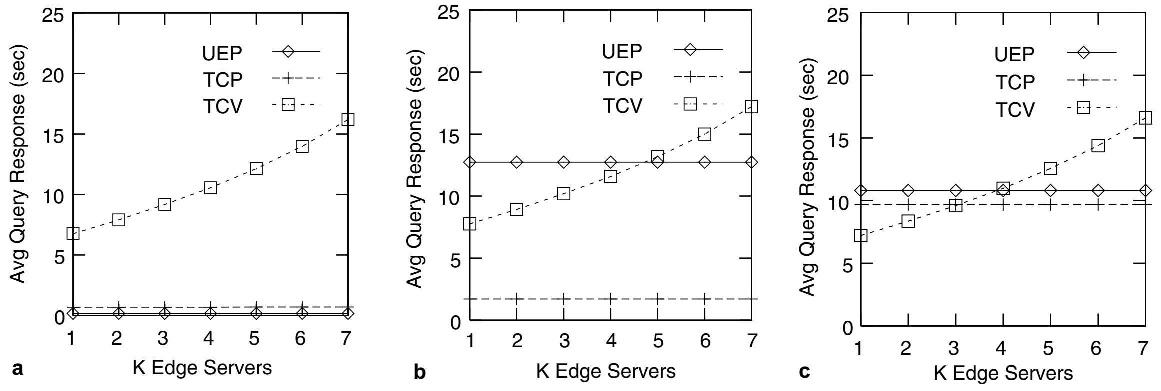

Fig. 11. Sensitivity to *k* of *n* servers. (a) Default speed and Internet; (b) Low internet bandwidth; (c) Slow processors.

In all the figures, as k increases, it affects the results of TCV adversely. This is because, the processing need at the edge servers and the intranet traffic (with the increasing number of VO) increase with larger k. This leads to slower responses back to the users. Similarly, TCP performance decreases, however the effect is not as significant as that compared to the TCV scheme as the overhead introduced per k value is much smaller. Clearly, UEP is unaffected through varying k as it does not depend on the threshold signature scheme.

In the first setting, both the speed of the processors and the internet bandwidth are set to the default values. The result is shown in Fig. 11(a). In this environment, the resources are in abundance and UEP outperforms the other two schemes as it does not involve any intermediaries.

In the next setting, we look at the impact of having a low internet bandwidth at 0.5% of *Inter BW*. In this environment, UEP performs badly as most users connect directly to the servers through the internet and the large VO are returned to the users for verification. This incurs greater traffic in the internet that results in UEP performing badly. With UEP performing worse, TCP now outperforms the other two schemes as seen in Fig. 11(b).

Lastly, we consider a system with servers comprising a few fast processors and many slower processors varying from 0.01 to 1 *XSpeed*. In Fig. 11(c), we observe that TCV now performs better than UEP and TCP for a number of k values. The reason for a slower UEP is because the presence of many slower processors are felt by the majority of the users. As for TCP, it performs badly as it depends heavily on the availability of a large group of processors for query processing. In contrast, TCV adapts better and outperforms the other two schemes for this system configuration.

4.6. Summary of experiment results

Based on the above analysis, the relative cost-performance of the three data dissemination schemes are summarized in Table 3. The "intuition" that have been confirmed include:

- UEP imposes the highest verification overheads on the client—the verification object that is sent to the client could be sizable, particularly for multipoint queries. Hence UEP works well only if the client has adequate processing capabilities and network bandwidth.

Table 3
Algorithm comparisons

	UEP	TCP	TCV
Security			
No tampered values	Yes	Yes	Yes
No spurious tuples	Yes	Yes	Yes
No missing result	No	Yes	Possible
No query result substitution	No	Yes	No
Costs			
Edge server processing	Low	High	Low
Inter-edge server traffic	NA	Low	High
Edge server-client traffic	High	Low	Low
Client processing	High	Low	Low

- TCP has lower query execution cost per server than UEP and TCV, because it uses B+-trees that have larger fan-out factors and hence potentially shorter tree height than CB-trees. However, this saving is not expected to offset the cost increase from having k servers execute each query simultaneously. Thus TCP requires abundant processing power at the edge servers.
- Like TCP, the TCV scheme diverts most of the verification overhead from the client, but at the expense of increased traffic between edge servers.

In addition, the experiments throw up a few interesting observations:

- TCV experiences super-linear slow-down in response time as query selectivity increases. Thus TCV is likely not the best choice for applications that frequently pulls back sizeable ranges of records.
- Slow edge servers affect UEP and TCP much more than TCV.

Finally, we note that TCP requires the master DBMS to transmit only data changes, whereas UEP and TCV additionally need the master DBMS to maintain and refresh the CB-trees on the edge servers. Hence UEP and TCV impose heavier burdens on the master DBMS and its network links to the edge servers.

Hence the three schemes are suitable for different resource configurations; none is consistently better than the other two.

5. Conclusion

This paper addresses the challenges of ensuring data security in an edge computing platform. We propose two new schemes for verifying the query results produced by the unsecured edge servers. The schemes are based on the observation that if necessary a group of edge servers can be running different operating systems and protected by different security products, thus increasing the difficulty for attackers to compromise all the edge servers concurrently without being detected. Our study shows that, where processing power is abundant at the edge servers, the scheme that assigns multiple edge servers to execute each query and to collectively certify the result is the most versatile, in terms of security guarantees and lower demand on other resources. In contrast, the second scheme engages only one of the edge servers for query execution, while the other edge servers perform verification and (collective) certification. This reduces the load on the edge servers, at the expense of increased traffic between the edge servers, and imposing extra work on the master DBMS to generate some auxiliary structures to facilitate verification. The two schemes present different security and resource trade-offs from the existing work, and are useful for different application scenarios and resource configurations.

Appendix A. Analysis of the schemes

The analysis of the schemes is as follows: We will focus only on selection-projection queries, in particular, (a) point queries that return at most one record based on an equality selection on a key attribute; (b) multipoint queries that may return several non-contiguous records based on an equality selection on a non-key attribute; and (c) range queries that return a set of records whose values for some attribute lie within a specified interval.

The cost metrics that we use to evaluate the data dissemination schemes include the query execution cost incurred by the edge processor(s), the edge server to edge server traffic, the edge server to client traffic, and the client processing cost.

A.1. Costs of untrusted edge processor

The fan-out factor of the CB-tree is:

$$f_{\text{CBtree}} = \left\lfloor \frac{|B| - |\text{ptr}| - |D|}{|K| + |\text{ptr}| + |D|} \right\rfloor + 1$$

This means that there is a space overhead of $f_{\text{CBtree}} \times |D|$ bytes per node for storing digests. Moreover, the height of the CB-tree is at least:

$$h_{\text{CBtree}} = \left\lceil \log_{f_{\text{CBtree}}} = \frac{\#Rec}{f_{\text{CBtree}} - 1} \right\rceil$$

and the total number of nodes in the CB-tree is:

$$\#Nodes = \left\lceil \frac{(f_{\text{CBtree}})^{h_{\text{CBtree}}+1} - 1}{f_{\text{CBtree}} - 1} \right\rceil$$

A.1.1. Point queries

- Query processor

  The query execution cost includes $h_{\text{CBtree}} + 1$ I/Os for traversing the CB-tree, plus one I/O to retrieve the result tuple, giving a total of $h_{\text{CBtree}} + 2$ I/Os. (We count only the I/O cost as it dominates the CPU cost.)
- Query processor to client network

  In each CB-tree node on the path from the root to the result tuple, there are $f_{\text{CBtree}} - 1$ digests to be copied into the VO. Hence there are altogether $(h_{\text{CBtree}} + 1) \times (f_{\text{CBtree}} - 1)$ of these digests. Assuming that the edge server chooses to return the digests of the projected attributes because the digests are more compact than the actual attributes (which could be binary large objects), that adds $\#Col \times (1 - sf_{\text{project}})$ digests. Including the signed root digest, the size of the verification object is $((h_{\text{CBtree}} + 1) \times (f_{\text{CBtree}} - 1) + 1 + \#Col \times (1 - sf_{\text{project}})) \times |D|$.

  The expected result size is $RecSize \times sf_{\text{project}}$. The edge processor to client traffic is the sum of the VO size and the result size.
- Client

  The client processing involves hashing the result values, then computing the root digest from the VO. The latter includes (a) concatenating $\#Col$ attribute digests, plus one hash to derive the digest for the result tuple at a cost of $\#Col \times C_{\text{concat}} + C_{\text{hash}}$; and (b) for each CB-tree node on the path from the root to the result tuple, concatenating the digests of f_{CBtree} children plus one hash to derive the node digest; this part of the cost is $(h_{\text{CBtree}} + 1) \times (f_{\text{CBtree}} \times C_{\text{concat}} + C_{\text{hash}})$.

A.1.2. Multipoint queries

- Query processor
  The query result contains $\#Rec \times sf_{\text{select}}$ tuples. If they are located through the CB-tree index, the edge server would have to traverse the height of the CB-tree once for every tuple, with the exception of the root which can be cached. Including the I/Os for retrieving the result tuples, the query execution cost is $\#Rec \times sf_{\text{select}} \times (h_{\text{CBtree}} + 1) + 1$ I/Os. However, if there are too many result tuples, then it would be cheaper to simply scan the table, at a cost of $\lceil \frac{DBSize}{|B|} \rceil$ I/Os. Therefore the query execution cost is $\min(\#Rec \times sf_{\text{select}} \times (h_{\text{CBtree}} + 1) + 1, \lceil \frac{DBSize}{|B|} \rceil)$ I/Os.
- Query processor to client network
  As for the VO, it would contain $f_{\text{CBtree}} - 1$ digests from every node that leads to a result tuple. There are up to $\#Rec \times sf_{\text{select}} \times h_{\text{CBtree}} + 1$ such distinct nodes, subject to the maximum of the entire CB-tree which has a total of $\lceil \frac{(f_{\text{CBtree}})^{h_{\text{CBtree}}+1} - 1}{f_{\text{CBtree}} - 1} \rceil$ nodes. Including the signed root digest, the size of this part of the VO is thus $[\min(\#Rec \times sf_{\text{select}} \times h_{\text{CBtree}} + 1, \lceil \frac{(f_{\text{CBtree}})^{h_{\text{CBtree}}+1} - 1}{f_{\text{CBtree}} - 1} \rceil) \times (f_{\text{CBtree}} - 1) + 1] \times |D|$. In addition, the VO also contains digests for the projected attributes, which expands the VO size by $(\#Rec \times sf_{\text{select}} \times \#Col \times (1 - sf_{\text{project}})) \times |D|$.
  Since each result tuple is $RecSize \times sf_{\text{project}}$ in size, the result size is $\#Rec \times sf_{\text{select}} \times RecSize \times sf_{\text{project}}$. Adding that to the VO size gives the edge processor to client traffic.
- Client
  The client processing involves (a) hashing the result values, then concatenating with the digests for the projected attributes at a cost of $\#Rec \times sf_{\text{select}} \times (\#Col \times C_{\text{concat}} + C_{\text{hash}})$; and (b) computing the root digest from the VO at a cost of $\min(\#Rec \times sf_{\text{select}} \times h_{\text{CBtree}} + 1, \lceil \frac{(f_{\text{CBtree}})^{h_{\text{CBtree}}+1} - 1}{f_{\text{CBtree}} - 1} \rceil) \times (f_{\text{CBtree}} \times C_{\text{concat}} + C_{\text{hash}})$.

A.1.3. Range queries

- Query processor
  The query processing involves traversing down the CB-tree to reach the first qualifying tuple, then scanning the table till the last qualifying tuple. The query execution cost is $h_{\text{CBtree}} + 1 + \lceil \frac{DBSize}{|B|} \times sf_{\text{select}} \rceil$ I/Os.
- Query processor to client network
  The VO contains digests in the nodes along the boundaries of the smallest subtree that covers the result tuples, and the nodes from the top of the subtree up to the root of the CB-tree [20]. The height of the subtree is $h_{\text{result}} = \lceil \log_{f_{\text{CBtree}}} \frac{\#Rec \times sf_{\text{select}}}{f_{\text{CBtree}} - 1} \rceil$. There are $2 \times h_{\text{result}} + 1$ nodes along the boundaries of the subtree, each of which contributes $f_{\text{CBtree}} - 1$ digests to the VO. From the top of the subtree up to the root of the CB-tree, there are $h_{\text{CBtree}} - h_{\text{result}}$ nodes that adds another $f_{\text{CBtree}} - 1$ digests each. Including the signed root digest, this part of the VO therefore has a size of $[(2 \times h_{\text{result}} + 1 + (h_{\text{CBtree}} - h_{\text{result}})) \times (f_{\text{CBtree}} - 1) + 1] \times |D|$. In addition, the VO also contains digests for the projected attributes, which expands the VO size by $(\#Rec \times sf_{\text{select}} \times \#Col \times (1 - sf_{\text{project}})) \times |D|$. The result size is $\#Rec \times sf_{\text{select}} \times RecSize \times sf_{\text{project}}$, as with multipoint queries.

- Client

  The client processing involves (a) computing the result tuples' digests at a cost of $\#Rec \times sf_{\text{select}} \times (\#Col \times C_{\text{concat}} + C_{\text{hash}})$; and (b) deriving the root digest, from all the nodes in the result subtree plus the nodes up to the CB-tree root, for verification at a cost of $[\frac{(f_{\text{CBtree}})^{h_{\text{result}}+1}-1}{f_{\text{CBtree}}-1} + (h_{\text{CBtree}} - h_{\text{result}})] \times (f_{\text{CBtree}} \times C_{\text{concat}} + C_{\text{hash}})$.

A.2. Costs of trusted cluster processors

The fan-out factor of the conventional B-tree is:

$$f_{\text{Btree}} = \left\lfloor \frac{|B| - |\text{ptr}|}{|K| + |\text{ptr}|} \right\rfloor + 1$$

and the height is:

$$h_{\text{Btree}} = \left\lceil \log_{f_{\text{Btree}}} \frac{\#Rec}{f_{\text{Btree}} - 1} \right\rceil$$

A.2.1. Point queries

- Query processor

  Since each query is executed by k processors, the total query execution cost = $k \times (h_{\text{Btree}} + 2)$ I/Os.
- Query processor to scheduler network

  Assuming that the k processors transmit their partial signatures, but only one sends the query result to the scheduler, the traffic is $RecSize \times sf_{\text{project}} + k \times |D|$.
- Scheduler to client network

  This traffic consists of the query result plus the final signature, and costs $RecSize \times sf_{\text{project}} + |D|$.
- Client

  The client processing here involves only hashing the result values for matching with the final signature.

A.2.2. Multipoint queries

- Query processor

  The total query execution cost is $k \times \min(\#Rec \times sf_{\text{select}} \times (h_{\text{Btree}} + 1) + 1, \lceil\frac{DBSize}{|B|}\rceil)$ I/Os, similar to the corresponding cost for UEP except for the different index tree heights.
- Query processor to scheduler network

  The cost of sending k partial signatures and one set of query result is $\#Rec \times sf_{\text{select}} \times RecSize \times sf_{\text{project}} + k \times |D|$.

- Scheduler to client network
  The amount of traffic that is sent to the client is $\#Rec \times sf_{\text{select}} \times RecSize \times sf_{\text{project}} + |D|$.
- Client
  The client processing here involves only hashing the result values for matching with the signature.

A.2.3. Range queries

- Query processor
  The total query execution cost is $k \times (h_{\text{CBtree}} + 1 + \lceil \frac{DBSize}{|B|} \times sf_{\text{select}} \rceil)$ I/Os, again similar to the corresponding cost for UEP.
- Query processor to scheduler network
  The cost of sending k partial signatures and one set of query result is $\#Rec \times sf_{\text{select}} \times RecSize \times sf_{\text{project}} + k \times |D|$ as with multipoint queries.
- Scheduler to client network
  The amount of traffic that is sent to the client is $\#Rec \times sf_{\text{select}} \times RecSize \times sf_{\text{project}} + |D|$.
- Client
  The client processing here involves only hashing the result values for matching with the signature.

In the cost analysis for TCP, we have ignored the scheduler processing cost in combining k partial signatures as that is negligible compared to the other cost components.

A.3. Costs of trusted cluster verifiers

The costs that TCV imposes on the various resources are similar to those for UEP and TCP, as TCV is a combination of those two schemes.

- Query processor
  Since the computation in combining the partial signatures from the verifiers is negligible, the cost at the query processor for the three types of queries are the same as UEP's.
- Query processor to verifier network
  The query processor sends the result and VO to each of the verifiers. This generates k times the traffic of UEP. The partial signatures sent back by the verifiers produce an additional traffic of $k \times |D|$.
- Query processor to client network
  The amount of traffic sent back to the client here is the same as the TCP scheme.
- Client
  The client processing here is the same the TCP scheme.

References

[1] D. Margulius, Apps on the Edge, InfoWorld 24(21), Available from: <http://www.infoworld.com/article/02/05/23/020527feedgetci_1.html>.

[2] G. Barish, K. Obrazka, World wide web caching: trends and techniques, IEEE Communications Magazine 38 (5) (2000) 178–184.
[3] J. Wang, A survey of web caching schemes for the internet, ACM Computer Communication Review 25 (9) (1999) 36–46.
[4] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, G. Alonso, Understanding replication in databases and distributed systems, in: Proceedings of 20th International Conference on Distributed Computing Systems (ICDCS'2000), 2000, pp. 264–274.
[5] M. Wiesmann, A. Schiper, F. Pedone, B. Kemme, G. Alonso, Database replication techniques: a three parameter classification, in: Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00), 2000, pp. 206–215.
[6] Encrypting File System (EFS) for Windows 2000, Available from: <http://www.microsoft.com/windows2000/techinfo/howitworks/security/encrypt.asp>.
[7] H. Pang, K. Tan, X. Zhou, StegFS: a steganographic file system, in: Proceedings of the 19th International Conference on Data Engineering, Bangalore, India, 2003, pp. 657–668.
[8] R. Anderson, R. Needham, A. Shamir, The steganographic file system, in: D. Aucsmith (Ed.), Information Hiding, 2nd International Workshop, Portland, Oregon, USA, 1998.
[9] Secure Hashing Algorithm, National Institute of Science and Technology, FIPS 180-2, 2001.
[10] R. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (2) (1978) 120–126.
[11] Proposed Federal Information Processing Standard for Digital Signature Standard (DSS), Federal Register 56(169) (1991) 42980–42982.
[12] V. Shoup, Practical threshold signatures, Advances in Cryptology—Eurocrypt 2000, LNCS 1807, Springer-Verlag, 2000, pp. 207–220.
[13] A. Boldyreva, A. Palacio, B. Warnschi, Secure Proxy Signature Schemes for Delegation of Signing Rights, Available from: <http://venona.antioffline.com/2003/096.pdf>.
[14] R. Merkle, A certified digital signature, in: Proceedings of Advances in Cryptology-Crypto'89, Lecture Notes in Computer Science, vol. 0435, 1999, pp. 218–238.
[15] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, G. Stubblebine, Flexible authentication of XML documents, in: Proceedings of the 8th ACM Conference on Computer and Communications Security, 2001, pp. 136–145.
[16] B. Carminati, E. Ferrari, E. Bertino, Secure third party distribution of XML data, in: Proceedings of the IEEE International Conference on Data Engineering, 2005.
[17] M. Goodrich, R. Tamassia, A. Schwerin, Implementation of an authenticated dictionary with skip lists and commutative hashing, in: DARPA Information Survivability Conference and Exposition (DISCEX II), vol. 2, 2001, pp. 1068–1084.
[18] M. Naor, K. Nissim, Certificate revocation and certificate update, in: Proceedings of the 7th USENIX Security Symposium, 1998, pp. 217–228.
[19] C. Peng, R. Deng, Y. Wu, W. Shao, A flexible and scalable authentication scheme for JPEG2000 images, in: Proceedings of the ACM International Conference on Multimedia, 2003.
[20] P. Devanbu, M. Gertz, C. Martel, S. Stubblebine, Authentic data publication over the Internet, in: 14th IFIP 11.3 Working Conference in Database Security, 2000, pp. 102–112.
[21] M. Roos, A. Buldas, J. Willemson, Undeniable replies for database queries, in: Proceedings of the Baltic Conference, BalticDB&IS, 2002, pp. 215–226.
[22] D. Barbara, R. Goel, S. Jajodia, Using checksums to detect data corruption, in: Proceedings of the International Conference on Extending Database Technology, 2000, pp. 136–149.
[23] E. Mykletun, M. Narasimha, G. Tsudik, Authentication and integrity in outsourced databases, in: Proceedings of the Network and Distributed System Security Symposium, 2004.
[24] H. Pang, K. Tan, Authenticating query results in edge computing, in: IEEE International Conference on Data Engineering, 2004, pp. 560–571.
[25] Y. Desmedt, Society and group oriented cryptography: a new concept, in: Advances in Cryptology—Crypto'87, 1987, pp. 120–127.

[26] M. Mambo, K. Usuda, E. Okamoto, Proxy signatures for delegating signing operation, in: Proceedings of the 3rd ACM Conference on Computer and Communications Security, 1996, pp. 48–57.
[27] R.S.M. Pease, L. Lamport, Reaching agreement in the presence of faults, Journal of the ACM 27 (2) (1980) 228–234.
[28] Crypto++ 5.2.1 Benchmarks, Available from: <http://www.eskimo.com/weidai/benchmarks.html>.

Shen-Tat Goh received his B.Sc. and M.Sc. from National University of Singapore. He is currently a Senior Research Engineer at the Institute for Infocomm Research in Singapore. His research interests includes database management systems and data security.

HweeHwa Pang received the B.Sc.—with first class honors—and M.S. degrees from the National University of Singapore in 1989 and 1991, respectively, and the Ph.D. degree from the University of Wisconsin at Madison in 1994, all in Computer Science. He is currently a Principal Scientist at the Institute for Infocomm Research in Singapore. His research interests include database management systems, data security and quality, operating systems, and multimedia servers. He has many years of hands-on experience in system implementation and project management. He has also participated in transferring some of his research results to industry.

Robert H. Deng received his B.Eng from National University of Defense Technology, China, his M.Sc. and Ph.D. from Illinois Institute of Technology, Chicago. He is currently Professor and Director of SIS Research Center, School of Information Systems, Singapore Management University. Prior to this, he was Principal Scientist and Manager of Infocomm Security Department, Institute for Infocomm Research. He has 21 patents and more than 140 technical publications in international conferences and journals in the areas of digital communications, network and distributed system security and information security. He has served as general chair, program chair, and program committee member of numerous international conferences. He received the University Outstanding Researcher Award from the National University of Singapore in 1999.

Feng Bao received his BS in mathematics, MS in computer science from Peking University and his Ph.D. in computer science from Gunma University in 1984, 1986 and 1996 respectively. He was an assistant/associate professor of the Institute of Software, Chinese Academy of Sciences from 1987 to 1993 and a visiting scholar of Hamberg University, Germany from 1990 to 1991. Since 1996 he has been with the Institute for Infocomm Research, Singapore. Currently he is a Lead Scientist and the head of Infocomm Security Department and Cryptography Lab of the institute. His research areas include algorithm, automata theory, complexity, cryptography, distributed computing, fault tolerance and information security. He has over 120 publications and 16 patents.