



Tecnologias de Reforço da Privacidade

---

# Secure Multiparty Computation for Privacy in Practice

---

Maria Sousa Carreira up202408787  
Matilde Isabel da Silva Simões up202108782

Abril 2025

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Descrição e Comparação dos 4 Protocolos PSI</b>	<b>2</b>
2.1	Protocolo <i>Naive Hashing</i> . . . . .	2
2.2	Protocolo <i>Server-Aided</i> . . . . .	3
2.3	Protocolo baseado em <i>Diffie-Hellman</i> . . . . .	3
2.4	Protocolo baseado em <i>Oblivious Transfer</i> . . . . .	4
2.5	Comparação dos Protocolos de PSI . . . . .	5
<b>3</b>	<b>Teste dos Protocolos PSI</b>	<b>6</b>
3.1	Protocolo <i>Naive Hashing</i> . . . . .	6
3.1.1	Vulnerabilidades - Ataque de Dicionário . . . . .	7
3.1.2	Execução para <code>AppList1.csv</code> e <code>AppList2.csv</code> . . . . .	8
3.2	Protocolo <i>Server-Aided</i> . . . . .	8
3.2.1	Vulnerabilidades . . . . .	9
3.3	Protocolo baseado em <i>Diffie-Hellman</i> . . . . .	9
3.4	Protocolo baseado em OT . . . . .	10
3.5	Segurança <i>vs.</i> Qualidade da Comunicação . . . . .	10
<b>4</b>	<b><i>Benchmarking</i> dos protocolos PSI</b>	<b>11</b>
<b>5</b>	<b>Aplicação dos Protocolos a um <i>Dataset</i> Escolhido</b>	<b>13</b>
<b>6</b>	<b>Conclusão</b>	<b>14</b>

# 1 Introdução

A interseção privada de conjuntos (*Private Set Intersection* – PSI) é uma técnica criptográfica que permite a diferentes partes identificar elementos em comum entre os seus conjuntos de dados sem revelar os restantes elementos. Este tipo de protocolo é particularmente relevante em cenários onde a privacidade dos dados é essencial, como na comparação de listas de clientes, contactos pessoais, etc. .

O presente trabalho tem como objetivo estudar e comparar diferentes protocolos de PSI do ponto de vista da segurança, desempenho e eficiência no contexto da comunicação. Para isso, foram analisados quatro protocolos distintos: *Naive Hashing*, *Server-Aided*, baseado em *Diffie-Hellman* e baseado em *Oblivious Transfer*. Cada protocolo apresenta características próprias em termos de robustez contra adversários e custos computacionais ou de comunicação.

A análise foi conduzida em várias fases: primeiro, foram estudados os fundamentos teóricos de cada protocolo; de seguida, foi realizada uma série de experiências práticas com o auxílio da ferramenta *Wireshark*, permitindo observar os dados transmitidos em cada caso. Posteriormente, foi realizado um processo de *benchmarking* para comparar o desempenho dos protocolos com diferentes volumes de dados. Por fim, aplicaram-se os protocolos a um conjunto alternativo de dados, neste caso listas de músicas gostadas por diferentes utilizadores, avaliando a sua adequação prática ao contexto.

Este relatório apresenta os resultados obtidos, discutindo as vantagens e limitações de cada protocolo, bem como a escolha do mais apropriado para diferentes contextos de aplicação.

## 2 Descrição e Comparação dos 4 Protocolos PSI

Para iniciar o estudo e compreender o funcionamento prático do *Private Set Intersection* (PSI), foi executado o teste disponibilizado para os 4 protocolos, sendo apenas necessário alterar a opção `-p`. Os resultados obtidos foram os esperados em todos os casos. Foram interceptados os três elementos: `Michael.Zohner@ec-spride.de`, `Evelyne.Wagener@tvcablenet.be` e `Ivonne.Pfisterer@mail.ru`.

### 2.1 Protocolo *Naive Hashing*

Este protocolo baseado em *hashing* consiste num método simples e utilizado na prática.

Cada parte envolvida aplica uma função de *hash* criptográfica aos elementos do seu conjunto, ao qual apenas tem acesso. Os valores resultantes são posteriormente comparados com os recebidos, de forma a identificar os elementos comuns [1], ou seja, o resultado da interseção.

Este protocolo é bastante eficiente e rápido a executar [1], o que se justifica pela sua natureza simples. No entanto, a sua segurança depende da entropia dos dados e da dimensão do domínio de entrada [1]. Quando os elementos do conjunto pertencem a domínios pequenos ou com baixa entropia – ou seja, elementos comuns e previsíveis –, um atacante pode realizar ataques de *brute force* e/ou dicionário, ao calcular os *hashes* de todos os valores possíveis para identificar correspondências. Isto deve-se a facto do protocolo ser determinístico, pois permite que o mesmo elemento seja sempre mapeado para o mesmo *hash*. Por outro lado, se os elementos forem imprevisíveis – com alta entropia – e fizerem parte de um domínio suficientemente grande, este tipo de ataques tornam-se impraticáveis e a comparação direta entre as *hash* pode ser uma solução viável.

Apesar de ser considerado inseguro, este protocolo continua a ser utilizado devido à sua elevada eficiência e simplicidade. Um exemplo é as aplicações *Signal* e *Secret* [1] que recorrem a este método para identificar contactos de utilizadores. Nestas aplicações, a lista de contactos

é convertida em valores *hash*, que são comparados com os *hashes* existentes na base de dados da aplicação, o que exige desempenho e escalabilidade, proporcionada pela abordagem *naive hashing*. Outro exemplo prático é o serviço utilizado pelo Facebook em colaboração com empresas como *Datalogix*, *Epsilon* e *Acxiom* [1], para calcular taxas de conversão de anúncios, através de uma variante deste protocolo. Neste contexto, cada parte aplica a função *hash* aos elementos e intersesta os valores *hash* para encontrar correspondências. Tal como no exemplo anterior, a motivação para utilizar este protocolo deve-se à eficiência computacional.

## 2.2 Protocolo *Server-Aided*

Neste protocolo, além das duas partes envolvidas, existe um terceiro participante, um servidor externo.

O servidor [2] não é confiável, mas assume-se que não colabora secretamente com nenhuma das partes para comprometer a privacidade do protocolo. Recebe apenas valores que são transformados por uma função *pseudorandom permutation* (PRP), não tendo, assim, acesso aos dados originais. A sua função é apenas fornecer capacidade computacional para aliviar os participantes, limitando-se a realizar a interseção sem conhecer o seu conteúdo. Assim, só os envolvidos têm acesso ao resultado da operação. O objetivo é permitir a execução em conjuntos com muitos elementos e a escalabilidade do protocolo sem comprometer a segurança.

Tal como no protocolo visto anteriormente, cada participante  $P_i$  possui um conjunto privado  $S_i$ . Neste contexto, todos os participantes acordam previamente uma PRP com uma chave secreta comum  $K$ . Cada parte aplica, então, a PRP ao seu conjunto, obtendo  $\pi_i(F_K(S_i))$ , onde  $\pi_i$  é uma permutação aleatória, e envia o resultado ao servidor. O servidor calcula a interseção dos valores que recebeu e retorna o resultado aos envolvidos. Finalmente, cada parte aplica a função inversa da PRP,  $F_K^{-1}$  para recuperar os elementos originais da interseção [2].

O protocolo é seguro no modelo com servidor semi-honesto, onde se assume que o servidor segue corretamente o protocolo, embora possa tentar inferir informações confidenciais a partir dos dados que recebe. O uso da PRP garante que os elementos enviados ao servidor são indistinguíveis de valores aleatórios, o que impede a extração de informação. Além disso, o protocolo também é seguro contra participantes maliciosos, desde que o servidor não colabore com eles, permanecendo neutro devido aos incentivos legais e à sua reputação – modelo de segurança *non-collusion*. No caso do servidor ser malicioso, que pode alterar o protocolo ou manipular os resultados, os participantes devem repetir os elementos do seu conjunto várias vezes e introduzir elementos falsos (*dummies*) com uma estrutura específica – garantem que a interseção nunca é vazia – que funcionam como mecanismos de verificação de integridade da resposta do servidor. Se o servidor for malicioso e não devolver a interseção correta deve ser detetado pelos envolvidos, uma vez que a estrutura com os elementos falsos torna estatisticamente improvável (com uma probabilidade de  $\frac{1}{t^{\lambda-1}} + \text{negl}(k)$ , onde  $t$  é o número de *dummies* por subconjunto,  $\lambda$  é o número de repetições de cada elemento e  $\text{negl}(k)$  é uma função negligenciável no parâmetro  $k$ ) que adulterações não sejam percebidas [2].

## 2.3 Protocolo baseado em *Diffie-Hellman*

Este protocolo tem como objetivo permitir que as duas partes – A e B – verifiquem se partilham um segredo em comum, sem revelarem os seus dados privados a menos que exista uma correspondência. Para isso, o protocolo utiliza operações de exponenciação modular com base em propriedades matemáticas do algoritmo Diffie-Hellman [3].

Deste modo, cada participante possui um segredo – A tem  $S_A$ , B tem  $S_B$ , com  $S_A, S_B \in \mathbb{Z}_p^*$ , onde  $p$  é um primo partilhado entre as partes. Além disso, cada um escolhe um número privado aleatório  $M_A$  e  $M_B$ . A segurança do protocolo baseia-se na propriedade comutativa da

exponenciação modular, representada por  $(S_B^{M_B})^{M_A} = (S_B^{M_A})^{M_B}$  [3]. Esta propriedade permite que ambas as partes obtenham um valor comum  $K$  (que pode posteriormente ser utilizado como chave partilhada) sem que uma delas conheça o segredo da outra.

O protocolo decorre da seguinte forma: A envia a B o valor  $S_A$  e B envia a A o valor  $S_B$ . Em seguida, A calcula  $S_B^{M_A} \bmod p$  e envia o resultado a B; simultaneamente, B calcula  $S_A^{M_B} \bmod p$  e envia o resultado a A. A partir dos valores recebidos, A calcula o valor comum  $K = (S_A^{M_B})^{M_A} \bmod p$ , enquanto B calcula  $K = (S_B^{M_A})^{M_B} \bmod p$  [3].

Dado que a exponenciação modular é comutativa neste contexto, ambos chegam ao mesmo valor  $K$ , que é a chave secreta partilhada resultante da operação. Esta abordagem permite confirmar se existe correspondência entre os segredos sem os revelar, garantindo, assim, privacidade mútua, no caso de não existir correspondência.

Contudo, esta versão do protocolo requer um grau elevado de confiança, pois um dos participantes pode fingir uma correspondência. Para mitigar este risco, o protocolo é reforçado com técnicas criptográficas adicionais [3]. Em vez de operar em  $\mathbb{Z}_p$ , passa a operar no grupo multiplicativo  $\mathbb{Z}_N^*$ , onde  $N = (2pq + 1)(2r + 1)$ , sendo a segurança baseada na dificuldade de fatorizar  $N$ . Utiliza-se um gerador comum  $X \in \mathbb{Z}_N^*$  e ambas as partes computam um valor comum  $K = X^{M_B M_A} \bmod N$ , que é utilizado como chave para a comunicação subsequente. Os segredos são codificados como  $S = u^a \bmod N$ , com parâmetros escolhidos de forma segura e autenticados com assinaturas digitais. A verificação da autorização mútua é assegurada pelo facto de que seria necessário conhecer  $\phi(N) = 4pqr$  — algo considerado computacionalmente inviável sem acesso à fatorização de  $N$ , o que seria equivalente a quebrar o RSA. Assim, o protocolo garante robustez mesmo perante participantes potencialmente maliciosos.

## 2.4 Protocolo baseado em *Oblivious Transfer*

Este protocolo é uma abordagem eficiente e escalável, adequada para conjuntos de dados de grande dimensão. A segurança do protocolo é fundamentada na utilização de *Oblivious Pseudo-Random Functions* (OPRFs), cuja avaliação é realizada de forma eficiente com recurso a *OT extensions*. Neste processo, o servidor avalia previamente a PRF sobre múltiplos pontos e a *OT extension* permite que o cliente obtenha apenas a avaliação correspondente ao seu *input*, sem aprender as restantes saídas nem a chave privada do servidor. Esta abordagem reduz significativamente o custo computacional, uma vez que se baseia predominantemente em operações criptográficas simétricas.

O protocolo utiliza uma estrutura de *hashing*. Inicialmente, o cliente aplica *Cuckoo hashing*, que utiliza múltiplas funções *hash* para evitar colisões, para distribuir os seus elementos em *bins* — entradas na tabela de *hash* onde as *hashes* dos elementos são guardadas —, garantindo que cada elemento ocupe um único *bin*. Paralelamente, o servidor aplica *simple hashing* para distribuir os seus elementos por múltiplos *bins* e introduzir *dummies* que contribuem para proteger contra inferências estatísticas, ao ocultar o verdadeiro número de elementos em cada *bin*. Para lidar com eventuais falhas na inserção de elementos é utilizado um *stash*, onde esses elementos são temporariamente guardados. O protocolo lida com estes casos realizando uma avaliação adicional da OPRF de todo o conjunto do servidor. Além disso, para garantir a unicidade dos elementos em cada *bin* antes da aplicação da OPRF, mesmo quando diferentes funções *hash* mapeiam o mesmo elemento para o mesmo *bin*, o protocolo usa uma técnica de *hashing* baseada em permutação. Essa representação inclui também o índice da função *hash* usada, evitando colisões prematuras.

Na fase seguinte, ambos os participantes avaliam OPRFs através de *OT extension* —  $\binom{N}{1}$ -OT [1]. Para cada *bin*, o servidor atua como remetente e o cliente como recetor. O servidor possui  $N = 2^\mu$  mensagens possíveis, correspondentes às potenciais entradas de  $\mu$  *bits* resultantes do *hashing* baseado em permutação. O cliente calcula o índice do seu elemento no *bin* e obtém,

através de OT, apenas a avaliação da OPRF nesse ponto, sem aprender a chave da OPRF do servidor. Paralelamente, o servidor define uma chave secreta  $t_i$  para cada *bin*  $i$ , e avalia a OPRF com essa chave sobre cada um dos seus próprios elementos nesse *bin*. O conjunto de máscaras resultantes é então permutado e enviado para o cliente. Este, por sua vez, compara os valores recebidos com as suas próprias máscaras calculadas localmente a partir da OPRF. Uma correspondência entre máscaras indica que o elemento do cliente está presente no conjunto do servidor, permitindo determinar a interseção de forma segura e eficiente, preservando a confidencialidade dos restantes elementos.

Para garantir a robustez do protocolo contra falsificação de interseções e colisões de máscaras, é utilizado um comprimento de máscara definido por  $\ell = \lambda + \log_2(kn_1) + \log_2(n_2)$ , onde  $\lambda$  é o parâmetro de segurança estatístico,  $k$  é o número de funções de *hash* utilizadas e  $n_1, n_2$  são os tamanhos dos conjuntos das duas partes envolvidas no protocolo. Este valor de  $\ell$  assegura que a probabilidade de colisões entre elementos diferentes – que poderiam levar a falsos positivos na interseção – sejam estatisticamente negligenciável. Como os *outputs* da OPRF são indistinguíveis de valores aleatórios, a probabilidade de duas máscaras colidirem é no máximo  $2^{-\ell}$ , garantindo que a probabilidade do protocolo funcionar sem erros é de pelo menos  $1 - 2^{-\lambda}$ . A análise do funcionamento correto do protocolo baseia-se, portanto, na escolha adequada deste comprimento de máscara, que depende diretamente da dimensão dos conjuntos e do número de funções de *hash*.

O protocolo oferece segurança contra adversários semi-honestos, podendo ser reforçado para resistir a adversários maliciosos através de mecanismos como verificações de consistência. Além disso, a propriedade de *obliviousness* das OPRFs assegura que o cliente não obtém qualquer informação sobre a chave do servidor, nem consegue inverter os resultados da OPRF para deduzir elementos do conjunto do outro participante.

## 2.5 Comparação dos Protocolos de PSI

É realizada uma comparação dos quatro protocolos de PSI analisados: *naive hashing*, *server-aided*, protocolo baseado em DH e protocolo baseado em OT. A análise considera critérios como **eficiência**, **escalabilidade**, **segurança** e **aplicabilidade prática**, com base nos resultados experimentais e discussões dos artigos analisados.

- *Naive Hashing*: Este protocolo é o mais **simples** e **eficiente** em termos computacionais e de comunicação. De acordo com [1], esta abordagem tem várias ordens de magnitude mais rápida do que protocolos PSI criptograficamente seguros, servindo frequentemente como base nas comparações de desempenho. Contudo, apresenta **fracas garantias de segurança**, sendo vulnerável a ataques de *brute force* e/ou dicionário quando os dados têm baixa entropia.
- *Server-Aided*: Neste protocolo, o servidor apenas realiza uma interseção, utilizando algoritmos lineares [2]. O protocolo é bastante **escalável** – foi testado com conjuntos muito grandes e apresentou desempenho muito próximo do de uma interseção — apenas 10% mais lento [2]. A eficiência obtida **reduz drasticamente o custo computacional dos clientes** e permite execução prática em diversos ambientes. A principal **limitação** reside no pressuposto de que o **servidor não colabora com nenhuma das partes**, e em versões básicas não protege dados, como o tamanho da interseção – o protocolo *SizePSI* foi desenhado para mitigar esse problema.
- Baseado em DH: Este protocolo tem uma implementação relativamente **simples** e não requer uma terceira parte após a fase de inicialização com uma **autoridade confiável** [3]. No entanto, a sua segurança depende de problemas difíceis como o logaritmo discreto

ou a fatorização de inteiros. Além disso, a **escalabilidade** do protocolo **não foi pensada para grandes conjuntos de elementos**, sendo mais adequado para verificar a correspondência entre entidades individuais.

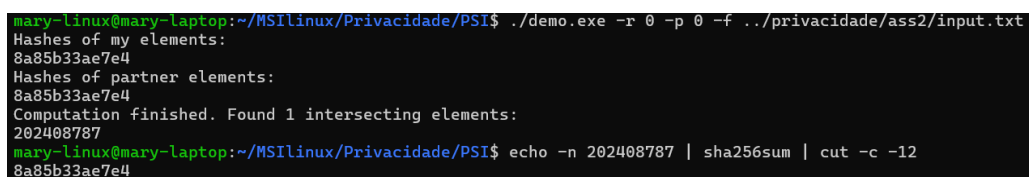
- Baseado em OT: Esta abordagem é **bastante escalável** e tem fortes garantias de **segurança**, sendo segura contra adversários semi-honestos e extensível para o modelo malicioso. Utiliza OPRFs instanciadas com OT *extensions*, o que **reduz drasticamente a carga computacional**, baseando-se apenas em operações simétricas. O protocolo também suporta *hashing* avançado – *Cuckoo hashing* – para distribuir os elementos e é paralelizado, pois pode executar o processo de inserção dos seus elementos na tabela *hash*, incluindo potenciais relocações de elementos para evitar colisões, de forma independente e sem precisar de interagir com a outra parte, podendo assim pré-processar etapas do protocolo. A principal desvantagem está na **complexidade** da implementação [1].

Em conclusão, os protocolos analisados oferecem **diferentes compromissos** entre **eficiência**, **segurança**, **complexidade** e **escalabilidade**. O *naive hashing* destaca-se pela sua simplicidade e velocidade, sendo útil para contextos onde a segurança é menos crítica. O protocolo *server-aided* surge como uma alternativa prática e escalável, desde que o pressuposto de um servidor honesto seja aceitável. O protocolo baseado em *Diffie-Hellman* oferece uma solução criptograficamente sólida, mas com limitações de escalabilidade, sendo mais apropriado para contextos com poucos elementos. Por fim, o protocolo baseado em *OT* revela-se o mais robusto e versátil, proporcionando um bom equilíbrio entre segurança e desempenho, sendo particularmente adequado para aplicações práticas que exigem uma privacidade rigorosa e capacidade de lidar com grandes volumes de dados.

## 3 Teste dos Protocolos PSI

### 3.1 Protocolo *Naive Hashing*

De modo a executar este protocolo, criou-se um ficheiro `input.txt` com apenas uma linha `"202408787"`.



```
mary-linux@mary-laptop:~/MSIlinux/Privacidade/PSI$ ./demo.exe -r 0 -p 0 -f ../privacidade/ass2/input.txt
Hashes of my elements:
8a85b33ae7e4
Hashes of partner elements:
8a85b33ae7e4
Computation finished. Found 1 intersecting elements:
202408787
mary-linux@mary-laptop:~/MSIlinux/Privacidade/PSI$ echo -n 202408787 | sha256sum | cut -c -12
8a85b33ae7e4
```

Figura 1: *Output* do protocolo para o ficheiro `input.txt`.

Posto isto, obteve-se o esperado, já que cada parte tinha exatamente os mesmos dados (`input.txt`) e o protocolo retornou corretamente a sua interseção.

Na Fig.1, é notório que as *hashes* trocadas são iguais e que correspondem ao único texto de `input.txt`. Para além disso, são os únicos elementos a serem enviados e recebidos pelos *sockets* e apenas a sua interseção é revelada, como se corfima pelas duas linhas do Lst.1, retiradas de `naive-psi.cpp`.

```
86     snd_and_rcv(hashes, neles * maskbytelen, phashes, pneles * maskbytelen,
    tmpsock);

    ...
```

```
106 intersect_size = find_intersection(hashses, neles, phashes, pneles,
    maskbytelen, perm, matches);
```

**Listing 1:** Excerto de código do ficheiro naive-psi.cpp.

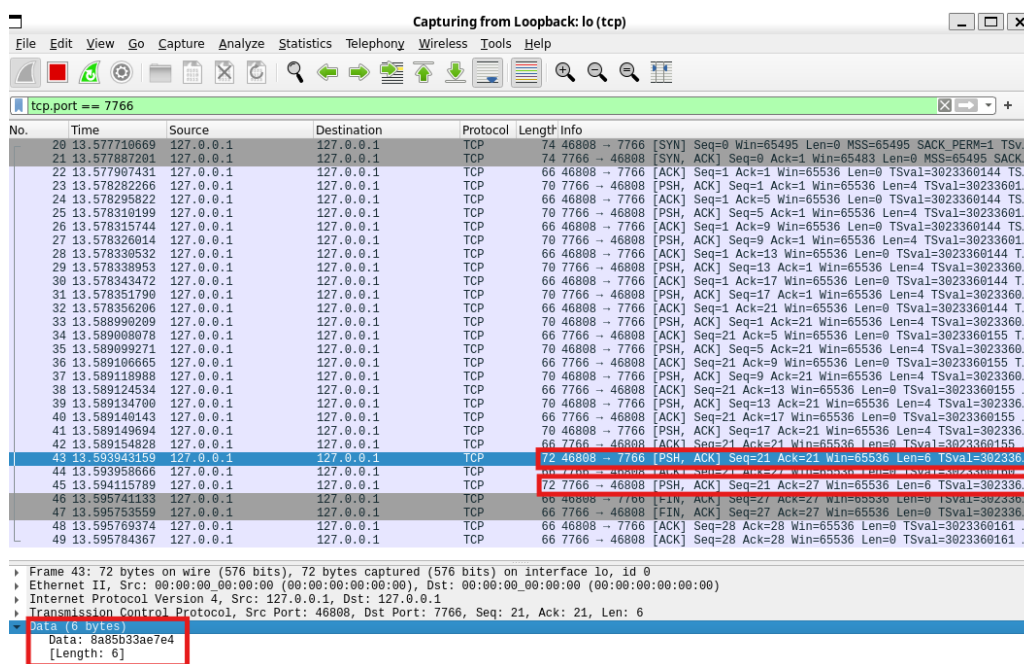
De seguida, criou-se um ficheiro `input2.txt` com uma única linha `"test"`. Dado que os ficheiros `input.txt` e `input2.txt` não têm elementos em comum, observa-se, na Fig.2, que as *hashses* partilhadas são diferentes e a interseção é vazia.

```
mary-linux@mary-laptop:~/MSILinux/Privacidade/PSI$ ./demo.exe -r 0 -p 0 -f ../privacidade/ass2/input2.txt
Hashses of my elements:
9f86d081884c
Hashses of partner elements:
8a85b33ae7e4
Computation finished. Found 0 intersecting elements:
```

**Figura 2:** Output do protocolo para os ficheiros `input.txt` e `input2.txt`.

Durante as execuções anteriores, utilizou-se o *Wireshark* para captar pacotes tcp com o filtro `tcp.port == 7766`, a porta padrão para este tipo de pacotes.

Na Fig.3, observa-se que se conseguiu identificar os dois pacotes onde são trocadas as *hashses* de cada parte. Cada pacote com 6 *bytes* de dados, um com destino à porta 7766 e outro proveniente da mesma. No caso da primeira execução, os dados trocados são iguais. Por outro lado, no caso da segunda execução, esses dados são diferentes.



**Figura 3:** Pacotes tcp captados com o *Wireshark*, durante as execuções anteriores.

### 3.1.1 Vulnerabilidades - Ataque de Dicionário

Apesar da sua simplicidade e eficiência, o protocolo *Naive Hashing* apresenta vulnerabilidades críticas em termos de segurança. Em particular, está sujeito a **ataques de dicionário**, quando o **domínio** dos dados é **pequeno** ou **previsível**, como é o caso dos números de estudantes, que contém com baixa entropia.

Durante a execução do protocolo, cada parte calcula os *hashses* dos seus elementos e envia-os diretamente à outra parte. Se um adversário, como Bob, tem intenções maliciosas, ou um atacante externo com acesso à rede (por exemplo, utilizando o *Wireshark* para monitorizar a interface loopback na porta 7766), conseguir capturar estes *hashses*, torna-se trivial realizar



um ataque de dicionário. Isto consiste em gerar uma tabela de *hashes* para todos os valores possíveis dentro de um determinado domínio e compará-los com os *hashes* recebidos, revelando assim os elementos originais da outra parte.

Este ataque é facilitado pelo facto de o protocolo utilizar SHA-256 truncado aos 6 *bytes* mais significativos, o que reduz significativamente o domínios de valores de saída e, consequentemente, aumenta a probabilidade de colisões e facilita ataques de pré-imagem. Embora a operação de truncar tenha sido realizada por motivos de eficiência, compromete a robustez do protocolo contra **ataques de brute force** ou dicionário.

Adicionalmente, como os *hashes* são enviados diretamente pelas partes sem qualquer forma de encriptação ou aleatorização (como *salts*), o protocolo viola o princípio principal: nenhuma das partes (ou terceiros) deve ser capaz de aprender mais do que a própria interseção. No caso do *Naive Hashing*, as partes podem deduzir facilmente os elementos do outro conjunto, mesmo que não pertençam à interseção, o que constitui uma falha de confidencialidade.

Deste modo, conclui-se que, apesar de ser computacionalmente eficiente, o protocolo *Naive Hashing* não é adequado para operações que exijam garantias de privacidade.

### 3.1.2 Execução para AppList1.csv e AppList2.csv

Os passos anteriores foram executados novamente para as listas AppList1.csv e AppList2.csv e o *output* do protocolo foi guardado nos ficheiros output1.txt e output2.txt. A interseção encontrada apresenta-se no Lst.2.

```
Computation finished. Found 5 intersecting elements:
com.whatsapp
org.meowcat.edxposed.manager
com.google.android.apps.maps
com.android.chrome
com.delaware.empark
```

**Listing 2:** *Output* do protocolo para as listas AppList1.csv e AppList2.csv.

Durante a realização do protocolo, foram obtidas as estatísticas da Fig.4, através do *Wireshark*. Observa-se que foram captados **57 pacotes**, correspondentes a **6353 bytes** de dados.

Statistics			
Measurement	Captured	Displayed	Marked
Packets	38	38 (100.0%)	—
Time span, s	5.001	5.001	—
Average pps	7.6	7.6	—
Average packet size, B	92	92	—
Bytes	3489	3489 (100.0%)	0
Average bytes/s	697	697	—
Average bits/s	5581	5581	—

**Figura 4:** Estatísticas dos pacotes captados, com filtro `tcp.port == 7766`.

## 3.2 Protocolo *Server-Aided*

Para executar este protocolo e de modo a "corrigir" o *bug* referido no enunciado, foram adicionadas duas linhas à lista AppList2.csv. Como *input file* (-f), foi dada a *sample emails\_alice.txt*. O *output* do servidor deste protocolo está representado na Fig. 5. O *output* do cliente corresponde ao representado acima na Lst.2.

```

Computing intersection for the clients
Inserting the items into the hash table
sending all 5 intersecting elements to the clients
mary-linux@mary-laptop:~/MSIlinux/Privacidade/PSI$

```

Figura 5: *Output* do protocolo na parte do servidor.

As estatísticas obtidas através do *Wireshark*, apresentam-se na Fig.6. Observa-se que foram captados **53 pacotes**, correspondentes a **5899 bytes** de dados.

Statistics			
Measurement	Captured	Displayed	Marked
Packets	53	53 (100.0%)	—
Time span, s	15.015	15.015	—
Average pps	3.5	3.5	—
Average packet size, B	111	111	—
Bytes	5899	5899 (100.0%)	0
Average bytes/s	392	392	—
Average bits/s	3142	3142	—

Figura 6: Estatísticas dos pacotes captados, com filtro `tcp.port == 7766`.

### 3.2.1 Vulnerabilidades

Na Fig. 5, observa-se parte do *output* do servidor, onde é indicada a presença dos cinco elementos da interseção mostrados na Lst. 2. No entanto, verifica-se que os dados recebidos pelo servidor correspondem exatamente às mesmas *hashes* utilizadas no protocolo *Naive Hashing*, embora neste caso não estejam truncadas. Esta semelhança revela **riscos de privacidade** associadas ao **papel do servidor** como terceiro elemento no protocolo.

Em primeiro lugar, o servidor tem acesso direto às *hashes* dos elementos enviados por ambos os clientes. Tal como no *Naive Hashing*, este acesso possibilita a execução de ataques de dicionário, nos quais o servidor pode, computar *hashes* de todos os elementos do domínio reduzido e com baixa entropia e compará-los com os valores recebidos, recuperando assim informação sensível dos conjuntos originais.

Adicionalmente, um protocolo de PSI ideal deve assegurar que apenas a interseção entre os conjuntos é revelada, e apenas às partes legítimas. No entanto, neste modelo, o servidor – que atua como terceiro neutro – obtém toda a informação necessária para reconstruir parcialmente ou totalmente os conjuntos dos clientes, mesmo fora da interseção. Isto compromete a premissa de mínima divulgação de informação.

Caso o servidor seja malicioso ou venha a ser comprometido, os dados dos clientes ficam expostos. A falta de mecanismos criptográficos adicionais torna o protocolo frágil sob adversários que não sigam o modelo semi-honesto. Assim, apesar de eficiente, o protocolo *Server-Aided* apresenta riscos consideráveis de privacidade e deve ser utilizado apenas em contextos onde o servidor seja confiável ou em ambientes controlados.

## 3.3 Protocolo baseado em *Diffie-Hellman*

Ao se executar este protocolo, percebeu-se que apenas o terminal com a opção `-r 1` computou a interseção. Na Fig.9, observa-se, mais uma vez, a interseção esperada em todos os protocolos.

```

mary-linux@mary-laptop:~/MSIlinux/Privacidade/PSI$ ./demo.exe -r 1 -p 2 -f ../privacidade/ass2/samples_prof/AppList2.csv
Computation finished. Found 5 intersecting elements:
com.whatsapp
org.meowcat.edxposed.manager
com.google.android.apps.maps
com.android.chrome
com.delaware.empark

```

Figura 7: *Output* do protocolo no terminal com a opção `-r 1`.

As estatísticas obtidas através do *Wireshark*, apresentam-se na Fig.8. Observa-se que foram captados **38 pacotes**, correspondentes a **6440 bytes** de dados.

Statistics			
Measurement	Captured	Displayed	Marked
Packets	38	38 (100.0%)	—
Time span, s	5.156	5.156	—
Average pps	7.4	7.4	—
Average packet size, B	169	169	—
Bytes	6440	6440 (100.0%)	0
Average bytes/s	1249	1249	—
Average bits/s	9993	9993	—

Figura 8: Estatísticas dos pacotes captados, com filtro `tcp.port == 7766`.

### 3.4 Protocolo baseado em OT

Durante a execução do último protocolo, percebeu-se novamente que apenas o terminal com a opção `-r 1` computou a interseção. Na Fig.9, observa-se, mais uma vez, a interseção esperada em todos os protocolos.

```

mary~linux@mary-laptop:~/MSIlinux/Privacidade/PSI$ ./demo.exe -r 1 -p 3 -f ../privacidade/ass2/samples_prof/AppList2.csv
Hashing 36 elements with arbitrary length into 7 bytes
Client: bins = 44, elebitlen = 51 and maskbitlen = 56 and performs 44 OTs
Computation finished. Found 5 intersecting elements:
com.whatsapp
org.meowcat.edxposed.manager
com.google.android.apps.maps
com.android.chrome
com.delaware.empark

```

Figura 9: *Output* do protocolo no terminal com a opção `-r 1`.

As estatísticas obtidas através do *Wireshark*, apresentam-se na Fig.10. Observa-se que foram captados **46 pacotes**, correspondentes a **55726 bytes** de dados.

Statistics			
Measurement	Captured	Displayed	Marked
Packets	46	46 (100.0%)	—
Time span, s	5.227	5.227	—
Average pps	8.8	8.8	—
Average packet size, B	1211	1211	—
Bytes	55726	55726 (100.0%)	0
Average bytes/s	10 k	10 k	—
Average bits/s	85 k	85 k	—

Figura 10: Estatísticas dos pacotes captados, com filtro `tcp.port == 7766`.

### 3.5 Segurança vs. Qualidade da Comunicação

Nas anteriores Sec. 3.1.2, 3.2, 3.3 e 3.4, foram captados e registados os pacotes `tcp` enviados durante a execução de cada protocolo.

Protocolo	N.º de Pacotes	Tamanho dos Dados
<i>Naive Hashing</i>	38	3489
<i>Server-Aided</i>	53	5899
Baseado em DH	38	6440
Baseado em OT	46	55726

Tabela 1: Comparação entre diferentes protocolos.

A Tab.1 evidencia um claro compromisso entre segurança e eficiência na comunicação.

O protocolo ***Naive Hashing*** regista um volume de dados mais reduzido, com apenas 38 pacotes e um total de 3489 bytes transmitidos. Esta eficiência advém da natureza simplista, na qual apenas *hashes* truncadas (6 bytes) são trocadas entre as partes. Contudo, este baixo custo traduz-se em limitações de segurança: as *hashes* partilhadas podem ser facilmente alvo de ataques de dicionário, comprometendo a confidencialidade dos dados quando o domínio é previsível ou com baixa entropia.

Já o protocolo ***Server-Aided*** apresenta um aumento nos valores obtidos (53 pacotes e 5899 bytes), o que reflete a adição de uma terceira parte no processo (maior complexidade). Esta configuração visa facilitar a computação da interseção e reduzir o custo computacional dos clientes, mas introduz riscos adicionais se o servidor for malicioso ou comprometido. Em termos comunicacionais, o custo continua relativamente contido, mas a segurança permanece limitada, uma vez que os dados enviados continuam a ser *hashes* diretamente comparáveis.

O protocolo baseado em **Diffie-Hellman** mantém o mesmo número de pacotes do *Naive Hashing* (38), mas o volume de dados sobe para 6440 bytes. Este crescimento reflete a utilização de chaves públicas e operações criptográficas mais complexas, que aumentam significativamente a segurança contra adversários.

Por fim, o protocolo baseado em ***Oblivious Transfer*** apresenta o maior custo comunicacional, com 46 pacotes e 55726 bytes transmitidos — quase 10 vezes mais do que os restantes. Este aumento está diretamente associado ao uso de OPRFs e *OT extensions*, que garantem fortes propriedades de privacidade, mesmo perante adversários semi-honestos, e que podem ser estendidas para resistir a adversários maliciosos. No entanto, este elevado nível de segurança implica uma maior exigência computacional, podendo limitar a sua aplicabilidade em sistemas com recursos restritos.

Em conclusão, a escolha do protocolo mais adequado dependerá dos requisitos específicos: protocolos como *Naive Hashing* e *Server-Aided* são eficientes mas pouco seguros, enquanto soluções baseadas em *OT* oferecem maior proteção à custa de um custo mais elevado. O equilíbrio entre desempenho e segurança deve, portanto, ser ponderado, tendo em conta a sensibilidade dos dados e do ambiente de execução.

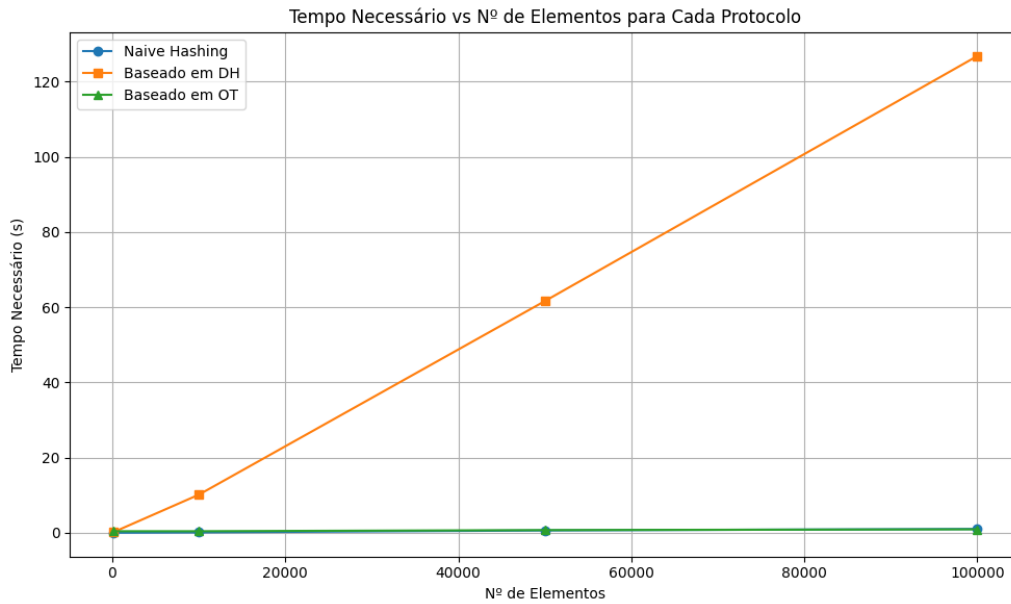
## 4 ***Benchmarking*** dos protocolos PSI

Nesta secção procedeu-se com a técnica de *Benchmark* para os protocolos *Naive Hashing*, Baseado em DH e Baseado em OT. Na Tab.1, apresentam-se os resultados obtidos. De igual forma, a Fig.11 representa o Tempo Necessário para executar cada protocolo considerando o N<sup>o</sup> de Elementos do conjunto utilizado. Já na Fig.12, observa-se a relação entre o Total de Dados Recebidos e Enviados e o o N<sup>o</sup> de Elementos do conjunto utilizado.

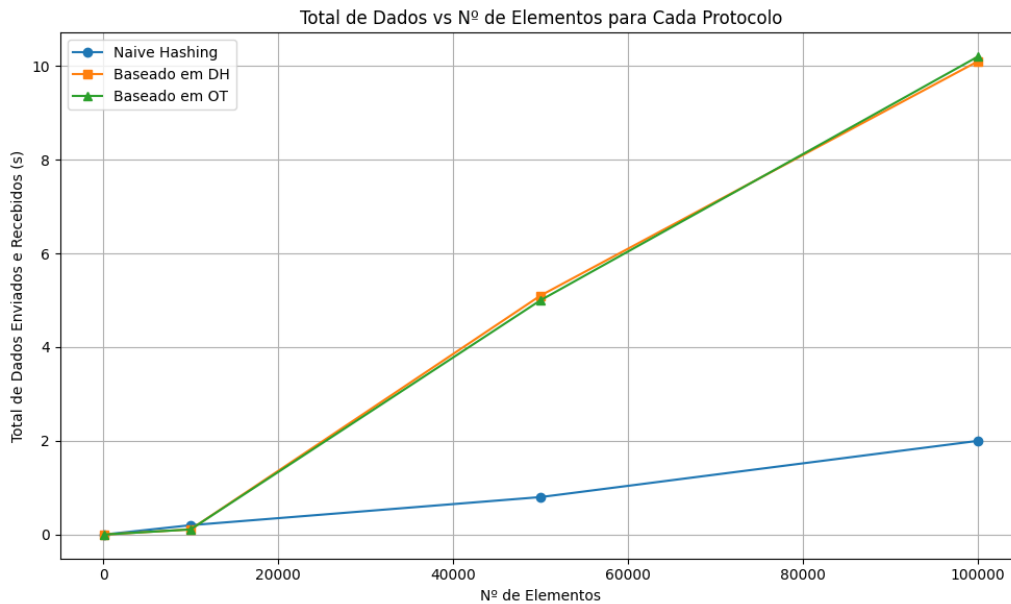
Protocolo	N <sup>o</sup> de Elementos	T. Necessário	D. Enviados	D. Recebidos
<i>Naive Hashing</i>	100	0.0 s	0.0 MB	0.0 MB
	10000	0.1 s	0.1 MB	0.1 MB
	50000	0.6 s	0.4 MB	0.4 MB
	100000	1.0 s	1.0 MB	1.0 MB
Baseado em DH	100	0.1 s	0.0 MB	0.0 MB
	10000	10.1 s	0.7 MB	0.4 MB
	50000	61.6 s	1.8 MB	3.3 MB
	100000	126.8 s	3.5 MB	6.6 MB
Baseado em OT	100	0.4 s	0.0 MB	0.0 MB
	10000	0.4 s	0.8 MB	0.3 MB
	50000	0.7 s	3.7 MB	1.3 MB

Protocolo	Nº de Elementos	T. Necessário	D. Enviados	D. Recebidos
	10000	0.9 s	7.3 MB	2.9 MB

**Tabela 2:** Tempo, Dados Enviados e Recebidos por protocolo e Nº de Elementos.



**Figura 11:** Representação de Tempo Necessário *vs* Nº de Elementos para cada protocolo.



**Figura 12:** Representação de Total de Dados *vs* Nº de Elementos para cada protocolo.

Com base nos resultados apresentados, é possível observar diferenças significativas no desempenho dos protocolos PSI avaliados, tanto em termos de tempo de execução como de volume de dados trocados.

O protocolo *Naive Hashing* destaca-se como a solução mais eficiente, apresentando tempos de execução consistentemente baixos e volumes reduzidos de dados transmitidos, mesmo com o aumento substancial do número de elementos. Esta abordagem, embora seja menos robusta do

ponto de vista da segurança, mostra-se adequada para contextos onde a eficiência se sobrepõe à confidencialidade, como em contextos de benchmarking ou ambientes controlados.

Por outro lado, o protocolo baseado em *Diffie-Hellman* revela um crescimento linear e acentuado no tempo de execução à medida que o tamanho do conjunto aumenta, acompanhado aumento progressivo no volume de dados trocados. Apesar de oferecer boas garantias de segurança baseadas em problemas difíceis, este protocolo revela-se o menos eficiente em tempo, tornando-se menos atrativo em casos com restrições de latência ou escalabilidade. Ainda assim, pode ser considerado em aplicações onde não é necessária alta performance, mas sim uma boa segurança criptográfica, ou seja, em contextos onde a segurança se sobrepõe à eficiência.

O protocolo baseado em *Oblivious Transfer*, por sua vez, embora apresente tempos de execução praticamente constantes, evidencia crescimento significativo no volume de dados trocados. Este comportamento reflete a forte ênfase na privacidade através de técnicas como OPRF e OT extension, implicando um custo computacional elevado. No entanto, o total de dados trocados é ligeiramente inferior ao protocolo de DH para conjuntos maiores, apesar de na Sec. 3.5 tal diferença não ser tão evidente. Este protocolo é indicado para contextos onde a privacidade dos dados é crítica, mesmo que isso implique um maior consumo de recursos.

Em suma, os testes de *benchmarking* reforçam que diferentes protocolos oferecem compromissos distintos entre segurança e desempenho, sendo a sua escolha dependente do contexto e das prioridades definidas. Protocolos mais seguros, como o baseado em OT, tendem a implicar maior carga computacional e comunicacional, enquanto soluções mais simples e rápidas, como o *Naive Hashing*, podem ser suficientes para aplicações com menor sensibilidade à exposição de dados.

## 5 Aplicação dos Protocolos a um *Dataset* Escolhido

Nesta secção ir-se-á obter dois *datasets* diferentes e calcular a sua interseção através dos protocolos estudados. Decidiu-se interseccionar as *Liked Songs* da aplicação *Spotify* e, para isso, utilizou-se a ferramenta *Exportify* para exportar as listas de músicas de cada parte em formato *.csv*.

A primeira lista, *liked\_songs\_maria.csv*, é constituída por 263 elementos e a segunda lista, *liked\_songs\_matilde.csv*, por 200 elementos. Relembrando que o protocolo *Server-Aided* só aceita *input files* com o mesmo número de linhas, foram adicionadas 63 linhas sem significado à segunda lista. Posto isto, procedeu-se à execução de cada protocolo abordado, como se observa nas Figs.13, 14, 15 e 16.

```
Computation finished. Found 6 intersecting elements:
Sailor Song
Iris
Riptide
See You Again (feat. Kali Uchis)
End of Beginning
Stick Season
```

Figura 13: Interseção calculada pelo protocolo *Naive Hashing*.

```
mary@laptop:~/Desktop/nsl/ano1/privacidade/PSI$ ./demo.exe -r 1 -p 1 -f ../privacidade/ass2/liked_son
gs_maria.csv -a 127.0.0.1
Computation finished. Found 6 intersecting elements:
Stick Season
Sailor Song
End of Beginning
Iris
See You Again (feat. Kali Uchis)
Riptide
```

Figura 14: Interseção calculada pelo protocolo *Server-Aided*.

```

mary@lnv-laptop:~/Desktop/psl/ano1/privacidade/PSI$ ./demo.exe -r 1 -p 2 -f ../privacidade/ass2/liked_son
gs_matilde.csv
Computation finished. Found 6 intersecting elements:
Sailor Song
Iris
Riptide
See You Again (feat. Kali Uchis)
End of Beginning
Stick Season

```

Figura 15: Interseção calculada pelo protocolo baseado em *Diffie-Hellman*.

```

mary@lnv-laptop:~/Desktop/psl/ano1/privacidade/PSI$ ./demo.exe -r 1 -p 3 -f ../privacidade/ass2/liked_son
gs_matilde.csv
Hashing 263 elements with arbitrary length into into 8 bytes
Client: bins = 316, elebitlen = 56 and maskbitlen = 64 and performs 316 OTs
Computation finished. Found 6 intersecting elements:
Sailor Song
Iris
Riptide
See You Again (feat. Kali Uchis)
End of Beginning
Stick Season

```

Figura 16: Interseção calculada pelo protocolo baseado em *Oblivious Transfer*.

O protocolo *Naive Hashing*, apesar de simples e eficiente, não oferece as garantias de segurança que se pretendem, sendo vulnerável a ataques passivos. Dado que os *datasets* escolhidos correspondem a um domínio bastante conhecido, um atacante poderia executar um ataque de dicionário (Sec. 3.1.1) utilizando, por exemplo, listas públicas de músicas mais ouvidas. De igual modo, o protocolo *Server-Aided* compromete parcialmente a privacidade, especialmente neste caso, onde a entidade intermediária — o servidor do protocolo — não é autenticada nem validada de forma confiável (Sec. 3.2.1).

Dado que os conjuntos de dados utilizados são bastante pequenos (263 e 200 elementos), tanto o protocolo baseado em *Diffie-Hellman* como o baseado em *Oblivious Transfer* são tecnicamente viáveis, tanto em termos de tempo de execução como de volume de dados trocados. No entanto, ao analisar os resultados de desempenho na Sec. 4, observa-se que, especificamente para um tamanho de conjunto de 100 elementos — o mais próximo do nosso caso — o protocolo baseado em DH é ligeiramente mais rápido (0.1 s contra 0.4 s) e transmite praticamente a mesma quantidade de dados (0.0 MB), o que o torna mais eficiente neste caso particular.

Assim, para conjuntos de pequena dimensão como os deste trabalho, o protocolo baseado em *Diffie-Hellman* revela-se a escolha mais eficiente. No entanto, à medida que a dimensão dos conjuntos aumenta, os gráficos e medições demonstram que o tempo de execução do DH cresce de forma acentuada, enquanto o protocolo baseado em OT mantém tempos praticamente constantes.

Além disso, o protocolo baseado em OT oferece garantias de privacidade superiores, pois protege eficazmente os elementos fora da interseção sem ser necessário reforçar o protocolo. Embora envolva um custo mais elevado, esse impacto torna-se mais relevante apenas em contextos com conjuntos de dados maiores.

Conclui-se, portanto, que o **protocolo baseado em *Diffie-Hellman* é mais indicado para o nosso caso específico**, dados os seus melhores resultados para pequenos volumes de dados. Contudo, em casos com conjuntos maiores, o protocolo baseado em OT continua a ser a opção mais equilibrada entre desempenho e segurança.

## 6 Conclusão

Ao longo deste trabalho, foi possível estudar, implementar e comparar quatro protocolos distintos de PSI, avaliando-os do ponto de vista da segurança, eficiência computacional e escalabilidade. Através das experiências realizadas com ferramentas como o *Wireshark*, e dos testes de

*benchmarking* com diferentes tamanhos de conjuntos, foi possível observar, de forma prática, o comportamento e as implicações de cada abordagem.

Este processo permitiu consolidar conhecimentos sobre *Secure Multiparty Computation* e demonstrar, na prática, como diferentes abordagens na realização da interseção privada de conjuntos podem influenciar diretamente os resultados obtidos, em função dos requisitos de cada caso.

De uma forma geral, este trabalho reforça a importância de uma análise bem fundamentada na seleção de mecanismos de privacidade, evidenciando que a escolha do protocolo mais adequado deve ter em conta o contexto específico, os objetivos de segurança e os recursos disponíveis.



## Referências

- [1] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. *Cryptology ePrint Archive*, 21, 1 2016.
- [2] Seny Kamara, Payman Mohassel, Mariana Raykova, and Saeed Sadeghian. Scaling private set intersection to billion-element sets. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8437:195–215, 2014.
- [3] Catherine Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. *IEEE Symposium on Security and Privacy*, pages 134–137, 7 1986.