Using secure multi-party computation when processing distributed health data

Anders Andersen
Department of Computer Science
Faculty of Science and Technology
University of Tromsø
9037 Tromsø, Norway

Abstract—Patient related health data are typically located at different general practices and hospitals. When processing and analyzing such data, the provided infrastructure and toolset has to take into consideration legal, security and privacy issues. The combination of secure multi-party computations (SMC) algorithms, encryption, public key infrastructure (PKI), certificates, and a certificate authority (CA) is used to implement an infrastructure and a toolset for statistical analysis of health data. The general practices and hospitals are considered nodes in a computing graph, and at each node a sub-process performs the local part of the computation. The described approach tries to support a wide range of possible SMC algorithms and computing graphs.

Keywords: SMC algorithms, Privacy, PKI, Health data.

1. Introduction

In this paper an infrastructure and a toolset for statistical analysis of health data is presented and discussed. It uses a combination of secure multi-party computations (SMC) algorithms [1], [2], encryption, public key infrastructure (PKI), certificates, and a certificate authority (CA) to manage the legal, security and privacy issues of such analysis. A coordinator that prepares the computation, and a set of sub-processes representing the parties in the multi-party computation. To send a message securely to a sub-process or the coordinator, the address and public encryption key of the receiver is needed.

In preparing the computation the coordinator creates a computing graph, and based on this a set of messages initiating the computation. The nodes in the computing graph are the coordinator and the sub-processes. The edges of the directed graph are the communication (messages sent) between nodes. A sub-process can only access or unwrap a limited part of the message it receives. The part it can access includes its input data and the address/key of the set of nodes it is forwarding its intermediate computational results to. The part of the message it cannot unwrap is encrypted with the public key of the receiver and is forwarded unchanged to the next nodes in the graph.

This approach will be evaluated in the context of the Snow SMSC (Secure Multi-party Statistical Computations) project. The Snow system [3] utilizes the mobile software agent approach. The system is deployed in health institutions and is used for data extraction for disease surveillance purposes. It extracts health data and performs data aggregation from general practices in Norway. In the Snow SMSC project the goal is to provide a toolset for statistical analysis of health data. Any medical research using health data is difficult because of the legal, security and privacy issues involved. This enforces a set of constrains upon the computations and their implementations. To address this, Snow SMSC will use the outcome of SMC research [4].

In [5], SMC for $N$ institutions with values $x_1, \ldots, x_N$ who wish to evaluate a known function $f(x_1, \ldots, x_N)$ is subject to four constrains:

C1: The correct value $f(x_1, \ldots, x_N)$ is obtained and known to all institutions.
C2: No institution $j$ learns more about the other institutions values $X_{-j} = \{x_n : n \neq j\}$ than it can deduce from $x_j$ and $f(x_1, \ldots, x_N)$.
C3: No trusted third party—human or machine—is part of the process.
C4: Semi-honesty. Institutions perform agreed upon computations correctly using their true data. However, they are permitted to retain the results of intermediate computations.

For practical purposes, the zero information disclosure implied by C3 is a difficult implement efficiently [6]. In the context of the Snow SMSC project we will use the following relaxed constrain replacing C3:

C3’: No human or machine is allowed to have access to both the patient identifier $i$, and a data value $x_i$ not previously known.

In [7] six requirements for secure disease surveillance are presented. These requirements includes what is special with health data and should be taken into consideration when performing such analysis. In [7] a secure multi-party computation protocol was developed to meet these requirements. However, in this paper we will focus on the four constrains presented above.
The implementation approach presented in this paper will consist of a combination of SMC algorithms and careful usage of encryption. The usage of public-key encryption is essential, but in many cases a combination of public-key and symmetric encryption should be used for performance reasons.

2. SMC in Snow SMSC

2.1 Computing graph and layered messages

A coordinator creates the computing graph for the computing processes. Each computing process is a sub-process of the overall computation. This relates to a MapReduce approach [8] where a coordinator performs the computation in a map and a reduce step. The map step divides a task and it inputs into a set of smaller sub-problems and distributes them to worker nodes. In the reduce step the answers from the sub-problems are collected and combined to the final result. However, in Snow SMSC (and in the case of many SMC problems) it is not always feasible to collect all the intermediate results at the coordinator. For some types of computations and data sets this might compromise the privacy of the patients. This is illustrated in the example below where a calculation of the mean value of some patient data is performed by forwarding the intermediate results to the next sub-process in a chain. This means that in Snow SMSC we have to support a wide range of computing graphs, and the tools and protocols developed in the project have to support this flexible arrangement.

The coordinator uses the computing graph to generate a set of layered messages. Each layer in these messages exposes the next edges in the directed graph. The computation is initiated by sending these messages to the first set of sub-processes. At each sub-process one layer of the received message is unwrapped exposing both the input data set for the calculation performed at this node, and the address/key of the next sub-processes in the computing graph. The calculation is performed using the input data set and local data available at this node. When the calculation is done the sub-process generates a set of messages forwarded to the next sub-processes in the computing graph. The data set included in these messages is based on the result of the performed calculation and a filter function. The filter function is used to ensure that a given sub-process is only forwarded data that it needs to perform its calculation. The filter function makes it possible for a sub-process to forward different data set to the next sub-processes in the computing graph. Local data at the node might also be updated when the calculation is performed.

2.2 Privacy

The privacy of the patients is a main concern in systems accessing personal health data. To maintain this privacy Snow SMSC has to ensure that constraints C2 and C3 (see above) are maintained [5]. This will be achieved with SMC algorithms [4], [2], encryption of data and messages, and a public-key infrastructure (PKI). SMC algorithms ensure that each participant performing a sub-process is unable to learn about the other participants input data and intermediate results. Public-key encryption (often in combination with symmetric key encryption) is used to ensure that data and messages are only visible to the intended receiver. And finally, a PKI and its certificate authorities (CAs) are used to ensure that the participants can use certificates to distribute and trust public keys. This enables public-keys as the tool to authenticate participants and maintain the integrity and privacy of the data exchanged.

2.3 Example: computation of the mean value

This is illustrated in a small example. The goal is to calculate the mean value of the values \( x_1, \ldots, x_N \) from \( N \) participants. With the help of the PKI, its certificates, and CAs, the coordinator \( c \) has access to and can trust the public keys \( P_1, \ldots, P_N \) of the \( N \) participants. The sub-process of each participant \( i \) will perform the calculation \( r_i = r_{i-1} + x_i \). To achieve SMC, the coordinator starts the computation by sending a large random number \( r_0 \) to the first participant. Since each message is encrypted with the public-key of the receiver, only the first participant can see this message. Each participant \( i \) adds its value \( x_i \) to the sum and forwards the message to the next participant in the chain. Also this message is encrypted with the public-key of the receiver. This continues to the last participant \( N \) who performs the same task and forwards the message encrypted with the public-key \( P_c \) of the coordinator to the coordinator \( c \). The coordinator subtracts the secret large random number from the sum and calculates the mean value \( m \):

\[
m = \frac{r_N - r_0}{N}
\]

The computing path is generated at the coordinator \( c \), and each participant only sees the next participant in the path. To achieve this the coordinator generates the following graph representation \( B_1 \):

\[
\begin{align*}
&\{(A_2, P_2), \ldots \{(A_N, P_N)\}, \{(A_c, P_c), n\}_{P_N-1}, \ldots \}_{P_1}, \\
&\{r_i\}_{i=1}^N
\end{align*}
\]

\( P_i \) is the the public key of \( i \), where \( i \) is either one of the participants \( 1, \ldots, N \) or coordinator \( c \). \( \{\ldots\}_{P_i} \) says that the message (inside the curly braces) is encrypted with \( P_i \). \( A_i \) and \( P_i \) are the address and public key of \( i \). \( r_i \) is the intermediate sum at participant \( i \). \( n \) is a large random and unique number (nonce) generated by the coordinator.

In the message forwarded to each participant the intermediate result of the computation \( r_i \) also has to be included. To the first participant the coordinator includes the secret large random number \( r_0 \). The message forwarded to the first participant will then be \( \{r_0, B_1\}_{P_1} \). However, since \( B_1 \) already is encrypted with the public key of participant 1,
there is no need to re-encrypt that part of the message. Instead the message forwarded to each participant \( i \) will have the following structure:

\[
\{\{r_{i-1}\}_i, B_i\}
\]

Each participant can only see the part of the message that is encrypted with its public key. The rest of the message is treated as a data blob that is forwarded unmodified to the next participant. Any participant \( i \) can see the following received message:

\[
\{\{r_{i-1}\}_i, \{(A_{i+1}, P_{i+1}), B_{i+1}\}_i\}
\]

\( P_i \) is the the public key of participant \( i \), and \( \{\ldots\}_i \) says that the message-part (inside the curly braces) is encrypted with \( P_i \), \( r_{i-1} \) is the intermediate result from the previous participant in the path. \( A_{i+1} \) and \( P_{i+1} \) are respectively the address and public key of participant \( i + 1 \), the next participant in the path. \( B_{i+1} \) is the data blob that participant \( i \) is going to forward unmodified to participant \( i + 1 \). Participant \( i \) performs the calculation \( r_i = r_{i-1} + x_i \) and forwards the following message to the next participant \( A_{i+1} \) in the path:

\[
\{\{r_i\}_{i+1}, B_{i+1}\}
\]

The original message created by the coordinator specifies the computing graph (a path between the sub-processes in this example). Any participant \( i \) uses the address \( A_{i+1} \) to find the next node in the graph and the public key \( P_{i+1} \) to securely communicate the intermediate results to the next node in the graph. Figure 1 illustrates this example with a coordinator \( c \) and three participants 1, 2 and 3. The start of each edge in the directed graph illustrates how the senders see the message it is forwarding. The end of the edge illustrates how the receiver sees the message. The difference between what the sender and receiver can see is due to the message part \( B_i \) that is encrypted with public key \( P_i \) of the receiver \( i \). The coordinator encrypted \( B_i \) and only \( i \) has the private key that can decrypt \( B_i \). The sender \( i \) has the knowledge about the intermediate value \( r_i \) since it calculated it. After it was calculated, sender \( i \) encrypted it with the public key of receiver \( i + 1 \). To summarize, at each node the following operations are performed:

(0) Coordinator \( c \) prepares and forward the message \( \{\{r_0\}_1, B_1\} \) to participant 1. Both \( r_0 \) and \( n \) are large random numbers. \( B_1 \) is important since it specifies the computing path and includes the public keys needed to securely forward the intermediate results to the next participant.

(1) Participant 1 decrypts and interprets the received message and uses its content to perform the calculation and securely forward the message to the next participant. The part of the message that it cannot decrypt \( (B_2) \) is forwarded to the next participant unchanged.

(2) Participant 2 decrypts and interprets the received message and uses its content to perform the calculation and securely forward the message to the next participant. The part of the message that it cannot decrypt \( (B_3) \) is forwarded to the next participant unchanged.

(3) Participant 3 decrypts and interprets the received message and uses its content to perform the calculation and securely forward the message to the coordinator. Participant 3 do no necessarily know that the receiver...
of its message is the coordinator. It uses the address and public key found in the message it received. The part of the message that it cannot decrypt ($B_c$) is forwarded to the coordinator unchanged.

(4) The coordinator decrypts and interprets the received message and uses its content, the stored value $r_0$, and the known number of participants, to perform the final calculation of the mean value $m$. Before the actual calculation is performed the coordinator will validate that the received nonce $n$ matches the value it originally included in $B_1$.

3. A more generic approach to SMC

The example above illustrates a simple computing graph, a simple data set, and a simple algorithm. The computing graph is a simple path starting and ending at the coordinator. The data set is a single (intermediate) value updated based on local data at each sub-process. The algorithm at each sub-process is to add a local value to the received intermediate result. We will for now ignore the SMC algorithms and focus on a generic approach to represent and distribute the computation graph and the data set.

3.1 Supporting a wide range of graphs

As described above the coordinator initially generates the computation graph $G$. It is wrapped in the recursively encrypted blob that is unwrapped one layer at the time at each processing node using the private key of the node. In the generic approach each layer does not provide a single receiver of the intermediate results from the current node. It provides a set of receivers. A node $i$ will receive and unwrap an input data set $I_i$ and a set of data blobs $B_i$ containing the necessary information needed to forward its intermediate results to the next set of processing nodes. $I_i$ represents sufficient data to perform the calculation. It might be aggregated from a set of input messages containing a subset $I_i'$ of the data. The example above illustrates a simple computing graph, a simple data set, and a simple algorithm. The computing graph is a simple path starting and ending at the coordinator. They are used to validate that the computation has followed the specified paths in the computation graph.

At each step in the computation the following operations are performed:

1. Node 1 and 2 decrypts and interpret the received message and performs the following operation to create the intermediate results $I_{(i,j)}$, $i, j \in 1, 2$ forwarded to the next set of nodes in the computation graph:

$$I'_{(i,j)} = F(I_i, X_i), (i, j)$$

2. Node (1, 1), (1, 2), (2, 1) and (2, 2) decrypts and interprets the received message and performs the following operation to create the intermediate results $I_{(i,j,c)}$, $i, j \in 1, 2$ forwarded to the coordinator $c$:

$$I'_{(i,j,c)} = F(f(I_{(i,j)}, X_{(i,j)}), c)$$

3. The coordinator collects, decrypts and interprets the intermediate results from node (1, 1), (1, 2), (2, 1) and (2, 2) and forwards the two messages $c_{i, k}$, $i, k \in 1, 2$ to all receivers $B_{i,k}$. All receivers need to perform their tasks.

To illustrate the above method we will now use the example from Section 3.1. In the example we consider the following situations:

- The coordinator initially generates $G$ and forwards the two messages $c_{i, k}$, $i, k \in 1, 2$ to all receivers $B_{i,k}$.
- All receivers decrypt and interpret the received message and perform the following operation to create the intermediate results $I_{(i,k)}$, $i, k \in 1, 2$ forwarded to the next set of processing nodes.

By using the above method we can now perform the following operations:

1. Node 1 and 2 decrypts and interpret the received message and performs the following operation to create the intermediate results $I_{(i,j)}$, $i, j \in 1, 2$ forwarded to the next set of nodes in the computation graph:

$$I'_{(i,j)} = F(I_i, X_i), (i, j)$$

2. Node (1, 1), (1, 2), (2, 1) and (2, 2) decrypts and interprets the received message and performs the following operation to create the intermediate results $I_{(i,j,c)}$, $i, j \in 1, 2$ forwarded to the coordinator $c$:

$$I'_{(i,j,c)} = F(f(I_{(i,j)}, X_{(i,j)}), c)$$

3. The coordinator collects, decrypts and interprets the intermediate results from node (1, 1), (1, 2), (2, 1) and (2, 2) and forwards the two messages $c_{i, k}$, $i, k \in 1, 2$ to all receivers $B_{i,k}$.

Based on the graph representation $G$, the coordinator $c$ generates and forwards the two messages $M_1$ and $M_2$ for the first two processing nodes in the graph. Each message contains the filtered data set encrypted with the receivers’ public key and the information the receivers need to perform their tasks.
and (2, 2) in the computation graph. The intermediate results are then used to calculate the final results of the computation.

### 3.2 Authenticity and integrity

Above we have focused on the computation, communication and protection of data. Authenticity and integrity is also important in a system like this. The receiver of a message has to be assured that the message is received from an expected participant in the computation and that the message has not been altered with. Based on the above description of the system it is possible for ill-behaved nodes to generate messages that trigger a receiving node to perform the local calculation and disclose data in conflict with privacy concerns. To protect against such attacks each node signs every messages it forwards to other nodes. This signature is verified before the message is interpreted. It is verified both to check that the sender is who is claimed to be the sender and that the message is expected from the sender. The last verification is done against information found in the part of the computation graph unwrapped by the receiving node.

At node \( i \) a message \( M_i \) contains data blob \( B_i \) encrypted with the public key \( P_i \). Above we have seen that this data blob is a layered representation of the computing graph with address/key pairs and sub-layers of the graph. So far we have ignored that it also contains an identifier of whom you should expect to receive this message from. This identifier is verified against the signature of the message. Since \( B_i \) was generated and encrypted at the coordinator we can be assured that the intention was that node \( i \) is expected to receive message \( M_i \) from the sender. For this to be valid \( B_i \) has to be signed by the coordinator. Message \( M_{(1,1)} \) from Figure 2 with signatures can then be written like this:

\[
M_{(1,1)} = \{ \{ I_{(1,1)} \}, B_{(1,1)} \} \}
\]

The meaning of \( D_{S_i} \) is that \( D \) is signed by \( i. \) \( B_{(1,1)} \) is signed by the coordinator \( c \):

\[
B_{(1,1)} = \{ \{ S_{(1,1)}, P_{c}, B_{(1,1)} \} \}
\]

The trust we can establish from this is the following:

(i) Message \( M_{(1,1)} \) originates from node 1 since it is signed by 1.

(ii) Message \( M_{(1,1)} \) was expected to come from node 1 since the first element of \( B_{(1,1)} \) is its identifier.

(iii) \( B_{(1,1)} \) originates from the coordinator since it is signed by \( c. \)

(iv) \( B_{(1,1)} \) is created for node (1, 1) since it is encrypted with its public key. It can also be concluded that this was the coordinator’s intention since the encrypted \( B_{(1,1)} \) is signed by the coordinator.

Every message and data blob in this infrastructure are encrypted and signed as illustrated with message \( M_{(1,1)} \) above. This ensures the authenticity and integrity of the messages and their content.

### 3.3 Progress and failure

In the example illustrated in Figure 2 each processing nodes is receiving a single message. The coordinator is however collecting the intermediate results from 4 different nodes. Before the coordinator can perform its final computation it waits for the intermediate results from all 4 nodes. In the generic case it is also possible that one of the processing nodes should receive input from more than one node. The calculation in a given node (and at the coordinator) is performed when the node has received sufficient data to perform the calculation. Sufficient data can mean all possible inputs, a given number of inputs, a percentage of the possible inputs, and all this in a combination with timeout values (to ensure progress by sacrificing the quality of the results). At a given node \( t \) the input data \( I_t \) to the calculation \( f \) should be interpreted as sufficient data to perform the calculation. This calculation is blocked until sufficient data is available.

Table 1 lists some examples of sufficient data specifications where \( t \) is time since first input arrived, \( n \) is the number

![Fig. 2: The computation graph G with six participants and a coordinator c including the processing performed and messages transmitted in the computation.](image-url)
of inputs, and \(|N|\) is the number of expected inputs. The meaning of the expression \(t \rightarrow v\) is “when time \(t\) passes the timeout value \(v\).”

Table 1: Examples of sufficient data specifications.

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t \leq 120 \land n =</td>
<td>N</td>
</tr>
<tr>
<td>((t &lt; 180 \land n =</td>
<td>N</td>
</tr>
<tr>
<td>((t &lt; 60 \land n \geq 10) \lor (t \rightarrow 60 \land n \geq 8))</td>
<td>Either wait less than 60 seconds for 10 inputs or wait 60 seconds and get at least 8 inputs. Otherwise abort.</td>
</tr>
</tbody>
</table>

It is possible for messages to get lost and nodes to fail, and the sufficient data specification can be used in algorithms to ensure progress even if this occurs. In some circumstances it is not possible to ensure progress. A node that has not yet received any messages are not a part of the computation and have no roles in detecting such errors. A node that has received some input but not sufficient data to perform its calculation will abort the computation at a specified timeout time. It will then forward an abort message to the receivers it knows about. A node that receives an abort message will ignore it if it has sufficient data to perform the calculation. Otherwise it will aggregate the received abort messages in an abort message that is forwarded to the receivers it knows about. However, a node that receives an abort message might have to wait for a timeout before it can conclude that it should forward abort messages to its known receivers. It could be that it gets sufficient data later even if it receives an abort message at a state without sufficient data. In that case the abort message should be ignored.

4. Evaluation

The infrastructure and tool set described above are evaluated with an example. The purpose of this evaluation is to demonstrate its usability.

4.1 Pearson’s \(r\)

The Pearson product-moment correlation coefficient (Pearson’s \(r\)) measure of the correlation (linear dependence) between \((n\) samples of) two variables \(x\) and \(y\):

\[
r = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2 \sum_{i=1}^{n} (y_i - \bar{y})^2}}
\]

In the case of \(m\) health institutions with \(s_j\) samples of \(x_{ji}\) and \(y_{ji}\) at each institution, \(r\) can be rewritten like this:

\[
r = \frac{\sum_{j=1}^{m} \sum_{i=1}^{s_j} (x_{ji} - \bar{x})(y_{ji} - \bar{y})}{\sqrt{\sum_{j=1}^{m} \sum_{i=1}^{s_j} (x_{ji} - \bar{x})^2 \sum_{j=1}^{m} \sum_{i=1}^{s_j} (y_{ji} - \bar{y})^2}}
\]

At each node \(j\) the following three intermediate results have to be calculated:

\[
a_j = \sum_{i=1}^{s_j} (x_{ji} - \bar{x})(y_{ji} - \bar{y})
\]

\[
b_j = \sum_{i=1}^{s_j} (x_{ji} - \bar{x})^2
\]

\[
c_j = \sum_{i=1}^{s_j} (y_{ji} - \bar{y})^2
\]

We can then at the coordinator generate the following representation of the computation graph:

\[
\left\{(A_1, P_1), \{c, (A_c, P_c), B_{(1,c)}\} p_1\right\}_{S_e};
\]

\[
(A_2, P_2), \{c, (A_c, P_c), B_{(2,c)}\} p_2\right\}_{S_e};
\]

\[
\vdots
\]

\[
(A_m, P_m), \{c, (A_c, P_c), B_{(m,c)}\} p_m\right\}_{S_e}
\]

From this graph the coordinator generates the following message to each health institution:

\[
M_j = \left\{(\bar{x}, \bar{y}) p_j; \{c, (A_c, P_c), B_{(j,c)}\} p_j\right\}_{S_e}
\]

The initial mean values \(\bar{x}\) and \(\bar{y}\) can be securely calculated using an approach similar to the one discussed above in 2.3. At each health institution \(j\) the three intermediate results are calculated and the they are included in the following message to the coordinator \(c\):

\[
M_{(j,c)} = \left\{(a_j, b_j, c_j) p_c; B_{(j,c)}\right\}_{S_j}
\]

The coordinator validates the signatures (and nonces) of the received messages and when all input is received Pearson’s \(r\) is calculated:

\[
r = \frac{\sum_{j=1}^{m} a_j}{\sqrt{\sum_{j=1}^{m} b_j \sum_{j=1}^{m} c_j}}
\]
4.2 Discussion

The computation in Pearson’s $r$ is a MapReduce. We have demonstrated that MapReduce based SMC algorithms can be implemented using our infrastructure and toolset. Earlier we have also demonstrated the implementation of other types of computation graphs and algorithms. The provided infrastructure and toolset supports a wide range of computing graphs and SMC algorithms.

In this paper we have ignored the details of the current Python based prototype implementation and we have not provided any code examples. This is done to keep the focus on the overall overview of the system and to avoid any implementation details. It is believed that the described system can be implemented using a wide range of programming languages, PKIs, crypto libraries and message systems. The current Python prototype demonstrates the main concepts and is one of the candidates for further development in a production system.

5. Conclusion

The privacy of the patients is granted by the combination of the SMC algorithms and the infrastructure and toolset discussed above. A PKI and public key encryption ensures confidentiality. Nodes not part in a computation has no access to any data or intermediate results, and nodes part in a computation only sees data and intermediate results explicit made available for them. When designing a computation it is now possible to focus on the SMC algorithm itself and not how to ensure confidentiality of data flowing through the system.

6. Acknowledgement

A special thank you to Johan Gustav Bellika who introduced me to this problem domain and the Snow SMSC project ideas. Also thank you to Yigzaw Kassaye Yitbarek for feedback and comments on this work.

References