

Secure Multiparty Computation
para Privacidade na Prática

Tecnologias de Reforço da Privacidade

Grupo C

Manuel Ramos Leite Carvalho Neto – up202108744

Ricardo António Araújo Martins – up202408272

Mestrado em Segurança Informática

2024/2025

Índice

Introdução.....	3
Protocolos para a Interseção de Conjuntos Privados: Estudo, Descrição e Comparação.....	4
Protocolo de <i>Naïve Hashing</i>	5
Protocolo Apoiado pelo Servidor.....	6
Protocolo Baseado em Diffie-Hellman	9
Protocolo Baseado em <i>Oblivious Transfer</i>	12
Comparação	14
Experiência com Protocolos para a Interseção de Conjuntos Privados.....	16
Passo 1	16
Passo 2	17
Passo 3	17
Passo 4	18
Passo 5	19
Passo 6	21
Passo 7	22
Passo 8	23
Passo 9	24
Passo 10	25
<i>Benchmarking</i> de Protocolos para a Interseção de Conjuntos Privados	27
Análise Crítica	31
Aplicação de Protocolos de Interseção Segura a Outro <i>Dataset</i>	32
Conclusão	35

Índice de Figuras

Figura 1 – Passo 1	16
Figura 2 – Passo 2	17
Figura 3 – Passo 3 (Passo 1)	17
Figura 4 – Passo 3 (Passo 2)	18
Figura 5 – Passo 5	20
Figura 6 – Passo 6	21
Figura 7 – Passo 8	23
Figura 8 – Passo 9	24
Figura 9 – Tempo de Execução dos Protocolos	28
Figura 10 – Dados Trocados pelos Protocolos	30
Figura 11 – Datasets para Aplicação	32
Figura 12 – Protocolo Apoiado pelo Servidor	33
Figura 13 – Protocolo Baseado em Oblivious Transfer	34

Índice de Tabelas

Tabela 1 – Passo 5	20
Tabela 2 – Passo 6	22
Tabela 3 – Passo 8	24
Tabela 4 – Passo 9	24
Tabela 5 – Passo 10	25
Tabela 6 – Tempo de Execução (s)	27
Tabela 7 – Dados Trocados (MB)	29

Introdução

No âmbito da Unidade Curricular Tecnologias de Reforço da Privacidade, propõe-se o desenvolvimento de um projeto que consiste na exploração de algumas técnicas de *Secure Multiparty Computation* para privacidade na prática. Nesse sentido, este relatório visa documentar, analisar e justificar todas as etapas integrantes desta exploração, dividindo-a em quatro fases.

Em primeiro lugar, estudam-se, descrevem-se e comparam-se quatro protocolos diferentes para a interseção de conjuntos privados. Os protocolos em causa são: (1) uma solução de *naïve hashing* na qual é comparada a *hash* de cada valor, (2) um protocolo com apoio de um servidor, (3) um protocolo baseado em Diffie-Hellman e (4) um protocolo assente no conceito de *oblivious transfer*. Estes quatro protocolos são devidamente analisados e explicados.

Em segundo lugar, seguem-se alguns passos para experimentar a execução destes protocolos em conjuntos de dados reais, tendo em vista uma melhor compreensão do funcionamento dos mesmos. Em particular, realiza-se uma análise crítica sobre (1) porque é que o protocolo de *naïve hashing* pode ser inseguro, (2) que problemas de privacidade surgem devido à utilização de um servidor como terceira parte e (3) a relação entre o custo de comunicação e a privacidade/segurança de cada protocolo. Deste modo, aprofunda-se o conhecimento sobre os objetos de estudo.

Em terceiro lugar, compara-se o desempenho dos protocolos relativamente ao tempo de execução e aos dados trocados entre as partes envolvidas. Para tal, apresentam-se as tabelas e os gráficos necessários, de maneira a efetuar uma comparação adequada e devidamente fundamentada.

Por último, aplicam-se os protocolos estudados a um conjunto de dados específico, de modo a avaliar a execução dos mesmos na prática, através da análise de um caso real.

Em todos os momentos, procura-se proceder a uma análise crítica dos diferentes aspetos específicos de cada protocolo, bem como justificar todas as decisões tomadas tanto quanto possível. Para o efeito, utilizam-se algumas ferramentas de *software* posteriormente introduzidas, que atuam como elementos essenciais para a aplicação dos protocolos.

Em suma, este relatório pretende incluir uma descrição e comparação dos quatro protocolos de interseção de conjuntos privados, uma análise crítica sobre determinados aspetos específicos dos mesmos, em particular o seu desempenho, juntamente com uma aplicação dos protocolos em contexto real.

Protocolos para a Interseção de Conjuntos Privados: Estudo, Descrição e Comparação

O conceito *Secure Multiparty Computation* transmite a ideia de que, para um conjunto de partes/entidades com os seus respetivos *inputs* privados, existe alguma computação que todas as partes querem realizar, mas para a qual são necessários os dados privados de cada parte, pelo que a sua realização está condicionada à satisfação de determinadas garantias de privacidade/segurança da informação. Nesse sentido, existem múltiplas aplicações para as quais isto é central, sendo uma delas a interseção de conjuntos privados.

A interseção de conjuntos privados é um problema que passa por determinar a interseção, ou seja, os elementos comuns, entre um dado número de conjuntos de valores pertencentes a cada entidade, sem partilhar os conjuntos entre as partes envolvidas. Assim, pretende-se que cada parte seja capaz de conhecer os elementos que existem, simultaneamente, no seu conjunto e nos restantes, mas que nenhuma das partes seja capaz de conhecer qualquer elemento para além dos comuns e dos presentes no seu próprio conjunto.

A título de exemplo, pode observar-se o caso típico de diferentes agências de inteligência, em que cada uma detém uma lista de potenciais terroristas. O objetivo comum entre as agências é computar/determinar os indivíduos que pertencem a todas as listas, sem revelar para as restantes entidades os indivíduos conhecidos por cada agência. De forma mais simplista, isto é semelhante a determinar quais os contactos telefónicos comuns entre duas pessoas, sem partilhar a totalidade da lista de contactos, ou, analogamente, identificar as aplicações móveis comuns em dois telemóveis, sem dar a conhecer as restantes aplicações presentes em cada telemóvel às partes envolvidas.

Os quatro protocolos principais para a resolução deste problema são:

1. Protocolo de *Naïve Hashing*;
2. Protocolo Apoiado pelo Servidor;
3. Protocolo Baseado em Diffie-Hellman;
4. Protocolo Baseado em *Oblivious Transfer*.

Cada um destes protocolos funciona de forma diferente, tendo as respetivas vantagens, desvantagens e garantias de segurança/privacidade. Como tal, pretende-se estudar, descrever e comparar os quatro protocolos, nomeadamente o seu funcionamento e as suas características/propriedades.

Por simplificação, explica-se cada protocolo considerando apenas duas partes (Alice e Bob), notando que o procedimento é facilmente generalizável para mais entidades, em todos os casos. Considere-se que a Alice contém o conjunto $A = \{a_1, \dots, a_n\}$ e que o Bob contém o conjunto $B = \{b_1, \dots, b_m\}$.

Protocolo de *Naïve Hashing*

O protocolo de *naïve hashing* é o mais simples dos quatro protocolos em estudo, sendo, igualmente, o que oferece menos garantias de segurança e privacidade.

Neste protocolo, ambas as partes devem concordar previamente numa função de *hash* a utilizar. Após isto, cada parte é responsável por aplicar essa função a cada um dos elementos do seu conjunto e enviar o resultado obtido para cada elemento para a outra parte. Assim que cada parte receber os valores de *hash* enviados pela outra parte, basta comparar os valores recebidos com os valores computados para o seu próprio conjunto, identificando as *hashes* que coincidem, isto é, que foram enviadas e recebidas. Os elementos correspondentes a cada uma destas *hashes* coincidentes formam a interseção entre os conjuntos privados. Note-se que não é necessário (nem possível, idealmente) inverter a computação da função de *hash* – e que as garantias de segurança deste protocolo assentam precisamente neste facto –, visto que cada parte computa as *hashes* para os seus próprios valores e compara-as com as *hashes* recebidas, pelo que cada parte conhece o seu elemento que resultou na respetiva *hash*.

Exemplifique-se o funcionamento deste protocolo com a Alice e o Bob:

1. A Alice e o Bob concordam na utilização de uma função de *hash* $H(x)$;
2. A Alice computa $A' = \{H(a_1), \dots, H(a_n)\}$ e envia o resultado para o Bob;
3. O Bob computa $B' = \{H(b_1), \dots, H(b_m)\}$ e envia o resultado para a Alice;
4. A Alice e o Bob comparam A' com B' , sendo que $H(a_i) = H(b_j) \rightarrow a_i = b_j$, pelo que a interseção entre os conjuntos é dada pelo conjunto de todos estes valores $a_i = b_j$.

Evidentemente, este protocolo é extremamente simples, o que facilita a sua implementação. Além disto, tendo em conta que só é necessário aplicar a função de *hash* e trocar os valores obtidos, a execução deste protocolo é computacionalmente leve, logo, extremamente rápida.

No entanto, toda esta simplicidade acarreta algumas limitações.

Em primeiro lugar, para aplicações deste algoritmo em domínios restritos ou limitados, pode ser possível realizar um ataque de dicionário contra as *hashes* recebidas, possibilitando a descoberta dos *inputs* originais. Por exemplo, se forem partilhados contactos telefónicos portugueses por este meio, sendo o espaço de pesquisa relativamente pequeno em termos computacionais – inferior a 10^9 –, torna-se perfeitamente viável computar o resultado da função de *hash* para todos os números de telefone possíveis, conseguindo, assim, identificar os valores originais correspondentes às *hashes* trocadas, o que configura uma falha de privacidade/segurança extremamente grave.

Em segundo lugar, note-se que a correção de todo o funcionamento e a segurança/privacidade deste protocolo residem na função de *hash* utilizada. Por um lado, se esta função não for resistente a colisões, o funcionamento do protocolo fica comprometido, visto que pode dar-se o caso de $H(a_i) = H(b_j)$, mas $a_i \neq b_j$, pelo que o resultado computado por ambas as partes estará incorreto. Por outro lado, se a função não for resistente a pré-imagens, torna-se possível computar x a partir de $H(x)$, pelo que uma parte maliciosa pode computar todos os *inputs* privados da outra parte a partir das *hashes* recebidas, quebrando a segurança potencialmente oferecida pelo protocolo. Por esta razão, no caso de se aplicar este algoritmo na prática, é essencial utilizar funções de *hash* criptográficas seguras, como SHA-256, evitando funções de propósito não criptográfico – como MD5 – ou já quebradas, como SHA1.

Por último, no caso de uma das partes ser maliciosa ou desonesta, pode não enviar para a outra os valores corretos correspondentes às *hashes* dos elementos do seu conjunto. Por exemplo, um adversário pode não enviar a *hash* de todos os seus valores, ou enviar *hashes* incorretas, ou não enviar qualquer *hash*, entre outras possibilidades. Deste modo, ao receber os valores provenientes da outra parte, o ator malicioso consegue na mesma computar a interseção dos conjuntos ao calcular as *hashes* dos seus valores, mas sem as partilhar, adquirindo mais conhecimento do que a outra parte. Portanto, este procedimento não oferece garantias de segurança, nem no modelo semi-honesto (no qual alguma parte segue o protocolo, mas tenta inferir informação privada), nem no modelo malicioso (em que pelo menos uma parte se desvia ativamente do cumprimento do protocolo), visto que qualquer parte com más intenções pode subverter o funcionamento esperado do mesmo.

Assim sendo, este protocolo apresenta mais desvantagens do que vantagens, pelo que apenas pode ser considerado adequado para cenários nos quais os dados não são sensíveis e a segurança/privacidade dos mesmos não é muito importante, o que não tende a ser o caso nos problemas de *Secure Multiparty Computation*. A velocidade e simplicidade deste algoritmo não são suficientes para sustentar a sua utilização num contexto real.

Protocolo Apoiado pelo Servidor

O protocolo apoiado pelo servidor, tal como apresentado por Seny Kamara (*Microsoft Research*), Payman Mohassel (*University of Calgary*), Mariana Raykova (SRI) e Saeed Sadeghian (*University of Calgary*) no artigo intitulado “*Scaling Private Set Intersection to Billion-Element Sets*”, publicado em 2014, transfere os custos da computação para um servidor externo, que funciona como terceira parte. O artigo expõe também algumas extensões ao protocolo original, que oferecem mais garantias de segurança em vários modelos de ataque.

Segundo este protocolo, ambas as partes têm de concordar numa permutação pseudoaleatória (PRP), que será aplicada aos dados. Esta permutação deve ter uma chave associada, partilhada entre as partes. Cada parte deve computar o valor resultante da aplicação da PRP a cada elemento do seu conjunto e permutar os resultados segundo uma permutação aleatória. Os resultados desta permutação são, então, partilhados com o servidor, que calcula a interseção entre todos os valores recebidos, devolvendo-a para todas as partes. Assim, cada parte pode inverter localmente a PRP anteriormente aplicada, obtendo os valores que fazem parte da interseção dos conjuntos.

A título de exemplo, demonstre-se o funcionamento deste protocolo com a Alice e o Bob:

1. A Alice e o Bob concordam na utilização de uma PRP F com chave K, F_K ;
2. A Alice computa $A' = \pi_{Alice}(F_K(A))$, sendo π_{Alice} uma permutação aleatória, e envia o resultado para o servidor;
3. O Bob computa $B' = \pi_{Bob}(F_K(B))$, sendo π_{Bob} uma permutação aleatória, e envia o resultado para o servidor;
4. O servidor computa $I = A' \cap B'$ e envia o resultado para a Alice e o Bob;
5. A Alice e o Bob computam $F_K^{-1}(I)$, obtendo a interseção entre os conjuntos privados.

Ora, este protocolo transfere os custos computacionais do processamento para o servidor, mas utiliza a permutação pseudoaleatória com chave de modo a assegurar que o servidor não é capaz de conhecer os valores partilhados. Com isto, garante-se a segurança da computação mesmo que o servidor seja semi-honesto, isto é, cumpra o protocolo, mas procure inferir conhecimento sobre a informação privada, visto que a PRP com chave impede isto mesmo, ao tornar cada elemento indistinguível de um valor aleatório para qualquer observador que não detenha a chave. Igualmente, o protocolo também garante segurança se o servidor for honesto e as partes maliciosas, dado que as partes nunca trocam mensagens entre si, sendo que o único comportamento malicioso possível consiste em alterar o resultado de aplicação da PRP aos próprios dados o que, na prática, é equivalente a alterar diretamente os próprios dados. Note-se que a assunção de segurança essencial ao protocolo é o facto de o servidor não colaborar de forma maliciosa ou indevida com alguma das partes, o que quebraria toda a segurança do mesmo. No entanto, esta assunção pode ser considerada admissível, visto que o servidor deve ser concordado/escolhido por ambas as partes.

Contudo, caso o servidor seja malicioso, pode devolver uma interseção errada para as partes envolvidas. Este comportamento pode ser prevenido ao estender o protocolo original para que cada parte replique/copie cada elemento do seu conjunto λ vezes, sendo $\lambda > 1$ um parâmetro concordado previamente entre as partes. Deste modo, dificulta-se que o servidor possa mentir na interseção retornada, visto que teria de adivinhar corretamente quais os elementos repetidos, de maneira a incluí-los ou omiti-los a todos.

A par disto, para impedir que o servidor possa retornar indevidamente como interseção o conjunto vazio ou o conjunto de todos os elementos enviados por cada parte, esta mesma extensão do protocolo propõe que as duas partes concordem em três conjuntos D_0 , D_1 e D_2 , de igual cardinalidade, para que sejam adicionados ao conjunto a partilhar com o servidor. Assim, cada parte deve partilhar o seu conjunto de *input*, juntamente com D_0 e com D_1 ou D_2 , devendo D_1 ser utilizado por uma parte e D_2 pela outra parte. A título de exemplo, no caso da Alice e do Bob, a Alice deveria partilhar com o servidor o conjunto resultante de $\pi_{Alice}(F_K(A^\lambda + D_0 + D_{Alice}))$, enquanto o Bob deveria enviar $\pi_{Bob}(F_K(B^\lambda + D_0 + D_{Bob}))$, sendo λ , D_0 , D_{Alice} e D_{Bob} estabelecidos previamente entre ambos e onde X^λ representa o conjunto que contém λ vezes cada elemento original do conjunto X . Deste modo, qualquer parte consegue perceber que a interseção não está correta se não contiver D_0 , ou se contiver algum valor de D_1/D_{Alice} ou de D_2/D_{Bob} , ou se não existirem λ cópias de um elemento no conjunto resultante. Note-se que, neste caso, o conjunto D_0 deve, evidentemente, ser posteriormente excluído do conjunto resultante da interseção, sendo este processamento feito por cada parte.

O artigo propõe ainda mais duas extensões ao protocolo original. Em primeiro lugar, procura-se impedir que uma parte maliciosa consiga submeter um *input* incorretamente estruturado que leve a outra parte a abortar, enquanto a parte maliciosa consegue absorver a interseção correta dos conjuntos privados. Neste sentido, para garantir justiça, basta que o servidor se comprometa criptograficamente com o resultado antes de o revelar às partes, impedindo alterações posteriores. Deve ainda ser aplicada uma camada extra da permutação pseudoaleatória por cada parte, para permitir a deteção deste comportamento indevido da outra parte. Em segundo lugar, caso seja necessário ocultar do servidor o tamanho do conjunto resultante da interseção dos conjuntos privados, sugere-se que o servidor não compute diretamente a interseção, mas apenas auxilie uma das partes com a computação. Para tal, é necessário que o servidor e a outra parte partilhem também uma chave a aplicar numa nova permutação pseudoaleatória, que visa garantir que nenhuma das partes seja capaz de impedir a outra parte de conhecer a interseção dos conjuntos privados.

Tendo em conta que o objeto de estudo incide principalmente sobre o protocolo original, este deve ser alvo de uma análise mais aprofundada quanto às suas vantagens e desvantagens.

Por um lado, ao transferir o principal custo computacional – a interseção de conjuntos – para uma terceira parte – o servidor –, este protocolo mantém-se eficiente e escalável, tal como no caso anterior. A par disto, continua a ser um protocolo de implementação simples e que permite satisfazer algumas garantias de segurança e privacidade, nomeadamente no que concerne ao facto de o servidor não ser capaz de conhecer os dados originais dos clientes, assim como dificulta a fraude de uma das partes, desde que se respeite a assunção de que o servidor não colabora indevidamente com nenhuma das entidades.

Por outro lado, o protocolo apoiado pelo servidor pressupõe o cumprimento de mais assunções de segurança/privacidade do que o caso anterior, como a já explicada. Além disto, na sua versão base, surgem todos os problemas que as extensões procuram resolver, particularmente na situação de o servidor ser malicioso ou na eventualidade de alguma das partes impedir a outra parte de obter o resultado correto. Existe, ainda, o problema de que as partes não têm qualquer meio para confirmar que a interseção obtida está correta, recaindo isto na confiança de que o servidor se comporta de forma legítima. Finalmente, o protocolo exige também a partilha prévia de, pelo menos, uma chave criptográfica entre as partes, o que pode ser entendido como uma ligeira limitação.

Em suma, esta abordagem permite computar a interseção entre conjuntos privados de forma mais segura do que no caso anterior e quase tão eficiente, ao transferir os custos computacionais e as assunções de segurança para uma terceira parte. Contudo, a versão base do protocolo tem ainda algumas limitações, que tentam ser resolvidas pelas diferentes extensões e que podem constituir soluções eficazes em determinados contextos, embora impliquem a abdicação de alguma simplicidade do protocolo original. Por isso, este protocolo pode ser viável na prática, principalmente em aplicações em grande escala, desde que o modelo de ameaça e as assunções de confiança o permitam.

Protocolo Baseado em Diffie-Hellman

O protocolo baseado em Diffie-Hellman, conforme explicado por Catherine Meadows (*Naval Research Laboratory*) no artigo “*A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party*”, de 2014, é construído por cima do protocolo criptográfico Diffie-Hellman, que assenta no problema do logaritmo discreto.

O problema do logaritmo discreto reside na dificuldade computacional de, num dado espaço modular, encontrar um valor y que satisfaça $g^y \equiv x \pmod{p}$, sendo p um número primo e g um gerador no campo finito \mathbb{Z}_p^* . Este problema é considerado computacionalmente difícil/complexo, pelo que serve como assunção basilar para alguns protocolos criptográficos.

O protocolo Diffie-Hellman original pretende estabelecer um meio para duas partes estabelecerem um valor partilhado de forma segura, sendo tradicionalmente utilizado para a definição de uma chave criptográfica. Para tal, o procedimento a seguir pela Alice e pelo Bob deve ser o seguinte:

1. A Alice e o Bob concordam num grupo multiplicativo público \mathbb{Z}_p^* , ou seja, concordam num número primo p e numa base $g \in \mathbb{Z}_p^*$;
2. A Alice e o Bob geram chaves privadas aleatórias: $a \in \mathbb{Z}_p^*$ e $b \in \mathbb{Z}_p^*$, respetivamente;

3. A Alice e o Bob geram chaves públicas: $A = g^a \text{ mod } p$ e $B = g^b \text{ mod } p$, respetivamente;
4. A Alice e o Bob trocam as respetivas chaves públicas através de um canal público desprotegido;
5. A Alice computa $K \equiv B^a \equiv (g^b)^a \equiv g^{ba} \equiv g^{ab} \text{ (mod } p)$ e o Bob computa $K \equiv A^b \equiv (g^a)^b \equiv g^{ab} \text{ (mod } p)$, estabelecendo um segredo partilhado.

Assim, o protocolo para a interseção de conjuntos privados é baseado neste. Em particular, o protocolo explicado no artigo pretende resolver o problema de autenticar partes que suspeitam uma da outra, ao permitir que cada utilizador verifique se as suas credenciais correspondem às credenciais recebidas, sem as revelar. Deste modo, o problema é análogo à interseção de conjuntos privados.

O protocolo para a interseção de conjuntos privados é dividido em duas fases. Na primeira fase, cada parte regista-se no sistema e recebe, de uma terceira parte: um número inteiro N , um número primo p , um segredo encriptado S , os números inteiros M e D , bem como um tuplo assinado digitalmente pela terceira parte. Os valores de N e p são públicos e conhecidos por todas as entidades, enquanto os restantes valores são privados. Na segunda fase, as partes começam por se autenticar, trocando os seus tuplos assinados recebidos anteriormente e verificando as respetivas assinaturas. Após isto, computam um segredo partilhado que funcionará como chave de encriptação. Assim, cada parte envia para a outra uma mensagem encriptada com um tuplo que permite à entidade recetora verificar se os segredos correspondem/coincidem, sem revelar qualquer informação sobre o seu segredo.

Para maior clareza, detalhe-se este protocolo através da Alice e do Bob:

1. A Alice recebe de uma terceira parte os valores iniciais: $N, p, S_{Alice}, M_{Alice}, D_{Alice}$ e o tuplo $T_{Alice} = (Alice, O_{Alice}, S_{Alice}^{M_{Alice}}, X^{M_{Alice}}, (S_{Alice}^{M_{Alice}} X^{M_{Alice}})^{D_{Alice}})$, devidamente assinado pela terceira parte, no espaço modular de N ;
2. O Bob recebe de uma terceira parte os valores iniciais: $N, p, S_{Bob}, M_{Bob}, D_{Bob}$ e o tuplo $T_{Bob} = (Bob, O_{Bob}, S_{Bob}^{M_{Bob}}, X^{M_{Bob}}, (S_{Bob}^{M_{Bob}} X^{M_{Bob}})^{D_{Bob}})$, devidamente assinado pela terceira parte, no espaço modular de N ;
3. A Alice e o Bob trocam os seus tuplos e verificam as respetivas assinaturas;
4. A Alice computa $K = (X^{M_{Bob}})^{M_{Alice}}$ e o Bob computa $K = (X^{M_{Alice}})^{M_{Bob}}$, estabelecendo um segredo partilhado;
5. A Alice envia para o Bob $E_K(S_{Bob}^{M_{Bob}M_{Alice}})$ e $E_K((S_{Bob}^{M_{Bob}} X^{M_{Bob}})^{D_{Bob}M_{Alice}})$, sendo que E_K significa encriptar com a chave K ;
6. O Bob computa $(S_{Bob}^{M_{Bob}M_{Alice}} \times X^{M_{Bob}M_{Alice}})^{D_{Bob}} = (S_{Bob}^{M_{Bob}} X^{M_{Bob}})^{D_{Bob}M_{Alice}}$, verificando se este resultado é igual à descriptação do segundo valor enviado pela Alice e se $(S_{Bob}^{M_{Bob}M_{Alice}})^p \equiv 1 \text{ (mod } N)$;
7. O Bob envia para a Alice $E_K(S_{Alice}^{M_{Alice}M_{Bob}})$ e $E_K((S_{Alice}^{M_{Alice}} X^{M_{Alice}})^{D_{Alice}M_{Bob}})$, sendo que E_K significa encriptar com a chave K ;

8. A Alice computa $(S_{Alice}^{M_{Alice}M_{Bob}} \times X^{M_{Alice}M_{Bob}})^{D_{Alice}} = (S_{Alice}^{M_{Alice}} X^{M_{Alice}})^{D_{Alice}M_{Bob}}$, verificando se este resultado é igual à descriptação do segundo valor enviado pelo Bob e se $(S_{Alice}^{M_{Alice}M_{Bob}})^p \equiv 1 \pmod{N}$;
9. A Alice e o Bob verificam se $S_{Alice}^{M_{Alice}M_{Bob}} = S_{Bob}^{M_{Bob}M_{Alice}}$, o que, se sim, significa que $S_{Alice} = S_{Bob}$, pelo que este valor pertence à interseção dos conjuntos privados.

Efetivamente, esta interação permite que ambas as partes consigam saber quais os valores do seu conjunto privado que também pertencem ao conjunto privado da outra parte, sem revelar quais são os seus valores. Esta garantia de segurança reside na dificuldade computacional do problema do logaritmo discreto já utilizado no contexto do protocolo Diffie-Hellman original que, neste caso, impede o cálculo de S_{Bob} a partir de $S_{Bob}^{M_{Bob}M_{Alice}}$ e de S_{Alice} a partir de $S_{Alice}^{M_{Alice}M_{Bob}}$, a não ser que $S_{Alice} = S_{Bob}$. Acresce ainda a vantagem de que a existência de uma terceira parte só é necessária na fase inicial da execução do protocolo, servindo apenas para atestar a autenticidade de cada participante.

Uma primeira interpretação deste protocolo pode levar a crer que a assimetria no processo de troca de informação permite que a segunda entidade interveniente – o Bob –, depois de receber o segredo encriptado da primeira entidade – a Alice – e de verificar a sua autenticidade, seja capaz de verificar se o elemento enviado pertence à interseção dos conjuntos privados, sem ser obrigado a divulgar a informação que permite à outra parte (Alice) realizar esta mesma computação. Deste modo, uma parte maliciosa poderia receber toda a informação da outra parte, mas depois enviar dados incorretos ou apenas parcialmente corretos, quebrando a reciprocidade do protocolo, tal como sucedia na solução de *naïve hashing*. Todavia, um dos pontos fortes deste protocolo é precisamente a robustez contra este facto. O artigo apresenta um argumento heurístico que sustenta isto mesmo, baseando-se no facto de que as verificações efetuadas nos passos (6) e (8) garantem a simetria/reciprocidade da troca de informação, visto que a possibilidade de alguma parte maliciosa encontrar e enviar algum valor que satisfizesse essas verificações sem ser o valor correto implicaria que essa parte tivesse sido capaz de quebrar o sistema criptográfico RSA, o que não se considera computacionalmente possível, atualmente, para bons valores dos parâmetros públicos estabelecidos. Assim sendo, prova-se a segurança do protocolo no modelo semi-honesto, ou seja, desde que as partes cumpram o procedimento estabelecido.

Contudo, uma parte maliciosa pode não enviar quaisquer valores, impedindo o adversário de atingir o mesmo conhecimento. Por isso, as garantias de segurança deste protocolo não se verificam no modelo malicioso, no qual é permitido que alguma das partes não cumpra ativamente os passos previamente delineados.

As principais desvantagens deste protocolo são duas, além do problema já mencionado no caso do modelo malicioso. Em primeiro lugar, a segurança associada às operações de exponenciação num espaço modular acarreta, evidentemente, um maior custo computacional, que pode ser indesejável. Em segundo lugar, a segurança do protocolo assenta, em parte, na definição dos parâmetros públicos, pelo que valores de N e p desadequados podem reduzir consideravelmente a segurança oferecida pelo protocolo, tornando-o vulnerável a ataques.

Em síntese, o protocolo baseado em Diffie-Hellman traz vantagens e desvantagens. Por um lado, este protocolo é mais seguro do que os anteriores, resolvendo os problemas da assimetria na troca de informação sem necessidade de recorrer permanentemente a uma terceira parte. Por outro lado, a maior segurança oferecida traz consigo um maior custo computacional e complexidade na implementação. Como tal, este protocolo considera-se adequado para aplicações mais exigentes a nível de segurança e privacidade da informação, desde que com capacidade computacional adequada à execução do mesmo.

Protocolo Baseado em *Oblivious Transfer*

Finalmente, o protocolo baseado em *oblivious transfer*, descrito por Benny Pinkas (*Bar-Ilan University*), Thomas Schneider (*TU Darmstadt*) e Michael Zohner (*TU Darmstadt*) em 2019, no artigo denominado “*Scalable Private Set Intersection Based on OT Extension*”, assenta o seu funcionamento no conceito de *oblivious transfer*.

Oblivious transfer é uma abordagem que permite partilhar informação sem saber que informação foi partilhada. No contexto da Alice (emissor) e do Bob (recetor), considere-se que a Alice contém dois segredos e que o Bob pretende aprender um deles. A aplicação de um protocolo de *oblivious transfer* permite que o Bob aprenda um (e um só) segredo da Alice, sem que a Alice saiba qual dos segredos foi aprendido pelo Bob. Para tal, é necessário seguir um protocolo, que pode passar por:

1. O Bob escolhe um par de chaves assimétricas (pública e secreta) e uma chave pública, sem chave secreta associada;
2. O Bob envia as duas chaves públicas (pk_0 e pk_1) para a Alice;
3. A Alice computa $c_0 = E(pk_0, m_0)$ e $c_1 = E(pk_1, m_1)$, sendo que $E(pk_i, m_i)$ significa encriptar o segredo m_i com a chave pública pk_i ;
4. A Alice envia c_0 e c_1 para o Bob;
5. O Bob descripta o criptograma para o qual possui a chave secreta correspondente, passando a conhecer apenas m_0 ou m_1 .

Ora, este protocolo satisfaz as garantias pretendidas, ao permitir que o recetor aprenda um (e um só) segredo do emissor, sem que o emissor saiba qual foi esse segredo. No entanto, assume-se comportamento semi-honesto pela parte que gera as chaves, visto que, se incumprir o protocolo e gerar dois pares de chaves públicas com chaves secretas, consegue facilmente descriptar ambas as mensagens enviadas pelo emissor, obtendo os dois segredos, indevidamente.

O protocolo de interseção de conjuntos privados baseado em *oblivious transfer* aproveita esta primitiva para construir um procedimento seguro para trocar a informação e computar a interseção. O protocolo exige que ambas as partes estabeleçam uma função de *hash*, que deve ser aplicada a cada um dos elementos dos seus conjuntos privados. Depois, deve ser utilizada uma função pseudoaleatória *oblivious* (OPRF) construída a partir de uma extensão do conceito de *oblivious transfer*, para maior eficiência. Com isto, uma parte aprende uma chave aleatória para cada índice, que é utilizada pela outra parte para avaliar a OPRF para o elemento de índice correspondente e pela própria para computar o valor para cada um dos seus elementos. Posto isto, esta parte permuta aleatoriamente os valores computados para todos os elementos do seu conjunto e envia-os para a outra parte, que determina a interseção entre o conjunto recebido e o conjunto calculado anteriormente. Assim, obtém-se o resultado da interseção dos conjuntos privados, que deve ser partilhado pela entidade que o obteve com o outro interveniente, de acordo com o protocolo.

Como sempre, concretize-se esta explicação recorrendo à Alice e ao Bob:

1. A Alice e o Bob concordam na utilização de uma função de *hash*;
2. A Alice e o Bob aplicam a função de *hash* a cada um dos elementos do seu conjunto;
3. A Alice aprende uma chave aleatória associada a uma OPRF para cada índice;
4. O Bob avalia a OPRF para cada elemento do seu conjunto;
5. A Alice computa a OPRF para cada elemento do seu conjunto;
6. A Alice reúne todos os valores computados num conjunto, permuta-o aleatoriamente e envia-o para o Bob;
7. O Bob verifica que valores pertencem ao seu conjunto e ao enviado pela Alice, determinando a interseção entre os conjuntos privados ao comparar os valores resultantes da aplicação da OPRF;
8. O Bob partilha a interseção obtida com a Alice.

Ora, este protocolo é mais complexo do que os anteriormente estudados, mas oferece segurança no modelo semi-honesto de forma eficiente, graças ao aproveitamento das potencialidades da extensão da primitiva de *oblivious transfer*, particularmente no que diz respeito à criptografia simétrica utilizada após a inicialização. A par disto, o protocolo pode ainda ser aumentado para fornecer garantias de segurança no modelo malicioso e acresce ainda a vantagem de não necessitar de entidades terceiras para o seu correto funcionamento.

Essencialmente, a principal desvantagem deste protocolo reside na complexidade da implementação, que é o ponto negativo trazido pela ausência de uma terceira parte e pela manutenção das garantias de segurança. Além disto, existe ainda o custo computacional associado à inicialização do processo de *oblivious transfer*, embora isto acabe por ser compensado pela eficiência do mecanismo de extensão, principalmente quando aplicável em conjuntos com um grande número de elementos, nos quais a escalabilidade do protocolo minimiza esta limitação. Por último, pode realçar-se a necessidade de aumentar ainda mais a complexidade caso seja necessário oferecer propriedades idênticas de segurança no modelo malicioso.

Em conclusão, este protocolo entende-se adequado para aplicações práticas inseridas no modelo semi-honesto e nas quais a segurança/privacidade dos dados seja extremamente importante, desde que exista capacidade técnica adequada para a sua implementação. No caso do modelo malicioso, também podem ser aplicadas as versões alternativas deste protocolo com o devido reforço da segurança.

Comparação

Tendo estudado e descrito os quatro protocolos principais para solucionar o problema da interseção de conjuntos privados, estes podem agora ser devidamente comparados.

Em primeiro lugar, relativamente à segurança da informação, sendo este um aspeto central da necessidade de *Secure Multiparty Computation*, note-se que a solução menos segura das estudadas é o *naïve hashing*, dada a multiplicidade de ataques possíveis contra esta abordagem. Dos restantes três algoritmos, todos, à exceção do Diffie-Hellman, têm a possibilidade de ser estendidos para oferecer garantias de segurança não só no modelo semi-honesto, mas também no modelo malicioso. Note-se, no entanto, que o modelo apoiado pelo servidor necessita de uma terceira parte honesta, na sua versão base, o que pode tornar os requisitos de segurança mais exigentes ou difíceis de cumprir.

Em segundo lugar, surge a necessidade de comparar o desempenho destes algoritmos. Inversamente à segurança, o algoritmo com melhor desempenho é o mais simples (*naïve hashing*), enquanto o protocolo baseado em Diffie-Hellman é aquele tem associado um maior custo computacional. Efetivamente, o protocolo assente em *oblivious transfer* consegue uma eficiência muito satisfatória em troca de uma implementação mais complexa, enquanto o protocolo apoiado pelo servidor transfere os custos computacionais para uma terceira parte, libertando os clientes.

Assim sendo, depois de comparar a segurança e o desempenho de cada solução, é possível sintetizar os principais pontos de destaque de cada uma. A abordagem de *naïve hashing* é extremamente simples, pelo que tem um bom desempenho computacional e baixo custo associado, mas oferece poucas garantias de segurança, estando vulnerável a ataques de dicionário para domínios reduzidos e altamente dependente da função de *hash* escolhida, sendo que estas garantias só se verificam no modelo semi-honesto. Como evolução natural deste processo, o protocolo apoiado pelo servidor reforça as garantias de segurança ao exigir uma terceira parte, para a qual também são transferidos os custos de processamento. Deste modo, consegue-se, simultaneamente, eficiência e segurança, mas requer-se um servidor independente e honesto. Este protocolo tem ainda a vantagem de poder ser estendido para funcionar no modelo malicioso, o que já não sucede com o apoiado por Diffie-Hellman que, por sua vez, apesar das garantias de segurança fortes, só funciona em contextos semi-honestos. No entanto, esta solução não requer terceiros para satisfazer todos os requisitos de segurança estabelecidos, ainda que o faça em prol de maiores custos de computação e comunicação, o que o torna pouco escalável. Por último, o protocolo baseado em *oblivious transfer* oferece total segurança e privacidade dos dados no modelo semi-honesto, sem depender de terceiros e podendo ainda ser estendido para o modelo malicioso. Além disso, é escalável e rápido após a fase inicial, mas é de difícil implementação.

Deste modo, cada protocolo pode ser destacado para um uso específico, de acordo com as suas características. O protocolo de *naïve hashing*, sendo o mais simples, eficiente e inseguro, só se adequa para casos em que a segurança dos dados não seja o fator mais importante, preferencialmente em domínios com muitos valores possíveis e nos quais a velocidade de computação independente de entidades externas seja um elemento preponderante. A solução apoiada pelo servidor é a ideal caso exista um terceiro conhecido confiável, com grande poder de computação disponível. Por sua vez, no caso de não existirem terceiros e de se pretenderem garantias fortes de segurança em conjuntos de dados relativamente pequenos, pode ser utilizada a abordagem apoiada por Diffie-Hellman, desde que inserida no modelo semi-honesto. Finalmente, para grandes conjuntos de dados com fortes necessidades de segurança, a melhor solução é a baseada em *oblivious transfer*, sendo a mais segura e escalável para estas situações.

Em suma, cada protocolo tem as suas vantagens e desvantagens, procurando-se a existência de um equilíbrio entre segurança e eficiência. Como tal, a escolha da solução mais adequada deve ser cuidadosamente ponderada em função do contexto operacional, da sensibilidade e do volume dos dados a tratar, bem como das exigências impostas pelo modelo de ameaça em questão.

Experiência com Protocolos para a Interseção de Conjuntos Privados

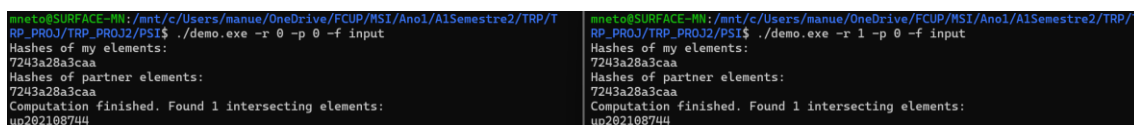
No sentido de compreender melhor os protocolos para a interseção de conjuntos privados anteriormente estudados, descritos e comparados, é possível utilizar o programa PSI¹ do *Cryptography and Privacy Engineering Group* da *Technische Universität Darmstadt*, juntamente com o analisador de rede Wireshark².

Para tal, descreve-se a execução de uma série de experiências relacionadas com os protocolos para a interseção de conjuntos privados, divididas em dez passos sequenciais.

Passo 1

Para explorar a solução de *naïve hashing* – na qual cada parte faz *hash* dos seus *inputs*, envia as *hashes* para as outras partes e depois computa a interseção com a *hash* do seu próprio *input* –, cria-se o ficheiro **input** com o conteúdo **up202108744**, através da execução do comando **echo “up202108744” >> input**.

De seguida, introduzem-se os comandos **./demo.exe -r 0 -p 0 -f input** e **./demo.exe -r 1 -p 0 -f input** em dois terminais, de maneira a executar a solução de *naïve hashing*, obtendo-se o *output* exposto na Figura 1.



```
mneto@SURFACE-PM: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/AISemestre2/TRP/T
RP_PROJ/TRP_PROJ2/PSI$ ./demo.exe -r 0 -p 0 -f input
Hashes of my elements:
7243a28a3caa
Hashes of partner elements:
7243a28a3caa
Computation finished. Found 1 intersecting elements:
up202108744

mneto@SURFACE-PM: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/AISemestre2/TRP/T
RP_PROJ/TRP_PROJ2/PSI$ ./demo.exe -r 1 -p 0 -f input
Hashes of my elements:
7243a28a3caa
Hashes of partner elements:
7243a28a3caa
Computation finished. Found 1 intersecting elements:
up202108744
```

Figura 1 – Passo 1

Efetivamente, verifica-se que a *hash* dos *inputs* de ambas as partes é a mesma, tal como seria de esperar, pelo que o valor do ficheiro **input** (**up202108744**) é a única interseção resultante dos dois conjuntos, que são, na verdade, o mesmo ficheiro.

¹ <https://github.com/eduardo010174/PSI>

² <https://www.wireshark.org/>

Passo 2

De forma análoga ao passo anterior, cria-se o ficheiro **input2** com o conteúdo **test**, através da execução do comando **echo "test" >> input2**.

Igualmente, executam-se os comandos **./demo.exe -r 0 -p 0 -f input** e **./demo.exe -r 1 -p 0 -f input2** em dois terminais, com o mesmo propósito anterior, expondo-se o resultado obtido na Figura 2.

```
minikube@SURFACE-PM1: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/A1Semestre2/TRP/T
RP_PROJ/TRP_PROJ/PSI$ ./demo.exe -r 0 -p 0 -f input
Hashes of my elements:
7243a28a3caa
Hashes of partner elements:
9f86d081884c
Computation finished. Found 0 intersecting elements:

minikube@SURFACE-PM1: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/A1Semestre2/TRP/T
RP_PROJ/TRP_PROJ/PSI$ ./demo.exe -r 1 -p 0 -f input2
Hashes of my elements:
9f86d081884c
Hashes of partner elements:
7243a28a3caa
Computation finished. Found 0 intersecting elements:
```

Figura 2 – Passo 2

Desta vez, a Figura 2 mostra que não foi encontrada nenhuma interseção entre ambos os conjuntos, ou seja, os ficheiros **input** (que contém **up202108744**) e **input2** (com o conteúdo **test**) não têm nada em comum. Esta conclusão deriva do facto de as *hashes* do conteúdo de cada ficheiro serem diferentes, pelo que a interseção é o conjunto vazio.

Passo 3

Analisando o tráfego na rede – particularmente, na interface de *loopback* – através do Wireshark, é possível extrair algumas conclusões. As Figuras 3 e 4 mostram, para os Passos 1 e 2, respetivamente, os dois pacotes TCP trocados em cada caso entre ambas as partes, com origem e destino no porto 7766.

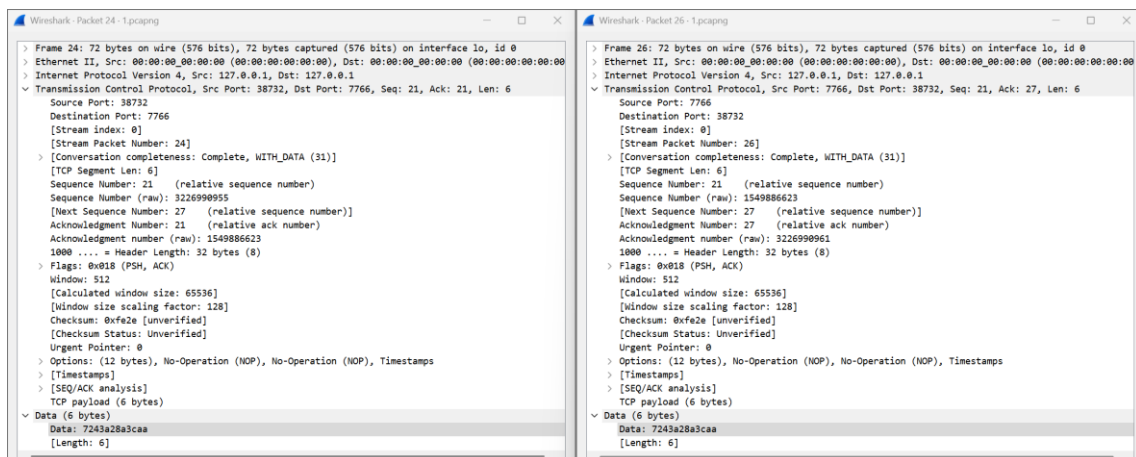


Figura 3 – Passo 3 (Passo 1)

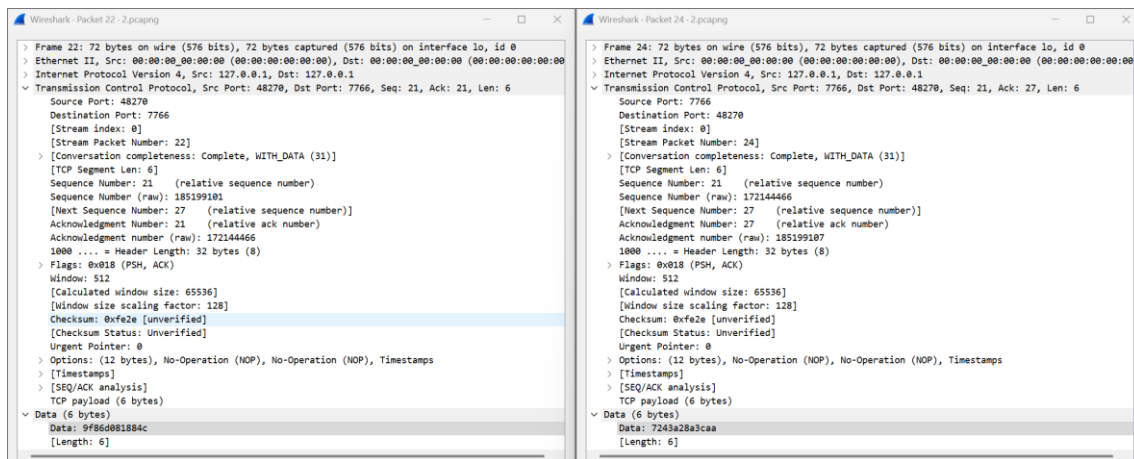


Figura 4 – Passo 3 (Passo 2)

Ora, quer a Figura 3, quer a Figura 4, mostram que os dados trocados entre ambas as partes para a computação da interseção dos conjuntos consistem nas *hashes* dos valores de cada conjunto. Efetivamente, na Figura 3, referente ao Passo 1, ambas as partes enviam a *hash* **7243a28a3caa**, que corresponde ao valor **up202108744**. Na Figura 4, relativa ao Passo 2, a parte que contém o ficheiro **input2**, com o conteúdo **test**, envia a *hash* **9f86d081884c**, enquanto a outra parte envia a *hash* anterior, visto que se trata do mesmo ficheiro **input**. Todos estes pacotes trocados têm seis *bytes* de dados.

Passo 4

Considerando os últimos três pontos, é possível tecer algumas considerações que explicam porque é que este protocolo de *naïve hashing* pode ser inseguro.

Em primeiro lugar, convém realçar que computar a *hash* de um valor não fornece as mesmas garantias de segurança do que cifrá-lo/criptá-lo. Como as funções de *hash* são determinísticas, em domínios limitados ou restritos pode ser viável realizar ataques de força-bruta ou usando *rainbow tables* contra as *hashes*, de modo a descobrir os *inputs* originais.

Em segundo lugar, como cada parte envia as *hashes* de todos os valores do seu conjunto, a outra parte não só é capaz de verificar se um valor está em ambos os conjuntos, mas também consegue determinar se um determinado valor está presente no conjunto da outra parte, mesmo que não esteja presente no seu. Para tal, basta apenas computar a *hash* do valor em causa utilizando a mesma função de *hash* e comparar o valor resultante contra as *hashes* enviadas pela outra parte. Este facto configura um problema de significativo de privacidade do *input*, já que o *input* de cada parte deveria ser privado, a menos dos elementos comuns.

Em terceiro lugar, esta instanciação concreta do algoritmo utiliza apenas os primeiros seis *bytes* da *hash*, o que é um valor extremamente reduzido, visto que só fornece $2^{6 \times 8} = 2^{48}$ combinações possíveis, aumentando a probabilidade de ocorrência de colisões e de ataques de força-bruta. A possibilidade de colisões afeta não só a segurança do protocolo, mas também o seu correto funcionamento, visto que elementos diferentes que se mapeiem na mesma *hash* – uma colisão – são identificados como um valor comum, incorretamente. No entanto, este problema existe apenas nesta implementação concreta, sendo facilmente resolvido através da utilização do valor completo da *hash*.

Em penúltimo lugar, a execução do protocolo não obriga a que algum dos intervenientes envie sequer a sua informação, podendo apenas receber a informação da outra parte e computar localmente a interseção, sem ter de partilhar com a outra parte o seu conjunto. Como tal, uma parte maliciosa pode simplesmente obter as *hashes* do conjunto da outra parte e não partilhar as *hashes* dos valores do seu próprio conjunto, conseguindo computar a interseção e inviabilizando a computação pela outra parte. Além disso, a parte maliciosa pode também partilhar informação parcial ou incorreta, prejudicando a aquisição de conhecimento pela outra parte. Este problema é uma realidade no modelo malicioso, ainda que seja desconsiderado no modelo semi-honesto, no qual se pressupõe o cumprimento do protocolo por ambas as partes.

Por último, como a informação é transmitida na rede de forma não encriptada, um *eavesdropper* pode intersetar as comunicações e guardar as *hashes* trocadas de modo a concretizar um ataque *offline*. Ainda assim, este problema não se deve propriamente ao protocolo utilizado, mas sim à forma de comunicação, visto que esta poderia facilmente ser encriptada.

Sumariamente, o protocolo de *naïve hashing* tem várias razões pelas quais pode ser considerado inseguro. Em particular, destaca-se a viabilização de ataques de força-bruta ou por *rainbow tables* em domínios reduzidos, dado o determinismo das funções de *hash*, bem como a facilidade de uma parte determinar se um certo valor pertence ao conjunto da outra parte e a assimetria do protocolo, ou seja, a não reciprocidade, que configura um problema no modelo malicioso, ainda que seja admissível no paradigma semi-honesto.

Passo 5

Para aplicar este mesmo algoritmo às listas de aplicações fornecidas, executam-se os comandos `./demo.exe -r 0 -p 0 -f AppList1.csv` e `./demo.exe -r 1 -p 0 -f AppList2.csv`, apresentando-se um excerto do *output* na Figura 5.

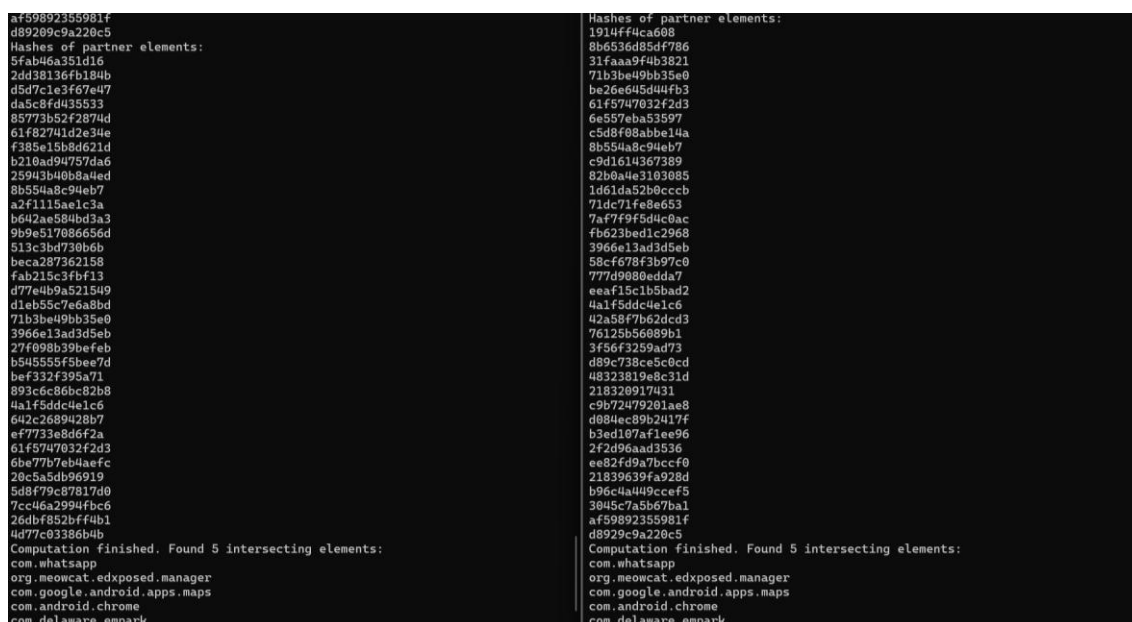


Figura 5 – Passo 5

A Figura 5 mostra que o resultado computado para a interseção entre as duas listas de aplicações (**AppList1.csv** e **AppList2.csv**) consiste nas seguintes cinco aplicações:

1. **com.whatsapp;**
2. **org.meowcat.edxposed.manager;**
3. **com.google.android.apps.maps;**
4. **com.android.chrome;**
5. **com.delaware.empark.**

Utilizando o Wireshark para capturar os pacotes trocados na rede, visualizam-se as estatísticas apresentadas na Tabela 1, relativamente ao número total de pacotes e *bytes* capturados.

Medida	Capturados
Pacotes	28
Bytes	2394

Tabela 1 – Passo 5

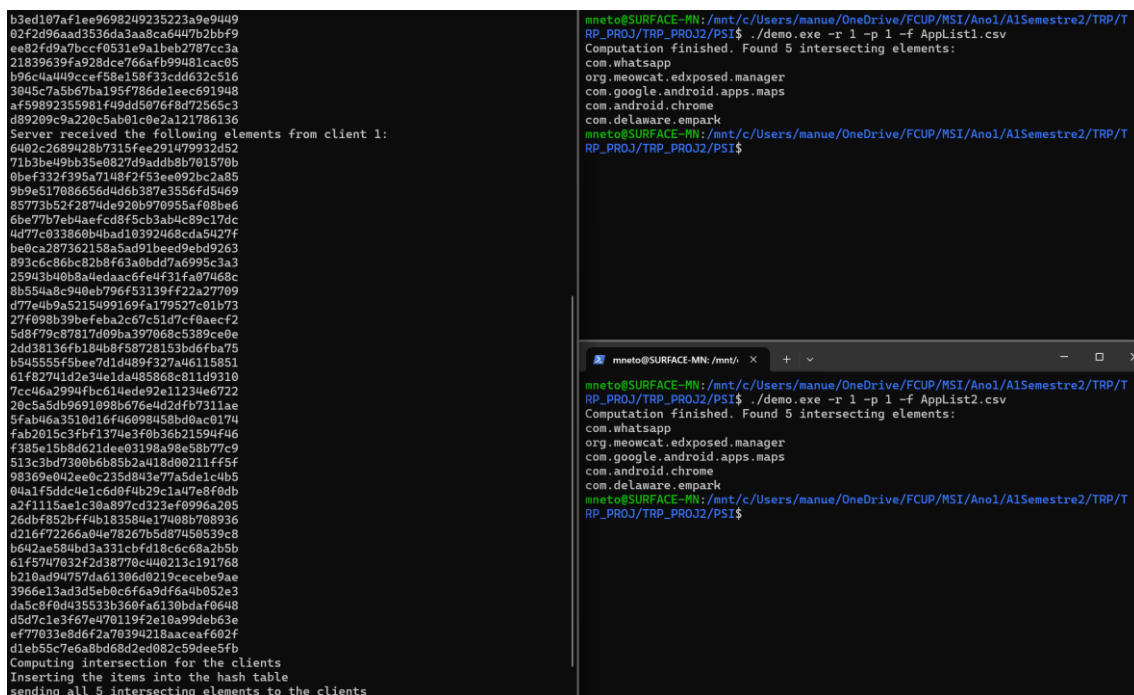
Passo 6

Para explorar o protocolo apoiado pelo servidor – que assume uma terceira parte responsável por calcular a interseção entre os conjuntos – é necessário editar o ficheiro **AppList2.csv** de maneira a conter o mesmo número de linhas (36) que o ficheiro **AppList1.csv**, acrescentando-lhe, por exemplo, dois e-mails extraídos do ficheiro **emails_alice.txt**, como **Michael.Zohner@ec-spride.de** e **Evelyne.Wagener@tvcablenet.be**.

Para tal, executam-se os seguintes três comandos, em três terminais distintos:

1. `./demo.exe -r 0 -p 1 -f README.md;`
2. `./demo.exe -r 1 -p 1 -f AppList1.csv;`
3. `./demo.exe -r 1 -p 1 -f AppList2.csv.`

O primeiro comando lança o servidor, enquanto os outros dois comandos correspondem aos clientes. O resultado truncado da execução demonstra-se na Figura 6.



```
b3ed107af1ee9698249235223a9e9449
02f2d96aad3536da3aa8ca6447b2bbf9
ee82fd9a7bccc0531e9a1beb2787cc3a
21839639fa928dce766af6b99481cac05
b96e4a449cecf58e158f33cdde62e516
3045c7a5b67ba195f786d1ee691948
af59892355981f49dd5876f8d72565c3
d89209c9a220c5ab81c0e2a121786136
Server received the following elements from client 1:
6402c2689428b7315fee291479932d52
71b3be49bb35e0827d9add8b781570b
0bef332f395a7148f2f53ee092bc2a85
9b9e517086656dd6b387e3556fd5469
85773b52f2874de920b970955af08be6
6be77b7b4aefcd8f5cb3abdc89c17dc
4d77c033860b4bad10392468da5427f
be0ca287362158a5ad91beed9ebd9263
893c6c86bc82b8f63a0bdd7a6995c3a3
25943b40b8a4edaac6fe4f31fa07468c
8b554a8c940eb796f53139ff22a27709
d77e4b9a5215499169fa179527c01b73
27f098b39befeba2c67c51d7cf0aecf2
5d8799c87817d09ba397068c5389ce0e
2dd38136fb184b9f58728153bd66bf75
b54555f5bee7d1d489f327a46115851
61f82741d2e34e1da485868c81d9310
7cc46a2994fbc614ede92e11234e6722
20c5a5db9691098b676e4d2dfb7311ae
5fab46a3510d16f4609848bd0ac6174
fab2015c3fbf1374e3f0b36b21594f46
f385e15b8d621dee03198a98e58b77c9
513c3bd7308b6b5b2a418d00211ff5f
98369e042e0e225d043e77a5de1c4b5
04a1f5ddc4e1c6d0f4b29c1a47e8f0db
a2f1115ae1c30a897c323ef0996a205
26dbf852bfff4b183584e17408b708936
d216f72266a04e78267b5d87450539c8
b642ae584bd3a331cbfd18c6c68a2b5b
61f5747032f2d38770c440213c191768
b210ad94757da61306d0219cecebe9ae
396e13ad3d5eb0cf6a9df6a4b052e3
da5c8f0d43533b360fa6130bda4f0648
d5d7c1e3f67e470119f2e10a99da6b3e
af77033e8d6f2a70394218aacea602f
d1eb55c7e6a8bd68d2ed082c59dee5fb
Computing intersection for the clients
Inserting the items into the hash table
sending all 5 intersecting elements to the clients

mneto@SURFACE-MN: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/A1Semestre2/TRP/T
RP_PROJ/TRP_PROJ2/PSI$ ./demo.exe -r 1 -p 1 -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
mneto@SURFACE-MN: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/A1Semestre2/TRP/T
RP_PROJ/TRP_PROJ2/PSI$

mneto@SURFACE-MN: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/A1Semestre2/TRP/T
RP_PROJ/TRP_PROJ2/PSI$ ./demo.exe -r 1 -p 1 -f 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
mneto@SURFACE-MN: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/A1Semestre2/TRP/T
RP_PROJ/TRP_PROJ2/PSI$
```

Figura 6 – Passo 6

A Figura 6 evidencia que o resultado obtido pelos clientes consiste nas mesmas cinco aplicações determinadas anteriormente, tal como seria expectável.

A Tabela 2 apresenta as estatísticas capturadas pelo Wireshark para o total de pacotes e *bytes* trocados na rede durante a execução deste protocolo.

Medida	Capturados
Pacotes	32
Bytes	3330

Tabela 2 – Passo 6

Passo 7

No *output* presente no servidor da Figura 6, mostram-se as *hashes* dos dados recebidos pelos clientes. Como tal, cada cliente envia ao servidor as *hashes* dos valores do seu conjunto e o servidor só devolve a cada cliente as *hashes* comuns em ambos os conjuntos. No entanto, tal como anteriormente, este protocolo também tem algumas falhas de segurança, que se expõem de seguida.

Antes de mais, note-se que este protocolo já mitiga algumas das vulnerabilidades do caso anterior. Em particular, a possibilidade de uma das partes receber as *hashes* da outra parte, mas não enviar nenhuma informação, deixa de existir, visto que as ambas as partes devem enviar os seus dados para o servidor, que, admitindo o seu correto funcionamento, só envia para os clientes a interseção dos conjuntos recebidos. Deste modo, impede-se que alguma das partes seja capaz de obter informação da outra parte, sem disponibilizar a sua informação. Além disso, também se impede que alguma das partes concretize um ataque *offline*, aproveitando o conteúdo armazenado proveniente da outra, porque os únicos dados recebidos são as interseções. Assim, são resolvidas as duas principais falhas do protocolo anterior.

Contudo, existem ainda alguns problemas que prevalecem neste protocolo. Desde logo, todas as assunções de segurança são transpostas para o servidor, que ambas as partes devem assumir ser uma terceira parte honesta, sem colaborar maliciosamente com nenhuma das entidades. Efetivamente, caso seja malicioso, o servidor é capaz de concretizar qualquer ataque anteriormente mencionado, ou seja, verificar se um determinado valor pertence a algum dos conjuntos, sem disponibilizar essa informação a ambas partes, bem como materializar ataques de força-bruta e/ou utilizando *rainbow tables*. Esta centralização da computação no servidor torna-o um ponto único de falha e faz recair sobre ele toda a confiança e assunções necessárias ao funcionamento seguro do protocolo que, não sendo verificadas, tornam-no vulnerável.

Além disso, caso o servidor seja desonesto, pode devolver resultados incorretos a uma ou a ambas as partes, enganando-as, bem como retornar conjuntos apenas parcialmente corretos, ou diferentes entre as partes. Por exemplo, um servidor malicioso pode devolver sempre o conjunto vazio ou o conjunto enviado por cada parte, computando ele próprio a interseção, mas não o disponibilizando a nenhuma das partes. Por isso, esta versão do protocolo não oferece quaisquer garantias de segurança perante um servidor malicioso, sendo que, para tal, é necessário estender o protocolo conforme explicado anteriormente.

A par disto, nesta implementação do protocolo, o servidor passa a conhecer o tamanho da interseção entre os conjuntos privados, o que pode não ser desejável. Para o impedir, o protocolo teria, também, de ser adaptado para uma das possíveis extensões.

Em síntese, este protocolo transfere as garantias e assunções de segurança/privacidade para uma terceira parte que se assume/considera ser honesta, mas, se não o for, está sujeito a falhas idênticas às do protocolo anterior. Em particular, o protocolo não é robusto para cenários nos quais o servidor seja malicioso ou desonesto, casos em que está suscetível a todas as vulnerabilidades anteriormente descritas.

Passo 8

De maneira a explorar o protocolo de interseção de conjuntos privados baseado em Diffie-Hellman, executam-se os seguintes comandos, depois de restabelecer o ficheiro **AppList2.csv** ao seu conteúdo original, removendo as duas linhas previamente adicionadas: **./demo.exe -r 0 -p 2 -f AppList1.csv** e **./demo.exe -r 1 -p 2 -f AppList2.csv**.

A Figura 7 mostra o resultado da execução.



```
mneto@SURFACE-MN: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/A1Semestre2/TRP/TRP_PROJ/TRP_PROJ2/PSI$ ./demo.exe -r 0 -p 2 -f AppList1.csv
mneto@SURFACE-MN: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/A1Semestre2/TRP/TRP_PROJ/TRP_PROJ2/PSI$
mneto@SURFACE-MN: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/A1Semestre2/TRP/TRP_PROJ/TRP_PROJ2/PSI$
mneto@SURFACE-MN: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/A1Semestre2/TRP/TRP_PROJ/TRP_PROJ2/PSI$ ./demo.exe -r 1 -p 2 -f 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 – Passo 8

Através da visualização da Figura 7, verifica-se que o terminal com **-r 0** não computa a interseção, sendo esta informação unicamente apresentada no terminal com **-r 1**. Como expectável, o resultado da interseção dos conjuntos privados coincide com o obtido anteriormente, nas mesmas cinco aplicações.

A Tabela 3 contém as estatísticas obtidas pelo Wireshark, relativamente ao número de pacotes e *bytes* capturados nesta troca de informação entre as partes.

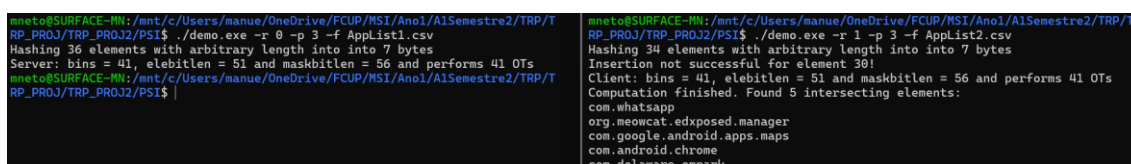
Medida	Capturados
Pacotes	30
Bytes	5714

Tabela 3 – Passo 8

Passo 9

De modo a explorar o protocolo de interseção de subconjuntos privados baseado em *oblivious transfer*, executam-se os comandos `./demo.exe -r 0 -p 3 -f AppList1.csv` e `./demo.exe -r 1 -p 3 -f AppList2.csv`.

O *output* apresenta-se na Figura 8.



```

mneto@SURFACE-MN: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/A1Semestre2/TRP/T
RP_PROJ/TRP_PROJ2/PSI$ ./demo.exe -r 0 -p 3 -f AppList1.csv
Hashing 36 elements with arbitrary length into into 7 bytes
Server: bins = 41, elebitlen = 51 and maskbitlen = 56 and performs 41 OTs
mneto@SURFACE-MN: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/A1Semestre2/TRP/T
RP_PROJ/TRP_PROJ2/PSI$
mneto@SURFACE-MN: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/A1Semestre2/TRP/T
RP_PROJ/TRP_PROJ2/PSI$ ./demo.exe -r 1 -p 3 -f AppList2.csv
Hashing 34 elements with arbitrary length into into 7 bytes
Insertion not successful for element 30!
Client: bins = 41, elebitlen = 51 and maskbitlen = 56 and performs 41 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 8 – Passo 9

Tal como anteriormente, a Figura 8 evidencia que o resultado da interseção dos conjuntos coincide com as cinco aplicações que seriam expectáveis, salientando-se também o facto de que o terminal com `-r 0` não computa a interseção, mas apenas o terminal com `-r 1`.

A Tabela 4 mostra as informações resultantes da captura ao tráfego da rede executada pelo Wireshark aquando da execução deste protocolo.

Medida	Capturados
Pacotes	37
Bytes	55072

Tabela 4 – Passo 9

Passo 10

Para efetuar uma análise global, a Tabela 5 agrega todos os dados recolhidos anteriormente para os quatro protocolos, no que concerne ao número de pacotes trocados e ao total de *bytes* dos mesmos.

Protocolo	Pacotes	Bytes
<i>Naïve Hashing</i>	28	2394
Servidor	32	3330
Diffie-Hellman	30	5714
<i>Oblivious Transfer</i>	37	55072

Tabela 5 – Passo 10

Os resultados da Tabela 5 evidenciam que, em traços gerais, conforme aumenta a complexidade e, simultaneamente, as garantias de segurança oferecidas pelo protocolo, maior é o número de pacotes e de *bytes* trocados.

Em primeiro lugar, o protocolo mais simples – de *naïve hashing* – é aquele que leva à troca de menos pacotes e de menos *bytes*, ou seja, que requer menos informação. Este comportamento deve-se ao facto de, para a execução correta do protocolo, ser apenas necessário trocar as *hashes* de cada valor de cada conjunto privado, pelo que a quantidade de informação é mínima, estando extremamente próxima da informação original em termos de quantidade. No entanto, este protocolo é aquele que oferece menos garantias de segurança pelos motivos já explicitados.

Em segundo lugar, o protocolo apoiado pelo servidor exige um número de pacotes e de *bytes* ligeiramente maior do que o anterior. Ora, tendo em conta que o funcionamento deste protocolo é em tudo idêntico ao da solução de *naïve hashing*, mas os dados são enviados para computação da interseção por parte do servidor, em vez de diretamente para a outra parte, a justificação para este facto é facilmente observável: os pacotes adicionais devem corresponder ao envio do conjunto resultante da interseção a partir do servidor para cada uma das partes. Assim, o custo de comunicação é extremamente semelhante ao protocolo anterior, mas, admitindo a honestidade do servidor, as garantias de segurança oferecidas são superiores, pelo que este protocolo se afigura mais vantajoso. Contudo, veja-se que um aumento da segurança se reflete num maior custo comunicacional.

Em terceiro lugar, o protocolo baseado em Diffie-Hellman tem associado um número de pacotes ligeiramente inferior à solução anterior, mas superior à abordagem de *naïve hashing*. A ausência de interação com uma terceira parte, isto é, a comunicação direta entre as partes, deve justificar este menor número de pacotes trocados. Em sentido contrário, o total de *bytes* é extremamente maior do que nas situações anteriores. Efetivamente, isto sucede porque, de acordo com o funcionamento deste algoritmo, as partes têm de trocar muita informação para serem capazes de computar a interseção de forma segura. Nomeadamente, segundo o protocolo Diffie-Hellman, é necessário computar e trocar valores de grande magnitude, resultantes de exponenciações e operações modulares, que visam garantir a segurança da informação partilhada, ao tornar computacionalmente difícil a computação inversa. Deste modo, a maior segurança sem recorrer a uma terceira parte tem como consequência um custo comunicacional mais elevado.

Por fim, o protocolo baseado em *oblivious transfer* é aquele que troca mais pacotes – ainda que a diferença não seja muito significativa – e mais *bytes* – já com uma diferença considerável. De facto, a inicialização deste protocolo requer a troca de muita informação associada à primitiva de *oblivious transfer* e à sua extensão, o que se reflete no maior número de pacotes e *bytes*. Além disso, a partilha do resultado da aplicação da função pseudoaleatória *oblivious* a cada elemento vem somar à quantidade de informação trocada. No entanto, este custo comunicacional não está diretamente associado a um maior custo computacional, medido em tempo de processamento, como se verá de seguida. Ainda assim, verifica-se, uma vez mais, que as garantias de segurança mais fortes têm associadas a necessidade de trocar mais informação, como seria expectável.

Em síntese, constata-se que a segurança e o custo comunicacional seguem uma relação aproximadamente direta. Objetivamente, a solução de *naïve hashing* é a mais simples e a menos segura, pelo que é aquela que requer menos informação, tanto em termos de pacotes como em relação ao número de *bytes*. De seguida, a abordagem apoiada pelo servidor oferece mais garantias de segurança, transferindo a computação para uma terceira parte, mas isto acarreta uma maior necessidade de troca de informação, sob a forma de pacotes e *bytes*. Por sua vez, a solução assente em Diffie-Hellman, segura dada a complexidade computacional dos cálculos associados, já permite a comunicação direta entre as partes, reduzindo o número de pacotes, mas aumenta substancialmente o número de *bytes* trocados, dada a magnitude dos valores envolvidos, para satisfazer as garantias de segurança. Por último, o protocolo baseado em *oblivious transfer* destaca-se por fornecer a maior segurança e escalabilidade computacional, mas fá-lo à custa da troca de muito mais informação entre as partes envolvidas, não só em número de pacotes, mas também no tamanho dos mesmos.

Como tal, existe uma relação indissociável entre a segurança oferecida por cada protocolo e o custo comunicacional associado.

Benchmarking de Protocolos para a Interseção de Conjuntos Privados

No sentido de comparar o desempenho dos diferentes protocolos para a interseção de conjuntos privados, executa-se a ferramenta de *benchmarking* através dos comandos `./psi.exe -r 0 -p <PROTOCOLO> -b 16 -n <N>` e `./psi.exe -r 1 -p <PROTOCOLO> -b 16 -n <N>`, em dois terminais.

Os protocolos em comparação são a solução de *naïve hashing*, o protocolo baseado em Diffie-Hellman e o apoiado por *oblivious transfer*. O valor de N representa o número de elementos em cada conjunto privado para *benchmarking*, tendo sido variado entre 10 e 50000 para permitir medir os seus efeitos e extrair conclusões significativas.

A Tabela 6 apresenta o tempo de execução, em segundos, para cada protocolo.

Protocolo N	<i>Naïve Hashing</i>	Diffie-Hellman	<i>Oblivious Transfer</i>
10	0	0	0,2
50	0	0,1	0,2
100	0	0,2	0,3
500	0	0,3	0,3
1000	0	0,7	0,2
5000	0	3,4	0,3
10000	0,1	6,2	0,3
50000	0,3	31,9	0,4
100000	0,6	66,0	0,5
500000	3,2	342,8	1,6

Tabela 6 – Tempo de Execução (s)

Os resultados da Tabela 6 mostram de forma clara que a solução de *naïve hashing* e o protocolo apoiado em *oblivious transfer* apresentam os tempos de execução mais reduzidos, enquanto o protocolo baseado em Diffie-Hellman tem tempos de execução consideravelmente superiores. Ora, sendo a solução de *naïve hashing* a mais simples, é esperado que o seu tempo de execução seja diminuto. Na verdade, a abordagem assente em *oblivious transfer* demonstra, para conjuntos pequenos, um tempo ligeiramente superior, mas, para conjuntos maiores, executa ainda mais rápido do que *naïve hashing*, o que confirma o seu grande potencial de escalabilidade devido à extensão da primitiva de *oblivious transfer*, que acrescenta um ligeiro *overhead* computacional na inicialização, mas compensa-o com interações mais rápidas para conjuntos maiores. Por sua vez, o protocolo apoiado por Diffie-Hellman tem tempos de execução extremamente superiores, devidos ao elevado custo computacional das computações necessárias associadas.

A Figura 9 ilustra estes mesmos dados num gráfico.

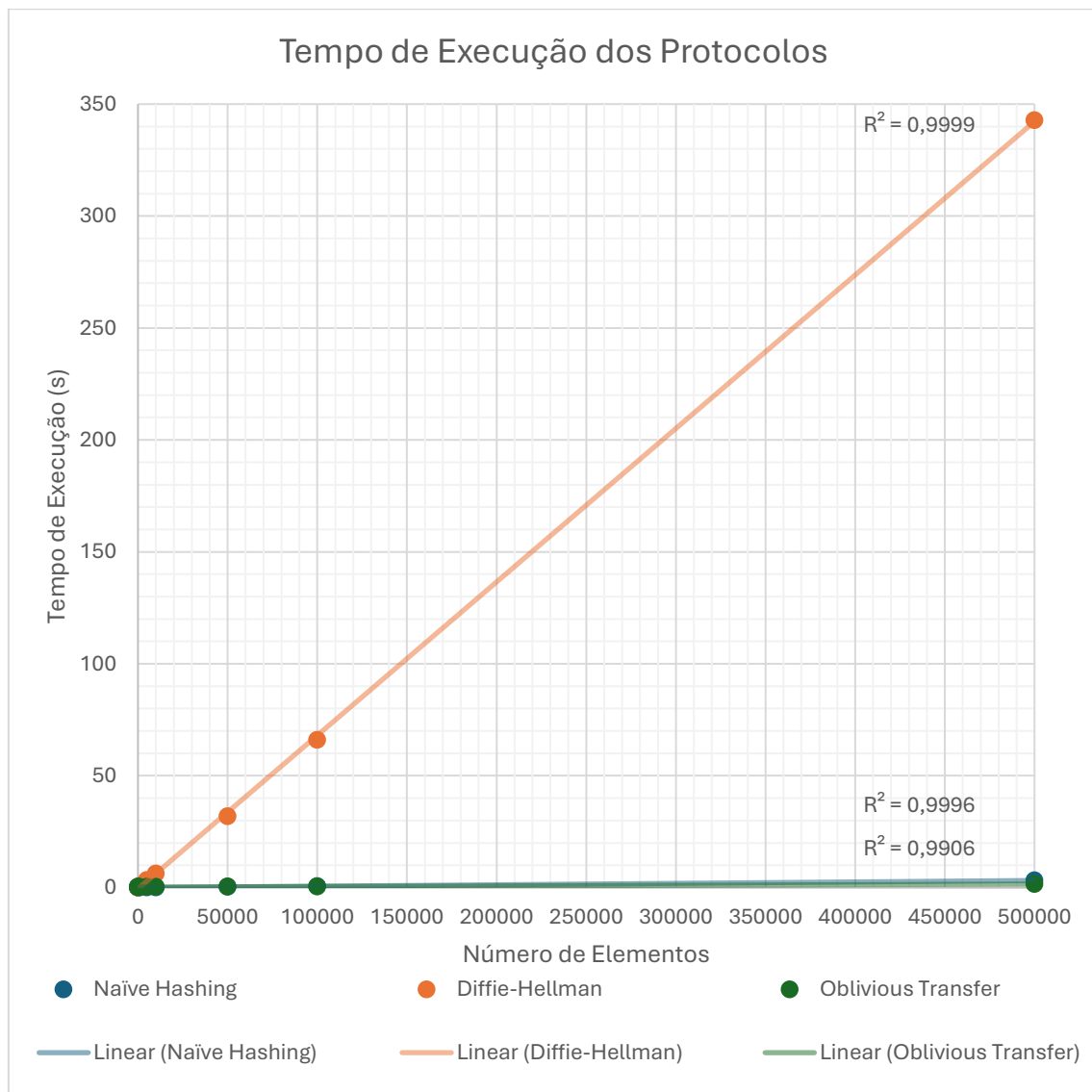


Figura 9 – Tempo de Execução dos Protocolos

Efetivamente, o gráfico da Figura 9 mostra que o tempo de execução de cada protocolo segue uma relação aproximadamente linear, relativamente ao número de elementos dos conjuntos privados, como é visível pelos valores do coeficiente de correlação linear, que se encontram extremamente próximos de um. Além disto, a figura permite também reforçar a conclusão extraída anteriormente: o algoritmo baseado em Diffie-Hellman leva consideravelmente mais tempo a executar, pelas razões já explicitadas. Verifica-se igualmente que os protocolos de *naïve hashing* e *oblivious transfer* têm uma taxa de crescimento linear do tempo de execução com o número de elementos, mas extremamente reduzida, encontrando-se muito próxima do eixo horizontal.

Assim, confirma-se a maior eficiência das soluções de *naïve hashing* e *oblivious transfer*, em comparação com Diffie-Hellman. De facto, evidencia-se que o protocolo apoiado em *oblivious transfer* é aquele que aparenta ser mais escalável, quando considerado apenas o tempo de execução. Em sentido inverso, o protocolo apoiado em Diffie-Hellman é o mais demorado, pelo já explicado.

A Tabela 7 contém a quantidade de informação trocada entre as partes, somando os dados enviados com os dados recebidos para cada protocolo, em *megabytes*.

Protocolo N	<i>Naïve Hashing</i>	Diffie-Hellman	<i>Oblivious Transfer</i>
10	0	0	0
50	0	0	0
100	0	0	0
500	0	0	0,1
1000	0	0,1	0,1
5000	0	0,5	0,5
10000	0,2	1,1	1,1
50000	0,8	5,1	5
100000	2	11,2	10,2
500000	9,6	50,5	50,9

Tabela 7 – Dados Trocados (MB)

A Tabela 7 demonstra resultados que complementam a análise anterior. Por um lado, o protocolo de *naïve hashing* é, tal como anteriormente, aquele que requer a menor quantidade de informação trocada, visto que esta informação corresponde apenas às *hashes* dos elementos dos conjuntos privados. Por outro lado, os protocolos apoiados em Diffie-Hellman e em *oblivious transfer* são os que têm associada a troca de mais informação, com valores idênticos entre ambos. Efetivamente, isto deve-se precisamente ao modo de funcionamento destes algoritmos, que exigem a partilha de muita informação entre as partes de modo a assegurar as garantias de segurança pretendidas, sem depender de uma terceira parte. Os resultados demonstram que a quantidade de informação devida ao protocolo Diffie-Hellman é essencialmente equivalente à do protocolo por *oblivious transfer*, ainda que sejam dados fundamentalmente diferentes.

O gráfico da Figura 10 permite detalhar a análise destes dados.

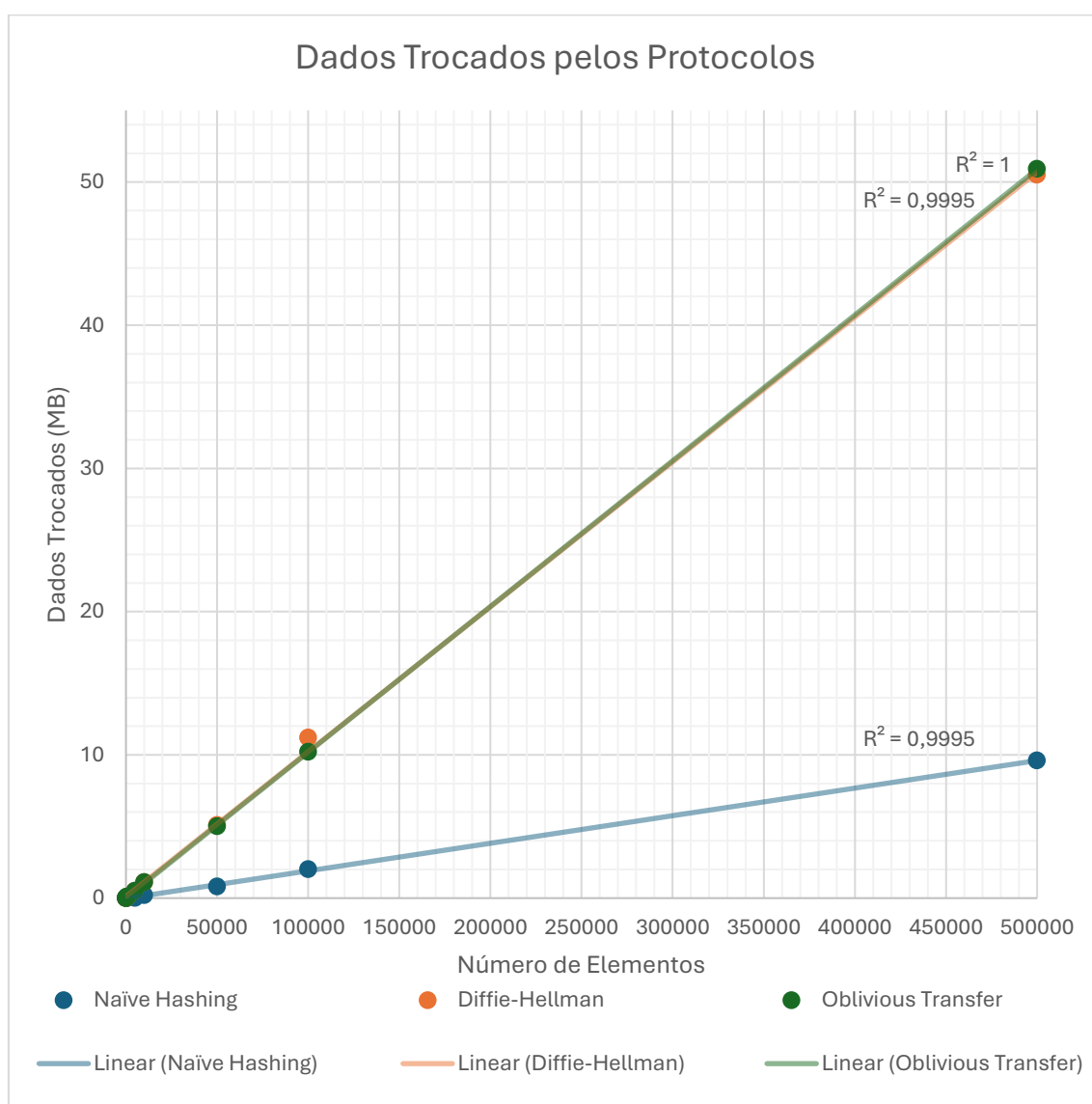


Figura 10 – Dados Trocados pelos Protocolos

Ora, o gráfico apresentado na Figura 10 evidencia a relação linear dos dados trocados por cada protocolo com o número de elementos de cada conjunto privado. De facto, todas as relações são aproximadamente lineares, tendo em conta que o valor do coeficiente R^2 se aproxima de um, em todos os casos. Assim, o gráfico permite reforçar a conclusão de que o protocolo de *naïve hashing* é o mais eficiente dos três, na medida em que exige a troca de muito menos informação, enquanto os protocolos assentes em Diffie-Hellman e em *oblivious transfer* requerem a troca de uma quantidade de informação semelhante entre ambos, mas muito maior do que na aplicação de *naïve hashing*. Com isto, reforça-se que as maiores garantias de segurança trazem consigo, de forma indissociável, uma maior quantidade de informação que é necessário trocar, na ausência de uma terceira parte.

Mais uma vez, observa-se que o protocolo apoiado por Diffie-Hellman é o mais custoso, desta vez em termos da quantidade de dados trocados entre as partes. No entanto, nesta métrica, o protocolo baseado em *oblivious transfer* acompanha estes resultados, voltando a destacar-se a solução de *naïve hashing* pelas suas menores exigências comunicacionais, mas que também se refletem em menores garantias de segurança.

Análise Crítica

Comparados estes valores para os três protocolos, é possível sintetizar as conclusões obtidas e realizar uma análise crítica global.

Objetivamente, verifica-se que o protocolo apoiado por Diffie-Hellman é aquele que tem não só um maior tempo de execução, mas também mais dados trocados entre as partes. Em sentido inverso, a solução assente em *naïve hashing* é a que executa mais rapidamente e com menor quantidade de informação partilhada. O protocolo baseado em *oblivious transfer* encontra-se num ponto intermédio entre ambos, com um tempo de execução comparável a *naïve hashing*, mas uma quantidade de dados trocados semelhante a Diffie-Hellman.

Como tal, comprova-se a existência da já referida relação entre segurança e desempenho, ao verificar-se que o protocolo menos seguro (*naïve hashing*) é aquele que apresenta um melhor desempenho, precisamente por satisfazer menos requisitos de segurança. No entanto, para o mesmo grau de segurança, o protocolo apoiado por *oblivious transfer* demonstra um desempenho consideravelmente superior ao assente em Diffie-Hellman, relativamente ao tempo de execução.

Assim, o algoritmo baseado em *oblivious transfer* parece ser a escolha ideal para a interseção de conjuntos privados, ao conjugar escalabilidade e segurança.

Aplicação de Protocolos de Interseção Segura a Outro Dataset

De maneira a experimentar a aplicação de protocolos de interseção segura na prática, define-se um cenário real de exploração. Nesse sentido, opta-se por procurar determinar os contactos telefónicos comuns entre os dois membros do grupo.

Para tal, cada elemento do grupo exportou os seus contactos telefónicos para um ficheiro CSV. Após isto, normalizaram-se os dados resultantes, nomeadamente através da remoção dos indicadores internacionais (como +351, entre outros) e de eventuais espaços entre os dígitos, por exemplo. Assim, ambos os conjuntos contêm números de telemóvel reais e estão preparados para o cálculo da sua interseção. Os ficheiros resultantes – **contacts1.csv** e **contacts2.csv** – têm tamanhos diferentes, visto que um tem pouco mais de 100 contactos e o outro tem mais de 450, mas isto não é problemático para os efeitos da demonstração.

A Figura 11 apresenta um excerto truncado dos dois *datasets*, aos quais aplicar os protocolos de interseção segura.

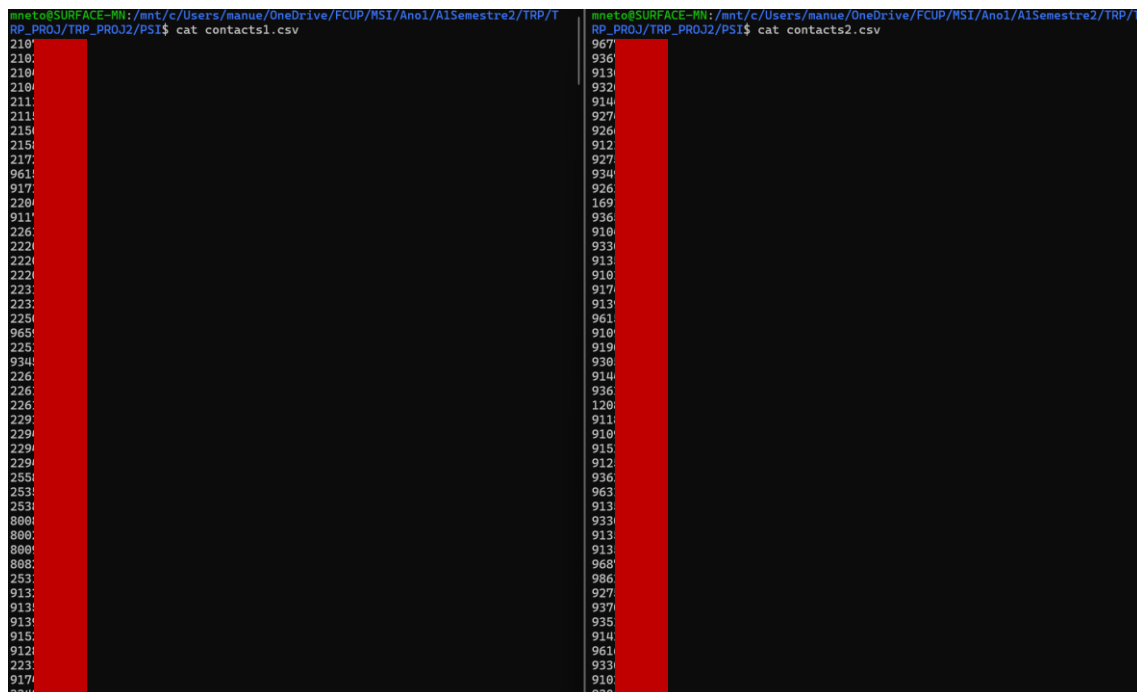


Figura 11 – Datasets para Aplicação

Tendo em conta que o *dataset* escolhido é de contactos telefónicos, constata-se que o domínio no qual um número de telemóvel se pode encontrar é relativamente pequeno, particularmente para contactos maioritariamente portugueses, tendo apenas cerca de 10^8 possibilidades. Como tal, conclui-se que a solução de *naïve hashing* – que faria uso da *hash* de cada número para a partilha de informação – é extremamente insegura nesta situação. Na verdade, este é um exemplo paradigmático de um caso em que este protocolo não é recomendado, devido à facilidade de concretizar um ataque de força bruta no qual se computa a *hash* de todos os números de telefone possíveis. Assim sendo, decide-se não utilizar este protocolo.

Deste modo, o protocolo apoiado pelo servidor surge como uma solução natural, tendo a vantagem de ser igualmente rápido e eficiente, mas com garantias de segurança bastante mais robustas. Assim, ao ter disponível uma terceira parte honesta para testes – o servidor local –, utiliza-se este protocolo para computar a interseção entre os conjuntos privados, conforme se observa na Figura 12.

```

718727e58995903280a7191a5a35fb9a
0e002e88c3e6aa3c7c1281dd373c9ead
2c532809d19c35bcb09479745bda6a4c
219277452e10cd21a33a3e610e546c03
b03a21e5b02e5c6ffe1fb0ad879c2355
3b6c0832664e7eca8f5421ad7263a276
5207dc0c47ac2147e3e9c04b0dd47aa31
fc7ad13a36a5317c2146200eb0b7a831
55a5c39e5182a7f4dada958a7a839c4f
b9dd3a4a9e506f81474fb3d7daa02273
2a918c71ad2c7e807e56aff584f8a11
42173c8c32d7111f86fa2056a08ebdc2
d82d4f3ebd468130a21f9e205ef96ba58
700aa6f87146f3cab7df7f11047aed
764b39947976a7f84ff70ccdb82b5e78
7dd63ef4c40c29c6b79e67a9625543c6
b71036c6c0885f13e38a516a930b347d
e503a5a4de687b4bc178b2933809abf
7596e9b7bbeb87745e18c4f4239b6009
50478b6f95dd0030864c3897367e64c1
09ae18ab016b20b4b2066b22c823d06c
b7a120a4c203d13b0b4f3c07ad30ff45
5333ebc6b0bf241e0e22a06d42c390c
69673f685651ef0a747df33b5585de5c
432c09de2dec724574987a2010f03a
39f4f229d4eb84abc7c4d0b47acada5
1391bb3ec9ef6bacb9771924a460bba
86e2ce95efeb30c2d0bfb90af30986b
8946284683d4706715ae495cd8910afb
8684349d26f64223ff1d63e605f6295d
b1d5dc602fd656c44438f5ae91f3ef88
fa7a216903aaba0b4389a2b6fa31f9a0
174320807f722774108c01a8ad194834
07a80d4150d2ab99fc55728877afcef
91d3cfcdf99493a60bb1acaa70f4720
90254d4d1138784ded14cb3550f664ef
1b6250883aa5f8d4e550c3e8f3b24cef
38ce2f609c3382f8a7b004305227e5d6
1b3a72143292618dbf5de44420c7e73f
343e3695d27423d1945059a2fd7d8ba8
d6d6cee821bb3e801c1b26ebdd2c8a48
17b0ce97fee1a39c94022320877ff2bf
c0758ed940a3fd4d48dbdec93be0de93
Computing intersection for the clients
Inserting the items into the hash table
sending all 4 intersecting elements to the clients

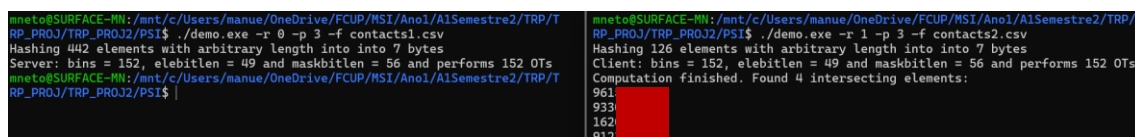
```

Figura 12 – Protocolo Apoiado pelo Servidor

Ora, a Figura 12 mostra os resultados obtidos através da execução do protocolo apoiado pelo servidor para a interseção de conjuntos privados, que resulta em quatro elementos comuns entre os dois conjuntos. Tal como anteriormente, os valores comuns omitem-se parcialmente para tentar preservar a privacidade dos mesmos.

No sentido de diversificar a aplicação prática, pode ainda experimentar-se a execução de outros protocolos, de entre os quatro que são objeto de estudo no presente relatório.

Como tal, pode ponderar-se entre o protocolo baseado em Diffie-Hellman e o apoiado em *oblivious transfer*. Com o estudo realizado anteriormente, verifica-se que ambos os protocolos conferem aproximadamente o mesmo nível de segurança. Contudo, o primeiro precisa de consideravelmente mais tempo do que o segundo para cumprir o mesmo efeito, devido às operações matemáticas computacionalmente dispendiosas que lhe estão subjacentes. Por isso, mesmo que em *datasets* de tamanho tão reduzido a diferença possa não ser tão significativa, opta-se pela aplicação do protocolo baseado em *oblivious transfer*, para efeitos de teste. Este protocolo permite uma computação segura e eficiente da interseção dos conjuntos privados, por todas as razões já explicitadas, tendo a vantagem de não necessitar de uma terceira parte para funcionar corretamente, ao contrário do que sucede no caso anterior. Os resultados obtidos expõem-se na Figura 13.



```

mneto@SURFACE-MN: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/A1Semestre2/TRP/T
RP_PROJ/TRP_PROJ2/PSI$ ./demo.exe -r 0 -p 3 -f contacts1.csv
Hashing 442 elements with arbitrary length into 7 bytes
Server: bins = 152, elebitlen = 49 and maskbitlen = 56 and performs 152 OTs
mneto@SURFACE-MN: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/A1Semestre2/TRP/T
RP_PROJ/TRP_PROJ2/PSI$

mneto@SURFACE-MN: /mnt/c/Users/manue/OneDrive/FCUP/MSI/Ano1/A1Semestre2/TRP/T
RP_PROJ/TRP_PROJ2/PSI$ ./demo.exe -r 1 -p 3 -f contacts2.csv
Hashing 126 elements with arbitrary length into 7 bytes
Client: bins = 152, elebitlen = 49 and maskbitlen = 56 and performs 152 OTs
Computation finished. Found 4 intersecting elements:
961
933
162
912

```

Figura 13 – Protocolo Baseado em Oblivious Transfer

De facto, como anteriormente, este protocolo é capaz de computar a correta interseção entre ambos os conjuntos, resultando nos mesmos quatro contactos telefónicos. Similarmente, omite-se a totalidade dos números de telefone da Figura 13 com o intuito de preservar a privacidade de todas as partes envolvidas.

Assim, tal como esperado, os dois protocolos conseguem computar os dados comuns aos dois conjuntos privados de forma correta e segura, identificando adequadamente os quatro contactos telefónicos comuns entre os *datasets*.

Em suma, a execução destes protocolos de interseção segura entre conjuntos privados permite demonstrar a sua viabilidade e aplicação prática num contexto real.

Conclusão

Finalmente, cumpre refletir sobre todo o estudo realizado e versado neste relatório sobre *Secure Multiparty Computation* para privacidade na prática, em concreto quanto à interseção de conjuntos privados, dividido em quatro principais etapas.

Em primeiro lugar, estudaram-se, descreveram-se e compararam-se os quatro protocolos para a interseção de conjuntos privados. Nesse sentido, analisaram-se as soluções de *naïve hashing*, o protocolo apoiado pelo servidor, a abordagem baseada em Diffie-Hellman e a assente em *oblivious transfer*. Para todos os casos, foi explicado o funcionamento do protocolo e exemplificada a execução do mesmo com a Alice e o Bob, bem como traçadas as principais vantagens e desvantagens, tanto em termos de segurança como relativamente à sua eficiência.

Em segundo lugar, os protocolos estudados foram experimentados para compreender o seu modo de funcionamento na prática. Com isto, explicou-se porque é que a solução de *naïve hashing* pode ser insegura, que problemas de privacidade surgem devido à utilização do servidor como uma terceira parte envolvida na computação da interseção e teceram-se considerações sobre o equilíbrio entre as garantias de privacidade e segurança *versus* o custo comunicacional. Estas experiências práticas permitiram aprofundar e consolidar o conhecimento teórico anteriormente adquirido.

Em terceiro lugar, efetuou-se *benchmarking* de três dos quatro protocolos estudados, de modo a comparar o seu desempenho quanto ao tempo de execução e à quantidade de informação trocada entre as partes. Para tal, apresentaram-se as tabelas e os gráficos relevantes, juntamente com a devida avaliação e análise crítica dos resultados obtidos.

Por último, foram aplicados dois protocolos concretos para um *dataset* escolhido, em particular para determinar os contactos telefónicos comuns entre os dois membros do grupo. Deste modo, verificou-se a importância e a aplicabilidade prática destes protocolos, quando o objetivo é computar a interseção entre dois conjuntos privados de forma segura. Em concreto, utilizou-se o protocolo apoiado pelo servidor e a abordagem apoiada em *oblivious transfer*, por serem os mais adequados aos dados em questão.

Em suma, através destas quatro fases, foi realizado um estudo aprofundado sobre uma área específica inserida no âmbito de *Secure Multiparty Computation*: a interseção de conjuntos privados. Assim, os objetivos do projeto consideram-se cumpridos.