UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

INTERSEÇÃO PRIVADA DE CONJUNTOS COM E SEM TERCEIRA PARTE CONFIÁVEL UTILIZANDO PAILLIER

VITOR VENEZA QUIMAS MACEDO

ORIENTADOR: ANDERSON CLAYTON ALVES NASCIMENTO, Dr.

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA
ÁREA DE CONCENTRAÇÃO INFORMÁTICA FORENSE E SEGURANÇA DA INFORMAÇÃO

PUBLICAÇÃO: PPGENE.DM - 623/16

BRASÍLIA/DF: 12/2016

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

INTERSEÇÃO PRIVADA DE CONJUNTOS COM E SEM TERCEIRA PARTE CONFIÁVEL UTILIZANDO PAILLIER

VITOR VENEZA QUIMAS MACEDO

DISSERTAÇÃO DE MESTRADO PROFISSIONAL SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:

Anderson C A Nascimento

ANDERSON CLAYTON ALVES NASCIMENTO, Dr., UFT
(ORIENTADOR)

Flávio Elias

FLÁVIO ELIAS GOMES DE DEUS, Dr., ENE/UNB
(EXAMINADOR INTERNO)

Dino Macedo Amaral

DINO MACEDO AMARAL , BANCO DO BRASIL
(EXAMINADOR EXTERNO)

Brasília, 05 de Dezembro de 2016.

FICHA CATALOGRÁFICA

MACEDO, VITOR VENEZA QUIMAS

Interseção Privada de Conjuntos com e sem Terceira Parte Confiável utilizando Paillier [Distrito Federal] 2016. x, 62p., 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2016).

Dissertação de Mestrado – Universidade de Brasília, Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1. Private Set Intersection
2. Interseção Privada de Conjuntos
3. Paillier
4. Oblivious Polynomial Evaluation
5. Avaliação Inconsciente de Polinômio
6. Secure Multiparty Computation
7. Computação Segura de Múltiplos Participantes

I. ENE/FT/UnB. II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

MACEDO, V. V. Q. (2016). Interseção Privada de Conjuntos com e sem Terceira Parte Confiável utilizando Paillier. Dissertação de Mestrado, Publicação PPGENE.DM - 623/2016, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 62p.

CESSÃO DE DIREITOS

NOME DO AUTOR: VITOR VENEZA QUIMAS MACEDO
TÍTULO DA DISSERTAÇÃO: Interseção Privada de Conjuntos com e sem Terceira Parte Confiável utilizando Paillier.
GRAU/ANO: Mestre/2016

Vitor Veneza Quimas Macedo
Rua Herotides de Oliveira 121
CEP 24230-230
Niterói/RJ- Brasil

A Deus, pois sem ele nada seríamos.
Aos meus pais pela dedicação
e eterno incentivo ao estudo.

AGRADECIMENTOS

Aos meus PAIS que sempre me incentivaram e que nunca pouparam esforços na educação de seus filhos, "*o saber não ocupa lugar*".

Ao meu orientador, Prof. Dr. ANDERSON CLAYTON ALVES NASCIMENTO, pelo apoio, incentivo, paciência e amizade.

Ao Prof. Flávio Elias Gomes de Deus, do Curso de Engenharia de Redes de Comunicação - Departamento de Engenharia Elétrica, pelo apoio durante este mestrado assim como no contato inicial com o professor Anderson.

Aos meus amigos de sala de aula e de ENAP CIRILO MAX MACEDO DE MORAIS, EGBERTO VILAS BOAS LEMOS FILHO, GUSTAVO HENRIQUE MOREIRA ALVARES DA SILVA e PAULO ROBERTO ROCHA VITORINO, pelo companheirismo e apoio durante todas as semanas que estivemos em Brasília.

Aos Peritos Criminais Federais que tornaram este mestrado uma realidade, em especial, HÉLVIO PEREIRA PEIXOTO, por sua iniciativa em idealizar e viabilizar a primeira turma, o que abriu o caminho para que a segunda turma pudesse se tornar realidade.

O presente trabalho foi realizado com o apoio do Departamento Polícia Federal – DPF, através de esforços do Serviço de Perícias em Informática da Diretoria Técnico Científica, com recursos da Secretaria Nacional de Segurança Pública – SENASP/MJ.

RESUMO

INTERSEÇÃO PRIVADA DE CONJUNTOS
COM E SEM TERCEIRA PARTE CONFIÁVEL
UTILIZANDO PAILLIER

Autor: Vitor Veneza Quimas Macedo
Orientador: Anderson Clayton Alves Nascimento, Dr.
Programa de Pós-graduação em Engenharia Elétrica
Brasília/DF, dezembro de 2016

O objetivo do presente trabalho é provar a segurança e eficiência de um protocolo que implementa a interseção privada de conjuntos (*Private Set Intersection - PSI*) entre dois participantes, Alice e Bob.

Neste protocolo Bob possui um conjunto de elementos e deseja saber se este pertence ao conjunto de pontos de Alice, sem, no entanto, Alice saber nada sobre o resultado e Bob não saber nada além da interseção e o tamanho do conjunto de Alice.

Para tanto é utilizada, numa primeira abordagem, a avaliação inconsciente de polinômio (*Oblivious Polynomial Evaluation*) com o uso de uma Terceira Parte Confiável (TPC), conseguindo obter dessa forma a segurança incondicional, em ambientes estáticos, contra adversários ativos.

Na sequência é apresentada uma segunda versão do protocolo onde Alice e Bob simulam o papel da TPC, através do uso do algoritmo criptográfico homomórfico de Paillier, porém neste caso obtendo segurança computacional, em ambientes estáticos, contra adversário passivos.

Por último é apresentada uma terceira versão do protocolo onde Alice e Bob, sem uma TPC, em uma única fase, usando o algoritmo criptográfico homomórfico de Paillier, em que Bob consegue testar um conjunto com mais do que apenas um elemento, obtendo novamente segurança computacional, em ambientes estáticos, contra adversário passivos

Considerando as pesquisas bibliográficas realizadas, este é o primeiro protocolo que implementa a interseção privada de conjuntos provado incondicionalmente seguro no modelo Composto Universalmente (*Universally Composable - UC*).

ABSTRACT

PRIVATE SET INTERSECTION

WITH AND WITHOUT A TRUSTED THIRD PARTY

APPLYING PAILLIER

Author: Vitor Veneza Quimas Macedo

Supervisor: Anderson Clayton Alves Nascimento, Dr.

Programa de Pós-graduação em Engenharia Elétrica

Brasília, December 2016

The present paper goal is to prove the security and efficiency of a protocol that implements private set intersection – PSI between two players, Alice and Bob.

Bob possess a set of elements and wishes to know if any of these elements belongs also in Alice set of elements, without, however, Alice knowing anything about the intersection and Bob knowing nothing besides the intersection and the size of Alice set.

To conquer this objective, it is used, at a first approach, the oblivious polynomial evaluation using a Trusted Third Party (TTP), which achieves unconditional security, in static environments, against active adversaries.

Moreover, it is presented a second version of the protocol where Alice and Bob simulate the TTP role through the use of Paillier homomorphic cryptography algorithm, but in this scenario it is achieved computational security, in static environments, against passive adversaries.

At last, it is presented a third version of the protocol, without a TTP, still using Paillier homomorphic cryptography algorithm, where it is possible to test Bob set with more than one element, with the same security level as of the second version.

Considering the previous bibliographic research, this is the first protocol that implements private set intersection proved unconditional secure in the Universally Composable - UC - model.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE SÍMBOLOS, NOMENCLATURA E ABREVIAÇÕES

IPC – Interseção Privada de Conjuntos

PSI – *Private Set Intersection*

AIP – Avaliação Inconsciente de Polinômio

OPE - *Oblivious Polynomial Evaluation*

TPC – Terceira Parte Confiável

TTP – *Trusted Third Party*

UC – *Universally Composable*

CMP – Computação de Múltiplas Partes

MPC – *Muliparty Computation*

NTL – *Number Theory Library*

ROM - *Random Oracle Model*

OPRF - *Oblivious Pseudo-Random Functions*

CSS – *Comunicação Síncrona Segura*

$\mathcal{S}$ – Simulador

π – Protocolo

F_R – Funcionalidade ideal de nome *R*

1 - INTRODUÇÃO

Em uma sociedade pós-industrial, conceito introduzido por (Bell, 1976), em que a informação tem um papel protagonista, pessoas e empresas necessitam compartilhar dados privados de forma rotineira.

Em uma abordagem inicial o que vem primeiro a mente é o uso de métodos criptográficos que visam impedir que, no caso de vazamento destes dados privados, o indivíduo, empresa ou governo que tiverem acesso a essa informação cifrada, quando usados métodos robustos, não serão capazes de decifrarem o seu real conteúdo, ou ao menos terão grande dificuldade em ter êxito nesta empreitada.

Entretanto, o assunto afeto ao presente texto, apesar de utilizar métodos criptográficos, visa um objetivo diferente, que está presente num conceito chamado de compartilhamento de segredos (*secret sharing*), que vem a ser como compartilhar informação de modo que participantes compartilhem seus dados privados, a fim de obter um resultado final, mas que a informação privada original de cada um não seja compartilhada para os demais, apenas o resultado final.

Inicialmente o conceito apresentado anteriormente pode parecer um tanto quanto contraditório, porém alguns exemplos apresentados por (Cramer *et al.*, 2015) e (Cristofaro, 2011) esclarecem esse aspecto, objeto de estudo da Computação de Múltiplas Partes (*Muliparty Computation - MPC*).

Um exemplo básico seria imaginar que Alice, Bob e Charlie possuem individualmente um número e desejam saber qual a soma destes três números sem, no entanto, ao final do cálculo terem que revelar aos demais qual o seu número original, sabendo apenas a soma.

Outro exemplo aplicado ao mundo empresarial seria o caso em que as instituições relacionadas a um mesmo ramo de atividade, por exemplo, o automobilístico, desejam saber se o seu lucro está acima ou abaixo da média de mercado.

Neste cenário as empresas não desejam permitir que as outras saibam qual o seu lucro de fato, pois este dado é estratégico, porém se faz necessário para o cálculo da média esta informação oriunda de todas as empresas.

Um cenário adicional seria, por exemplo, em que um órgão de fiscalização possui um suspeito e deseja saber se este trabalha em uma empresa, porém o órgão não possui um mandado judicial que obrigue a empresa a fornecer toda a lista de seus funcionários.

O órgão de fiscalização não deseja que a empresa saiba quem é o suspeito pois, caso este trabalhe na mesma, a informação de que ele está sendo procurado poderia atrapalhar as investigações, porém ao mesmo tempo, a empresa não quer fornecer a sua lista de funcionários.

Uma primeira abordagem seria em que o órgão de fiscalização e a empresa confiam em uma terceira parte, a qual chamamos de Terceira Parte Confiável (TPC), e enviam os seus dados para esta, que por sua vez calcula a interseção e envia a resposta de presença ou não do investigado para o órgão de fiscalização e, em seguida, apaga os dados.

Entretanto, esta abordagem tem algumas desvantagens pois a TPC possui, em um dado momento do tempo, os dados de ambas as partes e caso ocorra algum vazamento, tanto a identidade do investigado quanto a lista de funcionários seriam comprometidas de uma só vez.

Outro aspecto é poder não existir uma entidade que seja confiável para todas as partes ao mesmo tempo. Este caso pode ocorrer quando imaginamos, por exemplo, países como os Estados Unidos, China, Rússia e Suíça.

Felizmente é possível atingir o resultado desejado de se calcular essa interseção sem utilizar uma TPC, sendo ambos os casos, com e sem uma TPC, objetivo de protocolos que implementam a interseção privada de conjuntos ou IPC (*Private Set Intersection - PSI*), sendo esse o objetivo do presente trabalho.

1.1 - JUSTIFICATIVA

O exemplo apresentado na seção anterior é afeto a realidade tanto da Policia Federal, quanto de demais órgãos de fiscalização, como Receita Federal, Agência de Vigilância Sanitária além de outros.

Este pode ser estendido para o caso de listas de IPs de suspeitos de fraudar sistemas bancários, participar de grupos relacionados a pornografia infantil e outros crimes e infrações previstas em lei.

1.2 - OBJETIVO

O objetivo do presente trabalho é propor um protocolo que implemente IPC, provar a sua segurança e medir a sua performance.

Para tanto será usado inicialmente o método apresentado por (Tonicelli *et al.*,2015) o qual propõe um protocolo que implementa a avaliação inconsciente de polinômio (*Oblivious Polynomial Evaluation - OPE*) com o uso de uma Terceira Parte Confiável (TPC) entre dois

participantes, Alice e Bob, em que é permitido a Bob testar a interseção de apenas um elemento no conjunto de Alice.

Na sequência são propostas alterações a fim de eliminar a necessidade de uma TPC e por último é proposto um protocolo que além de não necessitar de uma TPC permite ainda a Bob testar um número ilimitado de elementos em uma única rodada do protocolo.

1.3 - METODOLOGIA

O protocolo IPC proposto foi implementado utilizando dois computadores pessoais com as seguintes configurações:

a) Equipamento A:

- Processador Intel Core i7-2677M 1.80GHz;
- 4 GB de memória RAM;
- Disco SSD Sandisk U100 256GB;
- Sistema Operacional Ubuntu 16.04 LTS;
- Biblioteca NTL versão 9.9.1;
- Biblioteca GMP versão 6.1.0;
- Biblioteca Boost versão 1.61.0;
- Compilador c++ (Ubuntu 5.3.1-14ubuntu2.1) 20160413.

b) Equipamento B:

- Processador Intel Core i5-4300U 1.90GHz;
- 8 GB de memória RAM;
- Disco SSD Toshiba THNSNJ256GCST 256GB;
- Sistema Operacional Ubuntu 16.04 LTS;
- Biblioteca NTL versão 9.9.1;
- Biblioteca GMP versão 6.1.0;
- Biblioteca Boost versão 1.61.0;
- Compilador c++ (Ubuntu 5.3.1-14ubuntu2.1) 20160413.

O modelo matemático para as provas de segurança do algoritmo são aqueles referentes ao modelo de Composto Universalmente (*Universally Composable* - UC), apresentado por (Cramer *et al.*, 2015).

1.4 - RESULTADOS ESPERADOS

Espera-se mostrar ao final do presente trabalho os diferentes tempos de execução entre as variações do protocolo IPC propostas, considerando diferentes tamanhos de conjuntos de Alice, assim como provar sua segurança.

1.5 - LIMITAÇÕES

Devido ao fato de que o método para implementar IPC é através da avaliação inconsciente de polinômio, proposto por (Tonicelli *et al.*,2015), o mesmo necessita que os conjuntos sejam formados apenas por números, não podendo fazer a comparação de *strings*, por exemplo.

Esta limitação pode ser contornada através do uso de números como CPF para representarem pessoas, ou a utilização de números IP na sua base decimal considerando os 32 bits da versão IPv4 cuja limitação superior é 4294967296.

Outro aspecto que cabe ressaltar é que o protocolo proposto ocorre entre duas partes apenas, não permitindo a execução para um conjunto maior de participantes.

Neste cenário é necessária a execução de múltiplas rodadas do protocolo com os diferentes participantes.

1.6 - ORGANIZAÇÃO

O restante do presente texto está organizado da seguinte forma:

- Capítulo 2: mostra a pesquisa bibliográfica referente ao tema deste trabalho;
- Capítulo 3: faz uma introdução aos conceitos que serão utilizados ao longo do texto e necessários, por exemplo, para a construção das provas de segurança;
- Capítulo 4: apresenta a primeira versão do protocolo que implementa IPC através do uso de uma TPC, com uma fase *off-line* e uma fase *online*;
- Capítulo 5: apresenta a segunda versão do protocolo que implementa IPC eliminando o uso de uma TPC, ainda com duas fases;

- Capítulo 6: apresenta a terceira versão do protocolo que implementa IPC sem uma TPC, com uma única fase *online*, onde Bob pode testar mais do que um elemento;
- Capítulo 7: exibe os resultados de performance das diferentes versões do protocolo e faz uma análise comparativa dos resultados;
- Capítulo 8: são apresentadas as conclusões, contribuições e sugestões para trabalhos futuros.

2 - ESTUDOS CORRELATOS

O conceito de Terceira Parte Confiável (TPC) apresentado anteriormente necessita que todos os participantes confiem na TPC durante toda a execução do protocolo. Entretanto (Beaver, 1997) introduziu o modelo de Criptografia Baseada em Commodity (*Commodity-Based Cryptography*) que é uma alternativa para essa abordagem inicial.

Neste modelo os participantes adquirem valores iniciais, ou commodities, de servidor(es) numa fase *off-line*, que é(são) visto(s) como uma TPC. A partir do recebimento destas commodities não mais é necessária qualquer interação dos participantes com a mesma. Vale ressaltar que neste modelo a TPC não recebe e, portanto, não sabe, nenhuma informação dos participantes, eliminando o risco de vazamento de informações destes.

A introdução do conceito e implementação para avaliação inconsciente de polinômio foi proposta por (Naor *et al.*, 1999), porém, se baseia na suposição de interpolação ruidosa de polinômios (*Noisy Polynomial Interpolation*), cuja dificuldade de resolução é um problema em aberto.

Uma outra abordagem foi apresentada por (Tonicelli *et al.*, 2015) para implementar a Avaliação Inconsciente de Polinômio, na qual utiliza o conceito de Criptografia Baseada em Commodity para construir um protocolo que possui segurança teórica e, portanto, seguro contra adversários com um poder computacional ilimitado.

O conceito de Interseção Privada de Conjuntos foi introduzido por (Freedman *et al.*, 2004) onde foi proposto um protocolo que utiliza a Avaliação Inconsciente de Polinômio com o uso de criptografia homomórfica, como por exemplo (Paillier, 1999). Este protocolo é seguro contra adversário ativos considerando o Modelo do Oráculo Aleatório (*Random Oracle Model - ROM*) apresentado em (Bellare *et al.*,1993).

Uma melhoria em relação a este último foi proposta por (Hazay *et al.*,2010), onde o protocolo é seguro contra adversários ativos sem, no entanto, recorrer ao Modelo do Oráculo Aleatório. Porém neste caso se faz necessário o uso de provas de conhecimento-zero (*zero-knowledge proofs*)

(Kissner *et al.*,2005) também apresentaram um protocolo para Avaliação Inconsciente de Polinômio podendo ter mais do que dois participantes, entretanto se baseia em provas de conhecimento-zero genéricas (*generic zero-knowledge proofs*) que apresentam alto custo

computacional, onde posteriormente foi proposta uma melhoria por (Dachman-Soled *et al.*,2009) para evitar o uso dessas provas custosas.

(Cristofaro, 2011) apresenta um extenso trabalho sobre protocolos que implementam Interseção Privada de Conjuntos com o objetivo de comparar a complexidade computacional de abordagens que utilizam diferentes premissas criptográficas.

Uma nova contribuição é feita em (Cristofaro *et al.*, 2012) cuja ferramenta base é a Avaliação Inconsciente de Funções Pseudo-Aleatórias (*Oblivious Pseudo-Random Functions – OPRF*), sendo que neste trabalho são apresentados resultados de performance, onde é argumentado que foram obtidos melhores resultados que aqueles apresentados em (Huang *et al.*,2012).

Recentes contribuições foram feitas por (Dong *et al*., 2013), (Kamara *et al*., 2014) e (Pinkas *et al*., 2014), sendo que este último utiliza os recursos de filtro de Bloom (*Bloom Filter*) para otimizar o protocolo.

O presente trabalho visa também o objetivo de melhoria de performance através do uso de uma fase *off-line* para melhorar o tempo de execução da fase *online* com os conceitos apresentados por (Beaver, 1997), (Freedman *et al*., 2004) e (Tonicelli *et al.*, 2015).

3 - CONCEITOS E DEFINIÇÕES

Nesta seção é feita uma introdução aos conceitos que serão utilizados ao longo do texto e necessários, por exemplo, para a construção das provas de segurança

3.1 - ARITMÉTICA MODULAR

De maneira simplificada, na aritmética modular os cálculos são feitos da seguinte forma:

$$Sejam\ a, q, r, N\ \in\ \mathbb{Z}, onde\ N > 1 \quad \text{(EQ 3.1)}$$

$$Existem\ q\ e\ r\ \text{únicos}\ tais\ que\ a = qN + r, onde\ 0 \leq r < N \quad \text{(EQ 3.2)}$$

$$Definimos\ [a\ mod\ N] = r \quad \text{(EQ 3.3)}$$

Desta forma, $[a\ mod\ N]$ é o resto da divisão de a por N, sendo que cálculos referentes a aritmética modular possuem propriedades como:

$$[a\ mod\ N] + [b\ mod\ N] = [a + b\ mod\ N] \quad \text{(EQ 3.4)}$$

$$[a\ mod\ N] - [b\ mod\ N] = [a - b\ mod\ N] \quad \text{(EQ 3.5)}$$

$$[a\ mod\ N] * [b\ mod\ N] = [a * b\ mod\ N] \quad \text{(EQ 3.6)}$$

Para maiores aprofundamentos recomendamos a leitura de (Katz *et al.*, 2015), capítulo 8.

3.2 - BIBLIOTECA NTL

NTL (*Number Theory Library*) é uma biblioteca para realização de cálculos referentes a teoria de números para compiladores em C++, desenvolvida e mantida por Victor Shoup (Shoup, 2016).

Os recursos como multiplicações, adições e exponenciações modulares otimizadas e geração de números primos aleatórios, presentes nesta biblioteca, foram maciçamente utilizados na implementação dos protocolos propostos neste trabalho.

3.3 - ALGORITMOS CRIPTOGRÁFICOS DE CHAVE PÚBLICA (ASSIMÉTRICOS)

O surgimento dessa categoria de algoritmos marcou uma revolução no ramo da criptografia, uma vez que, ao contrário de algoritmos criptográficos simétricos, que utilizam a mesma chave para cifrar e decifrar uma mensagem, algoritmos criptográficos de chave pública, ou assimétricos, possuem duas chaves, chamadas de chave pública e chave privada.

Ao utilizar um algoritmo simétrico duas pessoas necessitam compartilhar previamente a chave a ser utilizada através de um canal seguro, o que muitas vezes não é uma opção viável.

Em um cenário em que Alice e Bob decidam utilizar um algoritmo de chave pública, Alice simplesmente gera um par de chaves (pk, sk), pública e privada, respectivamente, e envia a chave pk para Bob, em um canal que pode ser completamente aberto e público.

A partir deste momento Bob pode passar a enviar mensagens para Alice cifrando com a chave pk e Alice utiliza a chave sk para decifrar.

Repare que neste cenário mesmo que um ouvinte mal-intencionado, que queira decifrar as mensagens, e aprenda a chave pk, não será bem sucedido pois apenas Alice possui a chave sk e a mesma não foi enviada e, portanto, revelada para nenhum outro participante.

Para maiores aprofundamentos recomendamos a leitura de (Katz *et al.*, 2015), capítulos 8, 9, 10 e 11.

3.4 - ALGORITMO CRIPTOGRÁFICO DE PAILLIER

É um algoritmo criptográfico de chave pública proposto por (Paillier, 1999), que possui a propriedade de ser aditivo homomórfico.

Algoritmos criptográficos com essa propriedade permitem que operações sejam feitas diretamente nos dados cifrados.

O algoritmo criptográfico de chave-pública de Paillier possui um domínio em módulo $N = pq$ onde os componentes da chave privada são primos p e q de mesmo tamanho e onde $mdc\big(pq, (p-1) * (q-1)\big) = 1$.

A chave pública é composta por $(N, g), onde\ g \in \mathbb{Z}^*_{N^2}$. Valores $a \in \mathbb{Z}^*_F$ são números inteiros cujo $mdc(a, F) = 1, onde\ a \in \{1, \dots, F-1\}$.

A chave privada é:

$$\lambda = mmc(p-1, q-1) \qquad \text{(EQ 3.7)}$$

A função de cifragem é definida como $E(m, r)\ onde\ m \in \mathbb{Z}_N$ e $r \in \mathbb{Z}^*_N$ é $um\ valor\ aleatório$. Então a mensagem cifrada é definida como:

$$c = E(m, r) = g^m r^N\ mod\ N^2 \qquad \text{(EQ 3.8)}$$

A função de decifragem é definida como:

$$D(c) = \frac{L(c^{\lambda} \bmod N^2)}{L(g^{\lambda} \bmod N^2)} \bmod N \quad \text{(EQ 3.9)}$$

$$onde\ L(x) = \frac{x-1}{N}\ \text{é}\ calulado\ como\ uma\ divisão\ inteira \quad \text{(EQ 3.10)}$$

A fim de ser seguro, o comprimento de N deve ser de pelo menos 2048 bits, ou seja, p e q devem ter no mínimo 1024 bits, como pode ser visto em (Giry, 2016).

A propriedade homomórfica deste algoritmo é apresentada a seguir:

$$E(m_1 + m_2, r_1 \cdot r_2) = E(m_1, r_1) \cdot E(m_2, r_2) \quad \text{(EQ 3.11)}$$

Sendo assim, é possível efetuar a soma de valores através da multiplicação de seus respectivos conteúdos cifrados, sem necessitar de qualquer acesso aos dados em aberto.

Esta propriedade é essencial na construção da versão do protocolo em que não é utilizada uma Terceira Parte Confiável - TPC.

3.5 - TEOREMA CHINÊS DOS RESTOS

O Teorema Chinês dos Restos (*Chinese remainder theorem - CRT*) é um resultado da teoria de números que encontra várias aplicações em criptografia.

Ele afirma que um conjunto de equações na forma:

$$x = a_i \, mod \, m_i \, , i \in \{1, \dots, k\} \quad \text{(EQ 3.12)}$$

Onde todos m_i são primos ente si, dois a dois, tem uma solução única para x módulo $M = m_1 \cdot \dots \cdot m_k$. E a solução é dada por:

$$x = \sum_{i=1}^{k} a_i \cdot b_i \cdot \frac{M}{m_i} \, mod \, M \quad \text{(EQ 3.13)}$$

$$\text{Onde } b_i = \left(\frac{M}{m_i}\right)^{-1} mod \, m_i \quad \text{(EQ 3.14)}$$

Ele é comumente utilizado para reduzir a complexidade dos cálculos módulo M, quando este é muito grande, através de cálculos módulo seus fatores m_i.

Esta abordagem é uma das melhorias utilizadas para implementar uma versão otimizada do algoritmo homomórfico de Paillier.

3.6 - IMPLEMENTAÇÃO OTIMIZADA DE PAILLIER

A funções de cifragem e decifragem do algoritmo de Paillier utilizadas no presente trabalho foram implementadas seguindo as otimizações sugeridas no Capítulo 2 de (Pullonen *et al.*, 2012).

A seguir apresentamos em detalhes as respectivas otimizações. O código fonte está presente no Anexo A ao final do presente texto.

Para a função de decifragem é possível calcular de forma antecipada e armazenar o valor de $L(g^{\lambda} \, mod \, N^2)^{-1} \, mod \, N$.

Assim como é possível reduzir a complexidade da função auxiliar $L(x)$, apresentada na equação (EQ 3.7) definindo a mesma como:

$$L(x) = (x - 1) \cdot u \, mod \, 2^{|N|} \quad \text{(EQ 3.15)}$$

$$onde \, |N| \text{ é } o \, comprimento \, em \, bits \, de \, N. \quad \text{(EQ 3.16)}$$

$$u = N^{-1} \, mod \, 2^{|N|} \, e \, pode \, ser \, calculado \, de \, forma \, antecipada \quad \text{(EQ 3.17)}$$

Entretanto, é possível reduzir ainda mais o custo computacional do cálculo de $L(x)$ utilizando os primos p e q juntamente com o Teorema Chinês dos Restos apresentado na seção 3.5. Desta forma são definidas duas novas funções auxiliares, a saber:

$$L_p(x) = \frac{x-1}{p} = (x-1) \cdot u_p \; mod \; 2^{|p|} \quad \text{(EQ 3.18)}$$

$$L_q(x) = \frac{x-1}{q} = (x-1) \cdot u_q \; mod \; 2^{|q|} \quad \text{(EQ 3.19)}$$

$$u_p = p^{-1} \; mod \; 2^{|p|} \; e \; u_q = q^{-1} \; mod \; 2^{|q|} \quad \text{(EQ 3.20)}$$

Então as constantes para decifragem também podem ser calculadas de forma antecipada

$$h_p = L_p(g^{p-1} \; mod \; p^2)^{-1} \; mod \; p \quad \text{(EQ 3.21)}$$

$$h_q = L_q(g^{q-1} \; mod \; q^2)^{-1} \; mod \; q \quad \text{(EQ 3.22)}$$

A função de decifragem apresentada na equação (EQ 3.9) é então desmembrada através dos seguintes cálculos:

$$m_p = L_p(c^{p-1} \; mod \; p^2) \cdot h_p \; mod \; p \quad \text{(EQ 3.23)}$$

$$m_q = L_q(c^{q-1} \; mod \; q) \cdot h_q \; mod \; q \quad \text{(EQ 3.24)}$$

A mensagem final m é então calculada a partir de m_p e m_q através da resolução do Teorema Chinês dos Restos.

Para a função de cifragem, conforme sugerido por (Damgård *et al.*, 2001) definimos $g = N + 1$ o que permite o cálculo da equação (EQ 3.8) como:

$$c = E(m, r) = (Nm + 1)r^N \; mod \; N^2 \quad \text{(EQ 3.25)}$$

Para o caso em que o dono da chave privada necessita realizar cifragem de mensagens, então é possível utilizar novamente o conhecimento de p e q para usar o Teorema Chinês dos Restos também neste cálculo, através de:

$$c_p = (Nm + 1)r^N \; mod \; p^2 \quad \text{(EQ 3.26)}$$

$$c_q = (Nm + 1)r^N \; mod \; q^2 \quad \text{(EQ 3.27)}$$

Como antes, c_pe c_q são combinados através do Teorema Chinês dos Restos.

3.7 - MODELO UC

O Modelo UC (*Universally Composable - UC*), Composto Universalmente, é um arcabouço utilizado para realizar provas de segurança de protocolos.

As definições e conceitos utilizados no presente texto são aquelas definidas no Capítulo 4 de (Cramer *et al.*, 2015) e iremos apresentar aqui os aspectos principais, não sendo o objetivo apresentar todas as definições.

O ponto de partida no Modelo UC, que é necessário ter sempre em mente, é que não é possível afirmar que um protocolo é seguro por si só, de forma absoluta.

Podemos fazer um paralelo desta visão com aquela utilizada para se mensurar o grau de dureza de mineiras na Escala de Mohs, onde o diamante possui a escala 10. Nesta metodologia, portanto, a escala de dureza é comparativa.

Desta forma podemos falar que um protocolo π é sempre tão seguro quanto alguma referência. Porém dada a infinidade de protocolos que podem ser construídos, como poderíamos encontrar tal referência para servir de base comparativa?

É neste momento que introduzimos o conceito de Funcionalidade Ideal, a funcionalidade F.

Devemos imaginar qual deveria ser o comportamento ideal que o protocolo π almeja implementar sem nos preocuparmos, a grosso modo, com recursos computacionais ou outras restrições, e então definimos este funcionamento como sendo aquele da funcionalidade F.

Chegamos então ao conceito básico em que comparamos o protocolo π a funcionalidade ideal F.

O conceito de segurança é definido comparativamente quando um dado observador $\mathcal{Z}$ tenta distinguir, dado um conjunto de suposições, entre o funcionamento de π e de F. Caso $\mathcal{Z}$ não seja capaz de distinguir entre π e F, com algumas considerações, então dizemos que π e tão seguro quanto F.

As exposições apresentadas até agora são uma versão muito simplificada e necessitamos agora acrescentar outros conceitos para que possamos atingir um maior grau de detalhamento matemático.

Em primeiro lugar um protocolo, via de regra, trabalha considerando a existência de outros protocolos já construídos. Podemos citar como exemplo um protocolo de camada de aplicação, como o HTTP ou π_{HTTP}, que utiliza como base as funcionalidades do protocolo TCP ou F_{TCP}.

Sendo assim um protocolo opera utilizando um conjunto de recursos já construídos, e chamamos essa composição de $\pi_{HTTP} \diamond F_{TCP}$.

Quando o observador $\mathcal{Z}$ tenta fazer a distinção do protocolo, ele na verdade enxerga o conjunto $\pi_{HTTP} \diamond F_{TCP}$ e não somente π_{HTTP}.

Porém a composição de $\pi_{protocolo} \diamond F_{recurso}$ possui um conjunto de portas de entrada e saída de dados, assim como um conjunto de portas especiais, diferentes daquelas disponíveis na funcionalidade ideal $F_{protocolo}$. Essas portas são através das quais são recebidas e enviadas mensagens durante a execução do protocolo.

Devemos então lembrar que para um protocolo ser seguro o observador $\mathcal{Z}$ não deve ser capaz de distinguir entre a composição $\pi_{protocolo} \diamond F_{recurso}$ e a funcionalidade ideal $F_{protocolo}$, mas como isso seria possível se o conjunto de portas de entrada e saída são diferentes?

Para resolver esta questão é introduzido o conceito do Simulador $\mathcal{S}$, que vem a ser um intermediário que é acoplado a funcionalidade ideal $F_{protocolo}$ de forma a criar a composição $F_{protocolo} \diamond S$. Desta maneira o observador $\mathcal{Z}$ compara, por fim, $\pi_{protocolo} \diamond F_{recurso}$ e $F_{protocolo} \diamond S$. Caso não seja possível realizar a distinção então escrevemos que:

$$\pi_{protocolo} \diamond F_{recurso} \overset{\mathcal{Z}}{\equiv} F_{protocolo} \diamond \mathcal{S} \qquad \text{(EQ 3.28)}$$

Provar a segurança do protocolo $\pi_{protocolo}$ tem então como etapa essencial ser capaz de construir este Simulador $\mathcal{S}$ de forma a não permitir ao observador $\mathcal{Z}$ distinguir entre as duas composições apresentadas na equação (EQ 3.28).

Provas de segurança quando todos os participantes são honestos e seguem à risca o que o protocolo determina são normalmente muito simples e diretas, até porque protocolos que não funcionam nestas condições, na verdade, não tem grandes utilidades, se é que possuem alguma.

Sendo assim é necessário vislumbrar os casos em que os participantes não seguem o protocolo e onde detalhamos como isto ocorre e é tratado no Modelo UC.

O primeiro aspecto de corrupção é aquele em que os participantes seguem o protocolo mas tentam obter mais informações do que aquelas originalmente projetadas, neste caso os chamamos de adversários passivos. Quando o participante, além disso, passa a poder trocar dados em suas entradas e saídas, desviando completamente do protocolo, temos os adversários ativos.

O segundo aspecto da modelagem da corrupção é saber qual ou quais participantes serão corrompidos durante a execução do protocolo. Quando sabemos de forma antecipada da execução do protocolo qual o conjunto de participantes será corrompido, temos segurança estática. Quando não sabemos essa informação e qualquer participante pode ser corrompido a qualquer momento durante a execução do protocolo, temos segurança adaptativa.

Ao observador $\mathcal{Z}$, portanto, são permitidos certos poderes, pois a ele é dado a opção de poder corromper, seja passiva ou ativamente, ou estática ou adaptativamente, um conjunto de participantes, a fim de tentar distinguir entre $\pi_{protocolo} \diamond F_{recurso}$ e $F_{protocolo} \diamond \mathcal{S}$.

Também é possível definir qual o poder computacional que $\mathcal{Z}$ terá ao tentar realizar a tarefa de distinção. Caso $\mathcal{Z}$ tenha poderes computacionais de tempo polinomial e este não é capaz de realizar a distinção, temos segurança computacional, então escrevemos:

$$\pi_{protocolo} \diamond F_{recurso} \overset{comp}{\equiv} F_{protocolo} \diamond \mathcal{S} \qquad \text{(EQ 3.29)}$$

De forma mais rigorosa, $\mathcal{Z}$ representa na verdade uma classe de ambientes, então caso $\mathcal{Z}$ tenha poder computacional ilimitado e for capaz de distinguir entre $\pi_{protocolo} \diamond F_{recurso}$ e $F_{protocolo} \diamond S$ apenas com uma vantagem negligenciável, temos segurança incondicional, então podemos escrever:

$$\pi_{protocolo} \diamond F_{recurso} \overset{stat}{\equiv} F_{protocolo} \diamond \mathcal{S} \qquad \text{(EQ 3.30)}$$

Caso $\mathcal{Z}$ tenha zero vantagem em distinguir as composições então escrevemos:

$$\pi_{protocolo} \diamond F_{recurso} \overset{perf}{\equiv} F_{protocolo} \diamond \mathcal{S} \qquad \text{(EQ 3.31)}$$

Caso a segurança seja provada apenas contra adversários passivos, escrevemos:

$$\mathcal{Z} \in Env^{\,passivo} \qquad \text{(EQ 3.32)}$$

Caso a segurança seja provada apenas para ambientes estáticos escrevemos:

$$\mathcal{Z} \in Env^{\,estático} \qquad \text{(EQ 3.33)}$$

Caso a segurança seja provada apenas para ambientes com tempo recursivo polinomial escrevemos:

$$\mathcal{Z} \in Env^{poli} \quad \text{(EQ 3.34)}$$

Sendo assim, agora podemos apresentar a definição de Segurança para Protocolos no Modelo UC, que é descrita a seguir, conforme Definição 4.19 de (Cramer *et al.*, 2015):

$$Seja\ F_F\ uma\ funcionalidade\ ideal\ de\ nome\ F \quad \text{(EQ 3.35)}$$

$$Seja\ \pi_F\ um\ protocolo\ de\ nome\ F\ que\ usa\ um\ recurso\ de\ nome\ R \quad \text{(EQ 3.36)}$$

$$Seja\ F_R\ uma\ funcionalidade\ ideal\ de\ nome\ R \quad \text{(EQ 3.37)}$$

$$Seja\ Env\ uma\ classe\ de\ ambientes \quad \text{(EQ 3.38)}$$

$$Dizemos\ que\ \pi_F \diamond F_R\ implementa\ seguramente\ F_F \quad \text{(EQ 3.39)}$$

$$Se\ existe\ um\ Simulador\ \mathcal{S} \in Sim\ para\ \pi_F, tal\ que \quad \text{(EQ 3.40)}$$

$$\pi_F \diamond F_R \underset{\equiv}{^{Env}} F_F \diamond \mathcal{S} \quad \text{(EQ 3.41)}$$

4 - PROTOCOLO IPC COM TPC - VERSÃO 1

Esta primeira versão do protocolo visa implementar a Interseção Privada de Conjuntos utilizando uma Terceira Parte Confiável. Conforme já exposto no Capítulo 2, esta implementação se baseia na proposta apresentada no capítulo 4 de (Tonicelli *et al.*, 2015), que exibe um algoritmo que implementa a avaliação inconsciente de polinômio.

Os participantes são nomeados, classicamente, como Alice, que possui um conjunto de elementos e Bob, que possui um elemento a ser testado no conjunto de Alice. A Terceira Parte Confiável – TPC é chamada de Ted.

O objetivo ao final da execução do protocolo é Bob saber se o seu elemento está contido no conjunto de Alice e, como efeito colateral, também saber o tamanho do conjunto de Alice e nada mais.

Sendo assim, as seguintes invariantes devem permanecer verdadeiras durante toda a execução do protocolo:

a) Alice não aprende qualquer dado sobre o elemento de Bob;
b) Alice não aprende se o elemento de Bob pertencente ao seu conjunto;
c) Bob não aprende qualquer dado sobre nenhum dos elementos do conjunto de Alice;
d) Um observador externo não é capaz de obter nenhum dado de Alice ou Bob;

Na Figura 4.1 reapresentamos o algoritmo proposto por (Tonicelli *et al.*, 2015) para esta versão 1 do protocolo.

Protocolo IPC Com TPC - Versão 1
Fase Off-line
1. Ted cria aleatoriamente um polinômio $s(x)$ de grau n e um ponto d, ambos $\in \mathbb{F}_q$; 2. Ted envia os coeficientes de $s(x)$ para Alice; 3. Ted envia o ponto d e $g = s(d)$ para Bob;
Fase Online
1. Parâmetros de entrada: 1.1. Alice possui um conjunto de elementos $\mathbb{A}$, que são as raízes do polinômio $p(x)$ de grau n; 1.2. Bob possui um elemento x_0; 2. Bob calcula $t = x_0 - d$ e envia t para Alice; 3. Alice calcula $f(x) = p(x + t) + s(x)$ e envia $f(x)$ para Bob; 4. Bob calcula $r = f(d) - g$: 4.1. Se $r = 0$ então $x_0 \in \mathbb{A}$; 4.2. Caso contrário $x_0 \notin \mathbb{A}$;

Figura 4.1: Protocolo IPC com TPC - Versão 1

Vale aqui pontuar que o objetivo de protocolos que possuem duas fases, uma off-line e um online, é que a primeira seja capaz de acelerar a execução da segunda, quando comparado com um protocolo com o mesmo objetivo que, porém, execute numa única fase.

A fase off-line pode ser executada num momento anterior em que, por exemplo, o consumo de processamento de servidores e de banda da rede seja reduzido como em períodos noturnos ou finais de semana.

A fim de exemplificar o acima exposto imaginemos dois protocolos denominados π_A e π_B que implementam a exata mesma funcionalidade e, portanto, têm o mesmo objetivo. A diferença entre ambos é que π_A executa em uma única fase online enquanto π_B executa em duas fases, uma off-line e um online.

Suponhamos então que π_A leve $online_A$ segundos para completar a execução de sua fase única e que π_B leve $offline_B$ e $online_B$ para executar suas fases off-line e online, respectivamente.

Podemos dizer então que π_B cumpriu seu papel em acelerar a execução da fase online se $online_B < online_A$, mesmo que o tempo total de execução seja maior, ou seja, $offline_B + online_B \geq online_A$.

Repare que caso $online_B > online_A$ o protocolo π_B que tinha como propósito acelerar a execução da fase online através de uma fase off-line falhou completamente em seu objetivo, sendo melhor utilizar o protocolo π_A neste caso.

4.1 - PROVA DE CORRETUDE

A demonstração de que o protocolo apresentado é correto é obtida diretamente através da análise do resultado final.

Conforme definição no item 4 da Fase Online da Figura 4.1:

$$r = f(d) - g \quad \text{(EQ 4.1)}$$

Conforme definição no item 3 da Fase Off-line da Figura 4.1, $g = s(d)$, temos então que:

$$r = f(d) - s(d) \quad \text{(EQ 4.2)}$$

Conforme definição no item 3 da Fase Online da Figura 4.1, $f(x) = p(x + t) + s(x)$:

$$r = p(d + t) + s(d) - s(d) \quad \text{(EQ 4.3)}$$

Conforme definição no item 2 da Fase Online da Figura 4.1, $t = x_0 - d$ e, portanto, $x_0 = d + t$, sendo assim:

$$r = p(x_0) \quad \text{(EQ 4.4)}$$

Conforme apresentado no item 1.1 da Fase Off-line da Figura 4.1, as raízes do polinômio $p(x)$ são os elementos do conjunto $\mathbb{A}$ de Alice e, dessa forma, caso $x_0 \in \mathbb{A}$ então $p(x_0)$ resultará em zero, caso contrário, num valor diferente de zero, provando assim que o protocolo está correto.

4.2 - PROVA DE SEGURANÇA NO MODELO UC

Teorema 1. O Protocolo IPC com TPC - Versão 1 é Seguro no Modelo UC, com segurança estática, para adversários ativos.

A fim de provarmos o Teorema 1 precisamos das definições a seguir:

a) F_{IPC} é a funcionalidade ideal de nome IPC, Interseção Privada de Conjuntos, conforme Figura 4.2;

b) F_{TPC} é a funcionalidade ideal de nome TPC, Terceira Parte Confiável, conforme Figura 4.4;

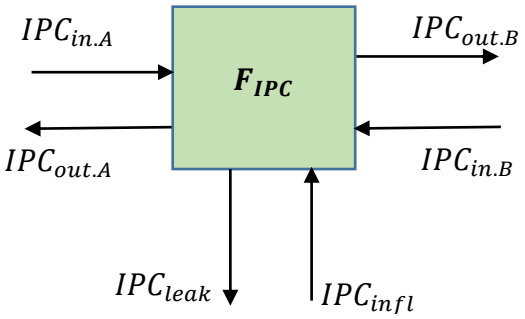
c) F_{CSS} é a funcionalidade ideal de nome CSS, Comunicação Síncrona Segura, conforme Figura 4.5;

d) π_{IPC-1} é o protocolo de nome IPC-1 que utiliza os recursos de nomes TPC e CSS, conforme Figura 4.6;

e) $Env^{estático}$[1] é a classe de ambientes;

[1] Vide definição (EQ 3.33).

Agente F_{IPC} – Funcionalidade Ideal - Interseção Privada de Conjuntos



Comportamento

O comportamento em casos de corrupção passiva e ativa para funcionalidades ideais está descrito na Figura 4.3;

1. Ao receber uma mensagem (*test*, p_n) na porta $IPC_{in.A}$:
   1.1. Armazene (*test*, A, p_n);
   1.2. Envie a mensagem (mid, A, n) na porta IPC_{leak};
   1.3. Caso haja uma mensagem (*test*, B, x_0) armazenada:
      1.3.1. Envie a mensagem (*result*, $p(x_0)$) na porta $IPC_{out.B}$;
2. Ao receber uma mensagem (*test*, x_0) na porta $IPC_{in.B}$:
   2.1. Armazene (*test*, B, x_0);
   2.2. Envie a mensagem (mid, B) na porta IPC_{leak};
   2.3. Caso haja uma mensagem (*test*, A, p_n) armazenada:
      2.3.1. Proceda conforme 1.3.1;

Figura 4.2: F_{IPC}

Comportamento Padrão para Corrupção em Funcionalidades Ideais

O comportamento em casos de corrupção passiva e ativa é o mesmo comportamento padrão para funcionalidades ideais descrito na página 63 de (Cramer *et al.*, 2015), seção 4.2.2;

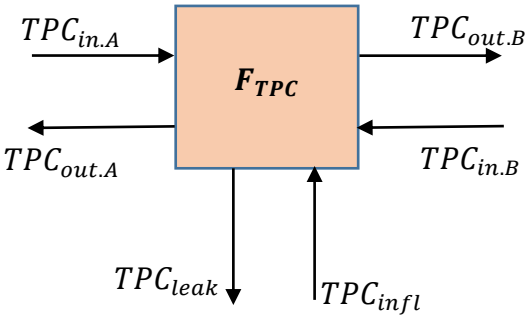
1. Ao receber uma mensagem (passive corrupt, *i*) na porta F_{infl}:
   1.1. Envie a mensagem (state, *i*, σ) na porta F_{leak};
      1.1.1. Onde σ é chamado de estado interno ideal do participante *i* e é definido como sendo todas as entradas prévias recebidas na porta $F_{in.i}$ juntamente com todas as entradas que estão na fila da porta $F_{in.i}$, além de todas as saídas enviadas na porta $F_{out.i}$;
2. Ao receber uma mensagem (active corrupt, *i*) na porta F_{infl}:
   2.1. Proceda conforme 1.1.1;
   2.2. Defina que o participante *i* está corrompido;
   2.3. Não processe as entradas na porta $F_{in.i}$;
   2.4. Ao receber uma mensagem (input, *i*, *x*) na porta F_{infl}:
      2.4.1. Processe como como se (input, *x*) fosse recebida na porta $F_{in.i}$;
   2.5. Pare de enviar mensagens na porta $F_{out.i}$;
      2.5.1. Quando estiver para enviar uma mensagem *y* na porta $F_{out.i}$, ao invés, envie a mensagem (output, *i*, *y*) na porta F_{leak};

Figura 4.3: Modelagem de Corrupção em Funcionalidades Ideais

Como se pode observar, quando ocorre uma corrupção passiva, o adversário somente aprende as entradas e saídas do participante corrompido.

Quando ocorre uma corrupção ativa de um participante, além do obtido anteriormente, todas as entradas passam a ser escolhidas pelo adversário através da porta F_{infl} e todas as saídas passam a ser vazadas através da porta F_{leak}.

Agente F_{TPC} – Recurso – Terceira Parte Confiável



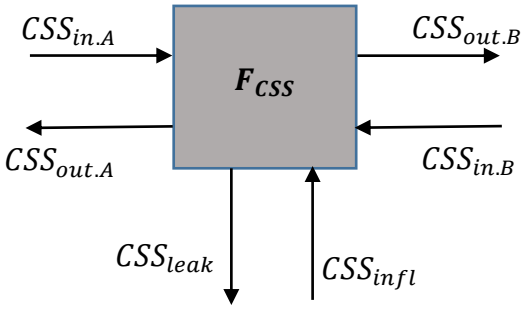
Comportamento

O comportamento em casos de corrupção passiva e ativa para funcionalidades ideais está descrito na Figura 4.3;

1. Ao receber uma mensagem (mid, A, n) na porta $TPC_{in.A}$:
   1.1. Gere o polinômio com os coeficientes $s_n | s_i = Aleatório \in F_q, \forall i = 1, \dots, n$ e armazene (mid, A, s_n);
   1.2. Envie a mensagem (mid, A, n) na porta TPC_{leak};
   1.3. Caso haja uma mensagem (mid, B, d) armazenada:
      1.3.1. Calcule $g = s(d)$;
      1.3.2. Envie a mensagem (mid, s_n) na porta $TPC_{out.A}$;
      1.3.3. Envie a mensagem (mid, d, g) na porta $TPC_{out.B}$;
2. Ao receber uma mensagem (mid, B) na porta $TPC_{in.B}$:
   2.1. Envie a mensagem (mid, B) na porta TPC_{leak};
   2.2. Gere o número $d | d = Aleatório \in F_q$ e armazene (mid, B, d);
   2.3. Caso haja uma mensagem (mid, A, s_n) armazenada:
      2.3.1. Proceda conforme 1.3;

Figura 4.4: F_{TPC}

Agente F_{CSS} – Recurso – Comunicação Síncrona Segura



Comportamento

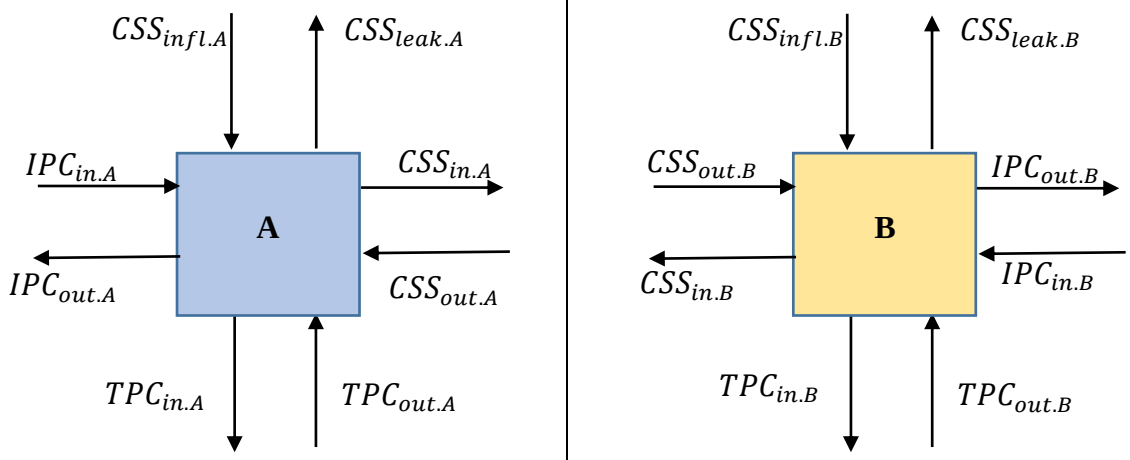
O comportamento em casos de corrupção passiva e ativa para funcionalidades ideais está descrito na Figura 4.3;

1. Ao receber uma mensagem (mid, m, i) na porta $CSS_{in.j}$:
   1.1. Envie a mensagem (mid, m, j) na porta $CSS_{out.i}$;
   1.2. Envie a mensagem (mid, m, j, i) na porta CSS_{leak};

Figura 4.5: F_{CSS}

Conforme se pode observar pelo comportamento da funcionalidade ideal F_{CSS} apresentado na Figura 4.5, é garantido que a mensagem será entregue ao destinatário correto, porém o conteúdo das mensagens será vazado para os adversários, assim como o remetente e o destinatário.

Agente π_{IPC-1} – Protocolo – Interseção Privada de Conjuntos



Comportamento

O comportamento em casos de corrupção passiva e ativa para participantes está descrito na Figura 4.7.

O comportamento de π_{IPC-1} é composto por dois agentes "A" e "B", descritos a seguir:

1. Agente "A", Alice
   - 1.1. Ao receber uma mensagem (*test*, p_n) na porta $IPC_{in.A}$:
     - 1.1.1. Armazene (*test*, A, p_n);
     - 1.1.2. Envie a mensagem (mid, A, n) na porta $TPC_{in.A}$;
   - 1.2. Ao receber uma mensagem (mid, s_n) na porta $TPC_{out.A}$:
     - 1.2.1. Armazene (mid, s_n);
     - 1.2.2. Caso existam as mensagens armazenadas (*test*, p_n) e (mid, t):
       - 1.2.2.1. Proceda conforme 1.3.1;
   - 1.3. Ao receber uma mensagem (mid, t, B) na porta $CSS_{out.A}$:
     - 1.3.1. Caso existam as mensagens armazenadas (*test*, p_n) e (mid, s_n):
       - 1.3.1.1. Calcule $f_n = p_n(x + t) + s_n$;
       - 1.3.1.2. Envie a mensagem (mid, f_n, B) na porta $CSS_{in.A}$;
2. Agente "B", Bob
   - 2.1. Ao receber uma mensagem (*test*, x_0) na porta $IPC_{in.B}$:
     - 2.1.1. Armazene (*test*, B, x_0);
     - 2.1.2. Envie a mensagem (mid, B) na porta $TPC_{in.B}$;
   - 2.2. Ao receber uma mensagem (mid, d, g) na porta $TPC_{out.B}$;
     - 2.2.1. Armazene (mid, d, g);
     - 2.2.2. Calcule $t = x_0 - d$;
     - 2.2.3. Envie a mensagem (mid, t, A) na porta $CSS_{in.B}$;
   - 2.3. Ao receber uma mensagem (mid, f_n, A) na porta $CSS_{out.B}$:
     - 2.3.1. Envie a mensagem (*result*, $f(d) - g$) na porta $IPC_{out.B}$;

Figura 4.6: π_{IPC-1}

Comportamento Padrão para Corrupção de Participantes

O comportamento em casos de corrupção passiva e ativa é o mesmo comportamento padrão para participantes descrito nas páginas 65-66 de (Cramer *et al.*, 2015), seção 4.2.3 – *Modeling Corruption*;

1. Ao receber uma mensagem (passive corrupt) na porta $R_{infl.i}$, que vem a ser a porta de comunicação do participante, sendo no presente texto a porta $CSS_{infl.i}$:
   1.1. Envie o estado interno σ na porta $R_{leak.i}$, sendo no presente texto a porta $CSS_{leak.i}$, Onde σ é definido como sendo:
      1.1.1. Todas as entradas prévias recebidas;
      1.1.2. Todas as entradas que estão na fila;
      1.1.3. Todas as saídas enviadas;
      1.1.4. Todos os valores aleatórios gerados internamente;
2. Ao receber uma mensagem (active corrupt, *i*) porta $R_{infl.i}$:
   2.1. Proceda conforme 1.1;
   2.2. Passe a executar as seguintes tarefas, e somente elas:
      2.2.1. Ao receber a mensagem (read, *P*) na porta $R_{infl.i}$, onde *P* é uma das portas de entrada do Participante *i*, leia a próxima mensagem *m* na porta *P* e envie *m* na porta $R_{leak.i}$;
      2.2.2. Ao receber a mensagem (send, *P*, *m*) na porta $R_{infl.i}$, onde *P* é uma das portas de saída do Participante *i*, envie a mensagem *m* na porta *P*;

Figura 4.7: Modelagem de Corrupção em Participantes

Conforme a definição de Segurança UC, introduzida na seção 3.7:

$\pi_{IPC-1} \diamond F_{TPC} \diamond F_{CSS}$ implementa de forma segura F_{IPC}, nos ambientes $Env^{estático}$, se existe um Simulador $\mathcal{S} \in Sim^2$ para π_{IPC-1} tal que $\pi_{IPC-1} \diamond F_{TPC} \diamond F_{CSS} \overset{Env}{\equiv} F_{IPC} \diamond \mathcal{S}$. (EQ 4.5)

Antes de apresentarmos o Simulador $\mathcal{S}$ é conveniente mostrar o diagrama contendo toda a composição $\pi_{IPC-1} \diamond F_{TPC} \diamond F_{CSS}$, conforme Figura 4.8, pois $\mathcal{S}$ terá que apresentar o mesmo conjunto de portas abertas quando estiver na composição $F_{IPC} \diamond \mathcal{S}$, que não sejam as portas de F_{IPC}.

Relembrando que isto se deve ao fato de que o objetivo é que observador $\mathcal{Z}$ não seja capaz de distinguir entre as composições ($\pi_{IPC} \diamond F_{TPC} \diamond F_{CSS}$) e ($F_{IPC} \diamond \mathcal{S}$) e um aspecto básico para isso é que ambas tenham as mesmas portas em aberto.

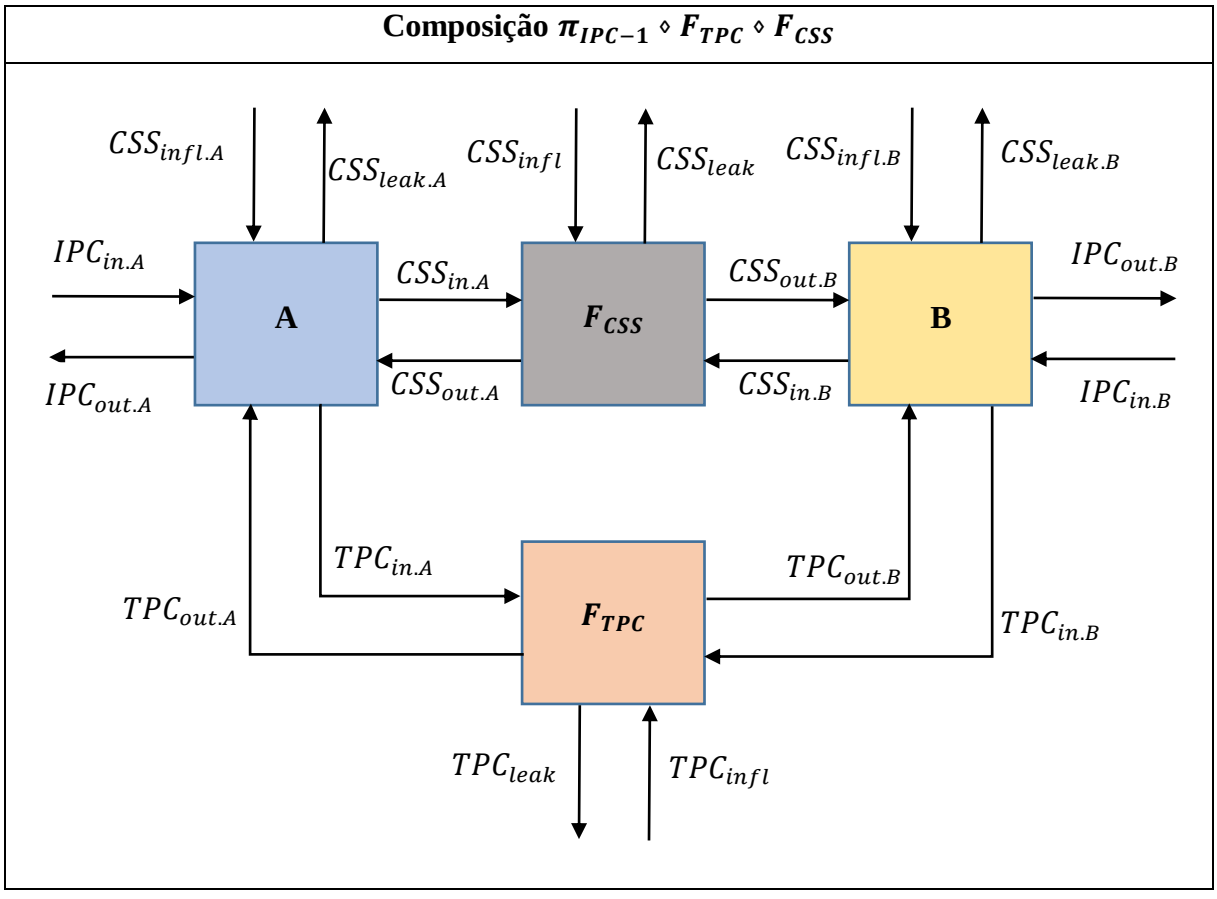


Figura 4.8: $\pi_{IPC-1} \diamond F_{TPC} \diamond F_{CSS}$

[2] $\mathcal{S} \in Sim$ segue a definição 4.15 conforme descrito nas páginas 66-67 de (Cramer *et al.*, 2015), seção 4.2.4.

As portas $IPC_{in.x}$ e $IPC_{out.x}$ existem na funcionalidade ideal F_{IPC}, portanto, $\mathcal{S}$ deverá apresentar todas as demais portas abertas presentes na Figura 4.8.

A seguir apresentamos a proposta deste Simulador $\mathcal{S}$, conforme Figura 4.9, já na composição $F_{IPC} \diamond \mathcal{S}$.

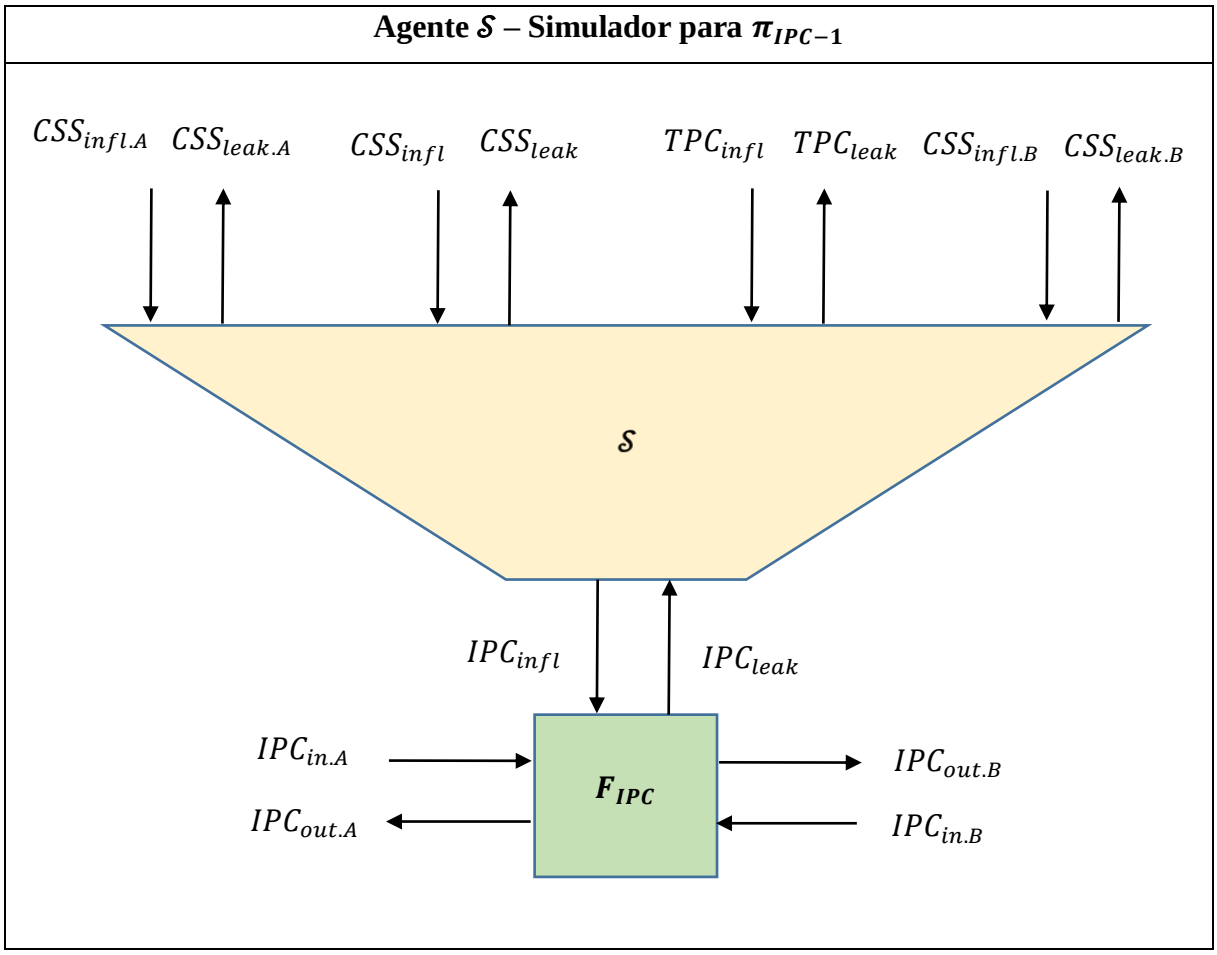


Figura 4.9: Simulador $\mathcal{S}$

Uma consequência importante de $\mathcal{S} \in Sim$ é que ele preserva a corrupção, ou seja, somente caso algum participante seja corrompido é que esta corrupção será repassada para a funcionalidade ideal, no caso F_{IPC} via porta IPC_{infl}, e exclusivamente nestes casos.

Duas consequências importantes de $\mathcal{S} \in Sim$ são:

a) Executa em tempo polinomial e, portanto, somente pode executar tarefas que retornem dentro desse limite. Repare que esta é uma restrição representativa, pois caso o Simulador $\mathcal{S}$ tivesse pode;
b) Preserva a corrupção, ou seja, somente corrompe um participante na funcionalidade ideal caso tenha recebido este comando durante a simulação. Sendo assim $\mathcal{S}$ somente

envia a mensagem (passive/active corrupt, i) na porta IPC_{infl} se, e somente se, receber uma mensagem (passive/active corrupt) na porta $CSS_{infl.i}$.

Agora que temos definidos todos os comportamentos da funcionalidade ideal F_{IPC}, dos recursos F_{TPC} e F_{CSS}, do protocolo π_{IPC-1} e do Simulador $\mathcal{S}$ podemos dar prosseguimento, finalmente, na análise dos possíveis cenários de corrupção.

4.2.1 - CENÁRIO 1 – ALICE E BOB HONESTOS

Neste cenário o comportamento para o Simulador $\mathcal{S}$ é descrito na Figura 4.10:

Agente $\mathcal{S}$ – Simulador para π_{IPC-1}
Comportamento quando Alice e Bob honestos
1. $\mathcal{S}$ ao receber a mensagem (mid, A, n) na porta IPC_{leak}: 1.1. Armazena (mid, A, n); 1.2. Caso haja uma mensagem armazenada (mid, B): 1.2.1. Gera um polinômio aleatório $z(x)\|z_i = Aleatório \in F_q, \forall i = 1, \dots, n$ e envia na porta CSS_{leak}; 2. $\mathcal{S}$ ao receber uma mensagem (mid, B) na porta IPC_{leak}: 2.1. Armazena (mid, B); 2.2. Gera $b\|b = Aleatório \in F_q$ e envia na porta CSS_{leak};

Figura 4.10: Comportamento Simulador $\mathcal{S}$ – Alice e Bob Honestos

Devido ao fato do recurso F_{CSS} vazar o conteúdo de todas as mensagens para o observador $\mathcal{Z}$, o Simulador $\mathcal{S}$ necessita tratar os casos do vazamento das duas mensagens trocadas durante a execução do protocolo, ou seja, t que é enviado de Bob para Alice e $f(x)$ que é enviado de Alice para Bob.

Para tanto, conforme descrito no comportamento do Simulador $\mathcal{S}$ este gera um valor aleatório b e um novo polinômio $z(x)$, então envia, na porta CSS_{leak}, $z(x)$ para simular $f(x)$ e envia b para simular t.

Devido ao fato de serem valores aleatórios $\mathcal{Z}$ não é capaz de distinguir entre $f(x)$ e t gerados pela composição $\pi_{IPC-1} \diamond F_{TPC} \diamond F_{CSS}$ e $z(x)$ e b gerados pela composição $F_{IPC} \diamond \mathcal{S}$.

É importante ressaltar que, apesar do recurso F_{CSS} vazar o conteúdo de todas as mensagens para o observador $\mathcal{Z}$, todos os dados trocados entre Alice e Bob durante a execução do protocolo, $f(x)$ e t, não divulgam qualquer informação sobre os dados originais destes, ou seja, $p(x)$ e x_0, pois estão ofuscados por valores aleatórios.

4.2.2 - CENÁRIO 2 – ALICE E BOB CORRUPTOS

Apesar deste cenário parecer complexo de abordar, na verdade, ele é trivial[3].

Quando ocorre a corrupção tanto de Alice quanto de Bob o Simulador $\mathcal{S}$ passa a ter acesso a todo o estado interno de todos os participantes e também da funcionalidade F_{IPC} e então é suficiente que $\mathcal{S}$ execute internamente uma cópia da composição $\pi_{IPC-1} \diamond F_{TPC} \diamond F_{CSS}$.

4.2.3 - CENÁRIO 3 – ALICE CORRUPTA E BOB HONESTO

Para o caso de corrupção de Alice o comportamento para o Simulador $\mathcal{S}$ é descrito na Figura 4.11.

Agente $\mathcal{S}$ – Simulador para π_{IPC-1}
Comportamento quando Alice corrupta e Bob honesto
1. $\mathcal{S}$ ao receber uma mensagem (passive corrupt) ou (active corrupt) na porta $CSS_{infl.A}$: 1.1. Envia a mensagem (passive corrupt, A) ou (active corrupt, A) na porta IPC_{infl}; 2. F_{IPC} ao receber (passive corrupt, A) ou (active corrupt, A) na porta IPC_{infl}: 2.1. Envia a mensagem (state, A, $p(x)$) na porta IPC_{leak}; 3. $\mathcal{S}$ ao receber a mensagem (state, A, $p(x)$) na porta IPC_{leak}: 3.1. Envia a mensagem (state, A, $p(x)$) na porta $CSS_{leak.A}$; 3.2. Gera um polinômio aleatório $k(x)\|k_i = Aleatório \in F_q, \forall i = 1, \dots, n$ e envia na porta $CSS_{leak.A}$; 4. $\mathcal{S}$ ao receber uma mensagem (mid, B) na porta IPC_{leak}: 4.1. Gera $w\|w = Aleatório \in F_q$ e envia na porta CSS_{leak}; 4.2. Gera um polinômio aleatório $l(x) = p(x + w) + k(x)$ e envia na porta CSS_{leak};

Figura 4.11: Comportamento Simulador $\mathcal{S}$ – Alice corrupta e Bob honesto

Note que a partir do momento em que Alice é corrompida o Simulador $\mathcal{S}$ passa a conhecer todos os dados de Alice, o que incluiu os coeficientes do polinômio $p(x)$, pois F_{IPC} entrega esses dados na porta IPC_{leak} e este por sua vez repassa para a porta $CSS_{leak.A}$, conforme detalhado no comportamento do Simulador $\mathcal{S}$.

Entretanto, relembramos que para que o protocolo seja seguro no modelo UC é necessário que o observador $\mathcal{Z}$ não seja capaz de diferenciar entre as composições ($\pi_{IPC-1} \diamond F_{TPC} \diamond F_{CSS}$) e ($F_{IPC} \diamond \mathcal{S}$) e nota-se que na execução $\pi_{IPC-1} \diamond F_{TPC} \diamond F_{CSS}$ Alice não possui apenas o polinômio $p(x)$, mas também o polinômio $s(x)$ enviado pela TPC, durante a fase off-line.

[3] Conforme descrito na página 86 de (Cramer *et al.*, 2015), seção 4.4.1.

Pelo fato de não haver uma TPC na composição $F_{IPC} \diamond \mathcal{S}$, a funcionalidade ideal F_{IPC} não gera o polinômio $s(x)$, por não fazer parte do seu comportamento.

É neste momento que o Simulador $\mathcal{S}$ assume o papel de F_{TPC} e gera um novo polinômio aleatório $k(x)$ de grau n e entrega na porta $CSS_{leak.A}$.

Por serem polinômios aleatórios de grau n, o observador $\mathcal{Z}$ não é capaz de distinguir entre o polinômio $s(x)$ gerado pela composição $\pi_{IPC-1} \diamond F_{TPC} \diamond F_{CSS}$ e o polinômio $k(x)$ gerado pela composição $F_{IPC} \diamond \mathcal{S}$, pois olhando apenas esses dados ele não tem vantagem estatística com base apenas nessa informação para saber se ela foi gerada por ($\pi_{IPC-1} \diamond F_{TPC} \diamond F_{CSS}$) ou por ($F_{IPC} \diamond \mathcal{S}$).

Novamente, devido ao fato do recurso F_{CSS} vazar o conteúdo de todas as mensagens para o observador $\mathcal{Z}$, além de simular $s(x)$ através de $k(x)$ o Simulador $\mathcal{S}$ necessita também tratar os casos do vazamento das duas mensagens trocadas durante a execução do protocolo, ou seja, t e $f(x)$.

Para tanto, conforme descrito no comportamento do Simulador $\mathcal{S}$ na Figura 4.11, este gera um valor aleatório w e calcula um novo polinômio $l(x) = p(x + w) + k(x)$, então envia, na porta CSS_{leak}, $l(x)$ para simular $f(x)$ e envia w para simular t.

Novamente, por serem valores aleatórios $\mathcal{Z}$ não é capaz de distinguir entre $f(x)$ e t gerados pela composição $\pi_{IPC-1} \diamond F_{TPC} \diamond F_{CSS}$ e $l(x)$ e w gerados pela composição $F_{IPC} \diamond \mathcal{S}$.

Para o caso de corrupção ativa, Alice pode decidir enviar $f(x) \neq p(x + t) + s(x)$ para Bob.

Ressalta-se, entretanto, que este comportamento seria equivalente a Alice decidir utilizar um polinômio $p(x)$ cujas raízes não sejam aquelas do seu conjunto $\mathbb{A}$. Porém substituir $p(x)$ é uma alteração de parâmetro de entrada, o que é uma influência permitida.

4.2.4 - CENÁRIO 4 – ALICE HONESTA E BOB CORRUPTO

Para o caso de corrupção de Bob o comportamento para o Simulador $\mathcal{S}$ é descrito na Figura 4.12.

Agente $\mathcal{S}$ – Simulador para π_{IPC-1}
Comportamento quando Alice honesta e Bob corrupto
1. $\mathcal{S}$ ao receber uma mensagem (passive corrupt) ou (active corrupt) na porta $CSS_{infl.B}$: 1.1. Envia a mensagem (passive corrupt, B) ou (active corrupt, B) na porta IPC_{infl}; 2. F_{IPC} ao receber (passive corrupt, B) ou (active corrupt, B) na porta IPC_{infl}: 2.1. Envia a mensagem (state, B, x_0) na porta IPC_{leak}; 3. $\mathcal{S}$ ao receber a mensagem (state, B, x_0) na porta IPC_{leak}: 3.1. Envia a mensagem (state, B, x_0) na porta $CSS_{leak.B}$; 3.2. Gera $c \mid c = Aleatório \in F_q$ e envia na porta $CSS_{leak.B}$; 3.3. Gera $q \mid q = Aleatório \in F_q$ e envia na porta $CSS_{leak.B}$; 3.4. Calcula $j = x_0 - c$ e envia na porta $CSS_{leak.B}$; 3.5. Envia a mensagem (mid, j, B, A) na porta CSS_{leak}; 4. $\mathcal{S}$ ao receber uma mensagem (mid, A, n) na porta IPC_{leak}: 4.1. Gera um polinômio aleatório $v(x) \mid v_i = Aleatório \in F_q, \forall i = 1, \dots, n$ e envia na porta CSS_{leak};

Figura 4.12: Comportamento Simulador $\mathcal{S}$ – Alice honesta e Bob corrupto

Neste cenário, assim como no anterior, a partir do momento em que Bob é corrompido o Simulador $\mathcal{S}$ passa a conhecer todos os seus dados, o que incluiu o ponto x_0 a ser testado, pois, novamente, F_{IPC} entrega esses dados na porta IPC_{leak} e este por sua vez repassa para a porta $CSS_{leak.B}$.

Mais uma vez há uma diferença entre a composição ($\pi_{IPC-1} \diamond F_{TPC} \diamond F_{CSS}$) e ($F_{IPC} \diamond \mathcal{S}$) pois na primeira Bob também possui os pontos d e $g = s(d)$ enviados pela TPC.

Como anteriormente, o Simulador $\mathcal{S}$ assume o papel de F_{TPC} e gera dois novos pontos aleatórios c e q e entrega na porta $CSS_{leak.\mathrm{B}}$.

Repare que por d ser um ponto aleatório e g ser um ponto gerado a partir de um polinômio aleatório, ambos possuem uma distribuição aleatória e o observador $\mathcal{Z}$ não é capaz de distinguir entre os pontos d e $g = s(d)$ gerados pela composição $\pi_{IPC-1} \diamond F_{TPC} \diamond F_{CSS}$ e os pontos c e q gerados pela composição $F_{IPC} \diamond \mathcal{S}$, pois analisando apenas esses dados ele, novamente, não tem vantagem estatística.

Como antes, o fato do recurso F_{CSS} vazar o conteúdo de todas as mensagens para o observador $\mathcal{Z}$, além de simular d e g através de c e q o Simulador $\mathcal{S}$ necessita também tratar os casos do

vazamento já mencionado das duas mensagens trocadas durante a execução do protocolo, ou seja, t e $f(x)$.

Para tanto, conforme descrito no comportamento do Simulador $\mathcal{S}$ na Figura 4.12 este calcula $j = x_0 - c$ e gera um novo polinômio $v(x)$, então envia, na porta CSS_{leak}, $v(x)$ para simular $f(x)$ e envia j para simular t. O observador $\mathcal{Z}$ não é capaz de distinguir entre os dois casos como já justificado anteriormente.

Para o caso de corrupção ativa, Bob pode decidir enviar $t \neq x_0 - d$ para Alice.

Porém este comportamento seria equivalente a Bob decidir utilizar um ponto diferente de x_0 e substituir x_0 é uma alteração de parâmetro de entrada, o que é uma influência permitida.

Bob também pode decidir utilizar um polinômio $f(x)$ diferente daquele enviado por Alice ou, de forma equivalente, apresentar um resultado final da execução do algoritmo $y \neq f(d) - g$. Repare que para tanto o observador $\mathcal{Z}$ deve explicitamente enviar este novo valor para Bob através de uma mensagem (send,$IPC_{out.B}$, y) na porta $CSS_{infl.B}$ quando manipulando a composição $\pi_{IPC-1} \diamond F_{TPC} \diamond F_{CSS}$.

Ao fazer isso na composição $F_{IPC} \diamond \mathcal{S}$ o Simulador $\mathcal{S}$ aprende o valor y e simplesmente repassa essa mensagem na porta IPC_{infl} fazendo com que a funcionalidade ideal emita este valor na porta $IPC_{out.B}$, replicando assim o exato mesmo resultado.

Tendo abordado todos os quatro cenários, tanto para os casos de corrupção passiva quanto corrupção ativa, conseguimos criar um Simulador $\mathcal{S}$ que satisfaz a condição (EQ 4.5) e, dessa forma, provar o Teorema 1.

Salienta-se, por último, que não foi feita qualquer premissa com relação ao poder computacional do observador $\mathcal{Z}$, tendo inclusive o recurso de comunicação F_{CSS} vazado todas as mensagens enviadas. Dessa forma o Protocolo IPC com TPC - Versão 1 é incondicionalmente seguro.

5- PROTOCOLO IPC SEM TPC - VERSÃO 2

Esta segunda versão do protocolo visa implementar a interseção privada de conjuntos, com o mesmo resultado e invariantes presentes na versão 1, porém sem utilizar uma Terceira Parte Confiável.

A ideia neste caso é que Alice e Bob consigam simular o papel da TPC através de uma troca de mensagens numa nova fase off-line do protocolo, apresentada a seguir na Figura 5.1;

Protocolo IPC Sem TPC - Versão 2
Fase Off-line
1. Alice: 1.1. Cria aleatoriamente um polinômio $h(x)$ de grau $n \in \mathbb{F}_q$; 1.2. Gera um par de chaves pública e privada Ek e Dk, respectivamente, para o algoritmo criptográfico de Paillier[4] $\in \mathbb{F}_N$, onde $N > q$; 1.3. Cifra os coeficientes de $h(x)$, com a chave privada Dk, criando o polinômio $h\_c(x)$; 1.4. Envia $h\_c(x)$ e Ek para Bob; 2. Bob: 2.1. Recebe $h\_c(x)$ e Ek de Alice; 2.2. Cria aleatoriamente dois pontos w e $delta$, ambos $\in \mathbb{F}_q$; 2.3. Calcula a avaliação de $h\_c(w) = \prod_{i=0}^{n} h\_c(x)_i^{\,u}$ $onde\ u = w^i\ e\ h\_c(x)_i\ representa\ o\ iésimo\ coeficiente\ de\ h\_c(x)$; 2.4. Ofusca $h\_c(w)$ calculando $h\_c(w) = h\_c(w) * E_{ek}(delta)$; 2.5. Envia $h\_c(w)$ para Alice; 3. Alice: 3.1. Recebe $h\_c(w)$ de Bob; 3.2. Decifra $h\_c(w)$ como $h\_d(w) = D_{dk}(h\_c(w))$; 3.3. Envia $h\_d(w)$ para Bob; 4. Bob: 4.1. Recebe $h\_d(w)$ de Alice; 4.2. Calcula $h(w) = h\_d(w) - delta$;
Fase Online
1. Parâmetros de entrada: 1.1. Alice possui um conjunto de elementos $\mathbb{A}$, que são as raízes do polinômio $p(x)$ de grau n; 1.2. Bob possui um elemento x_0; 2. Bob calcula $t = x_0 - w$ e envia t para Alice; 3. Alice calcula $f(x) = p(x + t) + h(x)$ e envia $f(x)$ para Bob; 4. Bob calcula $r = f(w) - h(w)$: 4.1. Se $r = 0$ então $x_0 \in \mathbb{A}$; 4.2. Caso contrário $x_0 \notin \mathbb{A}$;

Figura 5.1: Protocolo IPC Sem TPC - Versão 2

[4] Vide seção 3.4

5.1 - PROVA DE CORRETUDE

Considerando que a fase online apresentada na Figura 5.1 é a mesma que aquela apresentada na Figura 4.1 vamos analisar apenas o resultado da nova fase off-line apresentada na Figura 5.1.

Vale lembrar que objetivo é simular o comportamento de F_{TPC} apresentado na Figura 4.4, ou seja, ao final da execução Alice deve possuir um polinômio aleatório de grau n, anteriormente chamado de $s(x)$, e Bob deve possuir um ponto aleatório e a avaliação do polinômio de Alice neste ponto, anteriormente chamados de d e $g = s(d)$.

As invariantes são:

a) Alice não aprende qualquer dado sobre o ponto aleatório de Bob;
b) Alice não aprende qualquer dado sobre a avalição de seu polinômio no ponto aleatório de Bob;
c) Bob não aprende qualquer dado sobre o polinômio aleatório de Alice, a não ser seu tamanho;
d) Um observador externo, com poder computacional polinomial, não é capaz de obter nenhum dado de Alice ou Bob.

Alice, ao enviar o polinômio cifrado $h\_c(x)$ para Bob, não fornece qualquer dado sobre seu polinômio pois Bob não possui a chave privada Dk.

O aspecto essencial desta nova fase é a característica homomórfica do algoritmo criptográfico de Paillier, pois permite a Bob realizar a avaliação do seu ponto aleatório w em $h(x)$ sem ao menos conhecer seus coeficientes, calculando para isso o produtório apresentando na Figura 5.1.

Caso Bob enviasse o valor $h\_c(w)$ obtido através do produtório diretamente para Alice, esta obteria a avaliação de $h(w)$ quando realizasse a decifragem, descumprindo assim a segunda invariante apresentada e, para impedir tal fato, Bob ofusca $h\_c(w)$ multiplicando pelo valor de $delta$ cifrado com a chave pública Ek, o que provoca a soma de $delta$ no valor decifrado de $h\_c(w)$ em função da característica homomórfica do algoritmo.

Portanto, Alice ao decifrar $h\_c(w)$ nada obtém de $h(w)$, por não conhecer $delta$. Bob ao receber $h\_d(w)$ e subtrair $delta$ consegue com sucesso obter $h(w)$ sem nada saber sobre $h(x)$, provando assim que a nova etapa off-line consegue com sucesso simular o papel da TPC cumprindo todas as invariantes apresentadas.

5.2 - PROVA DE SEGURANÇA NO MODELO UC

Teorema 2. O Protocolo IPC sem TPC - Versão 2 é Seguro no Modelo UC, com segurança estática, para adversários passivos e com poder computacional polinomial.

A fim de provarmos o **Teorema 2** precisamos das definições a seguir:

a) F_{IPC} é a funcionalidade ideal de nome IPC, conforme Figura 4.2;

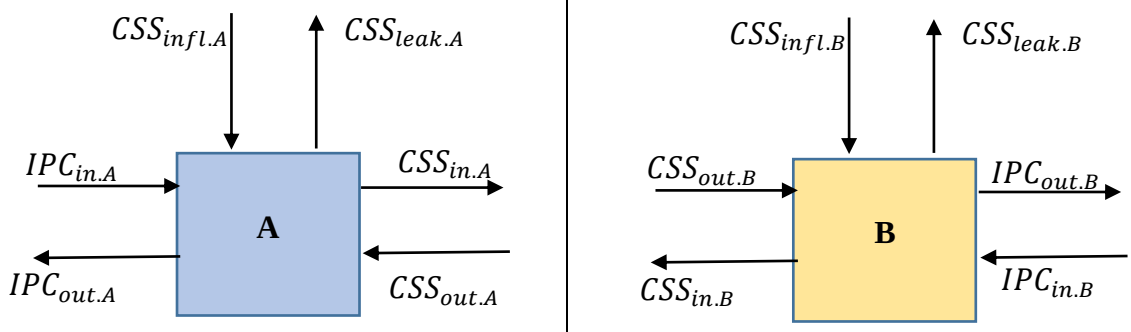
b) F_{CSS} é a funcionalidade ideal de nome CSS, conforme Figura 4.5;

c) π_{IPC-2} é o protocolo de nome IPC-2 que utiliza o recurso de nome CSS, conforme Figura 5.2;

d) $Env^{estático,poli,passivo}$ [5] é a classe de ambientes;

[5] Vide definições (EQ 3.32), (EQ 3.33) e (EQ 3.34)

Agente π_{IPC-2} – Protocolo – Interseção Privada de Conjuntos



Comportamento

O comportamento em casos de corrupção passiva e ativa para participantes está descrito na Figura 4.7.

Fase Off-line

1. Agente "A", Alice
   1.1. Ao inicializar:
      1.1.1. Gere o polinômio com os coeficientes $h_n|h_i = Aleatório \in F_q, \forall i = 1, \dots, n$;
      1.1.2. Gere um par de chaves pública e privada Ek e Dk, respectivamente, para o algoritmo criptográfico de Paillier $\in \mathbb{F}_N$, onde $N > q$;
      1.1.3. Cifre os coeficientes de $h(x)$ com a chave privada Dk criando o polinômio $h\_c(x)$;
      1.1.4. Armazene (mid, $h(x)$);
      1.1.5. Envie a mensagem (mid, $h\_c(x)|Ek, B$) na porta $CSS_{in.A}$;
   1.2. Ao receber uma mensagem (mid, $h\_c(w)$, B) na porta $CSS_{out.A}$:
      1.2.1. Decifre $h\_c(w)$ calculando $h\_d(w) = D_{dk}(h\_c(w))$;
      1.2.2. Envie a mensagem (mid, $h\_d(w), B$) na porta $CSS_{in.A}$;
2. Agente "B", Bob
   2.1. Ao receber uma mensagem (mid, $h\_c(x)|Ek, A$) na porta $CSS_{out.B}$:
      2.1.1. Gere o número $w|w = Aleatório \in F_q$ e armazene (mid, w);
      2.1.2. Gere o número $delta|delta = Aleatório \in F_q$ e armazene (mid, $delta$);
      2.1.3. Calcule $h\_c(w) = \prod_{i=0}^{n} h\_c(x)_i^{\,u}$
      $onde\ u = w^i\ e\ h\_c(x)_i\ representa\ o\ iésimo\ coeficiente\ de\ h\_c(x)$;
      2.1.4. Calcule $h\_c(w) = h\_c(w) * E_{ek}(delta)$;
      2.2. Envie a mensagem (mid, $h\_c(w)$, A) na porta $CSS_{in.B}$;
   2.3. Ao receber uma mensagem (mid, $h\_d(w)$, A) na porta $CSS_{out.B}$:
      2.3.1. Calcule $h(w) = h\_d(w) - delta$;
      2.3.2. Armazene (mid, $h(w)$);

Fase Online
1. Agente “A”, Alice 1.1. Ao receber uma mensagem (*test*, p_n) na porta $IPC_{in.A}$: 1.1.1. Armazene (*test*, A, p_n); 1.1.2. Caso exista a mensagem armazenada (mid, t): 1.1.2.1. Proceda conforme 1.2.1; 1.2. Ao receber uma mensagem (mid, t, B) na porta $CSS_{out.A}$: 1.2.1. Caso exista a mensagem armazenada (*test*, p_n): 1.2.1.1. Calcule $f_n = p_n(x + t) + h_n$; 1.2.1.2. Envie a mensagem (mid, f_n, B) na porta $CSS_{in.A}$; 2. Agente “B”, Bob 2.1. Ao receber uma mensagem (*test*, x_0) na porta $IPC_{in.B}$: 2.1.1. Armazene (*test*, B, x_0); 2.1.2. Calcule $t = x_0 - w$; 2.1.3. Envie a mensagem (mid, t, A) na porta $CSS_{in.B}$; 2.2. Ao receber uma mensagem (mid, f_n, A) na porta $CSS_{out.B}$: 2.2.1. Envie a mensagem (result, $f(w) -$ h(w)) na porta $IPC_{out.B}$;

Figura 5.2: π_{IPC-2}

Conforme a definição de Segurança UC, introduzida na seção 3.7:

$\pi_{IPC-2} \diamond F_{CSS}$ implementa de forma segura F_{IPC}, nos ambientes $Env^{estático,poli,passivo}$, se existe um Simulador $\mathcal{S} \in Sim$ para π_{IPC-2} tal que (EQ 5.1)

$$\pi_{IPC-2} \diamond F_{CSS} \overset{Env}{\equiv} F_{IPC} \diamond \mathcal{S}.$$

A seguir apresentarmos o diagrama contendo toda a composição $\pi_{IPC-2} \diamond F_{CSS}$, conforme Figura 5.3, pois $\mathcal{S}$ terá que apresentar o mesmo conjunto de portas abertas quando estiver na composição $F_{IPC} \diamond \mathcal{S}$, que não sejam as portas de F_{IPC}.

Relembrando que isto se deve ao fato de que o objetivo é que o observador $\mathcal{Z}$ não seja capaz de distinguir entre as composições ($\pi_{IPC-2} \diamond F_{CSS}$) e ($F_{IPC} \diamond \mathcal{S}$) e um aspecto básico para isso é que ambas tenham as mesmas portas em aberto.

Repare que não mais existe o recurso F_{TPC}, pois é justamente a funcionalidade fornecida por ele que buscamos reproduzir na nova fase off-line.

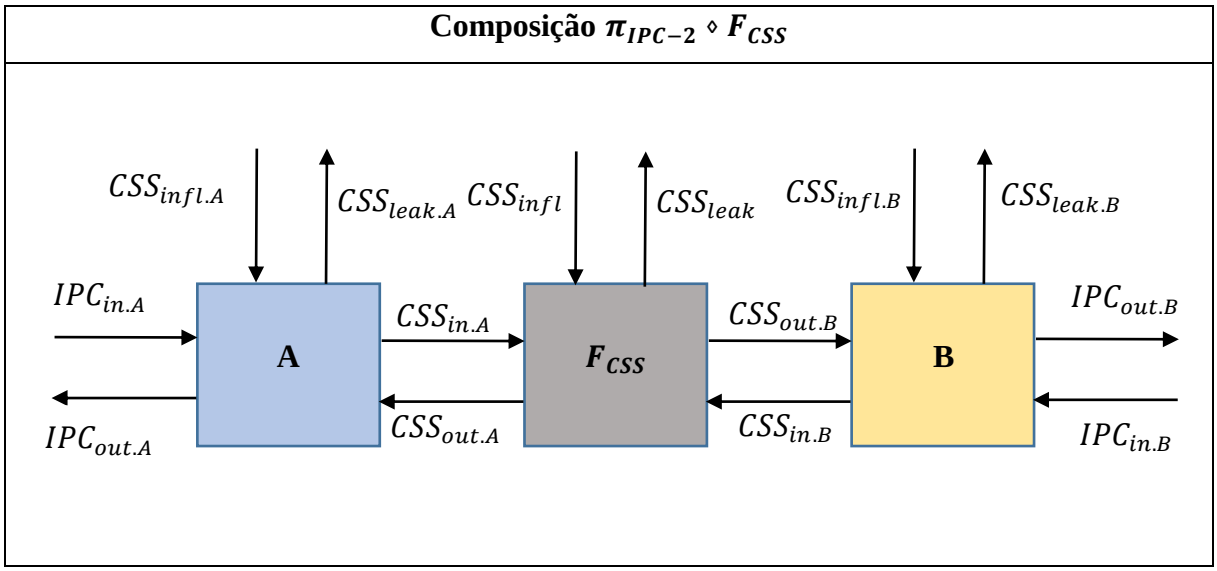


Figura 5.3: $\pi_{IPC-2} \diamond F_{CSS}$

A seguir apresentamos a proposta deste Simulador $\mathcal{S}$, conforme Figura 5.4, já na composição $F_{IPC} \diamond \mathcal{S}$.

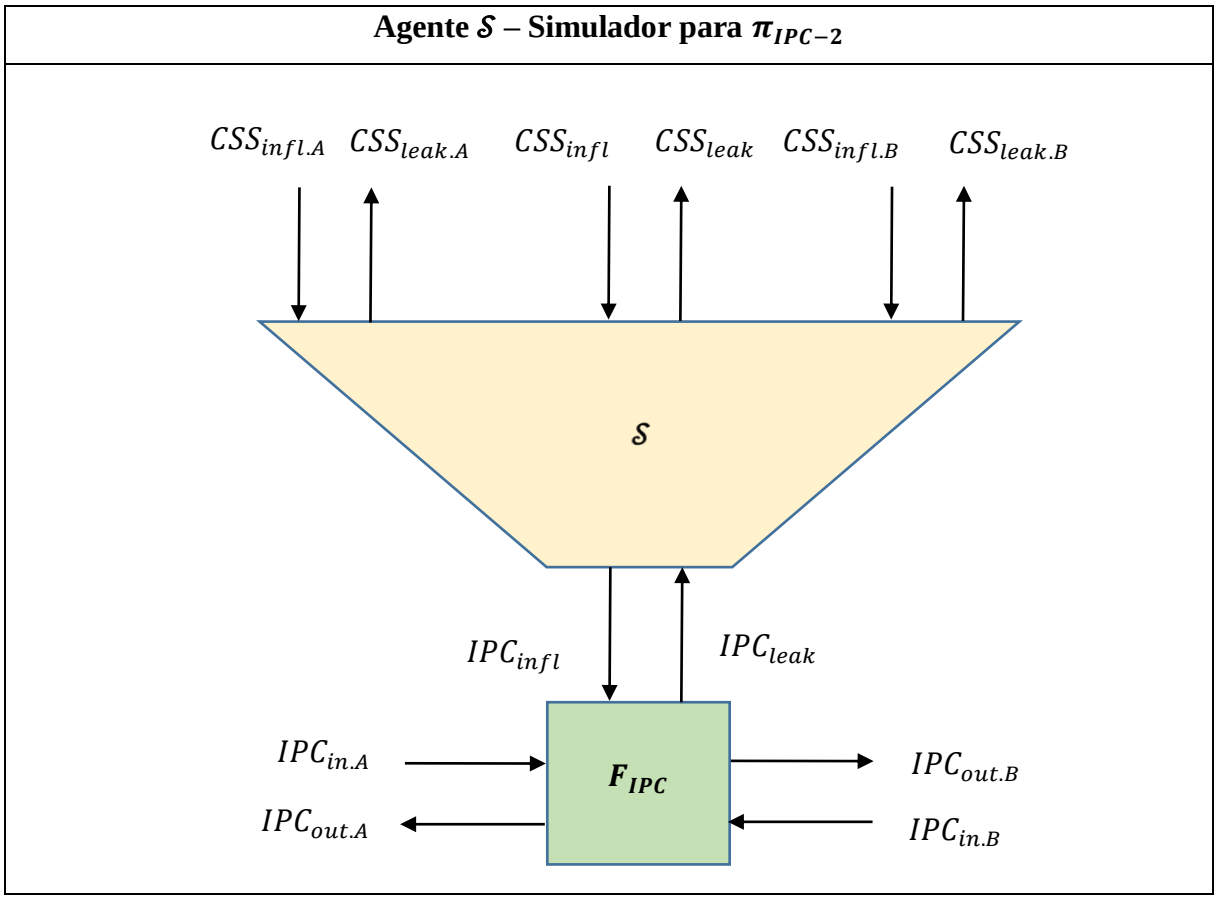


Figura 5.4: Simulador $\mathcal{S}$

5.2.1 - CENÁRIO 1 – ALICE E BOB HONESTOS

Neste cenário o comportamento para o Simulador $\mathcal{S}$ é descrito na Figura 5.5:

Agente $\mathcal{S}$ – Simulador para π_{IPC-2}
Comportamento quando Alice e Bob honestos
1. $\mathcal{S}$ ao inicializar: 1.1. Gere o polinômio com os coeficientes $sh_n \| sh_i = Aleatório \in F_q, \forall i = 1, \dots, n$; 1.2. Gere um par de chaves pública e privada E_{sk} e D_{sk}, respectivamente, para o algoritmo criptográfico de Paillier $\in \mathbb{F}_N$, onde $N > q$; 1.3. Cifre os coeficientes de $sh(x)$ com a chave privada Dsk criando o polinômio $sh\_c(x)$; 1.4. Armazene (mid, $sh(x)$); 1.5. Envie a mensagem (mid, $sh\_c(x) \| E_{sk}, A, B$) na porta CSS_{leak}; 1.6. Gere o número $sw \| sw = Aleatório \in F_q$ e armazene (mid, sw); 1.7. Cifre o número $sh\_c(sw) = E_{Esk}(sh(sw))$; 1.8. Envie a mensagem (mid, $sh\_c(sw), B, A$) na porta CSS_{leak}; 1.9. Decifre $sh\_c(sw)$ calculando $sh\_d(sw) = D_{Dsk}(sh\_c(sw))$; 1.10. Envie a mensagem (mid, $sh\_d(sw), A, B$) na porta CSS_{leak};

Figura 5.5: Comportamento Simulador $\mathcal{S}$ – Alice e Bob Honestos

Como ocorreu na versão 1 do protocolo, devido ao fato do recurso F_{CSS} vazar o conteúdo de todas as mensagens para o observador $\mathcal{Z}$, o Simulador $\mathcal{S}$ necessita tratar os casos do vazamento das mensagens trocadas durante a execução do protocolo em sua fase off-line:

a) A chave pública de Paillier Ek enviada de Alice para Bob;

b) O polinômio $h\_c(x)$ enviado de Alice para Bob;

c) O valor $h\_c(w)$ enviado de Bob para Alice;

d) O valor $h\_d(w)$ enviado de Alice para Bob;

Conforme pode ser visualizado na Figura 5.5, o Simulador $\mathcal{S}$ simula, por isso a letra "s" no começo dos nomes das mensagens, todos esses valores de forma aleatória e vaza na porta CSS_{leak}.

Devido ao fato de serem valores aleatórios, $\mathcal{Z}$ não é capaz de distinguir entre os gerados pela composição $\pi_{IPC-2}F_{CSS}$ e aqueles simulados pela composição $F_{IPC} \diamond \mathcal{S}$.

É importante ressaltar que, apesar do recurso F_{CSS} vazar o conteúdo de todas as mensagens para o observador $\mathcal{Z}$, todos os dados trocados entre Alice e Bob durante a execução do protocolo, listados nos itens anteriores, não divulgam qualquer informação sobre os dados originais destes, ou seja, $h(x)$, w e $h(w)$, pois estão cifrados pelo algoritmo criptográfico de Paillier.

5.2.2 - CENÁRIO 2 – ALICE E BOB CORRUPTOS

Este cenário segue o mesmo tratamento que aquele apresentado na seção 4.2.2.

5.2.3 - CENÁRIO 3 – ALICE CORRUPTA E BOB HONESTO

Para o caso de corrupção de Alice o comportamento para o Simulador $\mathcal{S}$ é descrito na Figura 5.6.

Agente $\mathcal{S}$ – Simulador para π_{IPC-2}
Comportamento quando Alice corrupta e Bob honesto
1. $\mathcal{S}$ ao receber uma mensagem (passive corrupt) na porta $CSS_{infl.A}$: 1.1. Envia a mensagem (passive corrupt, A) na porta IPC_{infl}; 2. F_{IPC} ao receber (passive corrupt, A) na porta IPC_{infl}: 2.1. Envia a mensagem (state, A, $p(x)$) na porta IPC_{leak}; 3. $\mathcal{S}$ ao receber a mensagem (state, A, $p(x)$) na porta IPC_{leak}: 3.1. Envia a mensagem (state, A, $p(x)$) na porta $CSS_{leak.A}$; 3.2. Gera o polinômio com os coeficientes $sh_n \mid sh_i = Aleatório \in F_q, \forall i = 1, \dots, n$; 3.3. Gera um par de chaves pública e privada E_{sk} e D_{sk}, respectivamente, para o algoritmo criptográfico de Paillier $\in \mathbb{F}_N$, onde $N > q$; 3.4. Envia a mensagem (state, A, $E_{sk} \mid D_{sk}$) na porta $CSS_{leak.A}$; 3.5. Cifra os coeficientes de $sh(x)$ com a chave privada D_{sk} criando o polinômio $sh\_c(x)$; 3.6. Armazene (mid, $sh(x)$); 3.7. Envia a mensagem (state, A, $sh(x)$) na porta $CSS_{leak.A}$;; 3.8. Envia a mensagem (mid, $sh\_c(x) \mid E_{sk}, A, B$) na porta CSS_{leak}; 3.9. Gera o número $sw \mid sw = Aleatório \in F_q$ e armazene (mid, sw); 3.10. Cifra o número $sh\_c(sw) = E_{Esk}(sh(sw))$; 3.11. Envia a mensagem (mid, $sh\_c(sw), B, A$) na porta CSS_{leak}; 3.12. Envia a mensagem (state, A, $sh\_c(sw)$) na porta $CSS_{leak.A}$; 3.13. Decifra $sh\_c(sw)$ calculando $sh\_d(sw) = D_{Dsk}(sh\_c(sw))$; 3.14. Envia a mensagem (mid, $sh\_d(sw), A, B$) na porta CSS_{leak}; 3.15. Envia a mensagem (state, A, $sh\_d(sw)$) na porta $CSS_{leak.A}$;

Figura 5.6: Comportamento Simulador $\mathcal{S}$ – Alice corrupta e Bob honesto

Seguindo o mesmo caso apresentado na versão 1 do protocolo, note que a partir do momento em que Alice é corrompida o Simulador $\mathcal{S}$ passa a conhecer todos os dados de Alice, o que incluiu os coeficientes do polinômio $p(x)$, pois F_{IPC} entrega esses dados na porta IPC_{leak} e este por sua vez repassa para a porta $CSS_{leak.A}$, conforme detalhado no comportamento do Simulador $\mathcal{S}$.

Entretanto, relembramos que para que o protocolo seja seguro no modelo UC é necessário que o observador $\mathcal{Z}$ não seja capaz de diferenciar entre as composições $\pi_{IPC-2} \diamond F_{CSS}$ e $F_{IPC} \diamond \mathcal{S}$ e nota-se que na execução $\pi_{IPC-2} \diamond F_{CSS}$ Alice não possui apenas o polinômio $p(x)$, mas agora também os dados necessários para a execução da fase off-line:

a) Par de chaves de Paillier E_{sk} e D_{sk};
b) O polinômio $h(x)$;
c) O valor $h\_c(w)$ enviado por Bob;
d) O valor $h\_d(w)$ enviado para Bob;

A funcionalidade ideal F_{IPC} não gera esses dados por não fazer parte do seu comportamento.

Conforme pode ser visualizado na Figura 5.6, mais uma vez o Simulador $\mathcal{S}$ simula todos esses valores de forma aleatória e vaza na porta $CSS_{leak.A}$.

Devido ao fato de serem valores aleatórios, $\mathcal{Z}$ não é capaz de distinguir entre os gerados pela composição $\pi_{IPC-2} \diamond F_{CSS}$ e aqueles simulados pela composição $F_{IPC} \diamond \mathcal{S}$.

Novamente, devido ao fato do recurso F_{CSS} vazar o conteúdo de todas as mensagens para o observador $\mathcal{Z}$, além de simular os valores listados anteriormente, o Simulador $\mathcal{S}$ necessita também tratar os casos do vazamento das mensagens trocadas durante a execução do protocolo.

Estas são as exatas mesmas mensagens que aquelas apresentadas e simuladas com sucesso na seção 5.2.1.

Para o caso de corrupção ativa, Alice pode decidir enviar $f(x) \neq p(x+t) + h(x)$ para Bob.

Ressalta-se, entretanto, que este comportamento seria equivalente a Alice decidir utilizar um polinômio $p(x)$ cujas raízes não sejam aquelas do seu conjunto $\mathbb{A}$. Porém substituir $p(x)$ é uma alteração de parâmetro de entrada, o que é uma influência permitida.

Alice também pode decidir enviar, na nova fase off-line, um valor $y \neq D_{Dsk}(h\_c(w))$ no lugar de $h\_d(w)$ para Bob. Entretanto, repare que para isso o observador $\mathcal{Z}$ deve explicitamente enviar este novo valor para Alice através de uma mensagem $(\text{send}, CSS_{in.\text{A}}, y)$ na porta $CSS_{infl.A}$ quando manipulando a composição $\pi_{IPC-2} \diamond F_{CSS}$. Ao fazer isso na composição $F_{IPC} \diamond \mathcal{S}$ o Simulador $\mathcal{S}$ aprende o valor y.

Porém é neste momento que o Simulador apresenta uma limitação que não é possível de ser superada, pois enviar um $y \neq D_{Dsk}(h\_c(w))$ provoca, quando se está manipulando a composição $\pi_{IPC-2} \diamond F_{CSS}$, que a avaliação do ponto aleatório de Bob w será $u \neq h(w)$, o que

causará o cálculo do resultado final de Bob $result\_s = f(w) -$ u emitido na porta $IPC_{out.B}$. Repare que $result\_s \neq f(w) - h(w)$ fazendo com que o protocolo deixe de ser correto.

O Simulador $\mathcal{S}$ não tem como enviar a mensagem (send,$IPC_{out.\mathrm{B}}$, $result\_s$) para a funcionalidade ideal F_{IPC} pois no presente cenário apenas Alice está corrompida e Bob é honesto e como $\mathcal{S}$ preserva a corrupção ele só poderia trocar uma saída de Bob caso tenha explicitamente recebido uma mensagem para corromper Bob na porta $CSS_{infl.B}$, o que não é o caso do presente cenário.

Ainda assim, mesmo que fosse possível simular com sucesso este comportamento, o protocolo deixaria de ser correto através de uma influência não permitida.

Por este motivo o Protocolo IPC sem TPC – Versão 2 não é Seguro no Modelo UC contra adversários ativos.

5.2.4 - CENÁRIO 4 – ALICE HONESTA E BOB CORRUPTO

Para o caso de corrupção de Bob o comportamento para o Simulador $\mathcal{S}$ é descrito na Figura 5.7.

Agente $\mathcal{S}$ – Simulador para π_{IPC-2}
Comportamento quando Alice honesta e Bob corrupto
1. $\mathcal{S}$ ao receber uma mensagem (passive corrupt) na porta $CSS_{infl.B}$: 1.1. Envia a mensagem (passive corrupt, B) na porta IPC_{infl}; 2. F_{IPC} ao receber (passive corrupt, B) na porta IPC_{infl}: 2.1. Envia a mensagem (state, B, x_0) na porta IPC_{leak}; 3. $\mathcal{S}$ ao receber a mensagem (state, B, x_0) na porta IPC_{leak}: 3.1. Envia a mensagem (state, B, x_0) na porta $CSS_{leak.B}$; 3.2. Gera o polinômio com os coeficientes $sh_n \mid sh_i = Aleatório \in F_q, \forall i = 1, \dots, n$; 3.3. Gera um par de chaves pública e privada E_{sk} e D_{sk}, respectivamente, para o algoritmo criptográfico de Paillier $\in \mathbb{F}_N$, onde $N > q$; 3.4. Cifra os coeficientes de $sh(x)$ com a chave privada D_{sk} criando o polinômio $sh\_c(x)$; 3.5. Armazena (mid, $sh(x)$); 3.6. Envia a mensagem (mid, $sh\_c(x) \mid E_{sk}, A, B$) na porta CSS_{leak}; 3.7. Gera o número $sw \mid sw = Aleatório \in F_q$ e armazena (mid, sw); 3.8. Envia a mensagem (state, B, sw) na porta $CSS_{leak.B}$; 3.9. Cifra o número $sh\_c(sw) = E_{Esk}(sh(sw))$; 3.10. Envia a mensagem (state, B, $sh\_c(sw)$) na porta $CSS_{leak.B}$; 3.11. Envia a mensagem (mid, $sh\_c(sw), B, A$) na porta CSS_{leak}; 3.12. Decifra $sh\_c(sw)$ calculando $sh\_d(sw) = D_{Dsk}(sh\_c(sw))$; 3.13. Envia a mensagem (mid, $sh\_d(sw), A, B$) na porta CSS_{leak}; 3.14. Envia a mensagem (state, B, $sh\_d(sw)$) na porta $CSS_{leak.B}$;

Figura 5.7: Comportamento Simulador $\mathcal{S}$ – Alice honesta e Bob corrupto

Com já exposto anteriormente, a partir do momento em que Bob é corrompido o Simulador $\mathcal{S}$ passa a conhecer todos os seus dados, o que incluiu o ponto x_0 a ser testado, pois, novamente, F_{IPC} entrega esses dados na porta IPC_{leak} e este por sua vez repassa para a porta $CSS_{leak.B}$.

Mais uma vez há uma diferença entre o que a composição $\pi_{IPC-2}F_{CSS}$ e $F_{IPC} \diamond \mathcal{S}$ pois agora Bob também possui os dados necessários para a execução da fase off-line:

a) A chave pública de Paillier Ek;

b) O polinômio $h\_c(x)$;

c) O valor aleatório w;

d) O valor $h\_c(w)$ enviado para Alice;

e) O valor $h\_d(w)$ enviado por Alice;

A funcionalidade ideal F_{IPC} não gera esses dados por não fazer parte do seu comportamento.

Conforme pode ser visualizado na Figura 5.7, mais uma vez o Simulador $\mathcal{S}$ simula todos esses valores de forma aleatória e vaza na porta $CSS_{leak.B}$.

Devido ao fato de serem valores aleatórios $\mathcal{Z}$ não é capaz de distinguir entre os gerados pela composição $\pi_{IPC-2} \diamond F_{CSS}$ e aqueles simulados pela composição $F_{IPC} \diamond \mathcal{S}$.

Novamente, devido ao fato do recurso F_{CSS} vazar o conteúdo de todas as mensagens para o observador $\mathcal{Z}$, além de simular os valores listados anteriormente, o Simulador $\mathcal{S}$ necessita também tratar os casos do vazamento das mensagens trocadas durante a execução do protocolo.

Estas são as exatas mesmas mensagens que aquelas apresentadas e simuladas com sucesso na seção 5.2.1.

Para o caso de corrupção ativa, Bob pode decidir enviar $l \neq E_{Esk}(h\_c(w))$ diferente, portanto, de $h\_c(w)$ para Alice. Repare que para tanto o observador $\mathcal{Z}$ deve explicitamente enviar este novo valor para Bob através de uma mensagem (send,$CSS_{in.B}$, l) na porta $CSS_{infl.B}$ quando manipulando a composição $\pi_{IPC-2} \diamond F_{CSS}$.

Ao fazer isso na composição $F_{IPC} \diamond \mathcal{S}$ o Simulador $\mathcal{S}$ aprende o valor l e como $\mathcal{S}$ possui tanto a chave pública quanto a chave privada ele pode decifrar este valor com sucesso, fazendo o papel de Alice, devolvendo na sequência a mensagem (send, $CSS_{out.B}$, $D_{Dsk}(l)$) na porta $CSS_{infl.B}$, simulando assim com sucesso este comportamento.

Porém, assim como no cenário anterior o protocolo emitiria um resultado incorreto através de uma influência não permitida, não sendo seguro, portanto, contra adversário ativos.

Tendo abordado todos os quatro cenários, conseguimos criar um Simulador $\mathcal{S}$ que que satisfaz a condição (EQ 5.1) e, dessa forma, provar o Teorema 2.

Repare que devido aos cenários em que Alice é corrupta e Bob honesto e também Alice honesta e Bob corrupto não poderem ser provados contra adversários ativos o protocolo somente é seguro contra adversários passivos.

Salienta-se, por último, que devido ao uso do algoritmo criptográfico de Paillier e o recurso de comunicação F_{CSS} vazar todas as mensagens enviadas, o Protocolo IPC Sem TPC - Versão 2 é apenas computacionalmente seguro, pois $\mathcal{Z}$ possuindo poder computacional ilimitado conseguiria quebrar a cifra e assim distinguir entre as mensagens da composição $\pi_{IPC-2} \diamond F_{CSS}$ e aquelas simuladas por $F_{IPC} \diamond \mathcal{S}$.

6 - PROTOCOLO IPC SEM TPC - FASE ÚNICA – VERSÃO 3

Esta terceira versão do protocolo visa implementar a interseção privada de conjuntos, com o mesmo resultado e invariantes que aqueles presentes na Versão 1, porém além de não utilizar uma Terceira Parte Confiável, executar em uma única fase online, sem necessitar de uma fase off-line.

Além desta alteração esta terceira versão do protocolo visa permitir a Bob avaliar a interseção com mais do que apenas um elemento.

O ponto de partida é verificar que o objetivo da fase off-line apresentada na Versão 2, seção 5, é, na prática, o mesmo objetivo daquele presente na fase online, ou seja, Alice possui um polinômio aleatório de grau n cujas raízes são os elementos de seu conjunto $\mathbb{A}$, e Bob possui agora um conjunto de pontos $\mathbb{B}$ e deseja saber a avaliação dos elementos $\in \mathbb{B}$ no polinômio de Alice.

As invariantes são:

a) Alice não aprende qualquer dado sobre os pontos de Bob, a não ser a sua quantidade;
b) Alice não aprende qualquer dado sobre as avalições de seu polinômio nos pontos de Bob;
c) Bob não aprende qualquer dado sobre o polinômio aleatório de Alice, a não ser seu tamanho;
d) Um observador externo, com poder computacional polinomial, não é capaz de obter nenhum dado de Alice ou Bob.

A estratégia, apresentada na Figura 6.1, é baseada na fase off-line observada na Figura 5.1, que foi inspirada na proposta introduzida por (Freedman *et al.*, 2004).

Protocolo IPC Sem TPC - Fase Única - Versão 3

Fase Única Online

1. Parâmetros de entrada:
   1.1. Alice possui um conjunto $\mathbb{A}$ de elementos $\in \mathbb{F}_q$, que são as raízes do polinômio $a(x)$ de grau n;
   1.2. Bob possui um conjunto $\mathbb{B}$ com m elementos $\in \mathbb{F}_q$;
2. Alice:
   2.1. Gera um par de chaves pública e privada Ek e Dk, respectivamente, para o algoritmo criptográfico de Paillier $\in \mathbb{F}_N$, onde $N > q$;
   2.2. Cifra os coeficientes de $a(x)$, com a chave privada Dk, criando o polinômio $a\_c(x)$;
   2.3. Envia $a\_c(x)$ e Ek para Bob;
3. Bob:
   3.1. Recebe $a\_c(x)$ e Ek de Alice;
   3.2. Para todo elemento $b_j \in \mathbb{B}$ calcula:
      3.2.1. A avaliação de $a\_c(b_j) = \prod_{i=0}^{n} a\_c(x)_i{}^{u}$
      $onde\ u = {b_j}^i\ e\ a\_c(x)_i\ representa\ o\ iésimo\ coeficiente\ de\ a\_c(x)$;
   3.3. Gera o vetor de valores aleatórios $o_m | o_l = Aleatório \in F_q, \forall l = 1, \dots, m$;
   3.4. Ofusca as posições de $a\_c(b_j)$ calculando $a\_c(b_j) = a\_c(b_j) * E_{ek}(o_j)$ $\forall j = 1, \dots, m$;
   3.5. Envia $a\_c(b_j)$ para Alice;
4. Alice:
   4.1. Recebe $a\_c(b_j)$ de Bob;
   4.2. Decifra todos os m elementos calculando $a\_c(b_j)$ como $a\_d(b_j) = D_{dk}(a\_c(b_j))$;
   4.3. Envia $a\_d(b_j)$ para Bob;
5. Bob:
   5.1. Recebe $a\_d(b_j)$ de Alice;
   5.2. Calcula $a(b_j) = a\_d(b_j) - o_j, \forall j = 1, \dots, m$;
      5.2.1. Se $b_j = 0$ então $b_j \in \mathbb{A}$;
      5.2.2. Caso contrário $b_j \notin \mathbb{A}$;

Figura 6.1: Protocolo IPC Sem TPC - Fase Única - Versão 3

6.1 - PROVA DE CORRETUDE

Considerando que a fase online apresentada na Figura 6.1 é, essencialmente, a mesma que a fase off-line apresentada Figura 5.1, cabe salientar que Bob agora passa a utilizar um vetor de elementos aleatórios para impedir que Alice aprenda a avaliação dos elementos de $\mathbb{B}$.

Fora este ponto, todos os aspectos necessários para provar que o protocolo proposto é correto já foram apresentados na Versão 2.

Repare que agora Bob pode testar não apenas um único elemento, mas vários.

6.2 - PROVA DE SEGURANÇA NO MODELO UC

Teorema 3. O Protocolo IPC sem TPC – Fase Única - Versão 3 é Seguro no Modelo UC, com segurança estática, para adversários passivos e com poder computacional polinomial.

Novamente a prova do Teorema 3 segue as exatas mesmas linhas que àquelas apresentadas para versão 2 do protocolo, pois continua-se a utilizar o algoritmo criptográfico de Paillier.

Ao invés de ofuscar um único elemento com o valor aleatório $delta$ da Versão 2, Bob agora usa o vetor de valores aleatórios o_m para ofuscar, de forma independente, cada avaliação homomórfica que calculou do polinômio $a\_c(x)$ de Alice.

7 - RESULTADOS E ANÁLISE COMPARATIVA

No presente capítulo são apresentados os resultados dos testes de performance executados.

Antes, no entanto, é necessário informar as configurações utilizadas:

a) O tamanho do primo q de F_q é 512 bits;
b) O tamanho da chave pública do algoritmo de Paillier é 2048 bits, seguindo a recomendação presente na seção 3.4;
c) O equipamento referente ao item "a)" da seção 1.3 foi utilizado para executar as tarefas desenhadas para Alice;
d) O equipamento referente ao item "b)" da seção 1.3 foi utilizado para executar as tarefas desenhadas para Bob;
e) A conexão utilizada entre os equipamentos é 1GbE direta.

O tempo de execução referente à transferência de dados entre Alice e Bob foi negligenciável nos testes realizados.

Vale ressaltar que todos as execuções foram repetidas 3 vezes.

7.1 – DESEMPENHO DO PROTOCOLO IPC COM TPC - VERSÃO 1

Podemos visualizar os tempos de execução na Figura 4.1.

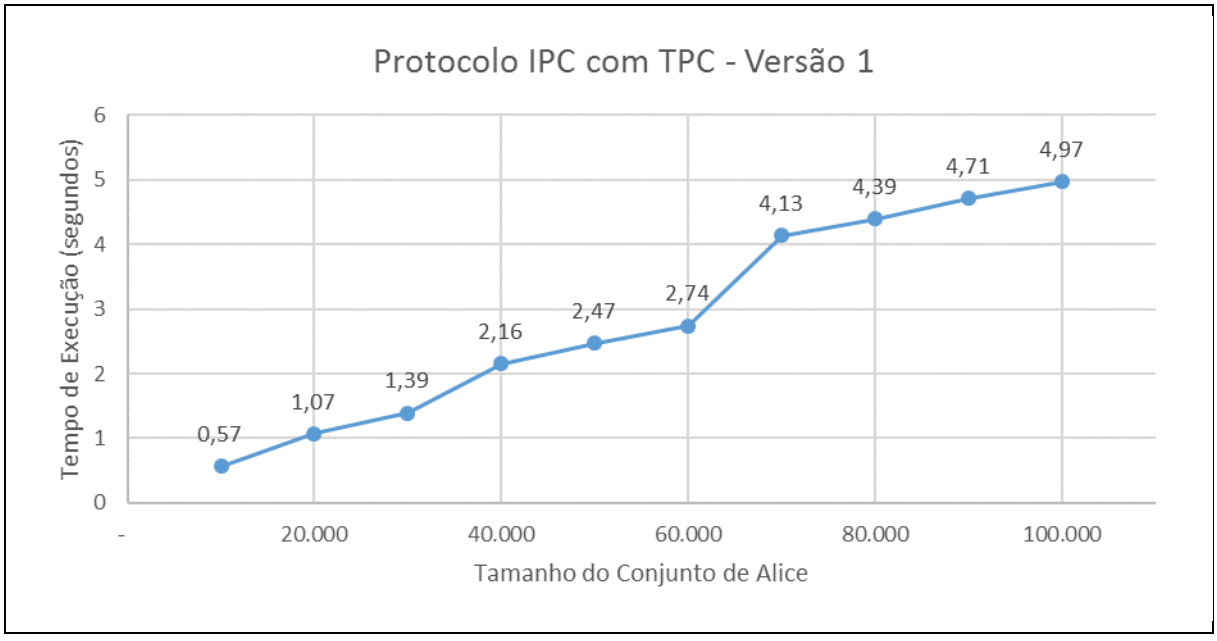


Figura 7.1: Performance do Protocolo IPC com TPC - Versão 1

Vale lembrar que nesta versão do protocolo Bob testa apenas um elemento no conjunto de Alice.

O tempo de execução começa a contar a partir do momento em que Alice e Bob já carregaram em memória os valores gerados pela TPC, ou seja, $f(x)$, d e g, e Bob está para enviar o valor t para Alice.

O tempo de execução interrompe após Bob receber $f(x)$ de Alice e calcular $f(d) - g$.

7.2 – DESEMPENHO DO PROTOCOLO IPC SEM TPC - VERSÃO 2

Como a fase online da Versão 2 do protocolo é idêntica à Versão 1 apenas são apresentados os tempos de execução da fase off-line, que visa simular uma TPC, presente na Figura 7.2.

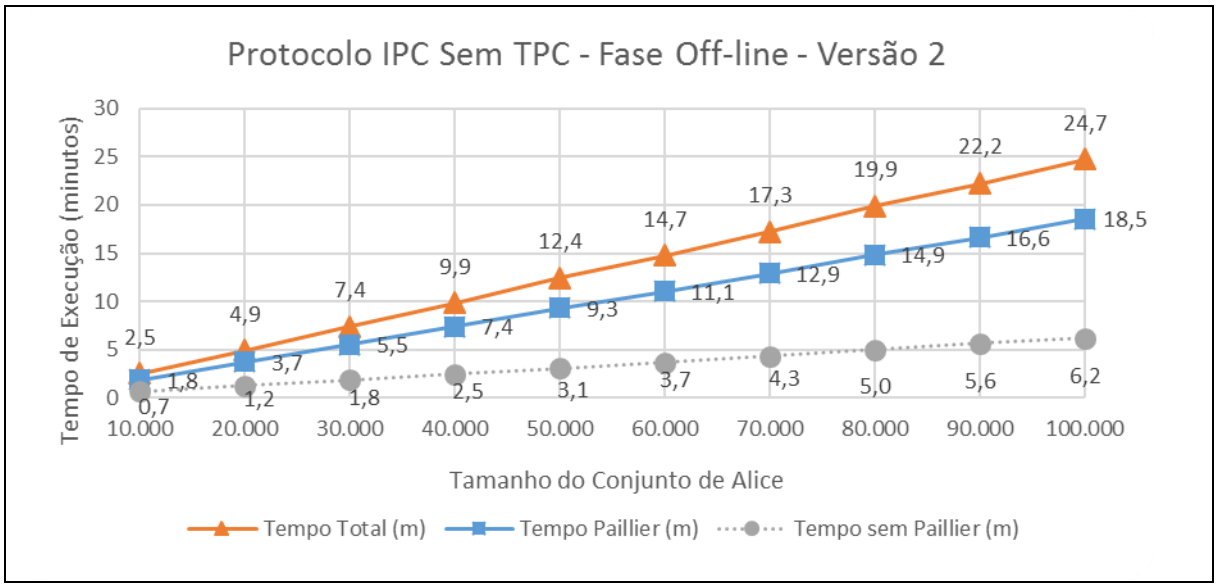


Figura 7.2: Performance do Protocolo IPC Sem TPC – Fase Off-line - Versão 2

Nesta Versão 2 do protocolo foram utilizados dois tempos de início de execução:

a) A primeira contagem de tempo - t_1- começa antes de Alice cifrar os coeficientes de seu polinômio $h(x)$ com a chave privada de Paillier, referente a etapa 1.1.3 presente na Figura 5.2;
b) A segunda contagem de tempo - t_2- começa após Alice ter terminado a cifragem mencionada acima e antes de enviar estes dados para Bob, referente a etapa 1.1.5 presente na Figura 5.2.

O tempo de execução - t_3- interrompe após Bob calcular $h(w)$, referente a etapa 2.3.1 presente na Figura 5.2;

Os tempos presentes na Figura 7.2, que representam as 3 linhas do gráfico, são calculados da seguinte forma:

a) $Tempo\ total\ (m) = t_3 - t_1$;

b) $Tempo\ sem\ Paillier\ (m) = t_3 - t_2$;

c) $Tempo\ Paillier\ (m) = t_2 - t_1$.

Vale ressaltar que enquanto a unidade de medida de tempo presente na Figura 7.1 é em segundos, a unidade de medida aqui presente é em minutos.

Como é possível observar, parte representativa do tempo total de execução, em todos os tamanhos do conjunto de Alice, 75%, ocorre justamente na cifragem dos coeficientes do polinômio de Alice.

Note também que Bob, nesta Versão 2, avalia, novamente, apenas um único elemento no conjunto de Alice.

7.3 – DESEMPENHO DO PROTOCOLO IPC SEM TPC – FASE ÚNICA - VERSÃO 3

Na Versão 3 do protocolo não mais é utilizada uma TPC, assim como não é necessário simulá-la, e também Bob pode testar mais do que apenas um elemento no conjunto de Alice.

Sendo assim foram feitos 3 cenários de teste, variando entre eles o tamanho do conjunto de Alice que apresenta os tamanhos de 10.000, 50.000 e 100.000 elementos conforme Figura 7.3, Figura 7.4 e Figura 7.5, respectivamente.

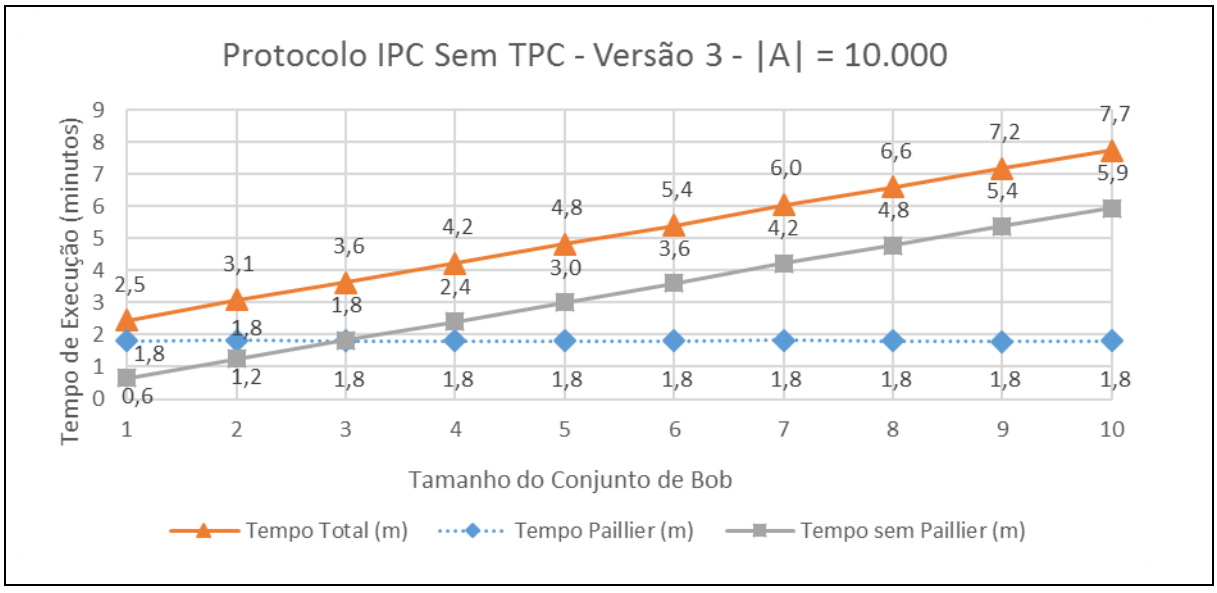


Figura 7.3: Performance do Protocolo IPC Sem TPC - Versão 3 – Alice 10.000 elementos

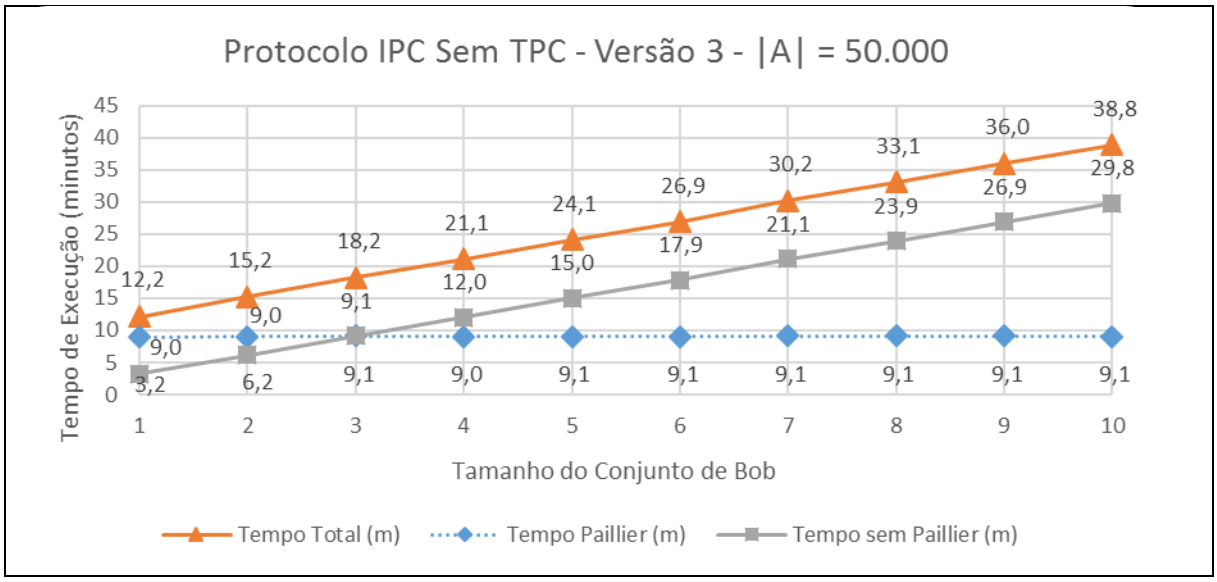


Figura 7.4: Performance do Protocolo IPC Sem TPC - Versão 3 – Alice 50.000 elementos

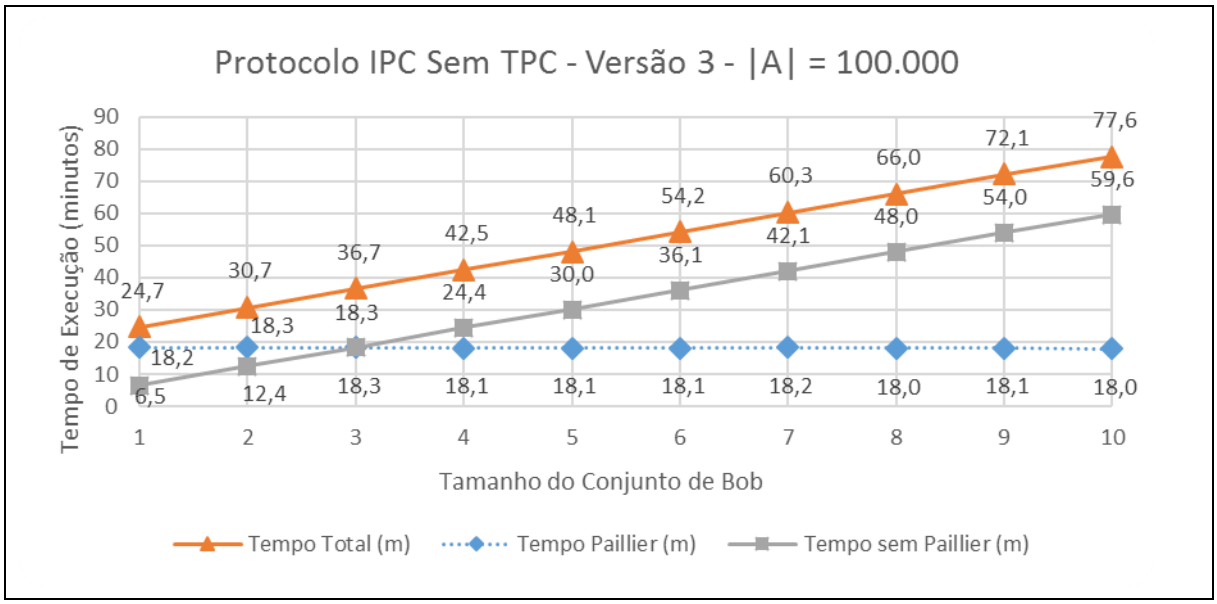


Figura 7.5: Performance do Protocolo IPC Sem TPC - Versão 3 – Alice 100.000 elementos

Os tempos de contagem são exatamente os mesmo que àqueles apresentados na seção 7.2.

Note que em todos os 3 gráficos a medida referente a $Tempo\ Paillier\ (m)$ se mantém constante e isso se deve ao fato de que o tamanho do conjunto de Alice é o mesmo enquanto se aumenta o tamanho do conjunto de Bob, portanto, o tempo necessário para cifrar o conjunto de Alice é o mesmo.

Vale ressaltar também que quando o tamanho do conjunto de Bob é maior que 3, a maior parte do tempo de execução passa a ocorrer no processamento de Bob e não de Alice. Isto ocorre

devido a etapa de avaliação homomórfica que Bob precisa realizar referente a etapa 3.2 presente na Figura 6.1.

Por último, note que enquanto Alice utilizada uma função otimizada de Paillier[6] para cifrar seus coeficientes, que usa a chave privada e o Teorema Chinês dos Restos, Bob precisa usar a função de cifragem mais lenta que utiliza a chave pública.

Em uma análise comparativa verificamos que as etapas que apresentam o tempo de execução mais representativo nas três versões do protocolo são:

a) Versão 1 – Com TPC: Cálculo do polinômio $f(x)$, referente a etapa 1.3.1.1 presente na Figura 4.6, que utiliza a função *BuildFromRoots()* da Biblioteca NTL[7];
b) Versão 2 – Alice e Bob simulando TPC: Cifragem dos coeficientes do polinômio $h(x)$ de Alice com a chave privada de Paillier, referente a etapa 1.1.3 presente na Figura 5.2, cujo código fonte é apresentado no Anexo A;
c) Versão 3 – Alice e Bob Fase Única, quando o tamanho do conjunto de Bob $|\mathbb{B}| \leq 3$: Cifragem dos coeficientes do polinômio $a(x)$ de Alice com a chave privada de Paillier, referente a etapa 2.2 presente na Figura 6.1, cujo código fonte é apresentado no Anexo A;
d) Versão 3 – Alice e Bob Fase Única, quando o tamanho do conjunto de Bob $|\mathbb{B}| > 3$: Avaliação homomórfica dos pontos de Bob no polinômio cifrado de Alice $a\_c(x)$ com o uso da chave pública de Paillier, referente a etapa 3.2 presente na Figura 6.1, cujo código fonte é apresentado no Anexo A;

Como é possível observar, os tempos de execução das Versões 2 e 3 do protocolo IPC ainda não são pequenos o suficiente para serem usados em aplicações reais, entretanto, temos que ressaltar que a implementação não foi construída utilizando programação paralela, o que fez com o que durante toda a execução, tanto de Alice quanto de Bob, apenas um núcleo dos quatro disponíveis nos processadores fosse usado por vez.

Desta forma uma implementação que apenas tivesse como objetivo usar, em paralelo, todos os núcleos do computador já reduziria o tempo de execução por um fator de 4, e isto considerando a execução em um computador de uso pessoal.

[6] Vide seção 3.6
[7] Vide seção 3.2

Caso fossem utilizados servidores de rede profissionais com doze núcleos ou mais teríamos uma redução ainda maior do tempo de execução o que eventualmente viabilizaria a aplicação das Versões 2 e 3 do Protocolo IPC em casos reais.

Ressalta-se ainda que esta questão se aplica apenas ao Versão 3 do Protocolo, dado que na Versão 2 as etapas listadas anteriormente são executadas numa fase off-line, o que garante um tempo maior de execução sem impactar a fase online subsequente.

8 - CONCLUSÕES

No presente trabalho são propostas três versões de um protocolo que implementa a Interseção Privada de Conjuntos (*Private Set Intersection - PSI*) entre dois participantes, Alice e Bob.

Foram construídas as estruturas conceituais e realizadas as provas de segurança no modelo Universalmente Composto (*Universally Composable - UC*) das três versões do protocolo.

Considerando as pesquisas bibliográficas foram obtidas as seguintes realizações no presente trabalho:

a) Primeira implementação da versão 1 do protocolo, descrito na Seção 4;
b) Primeira prova de segurança, no modelo Composto Universalmente (UC), de um protocolo que implementa a Interseção Privada de Conjuntos (IPC) contra adversários ativos e com poder computacional ilimitado, considerando a versão 1 do protocolo;

Os resultados das análises dos tempos de execução, das três versões do protocolo, mostraram que:

a) A versão 1 apresenta tempo de execução da ordem de grandeza de segundos;
b) As versões 2 e 3, descritas na Seção 5 e 6, respectivamente, apresentam tempo de execução da ordem de grandeza de minutos;
c) 75% do tempo de execução da fase off-line, da versão 2 do protocolo, ocorre na etapa de cifragem dos coeficientes do polinômio de Alice, através do uso do algoritmo criptográfico de Paillier, mesmo tendo essa implementação utilizado otimizações;
d) O tempo de execução das versões 1, 2 e 3 é diretamente proporcional ao tamanho do conjunto de entrada, com crescimento linear.
e) Na versão 3 do protocolo, em que Bob pode testar mais de que um único elemento, em todos os tamanhos do conjunto de entrada de Alice testados, o tempo de execução de Bob passava a ser maior que o de Alice quando o tamanho do seu conjunto superava quatro ou mais elementos.

Cabe salientar que apesar do tempo de execução da fase off-line da versão 2 do protocolo ser da ordem de minutos, esta é uma etapa que pode ser executada previamente, e que a fase online dessa versão 2 apresenta tempo de execução idêntico ao da fase online da versão 1 do protocolo que, conforme mencionado acima, foi da ordem de grandeza de segundos.

Como trabalho futuro a ser realizado está a implementação de paralelismo na etapa de cifragem dos coeficientes do polinômio de Alice, que tem grande potencial de redução no tempo de execução nas Versões 2 e 3 do protocolo.

Essa implementação pode ser feita simplesmente dividindo-se os coeficientes do polinômio de maneira igualitária entre os núcleos disponíveis nos processadores de forma que cada um realize a cifragem, de forma independente, utilizando o algoritmo homomórfico de Paillier.

Outro aspecto que deve ser ressaltado é que a utilização do algoritmo de Paillier não é mandatória, sendo possível a mudança para um futuro algoritmo criptográfico de chave pública que permita a avaliação homomórfica de polinômio e que possua um tempo de execução inferior.

9 - REFERÊNCIAS BIBLIOGRÁFICAS

Beaver D. (1997). Commodity-Based Cryptography (Extended Abstract). In *STOC 1997*, p. 446-455.

Bell D. (1976). The Coming of Post-Industrial Society: A Venture in Social Forecasting: Basic Books.

Bellare M., Rogaway P. (1993). Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proceeding CCS '93 Proceedings of the 1st ACM conference on Computer and communications security*, p. 62-73.

Cramer R., Damgård I., Nielsen J. B. (2015). Secure Multiparty Computation and Secret Sharing: Cambridge University Press.

Cristofaro E. (2011). Sharing Sensitive Information with Privacy. Tese de Doutorado. Universidade da Califórnia, EUA.

Cristofaro E., Tsudik G. (2012). On the performance of certain Private Set Intersection protocols. In *Cryptology Eprint Report 2012/54*.

Dachman-Soled D., Malkin T., Raykova M., Yung M. (2009). Efficient Robust Private Set Intersection. In *ACNS*, p. 125–142.

Damgård I., Jurik M. (2001). A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography. P. 119-136. Springer-Verlag.

Dong C., Chen L., Wen Z. (2013). When private set intersection meets big data: An eficiente and scalable protocol. In *Computer and Communications Security (CCS'13)*, p. 789–800. ACM.

Freedman M., Nissim K., Pinkas B. (2004). Efficient private matching and set intersection. In *EUROCRYPT*, p. 1–19.

Giry D. (2016). BlueKrypt - cryptographic key length recommendation. Online Documentation [http://www.keylength.com]

Hazay C., Nissim K. (2010). Efficient Set Operations in the Presence of Malicious Adversaries. In *PKC*, p. 312–331.

Huang Y., Evans D., Katz J. (2012). Private Set Intersection: Are Garbled Circuits Better than Custom Protocols? In *NDSS*.

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

Katz J., Lindell Y. (2015). Introduction to Modern Cryptography: CRC Press.

Kissner L., Song D. (2005). Privacy-preserving set operations. In *CRYPTO*, p. 241–257.

Naor M., Pinkas B. (1999). Oblivious Transfer and Polynomial Evaluation. In *STOC 1999*, p. 245-254.

Paillier P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology EUROCRYPT 99*, p. 223-238.

Pinkas B., Schneider T., Zohner M. (2014). Faster Private Set Intersection based on OT Extension. In *Cryptology Eprint Report 2014/447*.

Pullonen P. (2013). Actively Secure Two-Party Computation: Efficient Beaver Triple Generation. Tese de Mestrado. Instituto de Ciência da Computação, Universidade de Tartu, Estônia.

Pullonen P., Bogdanov D., Schneider T (2012). The design and implementation of a two-party protocol suite for SHAREMIND 3. In *Technical report, CYBERNETICA Institute of Information Security*.

Tonicelli R., Nascimento A. C. A., Dowsley R., Müller-Quade J., Imai H., Hanaoka G., Otsuka A. (2015). Information-Theoretically Secure Oblivious Polynomial Evaluation in the Commodity-Based Model. In *International Journal of Information Security*, Volume 14, Issue 1, p. 73-84.

Shoup V. (2016). NTL: A Library for doing Number Theory – Online Documentation [http://www.shoup.net/ntl/].

ANEXO A – BIBLIOTECA PAILLIER OTIMIZADA

```
// The Paillier protocol implementation here follows the optimization
recommended in page 4
// of the paper "The design and implementation of a two-party protocol
suite for SHAREMIND 3" by Pille Pullonen, et. al, 2013
#include <NTL/ZZ.h>

using namespace NTL;

class protocol_OPE_Paillier
{

public:
      void keygen(long modulusbits)
      {
            DEBUG("***** Initializing Paillier public and private keys
*****");

            DEBUG("Paillier bit key length N is: " << 2*modulusbits );
            ZZ temp_chinese_coef_p_n, temp_chinese_coef_q_n,
temp_chinese_coef_p_n2, temp_chinese_coef_q_n2, temp_GCD, r_temp;

            p = GenPrime_ZZ(modulusbits, 80);
            q = GenPrime_ZZ(modulusbits, 80);

            n = p * q;

            lambda = (p - 1) * (q - 1);

            n_squared = power(n, 2);
            p_squared = power(p, 2);
            q_squared = power(q, 2);

            two_power_modulusbits_p_q = power2_ZZ(modulusbits);
            two_power_modulusbits_n = power2_ZZ(2*modulusbits);

            u_p = InvMod(p, two_power_modulusbits_p_q);
            u_q = InvMod(q, two_power_modulusbits_p_q);
            u_n = InvMod(n, two_power_modulusbits_n);
            g = n + 1;

            h_p = InvMod( l_p_x(PowerMod(g%p_squared, p - 1,
p_squared)), p);
            h_q = InvMod( l_q_x(PowerMod(g%q_squared, q - 1,
q_squared)), q);
            h_n = InvMod( l_n_x(PowerMod(g, lambda, n_squared)), n);

            //CRT Coefs for dechipering
            XGCD(temp_GCD, temp_chinese_coef_p_n,
temp_chinese_coef_q_n, p, q);

            chinese_coef_p = MulMod(temp_chinese_coef_q_n, q, n);
            chinese_coef_q = MulMod(temp_chinese_coef_p_n, p, n);

            //CRT Coefs for chipering
            XGCD(temp_GCD, temp_chinese_coef_p_n2,
temp_chinese_coef_q_n2, p_squared, q_squared);
```

```
            chinese_coef_p2 = MulMod(temp_chinese_coef_q_n2, q_squared,
n_squared);
            chinese_coef_q2 = MulMod(temp_chinese_coef_p_n2, p_squared,
n_squared);

            DEBUG("Paillier public key N generated");
      }

      void initialize_with_public_key_n(ZZ public_key_n)
      {
            n = public_key_n;
            n_squared = power(n, 2);
      }

      ZZ enc_with_public_key( ZZ open_message )
      {
            ZZ r, cipher_message;

            do {
                  r = RandomBnd(n);
            } while( GCD(r,n)!=1 );

            cipher_message = MulMod(n, open_message, n_squared);

            return MulMod(cipher_message + 1, PowerMod(r, n,
n_squared), n_squared);
      }

      ZZ enc_with_private_key( ZZ open_message )
      {
            ZZ r, cipher_message_p, cipher_message_q,
cipher_message_combined;

            do {
                  r = RandomBnd(n);
            } while( GCD(r,n)!=1 );

            cipher_message_p = MulMod(n, open_message, p_squared);

            cipher_message_p = MulMod(cipher_message_p + 1,
PowerMod(r%p_squared, n, p_squared), p_squared);

            cipher_message_q = MulMod(n, open_message, q_squared);

            cipher_message_q = MulMod(cipher_message_q + 1,
PowerMod(r%q_squared, n, q_squared), q_squared);

            return ( MulMod(cipher_message_p, chinese_coef_p2,
n_squared ) + MulMod(cipher_message_q, chinese_coef_q2, n_squared) ) %
n_squared;
      }

      ZZ dec_with_pq( ZZ cipher_message )
      {
            ZZ m_p, m_q, temp_p, temp_q;

            temp_p = PowerMod(cipher_message%p_squared, p - 1,
p_squared);
            temp_q = PowerMod(cipher_message%q_squared, q - 1,
q_squared);
```

```
            m_p = MulMod( l_p_x( temp_p), h_p, p);
            m_q = MulMod( l_q_x( temp_q), h_q, q);

            return ( m_p*chinese_coef_p + m_q*chinese_coef_q) % n;
      }

      //Dec with lambda, which should be slower than dec with p and q,
is here just for testing purpose
      ZZ dec_with_lambda( ZZ cipher_message )
      {
            return MulMod( l_n_x( PowerMod(cipher_message, lambda,
n_squared) ), h_n, n);
      }

      ZZ mul( ZZ cipher_message_1, ZZ cipher_message_2)
      {
            return MulMod(cipher_message_1, cipher_message_2,
n_squared);
      }


      ZZ exp( ZZ cipher_message_1, ZZ open_message_2)
      {
            return PowerMod(cipher_message_1, open_message_2,
n_squared);
      }

      ZZ get_public_key_n( )
      {
            return n;
      }

private:
      ZZ l_p_x(ZZ x)
      {
            return MulMod(x - 1, u_p, two_power_modulusbits_p_q);
      }

      ZZ l_q_x(ZZ x)
      {
            return MulMod(x - 1, u_q, two_power_modulusbits_p_q);
      }

      ZZ l_n_x(ZZ x)
      {
            return MulMod(x - 1, u_n, two_power_modulusbits_n);
      }

ZZ n, p, q, lambda, n_squared, p_squared, q_squared, g, u_p, u_q, u_n,
h_p, h_q, h_n,
   chinese_coef_p, chinese_coef_q, chinese_coef_p2, chinese_coef_q2,
two_power_modulusbits_p_q, two_power_modulusbits_n;
```