Cláudia Maia - up201905492 e Cristina Pêra - up201907321

Análise e aplicação dos protocolos utilizados em *Secure Multiparty Computation*

Abstract

Neste trabalho, estudamos 4 protocolos possíveis de *Private Set Intersection*, de forma a poder intersetar *datasets* sem revelar informação privada de cada uma das *parties*. O relatório tem a seguinte estrutura: descrição, análise e comparação dos 4 protocolos de PSI (1); experimentação e comparação de resultados dos 4 protocolos (2); benchmarking dos protocolos de PSI (3); aplicação dos 4 protocolos PSI em *datasets* de *URL*s (4); e por fim a conclusão (5).

1. Protocolos desenvolvidos para o PSI

O *Private Set Intersection* (PSI) consiste num protocolo que permite duas *parties* calcularem a interseção dos seus conjuntos, tal que nenhuma informação acerca dos itens não incluídos na interseção seja revelada. Este protocolo criptográfico é aplicado em diversas áreas, incluindo a preservação de privacidade em *data mining*, serviços *location-based* e cálculos genómicos. Existem diversos protocolos desenvolvidos para o PSI, nomeadamente: *Naive Hashing*; *Server-aided*; *Diffie-Hellman-based PSI*; *OT-based PSI*.

1.1. Descrição e análise dos protocolos desenvolvidos para o PSI

1.1.1. *Naive Hashing*

Neste protocolo, ambas as *parties* aplicam uma função *hash* criptográfica aos seus *inputs* e comparam os *hashes*. Considera-se, assim as *parties* Alice e Bob que contêm os elementos $x_1,...,x_n$ e $y_1,...,y_n$, respetivamente. Em *Naive Hashing*, a Alice envia $H(x_1),...,H(x_n)$ para o Bob, que calcula de seguida $H(y_1),...,H(y_n)$, obtendo assim a interseção entre os dois conjuntos.

Este protocolo é eficiente, mas inseguro já que, depois de receber as funções *hash* $H(x_1),...,H(x_n)$ da Alice, o Bob pode realizar *brute force* de possíveis valores x_i contidos no conjunto da Alice, contrariando os princípios do PSI [1]. Se o *input* da Alice for de pequena dimensão e/ou baixa entropia, o protocolo revela-se ainda mais inseguro, já que o Bob consegue calcular as funções de *hash* de todo o domínio dos inputs, e assim descobrir o *dataset* da Alice. No entanto, mesmo que se verifique um valor elevado de entropia no *input*, não se garante segurança já que um atacante pode determinar se um dado elemento faz parte do conjunto de A depois da execução do protocolo [2].

1.1.2. Server-aided [3]

Neste protocolo é utilizado um pequeno conjunto de servidores (ou apenas um) que não contêm *inputs* e não recebem *outputs*, mas disponibilizam os seus recursos computacionais às *parties*. O objetivo é, assim, os servidores realizarem os cálculos da interseção pelas *parties*.

Considera-se S_i o conjunto de dados de uma *party* P_i. As *parties* começam por gerar uma chave secreta *K* de *k-bits* para uso numa permutação pseudo-aleatória *F*. Cada *party* permuta o conjunto $F_k(S_i)$, que consiste em *labels* calculadas pela aplicação da permutação *F* sobre os elementos de S_i, enviando de seguida o conjunto permutado para o servidor. O servidor realiza o cálculo de interseção e retorna a interseção das *labels*.

Como demonstrado em [3], o uso deste modelo no PSI permite o uso de conjuntos de milhares de milhões de elementos entre as *parties*, já que o servidor é responsável pela computação, comprovando assim a eficiência do protocolo.

1.2.3. Diffie-Hellman-based PSI [4]

Este protocolo baseia-se nas propriedades comutativas da função *Diffie-Hellman* ($g^{ab} \equiv g^{ba}(mod\ p)$). São geradas duas chaves de encriptação para cada *party*. Cada *party* encripta os seus elementos, recorrendo a uma função *Diffie-Hellman* e à sua chave, de seguida partilham o *dataset* encriptado com a outra *party* que encripta mais uma vez os dados com a sua chave, e por fim a *party* que escreve o *output* compara os *datasets* duplamente encriptados, se encontrar correspondências, é porque estas pertencem à interseção. Sem a posse de alguma das chaves partilhadas (da Alice ou do Bob), um *eavesdropper* não consegue descobrir os elementos de nenhum dos conjuntos de elementos, nem é capaz de saber os resultados da interseção, pois teria de saber as duas chaves para desencriptar a interseção. Assim sendo consideramos este protocolo seguro em comparação aos dois anteriores. Como são realizados vários cálculos e comparações neste protocolo, é de prever que o cálculo da interseção será mais lento.

1.2.4. OT-based PSI [5]

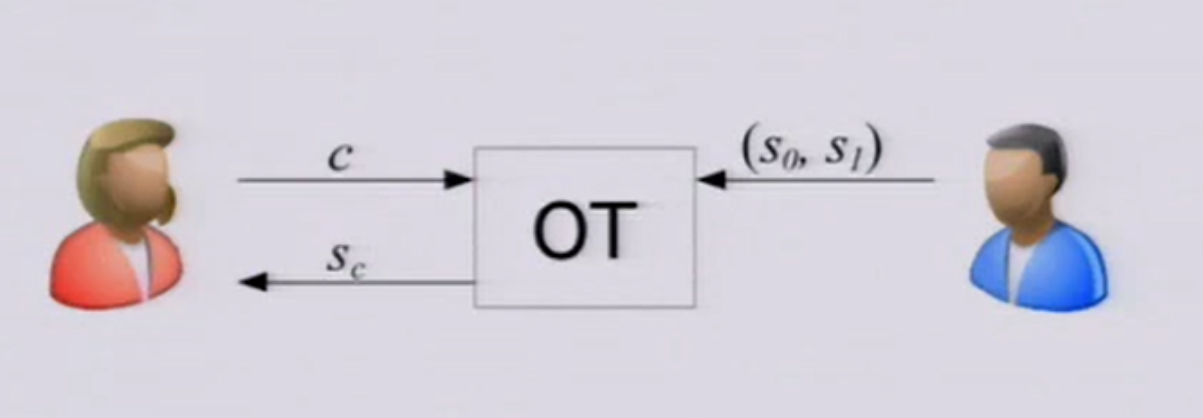

Figura 1. Funcionamento do protocolo Oblivious Transfer

Num protocolo *Oblivious Transfer* (OT), considerando as duas *parties* Alice e Bob, a Alice contém um *choice bit c* e o Bob contém duas *strings* (s_0, s_1). A Alice pretende obter uma das strings do Bob, recebe, assim, s_c mas não obtém qualquer informação acerca de s_{1-c}, e o Bob não obtém informação acerca de *c* (figura 1). O Bob não tem conhecimento de qual *string* foi escolhida pela Alice.

No protocolo *OT-based PSI*, são utilizados elementos *hash*, colocados em *bins* tendo em conta o *output* da função *hash* de cada *party*. Considerando, por exemplo, que a Alice contém X = {x1,…,x4} e Bob contém Y = {y1,…,y5}, obtém-se:

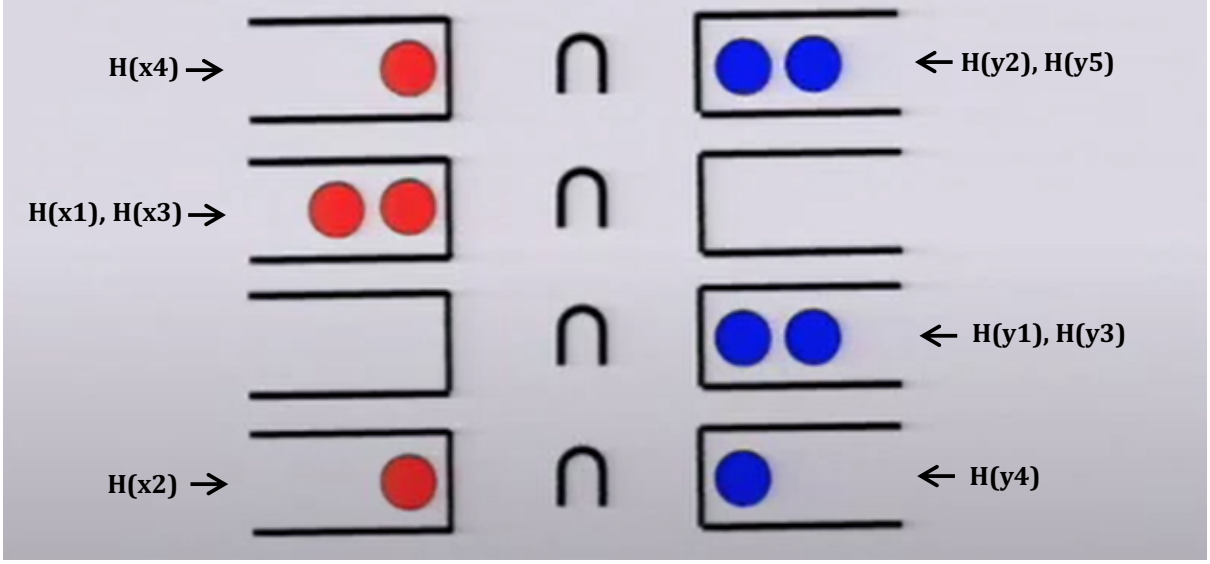

Figura 2. Interseção dos elementos no Protocolo OT-based PSI

Para calcular a interseção dos elementos basta, assim, realizar o cálculo entre os *bins*. Isto permite a redução da complexidade do cálculo das interseções de O(n^2) para O($n\ logn$).

1.2. Comparação dos protocolos

Com base nas definições, podemos prever que o protocolo *Naive Hashing*, embora seja o mais eficiente, será o mais inseguro, uma vez que a privacidade dos *datasets* das *parties* não é garantida, já que se o domínio dos *inputs* não for grande o suficiente, através de *brute force*, a *party* que recebe os *hashes* dos elementos, consegue descobrir os elementos da outra *party*.

O protocolo *Server-aided*, embora garanta a privacidade das parties entre elas, como o servidor pode ser atacado, deixa de garantir a privacidade das mesmas. Para além de inseguro, se o servidor for atacado, este protocolo é menos eficiente que o *Naive Hashing* uma vez que utiliza mais recursos (uma *third-party*) e são necessárias mais comunicações consequentemente.

O protocolo *Diffie-Hellman*, ao contrário dos dois anteriores, garante a privacidade dos *datasets* desde que as chaves das suas funções de *Diffie-Hellman* se mantenham secretas. No entanto, este protocolo é muito menos eficiente, pois tem de fazer muitas comparações e computações o que o torna mais lento e requer mais comunicações.

Por fim, o protocolo *OT-based*, embora requeira transferência de muita informação, garante a privacidade das duas *parties* e tem uma complexidade temporal baixa. Assim, é de prever que o protocolo *OT-based* consegue fazer o melhor balanço de eficiência e segurança dos *datasets*. E, em conjunto com o *Diffie-Hellman,* conseguem fazer o melhor balanço entre custo de comunicação e segurança.

2. Aplicação e análise dos protocolos desenvolvidos para o PSI

Depois de perceber o funcionamento de cada protocolo desenvolvido no âmbito do PSI, sucede-se à aplicação destes protocolos para estudo e avaliação dos resultados, verificando-se que o resultado da interseção é o mesmo para cada protocolo. Nos seguintes passos efetuados, consideram-se os terminais 1 e 2 como *parties* A e B, respetivamente.

2.1. Protocolo *Naive-Hashing*

Como analisado nos pontos 1, 2 e 3 do *Step #2*, é possível verificar que o uso do protocolo *Naive Hashing*, que requer apenas o uso de uma função *hash* aplicada aos *inputs* de cada *party*, é inseguro. De facto, depois de B receber o *input* (ficheiros input ou AppList1.csv), pode realizar *brute force* de todos os valores possíveis contidos no conjunto de A, principalmente se o conjunto for de baixa entropia e/ou pequena dimensão, que é o caso nos ficheiros analisados. As figuras 3, 4 e 5 correspondem ao *output* obtido na *party* B para cada *input* recebido.

```
cristina@Ubuntu22:~/PSI$ ./demo.exe -r 1 -p 0 -f input
Computation finished. Found 1 intersecting elements:
up201907321
```

Figura 3. Output da interseção utilizando o ficheiro "input" em ambas as parties

```
cristina@Ubuntu22:~/PSI$ ./demo.exe -r 1 -p 0 -f input2
Computation finished. Found 0 intersecting elements:
```

Figura 4. Output da interseção utilizando o ficheiro "input" na party A e "input2" na party B

```
cristina@Ubuntu22:~/TRPAssignment/PSI$ ./demo.exe -r 1 -p 0 -f AppList1.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 5. Output da interseção utilizando o ficheiro "AppList0.csv" na party A e "AppList1.csv" na party B, com aplicação do protocolo Naive Hashing

Tendo em conta o ficheiro *input* em A e B, através da captura de pacotes observada no *Wireshark*, visível nas figuras 7 e 8, é possível verificar o envio do *hash* na rede, nomeadamente nos dois pacotes anteriores à fase de terminação da ligação (denotada por [FIN, ACK]). Isto é inseguro, pois para além das *parties* poderem comprometer a sua privacidade entre elas através de *brute-force*, como a *hash* é enviada em *plaintext* pela rede, pode haver um ataque de *Eavesdropping* e pelo mesmo motivo da falta de confidencialidade do *hash*, o atacante fica a saber as interseções das duas *parties*.

```
claudia@claudia-IdeaPad-3-15ADA05:~/Desktop/Claudia/TRP/PSI_compiled/PSI$ echo -
n up201907321 | sha256sum | cut -c -12
0e4c114ba110
```

Figura 6. Hash de up201907321

```
    20 62.381009902  127.0.0.1  127.0.0.1  TCP  74 7766 → 42894 [PSH, ACK] Seq=21 Ack=21
    21 62.381373379  127.0.0.1  127.0.0.1  TCP  74 42894 → 7766 [PSH, ACK] Seq=21 Ack=27
    22 62.382125080  127.0.0.1  127.0.0.1  TCP  68 7766 → 42894 [FIN, ACK] Seq=27 Ack=27
    23 62.383711421  127.0.0.1  127.0.0.1  TCP  68 42894 → 7766 [FIN, ACK] Seq=27 Ack=28
    24 62.383741515  127.0.0.1  127.0.0.1  TCP  68 7766 → 42894 [ACK] Seq=28 Ack=28 Win=6

> Frame 20: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface any, id 0
> Linux cooked capture v1
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 7766, Dst Port: 42894, Seq: 21, Ack: 21, Len: 6
v Data (6 bytes)
    Data: 0e4c114ba110
    [Length: 6]
```

Figura 7. Hash observado na captura de pacotes na rede da porta 7766 para 42894

```
    20 62.381009902  127.0.0.1  127.0.0.1  TCP  74 7766 → 42894 [PSH, ACK] Seq=21 Ack=21
    21 62.381373379  127.0.0.1  127.0.0.1  TCP  74 42894 → 7766 [PSH, ACK] Seq=21 Ack=27
    22 62.382125080  127.0.0.1  127.0.0.1  TCP  68 7766 → 42894 [FIN, ACK] Seq=27 Ack=27
    23 62.383711421  127.0.0.1  127.0.0.1  TCP  68 42894 → 7766 [FIN, ACK] Seq=27 Ack=28
    24 62.383741515  127.0.0.1  127.0.0.1  TCP  68 7766 → 42894 [ACK] Seq=28 Ack=28 Win=6

> Frame 21: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface any, id 0
> Linux cooked capture v1
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 42894, Dst Port: 7766, Seq: 21, Ack: 27, Len: 6
v Data (6 bytes)
    Data: 0e4c114ba110
    [Length: 6]
```

Figura 8. Hash observado na captura de pacotes na rede da porta 42894 para 7766

2.2. Protocolo *Server-aided*

No protocolo *Server-aided*, é utilizado um servidor que funciona como *third-party* que realiza o cálculo da interseção entre os elementos das *parties* em questão. Como é visível na figura 9, o servidor é responsável por estabelecer a ligação com as duas *parties*, receber os seus *inputs*, calcular a interseção dos *inputs* para cada *party* e, por fim, aplicar funções *hash* aos elementos intersetados para os enviar para cada *party*.

```
cristina@Ubuntu22:~/TRPAssignment/PSI$ ./demo.exe -r 0 -p 1 -f README.md -a 127.0.0.1
Connections with all 2 clients established
Client 0 holds 36 elements of length 128-bit
Client 1 holds 36 elements of length 128-bit
Receiving the client's elements
Computing intersection for the clients
Inserting the items into the hash table
sending all 5 intersecting elements to the clients
```

Figura 9. Output obtido pelo pressuposto servidor que atua como third-party no cálculo da interseção

Verifica-se que o *output* nos terminais 1 e 2 (ou *parties* A e B) é, logicamente, igual ao obtido pelo protocolo *Naive Hashing*, uma vez que a única diferença é a máquina que realiza o cálculo da interseção.

```
cristina@Ubuntu22:~/TRPAssignment/PSI$ ./demo.exe -r 1 -p 1 -f AppList0.csv -a 127.0.0.1
Computation finished. Found 5 intersecting elements:
com.whatsapp
org.meowcat.edxposed.manager
com.google.android.apps.maps
com.android.chrome
com.delaware.empark
```

Figura 10. Output obtido pela party 1 com input: AppList0.csv

```
cristina@Ubuntu22:~/TRPAssignment/PSI$ ./demo.exe -r 1 -p 1 -f AppList1.csv -a 127.0.0.1
Computation finished. Found 5 intersecting elements:
com.whatsapp
org.meowcat.edxposed.manager
com.google.android.apps.maps
com.android.chrome
com.delaware.empark
```

Figura 11. Output obtido pela party 2 com input: AppList1.csv

Apesar deste protocolo ser mais eficiente do que o *Naive Hashing*, já que a computação é realizada pelo servidor, apresenta um risco de privacidade crítico caso o servidor seja atacado por um adversário, podendo originar ataques como *Eavesdropping*. Para além disto, tal como no protocolo *Naive Hashing*, podemos verificar pela figura 12 que as *hash* dos elementos que pertencem à interseção são enviados pela rede em *plaintext* para ambas as *parties*. Isto pode gerar ataques de *Eavesdropping* e mais uma vez o atacante tem acesso à interseção através de *brute force*.

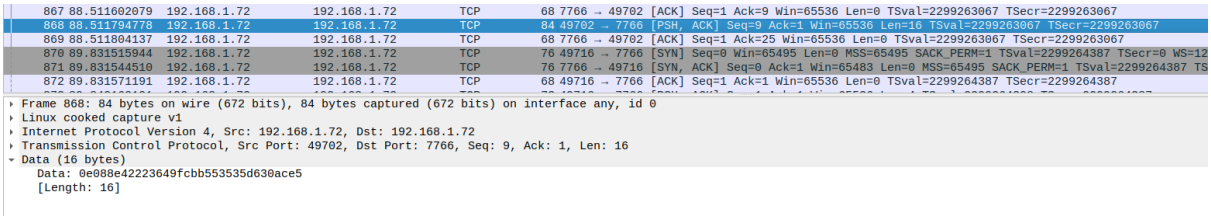

Figura 12. Captura de dados (hash) no Wireshark

2.3. Comparação estatística entre os protocolos

Para cada protocolo analisado, foram obtidas as estatísticas relativas à captura de pacotes, nomeadamente o número de pacotes e a quantidade de *bytes* capturados, tendo em conta o filtro *tcp.port == 7766*, como demonstrado na *Tabela 1*. Após a análise dos valores obtidos, é possível concluir que o protocolo *Naive Hashing* distingue-se em termos de eficiência na comunicação devido ao menor número de pacotes e *bytes* capturados. Apesar disso, não se verificam diferenças muito significativas quanto ao número de pacotes registados. Por outro lado, o protocolo *OT-based PSI* destaca-se pela elevada quantidade de *bytes* capturados, revelando-se, assim, menos eficiente em termos de comunicação. Desta forma, aquando da aplicação dos protocolos, torna-se importante analisar a relação segurança versus comunicação, isto é, averiguar se é compensatória a redução de eficiência de comunicação em prol de maior segurança, e vice-versa.

Protocolos	Número de pacotes capturados	Quantidade de *bytes* capturados
Naive Hashing Protocol	23	2110
Server-Aided Protocol	30	3258
Diffie-Hellman-based PSI Protocol	29	5844
OT-based PSI Protocol	31	54738

Tabela 1. Estatísticas observadas no Wireshark em relação aos pacotes e bytes capturados nos protocolos estudados

3. Benchmarking dos protocolos desenvolvidos para o PSI

O próximo passo consiste em realizar uma avaliação comparativa entre os três protocolos *Naive Hashing*, *Diffie-Hellman-based PSI* e *OT-based PSI*, em relação ao tempo de execução e à quantidade total de dados. A informação obtida para cada protocolo está representada nas tabelas 2, 3 e 4.

Set size	Required time (s)	Data sent (MB)	Data received (MB)
5000	0.01	0.04	0.04
7500	0.01	0.06	0.06
10000	0.02	0.09	0.09
12500	0.02	0.11	0.11
15000	0.03	0.13	0.13

Tabela 2. Tempo necessário de execução e dados permutados no Naive Hashing

Set size	Required time (s)	Data sent (MB)	Data received (MB)
5000	6.33	0.33	0.18
7500	9.79	0.49	0.26
10000	13.37	0.66	0.35
12500	17.82	0.82	0.44
15000	21.09	0.99	0.53

Tabela 3. Tempo necessário de execução e dados permutados no Diffie-Hellman PSI

Set size	Required time (s)	Data sent (MB)	Data received (MB)
5000	0.58	0.15	0.38
7500	0.62	0.21	0.56
10000	0.71	0.28	0.75
12500	0.56	0.34	0.94
15000	0.62	0.4	1.13

Tabela 4. Tempo necessário de execução e dados permutados no OT-based PSI

Para efeitos de comparação, foram criados dois gráficos: o primeiro (*Gráfico 1*) expressa a relação entre o tempo de execução (em segundos) e o tamanho do conjunto de dados a intersetar (*set size*), para cada protocolo; o segundo (*Gráfico 2*) relaciona a soma da quantidade de dados

enviados e a quantidade de dados recebidos (*total data exchanged*) com o *set size*, para cada protocolo.

Após análise do *Gráfico 1*, é possível verificar que, em relação aos outros dois protocolos, o *Diffie-Hellman-based PSI* requer muito mais tempo de execução para o processo de interseção. Já o *Naive Hashing* e o *OT-based PSI*, apresentam resultados muito similares, sendo bastante mais rápidos. Quanto ao *Gráfico 2*, observam-se resultados semelhantes para o *OT-based PSI* e *Diffie-Hellman-based PSI*, que, ao contrário do *Naive Hashing*, resultam numa maior quantidade de dados de enviados e recebidos. Assim, depois de comparar os resultados para os três protocolos, é possível concluir que, em termos de tempo de execução e quantidade de dados trocados, o *Naive Hashing* é o mais eficiente. Contrariamente, o protocolo *Diffie-Hellman-based PSI* apresenta-se como o menos eficiente. Em relação ao balanço entre segurança e custo da comunicação, os protocolos *Diffie-Hellman-based PSI* e *OT-based PSI* são os mais apropriados, já que são equivalentes nestes aspetos. No entanto, devido à falta de eficiência, consideramos que o protocolo *OT-based PSI* seja o mais indicado a usar, já que a segurança que oferece compensa os custos elevados de comunicação, e oferece uma eficiência equivalente à do *Naive Hashing*.

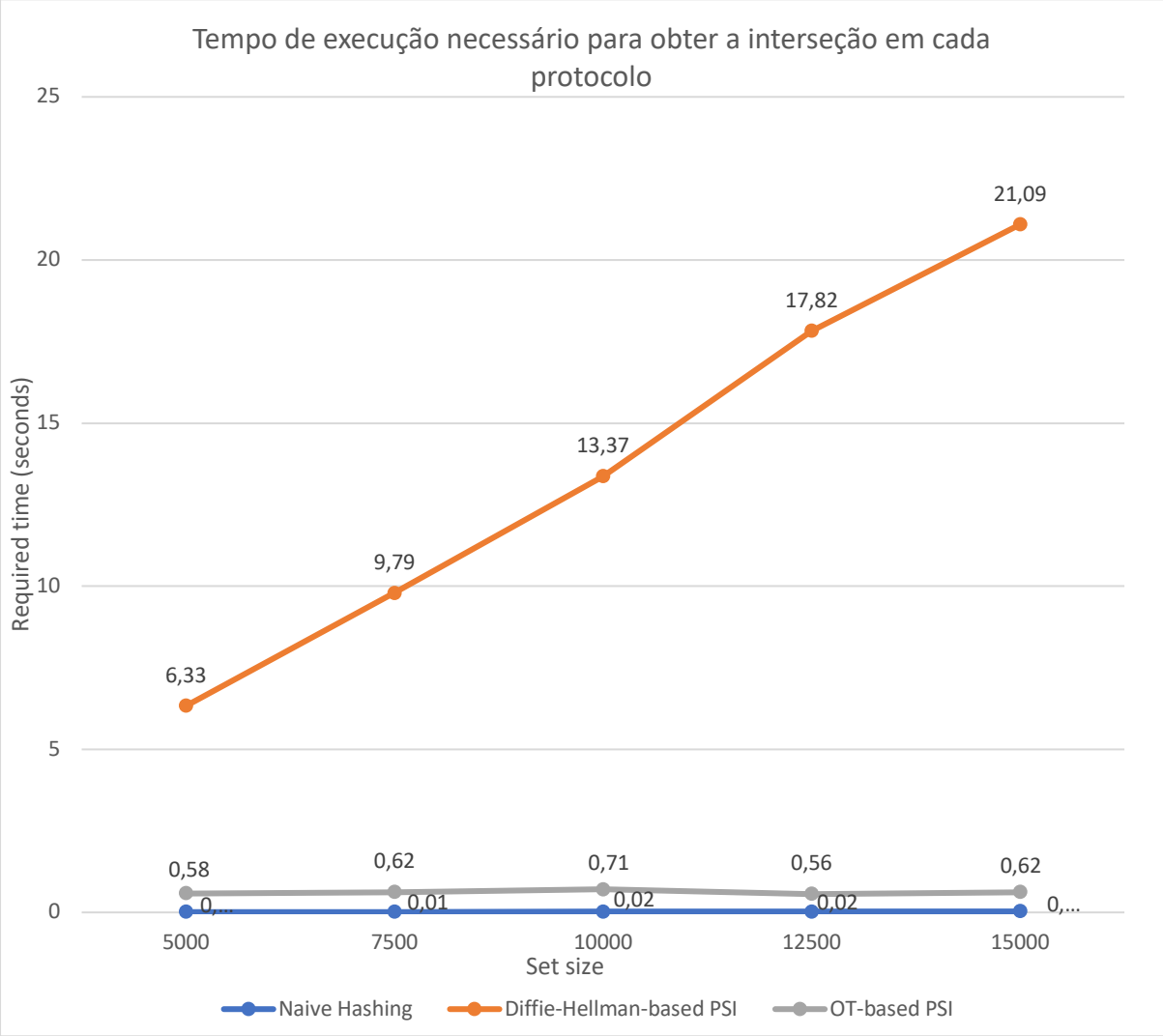

Gráfico 1. Tempos de execução dos protocolos

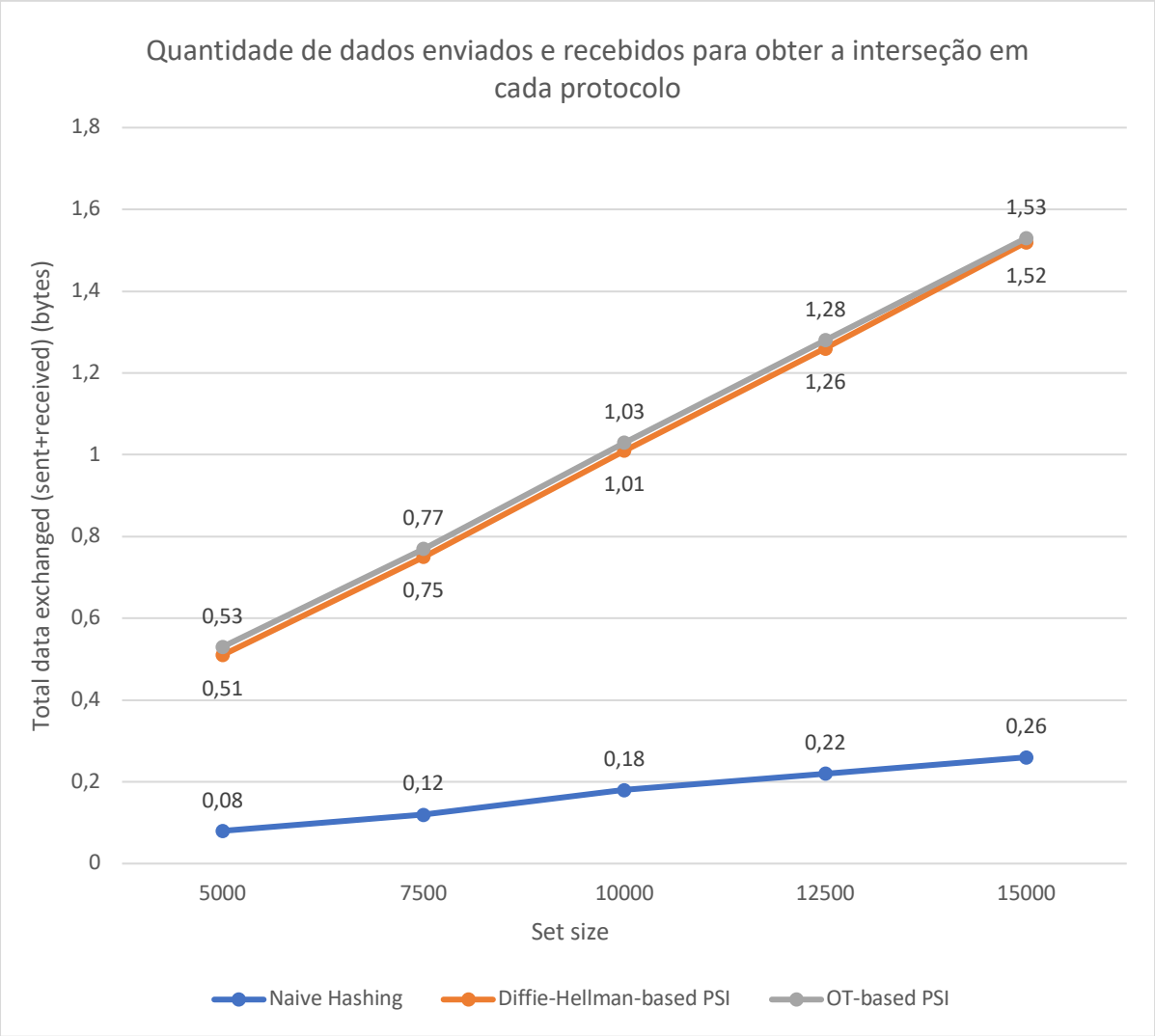

Gráfico 2. Dados transferidos pelas parties usando os protocolos PSI apresentados

4. Aplicação dos protocolos de interseção noutros *datasets*

Os *datasets* escolhidos são sobre *URL*s, sendo o primeiro um *dataset* de *URL*s [6] e a sua classificação como benignos ou maliciosos, e o segundo *dataset* de *URL*s maliciosos e de *phishing* [7]. O objetivo desta interseção é observar se os *datasets* acrescentam informação um ao outro, ou se os emails maliciosos são os mesmos, e o segundo *dataset* é uma partição do primeiro. Com base nos resultados, verificamos que efetivamente são dois datasets distintos, pois apenas encontramos algumas colisões, o que torna notório que acrescentam informação um ao outro. Começamos por remover os elementos duplicados do *dataset*, já que, caso contrário, não seria possível realizar a interseção para certos elementos no protocolo *OT-based PSI*. *Como* os *datasets* são muito extensos, de seguida reduzimos o número de elementos para 37 500 em cada um dos *inputs* para melhor análise dos resultados da interseção e, sendo assim o número de elementos o mesmo, ser possível aplicar o protocolo *Server-aided*. Para além disso, retiramos as colunas não referentes aos *URL*s, uma vez que apenas queríamos intersetar estes mesmos.

Com base nos resultados obtidos nas experimentações com os 4 protocolos, decidimos considerar o protocolo *OT-based PSI*, uma vez que, embora sejam capturados mais *bytes*, ou seja, o total de dados transferidos seja elevado, tal como o protocolo *Diffie-Hellman-based PSI*, o tempo de execução é equivalente ao protocolo de *Naive Hashing*. Tendo isto em conta, não utilizamos o *Naive Hashing* por ser inseguro para as *parties* e não garantir confidencialidade. Para além disto, não utilizamos o protocolo *Server-aided*, uma vez que este demonstra também ser inseguro como o *Naive Hashing*, se o servidor for atacado, utiliza mais recursos e ainda requer que ambos os *inputs* tenham o mesmo número de elementos.

Embora tenhamos apresentado o resultado do protocolo *OT-based PSI* na figura 13, certificamo-nos que obtemos os mesmos resultados (35 elementos intersetados) com os restantes protocolos, embora não sejam seguros e tão eficientes.

```
cristina@Ubuntu22:~/TRPAssignment/PSI$ ./demo.exe -r 1 -p 3 -f url2.csv
Hashing 37500 elements with arbitrary length into into 9 bytes
Client: bins = 45000, elebitlen = 57 and maskbitlen = 72 and performs 45000 OTs
Computation finished. Found 35 intersecting elements:
10starmovies.com/Watch-Movies-Online/Inside_Im_Dancing_2004/
10starmovies.com/Watch-Movies-Online/Jackie_Bouvier_Kennedy_Onassis_2000/
10starmovies.com/profiles/Peter_Benson_98030/
1337x.org/torrent/191829/Card-Captor-Sakura-OST-1-4-Movie-1-2-OST-Lossy-Mp3-128-320kbps-Tntvillage/
14for77.blogspot.com/2011/03/2011-kansas-city-royals-prediction.html
18gay.net/porn/gay/films/
1964topps.wordpress.com/2010/08/07/452-giants-rookie-stars-gil-garridojim-ray-hart/
1967topps.blogspot.com/
1972topps.blogspot.com/2009/03/77-ron-theobald.html
1980toppsbaseball.blogspot.com/2009/12/210-steve-carlton.html
199.115.25.156/aqueduct/Stakes/HollieHughes.shtml
1m1f.com/Reticuli/
1upbooks.com/
1upclan.info/
1uprc.com/
2002.timbersfanpage.com/
20050-wwsr.info/
20poundsofheadlines.wordpress.com/2011/11/07/kc-mayor-sly-james-bittersweet-reaction-to-mizzou/
216.92.161.171/vb/printthread.php?t=8432&pp=40&page=2
21stbattalion.ca/graves_t-z.html
23rdstorey.blogspot.com/
26th.theabcsofdeath.com/t-is-for-taste
2photographersandadog.com/ourminds/
31hours.com/live-streaming-shimizu-s-pulse-urawa-red-diamonds-tv-watch-17-09-2011/
399bellevue.com/
3alitydigital.wordpress.com/
3blmedia.com/theCSRfeed/Molson-Canadian-Red-Leaf-Project-Celebrates-Grey-Cup-Vancouver
419.bittenus.com/11/4/BelhassenTrabelsi.html
4261detroit.com/
42monticelloavenue.com/
44thandgoal.blogspot.com/
4508c35cb.blogspot.com/2009_01_01_archive.html
46eqez.blogspot.com/2011/05/charlene-tilton-now.html
4americanfootball.blogspot.com/2008/10/list-of-american-football-teams-in.html
4megaupload.com/ageless-desire-juliet-anderson.html
```

Figura 13. Output resultante da interseção dos dois datasets escolhidos

5. Conclusão

Em tom de conclusão, foi realizado o estudo e análise de resultados e estatísticas de quatro protocolos desenvolvidos em âmbito do PSI. Podemos verificar que o protocolo *OT-based* PSI é aquele que faz o melhor balanço entre o custo de comunicações, eficiência e segurança, uma vez que embora não seja tão rápido como o *Naive Hashing* e seja o protocolo com maior transferência de informação, este garante a privacidade que o protocolo *Naive Hashing* e o *Server-Aided* não garantem. Para além disso, ao contrário do *Diffie-Hellman-based* PSI, é muito mais eficiente em termos de complexidade temporal.

Referências

[1] "What are the advantages of mutual private set intersection methods over finding the intersection of hashed lists?", StackExchange. Disponível em: https://crypto.stackexchange.com/questions/65908/what-are-the-advantages-of-mutual-private-set-intersection-methods-over-finding

[2] B. Pinkas, T. Schneider, M. Zohner, G. Segev. Phasing: Private Set Intersection using Permutation-based Hashing. Disponível em: https://eprint.iacr.org/2015/634.pdf

[3] S. Kamara, P. Mohassel, M. Raykova, and S. Sadeghian. Scaling private set intersection to billion-element sets. In Financial Cryptography and Data Security (FC'14) , LNCS. Springer, 2014.

[4] "Private Set Intersection with Diffie-Hellman", OpenMined. Disponível em: https://blog.openmined.org/private-set-intersection-with-diffie-hellman/

[5] "Faster Private Set Intersection Based on OT Extension", usenix. Disponível em: https://www.usenix.org/node/184446

[6] "Url Dataset", kaggle. Disponível em: https://www.kaggle.com/datasets/teseract/urldataset

[7] "Malicious URLs dataset", kaggle. Disponível em: https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset