A Note on “Privacy Preserving n-Party Scalar Product Protocol”

Lihua Liu

Abstract. We show that the scalar product protocol [IEEE Trans. Parallel Distrib. Syst. 2023, 1060-1066] is insecure against semi-honest server attack, not as claimed. Besides, its complexity increases exponentially with the number $n$, which cannot be put into practice.

Keywords: privacy preserving scalar product, semi-honest server attack, diagonal matrix, trace map.

1 Introduction

In 2002, Du and Zhan [1] designed a privacy preserving two-party scalar product protocol. Recently, Daalen et al. [2] have generalized it to a general $n$-party protocol. In this note, we show that the Daalen et al.’s protocol is insecure against semi-honest server attack. Besides, the protocol cannot be practically implemented because of its exponential complexity.

2 The Du-Zhan two-party protocol

Alice and Bob want to calculate the scalar product of their private vectors $A$ and $B$, both of the same size $m$. Merlin is a semi-honest server, who generates two random vectors $R_a, R_b$ of size $m$ and two scalars $r_a$ and $r_b$ such that

$$r_a + r_b = R_a \cdot R_b$$ (1)

Then securely send $(R_a, r_a)$ to Alice, and $(R_b, r_b)$ to Bob. The protocol can be depicted as below (Table 1).

Clearly, we have

$$v_1 + v_2 = u - R_a \cdot \hat{B} + r_a + v_2$$
$$= \hat{A} \cdot B + r_b - v_2 - R_a \cdot \hat{B} + r_a + v_2$$
$$= (A + R_a) \cdot B - R_a \cdot (B + R_b) + r_a + r_b$$
$$= A \cdot B - R_a \cdot R_b + r_a + r_b = A \cdot B$$

Notice that the Du-Zhan protocol has two shortcomings. The first is the presence of an honest convener. In the protocol, only Bob can know the result $v_1 + v_2$. If Alice wants to know the result, it must introduce other mechanism enabling Bob to honestly and securely transfer the nonce $v_2$ to Alice.

---

L. Liu is with Department of Mathematics, Shanghai Maritime University, Shanghai, China. Email: liulh@shmtu.edu.cn
Table 1: The Du-Zhan two-party protocol

<table>
<thead>
<tr>
<th>Alice: ((R_a, r_a))</th>
<th>Bob: ((R_b, r_b))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input the private vector (A). Compute the vector (\hat{A} = A + R_a).</td>
<td>Input the private vector (B). Pick a nonce (v_2). Compute (\hat{B} = B + R_b).</td>
</tr>
<tr>
<td>Compute the scalar (v_1 = u - R_a \cdot \hat{B} + r_a).</td>
<td>(u = \hat{A} \cdot B + r_b - v_2).</td>
</tr>
</tbody>
</table>

The second is that insecurity against semi-honest server attack. 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. In the Du-Zhan protocol, once the server captured \(\hat{A}, \hat{B}\) via the open channels, it can retrieve the private vectors \(A, B\) using \(R_a, R_b\).

3 The Daalen et al.’s three-party protocol

3.1 Review of the Daalen et al.’s protocol

Table 2: The Daalen et al.’s three-party scalar product protocol

<table>
<thead>
<tr>
<th>Alice: ((R_a, r_a))</th>
<th>Bob: ((R_b, r_b))</th>
<th>Claire: ((R_c, r_c))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input the private (A). Convert it into (\hat{A}). Perform two-party scalar product protocol with Merlin who knows (R_b, R_c) to obtain (k_a = \phi(AM_a)), where (M_a = R_b R_c). Broadcast (\hat{A}).</td>
<td>Input the private (B). Convert it into (\hat{B}). Perform two-party scalar product protocol with Merlin who knows (R_a, R_c) to obtain (k_b = \phi(BM_b)), where (M_b = R_a R_c). Broadcast (\hat{B}).</td>
<td>Input the private (C). Convert it into (\hat{C}). Perform two-party scalar product protocol with Merlin who knows (R_a, R_b) to obtain (k_c = \phi(CM_c)), where (M_c = R_a R_b). Broadcast (\hat{C}).</td>
</tr>
<tr>
<td>Compute (u_1 = \phi(BAC) + 2r_b - v_2). (\leftarrow_{u_1, k_b} u_2, k_a, k_b) (\rightarrow_{u_3, k_a + k_b + k_c} u_3 = u_2 - \phi(R_c \hat{A} \hat{B}) + 2r_c + k_a + k_b + k_c).</td>
<td>(\rightarrow_{u_3} u_3 + v_2).</td>
<td></td>
</tr>
</tbody>
</table>

In the Daalen et al.’s three-party protocol [2], we use lowercase letters to denote scalars, uppercase for vectors and uppercase with a bold face for matrices. It needs to convert a vector into its
corresponding diagonal matrix. Let $\phi$ be the trace map of a matrix. The protocol can be rephrased and depicted as below (Table 2).

The server Merlin generates three random diagonal matrices $R_a$, $R_b$, $R_c$ and three scalars $r_a, r_b, r_c$ such that

$$r_a + r_b + r_c = \phi(R_a R_b R_c)$$

(2)

Send $\{R_a, r_a\}$ to Alice, $\{R_b, r_b\}$ to Bob and $\{R_c, r_c\}$ to Claire.

Its correctness is due to that

$$u_3 = u_2 - \phi(R_c \hat{A} \hat{B}) + 2r_c + k_a + k_b + k_c$$

$$= u_1 - \phi(R_a \hat{B} \hat{C}) + 2r_a - \phi(R_c \hat{A} \hat{B})$$

$$+ 2r_c + k_a + k_b + k_c$$

$$= \phi(A \hat{C} \hat{B}) + 2r_b - v_2 - \phi(R_a \hat{B} \hat{C})$$

$$+ 2r_a - \phi(R_c \hat{A} \hat{B}) + 2r_c + k_a + k_b + k_c$$

$$= \phi((A + R_a)(C + R_c)B) - \phi(R_a(B + R_b)(C + R_c))$$

$$- \phi(R_c(A + R_a)(B + R_b))$$

$$+ 2(r_a + r_b + r_c) - v_2 + k_a + k_b + k_c$$

$$= \phi(ABC) - \phi(AR_b R_c) - \phi(BR_a R_c) - \phi(CR_a R_b)$$

$$- v_2 + k_a + k_b + k_c = \phi(ABC) - v_2$$

The three-party protocol can be generalized to an $n$-party protocol, but which should recursively perform plenty of $(n-1)$-party protocols, $(n-2)$-party protocols, ..., and 2-party protocols. We refer to the original description (page 1062, [2]).

### 3.2 Insecure against semi-honest server attack

In order to solve the so-called left-over problems [2]

$$\phi(AR_b R_c), \ \phi(BR_a R_c), \ \phi(CR_a R_b)$$

it needs to use two-party scalar product protocols, where Merlin is one of the parties. A big difference between two-party protocol and three-party protocol is whether the semi-honest server involves in the procedures after the setup phase is completed.

In the three-party protocol, Alice has to perform the two-party protocol with Merlin in order to compute $k_a = \phi(AM_a)$ where $M_a = R_b R_c$, and both $R_b$ and $R_c$ are known to Merlin. That means Alice needs to send $\hat{A} = A + R_a$ to Merlin. Since Merlin knows $R_a$, he can easily recover the private diagonal matrix $A$ and the corresponding private vector $A$. Likewise, Merlin can retrieve the private vectors $B$ and $C$.

To resist the semi-honest server attack, it suggests that the role of commodity server Merlin could be jointly played by many semi-honest parties. For instance, there are $\ell$ semi-honest servers, Server$_1$, Server$_2$, ..., Server$_\ell$, where $\ell > 3$. None of them can solely access $R_a, R_b, R_c$. In this case, they need to collaboratively and securely compute the diagonal matrix

$$M_a = R_b R_c$$

(3)
But we find the above problem is just a new secure ℓ-party computation problem, which is more intractable than the original 3-party computation problem.

It also suggests that (page 1064, [2]): “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.” But we find the argument is self-contradictory. If the server is full-honest, not semi-honest, the original protocol becomes unnecessary. Actually, in this case any party-\(i\) can send \(D_i = D_i + R_i\) to the honest server, who then retrieves \(D_i\) using \(R_i\). After all vectors are collected, the server computes the final scalar product and securely sends the result to any target user.

3.3 Exponential complexity

In the proposed \(n\)-party protocol, the party-1 needs to compute

\[
\phi(D_1 R_2 R_3), \phi(D_1 R_2 R_4), \ldots, \phi(D_1 R_2 R_n);
\]
\[
\phi(D_1 R_2 R_3 R_4), \phi(D_1 R_2 R_3 R_5), \ldots;
\]
\[
\ldots
\]
\[
\phi(D_1 R_2 R_3 R_4 \cdots R_n)
\]

That means any party needs to perform \(2^{n-1} - n\) sub-protocols. The complexity increases exponentially with the number \(n\). Therefore, the \(n\)-party protocol cannot be put into practice.

4 Conclusion

We show that the Daalen et al.’s scalar product protocol is insecure against semi-honest server attack. Its massive complexity is a big issue for practical implementation. The findings in this note could be helpful for the future work on designing such protocols.

References
