Chapter 3

PRIVATE MATCHING

Yaping Li
UC Berkeley

J. D. Tygar
UC Berkeley

Joseph M. Hellerstein
UC Berkeley
Intel Research Berkeley

Abstract Consider two organizations that wish to privately match data. They want to find common data elements (or perform a join) over two databases without revealing private information. This was the premise of a recent paper by Agrawal, Evfimievski, and Srikant. We show that Agrawal et al. only examined one point in a much larger problem set and we critique their results. We set the problem in a broader context by considering three independent design criteria and two independent threat model factors, for a total of five orthogonal dimensions of analysis.

Novel contributions include a taxonomy of design criteria for private matching , a secure data ownership certificate that can attest to the proper ownership of data in a database, a set of new private matching protocols for a variety of different scenarios together with a full security analysis. We conclude with a list of open problems in the area.

1. Introduction

Agrawal, Evfimievski, and Srikant recently presented a paper [Agrawal et al., 2003] that explores the following *private matching* problem: two parties each have a database and they wish to determine common entries without revealing any information about entries only found in one database. This paper has generated significant interest in the research community and technical press. While the Agrawal/Evfimievski/Srikant (AgES) protocol is correct within in its assumptions, it is not robust in a variety of different scenarios. In fact, in many likely scenarios, the AgES protocol can easily be exploited to obtain a great deal of information about another database. As we discuss in this paper, the private matching problem has very different solutions depending on assumptions about the different parties, the way they interact, and cryptographic mechanisms available. Our paper discusses flaws in the AgES protocol, presents that protocol in the context of a framework for viewing private matching and a family of possible protocols, and gives a number of new techniques for addressing private matching, including a flexible powerful Data Ownership Certificate that can be used with a variety of matching protocols.

The private matching problem is a practical, constrained case of the more general (and generally intractable) challenge of *secure multi-party computation* . Private set matching is a simple problem that is at the heart of numerous data processing tasks in a variety of applications. It is useful for relational equijoins and intersections, as well as for full-text document search, cooperative web caching, preference matching in online communities, and so on. Private matching schemes attempt to enable parties to participate in such tasks without worrying that information is leaked.

In this paper we attempt a holistic treatment of the problem of two-party private matching. We lay out the problem space by providing a variety of possible design goals and attack models. We place prior work in context, and present protocols for points in the space that had been previously ignored. We also point out a number of additional challenges for future investigation.

1.1 Scenarios

We begin our discussion with three scenarios, which help illustrate various goals of a private matching protocol.

Our first scenario comes from multi-party customer relationship management in the business world. Two companies would like to identify their common customers for a joint marketing exercise, without divulging any additional customers. In this scenario, we would like to ensure that (a) neither party learns more than their own data and the answer (and anything implied by the pair), and (b) if one party learns the results of the match, both parties should learn it. Agrawal, et al. discuss a special instance of this case in their work [Agrawal et al., 2003], which they call *semi-honesty* , after terminology used in secure multi-party literature [Goldreich, 2002]. In particular, the two companies are assumed to honestly report their customer lists (or, more generally, the lists they wish to intersect), but may try otherwise to discover additional information about the other's customer list. The semi-honest scenario here rests on the presumption that a major corporation's publicity risk in being detected lying outweighs its potential benefit in one-time acquisition of competitive information. Below, we comment further on difficulties raised by this notion of semi-honesty.

In many cases, we do not desire symmetric exchange of information. As a second example, consider the case of a government agency that needs to consult a private database. Privacy and secrecy concerns on the part of the government agency may lead it to desire access to the private database without revealing any information about the nature of the query. On the other hand, the database owner may only want to release information on a “need-to-know” basis: it may be required by law to release the answers to the specific query, but may be unwilling to release any other information to the government. In short, a solution to the situation should enable the government to learn only the answer to its query, while the database owner will learn nothing new about the government. In this asymmetric scenario, we need a different choice than (b) above.

Finally, we consider a scenario that could involve anonymous and actively dishonest parties. Online auction sites are now often used as a sales channel for small and medium-sized private businesses. Two competing sellers in an online auction site may wish to identify and subsequently discuss the customers they have in common. In this case, anonymity of the sellers removes the basis for any semi-honesty assumption, so guaranteed mechanisms are required to prevent one party from tricking the other into leaking information.

Each of these examples has subtly different design requirements for a private matching protocol. This paper treats these examples by systematically exploring all possible combinations of security requirements along a number of independent design criteria.

1.2 Critique of AgES

In their paper [Agrawal et al., 2003], Agrawal, Evfimievski, and Srikant consider the first scenario listed above, building on an earlier paper by Huberman et al. [Huberman et al., 1999]. Here is an informal summary of the AgES Set Intersection Protocol result; we discuss it more formally below in Section 3.

Agrawal, et al. suggest solving the matching problem by introducing a pair of encryption functions E (known only to A) and E' (known only to B) such that for all x, $E(E'(x)) = E'(E(x))$. Alice has customer list A and Bob has customer list B. Alice sends Bob the message $E(A)$; Bob computes and then sends to Alice the two messages $E'(E(A))$ and $E'(B)$. Alice then applies E to $E'(B)$, yielding (using the commutativity of E and E') these two lists: $E'(E(A))$ and $E'(E(B))$. Alice computes $E'(E(A)) \cap E'(E(B))$. Since

Alice knows the order of items in A, she also knows the the order of items in $E'(E(A))$ and can quickly determine $A \cap B$.

Two main limitations are evident in this protocol. First, it is *asymmetric*: if we want both parties to learn the answer, we must trust Alice to send $A \cap B$ to Bob. This asymmetry may be acceptable or even desirable in some scenarios, but may be undesirable in others.

Second, we find the AgES assumption of *semi-honesty* to be hard to imagine in a real attack scenario. Any attacker who would aggressively decode protocol messages would presumably not hesitate to "spoof" the contents of their queries. If we admit the possibility of the attacker spoofing queries, then the AgES protocol is not required; a simpler hash-based scheme suffices. In this scheme (also suggested by Agrawal, et al.) the two parties hash the elements of their lists $h(A)$ and $h(B)$ and then compute the intersection of those two lists of hashes. Later in this paper, we augment this hash-based protocol with an additional mechanism to prevent spoofing as well.

1.3 A broader framework

Below, we consider a broader framework for thinking about private matching.

First, we break down the protocol design space into three independent criteria :

Design criteria

- protocols that leak no information (*strong*) vs. protocols that leak some information (*weak*)
- protocols that protect against spoofed elements (*unspoofable*) vs. protocols that are vulnerable (*spoofable*).
- symmetric release of information vs. asymmetric release (to only one party).

We will also consider two different dimensions for threat models:

Threat models

- semi-honest vs. malicious parties
- small vs. large data domains

We discuss the design criteria in more detail in the next section and cover the threat models below in Section 3.

2. Problem Statement

We define the *private matching* problem between two parties as follows. Let the two parties Alice and Bob have respective sets A and B of objects in some domain D. Suppose Alice wants to pose a matching query $Q \subseteq D$ to Bob. We call Alice the *initiator* of the query and Bob the *recipient* of the query. We say Q is *valid* if $Q \subseteq A$ and *spoofed* otherwise. A *matching* computes $P = Q \cap B$ or $\perp$; note that $\perp$ is a message distinguishable from the set $\emptyset$, and can be thought of as a warning or error message.

We elaborate upon the three design criteria for private matching described in the previous section:

- We say that a matching protocol is *strong* if any party can learn only: P, any information that can be derived from P and this party's input to the protocol, the size of the other party's input, and nothing else; otherwise the protocol is *weak* with respect to the additional information learnable.

- We define a matching protocol to be *unspoofable* if it returns $\perp$ or $Q \cap A \cap B$ for all spoofed Q. Otherwise it is *spoofable*.

- We say that a matching protocol is *symmetric* if both parties will know the same information at any point in the protocol. Otherwise it is *asymmetric*.

For each of these three dimensions, a bit more discussion is merited. We begin with the *strong/weak* dichotomy. After executing a protocol, a party can derive information by computing functions over its input to the protocol and the protocol's output. An example of such derived information is that a party can learn something about what is *not* in the other party's set, by examining its input and the query result. Since any information that can be computed in this way is an unavoidable consequence of matching, we use P to denote both P and the derived information throughout our paper. Note that weak protocols correspond to the notion of semi-honesty listed above — weak protocols allow additional information to be leaked, and only make sense when we put additional restrictions on the parties — typically, that they be semi-honest. In contrast, strong protocols allow malicious parties to exchange messages. Note that we allow the size of a party's input to be leaked; the program of each party in a protocol for computing a desired function must either depend only on the length of the party's input or obtain information on the counterpart's input length [Goldreich, 2002].

For the *spoofable/unspoofable* dimension, there are scenarios where a protocol that is technically spoofable can be considered effectively to be unspoofable. To guarantee that a protocol is unspoofable, it requires the protocol to detect spoofed queries. Given such a mechanism, either of the following two responses are possible, and maintain the unspoofable property: (a) returning $\perp$, or (b) returning $Q \cap A \cap B$. When a party lacks such a detection mechanism, it cannot make informed decision as when to return $\perp$. However, in some situations, the party may be expected to return the set $Q \cap A \cap B$ with high probability, regardless of whether the query is spoofed or not. This may happen when it is very difficult to spoof elements. We will give an example of this scenario later.

It is also useful to consider the the issue of symmetry vs. asymmetry for the threat models covered in Section 3. In the semi-honest model, parties follow the protocols properly, and so symmetry is enforced by agreement. However, in a malicious model, the parties can display arbitrary adversarial behavior. It is thus difficult to force symmetry, because one party will always receive the results first. (A wide class of cryptographic work has revolved around "fair exchanges" in which data is released in a way that guarantees that both parties receive it, but it is not clear if those concepts could be efficiently applied in the private matching application.)

2.1 Secure multi-party computation

The private matching problem is a special case of the more general problem from the literature called *secure multi-party computation*. We now give a brief introduction to secure multi-party computation in the hope of shedding light on some issues in private matching. In a secure m-party computation, the parties wish to compute a function f on their m inputs. In an *ideal model* where a trusted party exists, the m parties give their inputs to the trusted party who computes f on their inputs and returns the result to each of the parties. The results returned to each party may be different. This ideal model captures the highest level of security we can expect from multi-party function evaluation [Canetti, 1996]. A secure multi-party computation protocol emulates what happens in an ideal model. It is well-known that no secure multi-party protocol can prevent a party from cheating by changing its input before a protocol starts [Goldreich, 2002]. Note however, that this cannot be avoided in an ideal model either. Assuming the existence of trapdoor permutations, one may provide secure protocols for any two-party computation [Yao., 1986] and for any multi-party computation with honest-majority [Goldreich et al., 1987]. However, multi-party computations are usually extraordinarily expensive in practice, and impractical for real use. Here, our focus is on *highly efficient* protocols for private matching, which is both tractable and broadly applicable in a variety of contexts.

3. Threat Models

We identify two dimensions in the threat model for private matching. The first dimension concerns the domain of the sets being matched against. A domain can be *small*, and hence vulnerable to an exhaustive search attack , or *large*, and hence not vulnerable to an exhaustive search attack.

If a domain is small, then an adversary Max can enumerate all the elements in that domain and make a query with the entire domain to Bob. Provided Bob answers the query honestly, Max can learn the entirety of Bob's set with a single query. A trivial example of such a domain is the list of Fortune 500 companies; but note that there are also somewhat larger but tractably small domains like the set of possible social security numbers.

A large uniformly distributed domain is not vulnerable to an exhaustive search attack. We will refer to this type of domain simply as *large* in this paper. An example of such a domain is the set of all RSA keys of a certain length. If a domain is large, then an adversary is limited in two ways. First, the adversary cannot enumerate the entire domain in a reasonable single query, nor can the adversary repeatedly ask smaller queries to enumerate the domain. In this way the adversary is prevented from mounting the attack described above. Second, it is difficult for her to query for an arbitrary individual value that another party may hold, because each party's data set is likely to be a negligible-sized subset of the full domain.

The second dimension in the threat model for private matching captures the level of adversarial misbehavior. We distinguish between a semi-honest party and a malicious party [Goldreich, 2002]. A semi-honest party is honest on its query or data set and follows the protocol properly with the exception that it keeps a record of all the intermediate computations and received messages and manipulates the recorded messages in an aggressively adversarial manner to learn additional information.[1] A malicious party can misbehave in arbitrary ways: in particular, it can terminate a protocol at arbitrary point of execution or change its input before entering a protocol. No two-party computation protocol can prevent a party from aborting after it receives the desired result and before the other party learns the result. Also no two-party computation protocol can prevent a party from changing its input before a protocol starts.

Hence we have four possible threat models: a semi-honest model with a small or large domain, and a malicious model with a small or large domain. In the rest of the paper, we base our discussion of private matching protocols in terms of these four threat models.

3.1 Attacks

In this section we enumerate a number of different attacks that parties might try to perform to extract additional information from a database. In the scenarios below, we use the notation A and B to denote parties, and A is trying to extract information from B's database.

- **Guessing attack:** In this attack, the parties do not deviate from the protocol. However, A attempts to guess values in B's database and looks for evidence that those values occur in B's database. Typically, A would guess a potential value in B's database, and then look for an occurrence of the hash in B's database. Alternatively, A could attempt to decrypt values in a search for an encrypted version of a particular potential value in B's database (following the pattern in the AgES protocol.) Because of the limitations of this type of attack, it is best suited when the domain of potential values is small. (A variant of this attack is to try all potential values in the domain, an *exhaustive search attack.*)
- **Guess-then-spoof attack:** In this attack, the parties deviate from the protocol. As in the guessing attack, A generates a list of potential values in B's database. In the spoofing attack , A runs through the protocol pretending that these potential values are already in A's database. Thus A will compute hashes or encrypt, and transmit values as if they really were present in A's database. Because this attack involves a guessing element, it is also well suited for small domains of potential database values (e.g. social security numbers, which are only 10 digits long).
- **Collude-then-spoof attack:** In this attack, A receives information about potential values in B's database by colluding with outside sources. For example, perhaps A and another database owner C collude by exchanging their customer lists. A then executes a spoofing attack by pretending that these entries are are already on its list. As in guess-then-spoof attack, A computes hashes or encrypts, and transmits values as if they were really present in A's database. Since A is deriving its information from third party sources in this attack, it is suited for both small and large domains of potential database values. (N.B.: we group both the guess-then-spoof attack and the collude-then-spoof attack together as instances of *spoofing attacks*. Spoofing attacks occur in the malicious model; in the semi-honest model they can not occur.)
- **Hiding attacks:** In a hiding attack, A only presents a subset of its customer list when executing a matching protocol, effectively hiding the unrevealed members. This paper does not attempt to discuss defenses against hiding attacks.

Although we would like to prevent all collusion attacks involving malicious data owners, there are limits to what we can accomplish. For example, if Alice and Bob agree to run a matching protocol, nothing can prevent Bob from simply revealing the results to a third party Charlie. In this case, Bob is acting as a proxy on behalf of Charlie, and the revelation of the results occurs out-of-band from the protocol execution. However, we would like to resist attacks where Bob and Charlie collude to disrupt the protocol execution or use inputs not otherwise available to them.

4. Terminology and Assumptions

We begin by assuming the existence of *one-way collision resistant hash functions* [Menezes et al., 1996] . A hash function $h(\cdot)$ is said to be one-way and collision resistant if it is difficult to recover M given $h(M)$, and it is difficult to find $M' \neq M$ such that $h(M') = h(M)$. Let $\textsc{Sign}(\cdot,\cdot)$ be a public key signing function which takes a secret key and data and returns the signature of the hash of the the data signed by the secret key. Let $\textsc{Verify}(\cdot,\cdot,\cdot)$ be the corresponding public key verification function which takes a public key, data, and a signature and returns $\mathtt{true}$ if the signature is valid for the data and $\mathbf{false}$ otherwise. For shorthand, we denote $\{P\}_{sk}$ as the digital signature signed by the secret key sk on a plaintext P. The function $isIn(\cdot,\cdot)$ takes an element and a set and returns $\mathtt{true}$ if the element is in the set and $\mathbf{false}$ otherwise.

The *power function* f : Key$\mathcal{F}$ $\times$ Dom$\mathcal{F}$ $\rightarrow$ Dom$\mathcal{F}$ where f defined as follows:

$$f_e(x) \equiv x^e \bmod p$$

is a *commutative encryption* [Agrawal et al., 2003]:

- The powers commute:

$$(x^d \bmod p)^e \bmod p \equiv x^{de} \bmod p \equiv (x^e \bmod p)^d \bmod p$$

- Each of the powers f_e is a bijection with its inverse being $f_e^{-1} \equiv f_{e^{-1} \bmod q}$.

where both p and $q = (p-1)/2$ are primes.

We use the notation $e \overset{r}{\leftarrow} S$ to denote that element e is chosen randomly (using a uniform distribution) from the set S.

We assume there exists an encrypted and authenticated communication channel between any two parties.

1 Alice's local computation:

   (a) $Q_h := \{h(q) : q \in Q\}$.

   (b) $e_A \overset{r}{\leftarrow} Key\mathcal{F}$.

   (c) $Q_{e_A} := \{f_{e_A}(q_h) : q_h \in Q_h\}$.

2 Bob's local computation:

   (a) $B_h := \{h(b) : b \in B\}$,

   (b) $e_B \leftarrow_r Key\mathcal{F}$.

   (c) $B_{e_B} := \{f_{e_B}(b_h) : b_h \in B_h\}$.

3 Alice$\rightarrow$ Bob: Q_{e_A}.

4 Bob's local computation:
$Q_{e_A,e_B} := \{(q_{e_A}, f_{e_B}(q_{e_A})) : q_{e_A} \in Q_{e_A}\}$.

5 Bob$\rightarrow$ Alice: B_{e_B}, Q_{e_A,e_B}.

6 Alice's local computation:

   (a) $Q^{'}_{e_A,e_B} := \emptyset$, $P := \emptyset$

   (b) $B_{e_B,e_A} := \{f_{e_A}(b_{e_B}) : b_{e_B} \in B_e\}$.

   (c) For every $q \in Q$, we compute $q_{e_A} = f_{e_A}(h(q))$, and find the pair $(q_{e_A}, q_{e_A,e_B}) \in Q_{e_A,e_B}$; given this we let $Q'_{e_A,e_B} := Q'_{e_A,e_B} \cup \{(q, q_{e_A,e_B})\}$.

   (d) For every $(q, q_{e_A,e_B}) \in Q^{'}_{e_A,e_B}$, $\mathbf{if\ \ isIn(q_{e_A,e_B}, B_{e_B,e_A}),\ \ thenP := P \cup \{q\}}$.

7 Output P.

Figure 3.1. AgES protocol

5. Techniques

We present three matching protocols in this section: the trusted third party protocol , the hash protocol , and the AgES protocol [Agrawal et al., 2003]. In the next section, we describe a data ownership certificate mechanism that can be combined with all three protocols to *despoof* all of the original protocols even in threat models with small domains.

5.1 Trusted Third Party Protocol (TTPP)

Suppose Alice and Bob trust a third party Trudy. Alice and Bob can compute their private matching through Trudy. Alice (resp. Bob) sends her query Q (resp. his data set B) to Trudy, and Trudy computes the intersection P of the two sets. Trudy then returns the result to both parties in the symmetric case, or to one of the parties in the asymmetric case.

We discuss the security of the TTPP in Section 7.

5.2 Hash Protocol (HP)

In this section, we present a *Hash Protocol* that do not require a trusted third party . In the hash protocol, Alice sends Bob her set of hashed values. Bob hashes his set with the same hash function, and computes the intersection of the two sets. Bob may send Alice the result based on their prior agreement.

We discuss the security of the hash protocol in Section 7.

5.3 The AgES protocol

We gave a summary of the AgES protocol in Section 1.2. Now we present the complete version of the protocol in Figure 3.1. For consistency we adapt this protocol to our notation, but the essence of the protocol remains the same as the original paper.

We discuss the security of the AgES protocol in Section 7.

6. Data Ownership Certificate (DOC)

An especially difficult attack for private matching to handle is the spoofing problem. In this section, we propose a new approach to address spoofing: the use of *Data Ownership Certificates*. The idea is to have the creator of data digitally sign the data in a particular way so that parties that control databases that include the data can not spoof data. For example, consider the case of two companies each of which wants to find out as much as possible about the other's customer list. If one of the companies has access to a list of all residents in a particular area, a straightforward spoofing attack is quite simple — it could simply create false entries corresponding to a set of the residents. If any of those residents were on the other company's customer list, private matching would reveal their membership on that list. However, if the companies are obligated to provide digitally signed entries, this type of spoofing would be eliminated: neither of the companies would be able to falsify entries.

The above sketch is not sufficient, however, because it still leaves open the possibility that corrupt companies could broker in digitally signed data entries. For example, if customer E is a legitimate customer of firm F, we would have the possibility that F might try to trade or sell G's digitally signed entry to A. Then A would be able to falsely claim that G was a customer and during private matching, steal information through a spoofing attack. Below, we discuss an architecture for data ownership certificates that resists both regular spoofing attacks and colluding spoofing attacks .

Data Ownership Certificates do require more work on the part of individuals creating data, and they are probably only practical in the case of an individual who uses his or her computer to submit information to a database. Despite the extra work involved, we believe that data ownership certificates are not far-fetched. In particular, the European Union's Privacy Directive [Parliament, 1995] requires that individuals be able to verify the correctness of information about them and control the distribution of that information. Data Ownership Certificates give a powerful technical mechanism supporting that distribution. Similarly, Agrawal, Kieran, Srikant, and Xu have recently argued for a type of "Hippocratic Database" that would provide similar functionality [Agrawal et al., 2002]. Data Ownership Certificates would work well with these Hippocratic Databases.

Now we begin a formal presentation of Data Ownership Certificates (DOC). A Data Ownership Certificate is an authorization token which enables a set owner to prove it is a legitimate owner of some particular data. The first goal of the DOC is to prevent spoofing in a small domain. Data Ownership Certificates prevent spoofing by 'boosting" the size of the small domain D to a larger domain $D \times S$, where S is the domain of the DOCs. The intuition is that by expanding the domain, DOCs make the probability of guessing a correct value negligible in the cryptographic sense and protect database owners from guess-then-spoof attacks. Now, if an attacker wants to spoof a particular value, e.g. John's information, the attacker needs to correctly guess the associated DOC as well.

A second goal of Data Ownership Certificates is access control. A DOC is essentially a non-transferable capability issued by the originator of data to a database owner. We refer to the originators of data as *active entities*. We say that an active entity E *authorizes* a set owner O *sharing access* to its information d when E issues O a DOC C_d^O for d. Ideally, a common element between two databases should be discovered only when both databases have been authorized with DOCs by the corresponding active entity for that element. More precisely, we require two security properties from Data Ownership Certificates:

- Confidentiality: If Bob is not an authorized owner of d, Bob should not be able to learn that Alice possesses d if he runs a matching protocol directly with Alice.
- Authenticity: If Bob is not an authorized owner of d and Alice is an authorized owner of d, Bob should not be able to pollute Alice's matching result, i.e., Bob cannot introduce d into the matching result.

We find that confidentiality is difficult to achieve. We thought of two approaches to do the access control. First, Alice checks whether Bob has the authorization before she gives an element v to Bob. It seems essential that Alice obtains some knowledge k that links the access controlled object v to requester Bob before granting the access. This requester-specific knowledge k reveals at least partial information of what element Bob has. It is then only fair that Bob checks for Alice's permission to access k. This leads to an infinite reduction. Second, Alice can give Bob a box which contains John's information d. The box is locked by John. Bob can only open the box if he has the key. This implies that John uses a lock for which he knows Bob has the key. This kind of precomputation on John's part is not desirable. We leave this as an open problem for future work and we relax our requirement for access control in this paper. We propose a third solution that allows two parties to learn their common element d if both of them have d and some *common nonce* for d instead of some requester specific access token. We refer to the goal of DOC as *reduced confidentiality requirement*.

6.1 Our instantiation of DOCs

Our instantiation of Data Ownership Certificates consists of two parts: a common nonce (random string) and an ownership attestation component. The common nonce serves the purpose of both boosting the domain and satisfying the reduced confidentiality requirement. The ownership attestation component satisfies the authenticity requirement.

A Data Ownership Certificate C has the form of $\langle pk, n, \sigma \rangle$. Each active entity E maintains three keys k_1, sk, and pk. For each piece of information d originating from E, E generates a unique $n = G(k_1||d)$ where $G(\cdot)$ is a pseudo-random number generator and $||$ is the concatenation function. Assume that the output n of $G(\cdot)$ is l bits long and $G(\cdot)$ is cryptographically secure, then by the birthday paradox, one needs to guess approximately $\sqrt{2^l}$ numbers to have one of them collide with n. If l is large enough, say 1024, then guessing the correct n is hard. This nonce n will be used in matching protocols instead of the original data d.

When E submits d to some database A, it generates a signature $\sigma = \{d||A\}_{sk}$ where A is the unique ID of the database. The signature does not contain the plaintext information d or A, however anyone knowing the public pk and the plaintext information d and A may verify that A is indeed an authorized owner of d by verifying the authenticity of σ using pk.

6.2 Certified matching protocols

In this section, we describe the integration of Data Ownership Certificates with the proposed protocols from Section 5.

We assume that each set element in database A is a pair $\langle d, C\rangle$ of data and a Data Ownership Certificate $C = \langle pk, n, \sigma\rangle$ where $\sigma = \{d||A\}_{sk}$. The owner of database A now runs a matching protocol with n instead of d as the data.

6.2.1 Certified Trusted Third Party Protocol (CTTPP). We describe how to use Data Ownership Certificates to extend the Trusted Third Party Protocol . Let A (resp. B) be the ID of Alice (resp. Bob). The set that Alice (resp. Bob's) sends to Trudy contains elements in the form of (n_a, σ_a, pk_{n_a}) (resp. (n_b, σ_b, pk_{n_b})), i.e., triples of a common nonce, ownership attestation component, and the corresponding public key. The nonce n_a (resp. n_b) is associated with elements a (resp. b).

When Trudy finds a matching between two common nonces n_a and n_b, she compares the corresponding public keys pk_{n_a} and pk_{n_b}. If they are not the same, then it means that Alice and/or Bob spoofed the element and forged the corresponding certificate. Trudy cannot tell which is the case and she simply returns $\perp$ to both of them. If the corresponding public keys are the same, Trudy runs the verification algorithm on Alice's and Bob's ownership attestation component $\textsc{Verify}(pk_{n_a}, a||A, \sigma_a) = v_2$ and $\textsc{Verify}(pk_{n_b}, b||B, \sigma_b) = v_2$ to check whether Alice and/or Bob are authorized owners of the matching value. Trudy will find one of the following three cases to be true:

1 $v_1 = \mathbf{true}$ and $v_2 = \mathbf{true}$

2 $v_1 = \mathbf{true}$ and $v_2 \neq \mathbf{true}$ or vice versa

3 $v_1 \neq \mathbf{true}$ and $v_2 \neq \mathbf{true}$

If Trudy encounters case (1), then she concludes Alice and Bob are the authorized owners of the matching element. She adds the element to the result set and continues with the matching computation. We show why this is the case. Suppose only Bob is the authorized owner of the element associated with n_b. It is unlikely that Alice spoofs the common nonce n_a where $n_a = n_b$ as discussed in Section 6.1. Suppose Alice obtains n_a and the associated DOC for some other database owner, it is highly unlikely that Alice can generate a public/private key pair that is the same as the key pair for n_b. By symmetry, it is highly unlikely to be the case that Alice is the authorized owner of the element associated with n_a and Bob spoofs n_b or the public/private key pair. If (2) or (3) is the case, it implies Alice and/or Bob spoofed the nonce and an associated DOC or obtained her/his element from some other authorized owner(s) and spoofed a DOC. Trudy returns $\perp$ for this case.

If Alice (resp. Bob) did not pose a spoofed query and receives $\perp$ from Trudy, then she (resp. he) knows that the other party was not honest.

6.2.2 Certified Hash Protocol (CHP). The integration of data ownership certificates with the Hash Protocol is slightly different from that with the Trusted Third Party Protocol . We assume that Alice poses a query Q_h each element of which is in the form of $\langle h(n_a), \sigma_a \rangle$ where $\sigma_a = \{a||A\}_{sk_a}$.

Bob hashes each of his common nonces and checks if it matches one of $h(n_a)$. If he discovers a match between $h(n_a)$ and $h(n_b)$, then he assumes that the two corresponding ownership attestation components were signed by the same private key and does the following check. Bob first looks up his copy of the public key pk_b for n_b and checks if $\text{VERIFY}(pk_b, b||A, \sigma_a)$ returns $\mathbf{true}$. If it does return $\mathbf{true}$, it means that Alice is an authorized owner of b. Bob may add b to the result set P and continue with his matching computation. Otherwise Bob can conclude that Alice is not the authorized owner of b — she either obtained $h(n_a)$ and the corresponding certificate from some other authorized owner of a or she was able to guess $h(n_a)$ and forged the ownership attestation component. Bob cannot tell which was the case. Now Bob has the following two options: (a) returning $\perp$ to Alice, or (b) continuing with the matching computation but omitting b from the final result. Either way the modified protocol satisfies the security goal of being unspoofable and it enables parties to detect cheating.

We need to be careful about the usage of hash functions in the Certified Hash Protocol. Consider the following two scenarios. In the first scenario, assume that Alice, Bob, and Charlie are authorized owners of some customer John's information d. Imagine Alice executes the Certified Hash Protocol with Bob and Charlie and she receives data from Bob and Charlie. If Bob and Charlie use the same hash function, e.g. MD5 or SHA1 , then Alice may infer that all three of them have d after the protocol executions with Bob and Charlie respectively. Alice hashes her own copy of the nonce n_d associated with d and discovers n_b is in the sets that Bob and Charlies sends to her. The second scenario is that both Bob and Charlie are authorized owners of d but Alice is not. Furthermore, assume Alice does not have a copy of d and its DOC from some other authorized owner. In this case, Alice may infer that Bob and Charlie share some common information although she does not know what it is.

We propose using an HMAC in the Hash Protocol to prevent the inference problem in the second scenario. An HMAC is a keyed hash function that is proven to be secure as long as the underlying hash function has some reasonable cryptographic strength [Bellare et al., 1996]. An $\mathrm{HMAC}_k(T) =$

$$h(k \oplus \text{ opad}, h(k \oplus \text{ipad}, T))$$

is a function which takes as inputs a secret key k and a text T of any length; "opad" and "ipad" are some predetermined padding. The output is an l-bit string where l is the output of the underlying hash function $h(\cdot)$.

Using HMAC in the Certified Hash Protocol avoids the problem in the second scenario as long as each pair of parties uses a different key every time they run the Certified Hash Protocol. This prevents adversaries from correlating elements from different executions of the Hash Protocol.

6.2.3 Certified AgES protocol (CAgES). We need to modify the AgES protocol in Figure 3.1 in three ways. First, both Alice and Bob hash and encrypt the common nonce instead of the actual data. Second, Bob returns pairs $\langle \sigma_b, f_{e_B}(h(n_b))\rangle$ for each of his encrypted elements $f_{e_B}(h(n_b))$. Third, whenever there is a match, Alice verifies whether Bob is an authorized owner by checking the corresponding σ_b.

6.3 Homomorphic DOC (HDOC)

The data ownership certificate as proposed is limited in a way that it introduces linear storage growth if authorized set owners wish to match a subset of the attribute values of an active entity's information. This *partial matching* property is desirable in many situations. For example, customer database A is an authorized owner of some customers' name, credit card number, and mailing address and customer database B is an authorized owner of the same customers' name, credit card number and email addresses. Suppose A and B wish to find out their common costumers by intersecting their respective set of credit card numbers. This cannot be done efficiently with our proposed DOC since A's (resp. B's) customers need to generate one DOC for their names, credit card numbers and mailing addresses (resp. email addresses) respectively. When a database has various information about a customer, the storage overhead can be quite high. In this section, we describe a *Homomorphic Data Ownership Certificate* scheme that allows a customer to generate *one* DOC for all of his or her information submitted to a database and still enables the databases to intersect certain attributes of customer information.

The semantics for a homomorphic data ownership certificate call for a malleable DOC scheme. Given a DOC C_S^O for S from an active entity E, we would like the set owner to generate a valid $\bar{C}_S^O$ for S' where $S' \subset S$ without the help of E.

Homomorphic signatures have the right property we are looking for. Let $\odot$ be a generic binary operator. Intuitively, a homomorphic signature scheme allows anyone to compute a new signature $\mathsf{Sig}(x \odot y)$ given the signatures $\mathsf{Sig}(x)$ and $\mathsf{Sig}(y)$ without the knowledge of the secret key. Johnson et. al introduced basic definitions of security for homomorphic signature systems and proposed several schemes that are homomorphic with respect to useful binary operators [Johnson et al., 2002].

We are interested in the set-homomorphic signature scheme proposed in [Johnson et al., 2002] that supports both union and subset operations. More precisely, the scheme allows anyone to compute $\mathsf{Sig}(S_1 \cup S_2)$ and $\mathsf{Sig}(S')$ where $S' \subseteq S_1$ if he possesses S_1, S_2, $\mathsf{Sig}(S_1)$ and $\mathsf{Sig}(S_2)$.

We now describe our construction for a Homomorphic Data Ownership Certificate (HDOC) scheme. We need to modify both the common nonce and the data ownership component to use the homomorphic signatures. Let S be a set of strings, E the active entity that originates S, and sk_S the signing key exclusively used for S. When E submits its information $S' \subseteq S$ to database A, it issues A an HDOC $H_S^A = \langle pk_S, \mathsf{Sig}_{sk_S}(S'), \mathsf{Sig}_{sk_S}(S' \cup A)\rangle$.

Computing intersection on data with HDOC is straight forward. Suppose databases A and B wish to compute intersection on their customers' credit card number. Then for each customer c_i's HDOC components $\mathsf{Sig}_{sk_S}(S_{c_i})$ and $\mathsf{Sig}_{sk_S}(S_{c_i} \cup A)$, database A computes $\mathsf{Sig}(S'_{c_i})$ and $\mathsf{Sig}(S'_{c_i} \cup A)$ where $S'_{c_i} = \{c_i's$ credit card #$\}$. B does similar computations. Now A and B may run any matching protocol as described in Section 6.2 using the recomputed HDOC.

7. Security Analysis

Recall that we consider four threat models in our paper: the malicious model with a large or small domain, and the semi-honest model with a large or small domain.

We have also identified three goals that a private matching protocol can satisfy: strong/weak, unspoofable/spoofable, and symmetric/asymmetric. In this section, we analyze the effectiveness of the three private matching protocols with respect to each of the threat models and determine what security goals each protocol achieves.

In this section, We analyze the fulfillment of the security goals of the TTPP , HP , AgES , CTTPP , CHP , and CAgES protocols in the four threat models. We summarize the results in Figure 3.2(a) through Figure 3.3(b).

7.1 The malicious model with a large domain

We now analyze the fulfillment of the security goals of the TTPP, HP, AgES, CTTPP, CHP, and CAgES protocols in the malicious model with a large domain. All the unmodified protocols are unspoofable in the absence of collude-then-spoof attacks. Although a large domain makes it difficult for an adversary to guess an element in the other party's set, the adversary can include values obtained from another database in the query to increase the probability of success.

7.1.1 Trusted Third Party Protocol. The Trusted Third Party Protocol (TTPP) is a spoofable, strong and either symmetric or asymmetric matching protocol. TTPP is strong because both parties learn only P and nothing else in a symmetric setting; in an asymmetric setting, one party learns P and the other party learns nothing. TTPP is always strong for this reason in all four threat models. TTPP can be either symmetric or asymmetric depending on whether the sends query results to one or both parties.

Technique		**Unspoofable**	**Strong**	**Symmetric**
TTPP	Sym		X	X
	Asym		X	
HP			(*)	
AgES			X	
CTTPP	Sym	X	X	X
	Asym	X	X	
CHP		$X^{(1)}$	(*)	
CAgES		$X^{(1)}$	X	

(a) Malicious model with a large domain

Technique		**Unspoofable**	**Strong**	**Symmetric**
TTPP	Sym		X	X
	Asym		X	
HP				
AgES			X	
CTTPP	Sym	X	X	X
	Asym	X	X	
CHP		$X^{(1)}$	(*)	
CAgES		$X^{(1)}$	X	

(b) Malicious model with a small domain

Figure 3.2. Security goals satisfied by the protocols in the malicious model. (*): Note that for these examples, we do not have a strong protocol. However, we do have a collusion-free strong protocol which is strong in the absence of colluding attacks . $X^{(1)}$ denotes a protocol is unspoofable in the absence of colluding adversaries.

Technique		**Unspoofable**	**Strong**	**Symmetric**
TTPP	Sym	X	X	X
	Asym	X	X	
HP		X	(*)	
AgES		X	X	
CTTPP	Sym	X	X	X
	Asym	X	X	
CHP		X	(*)	
CAgES		X	X	

(a) Semi-honest model with a large domain

Technique		**Unspoofable**	**Strong**	**Symmetric**
TTPP	Sym	X	X	X
	Asym	X	X	
HP		X		
AgES		X	X	
CTTPP	Sym	X	X	X
	Asym	X	X	
CHP		X	(*)	
CAgES		X	X	

(b) Semi-honest model with a small domain

Figure 3.3. Security goals satisfied by the protocols in the semi-honest model. (*): Note that for these examples, we do not have a strong protocol. However, we do have a collusion-free strong protocol which is strong in the absence of colluding attacks . $X^{(1)}$ denotes a protocol is unspoofable in the absence of colluding adversaries.

7.1.2 Hash Protocol. The Hash Protocol is spoofable, collusion-free strong, and asymmetric. It is strong in the absence of colluding attacks ; since the domain is large, it is difficult for the recipient of a hashed set to guess an element that is actually in the other party's set. However, it is easier for the recipient of a hashed set to learn whether an element is in the other party's set if the recipient uses values obtained from another database in the matching. The Hash Protocol is asymmetric since the party that receives the result first may or may not send the (correct) result to the other party.

7.1.3 AgES protocol. The AgES protocol is spoofable, strong, and asymmetric. The AgES is strong because no attacker may learn any additional information besides the query result and the size of the other party's set. It is asymmetric since the party that receives the result first may or may not send the (correct) result to the other party.

7.1.4 Certified matching protocols. CTTPP is unspoofable. If one of the parties spoofs some element d, the trusted third party can detect it by checking the ownership attestation component as described in Section 6.2.1.

Both CHP and CAgES are unspoofable in the absence of colluding adversaries. The common nonces in a DOC prevent a party from guessing the correct nonce associated with certain data and thus prevent guess-then-spoof attacks.

When colluding parties exist, CHP and CAgES are spoofable. Assume Alice is an authorized owner of some information d and Charlie is not. Alice colludes with Charlie and gives data d and the associated DOC to Charlie. When Bob sends his data set to Charlie in a CHP execution, Charlie can learn whether Bob has d by hashing the nonce n_d associated with d and checking if it is in Bob's set. There is a non-negligible probability that n_d is in Bob's set. This matching result violates the definition for unspoofable. Similarly, in a CAgES protocol execution, Charlie may encrypt the nonce n_d and send it to Bob. Charlie will discover whether Bob has d when Bob honestly responds to the query.

On the other hand, if Charlie and Bob switch roles in the CHP and CAgES protocol executions, Charlie cannot prove to Bob that he has d since he does not have a valid ownership attestation component for d.

7.2 The malicious model with a small domain

With a small domain, a malicious adversary can guess an element of the other party's set with non-negligible probability. An adversary can then launch a spoofing attack and learn elements of the other party's set not contained in its own with non-negligible probability. Therefore, without modification, all three protocols are spoofable in the malicious model with a small domain.

7.2.1 Trusted Third Party Protocol. The trusted third party is spoofable, strong, and either symmetric or asymmetric. The analysis is similar to that of the malicious model with a large domain presented in Section 7.1.1.

7.2.2 Hash Protocol. The hash protocol is spoofable, weak, and asymmetric. It is weak because a malicious party may launch a guess-then-spoof attack and succeed in learning the entire set of the other party with high probability. The analysis for asymmetry is similar to that of the hash protocol for the malicous model with a large domain presented in Section 7.1.2.

7.2.3 AgES protocol. The AgES protocol is spoofable, strong, and asymmetric. The AgES is spoofable because although the encryption scrambles the data, it cannot prevent spoofing attacks. The analysis for AgES being strong is similar to that of the malicious model with a large domain in Section 7.1.3. The analysis for asymmetry is similar to that of a large domain presented in Section 7.1.3.

7.2.4 Certified matching protocols. By combining the DOC with TTPP, HP, and AgES, we obtain protocols that satisfy the same security properties in the malicious model with a small domain as the corresponding certified protocols in the malicious domain with a large domain. In particular, by adding the DOC component, we enable the protocol to detect spoofed queries in the absence of colluding attacks .

7.3 The semi-honest model with a large domain

All three protocols are trivially unspoofable in a semi-honest model since parties do not cheat in a semi-honest model. For the strong/weak dimension, each protocol satisfies the same security goal as the corresponding protocol in a malicious model with a large domain in Section 7.1. The TTPP is can be either symmetric or asymmetric depending on whether the trusted party sends the result to one or both parties. The HP and AgES can also be either symmetric or asymmetric depending on whether the protocol prescribes the party which receives the result first sends it to the other party.

The TTPP is unspoofable, strong, and symmetric/asymmetric. The analysis of TTPP being strong is similar to that of a large domain presented in Section 7.1.1.

The AgES is an unspoofable, strong, and symmetric/asymmetric and protocol. The analysis of AgES beging strong is similar to that of a malicious model with a large domain in Section 7.1.3.

7.3.1 Certified matching protocols. All unmodified protocols are unspoofable in the semi-honest model. The DOC mechanism is not applicable in the semi-honest model with a large domain and this becomes clear in Section 7.4.

7.4 The semi-honest model with a small domain

The analysis for the semi-honest model with a small domain is similar to that of the semi-honest model with a large domain. The only difference is that the HP is collusion-free strong in the large domain and weak in the small domain and by combining the DOC with the HP, we obtain a protocol that is collusion-free strong in the semi-honest model with a small domain.

Protocol	**Cost**	**Complexity**
TTPP	$q \log q + b \log q$	$O(b \log b)$
HP	$C_h(q+b) + b \log b + q \log b$	$O(b \log b)$
AgES	$(C_h + 2C_e)(q+b) + 2b \log b + 3q \log q$	$O(C_e b)$
CTTPP	$q \log q + b \log q + 2C_x r$	$O(C_x r)$
CHP	$C_h(q+b) + b \log b + q \log b + C_x r$	$O(C_x r)$

(a) Computational cost

Protocol	**Cost**	**Complexity**
Asymmetric TTPP	$(q+b+r) \cdot n$	$O(bn)$
Symmetric TTPP	$(q+b+2r) \cdot n$	$O(bn)$
HP	$b \cdot l$	$O(bl)$
AgES	$(2q+b) \cdot m$	$O(bm)$
Asymmetric CTTPP	$(q+b+r) \cdot n + (q+b) \cdot k$	$O(bn)$
Symmetric CTTPP	$(q+b+r) \cdot n + (q+b) \cdot k$	$O(bn)$
CHP	$(l+k) \cdot b$	$O(bk)$

(b) Communication cost

Figure 3.4. Cost analysis

8. Cost Analysis

In this section, we use the following notations. Alice poses a query Q to Bob who has a set B. Let $P = Q \cap B$ be the query result. Let $q = |Q|$, $b = |B|$, and $p = |P|$. Let C_h be the cost of hashing and C_x be the cost of running the public key verification algorithm $\textsc{Verify}(\cdot, \cdot, \cdot)$. Let j be the length of a public key, k be the length of the ownership attestation component, l be the length of the output of $h(\cdot)$, m be the length of each encrypted code word in the range of $\mathcal{F}$, and n be the length of each element; all quantities are in bits. We assume that the set Q is larger than the set B, i.e. $b < q$, and we assume that $l \leq k + j \leq n$.

We present the computational and communication cost in Figure 3.4(a) and Figure 3.4(b) respectively.

The computational costs of the trusted third party and hashing protocols are dominated by the cost of sorting the list. For the AgES and certified protocols, the computation cost is dominated by the encryption/decryption and public key signature verification respectively. Further details can be found in Figure 3.

As we may see from Figure 3.4(b), the communication cost for any proposed protocol is linear in the size of the sets being sent. This linear communication cost is the lower bound of any set intersection protocols which compute exact matching [Kalyanasundaram and Schnitger, 1992].

9. Related Work

Private Information Retrieval (PIR) schemes allow a user to retrieve the i-th bit of an n-bit database without revealing i to the database [Beimel and Ishai, 2001, Cachin et al., 1999, Chor et al., 1995]. These schemes guarantee user privacy. Gertner et al. introduce Symmetrically-Private Information Retrieval (SPIR) where the privacy of the data, as well as the privacy of the user is guaranteed [Gertner et al., 1998]. In every invocation of a SPIR protocol, the user learns only a single bit of the n-bit database, and no other information about the data. Practical solutions are difficult to find since the PIR literature typically aims for very strong information-theoretic security bounds.

There has been recent work on searching encrypted data [Boneh and Franklin, 2004, Waters et al., 2004] inspired by Song, Wagner, and Perrig's original paper describing practical techniques for searching encrypted data [Song et al., 2000]. Song et al. proposed a cryptographic scheme to allow a party C to encrypt and store data on an untrusted remote server R. R can execute encrypted queries issued by C and return encrypted results to C.

10. Future Work

This paper explores some issues associated with private matching. But many areas remain to be explored. Here, we list a few particularly interesting challenges:

- In this paper, we examined two party protocols. What are the issues that arise with more complicated protocols with more than two parties?

- There is a basic asymmetry that arises between two parties where one party knows significantly more than a second party. Parties that control large sets may be able to extract significantly more interesting information than parties that control small sets. There may be instances where parties controling small sets can detect and reject these queries.

- Here, we only consider examples of matching elements from two sets. More interesting and more far-ranging examples are possible. For instance, this paper considered *listing queries* — we actually listed all the elements held in common between two sets. We can consider a broader range of *functional queries* which return a function calculated over the intersection of two sets. While a broad literature in statistical databases exists, the question of functional operations is a more general notion that deserves further attention.

- There is an interesting connection between our spoofing discussion and the database literature on updates through views. The view update literature provides (constrained) solutions for the following: given a query on relation instances R and S resulting in a set P, what changes to R and S could produce some new answer P'? The reasoning used to address that problem is not unlike the reasoning used to learn information via spoofing: by substituting R' for R and observing the query result P', what can be learned about S? The literature on updates through views is constrained because it seeks scenarios where there is a unique modification to R, S that can produce P'. By contrast, much can be learned in adversarial privacy attacks by inferring a non-unique set of *possible* values for S.

- In large distributed systems, it may be desirable to have a set of peer systems store information in a variety of locations. In this broader distributed system, can we still guarantee privacy properties.

- In our list of attacks in Section 3.1, we discussed a hiding attack where a database owner pretends certain values don't occur in its database. Can we provide effective defenses against hiding attacks?

Notes

1. In the introduction, we argued that semi-honest protocols were unrealistic in many situations. However, for completeness we will consider them here.

References

[Agrawal et al., 2003] Agrawal, R., Evfimievski, A., and Srikant, R. (2003). Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD Int'l Conf. on Management of Data, San Diego, CA.*

[Agrawal et al., 2002] Agrawal, Rakesh, Kiernan, Jerry, Srikant, Ramakrishnan, and Xu, Yirong (2002). Hippocratic databases. In *28th Int'l Conf. on Very Large Databases (VLDB), Hong Kong.*

[Beimel and Ishai, 2001] Beimel, Amos and Ishai, Yuval (2001). Information-theoretic private information retrieval: A unified construction. *Lecture Notes in Computer Science*, 2076.

[Bellare et al., 1996] Bellare, M., Canetti, Ran, and Krawczyk, Hugo (1996). Keying hash functions for message authentication. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology table of contents*, pages 1–15. Lecture Notes In Computer Science archive.

[Boneh and Franklin, 2004] Boneh, D. and Franklin, M. (2004). Public key encryption with keyword search. In *Eurocrypt 2004, LNCS 3027*, pages 56–73.

[Cachin et al., 1999] Cachin, Christian, Micali, Silvio, and Stadler, Markus (1999). Computationally private information retrieval with polylogarithmic communication. *Lecture Notes in Computer Science*, 1592.

[Canetti, 1996] Canetti, Ran (1996). *Studies in Secure Multiparty Computation and Applications*. PhD thesis, The Weizmann Institute of Science.

[Chor et al., 1995] Chor, Benny, Goldreich, Oded, Kushilevitz, Eyal, and Sudan, Madhu (1995). Private information retrieval. In *IEEE Symposium on Foundations of Computer Science*, pages 41–50.

[Gertner et al., 1998] Gertner, Yael, Ishai, Yuval, Kushilevitz, Eyal, and Malkin, Tal (1998). Protecting data privacy in private information retrieval schemes. In *The Thirtieth Annual ACM Symposium on Theory of Computing*, pages 151–160.

[Goldreich et al., 1987] Goldreich, O., Micali, S., and Wigderson, A. (1987). How to play any mental game. In *Proc. of 19th STOC*, pages 218–229.

[Goldreich, 2002] Goldreich, Oded (2002). Secure multi-party computation. Final (incomplete) draft, version 1.4.

[Huberman et al., 1999] Huberman, B. A., Franklin, M., and Hogg, T. (1999). Enhancing privacy and trust in electronic communities. In *ACM Conference on Electronic Commerce*, pages 78–86.

[Johnson et al., 2002] Johnson, Robert, Molnar, David, Song, Dawn Xiaodong, and Wagner, David (2002). Homomorphic signature schemes. In *CT-RSA*, pages 244–262.

[Kalyanasundaram and Schnitger, 1992] Kalyanasundaram, B. and Schnitger, G. (1992). The probabilistic communication complexicty of set intersection. SIAM J. Discrete Mathematics.

[Menezes et al., 1996] Menezes, A., van Oorschot, P., and Vanstone, S (1996). Handbook of applied cryptography. CRC Press.

[Parliament, 1995] Parliament, European (1995). Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data.

[Song et al., 2000] Song, Dawn Xiaodong, Wagner, David, and Perrig, Adrian (2000). Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55.

[Waters et al., 2004] Waters, Brent R., Balfanz, Dirk, Durfee, Glenn, and Smetters, D.K. (2004). Building an encrypted and searchable audit log. In *The 11th Annual Network and Distributed System Security Symposium*.

[Yao., 1986] Yao., A.C. (1986). How to generate and exchange secrets. In *In Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167.