Practical and Privacy-Preserving Policy Compliance for Outsourced Data

Giovanni Di Crescenzo[1]([✉]), Joan Feigenbaum[2], Debayan Gupta[2],
Euthimios Panagos[1], Jason Perry[3], and Rebecca N. Wright[3]

[1] Applied Communication Sciences, Middlesex County, NJ, USA
{gdicrescenzo,epanagos}@appcomsci.com
[2] Yale University, New Haven, CT, USA
{joan.feigenbaum,debayan.gupta}@yale.edu
[3] Rutgers University, Middlesex County, NJ, USA
{jasperry,rebecca.wright}@cs.rutgers.edu

Abstract. We consider a scenario for data outsourcing that supports performing database queries in the following three-party model: a client interested in making database queries, a data owner providing its database for client access, and a server (e.g., a cloud server) holding the (encrypted) outsourced data and helping both other parties. In this scenario, a natural problem is that of designing efficient and privacy-preserving protocols for checking compliance of a client's queries to the data owner's query compliance policy. We propose a cryptographic model for the study of such protocols, defined so that they can compose with an underlying database retrieval protocol (with no query compliance policy) in the same participant model. Our main result is a set of new protocols that satisfy a combination of natural correctness, privacy, and efficiency requirements. Technical contributions of independent interest include the use of equality-preserving encryption to produce highly practical symmetric-cryptography protocols (i.e., *two orders of magnitude faster* than "Yao-like" protocols), and the use of a query rewriting technique that maintains privacy of the compliance result.

1 Introduction

The recent information technology trend of outsourcing "big data" in the "cloud" is being embraced in banking, finance, government and other areas. Banks and financial institutions need to process huge data volumes on a daily basis; in government, large databases are needed in many contexts (e.g., no-fly lists, metadata of communication records, etc.). Cloud storage and computing provide tremendous efficiency and utility for users (as exemplified by the "database-as-a-service" application paradigm), but they also create privacy risks. To mitigate these risks, database-management systems can use *privacy-preserving* data-retrieval protocols that allow users to submit queries and receive results in a way that users learn nothing about the contents of a database except the results of their queries, data owners do not learn which queries are submitted. Of critical importance

for the success of database management systems is the notion of *access control*, which requires carefully crafted *data-access policies*. Compliance of these policies can be enforced by the database-management system but might need to be confidential, as the policies themselves may reveal sensitive facts about the data and its owner. In this paper, we formalize, design, and analyze practical and privacy-preserving policy compliance protocols for outsourced data.

Our Problem: Our goal is to augment natural encrypted database retrieval solutions with a query authorization property based on compliance to a policy, while preserving the privacy and efficiency properties of the basic database retrieval solution. For consistency with the database-as-a-service model, and to achieve practical solutions, we consider a 3-party model, shown in Fig. 1, including a client C (interested in private data retrieval), a data owner D (offering data for retrieval conditioned on compliance to a query-specific policy), and a server S (e.g., a cloud server) helping both parties to achieve their goals. In this paper, we focus on the policy-compliance building block. Our solutions can be combined in a modular fashion with database retrieval (DR) protocols in this 3-party model, provided that they satisfy some natural structure and properties (described later). Such protocols already exist in the literature (e.g., [5,11,18]).
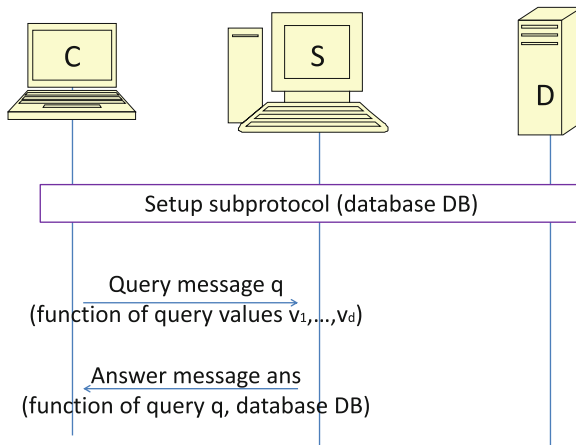

Fig. 1. Structure of a database-retrieval protocol

Our Contributions. We investigate the modeling and design of database policy compliance (DPC) protocols that combine with known DR protocols, as shown in Fig. 2, and that satisfy the following novel set of requirements:

1. *Preservation of Query Correctness*: A client that could retrieve all of the records that satisfy its query using a DR protocol can still do so if the query is compliant;
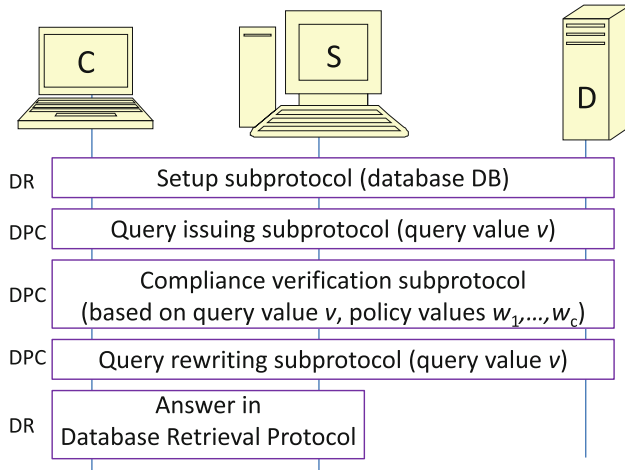
Fig. 2. Composition of a DR protocol with a DPC protocol

2. *Compliance Completeness*: All queries that satisfy (resp., do not satisfy) the policy are found to be compliant with probability 1 (resp., with negligible[1] probability);
3. *Compliance Soundness*: For any efficient (and even malicious) adversary impersonating the client, the data owner can correctly compute (except with negligible probability) the policy compliance of whichever query message is received and answered by the server according to the DR protocol;
4. *Privacy*: Privacy of database values, policy values and query values is preserved, in that no efficient semi-honest adversary corrupting one among the parties C, S, D learns more information at the end of the protocol than whatever is efficiently computable from the following: the system parameters (which are intended to be known by all parties), the compliance bit b (if the corrupted party is D, who is intended to learn b), the query message Q' (if the corrupted party is S, who is intended to learn Q'), where Q' has the same distribution as the query message Q in the DR protocol when the query is compliant, or otherwise represents a query that does not match any records in DB (to reduce leakage of the policy-compliance result to S or C);
5. *Efficiency*: The protocol should have low time, communication and round complexity. One of the most significant design criteria we target to reduce computational overhead of query compliance checking is to minimize or eliminate costly public-key cryptographic operations and to achieve protocols faster than a direct application of secure function evaluation techniques.

Implicit in the above privacy requirement is the fact that the protocol does not reveal new information about the data owner's policy to client and server,

---

[1] A function is *negligible* if for any positive polynomial p and all sufficiently large natural numbers $\sigma \in \mathcal{N}$, it is smaller than $1/p(\sigma)$.

other than what is revealed to clients by fulfilled queries. Still, to hide some additional information about the policy, the privacy requirement also demands that the result of a non-compliant query is indistinguishable from a query that matches zero records in the database, so that the protocol does not reveal to clients whether a query that returns no matches does so because it is non-compliant or because there are actually no matching records[2].

We design three protocols for enforcing compliance of *keyword search* queries. We only consider *whitelist* (resp., *blacklist*) policy types, where the query is compliant only if the query value is equal to one (resp., none) of the policy values. For such query and policy types, we provide highly efficient and scalable database policy compliance protocols that satisfy all our requirements, as detailed below (the PRP assumption being the existence of pseudo-random permutations).

Requirement	Protocol π_1	Protocol π_2	Protocol π_3
Correctness preservation	If DR protocol satisfies Added Property 1	If DR protocol satisfies Added Property 2	If DR protocol satisfies Added Property 1
Compliance completeness	Under no complexity assumption	Under no complexity assumption	Under no complexity assumption
Compliance soundness	(not satisfied)	Under no complexity assumption	Under no complexity assumption
Privacy	Under PRP assumption (some leakage to S)	Under PRP assumption (some leakage to S)	Under PRP assumption And based on SFE
Time	Linear in policy size	Linear in policy size	Linear in policy size
Communication	Linear in policy size	Linear in policy size	Linear in policy size
Rounds	$O(1)$ in policy size	$O(1)$ in policy size	$O(1)$ in policy size

An additional important property is that all our 3 DPC protocols only require $O(1)$ cryptographic operations per query and policy value, and are about two orders of magnitude faster than 2-party arbitrary function evaluation protocols [19], which require at least $\Omega(\ell)$ cryptographic operations per query and policy value, where ℓ denotes the length of these values (even in recent optimized solutions). Just like achieved previously for DR protocols in the 3-party model, our DPC protocols not only minimize or eliminate costly public-key cryptography operations, but they provide concrete time efficiency, which we document through performance numbers from our implementations (in Sect. 4). Our solutions rest on two main technical contributions: (1) using equality-preserving symmetric encryption with multiple keys shared among different subsets of

---

[2] Of course, sometimes a client is able to distinguish these cases due to auxiliary information.

parties (building on [3]) for efficient 3-party computation on encrypted data, and (2) performing policy-based query rewriting to make the results of non-compliant queries indistinguishable from queries matching no records. Formal definitions and proofs are omitted due to space restrictions.

Related Work. To the best of our knowledge, there is no previous work on privacy-preserving, efficient, *query* policy compliance checking for database queries. That is, although there has been previous work on 3-party protocols in which the data set being searched is encrypted, the query is kept private, and queries are only allowed if they satisfy certain structural conditions, we are unaware of previous work in which the restriction on allowable queries (i.e., the policy) depends on the query and/or is kept private from the clients. Existing work in this area focuses on policy conditions that mainly depend on the database attributes (see, e.g., [5,8,11,18]) or on the identity of the clients (see, e.g., [4,10,13,16], or consider different kinds of access control in such systems (see, e.g., [12,14]).

Our work is also somewhat related (in a complementary way) to a number of areas in theoretical and applied cryptography, including private information retrieval [6], searchable symmetric encryption [17], searchable public-key encryption [2] and oblivious RAMs [9]. Previous cryptographic work in a 3-party model (also referred as commodity-based, server-assisted, server-aided model) seems to have originated in [1], with respect to oblivious transfer protocols, and [7], with respect to private information retrieval.

2   Models, Definitions and Properties

We discuss models and DR protocol properties used in the rest of the paper, and further clarify the privacy and security properties that our DPC protocols must satisfy.

Data, Query and Policy Models. We model a *database table* (briefly, database) as a matrix DB with n rows and m columns, where each row is associated with a data *record*, each column is associated with a data *attribute*, and each database entry $DB(i,j)$ is the value of the j-th attribute of the i-th record. The database *schema* consists of n, m, and the *domains* of each of the m attributes (*i.e.*, the j-th domain is the set of values that the j-th attribute of a record can take), and is assumed to be known by all parties that participate in the protocol. We assume that domains are large in that a randomly chosen domain element is, with very high probability, not in DB. (If DB does not satisfy these conditions, then simple padding of domain strings can be used to make it so.)

A *query* q contains a database attribute and a *query value* v from the corresponding attribute domain. We consider keyword-match queries of the following form (using SQL notation): "SELECT $*$ FROM $main$ WHERE attribute_name $= v$".

A data owner's *query compliance policy* (briefly, *policy*) contains, for each attribute $j \in \{1, \ldots, m\}$, a set $W_j = \{w_{j,1}, \ldots, w_{j,c_j}\}$ of *policy values* drawn from the j-th domain. All of the clients that access DB through this data owner

are subject to the same policy. On input a query value v, an attribute name (or, equivalently, an attribute index j), and a set of attribute values W_j, the policy returns 1 (resp., 0) to denote query compliance (resp., non-compliance). We mainly consider the *whitelist* and *blacklist* policies:

1. *Whitelist*: If query q refers to the j-th attribute, then p returns 1 iff $v \in W_j$;
2. *Blacklist*: If query q refers to the j-th attribute, then p returns 1 iff $v \notin W_j$.

Intuitively, a blacklist policy captures the notion of a set of forbidden query values, while a whitelist policy restricts queries to a specified set of allowed values. We assume that the lengths c_j of whitelists and blacklists and the lengths of the policy values $w_{j,k}$ are system parameters known to all parties (although our protocols will keep the latter values hidden from C).

DR Protocol Properties. We consider DR protocols, as depicted in Fig. 1, with the following structure:

1. C, D and S run a preliminary setup subprotocol
   (this enables S to later answer C's query on the database owned by D)
2. Given a query q, C constructs a *query message* Q and sends it to S
3. S computes an *answer message* **ans** and sends it to C
4. Based on Q and **ans**, C can compute database records that satisfy q, if any.

The *unique-query* property requires that, for any database DB and any properly formatted query message Q, there is at most one pair (attribute_name, v) for which C could have generated query message Q. When such a pair exists, we refer to v as the "query value associated with Q."

The *query-correctness* property requires that, for any database DB, any input pair (attribute_name, v), and any Q with associated query value v, at the end of the DR protocol, C can compute all records in DB that satisfy query attribute_name $= v$.

We also impose some additional structural properties on DR protocols:

1. *Added DR Property 1:* At the end of step 1, S stores $F(k_{c,d}; DB(i,j))$, for each database entry $DB(i,j)$, where F is a pseudo-random permutation and $k_{c,d}$ denotes a key shared between C and D;
2. *Added DR Property 2:* At the end of step 1, for each database entry $DB(i,j)$, S stores the triple encryption $F(k_{c,d}; F(k_{c,s}; F(k_{c,d}; DB(i,j))))$, where F is a pseudo-random permutation and $k_{c,d}$ (resp., $k_{c,s}$) denotes a key shared between C and D (resp., C and S).

Note the following simple DR protocol satisfying Property 1: query values and data values are encrypted via a pseudo-random permutation, a query message contains the encrypted query value, and the answer message contains the records with encrypted data values equal to the encrypted query value. Other examples can be found in the literature (see, e.g., [11,18]). It should also be noted that any protocol satisfying Property 1 can be turned into one that satisfies Property 2, and that our techniques will work with a number of variations of these example properties.

DPC Protocol Properties. Our DPC protocols compose with DR protocols as follows (see Fig. 2): after the DR setup subprotocol, instead of a single query message Q sent from C to S, we now have three subprotocols (a query subprotocol, a compliance-verification subprotocol, and a query rewriting subprotocol) after which a query message is sent to S, and then the answer step of the DR protocol can be executed.

The requirements we demand from any DPC protocol were already informally described and motivated in Sect. 1. Here, we only further clarify its input/output behavior and privacy requirement. The inputs to a DPC protocol are a security parameter 1^σ (known to all parties), an attribute name and query value v (private inputs to C), and a database DB (schema known to all parties, but contents private to D). The outputs of a DPC protocol are a query message Q' (communicated privately to S) and a bit b (communicated privately to D) indicating whether the query complies with the policy ($b = 1$) or not ($b = 0$). We consider privacy in multiple runs of the DPC protocol against a semi-honest probabilistic polynomial-time adversary Adv (with history as auxiliary input) corrupting up to one party, by a natural adaptation of the real/ideal security framework, as typically used in the cryptography literature. Briefly speaking, a (real-world) execution of multiple runs of the DPC protocol are executed, does not leak to Adv more than the ideal-world leakage, defined as follows. On input of a query value v given by C, a database DB, policy values $w_1, \ldots, w_c$, and policy p input by D, each ideal execution of a single DPC protocol returns:

1. the output b of policy p on input query value v and policy values $w_1, \ldots, w_c$ to D
2. a random query message Q' to S, where Q' has no matching records if $b = 0$ or has associated query value v if $b = 1$.

In our first two protocols, we admit some additional leakage to S, and consider the variant of the above definition, where such leakage is also admitted in the ideal world.

Our design also targets a number of additional security properties, which can be obtained using network security protocols such as TLS: *confidentiality* of the communication between all participants, message *sender authentication*, message *receiver authentication*, and *communication integrity* protection.

3 DPC Protocols

In this section we present our three DPC protocols (whose properties are detailed in Sect. 1). Our first protocol π_1 falls short of satisfying all requirements formulated in Sects. 1 and 2 in two ways: (a) it does not satisfy compliance soundness (i.e., a malicious C could send inconsistent encryptions for compliance verification and query rewriting; thus, the compliance verification test would pass on a query value different than the one used for query rewriting); (b) privacy against D is only satisfied if the protocol is allowed to leak any repeated occurrences of the same query value. Our second protocol π_2 extends π_1 so to eliminate (a), and
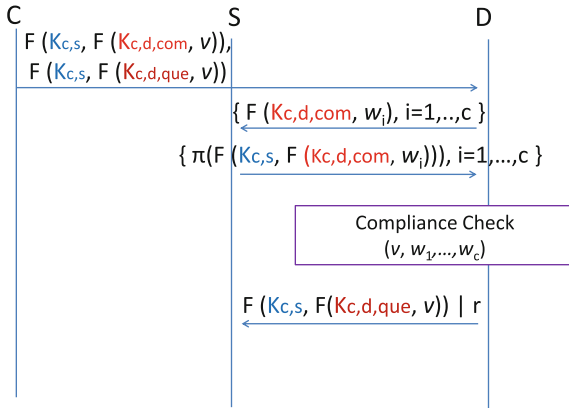
Fig. 3. The basic keyword match policy compliance protocol π_1

protocol π_3 eliminates both (a) and (b). In the following protocol descriptions, keys are named with two subscripts indicating by which parties they are shared. For example, a private key shared by C and S would be named $k_{c,s}$. There may optionally be a third subscript *com* or *que*, to indicate whether the key is used for policy compliance checking or query rewriting. Thus, $k_{c,d,com}$ means a key shared by C and D and used for compliance checking. We assume a standard secure 2-party key agreement protocol is executed in an initialization phase to produce these keys.

Protocol π_1. Our most basic protocol π_1 allows efficient enforcement of policy compliance for keyword search queries with whitelist policies (blacklist policies can be supported with minor modifications).

A pictorial description of the protocol can be found in Fig. 3. In the first step, C sends to D two double encryptions of its query value v, once using key $k_{c,d,que}$ as the inner layer, and a second time using key $k_{c,d,com}$. Then, D and S interact to analogously compute ciphertexts for the policy values, as follows: first, D encrypts each of the policy values $w_1, \ldots, w_c$ using key $k_{c,d,com}$ and sends the resulting ciphertexts to S; then, S further encrypts each of these ciphertexts using key $k_{c,s}$, and returns the resulting ciphertexts, *reordered using a random permutation* π, to D. At this point, D computes the whitelist policy output by simply checking whether one or zero of the policy value ciphertexts is equal to the ciphertext received by C. After the policy compliance calculation, if the query is compliant, D simply forwards the received encryption $k_{c,d,que}$ to S, who can remove the outer layer of encryption and fulfill the query. Otherwise, D performs *query rewriting*, sending S a random value indistinguishable from a double-encrypted query.

As described in the introduction, the two main technical ideas embedded in this protocol are: (1) using "equality-preserving encryption" to allow D to calculate the policy output without revealing the policy values to S or C and without learning why the policy was or was not satisfied (i.e., which policy

value(s) w_i may have textually matched value(s) in the query); (2) using "query rewriting" to allow D to rewrite the query q obtained by C into a query q' which guarantees that the same database records match q and q' if q is compliant, or no records match q' otherwise, without S or C obtaining any additional information on which is the case.

Protocol π_2: Soundness Against Malicious Clients. One problem with protocol π_1 is that a malicious C can violate the soundness property by sending two different queries for compliance verification and query rewriting. Protocol π_2 prevents this attack with minimal modifications from π_1. As a preliminary observation, we see that since C only sends one query message, the only opportunity for C to provide malicious input is before the compliance verification subprotocol. This naturally leads us to examine ways in which we could modify protocol π_1 to require only one input from C. Note that we cannot use the same encryption for both compliance checking and query rewriting, since that would allow S to identify encrypted query values that match policy items it has seen during the setup phase.

We can resolve this by storage of a triple encryption $F(k_{c,d}, F(k_{c,s}, F(k_{c,d}, v)))$ of each database value, as in Added DR Property 2, instead of a single encryption $F(k_{c,d}, v)$, as in π_1. The structure of the protocol is similar as for π_1. At query time, C encrypts the query value with *both* of its keys and sends the resulting doubly-encrypted value $F(k_{c,s}, F(k_{c,d}, v))$ to D. Then D encrypts each of the policy values w_i using key $k_{c,d}$ and sends them to S, which then re-encrypts each of these using key $k_{c,s}$, randomly permutes the order of keywords, and returns the re-encrypted values to D.

As before, D checks the encrypted query for equality with the double-encrypted policy values. If the query is non-compliant, D sends to S a random query indistinguishable from a triple-encrypted real query; otherwise, D re-encrypts $F(k_{c,s}, F(k_{c,d}, v))$ using $k_{c,d}$, and sends the triple-encrypted value $F(k_{c,d}, F(k_{c,s}, F(k_{c,d}, v)))$ to S for its answer generation in the DR protocol. Note that the outermost layer of encryption prevents S from identifying whether the query matches policy items it had previously encrypted from D—thus eliminating the need for separate *com* and *que* encryptions.

The resulting DPC protocol π_2 inherits the same properties as π_1, plus compliance soundness under a different assumption on the method used to encrypt the database values in the DR protocol (namely, Added Property 2). The triply-encrypted database of Added DR Property 2 can be generated during the setup phase as follows. First, D encrypts all items in the database with $k_{c,d}$ and sends them to S, which re-encrypts them using $k_{c,s}$ and returns them to D. Then D adds a third layer of encryption, again using $k_{c,d}$, and sends the triply-encrypted database to S. This interaction between D and S may be expensive, as it involves every item in the database being encrypted and sent over the network three times; this may render this method undesirable to practitioners, especially when dealing with large databases. We address this issue as well in π_3.

Protocol π_3: Privacy Across Multiple Queries. Protocols π_1, π_2 come with some leakage to D across multiple query executions: D learns, by checking for
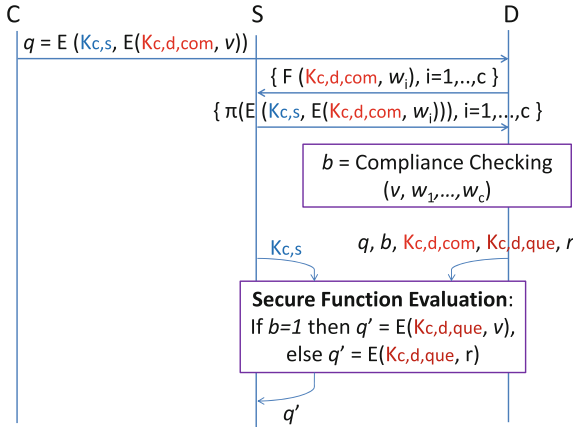
Fig. 4. Protocol π_3: Keyword search policy compliance with (multi-query) security against D

repetitions in the first message sent by C to D, whether the query value in the current execution is equal to a previously executed query. Although not a major form of leakage, it remains of interest to see if we can prevent it at some affordable efficiency cost.

We now describe a protocol π_3 that keeps all properties in π_1, the compliance soundness property achieved in π_2 and satisfies privacy against D without the mentioned leakage. (It also avoids the database setup inefficiency mentioned at the end of the description of π_2.) Protocol π_3 uses an additional cryptographic tool: 2-party Secure Function Evaluation (SFE) protocols [19]. Recall that in such protocols, two parties P_1 and P_2, with private inputs x_1 and x_2, respectively, can jointly compute a functionality $f(x_1, x_2) = (f_1, f_2)$, such that P_1 receives $f_1(x_1, x_2)$, and P_2 receives $f_2(x_1, x_2)$, and it is required that nothing is learned by either party other than the output.

Instead of using a triple encryption as in π_2, protocol π_3 uses a different shared key $k'_{c,d}$ for query rewriting. After the policy check, which remains unchanged, D and S perform a two-party SFE protocol, returning to S a newly-encrypted form of the query.

First, C sends $F(k_{c,s}, F(k_{c,d}, v))$ to D, at which point the same policy compliance check as in π_2 takes place. After the compliance check, D and S engage in a two-party SFE protocol, where D inputs $F(k_{c,s}, F(k_{c,d}, v)), k_{c,d}, k'_{c,d}$, a random query r, and the (one-bit) result of the compliance check. S inputs $k_{c,s}$. Together, the two parties securely compute the following output, which is received only by S: if the query was non-compliant, random value r is output; if the query was compliant, the doubly-encrypted value $F(k_{c,s}, F(k_{c,d}, v))$ is decrypted twice to produce v, which is then encrypted using key $k'_{c,d}$, and the result, $F(k'_{c,d}, v)$ is released to S. S then proceeds to compute the answer based on the DR protocol.

The resulting DPC protocol π_3, illustrated in Fig. 4, inherits the same properties as π_1 and π_2, plus multi-query privacy against D. However, π_3 is not strictly better than π_1, π_2 since two-party SFE protocols come with added running time, even when considering recent implementation advances (see, e.g., [15] and follow-up work). Accordingly, we only used two-party SFE executions on very short ℓ-bit inputs (as opposed to a generic SFE solution which would require inputs as large as the policy itself).

4   Performance Results

In this section we report initial performance results related to implementations of our basic DPC protocols. We focus on results for π_1 as π_2 and π_3 exhibit similar behaviour. Specifically, π_2 is only slower than π_1 by a small constant multiplicative factor and π_3 is only slower than π_2 by a small constant additive factor.

Setup. The Data Owner and Server processes were running on a Dell PowerEdge R710 server with two Intel Xeon 2.66 Ghz processors and 48 GB of memory, running 64-bit Ubuntu 12.04.1. The R710 server was connected to a Dell Power-Vault MD1200 disk array containing 12 2TB 7.2 K RPM SAS drives arranged in a RAID6 configuration. The Client was running on a Dell PowerEdge R810 server with two Intel Xeon 2.40 GHz processors and 64 GB of memory, running 64-bit Red Hat Enterprise Linux Server release 6.3 and connected to the R710 server via switched Gigabit Ethernet.

The database was populated by generating random values about fictitious people using demographic information from the US Census Bureau. A single table with 23 columns was used (e.g., last name *lname*, state *state*, and zip code *zip*), including several columns containing large text fields and one column containing binary data (*fingerprint*). We considered the following policies:

Policy	Compliant queries must include:
F	All queries are rejected as non compliant
T	All queries are accepted as compliant
B1	A conjunction of at least 3 keyword queries on *state*, *lname*, and *zip*
B2	A conjunction of at least 3 keyword queries on *state*, *lname*, and any one of the remaining columns, excluding *fingerprint*
W1	A keyword query on *lname* with query value in a 1-entry whitelist
W2	A keyword query on *lname* with query value in a 100-entry whitelist
W3	A keyword query on *lname* with query value in a 1000-entry whitelist
W4	A keyword query on *lname* with query value in a 10000-entry whitelist
W5	A keyword query on *lname* with query value in a 20000-entry whitelist

Compliance policy $B2$ was expressed as a disjunction of 23 sub-policies of $B1$ type, each of them requiring keyword query conjunctions on *state*, *lname*, and an additional (and different) database column. We considered the following queries:

Query	Template
Q1	SELECT * FROM main WHERE lname=value
Q2	SELECT * FROM main WHERE state=value AND lname=value AND zip=value

Results. Each query template was executed several times using different values. We note that policies F, T, $B1$ and $B2$ only refer to the query structure or database attributes and do not depend on query values, contrarily to queries $W1, \ldots, W5$, which depend on values in the query and in the (variable-length) whitelist.
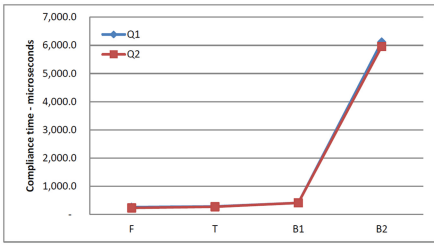
Fig. 5. Query compliance checking overhead for policies F, T, $B1$, and $B2$.
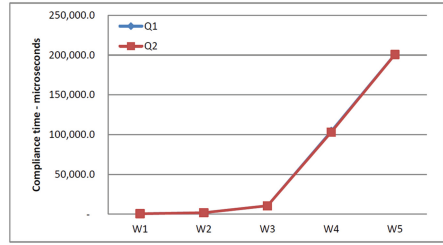
Fig. 6. Query compliance checking overhead for policies $W1$, $W2$, $W3$, $W4$, and $W5$.

Figure 5 shows results when checking compliance for policies F, T, $B1$, and $B2$ for $Q1$ and $Q2$ queries. Such checking was based on the query structure only and, thus, there is no impact from cryptographic operations on the measured running time. Three main observations can be derived from this figure: (1) running time for $B1$ is almost the same as for the trivial policies T and F; (2) running time for $B1$ and $B2$ is almost the same for the two policy types $Q1$ and $Q2$, with differences smaller than 3 %; and (3) running time for $B1$ and $B2$ is essentially linear with policy size.

Figure 6 shows computation results when running protocol π_1 for query classes $Q1$ and $Q2$ and policies $W1, \ldots, W5$. These policies depend on query values and, hence, trigger execution of π_1, with its cryptographic operations. Two main observations can be derived from this figure. First, running time is almost the same again for the two policy types $Q1$ and $Q2$, with differences of less than 7 % for shorter policies and less than 2 % as policies get longer. Second, the time required by π_1 grows linearly with the size of the whitelist. Specifically,

as the size of the whitelist grows, so does the time it takes to doubly encrypt its values, send/receive them between D and S, and checking by using sequential scan whether an attribute value referenced in C's query belongs to the doubly encrypted and permuted whitelist values. (Here, a speedup from the use of binary search does not seem to impact the running time substantially, due to the double encryption and network communication required).

When comparing the two figures, we observe that the impact of running π_1 when checking compliance is essentially minimal for policies with short-size whitelists (i.e., a factor of about 10, calculated by comparing the running time for $F, T, B1$ with the running time of policies $W1, W2, W3$).

References

1. Beaver, D.: Commodity-based cryptography (extended abstract), pp. 446–455. In: STOC (1997)
2. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004)
3. Brickell, E., Di Crescenzo, G., Frankel, Y.: Sharing block ciphers. In: Clark, A., Boyd, C., Dawson, E.P. (eds.) ACISP 2000. LNCS, vol. 1841, pp. 457–470. Springer, Heidelberg (2000)
4. Camenisch, J., Kohlweiss, M., Rial, A., Sheedy, C.: Blind and anonymous identity-based encryption and authorised private searches on public key encrypted data. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 196–214. Springer, Heidelberg (2009)
5. Ceselli, A., Damiani, E., De Capitani di Vimercati, S., Paraboschi, S.: Modeling and assessing inference exposure in encrypted databases. ACM TISSEC **8**, 119–152 (2005)
6. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM **45**(6), 965–981 (1998)
7. Di Crescenzo, G., Ishai, Y., Ostrovsky, R.: Universal service-providers for database private information retrieval, pp. 91–100. In: PODC (1998)
8. Evdokimov, S., Günther, O.: Encryption techniques for secure database outsourcing. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 327–342. Springer, Heidelberg (2007)
9. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM **43**(3), 431–473 (1996)

10. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: ACM CCS Conference, pp. 89–98 (2006)
11. Hacigümüs, H., Iyer, B.R., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database-service-provider model, pp. 216–227. In: SIGMOD Conference (2002)
12. Hamlen, K.W., Kagal, L., Kantarcioglu, M.: Policy enforcement framework for cloud data management. IEEE Data Eng. Bull. **35**(4), 39–45 (2012)
13. Jarecki, S., Lincoln, P.: Negotiated privacy. In: Okada, M., Babu, C.S., Scedrov, A., Tokuda, H. (eds.) ISSS 2002. LNCS, vol. 2609, pp. 96–111. Springer, Heidelberg (2003)
14. Li, M., Yu, S., Cao, N., Lou, W.: Authorized private keyword search over encrypted data in cloud computing, pp. 383–392. In: ICDCS (2011)
15. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - secure two-party computation system, pp. 287–302. In: USENIX Security Symposium (2004)
16. Miklau, G., Suciu, D.: Controlling access to published data using cryptography, pp. 898–909. In: VLDB (2003)
17. Song, D., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data, pp. 44–55. In: IEEE Symposium on Security and Privacy (2000)
18. Yang, Z., Zhong, S., Wright, R.N.: Privacy-preserving queries on encrypted data. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 479–495. Springer, Heidelberg (2006)
19. Yao, A.C.C.: How to generate and exchange secrets (extended abstract), pp. 162–167. In: FOCS (1986)