Privacy Preserving $n$-Party Scalar Product Protocol

Florian van Daalen, Graduate Student Member, IEEE, Lianne Ippel, Andre Dekker, and Inigo Bermejo

Abstract—Privacy-preserving machine learning enables the training of models on decentralized datasets without the need to reveal the information, both on horizontally and vertically partitioned data. However, it requires specialized techniques and algorithms to perform the necessary computations. The privacy preserving scalar product protocol, which enables the dot product of vectors without revealing them, is a popular example for its versatility. For example it can be used to perform analyses that require counting the number of samples which fulfill certain criteria defined across various sites, such as calculating the information gain at a node in a decision tree. Unfortunately, the solutions currently proposed in the literature focus on two-party scenarios, even though scenarios with a higher number of data parties are becoming more relevant. In this article, we propose a generalization of the protocol for an arbitrary number of parties, based on an existing two-party method. Our proposed solution relies on a recursive resolution of smaller scalar products. After describing our proposed method, we discuss potential scalability issues. Finally, we describe privacy guarantees and identify any concerns, as well as comparing the proposed method to the original solution in this aspect. Additionally we provide an online repository containing the code.

Index Terms—Federated learning, $n$-party scalar product protocol, privacy preserving.

I. INTRODUCTION

Federated learning is a field that has recently grown in prominence due to increasing awareness of data privacy issues and data ownership as well as the rising need to combine data originating from different sources [1]. It is a thriving research field that promises to make it possible to apply machine learning algorithms (or any other data analysis) on multiple decentralized datasets in a collaborative manner [2]. This applies to both horizontally and vertically split data. Horizontally partitioned data describes the situation where different organizations collect the same information from different individuals (e.g., the same clinical data collected in multiple hospitals). Vertically partitioned data occurs when different organizations collect different information about the same individuals (e.g., insurance claims and hospital records).

In order to apply machine learning algorithms on decentralized data, various techniques have been proposed to run the necessary analyses in a privacy-preserving manner. The techniques for vertically partitioned data are generally referred to with the umbrella term of secure multiparty computation (SMPC) [3]. SMPC is a research field that focuses on developing methods to calculate functions on decentralized data without revealing the data to other parties.

Examples of the various proposed techniques are machine learning algorithms to train Bayesian networks [4], neural networks [5], or random forests [6]. These algorithms may rely on techniques such as secret sharing [7] and homomorphic encryption [8]. Both secret sharing and homomorphic encryption work at their core by transforming the original values $\alpha$ and $\beta$, owned by different parties, into transformed values $\gamma$ and $\delta$ such that $f(\alpha, \beta) = g(\gamma, \delta)$, thus making it possible to calculate the result of function $f(\alpha, \beta)$ by calculating a different function $g(\gamma, \delta)$ without ever needing to reveal $\alpha$ or $\beta$. In the case of homomorphic encryption, this is achieved by using encryption schemes that are ‘homomorphic’ with respect to specific functions, allowing the user to calculate these functions using encrypted data [8]. In the case of secret sharing, the core concept relies on obfuscating the raw data with a secret share (e.g., a random number), and then applying calculations to the obfuscated data in such a way that the secret shares will cancel out in the end [7].

Other techniques focus on specific calculations that can be used as building blocks for machine learning algorithms, such as the scalar product (or dot product) of vectors. The scalar product is an integral part of various machine learning algorithms, such as neural network training [9]. Therefore, secure scalar product protocols have been widely studied in federated learning [10]. In addition, it can be used in combination with clever data representations to calculate various statistical measures in a privacy-preserving manner, such as the information gain of an attribute, as well as to classify an individual using a decision tree in a federated setting [10]. More generally speaking the scalar product protocol can be employed to determine the size of a subset of the population that fulfills a set of criteria in a privacy preserving manner, even if the relevant attributes are spread across multiple data owners.

Because of its importance, multiple scalar product variants have been proposed. Du and Atallah proposed several methods for the scalar product [11], [12]. Du et al. also proposed a
similar method for secure matrix multiplication to be used in multivariate statistical analysis [13]. Vaidya and Clifton [14] proposed a new method to alleviate the scalability issues of existing methods and used this method to determine globally valid association rules. Du and Zhan [10] proposed yet another alternative, with better time complexity than the method proposed by Vaidya and Clifton [14], and better communication cost than the methods proposed by Du and Atallah [11], [12]. Du and Zhan [10] then used it to train a decision tree in a federated setting. Goethals et al. [15] discovered certain privacy flaws in some of the earlier mentioned protocols, and suggested an alternative with improved privacy guarantees. Shmueli and Tassa utilize a scalar product protocol to solve a problem with $n$ parties [16], however, it should be noted that they solely use the scalar product protocol to solve multiple independent 2-party sub-problems.

However, all these solutions focus on two party scenarios where the scalar product is concerned. Translating them to scenarios involving more than two parties is not straightforward, if at all possible. This is a significant drawback since in practice often three, or even more parties, can be involved.

In this study, we look at the method proposed by [10] and determine if, and how, it can be scaled to an arbitrary number of parties. This has applications for the various calculations which can (partially) be transformed into a scalar product problem mentioned before, such as calculating information gain or anything else that can be represented as a set-inclusion problem.

II. METHOD

In this section, we first introduce the notation used, then we describe the original solution proposed [10]. We will then try to naively translate the original solution to an $n$-party situation. This naïve translation will result in several left-over terms in the equations which need to be solved. We will then discuss how these left-over terms can be solved. We will illustrate the steps in this translation with a three-party scenario. Finally, we will give a formal definition for the $n$-party scenario.

In this article, we use lowercase letters to denote scalars (e.g., ‘$s$’), uppercase for vectors (e.g., $V$) and uppercase with a bold face for matrices (e.g., ‘$M$’).

A. Original Protocol

The original protocol [10] works as follows. Alice and Bob have different features on the same individuals and want to calculate the scalar product of their private vectors $A$ and $B$, both of size $m$ where $m$ is semi-honest commodity server we have named Merlin. The protocol consists of the following steps.

1) Merlin generates two random vectors $R_a$, $R_b$, of size $m$ and two scalars $r_a$ and $r_b$ such that $r_a + r_b = R_a \cdot R_b$, where either $r_a$ or $r_b$ is randomly generated. Merlin then sends $(R_a, r_a)$ to Alice and $(R_b, r_b)$ to Bob.

2) Alice sends $A = A + R_a$ to Bob, and Bob sends $B = B + R_b$ to Alice.

3) Bob generates a random number $v_2$ and computes $u = A \cdot B + v_2$, then sends the result to Alice.

4) Alice computes $u - (R_a \cdot \hat{B}) + r_a = A \cdot B - v_2 = v_1$ and sends the result to Bob.

5) Bob then calculates the final result $v_1 + v_2 = A \cdot B$.

It should be noted that this protocol utilizes a secret sharing approach. Because of this, the extended $n$-party protocol will utilize the same secret sharing approach.

B. Naive Translation to a Three-Party Scenario

For our three-party scenario we now have Alice, Bob and Claire who want to calculate the scalar product of their three vectors $A$, $B$, and $C$ of size $m$ as well as Merlin who will aid them in the calculation by fulfilling the role of commodity server. The first problem we encounter here is that $A \cdot B \cdot C$ does not result in a scalar, it results in another vector. This means it is impossible to simply chain the scalar product protocol. Hence, we must first translate our scalar product problem into a different form so it can be solved for multiple parties.

To do this we create three diagonal matrices, matrices where only the diagonal has non-zero values, $A$, $B$, and $C$ of size $m \times m$, using the original vectors to fill the diagonals. This allows us to calculate $A \cdot B \cdot C$, the result of which is a matrix. To turn this back into a scalar we define a function $\varphi$ which allows us to calculate the sum of the diagonal of a matrix. This means we have translated our 2-party scalar product problem into a 3-party matrix product problem where we calculate $\varphi(A \cdot B \cdot C)$. This naïve translation has a similar form as the matrix multiplication method proposed by Du et al. [13] mentioned earlier in this article, however, it includes more than two parties and all of our matrices are diagonal matrices.

It should be noted that this matrix multiplication method cannot simply be used to replace the scalar product protocol, as this would result in individual level data being shared across parties. For example, when using the scalar product protocol to build a decision tree [10], we have diagonal matrices, and the diagonal only contains 0 and 1 values. It would be trivial to deduce which positions only contained a value of 1 at all parties based on the final result using the matrix multiplication approach, which would be a major breach of privacy, as this would allow one to know which individuals were selected.

Having successfully translated our problem into a form where we can work with three parties, we will now attempt to naïvely translate the protocol. First, it should be noted that Merlin should generate random diagonal matrices instead of vectors. Second, he needs to generate an extra matrix $R_c$ and scalar $r_c$ to send to Claire. Third, we need to introduce an extra step into our protocol for Claire that is equivalent to step 4 in the two-party protocol. And last, wherever vectors owned by Alice and Bob are multiplied we must now multiply matrices owned by Alice, Bob and Claire. It should also be noted that whenever we are now multiplying matrices, we need to apply the $\varphi$ function to turn the resulting matrix into a scalar. Consequently, our naïve adapted protocol will look as follows:

1) Merlin generates three random diagonal matrices $R_{a}$, $R_{b}$, $R_{c}$ and two random scalars $r_a$, $r_b$. It then calculates a third scalar $r_c$ such that $r_a + r_b + r_c = \varphi(R_a \cdot R_b \cdot R_c)$. 2) Alice sends $A = A + R_a$ to Bob, and Bob sends $B = B + R_b$ to Alice.

3) Bob generates a random number $v_2$ and computes $u = A \cdot B + v_2$, then sends the result to Alice.

4) Alice computes $u - (R_a \cdot \hat{B}) + r_a = A \cdot B - v_2 = v_1$ and sends the result to Bob.

5) Bob then calculates the final result $v_1 + v_2 = A \cdot B$.

It should be noted that this protocol utilizes a secret sharing approach. Because of this, the extended $n$-party protocol will utilize the same secret sharing approach.
Merlin then sends \( \{ R_a, r_a \} \) to Alice, \( \{ R_b, r_b \} \) to Bob and \( \{ R_c, r_c \} \) to Claire.

2) Alice calculates \( \hat{A} = A + R_a \) and sends it to Bob and Claire, Bob sends \( \hat{B} = B + R_b \) to Alice and Claire, and Claire sends \( \hat{C} = C + R_c \) to Alice and Bob.

3) Bob generates a random number \( v_2 \) and computes \( u_1 = \varphi(\hat{A} \cdot \hat{C} \cdot B) + r_y - v_2 \), then sends the result to Alice.

4) Alice computes \( u_2 = u_1 - \varphi(R_a \cdot B \cdot \hat{C}) + r_a \), then sends the result to Claire

5) Claire then computes \( u_3 = u_2 - \varphi(R_c \cdot \hat{B} \cdot \hat{C}) + r_c \) to Bob and Claire, and Claire sends \( u_3 \) to Bob.

6) Bob then calculates the final result \( u_3 + v_2 = \varphi(A \cdot B \cdot C) - \varphi(R_a \cdot R_b \cdot R_c) \rangle \varphi(A \cdot R_b \cdot R_c) - \varphi(B \cdot R_a \cdot R_c) - \varphi(C \cdot R_a \cdot R_b) \).

As we can see our final result is not equal to \( \varphi(A \cdot B \cdot C) \) because there are several left-over terms (i.e., \( \varphi(R_a \cdot R_b \cdot R_c) \), \( \varphi(A \cdot R_b \cdot R_c) \), \( \varphi(B \cdot R_a \cdot R_c) \), and \( \varphi(C \cdot R_a \cdot R_b) \)).

C. Solving the Left-Over Terms

The first left-over that should be solved is the left-over of the form \( \varphi(R_a \cdot R_b \cdot R_c) \). The protocol will naturally result in a left-over term of the form \( \varphi(R_a \cdot R_b \cdot R_c) \rangle \varphi(A \cdot R_b \cdot R_c) - \varphi(B \cdot R_a \cdot R_c) - \varphi(C \cdot R_a \cdot R_b) \).

As we can see our final result is not equal to \( \varphi(A \cdot B \cdot C) \) because there are several left-over terms (i.e., \( \varphi(R_a \cdot R_b \cdot R_c) \), \( \varphi(A \cdot R_b \cdot R_c) \), \( \varphi(B \cdot R_a \cdot R_c) \), and \( \varphi(C \cdot R_a \cdot R_b) \)).

E. Full Translation to an n-Party Scenario

The n-party protocol can be formalized as follows:

1) If \( n = 2 \), use the two-party protocol [10], else go to next step.
2) Let \( D_1, D_2, \ldots, D_n \) be the diagonal matrices containing the vectors owned by the \( n \) parties.
3) Let \( \varphi \) be a function that calculates the sum of the diagonal of a matrix.
4) \( R_1, R_2, \ldots, R_n \) are random diagonal matrices generated by a commodity server Merlin.
5) Let \( \varphi(R_1 \cdot R_2 \cdots R_n) = r_1 + r_2 + \cdots + r_n \) where all but one of the \( r_i \) terms are randomly generated.
6) Merlin shares the pairs \( \{ R_i, r_i \} \) with the \( i \)’th party for each \( i \in [1, n] \).
7) All parties calculate \( \hat{D}_i = D_i + R_i \) and share the result.
8) Party 1 generates \( v_2 \).
9) Party 1 then calculates \( u_1 = \varphi(\prod_{i=2}^{n} \hat{D}_i \cdot D_1) + (n - 1) \cdot r_1 - v_2 \).
10) For each other party \( i \) calculate \( u_i = u_{i-1} - \varphi(\prod_{j=1}^{i} \hat{D}_j | x \neq i \rangle R_i) + (n - 1) \cdot r_i \).
11) This results in \( \varphi(\prod_{1}^{n} D_1 \cdots D_n) - L_1 - L_2 - \cdots \cdots L_n - v_2 \) Where \( L_i \) corresponds to leftover terms of the form \( \varphi(\prod_{m=1}^{n} D_i \prod_{j=m}^{n} R_j \langle i \neq j \rangle) \), where all particles are involved, either as \( D_i \), providing their raw data, or as \( R_j \), using their random matrix, but never as both.
12) These leftover terms represent a scalar product problem of at most \( n - 1 \) parties. Thus these sub problems can be solved separately using a smaller \( n \)-party scalar product protocol.

1A full elaboration of the equation can be found in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2023.3238768
2A practical example of a 3-party scalar product protocol can be found in Appendix B, available in the online supplemental material
Algorithm 1: The n-Party Scalar Product Protocol.

\[
\text{nPartyScalarProduct}(D)\]

\textbf{Input:} The set $D$ of diagonal matrices $D_1 \ldots D_n$ containing the original vectors owned by the $n$ parties

\textbf{Output:} $\varphi(D_1 \cdot D_2 \ldots \cdot D_n)$

1. if $|D| = 2$
2. Else
3. for $i \leftarrow 0$ to $|D|$ by 1 do
4. \hspace{1em} $R_i \leftarrow \text{generateRandomDiagonalMatrix}()$
5. \hspace{1em} end
6. \hspace{1em} Let $\varphi(R_1 \cdot R_2 \ldots \cdot R_n) = r_1 + r_2 + \ldots + r_n$
7. \hspace{1em} Share $(R_i, r_i)$ with the $i$th party for each
8. \hspace{1em} $i \in [1, n]$
9. \hspace{1em} end
10. $v_2 \leftarrow \text{randomInt}()$
11. $u_1 \leftarrow \varphi(\prod_{i=2}^n D_i \cdot D_1) + (n - 1) \cdot r_1 - v_2$
12. for $i \leftarrow 2$ to $|D|$ by 1 do
13. \hspace{1em} $u_i = u_{i-1} - \varphi(\prod_{x=1}^n D_x | x \neq i) \cdot R_i$
14. \hspace{1em} $+ (n - 1) \cdot r_i$
15. \hspace{1em} end
16. $y \leftarrow u_n$
17. for $\text{subprotocol} \in \text{determineSubprotocols}(D, R)$ do
18. \hspace{1em} $y \leftarrow y - \text{nPartyScalarProduct}(\text{subprotocol})$
19. \hspace{1em} end
20. return $y + v_2$
21. determineSubprotocols($D, R$)

\textbf{Input:} The set $D$ of diagonal matrices $D_1 \ldots D_n$ of the original protocol. The set $R$ of random diagonal matrices used in the original protocol

\textbf{Output:} The sets $D_{\text{subprotocol}}$ for each subprotocol

22. for $k \leftarrow 2$ to $|D| - 1$ by 1 do
23. \hspace{1em} uniqueCombinations $\leftarrow \text{selectKSizeCombosFromSet}(k, D)$
24. \hspace{1em} for $\text{selected} \in \text{uniqueCombinations}$ do
25. \hspace{1em} \hspace{1em} subprotocol $\leftarrow D_i[i \in \text{selected} + R_j[j \notin \text{selected}]$
26. \hspace{1em} \hspace{1em} $D_{\text{subprotocols}} \leftarrow D_{\text{subprotocols}} + \text{subprotocol}$
27. \hspace{1em} end
28. end
29. return $D_{\text{subprotocols}}$

13. Solving these leftover terms allows party $n$ to calculate $\varphi(D_1 \cdot D_2 \ldots \cdot D_n) - v_2 = u_n$

14. Party 1 can then calculate the final result $u_n + v_2 = \varphi(D_1 \cdot D_2 \ldots \cdot D_n)$

This allows us to calculate the scalar product for an arbitrary amount of parties. Pseudocode of the protocol can be found in Algorithm 1. Now that we have shown that the protocol can be translated to a scenario with arbitrary $n$ we will discuss how the protocol scales as well as potential security issues in the next section.

F. Commodity Server

The $n$-party scalar product protocol contains multiple sub-protocols of at most $n - 1$ sized all of which involve data owned by the commodity server in the $n$-party scalar protocol. These sub protocols will need to use a commodity server as well. However, the original commodity server Merlin cannot be reused as Merlin fulfills the role of data-owner in these sub protocols. In Section III-B we will discuss what influence this will have as $n$ grows and how potential issues can be minimized.

III. DISCUSSION

In this article, we have translated an existing 2-party scalar product [10] protocol to an $n$-party protocol. We have shown that a naïve translation is insufficient. However, by using a more sophisticated approach, it is possible to adapt the protocol to work with an arbitrary number of parties. In Appendix B, available in the online supplemental material, a fully worked out example of the three-party protocol can be found. Appendix C, available in the online supplemental material, provides references to a repository containing java and python implementations of the $n$-party protocol.

We will now discuss the security and privacy guarantees this $n$-party protocol provides as well as how the complexity scales as the number of parties grows and how practical it is to use this protocol.

A. Security

The proposed method requires a commodity server, which is a semi-honest trusted third party within the calculation. A semi-honest party is a party which executes its part in the protocol accurately, but may try to learn as much as it can from the messages it receives in the process [17]. In this section we will discuss the exact risks involved with this.

As a method that relies on secret shares generated by a semi-trusted third party, this protocol utilizes an approach similar to asymmetric encryption [18], with the individual secret shares performing the role of private keys. This limits the risks involved. However, the trusted third party does introduce a risk in itself.

The risk posed by requiring a semi-honest trusted third party to be the commodity server would be that several semi-honest parties could potentially cooperate with the commodity server in order to jointly learn private data of the other parties. It should be noted that this risk is higher in an Internet of Things (IoT) setting than in a formalized joint research setting. An IoT setting consists of many unverified devices and parties. A formal joint research setting allows all parties involved to verify, and enforce, for example by requiring audits and adding other legal agreements, the integrity of the other parties to a certain extent. This will minimize the risk in practice in this setting. While it would be preferable if privacy could be protected by design with technical solutions, there will always be a need for a certain degree of trust in the various parties involved and legal means are a perfectly acceptable way of achieving the required trust [2].

However, this does not remove the technical possibility of a joint attack when all parties are semi-honest. The local calculations done at a given node $i$ are always of the form: $u_i = u_{i-1} - \varphi(\prod_{x=1}^n D_x | x \neq i) \cdot R_i + (n - 1) \cdot r_i$. Where $D_x$ is locally known by every data-owner participating in this protocol. However, $D_x$ is unknown to the commodity server in this
protocol. Assuming the node cooperates with the commodity server, they could then separate $\hat{D}_x$ into its components $D_x$ and $R_x$. Where $D_x$ is private data belonging to a different party and $R_x$ is the random diagonal matrix generated by the commodity server, thus learning $D_x$. This is a serious concern. This issue is especially relevant in an IoT setting where the trustworthiness of the commodity servers and individual parties is very difficult to verify and enforce.

However, in a formal joint research setting, a sufficient level of trust can be achieved to minimize the risk of this attack by enforcing the commodity server to act as an honest party, not just semi-honest [2], [19] First, it is possible to simply enforce this using legal means and mandate it is honest, however this may not be accepted in practice. Second, it is possible to give all parties involved joint custody over the commodity servers, thus allowing each party to individually verify the commodity server is completely honest.

Joint custody over the commodity servers could, for example, be achieved by allowing any party to execute independent audits of the commodity server and giving them a veto over the hardware and software setup used on the servers. Such a setup allows each party to individually verify that the commodity server is honest, which works because each party has a vested interest in ensuring the honesty of the commodity server to protect their own data. This should allow the parties to jointly guarantee the commodity server are honest, even if the individual parties themselves are semi-honest.

It is important to note that these security concerns, and the possible solutions, are the same regardless of the size of $n$. That is to say, our proposed $n$-party protocol is equally as secure as the original 2-party protocol proposed by Du and Zhan because the original protocol also uses a trusted third party as commodity server which as we just discussed is the vulnerability exploited in a collusion attack.

B. Scalability

The number of subprotocols will grow with a factorial order of growth with respect to $n$. The reason it scales in this manner is because the subprotocols have the form of $\varphi(\prod_{i=1}^{n} D_i \prod_{j=m}^{n} R_j | i \neq j)$. Where all parties are involved, either as $D_i$, providing their raw data, or as $R_j$, using their random matrix, but never as both. There will be $\frac{n!}{2^{n-2}}$ such subprotocols for each $2 \leq x < n$.

These subprotocols will have $x$ $R_i$ factors and $n-x$ $D_i$ factors. For example, a three-party protocol will have the following 3 subprotocols involving 2 $R_i$ factors: $\varphi(A \cdot R_b \cdot R_c)$, $\varphi(B \cdot R_a \cdot R_c)$ and $\varphi(C \cdot R_a \cdot R_b)$. A 4 party protocol will have 4 subprotocols involving 3 $R_i$ factors: $2 \cdot \varphi(A \cdot R_b \cdot R_c \cdot R_d)$, $2 \cdot \varphi(B \cdot R_a \cdot R_c \cdot R_d)$, $2 \cdot \varphi(C \cdot R_a \cdot R_b \cdot R_d)$, and $2 \cdot \varphi(D \cdot R_a \cdot R_b \cdot R_c)$. As well as 6 subprotocols involving 2 $R_i$ factors: $\varphi(A \cdot B \cdot R_c \cdot R_d)$, $\varphi(A \cdot C \cdot R_b \cdot R_d)$, $\varphi(A \cdot D \cdot R_b \cdot R_c)$, $\varphi(B \cdot C \cdot R_a \cdot R_d)$, $\varphi(B \cdot D \cdot R_a \cdot R_c)$ and $\varphi(C \cdot D \cdot R_a \cdot R_b)$.

This growth in subprotocols will have an effect on the scalability. We will discuss the two aspects in which this matters in the following two sections.

1) Time and Space Complexity: The first aspect affected by the factorial order of growth is the time complexity of the protocol. The amount of direct subprotocols for an $n$-party protocol will be equal to $\frac{n!}{2^{n-2}}$ for each $x \in [2; n]$. These subprotocols may also have further subprotocols themselves. Furthermore, the amount of messages that need to be send for a given protocol are as follows: 1 message needs to be send from the commodity server to each of the $n$ dataowners to share the relevant pair of $\{R_i, r_i\}$. Each party then shares its matrix $\hat{D}_i$ with each other party, resulting in $n \cdot (n - 1)$ messages. Finally each party has to share its subresult once, resulting in a further $n$ messages. This means a total of $n + n^2$ messages for a given protocol.

In order to put this into perspective we show the number of protocols as a function of $n$ in Fig. 1. In addition to this, the results of a small experiment measuring the runtime performance, where the $n$-party protocol was used to calculate the number of individuals fulfilling certain attribute requirements, can be found in Fig. 2. This experiment was run on a windows laptop using an Intel(R) Core(TM) i7-10750H processor with 16 GB of memory and 6 cores. All parties had a local datastation on this laptop, no significant optimization was implemented.

As can be seen in Fig. 1 the required number of protocols and messages grow quickly as $n$ grows. This is a significant downside of this protocol. The results of the small runtime experiment further supports this, as the runtime does grow rapidly as the number of parties grows. However, it also shows that the protocol can easily deal with larger datasets as dataset size barely influences the runtime. It should also be noted that there is considerable room for parallelization within the protocol, allowing the protocol to still be useable in practice. The following steps can be parallelized: first, every subprotocol can naturally be calculated in parallel as these are independent problems. Second every calculation in substep 11 detailed in Section II-E can be calculated in parallel as well. Both options will reduce the running time of the protocol, considerably, allowing it to still be a
practical solution in many settings. In addition to this, the actual use of the protocol within model training can be optimized, for example by running multiple $n$-party product protocols in parallel.

2) Commodity Servers: It should be noted that these subprotocols need their own commodity server because no party may be both data owner and commodity server in a given protocol. Hence, we cannot reuse the original commodity server Merlin as it fullfills the role of a data owner in the subprotocols.

A naïve solution to the problem posed by this need would be to set up sufficient commodity servers to deal with every subprotocol. However, the amount of commodity servers needed will scale linearly with $n$, since a commodity server can be shared across all subprotocols of the same size. As the largest subprotocol in an $n$-party protocol will be an $(n−1)$-party subprotocol, and a two-party protocol will have no subprotocol, we will need $n−1$ commodity servers to solve an $n$-party problem. While this might be manageable for small $n$ this eventually becomes untenable.

An alternative to this naïve solution would be to have the various parties double as commodity servers whenever they are not involved in a calculation themselves. To show that this is a viable, and safe solution, we will first divide the subprotocols into two categories. All subprotocols have the form $ϕ(∑^m_{i,j=1}D_iR_j)\forall i\neq j$, this can be further subdivided into subprotocols which contain only 1 $D_i$ term, which will have the form $ϕ(D_iR_j⋯R_m)$, and subprotocols with multiple $D_i$ terms.

The first category of subprotocols, which only contain one $D_i$ term, can be solved by simply sharing the result of random matrices $R_j$, $D_i$, $R_j$, with the owner of $D_i$. $R_j$ is itself a random matrix, provided there are at least two $R_j$ factors involved, which cannot be used to leak any information. For example, the sub-protocols in the three-party protocol can be solved this way without requiring extra commodity servers. Doing this will also be faster than using the two-party scalar product protocol as it only requires a straightforward multiplication instead of the entire scalar product protocol. It should however be noted that the solution to this subprotocol may never be revealed to the commodity server that owns the $D_i$ terms, as this would allow the commodity server to calculate $D_i$. For example, if we are calculating $A \cdot R_b \cdot R_e$ the result should never be revealed to Merlin, as revealing this would allow Merlin to learn Alice’s data. This is of course also true in the original 2-party protocol.

The second category of subprotocol, which contains multiple $D_i$ terms, can reuse one of the parties which is not currently providing data (i.e., a $D_i$ term) as the new commodity server. This is secure as there is no need to reveal anything to the commodity server during the calculation. All it needs to do is generate and share the new $\{R_i, r_i\}$ pairs for this subprotocol. As such, it never needs to see any (sub)results, and thus cannot reverse engineer anything. Additionally, the same party should never be used twice as a commodity server in any set of subprotocols. That is to say, if Alice handles a 3-party subprotocol then she should not handle any child protocols that arise as a consequence of this specific 3-party subprotocol. Fortunately, it is easy to avoid this as there will always be at least one new party available to fulfil the role of commodity server for the new subprotocols.

While this is a practical solution to the need for multiple commodity servers, it does come with the major caveat that one must be certain no parties will attempt to cooperate to jointly learn private data of the other parties. As pointed out in Section III-A, the protocol is vulnerable to this type of attack.

IV. Conclusion

In this article, we have explained how the two-party scalar product protocol by Du and Zhan [10] can be scaled to an $n$-party scalar product protocol. We have illustrated how it works using a three-party scenario, after which we have given the formal definition of the protocol for any number of parties. This protocol can be used to calculate a number of metrics, such as the information gain of an attribute [10], in a scenario with an arbitrary number of parties. The benefit of being able to calculate such metrics is that it opens up the door for other more complex analysis. For example, using the information gain one can build a decision tree or apply feature selection.

Similarly, by using an innovative data representation the $n$-party protocol can be used to classify an individual in a privacy preserving manner using a decision tree [10]. By using other innovative data representations this $n$-party protocol could potentially be used for a wide variety of analysis and calculations. Aside from these benefits, which require the problem at hand to be rephrased into a scalar product problem, there is also the obvious benefit that it allows the use of the scalar product itself in an $n$-party scenario. This allows the use of any calculation that would normally rely on the scalar product in a classical machine learning setting but which cannot be executed easily in a federated setting without a private $n$-party scalar product protocol.

While not appropriate in every scenario (scalability and the need for more commodity servers or semi-honest servers as the
number of parties grows are a practical concern), we believe this is still a valuable tool in the federated learning toolbox.

A. Future Work

For future work we would like to devise \( n \)-party protocols with better time complexity, as well as find a way to remove the vulnerability to joint-attacks introduced by the need for a commodity server.

In addition to this it would be valuable to investigate to which extend our extension to \( n \) parties can be applied to the secure matrix multiplication proposed by Du et al. [13]. The protocol used for matrix multiplication is very similar to the 2-party scalar product protocol we extended, as such our extension should be of use when extending this matrix multiplication protocol.

Lastly, we are planning to utilize the \( n \)-party scalar product protocol to implement various federated algorithms so we can test the practical viability of this protocol in a real life setting.

ACKNOWLEDGMENTS

The views expressed in this article are those of the authors and do not necessarily reflect the policy of Statistics Netherlands.

REFERENCES

[1] L. Li, Y. Fan, M. Tse, and K.-Y. Lin, “A review of applications in federated learning,” Comput. Ind. Eng., vol. 149, Nov. 2020, Art. no. 106854. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360835220305532

[2] P. Kairouz et al., “Advances and open problems in federated learning,” Dec. 2019, arXiv:1912.04977.

[3] A. C. Yao, “Protocols for secure computations,” in Proc. IEEE 23rd Annu. Symp. Found. Comput. Sci., 1982, pp. 160–164.

[4] H.-Y. Chen and W.-L. Chao, “FedBE: Making Bayesian model ensemble applicable to federated learning,” Jan. 2021, arXiv:2009.01974.

[5] M. Duan, D. Liu, X. Chen, R. Liu, Y. Tan, and L. Liang, “Self-balancing federated learning with global imbalanced data in mobile systems,” IEEE Trans. Parallel Distrib. Syst., vol. 32, no. 1, pp. 59–71, Jan. 2021.

[6] Y. Liu, Y. Liu, Z. Liu, J. Zhang, C. Meng, and Y. Zheng, “Federated forest,” IEEE Trans. Big Data, vol. 8, no. 3, pp. 843–854, Jun. 2022, arXiv.org/abs/1905.10053.

[7] A. Beimel, “Secret-sharing schemes: A survey,” in Proc. Int. Conf. Coding Cryptol., 2011, pp. 11–46.

[8] P. Y. Parmar, S. B. Padhar, S. N. Patel, N. I. Bhatt, and R. H. Jhaveri, “Survey of various homomorphic encryption algorithms and schemes,” Int. J. Comput. Appl., vol. 91, no. 8, pp. 26–32, Apr. 2014. [Online]. Available: http://research.ijcaonline.org/volume91/number8/pxc3895081.pdf

[9] S. Wiedemann, K.-R. Müller, and W. Samek, “Compact and computationally efficient representation of deep neural networks,” IEEE Trans. Neural Netw. Learn. Syst., vol. 31, no. 3, pp. 772–785, Mar. 2020.

[10] W. Du and Z. Zhan, “Building decision tree classifier on private data,” in Proc. IEEE Int. Conf. Privacyscur. Data Mining, 2002, pp. 1–8.

[11] W. Du and M. Atallah, “Privacy-preserving cooperative statistical analysis,” in Proc. 17th Annu. Comput. Secur. Appl. Conf., 2001, pp. 102–110. [Online]. Available: https://ieeexplore.ieee.org/document/991526/

[12] M. J. Atallah and W. Du, “Secure multi-party computational geometry,” in Proc. Workshop Algorithms Data Struct., 2001, pp. 165–179. [Online]. Available: http://link.springer.com/10.1007/3-540-44643-6_16

[13] W. Du, Y. S. Han, and S. Chen, “Privacy-preserving multivariate statistical analysis: Linear regression and classification,” in Proc. SIAM Int. Conf. Data Mining, 2004, pp. 222–233. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611972740.21

[14] J. Vaidya and C. Clifton, “Privacy preserving association rule mining in vertically partitioned data,” in Proc. 5th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2002, pp. 639–644.

[15] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen, “On private scalar product computation for privacy-preserving data mining,” in Proc. Int. Conf. Inf. Secur. Cryptol., 2005, pp. 104–120. [Online]. Available: http://link.springer.com/10.1007/11496618_9

E. Shmueli and T. Tassa, “Mediated secure multi-party protocols for collaborative filtering,” ACM Trans. Intell. Syst. Technol., vol. 11, no. 2, pp. 1–25, 2020.

Q. Do, B. Martini, and K.-K. R. Choo, “The role of the adversary model in applied security research,” Comput. Secur., vol. 81, pp. 156–181, Mar. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818306369

T. Schneider and A. Treiber, “A comment on privacy-preserving scalar product protocols as proposed in “SPOC’,” IEEE Trans. Parallel Distrib. Syst., vol. 31, no. 3, pp. 543–546, Mar. 2020.

N. Truong, K. Sun, S. Wang, F. Guittin, and Y. Guo, “Privacy preservation in federated learning: An insightful survey from the GDPR perspective,” Comput. Secur., vol. 110, 2021, Art. no. 102402. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821002261

Florian van Daalen (Graduate Student Member, IEEE) received the BSc degree in knowledge engineering from the University Maastricht, in 2012, and the MSc degree in artificial intelligence, in 2014. He is currently working toward the PhD degree in clinical data science with the Clinical Data Science Group, University Maastricht, The Netherlands. His research interests include privacy preserving techniques, federated learning, and ensemble based learning.

Lianne Ippel received the PhD degree in statistics from Tilburg University on analyzing data streams with dependent observations, for which she won the dissertation award from General Online Research conference (2018). After a postdoc with Maastricht University, she now works with Statistics Netherlands where she works at the Methodology Department on international collaborations and innovative methods for primary data collection.

Andre Dekker received the PhD degree, in 1974. He is a medical physicist and professor of clinical data science with Maastricht University Medical Center and Maastro Clinic in The Netherlands. His Clinical Data Science Research Group (50 staff) focuses on 1) federated FAIR data infrastructures, 2) AI for health outcome prediction models and 3) applying AI to improve health. He has authored more than 200 publications, mentored more than 30 PhD students and holds multiple awards and patents on the topic of federated data and AI. He has held visiting scientist appointments at Universities and Companies in the U.K., Australia, Italy, USA, and Canada.

Inigo Bermejo received the BSc degree in computer engineering from the University of the Basque Country, Spain, in 2006, and the PhD degree in intelligent systems from UNED, Spain, in 2015. He is currently a postdoctoral researcher with the Clinical Data Science Group, Maastricht University. His research interests include privacy preserving techniques, prediction modelling, and causal inference.