PriVeil Circle: using Secure Multiparty Computation for sharing threat information

Miguel Gil da Silva Simão
Instituto Superior Técnico, Universidade de Lisboa, Portugal

***Abstract*—Nowadays, data breaches occur frequently in organizations, causing them economical and reputational losses. To prevent them more effectively, organizations need to gather cyber threat information internally and externally from sharing communities. However, organizations are still reluctant to share their own cyber threat data, since they are afraid to disclose sensitive information about their infrastructure. Anonymization of indicators is a commonly used solution when handling sensitive information, but it reduces the data utility and can be vulnerable to re-identification of the anonymized parameters.**

In this work we present *Circle*, the second component of the *PriVeil* cyber threat sharing platform. In a first phase, users submit their encrypted security reports to the *Square* component of the platform, which matches them. Users whose reports matched are forwarded to a *Circle* instance, where the Secure Multiparty Computation operation of Private Set Intersection allows the participants to find the common tags contained in their reports through a privacy-preserving computation. This allows users to confirm that other users may be subject to a similar cyber threat described in their reports.

We implemented and evaluated *Circle* with nine experiments, in which we measured the duration, CPU and memory usage of a participant program during a session. The results showed that the performance of our prototype makes it applicable to real-case scenarios, providing an environment where users can share information about cyber threats.

***Keywords* - Cyber Threat Sharing, Security Reports, Privacy, Secure Multiparty Computation, Private Set Intersection**

I. Introduction

In the Information Age, data is a major asset of every organization. A part of this data is confidential and, consequently, meant to remain private. According to [1], a data breach occurs when confidential information from the victim is disclosed to unauthorized parties. When this occurs, the organization suffers financial and reputational losses.

Hackers are becoming craftier, finding new new vulnerabilities and creating complex and destructive exploits. Furthermore, hacking is made easier by exploits and malware that can be purchased at online anonymous markets. This commoditization is lowering the capabilities needed to engage in cybercrime activities [2]

As cooperation increases amongst criminals to discover new exploits, it is almost impossible for a single organization to handle cyber threats. To counter the ever-evolving cybercrime, organizations need to adopt new defensive tactics that also increase their cooperation. In particular, sharing cyber threat data externally is a crucial way to find new vulnerabilities being exploited by attackers, new threats that affect organizations and even Indicators of Compromise (IoC).

Despite the importance of cyber threat information sharing platforms, there are still obstacles to be addressed. In [3] the authors highlight the safeguarding of sensitive information as one of the major challenges to cyber threat information sharing. In fact, directly sharing sensitive security logs, network information, malware samples and packet captures could expose the infrastructure of an organization and its defensive capabilities, leading to the emergence of new cyber threats. As a result, the authors suggest the anonymization and sanitization of parameters when handling and sharing sensitive information. However, as stated by Fisk et al. [4], the process of anonymization of data in a cyber threat information sharing system often reduces its utility. Furthermore, anonymization techniques can be vulnerable to attacks.

The main goal of this work was to develop *Circle*, the second component of the *PriVeil* [5] cyber threat information sharing system. *PriVeil* was designed to be a platform that allows the participants to share threat information they possess with each other, to create a cooperation environment which allows them to better deal with cyber threats. *PriVeil* relies on Homomorphic Encryption (HE) and Secure Multiparty Computation (SMC) to create a system where users can share information which can contain sensitive indicators, without having to anonymize them. The *Circle* component allows the participants whose cyber threat security reports matched in the fist component, *Square*, to share cyber threat information contained in their reports, by relying on the SMC operation of Private Set Intersection (PSI). However, at any point during *Circle*, the participants can leave the system without any penalty, which ensures that the organizations retain control over their data, only sharing the information they are willing to.

II. Background & Related Work

Cryptographic protocols are used to ensure four security goals: *confidentiality*, *data integrity*, *authentication* and *non-repudiation*. Three distinct cryptographic primitives can be used to achieve these objectives: symmetric key cryptography, which is employed in the AES algorithm [6], public key cryptography, which is used in the RSA algorithm [7] and cryptographic hash functions, which are used in the SHA-2 [8].

The Transport Layer Security Protocol (TLS) is a protocol that relies on different cipher primitives to provide a secure channel, with the properties of *authentication*, *confidentiality* and *data integrity*. Despite the different versions of TLS,

there are two sub-protocols which are common to all: the *Handshake protocol*, where the server and the client agree on the support ciphers, authenticate themselves through the exchange of digital certificates and compute the symmetric session keys used to encrypt communications; the *Record protocol*, which is responsible for securing TLS application data.

A. Secure Multiparty Computation

SMC is a cryptographic primitive whose goal is to allow a group of distrusting parties to collectively compute a function, without revealing their inputs to each other. In the end of the computation either all parties receive the output or each party receives a subset of the it. When performing this computation, we want to guarantee two conditions:

1) **Privacy of inputs:** In the end of a SMC operation, each party should not be able to learn the other parties inputs nor any intermediate calculations that can lead to the discovery of other parties inputs.
2) **Correctness of the output:** In the end of a SMC operation, the output that each of the parties receives is correct.

Performing a computation f such that the above properties are guaranteed can be defined as *computing f securely*.

When creating a SMC protocol, one has to differentiate between the two-party computation (2PC) case, where the number of parties is exactly two, and the multiparty case, where the number of parties can be equal or greater than two [9]. The implementation of a specific SMC protocol usually relies on either one of these two primitives: Garbled Circuits (GC), which is used to evaluate boolean circuits in the 2PC setting or Secret Sharing, which is generally used when computing SMC functions as arithmetic circuits.

When creating a SMC protocol we have to consider to type of adversaries that may target it. In [10], three main distinctions are made when considering the adversary model of a SMC protocol. The first aspect to consider is the ability of the adversary to deviate from the protocol. We consider two types of adversaries:

1) **Semi-honest**: Only gathers whatever information it can from the corrupted parties. The corrupted parties still follow the intended protocol.
2) **Malicious**: The corrupted parties can deviate from the protocol.

The second aspect to consider is the ability of the adversary to corrupt parties when the protocol is already being executed. We can distinguish two types of adversaries:

1) **Static**: Can arbitrarily choose which parties to control before the execution of the protocol. Throughout the protocol, the number of corrupted parties remains fixed.
2) **Adaptive**: Can choose the parties to corrupt while the protocol is being executed.

Finally, we can have two different of settings in which information is exchanged:

1) **Secure channel setting**: Despite an adversary having unlimited computing power, the point-to-point communication channels are perfectly secure, making it impossible for an adversary to obtain the parties secret inputs.
2) **Computational setting**: An adversary can capture all communications between the parties, although it cannot obtain the parties secret inputs in a feasible amount of time.

GC are a solution proposed by Yao [11] [12] to approach 2PC problems in the semi-honest adversary setting. In GC, two parties, the Garbler, Alice, and the Evaluator, Bob, want to compute the output of a function which is defined as a boolean circuit. Alice *garbles* (obfuscates) the entire circuit and sends it to Bob together with with her encrypted input. She also sends to Bob his encrypted input, by using 1-out-of-2 oblivious transfer [13]. Bob, now with both encrypted inputs and the garbled circuit, is able to consecutively compute the encrypted outputs, for each gate, until the end of the circuit, where he is able to obtain the unencrypted output. Afterwards, he communicates the output to Alice.

Secret sharing is a technique through which a secret is split into pieces which are distributed to several parties and, when combined allow the reconstruction of the secret.

Shamir's Secret Sharing (SSS) [14] is an example of a secret sharing scheme which can be applied to SMC problems due to its homomorphic property [15]: operations, like addition and multiplication, that were to be performed on the secret inputs, can instead be performed on the secrets shares, yielding the same output as if they were performed on the secret itself.

In [16], Damgård et al. propose TinyTable, a *secure* 2PC protocol where the SMC function is specified as a boolean circuit. This protocol consists of two phases: a *preprocessing* phase and an *online* phase.

In the *preprocessing* phase, the two parties, A and B, respectively get a hold of scrambled versions of truth-tables, A_i and B_i, for each gate G_i and uniform random mask bits r_o, for each output wire w_o. The uniform random mask bits, r_j, corresponding to each input wire, w_j, are given to the party who owns w_j.

In the *online* phase, the parties perform look-ups on the scrambled tables A_i and B_i, for each gate G_i, by using the bits masked by the previously chosen uniformly random bits. For each input wire, w_j, the party who owns this wire, sends $e_j = r_j \oplus b_j$ to the other party, where b_j corresponds to the actual input bit at wire w_j. For each gate G_i with input wires w_u and w_v and output wire w_o, A sends $A_i[e_u, e_v]$ to B and B sends $B_i[e_u, e_v]$ to A. Both parties can, subsequently, compute $e_o = A_i[e_u, e_v] \oplus B_i[e_u, e_v]$, for each output wire and, from this, $b_o = e_o \oplus r_o = G_i[b_u, b_v]$, where b_o corresponds to the actual bit at output wire w_o.

The final result holds because the scrambled tables A_i and B_i are set up such that $A_i[e_u, e_v]$ and $B_i[e_u, e_v]$ are an additive secret sharing of e_0. The algorithm described allows the computation of the circuit C securely assuming a semi-honest adversary.

B. Private Set Intersection

In the context of SMC, PSI is a widely researched operation. In this operation, two or more parties, $P_1, ..., P_n | n \geq 2$, with the respective private sets, $S_1, ..., S_n$ wish to compute the intersection of those same sets, as represented in equation (1):

$$f(S_1, ..., S_n) = S_1 \cap ... \cap S_n. \quad (1)$$

In the end of this computation, either all parties or a subset of them learn the elements which are common to all sets, without anything being disclosed about the different ones. According to [17], implementations of PSI can be divided into two classes: generic protocols, which specify this operation as a circuit and rely on a general SMC techniques to solve it, and custom protocols, which are created for the specific structure of the PSI operation.

C. SMC Frameworks

We analyzed three promising SMC frameworks that implement PSI: FRESCO, ABY and Swanky.

FRESCO provides a PSI demonstration (demo) based on the TinyTable protocol. In this demo, the parties use the TinyTable protocol to perform the XOR of their input secret AES-128 keys and afterwards use it to compute the AES-128 encryption of their inputs. In the end of the computation, both parties obtain the AES-128 encryptions of a concatenated list of their inputs. In this list, the first half of the elements corresponds to the encryption of the input set of the first party and the second half corresponds to the encryption of the set of the second party. The parties are, subsequently, responsible for identifying the intersecting elements, by looking which encryption of elements in the first half of the concatenated list match the encryption of elements in the second half. Despite this demo being called PSI, it does not correspond to the definition of the PSI operation provided in section II-B, since the elements in the intersection are not a direct output of the 2PC computation.

The ABY framework relies on the *Sort-Compare-Shuffle* circuit [18], a generic protocol based on GC, to provide an implementation of PSI.

Swanky is an open-source suite of libraries for SMC operations. The *popsicle* library provides implementations of three different custom PSI protocols: the 2PC protocol based on OPRF [19] described in [20], the 2PC protocol based on OPPRF presented in [17] and the multiparty protocol described in [21].

III. PriVeil Circle

In *PriVeil*, the information which is shared to allow cooperation when dealing with cyber threats is contained in security reports. The security reports are generated by organizations when they identify a cyber threat and contain information about the threat: a small description of it, its severity, the data when it was detected and tags which contain keywords associated with the threat.

PriVeil is structured in two phases, supported by two components: *Square* and *Circle*. Figure 1 represents the architecture of *PriVeil*.

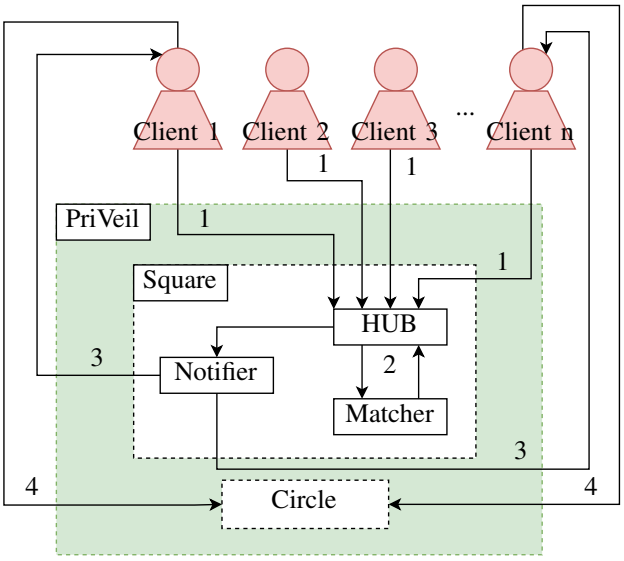

Fig. 1: *PriVeil* Architecture, adapted from Gonçalves [5].

Users start by submitting their encrypted security reports describing a cyber threat to the *Square* component (step 1 in figure 1). The system, subsequently, matches the encrypted security reports that describe similar cyber threats (2) and notifies the users whose reports matched (3). Afterwards, these participants receive a token which authorizes them to engage in a *Circle* session and are redirected to it (4).

A. Requirements

In order to develop *Circle*, we determined that we required two types of parties: the *Dealer* and the *Players*. The *Dealer* is the coordinator of a *Circle* session and acts as a communications facilitator, asking the *Players* for necessary information as the session advances. Although it puts the *Players* in touch with each other to share threat data, the *Dealer* does not have access to the cyber threat information that is being shared. The *Players* represent organizations willing to share cyber threat information. They were forwarded to *Circle* by *Square* and act as the clients of the *Circle* session. They send to the *Dealer* the information it requires and communicate with other *Players* in specific communication rounds, to share cyber threat data.

We defined the following functional requirements for *Circle*:

- FR_1: *Players* can at any time leave the *Circle* session they are participating in.
- FR_2: Only parties that received a token from *Square* to participate in the *Circle* session are authorized to participate in it.
- FR_3: Allow *Players* to share cyber threat data contained in their security reports, in a privacy-preserving manner.

Due to the fact that our system is a part of a cyber threat information sharing platform, we need to prevent leakage of data, as well as an unauthorized access to it. To fulfill these goals we had to define the attackers:

1) **Eavesdropper**: Passively listens to all communications.
2) ***Player* Impersonator**: Impersonates a *Player*.

3) ***Dealer* Impersonator**: Impersonates the *Dealer*.
4) **Tamperer**: Modifies the data exchanged between source and destination.

We had to consider two distinct types of communication that the attackers could target:

- C_{PD}: Bidirectional *Player* to *Dealer* communication.
- C_{PP}: Bidirectional *Player* to *Player* communication.

This distinction is necessary because the attackers involved are different for these two types of communication. In the case of C_{PD}, attackers of all types can target the system, whereas, when we consider a communication of type C_{PP}, the *Dealer* is not involved in it, and, as a result, a *Dealer* Impersonator is not relevant in this situation. We defined two security requirements for *Circle* based on the two types of communications and the attackers that could target them:

- SR_1: An attacker with eavesdropping, tampering and *Player* and *Dealer* impersonating capabilities is unsuccessful when targeting a communication of type C_{PD}.
- SR_2: An attacker with eavesdropping, tampering and *Player* impersonating capabilities is unsuccessful when targeting a communication of type C_{PP}.

B. Circle *Prototype*

In this work we wanted to allow *Players* to share the tags contained in their reports in a privacy-preserving way. During the development of *Circle*, we used a dataset consisting of images of real cyber threat security reports from the monitoring systems of NAV[1]. However, before using these reports, we first had to remove the personal data contained in them, to avoid leaking any personal information.

The solution we found to guarantee a cyber threat sharing environment between parties who do not trust each other, yet wish to share data that might contain sensitive parameters, was to resort to the PSI operation. By relying on this SMC operation, we designed *Circle* to allow its participants to compute PSI on the sets of tags in their security reports. In the end of this computation, the tags which are equal in the security reports are revealed, without any information being leaked about the ones that are different.

Our system is meant to have more than two participants, so we could have relied on a multiparty PSI solution. However, the output of this solution would be the intersection of the sets of all parties. This is not our objective for *Circle*, considering that, as an example, two parties may contain several tags in the intersection of their sets, which are not equal to those in the remaining *Players* reports. These two parties should still be able to know that they have equal tags in their security reports and, because of that, they might be affected by a similar cyber threat. As a result, we relied on a 2PC PSI approach, that allows each *Player* to compute PSI with all the remaining *Players* in *Circle*, over the course of a determined number of one-on-one PSI computations.

[1]NAV is the portuguese company which is responsible for monitoring the air traffic control in Portugal. The webpage of NAV can be visited at https://www.nav.pt/en/nav-portugal-newhp_en.

The final goal of performing the PSI operation between all *Players* in a *Circle* session is that, once this phase is terminated, each *Player* can make a better informed decision whether it wants to share more information about the cyber threat described in its security report.

In *Circle*, initially, the *Dealer* listens for *Players* looking to connect to the session, through the `connectionEstablishment` function call. During a certain timeframe, *Players* that were authorized by *Square* can join *Circle* by communicating with the *Dealer*. The end of this timeframe is marked by the value of `joincircletimeout`, while the minimum number of *Players* required for a *Circle* session to start is the `quorum`. When the `joincircletimeout` occurs and less *Players* than the `quorum` have connected, the *Circle* session is cancelled, the *Dealer* informs the *Players* and then terminates. Otherwise, if the `quorum` is achieved, the *Circle* session can start, since the minimum number of required *Players* to engage in this cyber threat information sharing system was assembled.

For the next step in the session, we defined an abstract event for each *Player*, the *Computation round*, which occurs several times in the same *Circle* session. When one of its *Computation rounds* starts, each *Player* has the option to leave *Circle*, giving the *Player* full control over what security reports data is shared. If they wish to continue in *Circle*, they make a call to the `psiRound` function of the *Dealer* and receive the identifier (id) and address of a *Player*, with whom they are to perform the two-party PSI operation with, in that *Computation round*. Nevertheless, a *Player* may not have a partner to compute PSI with in certain *Computation rounds*. This situation arises if all other *Players* that have not yet performed the PSI operation with it have already been assigned as PSI partners to other *Players* in this *Computation round*, or if this *Player* has already executed the PSI computation with all other *Players*. In this case, the *Dealer* sends it a message indicating that it will not have a partner for this round.

Each *Player* repeats this *Computation round* process until it decides to leave the *Circle* session, the *Dealer* cancels the session because there are less than two remaining *Players* connected or *Circle* is concluded, because there are no more PSI operations to be performed.

C. Technical Architecture

The architecture of the *Circle* entities, the *Dealer* and the *Player* is represented side-by-side in figure 2.

The development of both *Circle* parties was done with Java, because it is a mature programming language, it has an extensive online documentation and it has a large collection of libraries and modules available.

To describe the *Circle* session configuration file and the *Players* security reports we used Extensible Markup Language (XML). We chose this markup language because it provides a format that is both human-readable, to allow its easy edition by humans, and machine-readable, to allow its easier processing by machines.

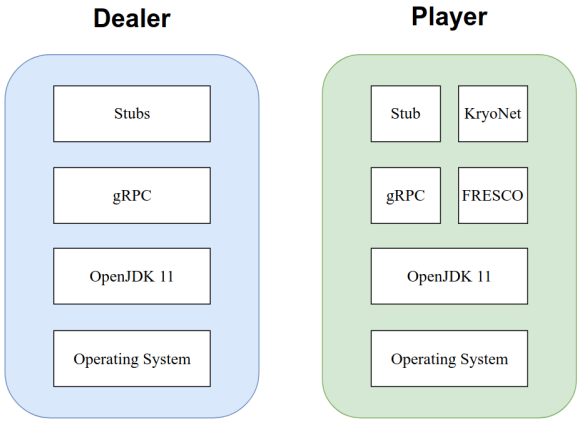

Fig. 2: *Dealer* and *Player* architectures.

To implement the functions `connectionEstablishment` and `psiRound`, called by the *Players* when communicating the *Dealer*, we relied on gRPC, a Remote Procedure Call (RPC) framework which also has support for Java, it supports synchronous and asynchronous calls and it is open-source and free to use. We used protocol buffers (Protobuf) as a serializing mechanism for these communications because it is the default serializing mechanism for gRPC, it is also open-source and provides a very efficient binary encoding format. Protobuf requires defining the structure of the data to be serialized in a *proto* file, which corresponds to a message definition language expressed in a text file with a .proto extension. In our work, we defined the RPC functions and messages in a *Circle.proto* file.

We required a SMC framework that provided a *secure* implementation of PSI. As a result, we chose FRESCO from the frameworks defined in subsection II-C, because it is a well-known framework that has been used in real-life projects, it has an extensive documentation and has great interoperability. The PSI demo provided by FRESCO relies on the TinyTable protocol to provide to both parties, in the end of the operation, the AES-128 encryption of both input sets. The two parties are, subsequently, responsible for computing the intersecting elements themselves, by comparing both input sets encryptions and finding those that are equal. This computation is still *secure* since the parties cannot decrypt the AES-128 encryptions of the output because they do not possess the secret key. To allow communication between parties during computations, FRESCO uses KryoNet as its default network communication supplier. KryoNet is a Java library that provides an API for both TCP and UDP network communication.

D. Dealer *Implementation*

The *Dealer* takes as input to its program a XML configuration file, *configuration.xml*, which is constructed with the results of a security reports match that occurred in *PriVeil Square*. An example of configuration file that was used in a *Circle* session is shown in figure 3.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <quorum>3</quorum>
    <timeouts>
        <joincircletimeout>120</joincircletimeout>
        <psiroundtimeout>300</psiroundtimeout>
    </timeouts>
    <maxnumberofplayers>3</maxnumberofplayers>
    <address>
        <ip>127.0.0.1</ip>
        <port>9999</port>
    </address>
</configuration>
```

Fig. 3: Example of *Dealer configuration.xml* file.

In *configuration.xml*, the `quorum` represents the minimum number of *Players* required to start the *Circle* session, otherwise it will not occur. The `joincircletimeout` and `psiroundtimeout` are the values of the timeouts for the *Players* to join a *Circle* session and answer to the *Dealer* after making the `psiRound` RPC call, respectively. The `maxnumberofplayers` represents the maximum number of *Players* allowed in the *Circle* session. At last, the values of the `ip` and `port` correspond to the IP address and port where the *Dealer* will be running the remote functions `connectionEstablishment` and `psiRound`.

To avoid the large time execution penalty a *Circle* session would incur in, if only a pair of *Players* was computing PSI at each *Computation round*, we created an algorithm to parallelize PSI operations between different pairs of *Players*. When each thread serving a *Player* call to `psiRound` enters the `getOtherPlayerId` function, it searches a possible partner for PSI from a linked list that contains all the pairs of *Players* that have not yet computed PSI in the *Circle* session. The only restriction when choosing a partner is that it has not been already assigned as PSI partner to another *Player* in this *Computation round*. The thread serving the given *Player* chooses as PSI partner the first *Player* that complies with this restriction. Nevertheless, if there is no other *Player* that complies with this restriction, this *Player* will not have a PSI partner for this *Computation round*. After each *Computation round* and *Players* make a new call to `psiRound`. The thread serving each *Player* then reaches the `hasPsiResult` function, which is responsible for making the necessary preparations for the `getOtherPlayerId` method to be able to assign new pairs of PSI partners for this new *Computation round*.

In *Circle*, we resorted to the TLS protocol to secure the communications between the *Dealer* and the *Players*. We used a model of *mutual authentication*, in which both parties need to authenticate themselves to the other party, thus proving their identity. Since our project is a prototype, we created our own Certification Authority (CA) and used it to sign both the digital certificates of both the *Players* and *Dealer*. However, the *PriVeil* system can operate with its own CA or an external one, so, when we designed *Circle* as the sequence of *Square*, we concluded that these certificates could have been delivered to the *Dealer* and the *Players* by the system.

E. Player *Implementation*

The *Players* take as input to their programs the IP address and port they will use to communicate with other *Players* during the PSI operation, their security reports in XML format, the IP address and port of the *Dealer* and the AES-128 secret key they will use in the 2PC PSI computation implemented by FRESCO. When developing our program we used images of ten security reports from NAV for testing, which we had to convert to XML, before we could use them.

The implementation of the 2PC PSI operation by FRESCO only accepts sets of `Integers` as the parties inputs for the intersection operation. Since we required to compute this operation between sets of tags, which contain text values and hence are represented as `Strings`, we had to find a solution which allowed us to make a conversion between the `String` tags to `Integers` values. To this end, we computed the hash function SHA-256 on the byte representation of the text value of each tag, obtaining a 32 bytes output. Afterwards, since each `Integer` in Java occupies four bytes of space, we truncated the SHA-256 hash value of each tag to its first four bytes and used them to construct an `Integer` value. This solution allowed us to be able to compute the two-party PSI operation between sets of text tags, while still using FRESCO.

Another restriction of the PSI operation implemented by FRESCO is that the input sets of both parties are required to have the same size. This is a problem because cyber threat security reports, which belong to different organizations, are very unlikely to contain the the exact same number of tags. As a consequence, we considered that, for each *Circle* session, we would need to establish a maximum number of tags allowed in *Players* security reports. When performing 2PC PSI, the first solution we found to address this restriction was for each *Player* to provide the actual tags contained in its security report together with empty tags, until the maximum number of security tags allowed was reached. This solution was not ideal since, when both parties received the output of the 2PC PSI, they would be able to identify which was the AES-128 encryption of the empty tags and, consequently, would be able to know what was the number of actual tags in the security report of the other *Player* . A better way to address this restriction of the implementation of PSI provided by FRESCO is for each *Player* to provide the actual tags contained in its security report, together with random tags, until the maximum number of security tags allowed is reached. When *Players* receive the output of the 2PC PSI computation they are not able to distinguish the random tags from the actual tags in the security report of the other *Player*. In *Circle*, the latter solution was implemented over the previous one, although the evaluation and results of section IV were obtained for the empty tags implementation. This change has no significant impact on performance

IV. Evaluation

A. Qualitative Evaluation

In table I we can observe the functional and security requirements from section III-A that were addressed by our *Circle* prototype.

TABLE I: Assessment of the requirements fulfilled

Requirement	Status
FR_1	Fulfilled
FR_2	Fulfilled
FR_3	Fulfilled
SR_1	Fulfilled
SR_2	Not fulfilled

The functional requirement FR_1 is fulfilled by the fact that the *Players* can choose whether to remain in *Circle* at the beginning of each one of their *Computation rounds*. Furthermore, the *Players* can also leave the *Circle* session at any time, without having to notify the *Dealer*. The functional requirement FR_2 is fulfilled due to the *authentication* mechanisms provided by the TLS protocol: if either the *Dealer* or the *Players* detect that they are communicating with a party which is not authorized to participate in the session, they can leave it. The functional requirement FR_3 is addressed by the choice of the PSI operation to allow *Players* to share cyber threat information. This operation allows a privacy-preserving computation to be performed on the sets of tags in the *Players* security reports. To fulfill SR_1, we used the TLS Protocol to prevent all attackers defined in subsection III-A, in communications of type C_{PD}. In this work, the security requirement SR_2 was not addressed. FRESCO does not natively use TLS to secure communications in its implementation of 2PC PSI and we were also not able to implement it.

B. Experimental Design

To evaluate our system we performed a total of nine experiments, for nine *Circle* sessions. In each session, we measured: the processing time of the most relevant functions executed by a *Player*, to assure that our system provided answers in a timely way; the system CPU and memory usage of the program of the same *Player*, to show that the required resources to participate in *Circle* are reasonable in commodity hardware. Between distinct experiments, we examined the impact on the aforementioned metrics of different combinations of two parameters: the number of *Players* participating in the *Circle* session and the maximum number of tags allowed in the *Players* cyber threat security reports.

In our experiments, we used a setup consisting of two machines: the first machine was running Ubuntu 20.04.1 LTS OS, with an Hexa Core Intel i5-8400 processor, 16 GB RAM, 1 TB Hard Disk Drive (7200 rpm), 128 GB Solid State Drive and an internet cable connection with bandwidth of 500 Mbit/s; the second machine was running Ubuntu 20.04.01 LTS OS, with an Octa Core Intel i7-8565U, 16 GB RAM, 500 GB Hard Disk Drive (7200 rpm), 128 GB Solid State Drive and an internet cable connection with bandwidth of 500 Mbit/s.

The measurement of the processing time of the most relevant functions executed by a *Player* was performed with the

help of the Java method `System.nanoTime()`. The code below was used to measure how long a specific part of code takes to execute:

```
long startTime = System.nanoTime();
\\ Code being evaluated
long endTime = System.nanoTime();
long executionTime = endTime-startTime;
```

We applied the time measurement code above to the time checkpoints specified in table II.

TABLE II: Description of time checkpoints introduced in the code of the *Player*.

Total measured time	Time measured from the beginning of the program of the *Player* until its successful termination.
Parse report	Time taken for the program of the *Player* to parse the input security report.
***Circle* connection**	Time it takes for the *Player* to send its address to the *Dealer* and the response of the *Dealer* indicating if *Circle* will occur or will be cancelled.
Computation round	Time duration of one of the *Computation rounds* of the *Player*. This measurement can occur several times in the same session.
PSI operation	Time duration of the PSI operation during the *Computation round*.
Other	All the time that is not included in the first three checkpoints.

We created the Python script `setup.py` to measure the system CPU and memory usage of the program of a *Player*. When running the `setup.py` script to benchmark a *Circle* session, a single measurement of the system CPU usage and system memory usage of process with identifier PID was done by using the following command:

```
top -b -n 1 -p <pid> | tail -n 1 | awk {
'print $9 \"\t\" $10'}
```

To benchmark the whole program of the *Player*, the `setup.py` script was running periodically while the process of the *Player* was running.

In all of these *Circle* sessions we used as input to the program of this *Player* the same XML report corresponding to a real-case cyber threat security report from NAV, with five tags contained in it.

In the first set of three experiments, the number of *Players* was fixed to 2 and the maximum number of tags was increasingly set to 10, 25 and 50. In these group of tests, the number of *Players* was equal to 2, because this corresponds to the minimum number of *Players* required for a 2PC PSI operation to be performed in a *Circle* session. Furthermore, this situation also corresponds to the simplest setting possible for Circle, since a session cannot occur if the number of participants is lesser than 2. For the first experiment we set the maximum number of tags allowed in *Players* security reports to 10, since in all the ten reports we obtained from NAV, the maximum number of tags we observed on a report was 5. Furthermore, in the work that addresses the *Square* component of *PriVeil*, it is estimated that the security reports have on average 10 tags contained in them. As a result, the value of 10 seemed like a reasonable value for the lower bound for the maximum number of tags allowed in the *Players* reports. On the other hand, to try to include all the cases of security reports that have a number of tags superior to 10, we established the value of 50 tags as the upper bound to the maximum number of tags allowed in security reports. Finally, we concluded that only performing tests for these two values of this parameter would be insufficient, so, we also performed experiments for the intermediate value of 25 tags.

In the next set of three experiments, the number of *Players* was set to 3 and the maximum number of tags allowed in their reports was also increasingly set to 10, 25 and 50. In this group of tests, we fixed the number of *Players* to 3, so that we could observe how the measured metrics would be affected when the *Circle* session involved more than one 2PC PSI operation.

Finally, in the last group of experiments, the number of *Players* was fixed to 10, while the maximum number of tags was also increasingly set to 10, 25 and 50. In this final set of experiments, we fixed the number of *Players* to 10, since this is the average number of *Players* we expect to have in each *Circle* session, if we were to employ this project in a real-life cyber threat information sharing scenario. Furthermore, this value seemed reasonable considering that, in *Circle*, we wanted to allow threat information sharing between a more restricted group of organizations, when compared to the type of sharing that occurs in *Square*.

C. Experimental Results

In the 2 *Players* setting, one *Player* was running on the first machine in the testbed, where the metrics were measured, and the *Dealer* and the other *Player* were running on second machine. In this set of experiments, the total execution time measured for the program of the *Player*, for a maximum of 10, 25 and 50 tags allowed in the security reports was 51,195 s, 96,605 s and 179,07 s, respectively. The results are presented in figure 4.

We can conclude that the time measured on checkpoints *Circle connection* and *Other* does not depend on the maximum number of tags allowed in *Players* security reports. This is expectable since the functions measured by these checkpoints include communications with the *Dealer*, interacting with local files and local processing. Although the *Parse report* checkpoint is dependent on the number of tags in the security report of the *Player*, it does not depend on the maximum number of tags allowed. The second *Computation round* also has similar durations as the number of tags increases. This is explained by the fact that in the second *Computation round* there are no PSI operations left to be computed, as the only PSI operation involved in this *Circle* session was already performed in the first *Computation round*. In the second *Computation round*, the *Dealer* only informs the *Players*

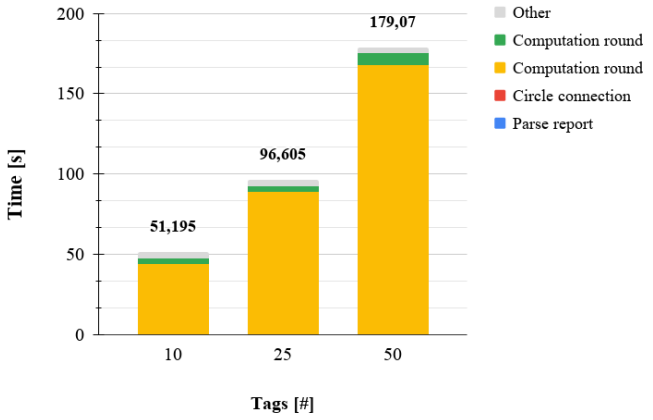

Fig. 4: Time breakdown as a function of the maximum number of tags allowed in the security reports for 2 *Players*.

that there are no more computations left and, consequently, the session will be terminated. The first *Computation round* is the checkpoint for which the measured execution time significantly depends on the maximum number of tags allowed in *Players* security reports, since it includes a PSI computation. In this round, we obtained results of 41,151 s, 86,405 s and 165,15 s in the *PSI operation* checkpoint, for 10, 25 and 50 tags, respectively. This indicates that the increase in the maximum number of tags allowed in *Players* security reports directly affects the time spent in the PSI operation. This is in fact true, because in the context of PSI, the more items in the sets of the parties, the more time it takes to compute this operation between those sets.

In the 3 *Players* experiments, one *Player* was running on the first machine of the testbed, while the remaining 2 *Players* and the *Dealer* were running on the second machine of the testbed. The time breakdown for these three experiments is represented in figure 5.

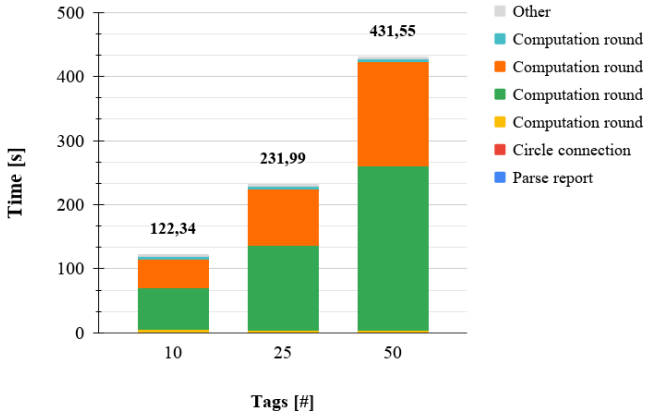

Fig. 5: Time breakdown as a function of the maximum number of tags allowed in the security reports for 3 *Players*.

From the results of figure 5 we can see that there are four *Computation rounds* involved in this *Circle* session. The first and last *Computation rounds*, the *Parse report*, *Circle connection* and *Other* checkpoints measurements do not depend on the maximum number of tags allowed in *Players* security reports because their values are similar in all these three experiments.

The first *Computation round* is very short when compared to the following two *Computation rounds*, because the *Player* where the measurements were being made did not have a partner to perform the PSI operation with in this round. The last *Computation round* corresponds to the same situation of the last *Computation round* of the 2 *Players* experiments. Finally, in the second and third *Computation rounds*, the *Player* where the measurements were made was doing PSI with the other 2 *Players* in the session. To provide a more detailed insight on the PSI operations of these rounds, in figure 6 we can see the duration of the two PSI computations.

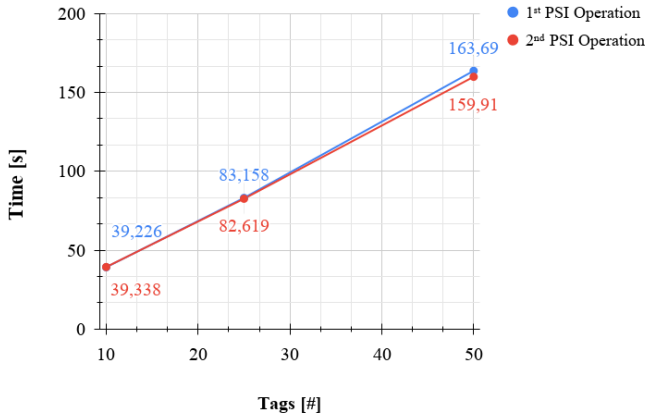

Fig. 6: PSI operation duration as a function of the maximum number of tags allowed in the security reports for 3 *Players*.

From the observation of figure 6 we can conclude that the PSI operation duration increases linearly with an increase in the maximum number of tags allowed in *Players* security reports.

In the 10 *Players* experiments, five *Players* were being executed on the first machine in the testbed while the remaining five *Players* and the *Dealer* were running on the second machine in the testbed. The time breakdown for the 10 *Players* experiments is represented in figure 7. From the results in figure 7 we can notice some differences on the time breakdowns for each maximum number of tags allowed in *Players* security reports. The same *Computation round* can have very different durations for a different number of tags. Since the order on which the PSI pairs are chosen depends on the algorithm described in subsection III-D, this can lead to a different order in which the PSI operations are computed, between different *Circle* sessions.

In figure 8 we represented the total execution time of the program of the *Player* for a varying number of *Players*. From

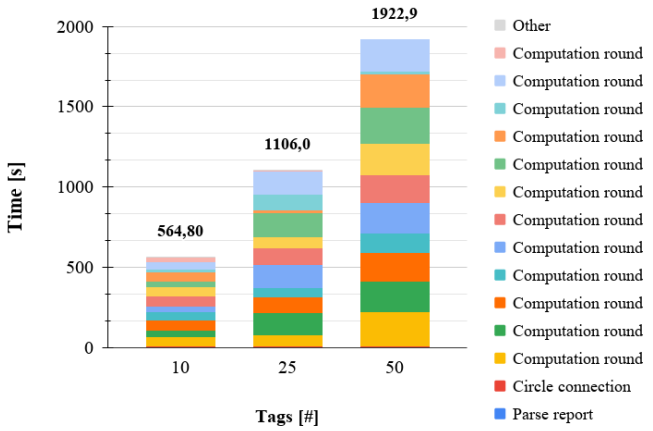

Fig. 7: Time breakdown as a function of the maximum number of tags allowed in the security reports for 10 *Players*.

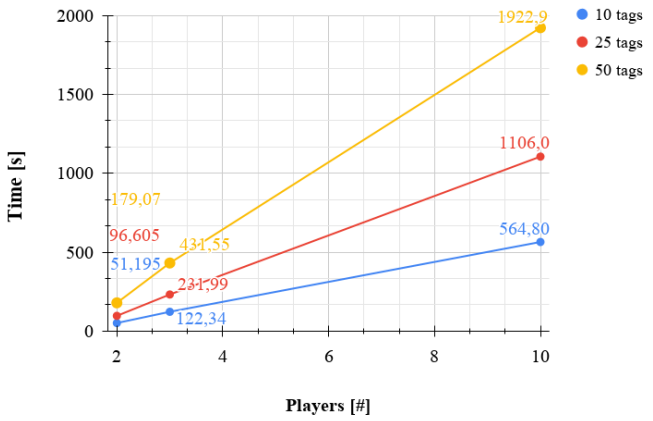

Fig. 8: Total time duration of the program of the *Player* as a function of the number of *Players* in the session.

the observation of figure 8 we can conclude that for a fixed number of tags, the total execution time of the program of the *Player* scales linearly with the number of *Players*, with the highest value being obtained for the experiment with 10 *Players* and 50 maximum tags allowed in the reports.

In figure 9 we represent the average system CPU usage as a function of the number of *Players* for 10, 25 and 50 maximum tags allowed in *Players* security reports. From the results of figure 9 we conclude that, in general, the average CPU usage decreases as a function of the number of the *Players* in the *Circle* session. This can be explained by the fact that although the number of PSI operations increase with the increase in the number of *Players*, there is more idle time, when the *Player* is not doing the PSI operation with any other *Player*, which significantly lowers the CPU usage during those periods of time and, consequently, the average CPU usage of the program. For the average CPU usage as a function of the maximum number of tags allowed in *Players* reports we cannot draw any significant conclusions.

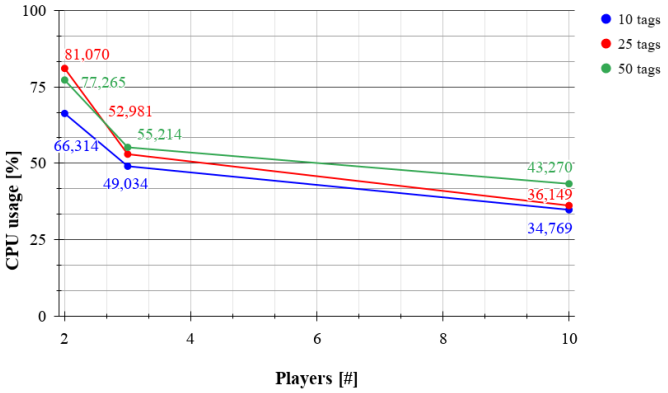

Fig. 9: Average CPU usage as a function of the number of *Players* in the session.

In figure 10 we represent the average system memory usage as a function of the number of *Players* for 10, 25 and 50 maximum tags allowed in *Players* security reports. With respect to the average memory usage, we can observe

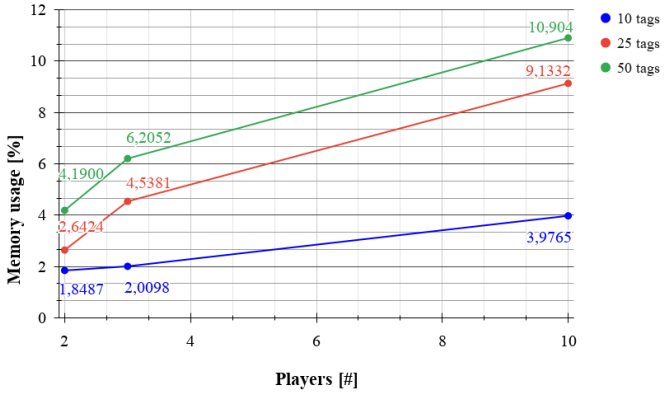

Fig. 10: Average memory usage as a function of the number of *Players* in the session.

that it increases as a function of the number of *Players* in the *Circle* session, being the highest average memory usage 10,904 %, for the 10 *Players* with 50 tags experiment, and the lowest 1,8487 %,for the 2 *Players* with 10 tags experiment. The average memory usage also increases as a function of the maximum number of tags allowed in *Players* security reports.

D. Discussion

The experiments performed and respective results show that *Circle* is applicable in a real-case scenario. For the longest and most resource-consuming experiment, with 10 *Players* and a maximum of 50 tags allowed in the security reports, we obtained a total execution time of 32,05 minutes. Despite this result being high for a cyber threat information sharing system, we conclude that the results are still acceptable, since we perform a *secure* computation that preserves the *Privacy* of the parties inputs. Regarding the CPU usage of the program

of a *Player*, the results obtained show that the program of a *Player* to be easily executed in commodity hardware. Finally, we have observed that the average memory usage increases as a function of both the maximum number of tags allowed in *Players* reports and the number of *Players* in the session. Nevertheless, in all experiments, these results did not have a noticeable impact on the system of the *Player*.

V. Conclusion

In this work we designed and implemented a prototype of *Circle*, the second component of the cyber threat information sharing system *PriVeil*. In *Circle*, each *Player* progressively computes the PSI operation on the tags on his security report with every other *Player*. By the end of this round all users know the tags in their security reports which are common to other users security reports, without any information being revealed about the different tags. This information is useful because each *Player* will know the *Players* that might contain the same or a similar cyber threat described in the security report. We evaluated *Circle* by performing nine different experiments in which the processing time, CPU and memory usage of a *Player* program were measured during a *Circle* session. The results show that this prototype can be used in real-life scenarios, since it creates an environment where participants can *securely* share the cyber threat information contained in their security reports.

As future work, our prototype needs improvements in the PSI partners assignment algorithm, TLS needs to be implemented on top of FRESCO communications during the PSI operation, *Circle* needs to be deployed and tested in a real-life situation and *Square* and *Circle* need to be integrated as a single system.

References

[1] L. Cheng, F. Liu, and D. Yao, "Enterprise data breach: causes, challenges, prevention, and future directions," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, 2017.

[2] R. van Wegberg, S. Tajalizadehkhoob, K. Soska, U. Akyazi, C. Gañán, B. Klievink, N. Christin, and M. van Eeten, "Plug and Prey? Measuring the Commoditization of Cybercrime via Online Anonymous Markets," in *Proceedings of the 27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 2018, pp. 1009–1026.

[3] C. S. Johnson, M. L. Badger, D. A. Waltermire, J. Snyder, and C. Skorupka, "Guide to Cyber Threat Information Sharing," National Institute of Standards and Technology, Special Publication 800-150, 2016.

[4] G. Fisk, C. Ardi, N. Pickett, J. Heidemann, M. Fisk, and C. Papadopoulos, "Privacy Principles for Sharing Cyber Security Data," in *Proceedings of the IEEE Symposium of Security and Privacy Workshops (SPW)*. IEEE, 2015, pp. 193–197, dOI:10.1109/SPW.2015.23.

[5] T. Gonçalves, "Priveil: Privacy-preserving threat information sharing," Master's thesis, Instituto Superior Técnico, December 2019.

[6] J. Daemen and V. Rijmen, *The Design of Rijndael*, 5^{th} ed. Springer, 2002, iSBN:978-3540425809.

[7] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978, dOI:10.1145/359340.359342.

[8] NIST, "Secure hash standard (SHS)," National Institute of Standards and Technology, FIPS Publication 180-2, August 2002.

[9] D. Evans, V. Kolesnikov, and M. Rosulek, "A Pragmatic Introduction to Secure Multi-Party Computation," *Foundations and Trends in Privacy and Security*, vol. 2, 2018, dOI:10.1561/3300000019.

[10] R. Canetti, "Security and Composition of Multiparty Cryptographic Protocols," *Journal of Cryptology*, vol. 13, pp. 143–202, 2000.

[11] A. C. Yao, "How to generate and exchange secrets," in *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. IEEE, 1986, pp. 162–167, dOI:10.1109/SFCS.1986.25.

[12] ——, "Protocols for Secure Computations," in *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. IEEE, 1982, pp. 160–164, dOI:10.1109/SFCS.1982.38.

[13] S. Even, O. Goldreich, and A. Lempel, "A Randomized Protocol for Signing Contracts," *Communications of the ACM*, vol. 28, no. 6, pp. 205–210, 1982, dOI:10.1145/3812.3818.

[14] A. Shamir, "How to Share a Secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979, dOI:10.1145/359168.359176.

[15] J. C. Benaloh, "Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret," in *Proceedings of Advances in Cryptology (CRYPTO '86)*, ser. Lecture Notes in Computer Science, vol. 263. Springer, Berlin, Heidelberg, 1987, pp. 251–260.

[16] I. Damgård, J. B. Nielsen, M. Nielsen, and S. Ranellucci, "The Tinytable Protocol for 2-Party Secure Computation, or: Gate-Scrambling Revisited," in *Proceedings of Advances in Cryptology (CRYPTO 2017)*, ser. Lecture Notes in Computer Science, vol. 10401. Springer, Cham, 2017, pp. 167–187.

[17] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai, "Efficient Circuit-Based PSI with Linear Communication," in *Proceedings of Advances in Cryptology (EUROCRYPT 2019)*, ser. Lecture Notes in Computer Science, vol. 11478. Springer, Cham, 2019, pp. 122–153.

[18] Y. Huang, D. Evans, and J. Katz, "Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?" in *Proceedings of the 19th Network and Distributed Security Symposium (NDSS 2012)*, 2012.

[19] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold, "Keyword Search and Oblivious Pseudorandom Functions," in *Proceedings of Theory of Cryptography (TCC 2005)*, ser. Lecture Notes in Computer Science, vol. 3378. Springer, Berlin, Heidelberg, 2005, pp. 303–324.

[20] B. Pinkas, T. Schneider, and M. Zohner, "Scalable Private Set Intersection Based on OT Extension," *ACM Transactions on Privacy and Security*, vol. 21, no. 2, 2018, dOI:10.1145/3154794.

[21] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu, "Practical Multi-party Private Set Intersection from Symmetric-Key Techniques," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. Association for Computing Machinery, 2017, pp. 1257–1272, dOI:10.1145/3133956.3134065.