SECURE COMPUTATION OF COMMON DATA AMONG MALICIOUS PARTNERS

Sebastian Obermeier and Stefan Böttcher
University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany

Keywords: Multiparty Computation, Intersection, Malicious Behavior.

Abstract: A secure calculation of common data $(D_1 \cap \ldots \cap D_n)$ of different participants without disclosing D_i is useful for many applications and has been studied as the Secure Multiparty Computation problem. However, proposed solutions assume all participants act "semi-honest", which means participants may neither alter the protocol execution nor fake database content. In this contribution, we focus on malicious participant behavior and prove that an atomic exchange of common data is not possible under the assumption of malicious participants. We propose a mechanism to calculate the intersection of multiple participants, which does not only reduce the disclosure in case participants cheat by altering the protocol to a negligible amount, it is also resistant against malicious participants that cooperate in order to cheat others. Furthermore, it impedes database content faking, which could be done when using other protocols by participants in order to check if data is contained in the other's databases. Last, we show experimentally the practical usability of our protocol and how the level of trust has an impact on the exchange speed of the intersection.

1 INTRODUCTION

Companies that store their enterprise information within a database often consider these information confidential. But there are also situations, in which *n* companies want to know if they have common data, and which data this is. However, the parties are not willing to disclose any other data than the intersection. The problem of securely computing the data all parties have in common is called the *sovereign information sharing* problem.

In this contribution, we address the problem of computing the intersection of multiple parties without trusting a third party, when one or more participants may act malicious. This means, we address the problem that participants may get an advantage by changing the exchange protocol in such a way, that the party that receives and fully decrypts the common data first can suppress the sending of the corresponding information that is necessary for the other party.

Whenever enterprises have need for an exchange of common data, but do not want to reveal other information, these data is often stored within database tables, say within database tables D_1 to D_n of the companies N_1 to N_n. A protocol is needed that returns the intersection $(D_1 \cap \ldots \cap D_n)$, but does not reveal any other information; it even should not disclose size information like $|D_i|$.

For example, let us look at an enterprise N_1 and its rivals N_2 and N_3, which do business in the service industry. All enterprises want to know whether all three have common customers in order to check whether these customers play the companies off against each other. Another use would be to detect customers of all three companies in order to start special marketing actions on these customers.

For this reason, N_1, N_2, and N_3 want to examine their customer databases for addresses of customers that are common to all three parties. However, no party is willing to disclose information about customers that the other parties do not have, and they do not want to disclose information about their number of customers, i.e. about the sizes $|D_i|$.

Existing approaches and proposals that also address the sovereign information sharing problem (Naor and Pinkas, 1999; Huberman et al., 1999; Freedman et al., ; Kissner and Song,) focus on solutions for two participants and either disclose the intersection to a *single* party only, or they assume an "honest-but-curious" behavior for all participants, which means they will follow the protocol strictly and send – as the protocol's last step – the information that the other parties needs to calculate the intersection. However, there might be situations in which participants that learned of the intersection suppresses the sending of information that is required for the other

Obermeier S. and Böttcher S. (2007).
SECURE COMPUTATION OF COMMON DATA AMONG MALICIOUS PARTNERS.
In *Proceedings of the Second International Conference on Security and Cryptography*, pages 345-354
DOI: 10.5220/0002117903450354

participants to decrypt the intersection information as well.

In addition, (Naor and Pinkas, 1999; Huberman et al., 1999) require that the database sizes $|D_i|$ are revealed. However, a sharing of size information may be not acceptable for companies, e.g. if $|D_i|$ represents the number of a company's customers.

There are business settings in which the partners do not want to rely on a trusted third party that performs a significant part of an intersection computation for a variety of reasons, e.g. a trusted third party may be too difficult or too time-consuming to find, it may be too expensive or simply not wanted for political reasons. Therefore, we do not assume to have a trusted third party and focus on the problem that each participant may stop the protocol execution whenever there is an advantage for him. In the following, we prove that when no trustworthy third party is available, an atomic exchange of common data is not possible if some participants may cheat in terms of altering the protocol. Second, we provide a mechanism that allows participants to exchange a bunch of information units like customer data or supplier information without having the risk of being cheated by more than one information unit.

Unfortunately, when data faking is considered as a possibility, participants could invent a lot of customer data in order to get the complete customer data set of another party as intersection. The only possibility to achieve that stored data complies with reality, i.e., to prevent participants from faking data, is a third-party auditing device that punishes participants for cheating (Agrawal and Terzi, 2006). Contrary to this approach, we present a mechanism appropriate especially for practical applications that makes it extremely hard to create faked data that will not be identified as such. Our approach bases on unique information, which is visible only to the real owners of the data item.

2 BASIC ASSUMPTIONS AND REQUIREMENTS

Besides the requirement to disclose only the intersection, we also need to guarantee that no party can cheat within the disclosing process. An attacker that has the goal to learn data from other participants that the attacker does not own by itself can use different techniques to achieve its goal: the attacker can inspect, manipulate, invent, or suppress messages; the attacker can manipulate the protocol, i.e. stop the protocol execution at any time or wrongly encrypt/decrypt messages, and the attacker can fake data, i.e. the attacker can invent data in order to learn whether other participants own this data.

Our protocol does not assume a failure-free network, and distinguishes between message loss and active message suppression. Thus, we assume that each participant acknowledges the received messages and a sender repeats the sending of non-acknowledged messages until the network will finally deliver the message.

We cannot guarantee that each party N_i provides all its data for the computation of the intersection $(D_1 \cap \ldots \cap D_n)$. Therefore, we assume that each participant contributes only that data to the intersection computation, which it accepts to disclose if it is in the intersection. Furthermore, we assume that participants agree on a common data format used for the data in the intersection computation.

An additional requirement is, that size information $|D_j|$ should not be disclosed to any other party.

Protocols that disclose $(D_1 \cap \ldots \cap D_n)$ only to one party (e.g. (Agrawal et al., 2003)) are not suitable since we cannot guarantee that the other parties will also receive the intersection. Therefore, we need a protocol that guarantees atomicity for the exchange of $(D_1 \cap \ldots \cap D_n)$.

Unfortunately, we can prove in Section 3.1 that a complete atomic exchange of $(D_1 \cap \ldots \cap D_n)$ cannot be guaranteed. Therefore, we need a protocol that at least reduces the suffered damage in the case that a company cheats.

3 SOLUTION

We first prove that an atomic exchange of the common data is not possible if participants may cheat in terms of message suppression.

3.1 Impossibility of Multiparty Atomic Data Exchange

Our proof that the atomic exchange of common data is not possible is based on a proof idea of the two generals' problem (Gray, 1978), where two generals want to agree on an attack time by using an uncertain medium. We will expand the proof to an information exchange among *n* participants.

Definition 3.1 *Let N_1 to N_n be the owners of the data D_1 to D_n. A sovereign information sharing protocol IP is said to be* multiparty intersection safe*, if it fulfills the following two conditions:*

1. *IP discloses $(D_1 \cap \ldots \cap D_n)$ to N_j exactly if it discloses $(D_1 \cap \ldots \cap D_n)$ to all other participants N_1 to N_n.*

2. *IP discloses no tuple of* $(D_i - (D_1 \cap \ldots \cap D_n))$ *to any participant* N_j *with* $j \neq i$.

Definition 3.2 *A participant* N_i *is called* distrustful*, if it will not send the information that is necessary to completely disclose the intersection* $(D_1 \cap \ldots \cap D_n)$ *to any other participant* N_j *without having the guarantee that* N_i *will also learn of* $(D_1 \cap \ldots \cap D_n)$.

Lemma 3.3 *Let* $N_1, \ldots, N_n$ *be the owners of the data sets* $D_1, \ldots, D_n$. *Without a trusted third party, there is no multiparty intersection safe protocol if all participants are distrustful.*

Proof By contradiction. Assume, there is an intersection safe protocol IP that delivers $(D_1 \cap \ldots \cap D_n)$ to all distrustful participants N_1 to N_n.

Then, there also exists a minimal protocol, i.e. a protocol that does not contain any superfluous message, to let all parties learn $(D_1 \cap \ldots \cap D_n)$. A minimal protocol is either the original protocol, or an equivalent protocol in which all superfluous messages are left out. To let the protocol compute and deliver the intersection, each participant N_i must send at least one message and must receive at least one message.

Because each successfully delivered message is received after it has been sent, each minimal protocol must contain also at least one last message M_{last} , i.e. a message which is sent by a participant N_j and received by a participant N_i after N_i has sent its last message. Since the protocol is minimal, M_{last} is needed by N_i to learn $(D_1 \cap \ldots \cap D_n)$. Furthermore, since the protocol delivers $(D_1 \cap \ldots \cap D_n)$ to all partners, it has used the information provided by N_i to compute $(D_1 \cap \ldots \cap D_n)$, i.e. N_i has sent the information that is necessary to completely disclose the intersection $(D_1 \cap \ldots \cap D_n)$, before N_i has received M_{last}. Therefore, N_j could suppress sending M_{last} without preventing N_i from sending its information that is necessary to completely disclose the intersection $(D_1 \cap \ldots \cap D_n)$. Therefore, N_i had no guarantee to learn about the intersection $(D_1 \cap \ldots \cap D_n)$, i.e., this behavior of N_i is a contradiction to the assumption that all participants act distrustful. □

The conclusion of this proof is that all but one parties must take the risk of being cheated and being the first who sends the information which is necessary to disclose $(D_1 \cap \ldots \cap D_n)$. Otherwise, there would be no exchange, since at least one party must be the last who sends a message M_{last} that completely discloses information of $(D_1 \cap \ldots \cap D_n)$.

However, although atomicity for the complete intersection is not possible, we can reduce the damage that the one-sided disclosure of the common data involves by an approach outlined in Section 3.4. The idea is to reveal only a small part of the intersection $(D_1 \cap \ldots \cap D_n)$, and let the next party send the next part in return. Since our proposed algorithm is able to detect faked data, the cheating of a party will uncover only a small part of the intersection, which in many cases reduces the damage.

However, as we can see in the next section, we cannot make the disclosed information parts, which we call *information units*, arbitrary small.

3.2 Information Units

Definition 3.4 *Let* $D_u := (D_1 \cap \ldots \cap D_n)$ *describe the intersection of the data of the companies* N_1 *to* N_n*, let* L *be the size of* D_u *in bits, and* $B : [1, L] \rightarrow \{0,1\}$ *be a bitarray containing the bits representing* D_u .

When we partition B *into several disjointed smaller parts* $\{d_1, \ldots, d_k\}$*, such that* $\{d_1 \cup \ldots \cup d_k = B\}$ *and* $d_i \cap d_j = \emptyset$ *for* $i \neq j$*, we call* $\{d_1, \ldots, d_k\}$ *a* set of information units *and we call each* d_i *an* information unit *of the intersection* D_u.

The part of the intersection that we want to disclose during one protocol exchange step in the second half of our protocol corresponds to one information unit. Note that information units are only considered for exchange purposes, and not for the calculation of D_u, which is based on tuples and not on information units.

Definition 3.5 *Given the set* $ds = \{d_1, \ldots, d_k\}$ *of information units, we call* $d_j \in ds$ independent *if the following holds.*

If $|D_i| > |(D_1 \cap \ldots \cap D_n)|$*, we cannot conclude an information unit* d_j *if we know* D_i *and* $ds \backslash \{d_j\}$.

Example 3.6 *Let* D_1 *and* D_2 *be customer database relations with* $|D_1|, |D_2| > |D_1 \cap D_2|$*, and let* $\{d_1, \ldots, d_k\}$ *be a set of information units of the intersection* $(D_1 \cap D_2)$*, such that each information unit* d_j *represents a single* customer*, and the customers occur in a randomized order. In this case, the set* $\{d_1, ..., d_k\}$ *is independent for the following reason. We cannot conclude a customer* $d_j \in (D_1 \cap D_2)$ *even if we know* D_i *and* $\{d_1, \ldots, d_k\} \backslash \{d_j\}$*, i.e. the complete intersection except the missing customer, because there are at least two remaining customers who might be* d_i *due to* $|D_i| > |(D_1 \cap D_2)|$.

Note that an independent information unit may contain more than one customer, but an independent information unit cannot be made arbitrarily small. The next example, which focuses on security but not on efficiency, shows this property.

Example 3.7 *Let* D_1 *and* D_2 *be customer database relations and* $\{d_1 \ldots d_k\}$ *information units representing* characters *occurring in* $(D_1 \cap D_2)$. *This means,*

each customer is represented by several information units $\{d_i \dots d_l\}$. However, the set $\{d_1 \dots d_k\}$ is not independent for the following reason. If we can use D_1 to identify the customer cu that is represented partially by the characters $\{d_i \dots d_l\}$ with $cu \in (D_1 \cap D_2)$, we can conclude the next character of cu. For example, if $\{d_i \dots d_l\}$ discloses the substring "`Miller, Flori`" of a customer name and we have only one Miller from Florida in our customer database D_1, we know that this Miller belongs to $(D_1 \cap D_2)$ and that further information units d_{l+1} and d_{l+2} will disclose "d" and "a". Therefore, if we use characters as information units, the information units used during the exchange process are not independent.

For this reason, if non-independent information units are used, a party can sometimes conclude more than one information unit while the other party may not necessarily know which data is meant. Therefore, if e.g. D_i knows which data is in the intersection and cheats by stopping the exchange of non-independent information units, another party D_j may have no chance to conclude any missing information unit.

When exchanging only independent information units, we can reduce the advantage that a cheating party may get by altering the protocol to one independent information unit (c.f. Section 3.4.3).

3.3 Cryptographic Basis

Our solution is based on commutative encryption, which means that given two cryptographic encryption keys $\texttt{key}_{N_i}$ used by N_i and $\texttt{key}_{N_j}$ used by N_j, the encryption order of applying an encryption function E is commutative:

$$E_{\texttt{key}_{N_i}}\left(E_{\texttt{key}_{N_j}}(d)\right) = E_{\texttt{key}_{N_j}}\left(E_{\texttt{key}_{N_i}}(d)\right) = c_d$$

Since the order of applying the encryption functions $E_{\texttt{key}_{N_i}}$ and $E_{\texttt{key}_{N_j}}$ and the corresponding decryption functions does not matter, we call this *commutative encryption*.

Cryptographic functions and their implementation have already been discussed in various cryptographic publications (Diffie and Hellman, 1976; Gamal, 1985). Therefore, we assume that secure algorithms are used, such that keys cannot be broken even if plain and ciphered text are known.

3.4 Exchange Algorithm

Given n participants, our algorithm is applied cyclically to each participant. To explain the calculations of each participant, we start labeling the first participant with N_1 and the last one with N_n, where the participant's indices form a ring, i.e. the successor of n is 1. We use the bracket notation $[X]$ as a shortcut for $((X-1) \bmod n)+1$.

Definition 3.8 *We define P_i^j as the data of the participant N_i, where the same hash function was sequentially applied j times to each data tuple of D_i. We further use the notion P_i as a shortcut for P_i^n, while P_i^0 describes the plain data.*

Definition 3.9 *We define $C_{P_i}^j$ to be the set of tuples that we get if each hashed tuple of participant N_i is encrypted sequentially by the participants N_i, $N_{[i+1]} \dots N_{[i+j-1]}$, i.e. it has been encrypted j times.*

Example 3.10 *For $n = 7$ participants, the notion $C_{P_6}^3$ means that each hash value of P_6 is first encrypted by N_6, the resulting tuples are then encrypted by N_7, and finally encrypted by N_1.*

The general idea of our algorithm can be summarized within three steps.

3.4.1 Initialization Phase

In this first phase, all participants agree on the data field(s) that they want to check for common data. In order to hide the real size information $|D_i|$, they agree on a common database size sz to be exchanged. sz should be chosen in such a way that $sz > \max(|D_1|, \dots |D_n|)$. After this, each party N_i adds randomly created data to their database D_i until $|D_i| = sz$.

Afterwards, each participant hashes each of its own data tuples n times with the same hash algorithm, such that each participant N_i has computed $P_i := P_i^n$ and has stored all of its computed intermediate hash results P_i^j for $1 \leq j \leq n$. The hashing of values n times is used later in the Verification Phase in such a way that each of the participants has to contribute one preimage of a hashed value as a proof of ownership.

3.4.2 Intersection Exchange Phase

N_i encrypts each data tuple of P_i, resulting in $C_{P_i}^1$. Then, N_i passes them to the next participant $N_{[i+1]}$, who encrypts them resulting in $C_{P_i}^1$ and passes them to $N_{[i+2]}$, who also encrypts them etc., until each data tuple is encrypted n times by n different participants. These n times encrypted data tuples $C_{P_j}^n$ are exchanged, such that each participant stores the set $\{C_{P_j}^n | 1 \leq j \leq n\}$. The exchange phase is illustrated in Figure 1, where N_1 starts encrypting its hashed data P_1, and passing it to N_2. After one cycle, N_1 gets

back its hashed data n times encrypted as $C^n_{P_1}$ from N_n. This data is cyclically sent to all other participants, i.e.

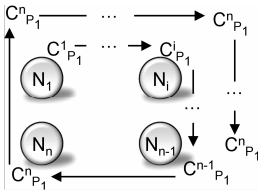

Figure 1: Intersection Exchange Phase.

Due to commutative encryption, the intersection of the encrypted data $C^n_{P_1} \cap C^n_{P_2} \cap \ldots \cap C^n_{P_n}$ represents the intersection of the original data $P_1 \cap P_2 \cap \ldots \cap P_n$.

Algorithm 1 Intersection Exchange Algorithm for N_i.

```
1:  D_act ← encrypt(P_i, key_{N_i})
2:  for j := 1 to n do
3:      sendWithThread(D_act, N_{[i+1]})          ▷ Send D_act
4:      D_act ← receive(N_{[i−1]})                ▷ Rec. from N_{[i−1]}
5:      D_act ← shuffle(encrypt(D_act, key_{N_i}))
6:  end for
7:  C^n_{P_{[i+1]}} ← D_act                       ▷ Store the fully encrypted data
8:  for j := 1 to n − 1 do
9:      sendWithThread(D_act, N_{[i+1]})          ▷ Send to N_{[i+1]}
10:     D_act ← receive(N_{[i−1)]})               ▷ Rec. from N_{[i−1]}
11:     C^n_{P_{[i−j]}} ← D_act                   ▷ Store encr. data of N_{[i−j]}
12: end for
```

Algorithm 1 shows the multiple encryption steps of a participant's data. First, each participant N_i encrypts each of its own tuples P_i with its own key (line 1). Then, this encrypted data is sent to the next participant $N_{[i+1]}$, and the data from the previous participant $N_{[i-1]}$ is received (lines 3-4). Since each participant's data has to be encrypted by every participant, N_i encrypts the all data with its own key and shuffles the resulting tuples (line 5). In the next iteration of the for loop (line 2-6), this data is again sent to $N_{[i+1]}$. After n iterations, $D_{\mathtt{act}}$ contains the data of participant $N_{[i+1]}$, which was n times encrypted, i.e. $C^n_{P_{[i+1]}}$ (line 7).

When all participants have exchanged the encrypted data, each participant has stored the same encrypted data after the execution of the second for-loop (lines 8-12).

Table 1 shows an example execution of the algorithm for $n = 3$. After the initial phase, each participant N_i has its own encrypted data $C^1_{P_i}$. In the next round, each participant sends its data to the next participant, which encrypts it with its key. After $n-1$ rounds, each participant holds data that is encrypted n times. These data are exchanged in the following exchange phase, until every participant is able to store all fully encrypted data.

Table 1: Example execution of Algorithm 1.

N_1	N_2	N_3	
$C^1_{P_1}$	$C^1_{P_2}$	$C^1_{P_3}$	each part. encr. its own data
$C^2_{P_3}$	$C^2_{P_1}$	$C^2_{P_2}$	after lines 2-6, $j = 1$
$C^3_{P_2}$	$C^3_{P_3}$	$C^3_{P_1}$	after lines 2-7, $j = 2$
exchange phase (lines 8–11)			
$C^3_{P_1}$	$C^3_{P_2}$	$C^3_{P_3}$	after lines 8-11, $j = 1$
$C^3_{P_3}$	$C^3_{P_1}$	$C^3_{P_2}$	after lines 8-11, $j = 2$

3.4.3 Revealing and Verification Phase

In this phase, participants agree on an order for determining the intersecting information units. The first participant passes the first information unit to the next participant, which decrypts it with its key, and passes it to the next participant, which decrypts with its key, etc. Finally, the first participant will receive the initial information unit, which is only encrypted by itself. Therefore, the first participant can decrypt the information unit and learn the hash value. From the hash value, it can conclude the plain data.

In order to make sure that the participant who just learned the plain data was not betrayed[1], it passes the hash value v to the next participant. Only if v is in the next participant's database, the next participant can conclude the plain data and can use the set of hashed data that was created in the Initialization Phase to look up the hash value v' that was hashed to v by applying $h(v') = v$. This means, each participant decodes the hashing that was applied n times in the Initialization Phase by one step.

After each participant has received all encrypted data $C^n_{P_1} \ldots C^n_{P_n}$, it can detect the intersecting data tuples. We assume that all participants apply the same algorithm to agree on a labeling of intersecting tuples, on a coordinator that starts the revealing of data tuples for each round, and on the number of tuples per information unit.

The revealing and verification algorithm is described by Algorithm 2. Each participant N_i starts by adding sequentially the amount of intersecting tuples that should be in a single information unit j (line 2). Then, participant N_i decides if it is responsible for coordinating the round. If this is the case, is sends the first information unit[2] that is in the intersection of the encrypted values to $N_{[i+1]}$ (line 4). $N_{[i+1]}$ receives the

[1] which may be the case if $N_{[i-1]}$ sends to N_i an arbitrary value of the set $C^1_{P_i}$

[2] for simplification, we assume in the following that an information units contains a single data tuple

Algorithm 2 Revealing and Verification Algorithm for N_i.

```
 1: round ← 1
 2: while ( j:= getCommonInfUnit()) != null do
 3:     if coordinatesRound(N_i, round) == true then
 4:         sendWithThread(j, N_[i+1])
 5:     end if
 6:     j ← receive(N_[i−1])              ▷ Receive from N_[i−1]
 7:     j ← decrypt(j, key_{N_i})
 8:     sendWithThread(j, N_[i+1])
 9:             ▷ if N_i is coordinator, j is plain hash and the
    verification starts
10:     if coordinatesRound(N_i, round) == true then
11:         originalText ← deHash(j))
12:     end if
13:     h ← receive(N_[i−1])          ▷ Rec. hash from N_[i−1]
14:     plain ← deHash(h)                 ▷ Lookup plain value
15:     if coordinatesRound(N_i, round) == false then
16:         verify ← deHash1Level(h)
17:         sendWithThread(verify, N_[i+1])
18:     else
19:         Check(originalText == deHash1Level(h))
20:         NotifyNextCoordinator()
21:     end if
22: end while
```

information unit (line 6), decrypts it, and passes it to the next participant (lines 7-8). After all participants except N_i decrypted the information unit, N_i will receive it and decrypt it (line 6-7). After this decryption, j contains the decrypted, n times hashed value of a single data tuple from P_j. This cyclic decryption is analogous to Algorithm 1. Then, N_i is able to lookup the original text that was hashed to this value (line 11). However, it wants to make sure that no one cheated, therefore a second cyclic exchange phase is started. At this time, each participant that receives a hash value h is able to lookup the plain data (line 11). Furthermore, it can also lookup the hash value `verify` that was hashed to h, i.e. h := hash(*verify*) by using the deHash1Level(h) function. This means, the deHash1Level(h) function searches the intermediate hash results that each participant generated in the Initialisation Phase, i.e. P_i^j for $1 \leq j \leq n$, to look up the hash value that was hashed to h. A detailed explanation why this step ensures security can be found in the following Section 3.5. The hash value *verify* is then passed to the next participant, which again deHashes one level. In the end, N_i have to perform a last deHash1Level step and will then receive the plain data. This plain data is compared with the plain data that the coordinator has looked up before the verification phase (line 11) started. If this data is equal, no one cheated since all had the hash values and the data item stored within their database.

3.5 Correctness

The real database size information of a participant N_i is hidden to all other participants, since in the Initialisation phase, N_i adds a bunch of random data to its database until $|D_i| \approx sz$. Since the random data tuples will not find corresponding data within the intersection, the original database size of a participant is concealed.

Although each participants receives the encrypted data of all other participants, it does not know which of his tuples correspond to the received data tuples due to encryption. Furthermore, since each encryption step involves shuffling the encrypted data, a participant that receives the encryption of its own data tuples does not even know which of its own plain text tuples correspond to a concrete encrypted data tuple.

In addition, message manipulation can be detected in the verification phase: The coordinator, who is the first participant who reveals the hash values and thereby the plain data, demands from each participant N_j a proof that N_j stores the revealed data in its database. This proof is done by using one of the n hash values for each tuple: A participant N_j that receives a hash value h can only lookup the value `verify` that was hashed to h if it owns the value `verify`. This means, the statement `verify` ← `deHash1Level`(h) (line 16) can only be evaluated if N_j stores the data. If N_j cheats, $N_{[j+1]}$ will either not find the hash value it received from N_j in its hash data for the actual hash level, or, if N_j simply forwards h, the coordinator will detect in the last step that one participant cheated and the hash value does not map to the plain value, but to another hash value instead (line 19). Since each data tuple is hashed n times, each of the n participants must have the original data tuple and all of its hash values stored to let the last participant send the hash value of level 1 to the coordinator.

The verification phase also detects as follows whether in the decryption part someone cheated and exchanged data: Since any decrypted hash value h will only pass the cyclic verification if all participants own the data, h is either in the intersection, or it will not pass the last check of the coordinator, since in this case at least one participant does not have the necessary information for the `deHash1Level`(h) function. If this happens, the participant that is the first to detect the cheat will stop the protocol and not continue decrypting the units. Furthermore, the participant that cheated can be identified since it does not own the hash value, and therefore it is expelled from the protocol.

This means, altering the protocol by suppressing or manipulating messages may only prevent the par-

ties from learning one information unit of the intersection, but does not disclose the complete set $(D_1 \cap \ldots \cap D_n)$.

A participant N_i that is supposed to encrypt data tuples of another participant N_j may delete some or all of these data. However, missing tuples will not occur in the intersection, and therefore these tuples will not be decrypted, so N_j will not get any information about the data it deleted. Since N_i agreed to bring in only those tuples that N_i is willing to share in case all others have the same data, there is no incentive for N_i to delete data in Algorithm 1, since it has the same effect as deleting data of its own data file in the Initialisation Phase, except that it does not even know which data will be excluded from the intersection.

3.6 Impeding Tuple Faking

Although cheating in terms of message suppression and manipulation can be detected by our algorithm and therefore the damage is reduced, one problem remains, which is faking database tuples. In our example, evil participants can add a huge amount of telephone book entries to their customer databases in order to abuse intersection computation to check which customers the other partners have. To impede this kind of faking, the parties must agree to additionally supply *tuple-specific information* that all parties have due to the real existence of the tuple in their database. This tuple-specific information should not be of such a kind that it can not be guessed or concluded from information that is available for public. An address field, for instance, is no tuple-specific information since it can be easily obtained from public resources.

Example 3.11 *Credit card data, for example, is a tuple-specific information. If this data is faked, the generated credit card number belonging to the customer will extremely unlikely match with the real credit card number, i.e. the complete customer record will differ and thus both customer entries differ in their hash values. Other examples for tuple specific information are social security number, income tax number, student number, etc.*

4 EXPERIMENTAL RESULTS

4.1 Exchange Speed Versus Trust

The efficiency of our intersection computation algorithm depends on given parameters like the intersection size, the connection speed, and on a chooseable parameter, i.e. the size of exchanged information units. As the exchange of small information units needs more communication steps than exchanging larger information units, we have a trade-off between trust and speed. When the parties do not expect a malicious behavior, an information unit may contain more data than it may contain when malicious behavior is expected.

Table 2: Preparation Times.

Desc	Time	Size
1,3 mill. entries	—	200 MB
Hashing, 1-Pass	27s	37 MB
Encrypting, 1-Pass	38s	37 MB
Decrypting, 1-Pass	48s	37 MB

4.2 Experimental Setup

We have prototypically implemented our protocol, in order to get information about the exchange speed when we change the number of exchanged information units, i.e. the level of how much the participants trust each other. For generating the test set, we have extracted a part of the German telephone book that contains about 1,3 mill. entries (i.e. data from Berlin), having the size of about 200MB. Table 2 shows our measurements using SHA-1 as hash function on an AMD 64 3700+ with 2 GB RAM.

After applying the hash function, the resulting file has the size of 37 MB. When applying the algorithm to n participants, the hashing and encryption times has to be multiplied with n, while the decryption is only applied to the information units of the intersection.

Since the exchange of the data is pure data exchange plus the time the encryption needs, we show in the following the results for the tuple exchange of the Revealing and Verification Phase and compare our algorithm for 2, 3, 5 and 7 participants exchanging 1, 2, 5, 10, 15 and 100 information units per message. In this test scenario, the participants are connected via a 100MBit network. In order to get information about low bandwidth connections as well, we measured the time for 2 participants, one of which is connected by "DSL light" that allows only 768kbit/s downstream and 128kbit/s upstream data transfer rates. The average ping rate for packets of size 1024 bytes between these two participants is 144ms.

Figure 2 shows the time needed for the exchange and verification phase of 10,000 tuples within an 100MBit network. The indicated time on the $y-$ axis is the time that is needed for passing the tuples two times cyclically to all participants in order to decrypt the information units and store the hash values to disc. However, we have omitted the time that is needed for

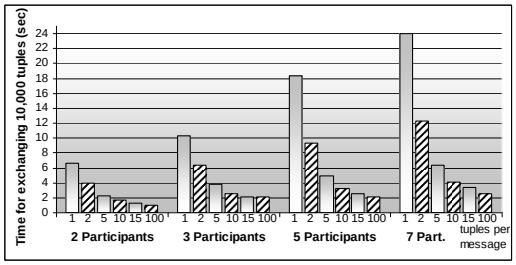

Figure 2: Algorithm Exchange Phase (LAN).

the hash lookup, since this constant factor highly depends on the number of intersecting tuples, the implementation (e.g. the use of indices or databases) and the available main memory. In our experiments, we were able to load all hashed data into main memory, therefore the time for the hash lookup (which we did by using hash tables) of 10,000 data values was around 13 milli seconds. On the $x-$ axis, the number n corresponds to the number of participants, while directly underneath each bar the number of tuples that are exchanged in one message is given, i.e. the number of tuples within one information unit.

If we have more participants, the number of participants that must decrypt and dehash the data rises analogously. This explains the additional amount of time that is needed for 5 and 7 participants. However, this additional time rises linearly, since each additional participant increases the amount of additional exchange steps by a constant factor.

Figure 3 shows the time which is needed for exchanging 1,000 tuples between two participants connected by a "DSL light" broadband connection. Since a LAN connection's response time is much superior to the DSL connection's response time, the exchange speed for the DSL connection is also much slower. However, the effect that a higher level of trust and an increased number of tuples bundled within one information unit have is much greater than in a LAN environment.

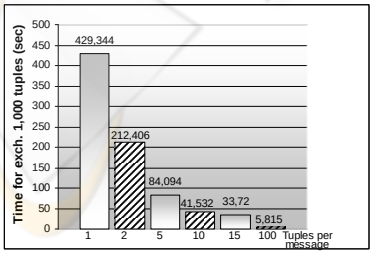

Figure 3: Exchange for 2 Participants (DSL).

To summarize, if participants do not trust each other very much and send only one tuple in advance, a fast connection highly pays-off, while for participants whose level of trust is higher and who therefore agree to send more tuples within an information unit, a fast connection is not an essential requirement. Due to the cyclic exchange, the number of (equivalently connected) participants has only linear impact on the exchange speed.

5 RELATED WORK

There are two aspects that have been studied widely within the sovereign information sharing scenario: the *multiparty computation* and the *fair exchange* of the data. The multiparty computation concentrates on the joint computation of $f(D_1,\ldots,D_n)$ without revealing the actual data D_i. Examples of functions f that are computed are cooperative scientific computations arising in linear programming (Du and Atallah, 2001), or Yao's Millionaire's protocol (Yao, 1982), in which $f(a,b)$ returns the information of whether or not $a > b$.

Cryptographic approaches like (Freedman et al.,) also discuss the multiparty computation problem with f as the intersection function for two datasets D_1 and D_2. However, these solutions guarantee that only one participant learns of the intersection, and not the other one. Therefore, the used malicious adversarial model for multi-party computation does not include the problem that one participant out of many participants may abort the protocol at every time, and must not get an information advantage out of this behavior compared to other participants. (Kissner and Song,) proposes a set intersection algorithm for malicious parties, but does neither tackle the problem that participants can take the "whole world" as customers in order to get knowledge the other participants complete database, nor gives experimental results for the algorithm.

When databases are used to store enterprise information, multiparty computation often relates to special database functions (Agrawal et al., 2003; Clifton et al., 2003). A secure computation of the join-operator, for instance, is discussed in (Agrawal et al., 2003). This solution, proposed for $n = 2$ parties, also uses commutative encryption, but reveals the data to one party only since it assumes an "semi-honest" behavior (Goldreich, 2000), which means that although a party might analyze messages, it will not alter the protocol execution. If we take participants that cheat into consideration, and would try to adapt our idea of a pair-wise exchange of intersecting tuples to (Agrawal et al., 2003), participants still may cheat, as explained in the following: In the first step, N_1 receives $C^1_{P_2}$ (notation defined as in Definition 3.9), N_2

gets $C^1_{P_1}$. Then both parties encrypt the data a second time, and determine the common tuples. The first party N_1 decrypts the first common tuple and sends it as h to N_2. However, N_2 has no evidence that the decrypted data is really in the intersection. N_1 may have cheated and have sent an arbitrary item of $C^1_{P_2}$ to N_2. N_2 would wrongly assume that this item is in the intersection, decrypt it with its key and send the hash value back to N_1. In this case, N_1 does not only know the hash value but also whether the associated data is really in the intersection. Furthermore, N_1 can store the hash value in order to check if future customers are also in the database of N_2. The crux of adapting this approach to a step-by-step exchange is that N_2 has no means to determine if N_1 plays fair.

In contrast, our solution is suitable for an arbitrary number of participants and is not restricted to $n = 2$ like our previous contribution (Böttcher and Obermeier, 2006), and focuses on a model where each of the n participants may act malicious and may not only stop the protocol execution, but may also change messages or fake data. We introduce the term information unit and show that no secure exchange protocol exists that can guarantee an atomic exchange of a single information unit. Furthermore, we add an additional verification phase, which will detect any cheating of a participant.

Since we reveal the decrypted information units of the intersection step by step, proposals for guaranteeing a *fair data exchange* are also relevant. Some of these proposals rely on a *trusted third party* (Ajmani et al., 2001; Jefferies et al., 1995), while other proposals do not necessarily need this third party. (Asokan et al., 1997; Asokan et al., 1998), for example, describe an approach for a fair exchange of items by using a third party only if participants cheat. If a third party is present but not trustable, (Franklin and Reiter, 1997) shows an approach to use this third party for fair data exchange. (Asokan et al., 1997) classifies the type of the exchanged items, and claims to guarantee an atomic exchange for items belonging to the categories revocable or generatable. However, since enterprise information is in many cases neither revocable nor generatable, the approach to use a third party for collecting affidavits and starting law suits in case of malicious participants is suitable for goods and items, but cannot be used to revoke the reveal of sensible enterprise data. In contrast, our approach does not rely on a certain item category; it is useful for non-revocable and non-generatable items as well.

6 SUMMARY AND CONCLUSION

In this contribution, we have presented an application scenario where multiple parties need a secure exchange of common information, although they do not trust each other and assume malicious behavior. We have shown that atomicity for the exchange of the common data is not possible if no trusted third party is used for this purpose. Furthermore, we have proposed a solution, which reduces the damage that each party suffers in case that another party alters the exchange protocol to the disclosure of one additional independent information unit. We have shown experimental results on the trade-off "trust vs. exchange speed", and demonstrated that even in an environment with high message latency our protocol is still feasible.

In the future, we plan to investigate a secure and secret processing of arbitrary database algebra expressions.

REFERENCES

Agrawal, R., Evfimievski, A. V., and Srikant, R. (2003). Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA*, pages 86–97.

Agrawal, R. and Terzi, E. (2006). On honesty in sovereign information sharing. In *10th International Conference on Extending Database Technology*, pages 240–256, Munich, Germany.

Ajmani, S., Morris, R., and Liskov, B. (2001). A trusted third-party computation service. Technical Report MIT-LCS-TR-847, MIT.

Asokan, N., Schunter, M., and Waidner, M. (1997). Optimistic protocols for fair exchange. In *CCS '97: Proceedings of the 4th ACM conference on Computer and communications security*, pages 7–17. ACM Press.

Asokan, N., Shoup, V., and Waidner, M. (1998). Asynchronous protocols for optimistic fair exchange. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 86–99.

Böttcher, S. and Obermeier, S. (2006). Sovereign information sharing among malicious partners. In *Secure Data Management, Third VLDB Workshop, Seoul, Korea*, pages 18–29.

Clifton, C., Kantarcioglu, M., Lin, X., Vaidya, J., and Zhu, M. (2003). Tools for privacy preserving distributed data mining.

Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654.

Du, W. and Atallah, M. J. (2001). Secure multi-party computation problems and their applications: A review and open problems. In *New Security Paradigms Workshop*, pages 11–20, Cloudcroft, New Mexico, USA.

Franklin, M. K. and Reiter, M. K. (1997). Fair exchange with a semi-trusted third party (extended abstract). In *ACM Conference on Computer and Communications Security*, pages 1–5.

Freedman, M., Nissim, K., and Pinkas, B. Efficient private matching and set intersection. In *Advances in Cryptology — EUROCRYPT 2004*.

Gamal, T. E. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA. Springer-Verlag New York, Inc.

Goldreich, O. (2000). Secure multi-party computation. Working Draft.

Gray, J. (1978). Notes on data base operating systems. In *Operating Systems, An Advanced Course*, pages 393–481, London, UK. Springer-Verlag.

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.

Jefferies, N., Mitchell, C. J., and Walker, M. (1995). A proposed architecture for trusted third party services. In *Cryptography: Policy and Algorithms*, pages 98–104.

Kissner, L. and Song, D. X. Privacy-preserving set operations. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference*.

Naor, M. and Pinkas, B. (1999). Oblivious transfer and polynomial evaluation. In *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 245–254, New York, NY, USA. ACM Press.

Yao, A. C. (1982). Protocols for secure computations. In *Proceedings of the 21st Annual IEEE Symposium on the Foundations of Computer Science*, pages 160–164, Chicago. IEEE.