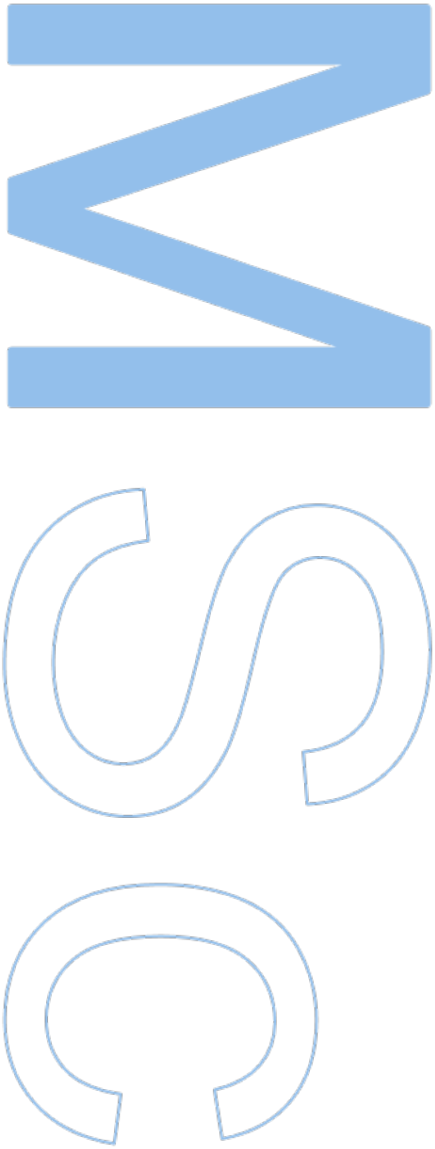
Privacy in Telecom Fraud Detection



Eduardo Carvalho Santos

Mestrado em Engenharia de Redes e Sistemas Informáticos

Departamento de Ciências de Computadores

2023

Orientador

Bernardo Luís Fernandes Portela, Professor Auxiliar, Faculdade de Ciências da Universidade do Porto

Coorientador

João Paulo da Silva Machado Garcia Vilela, Professor Auxiliar, Faculdade de Ciências da Universidade do Porto


U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Agradecimentos

Ao longo desta dissertação foram muitas as pessoas que me apoiaram. Um especial agradecimento:

- aos meus orientadores, Bernardo Portela e João Vilela por toda a disponibilidade, apoio e paciência durante a realização desta dissertação;
- à Mobileum pelo tema proposto, em especial ao Pedro Fidalgo, diretor de investigação e desenvolvimento, pela disponibilidade, paciência e apoio durante a realização desta dissertação;
- aos professores da Universidade do Porto e a equipa de CTFs xSTF pelo contributo no desenvolvimento pessoal e profissional;
- a todos os meus amigos e colegas da Universidade do Porto que sempre se mostraram disponíveis;
- a minha família por todo o suporte e ajuda no processo de concretização da dissertação.

O trabalho desta dissertação foi desenvolvido no âmbito do projeto AIDA: Adaptive, Intelligent and Distributed Assurance Platform (POCI-01-0247-FEDER-045907), co-financiado pelo Fundo Europeu de Desenvolvimento Regional através do programa COMPETE2020 e pela Fundação para a Ciência e a Tecnologia (FCT) sob o programa CMU Portugal.

Eu dedico este trabalho a minha família. A minha gratidão profunda pelo apoio inabalável que eles me proporcionaram ao longo desta jornada académica. Sem o seu amor, orientação e estímulo, esta dissertação não teria sido possível.

Resumo

A deteção e prevenção de fraude em telecomunicações são um tema de extrema importância na atualidade. Com o constante avanço das tecnologias de comunicação, surgem também novos desafios no que diz respeito à segurança das redes e dos utilizadores. As empresas de telecomunicações têm investido significativamente em sistemas avançados de deteção de fraude, utilizando algoritmos inteligentes e análise de dados em tempo real para identificar atividades suspeitas, como chamadas não autorizadas ou utilização indevida de serviços. Por outro lado, as autoridades reguladoras têm desempenhado um papel fundamental na criação de normas e regulamentos que visam proteger os consumidores e garantir a integridade das redes de telecomunicações. Estas normas e regulamentos dificultam estes mesmos algoritmos de deteção de fraude, pois os dados não podem ser observados de forma desprotegida.

O propósito desta dissertação é desenvolver soluções para deteção de fraude em telecomunicações, tendo em conta as atuais preocupações com a privacidade dos dados. Para abordar esta questão, foram criados dois protótipos com abordagens distintas.

O primeiro protótipo destina-se a situações em que não existe confiança mútua entre as partes envolvidas. Este utiliza técnicas criptográficas avançadas para permitir a deteção de números fraudulentos, através da distância de Hamming e algumas variações que permitem calcular a similaridade entre números potencialmente fraudulentos. O grande benefício deste protótipo é a sua versatilidade, já que pode ser utilizado em diversos sistemas e tipos de hardware. É particularmente útil para detetar fraudes após um incidente.

O segundo protótipo depende de hardware confiável, nomeadamente o Intel Software Guard Extensions (SGX), e oferece uma gama mais alargada de técnicas de deteção. Inclui a deteção de números fraudulentos, distância de hamming, deteção de prefixos e sufixos suspeitos e até mesmo a identificação de possíveis "sequenciações" que são variações de números em bloco para evitar serem detetados como spam. Além disso, este protótipo é suficientemente eficiente para ser utilizado em tempo real e pode ser facilmente adaptado para detetar novos padrões no futuro. Este protótipo mostrou-se promissor para outsourcing de computação e para deteção em tempo real.

Palavras-chave: Fraude em Telecomunicações, Privacidade de Dados, Computação Segura, Intel SGX, Hardware Confiável, Deteção Passiva, Deteção em Tempo Real

Abstract

The detection and prevention of fraud in telecommunications is an issue of extreme importance today. With the constant advancement of communication technologies, new challenges also arise with regard to the security of networks and users. Telecommunications companies have invested significantly in advanced fraud detection systems, using intelligent algorithms and real-time data analysis to identify suspicious activities such as unauthorized calls or misuse of services. On the other hand, regulatory authorities have played a key role in creating standards and regulations aimed at protecting consumers and ensuring the integrity of telecommunications networks. These standards and regulations make these same fraud detection algorithms difficult, as data cannot be observed unprotected.

The purpose of this dissertation is to develop solutions for detecting telecommunications fraud, taking into account current data privacy concerns. To address this issue, two prototypes were created with different approaches.

The first prototype is intended for situations where there is no mutual trust between the parties involved. This uses advanced cryptographic techniques to allow the detection of fraudulent numbers, through the distance of Hamming and some variations that allow to calculate the similarity between potentially fraudulent numbers. The great benefit of this prototype is its versatility, since it can be used in various systems and types of hardware. It is particularly useful for detecting fraud after an incident.

The second prototype relies on reliable hardware, notably Intel Software Guard Extensions (SGX), and offers a wider range of detection techniques. It includes detecting fraudulent numbers, Hamming distance, detecting suspicious prefixes and suffixes, and even identifying possible "sequencing" that are variations of block numbers to avoid being detected as spam. In addition, this prototype is efficient enough to be used in real time and can be easily adapted to detect new patterns in the future. This prototype proved promising for computing outsourcing and real-time detection.

Keywords: Telecommunications Fraud, Data Privacy, Secure Computing, Intel SGX, Trusted Hardware, Passive Detection, Real-Time Detection

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Blocos de Código

Acrónimos

AIDA Adaptive, Intelligent and Distributed Assurance Platform

API Application Programming Interface

AVX Advanced Vector Extensions

BC Bit Commitment

CMU Carnegie Mellon University

COT Committed Oblivious Transfer

DCAP Data Center Attestation Primitives

DE Distância Euclidiana

DH Diffie-Hellman

DH Distância de Hamming

ECALL Enclave CALL

EPC Enclave Page Cache

EPID Enhanced Privacy ID

IA Inteligência Artificial

IND-CPA Indistinguishability under chosen-plaintext attack

INESC TEC Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência

IoT Internet of Things

ML Machine Learning

OCALL Outside Enclave CALL

OKVS Oblivious Key-Value Stores

OPRF Oblivious Pseudo-Random Function

OT-E Oblivious Transfer Extensions

OT Oblivious Transfer

PAL Platform Adaptation Layer

PaXoS Probe-and-XOR of Strings

PPTA algoritmo probabilístico de tempo polinomial

PSI Private Set Intersection

RA-TLS Remote Attestation with Transport Layer Security

RGPD Regulamento Geral da Proteção de Dados

RSA Rivest-Shamir-Adleman

S2PC Secure Two-Party Computation

SDK Software Development Kit

SGX Software Guard Extensions

SHD Secure Hamming Distance

SMC Secure Multiparty Computation

TEE Trusted Execution Environment

TLS Transport Layer Security

TZ TrustZone

UC Universidade de Coimbra

UE União Europeia

VOLE Vector Oblivious Linear Evaluation

Capítulo 1

Introdução

O trabalho desta dissertação surge a partir de casos de uso cedidos pela empresa Mobileum no contexto do consórcio Adaptive, Intelligent and Distributed Assurance Platform (AIDA).

A AIDA é um consórcio composto por quatro parceiros da academia e da indústria, Mobileum, Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência (INESC TEC), Universidade de Coimbra (UC) e Carnegie Mellon University (CMU). Um dos propósitos do consórcio é conciliar garantias de privacidade e confidencialidade para lidar com um número crescente de fontes de dados, enquanto extrai-se informações valiosas dos dados que podem ser confiáveis, um problema complexo considerando múltiplos proprietários de dados ou domínios administrativos diferentes.

A Mobileum é uma empresa global que fornece soluções de análise de dados e serviços de *roaming* para a indústria de telecomunicações. A Mobileum utiliza análise de dados avançada e Inteligência Artificial (IA) para auxiliar as operadoras a otimizar as suas redes e serviços, bem como a detetar e prevenir fraudes e problemas de segurança.

A deteção de fraude, em concreto da Mobileum, é constituída por um conjunto de algoritmos, que, posteriormente, usa os seus resultados para alimentar modelos de IA. Os algoritmos usados são:

1. Deteção de prefixos suspeitos – Exemplo, chamadas vindas de países com alta incidência de fraude.
2. Deteção de blocos de sufixos – Exemplo, blocos conhecidos de números pre-pagos.
3. Deteção de números fraudulentos – Cruzamento com listas públicas de outras empresas de números reconhecidos como fraudulentos.
4. Deteção de proximidade com números conhecidos usando distância de Hamming – Exemplo, usar um número com um dígito apenas de diferença do banco nacional.
5. Deteção de tentativas de sequenciação – Entende-se por sequenciação fazer chamadas de

conjunto de números numericamente seguidos ou de forma aleatória, mas num bloco muito curto. É uma técnica usada para evitar serem detetados com fraudulentos.

Com a entrada do Regulamento Geral da Proteção de Dados (RGPD) [17, 18] em vigor, os números de telemóveis são considerados dados pessoais, e como tal, estes algoritmos deixam de poder serem aplicados da mesma forma. A Mobileum é uma empresa mundial, como tal possui datacenters em vários locais do mundo. No caso da União Europeia (UE) os dados deixam de poder sair do espaço da União em estado puro. Os algoritmos dependem dos números em estado puro para funcionarem [17, 18]. Por outro lado, a Mobileum para além dos números, poderia inferir quem liga e quando para os seus clientes, o que é uma clara violação do RGPD, pois os dados dos clientes estão a ser comunicados a outra empresa sem o seu consentimento [17, 18].

A empresa testou o uso de técnicas criptográficas padrão para proteger os dados. Já que as técnicas criptográficas comuns, por design, eliminam padrões na informação que protegem, não permitem computação sobre os dados cifrados. O desafio surge neste limbo entre conseguir realizar operações sobre dados que precisam de estarem protegidos, respeitando o RGPD, e ainda assim conseguir realizar computações sobre eles.

1.1 Contribuições

O desafio principal é repor a funcionalidade das diversas operações descritas anteriormente no novo contexto de privacidade. A pedido da Mobileum espera-se usar menos de 200ms para resposta as querys. Eles também informaram que mesmo resultados com tempos superiores podem ser interessantes noutros contextos como deteção de fraude offline. Nestes casos os tempos podem ir até uma semana para a deteção de fraude. Por outras palavras, tentar obter o menor tempo possível em cada operação, sendo abaixo de 200ms o ideal.

Embora existam diversas técnicas de criptografia avançadas para responder a algumas das operações, normalmente exigem mais tempo que os 200ms e não podem ser aplicadas a todos as operações solicitadas. Outro fator é a forma como é um número de telemóvel. Uma sequência de dígitos 12 dígitos entre “0” até “9”. Por ser um domínio tão curto, algumas das operações tornam-se vulneráveis, é o caso da deteção de prefixos em que só temos 3 dígitos para o prefixo. O que perfaz 1000 possibilidades apenas. Para dificultar ainda mais, a distribuição também não é uniforme. Temos mais números em certos países que noutros.

A nossa contribuição são dois protótipos, um baseado em software que pode ser usado pela Mobileum para interagir com qualquer entidade e um baseado em hardware que pretende respondem em concreto ao problema da Mobileum e a CMU, mas que pode ser generalizado para o uso geral.

No primeiro protótipo, contribuímos com uma implementação inovadora, baseada em software, de deteção de similaridade entre números. Isto é conseguido a custa do Secure Hamming Distance e as várias variantes criadas. Algumas das variantes possuem menos precisão, mas têm um custo

menor associado. Já as outras têm uma precisão superior mas também têm um custo superior. Isto garantindo a privacidade dos dados associados. No entanto, só terá utilidade na maioria dos casos para deteção de fraude offline.

No segundo protótipo, contribuímos com uma nova implementação de deteção de fraude recorrendo a hardware confiável Intel SGX. Neste protótipo, todas as operações propostas foram implementadas. Devido a um conjunto alargado de otimizações conseguimos responde em tempo real a querys em todo o tipo de operações, tendo sido armazenados até 4 milhões de números dentro do enclave.

1.2 Organização

O resto deste documento encontra-se dividido em 5 capítulos: Background, Estado da Arte, Computação Segura baseada em Software, Computação Segura baseada em Hardware e Conclusão.

Na Introdução (Capítulo 1) é apresentada a motivação e o contexto do problema, assim como os objetivos desta dissertação.

No Background (Capítulo 2) é apresentado mais detalhadamente o processo de deteção de fraude, as imposições legais por detrás do RGPD, uma pequena secção sobre criptografia, seguida de duas secções: técnicas de software que preservam a privacidade e outra de hardware confiável.

No Estado da Arte (Capítulo 3) é feita uma revisão bibliográfica das técnicas de software e hardware mais recentes incluídas neste trabalho: GSHADE, VolePSI e Gramine.

No Computação Segura baseada em Software (Capítulo 4) é apresentado o modelo e requisitos funcionais, o protocolo, a implementação e por fim a avaliação experimental do protótipo de software.

No Computação Segura baseada em Hardware (Capítulo 5) é apresentado o modelo e requisitos funcionais, o protocolo, a implementação e por fim a avaliação experimental do protótipo de hardware confiável.

Finalmente, a Conclusão (Capítulo 6) é feita uma avaliação geral dos objetivos propostos, assim como descritas algumas possíveis explorações e melhorias futuras para ambos os protótipos.

Capítulo 2

Background

Nesta secção, vamos começar por investigar como funciona a deteção de fraude em telecomunicações. A seguir falaremos do Regulamento Geral da Proteção de Dados (RGPD) que limitou a atuação da deteção de fraude. Depois iremos rever os conceitos de criptográfica básica, demonstrando o motivo de não poderem ser usados nesta situação. Terminamos com duas outras subsecções, uma orientada ao uso de algoritmos usando apenas software e outra recorrendo a tecnologias de hardware confiável.

2.1 Deteção de fraude

A deteção de fraude em telecomunicações é um campo importante de investigação e aplicação, pois as empresas de telecomunicações enfrentam perdas significativas de receita devido a atividades fraudulentas, como chamadas não autorizadas, utilização indevida de serviços, entre outros tipos de fraude [4, 7–9, 20].

Existem várias abordagens e técnicas que podem ser utilizadas para detetar fraudes em telecomunicações [4, 8, 9, 20]:

- Análise de Tráfego: Uma das formas mais comuns de deteção de fraude em telecomunicações é a análise de padrões de tráfego. Isso envolve a monitorização de dados de chamadas, mensagens e uso de dados para identificar comportamentos suspeitos. Por exemplo, se um utilizador começa a fazer um número anormalmente elevado de chamadas de longa distância ou a consumir uma quantidade de dados muito maior do que o habitual, isso pode ser indicativo de fraude.
- Análise de Comportamento: A deteção de fraude muitas vezes envolve a criação de perfis de utilizadores normais e a identificação de desvios significativos desse comportamento normal. Por exemplo, se um utilizador começa a utilizar serviços fora do seu padrão de comportamento (como fazer chamadas internacionais frequentes quando geralmente só faz chamadas locais), isso pode ser considerado suspeito.

- Análise de Regras: Muitas empresas de telecomunicações implementam regras específicas para detetar atividades fraudulentas. Por exemplo, se um utilizador tentar fazer chamadas internacionais após uma determinada hora (sem autorização explícita), isso pode acionar um alerta. Essas regras são geralmente definidas com base em políticas internas da empresa para limitar as perdas.
- Inteligência Artificial (IA) e Machine Learning (ML): Técnicas de IA, ML, têm sido cada vez mais usadas na deteção de fraudes em telecomunicações. Os algoritmos de ML podem analisar grandes volumes de dados e identificar automaticamente padrões de fraude que podem não ser óbvios para sistemas baseados em regras. Isso pode incluir a deteção de fraudes complexas, como a clonagem de cartões SIM ou a falsificação de identidade.
- Análise de Redes Sociais: Em alguns casos, a análise de redes sociais pode ser usada para identificar conexões entre utilizadores suspeitos de fraude. Por exemplo, se várias contas fraudulentas estiverem associadas a uma rede de amigos online, isso pode ser um indicador.
- Monitorização em Tempo Real: A deteção de fraude em telecomunicações muitas vezes requer a capacidade de monitorizar o tráfego em tempo real. Isso permite que as empresas de telecomunicações respondam rapidamente a atividades suspeitas e tomem medidas para minimizar as perdas.
- Análise de Fraude Interna e Externa: As fraudes podem ser perpetradas tanto por utilizadores externos quanto por funcionários internos das empresas de telecomunicações. Portanto, a deteção de fraude envolve, muitas vezes, a análise de ambos os tipos de atividade fraudulenta.

Os modelos de IA e ML tem sido atualmente mais explorados nesta área devido a sua capacidade de detetar novos tipos de ataques, que não são reconhecidos pelas análises baseadas em regras. Por consequência, a não deteção prematura deste tipo de ataques leva a custos elevados a longo prazo [8, 9, 20, 21].

2.2 Regulamento Geral da Proteção de Dados

O Regulamento Geral da Proteção de Dados (RGPD) é uma legislação da União Europeia (UE) que entrou em vigor em 25 de maio de 2018. Foi desenvolvida visando estabelecer um conjunto único de regras de proteção de dados para todos os países membros da UE, garantindo maior privacidade e controle sobre os dados pessoais dos cidadãos europeus [16–18].

Várias alterações na sociedade potenciaram a criação desta legislação. A mais impactante foi o crescimento exponencial das tecnologias de informação e rápida expansão da internet. Como consequência, foi observado um aumento significativo na coleta de dados pessoais e metadados, posterior processamento e armazenamento por parte de várias entidades. Este rastreamento começou a criar preocupação sobre a privacidade dos indivíduos e possível uso inadequado dos

seus dados por parte de empresas, organizações e até estados. Este problema obtive maior destaque quando começaram a aparecer dados pessoais resultado de ciberataques a venda ou mesmo disponibilizados publicamente. As violações de dados tornaram-se frequentes, o que fiz com que as autoridades começassem a levantar preocupações sobre a segurança dos sistemas que armazenam dados [24, 25, 33].

Com o RGPD a proteção da privacidade e dos dados pessoais é considerado um direito fundamental de todos os cidadãos da UE [16–18]. Esta legislação foi concebida para garantir que as empresas e instituições tratem os dados pessoais com respeito e conforme os direitos fundamentais dos indivíduos. Ela enfatiza o conceito de consentimento informado e explícito, dando aos indivíduos maior controlo sobre o uso dos seus dados pessoais. Permite que as pessoas acedam, corrijam ou mesmo solicitem a exclusão dos seus dados armazenados por empresas e organizações. O RGPD visou fortalecer a segurança dos dados e impor penalizações para organizações que não cumprem adequadamente as regras de proteção de dados.

O RGPD estabelece critérios rigorosos para a transferência de dados pessoais para países terceiros, que não pertencem à UE [17, 18]. Quando uma empresa dentro da UE precisa transferir dados para fora do espaço da União, ela deve garantir que o país de destino ofereça um nível adequado de proteção de dados. Caso o país não atenda a esse padrão, a transferência só pode ocorrer se forem implementadas salvaguardas adicionais, como cláusulas contratuais ou regras corporativas vinculativas. Essa restrição visa proteger a privacidade dos cidadãos europeus, evitando que os seus dados sejam transferidos para jurisdições com normas de proteção inadequadas. No entanto, essa questão pode criar desafios para empresas globais, que precisam cumprir regulamentações diferentes em cada país, criando um ambiente complexo de conformidade. Além disso, a aplicação do RGPD fora da UE pode ser difícil, uma vez que as autoridades de proteção de dados da União têm jurisdição limitada em outras regiões, tornando a garantia dos direitos de proteção de dados além das fronteiras da UE um desafio adicional.

2.3 Criptografia

Nesta secção iremos tratar conceitos básicos de criptografia como segurança semântica. Por sua vez iremos rever a indistinguibilidade contra ataques de texto-limpo escolhido, o formalismo usado para garantir segurança nas cifras simétricas. Depois iremos mais a fundo em conceitos como hash criptográficas. É importante rever estes conceitos para compreender porque são limitadas no contexto do problema a tratar.

2.3.1 Segurança Semântica

Segundo Goldwasser and Micali [22], um sistema criptográfico é semanticamente seguro se recorrermos a um algoritmo probabilístico de tempo polinomial (PPTA) que recebe o texto cifrado e o comprimento da mensagem inicial contra outros PPTA que só tem acesso ao comprimento da

mensagem inicial e não conseguir obter qualquer informação parcial sobre a mensagem inicial com uma probabilidade significativa. A assunção de segurança surge do conhecimento do comprimento da mensagem inicial e do texto cifrado de alguma mensagem desconhecida e não é revelada nenhuma informação adicional sobre a mensagem inicial [1].

2.3.2 Indistinguibilidade contra ataques de texto-limpo escolhido

A Indistinguibilidade contra ataques de texto-limpo escolhido, mais conhecida por Indistinguishability under chosen-plaintext attack (IND-CPA), é um formalismo usado para analisar a segurança dos esquemas de criptografia relativamente à propriedade de indistinguibilidade sob ataque de texto claro escolhido. Essa propriedade é crucial para garantir que um esquema criptográfico seja resistente a ataques em que um adversário tem a capacidade de escolher mensagens para serem cifradas e depois analisar as respostas do esquema [1].

1. Temos duas entidades, um adversário e um oráculo. O oráculo representa o esquema criptográfico a ser analisado. Ele será responsável por cifrar as mensagens e fornecer os criptogramas resultantes. O adversário é quem vai interagir com o oráculo.
2. O adversário pode enviar pares de mensagens de texto claro ao oráculo. Para cada par de mensagens, o oráculo cifra-as e devolve os criptogramas ao adversário. O adversário pode analisar as mensagens enviadas e os seus criptogramas correspondentes.
3. O adversário, a dado momento, escolhe duas mensagens de texto claro distintas M_0, M_1. O oráculo seleciona um $b \in \{0, 1\}$ uniforme e aleatoriamente, e envia o criptograma C desafio de volta ao adversário.
4. Agora o adversário tem que descobrir qual das duas mensagens M_0, M_1 foi escolhido e cifrado pelo oráculo. Por outras palavras, descobrir o valor b usado pelo oráculo.
5. O adversário tem sucesso se adivinhar o valo b usado pelo oráculo. O esquema criptográfico é considerado seguro em relação ao IND-CPA se a probabilidade de o adversário acertar estiver próxima de 50%. Por outras palavras, o adversário não poder fazer uma distinção confiável entre os criptogramas.

Se um adversário pudesse adivinhar a mensagem correta com probabilidade significativamente maior do que 50%, isso indicaria que o esquema de criptografia possui uma falha de segurança. O objetivo do jogo é demonstrar que, mesmo quando o adversário possui controle sobre as mensagens escolhidas para serem cifradas, ele ainda deve ser incapaz de distinguir com confiança as cifras resultantes [1].

2.3.3 Hash criptográficas

Uma função hash é um algoritmo que mapeia dados de comprimento variável para dados de comprimento fixo. O resultado dessa função é conhecido como hash. As funções de hash devem ser rápidas e simples de computar.

As funções de hash criptográficas são um subgrupo dessas funções e que tem um conjunto de propriedades concretas. Estas funções têm que ser determinísticas, eficientes, não serem invertíveis, terem uma distribuição uniforme e têm que possuir resistência a alguns ataques, tais como [1]:

- Resistência a ataques de pré-imagem.
  - Dado um valor hash h deve ser difícil encontrar qualquer mensagem m tal que $h = hash(m)$.
  - Este conceito está relacionado ao da função de mão única (ou função unidirecional).
  - Funções que não possuem essa propriedade estão vulneráveis a ataques de pré-imagem.
- Resistência a ataque de segunda pré-imagem.
  - Dada uma mensagem $m1$ deve ser difícil encontrar outra mensagem $m2$ tal que $hash(m1) = hash(m2)$.
  - Funções que não possuem essa propriedade estão vulneráveis a ataques de segunda pré-imagem.
- Resistência a colisões.
  - Deve ser difícil encontrar duas mensagens diferentes $m1$ e $m2$ tal que $hash(m1) = hash(m2)$. Tal par é chamado de colisão hash criptográfica.
  - Essa propriedade também é conhecida como forte resistência à colisão. Ela requer um valor hash com pelo menos o dobro do comprimento necessário para resistência a pré-imagem.
  - Caso contrário, colisões podem ser encontradas por meio de um ataque de aniversário.

2.3.4 Discussão

Para podermos aplicar as abordagens que listamos na Secção 1 sobre dados protegidos, então temos que fazer cálculos como comparações, ordenações, ou outras funções de similaridade sobre criptogramas. Observe-se que, caso se possa fazer isso, então também se conseguirá vencer o jogo IND-CPA. Como tal, qualquer sistema seguro que assegure IND-CPA não permitirá esse tipo de cálculos sobre criptogramas. Será então necessário encontrar alternativas que ofereçam diferentes tipos de segurança.

Já nas funções de hash criptográficas, estas podem ser aplicadas aos números de telemóvel, de forma a permitir identificação de duplicados. Uma vulnerabilidade natural desta abordagem são os ataques offline: utilizar uma tabela pré-computada de outputs destas funções para identificar pré-imagens. Como resposta a estes problemas, é prática comum utilizar salted hash functions. Este é um método em que se adiciona uma string aleatória junto do input da função (número de telemóvel, neste caso), de forma a dificultar estes ataques offline. Sem conhecimento do "salt", não é possível pré-computar tabelas de valores input/output da função de hash. Porém, o domínio da função de hash (os números de telefone possíveis) é relativamente pequeno. Como tal, a construção de um ataque de dicionário para recuperar números a partir de resultados destas funções de hash é uma abordagem viável, já que o tempo e poder computacional necessário para este ataque é relativamente reduzido. Se gerarmos passwords usando apenas números de 0 a 9 consegue-se, usando o poder computacional atual, gerar todas as permutações de hash para cada número de 12 dígitos em períodos muito curtos [32].

2.4 Computação Segura baseada em Software

A primeira abordagem foi explorar algoritmos que permitam repor as operações usando o hardware já existente. As soluções baseadas em software têm, por natureza, um alto custo computacional e alto custo de comunicação. São bastante limitadas, por serem desenhadas apenas para a sua finalidade. A assunção de segurança surge pela prova matemática sobrejacente aos algoritmos.

2.4.1 Compromissos de Bits

Os Compromissos de Bits, mais conhecidos por Bit Commitment (BC), é um protocolo que atua entre duas entidades, Alice e Bob, que numa fase inicial a Alice envia um bit X e o Bob recebe a confirmação que a Alice comprometeu-se com um certo valor (sem o conhecimento de qual é o valor real que corresponde ao X). Após a operação a realizar, a Alice revela o valor X ao Bob. O Bob usando o valor enviado pela Alice previamente confirma o valor real [31].

Um exemplo prático deste protocolo é simularmos um envio de um cartão com um número de telefone escrito no interior de um cofre. A Alice envia o cofre com o cartão para o Bob enquanto mantém a chave na sua posse. Após a Alice enviar a chave para o Bob, o Bob confirma o número de telefone [31]. Atualmente o BC é uma primitiva usada, por exemplo, para gerar de forma segura valores aleatórios em ambientes não confiáveis.

De uma forma mais ampla, é usado em provas de *zero-knowledge*, o compromisso de bit é utilizado para demonstrar o conhecimento de uma informação sem revelá-la, fornecendo uma forma de autenticação e validação eficiente [31].

Geralmente o BC surge com a primitiva do Oblivious Transfer (OT). Quando combinados permite tornar o OT resistente a um adversário bizantino. Isto deve-se ao facto de impossibilitar tomar decisões baseadas nos dados recebidos.

2.4.2 Transferência Oblívia

Em criptografia, um protocolo de Transferência Oblívia, mais conhecido por Oblivious Transfer (OT), consiste num tipo de protocolo em que um emissor transfere uma sequência de bits para o recetor, mas sem revelar que bits foram enviados [12]. O OT é um protocolo em que age como uma caixa negra, permitindo apenas observar *inputs* e *outputs*. Geralmente, os protocolos OT exigem alto custo de comunicação e computação [5, 36].

O trabalho que tem sido realizado mostrou que o OT é fundamental para problemas em criptografia. Em alguns protocolos passou a ser o elemento fundamental que permite preservar a privacidade dos dados a medida que permite realizar computações. Um exemplo é o caso do Secure Multiparty Computation (SMC) [5, 36].

O OT é um componente usado em outros protocolos, normalmente visando obter privacidade nos dados a serem distribuídos [36]. Com a combinação de protocolos conseguimos escalar melhor reduzindo o consumo de recursos do processador de rede [36]. Tem casos em que passou a ser possível resolver problemas em tempo polinomial, garantindo na mesma a privacidade esperada quando comparado com protocolos similares, mas usando outras primitivas [36].

Os protocolos OT podem ser implementados em diferentes variantes. Na variante OT_1^2 o remetente envia dois bits (b_0, b_1), e o recetor envia apenas um bit c. No fim da execução do protocolo o recetor recebe b_c que é se $c = 0$ recebe b_0, se $c = 1$ recebe b_1. No entanto, o remetente não tem conhecimento do c [5, 36]. Segue um exemplo na figura 2.1.

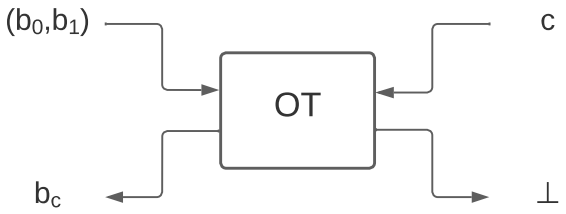


Figura 2.1: Estrutura básica de OT_1^2.

O OT_1^2 foi generalizado para n entidades e obtemos o OT_1^n. O OT_1^n é muito usado no contexto de SMC. No entanto, a complexidade computacional era na ordem do $O(n^2)$. Posteriormente foram criadas otimizações que permitiram reduzir a complexidade computacional e o número de mensagens usadas [5]. Segue um exemplo na figura 2.2.

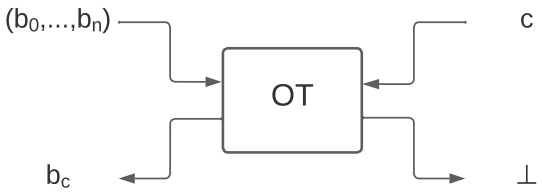


Figura 2.2: Estrutura básica de OT_1^n.

Extensões da Transferência Oblívia

Nesta secção, exploraremos em detalhe duas otimizações introduzidas por Ishai et al. [26] para Extensões da Transferência Oblívia, também designada como Oblivious Transfer Extensions (OT-E) [5, 6].

A primeira implementação do OT-E, descrita em [26], baseia-se num modelo de oráculo aleatório. Nesta abordagem, várias OT's são calculadas através da realização de k computações de OT, onde k é um parâmetro de segurança. Isto implica que o número total de OT's realizadas passa de n para apenas k [5, 6].

A segunda implementação do OT-E apresentada por Ishai et al. [26], concentra-se na redução do tamanho das strings de OT [5, 6]. Esta otimização é alcançada através da utilização de um gerador de números pseudo-aleatórios. A ideia subjacente é gerar strings mais curtas que representem as informações a serem transferidas, resultando numa significativa redução de recursos de rede quando comparada com a implementação original [6]. A utilização de um gerador de números pseudo-aleatório é crucial para assegurar que a redução do tamanho das strings seja feita de forma segura, sem comprometer a segurança da transferência [5, 6].

Estas otimizações no OT-E visa tornar as transferências de informações mais eficientes, reduzindo a carga na rede e a quantidade de computações necessárias, ao mesmo tempo que mantém um nível adequado de segurança [5]. Estas implementações ficaram conhecidas como IKNP.

Transferência Oblívia com Compromissos

A Transferência Oblívia com Compromissos, conhecida como Committed Oblivious Transfer (COT), é uma combinação de OT_1^2 e BC. Foi introduzida por Crépeau [15] sob o nome *Verifiable Oblivious Transfer* [5]. Esta variante, tanto o remetente quanto o recetor estão comprometidos com as suas entradas antes da OT [5, 15]. Além disso, o remetente recebe um compromisso com o *output* do recetor, e o recetor obtém a aleatoriedade para este compromisso [5, 15]. Um dos esquemas que considera COT para n bits é o de Kiraz et al. [27], que usa um sistema de criptografia homomórfica como esquema de compromisso [5]. O COT é resistente a um adversário malicioso. Por isso está fora do âmbito de protocolos explorados nesta dissertação.

Esquemas

O OT pode ser instanciado de diversas formas. Existem duas básicas, uma baseada em Rivest-Shamir-Adleman (RSA) [19] e outra baseada em Distância de Hamming (DH) [40].

Na figura 2.3 temos a instanciação baseado em RSA e que não dependem de uma entidade externa confiável para executar o protocolo.

1. Alice detém m_0, m_1, e um par de chaves RSA (e, d, N).
2. Bob detém a chave pública (e, N) de Alice, $c \in \{0, 1\}$ e quer m_c.
3. Alice gera aleatoriamente x_0, x_1 e envia-os ao Bob.
4. Bob escolhe um k aleatório, calcula $v = x_c + k^e$ e envia v para Alice.
5. Alice computa $k_0 = (v - x_0)^d$ e $k_1 = (v - x_1)^d$ e envia $m'_0 = m_0 + k_0$ e $m'_1 = m_1 + k_1$ para Bob.
6. Bob calcula $m_c = m'_c - k$ e não aprende nada sobre m_{1-c}.

Figura 2.3: Instanciação de OT_1^2 usando RSA (adaptado de Even et al. [19]).

O esquema baseado em DH está explicado na figura 2.4. Este esquema é depende de uma entidade externa confiável por ambas as entidades que vão executar o protocolo. No artigo, este terceiro é chamado de *Trusted Initializer*. Ele apenas está presente na fase inicial do protocolo. Este protocolo é uma extensão de um protocolo quântico proposto por Rivest [40].

1. Alice detém $m_0, m_1 \in \{0, 1\}^k$, Bob detém $c \in \{0, 1\}$ e quer m_c.
2. Ted envia em privada à Alice $r_0, r_1 \in \{0, 1\}^k$, escolhida ao acaso.
3. Ted gera um bit privado $d \in \{0, 1\}$ e envia em privado d, r_d para Bob.
4. Bob envia $e = c \oplus d$ para Alice.
5. Alice envia $f_0 = m_0 \oplus r_e$ e $f_1 = m_1 \oplus r_{1-e}$ para Bob.
6. Bob calcula $m_c = f_c \oplus r_d$. Bob não descobre nada acerca de m_{1-c}.

Figura 2.4: Instanciação de OT_1^2 usando DH (adaptado de Rivest [40]).

Modelo de segurança

O OT é seguro num cenário de um adversário semi-honesto. Quer isto dizer que um existe um adversário que executa o protocolo e observa mensagens trocadas entre ambas as partes. Mas ele limita-se a cumprir o protocolo definido e o máximo de informação que consegue é a que o protocolo lhe permite.

O OT não é imune a um adversário malicioso que irá tentar injetar mensagens e até não cumprir o protocolo para recuperar o máximo de informação possível [5].

Existem outras abordagens mais complexas de OT, como COT que são resilientes contra adversários maliciosos, mas que caem fora do âmbito de protocolos explorados nesta tese.

2.4.3 Distância de Hamming Segura

Distância de Hamming (DH) é uma soma das diferenças bit a bit. O valor resultante é o número de permutações distintas ao longo das sequências de bits. Distância de Hamming Segura, mais conhecido por Secure Hamming Distance (SHD), é uma construção que permite realizar o cálculo da DH entre duas entidades não confiáveis de forma segura. Nesta estrutura os participantes (um

ou ambos) recebem o resultado, neste caso a distância de Hamming e mais nenhuma informação sobre os inputs. Na figura 2.5 é descrito o esquema básico para o uso do SHD. Neste esquema uma entidade tem o vetor de bits X e a outra entidade tem o vetor de bits Y. O resultado será o cálculo da distância de Hamming.

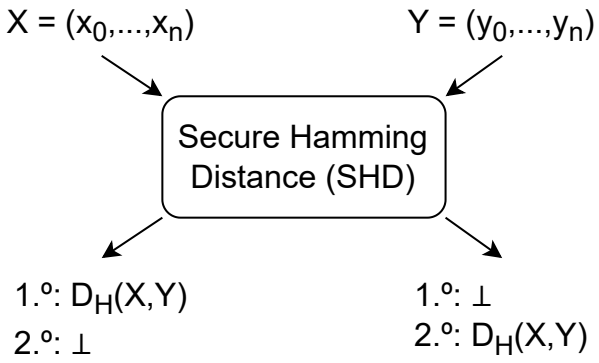


Figura 2.5: Estrutura básica do Secure Hamming Distance.

SHADE

Bringer et al. [5] descreveu um protocolo que permite calcular o DH e manter os dados cifrados, garantindo assim a sua privacidade. A estrutura foi batizada de SHADE. No caso, o SHD é uma construção que usa o OT. Na figura 2.6 está descrita uma versão simplificada do SHD. Este esquema é útil no sentido de entender que dados são trocados no protocolo. A figura 2.7 descreve o protocolo SHD detalhadamente. Esta figura contém todos os passos da execução do protocolo.

A estrutura SHADE [5] é resistente a um adversário semi-honesto. Por outras palavras, um adversário que cumpre as restrições impostas pelo esquema e recupera apenas esta informação. Na implementação do SHADE recorre-se ao uso de OT para tornar ambíguo quais os valores de X e Y. No final apenas é revelado o somatório dos aleatórios e não os de cada posição em concreto. Assim sendo, apenas é possível inferir o resultado da DH, e nada mais.

2.4.4 Interseção Segura de Sets

A Interseção Segura de Sets, conhecida como Private Set Intersection (PSI), é um protocolo criptográfico que permite a comparação segura de conjuntos de dados entre duas partes sem revelar os elementos específicos dos conjuntos [30, 35, 36]. É frequentemente usado para determinar se dois conjuntos têm elementos em comum sem compartilhar informações confidenciais. Na figura 2.8 é descrito a estrutura básica de qualquer PSI.

Na secção 2.3, o uso de hashs mostrou-se inseguro no contexto das telecomunicações. As implementações antigas usavam cruzamento de hashs, o que é um processo inseguro. Outro problema é que o envio das hashs para outra entidade permite sem mais interação com a outra

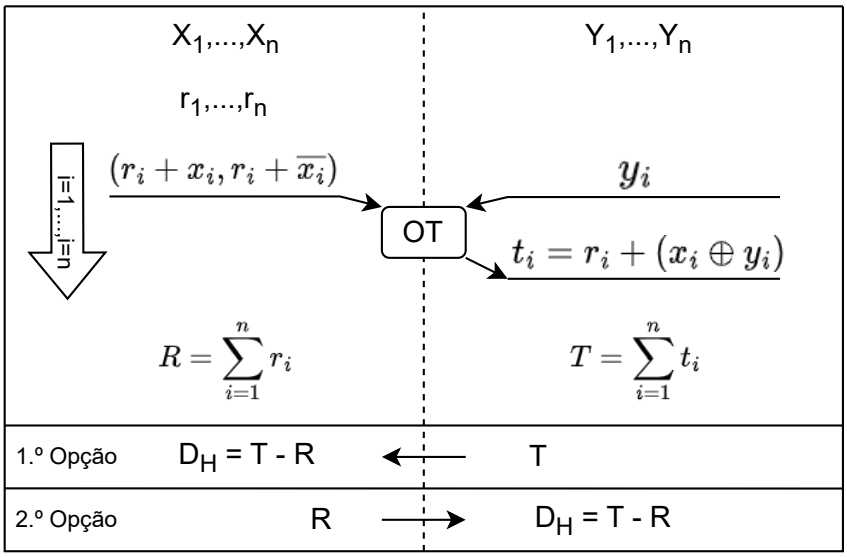


Figura 2.6: Protocolo simplificado do Secure Hamming Distance.

Input:

P_1 tem como input uma n-bit string $X = (x_1, \ldots, x_n)$.

P_2 tem como input uma n-bit string $Y = (y_1, \ldots, y_n)$.

Output:

1.º Opção: P_1 obtém $d_H(X, Y)$ e P_2 não obtém nada.

2.º Opção: P_2 obtém $d_H(X, Y)$ e P_1 não obtém nada.

Protocolo:

1. P_1 gera n valores aleatórios $r_1, \ldots, r_n \in_R \mathbb{Z}_{n+1}$ e calcula $R = \sum_{i=1}^{n} r_i$.
2. Para cada $i = 1, \ldots, n$, P_1 e P_2 envolvem-se num OT_1^2 onde:

   P_1 atua como o remetente e P_2 como o recetor.

   P_2 escolhe o bit y_i.

   P_1 envia $(r_i + x_i, r_i + \overline{x_i})$.

   O output obtido por P_2 é o resultado de calcular $t_i = r_i + (x_i \oplus y_i)$.
3. P_2 calcula $T = \sum_{i=1}^{n} t_i$.
4. 1.º Opção: P_2 envia T para P_1, P_1 calcula e devolve $T - R$.

   2.º Opção: P_1 envia R para P_2, P_2 calcula e devolve $T - R$.

Figura 2.7: Protocolo detalhado do Secure Hamming Distance (adaptado de Bringer et al. [5]).

parte realizar uma infinidade de computações com esses dados. O PSI vem resolver o problema, pois cada interseção obtida exige interação com a outra entidade, o que permite limitar, por exemplo, a realização de ataques de força bruta.

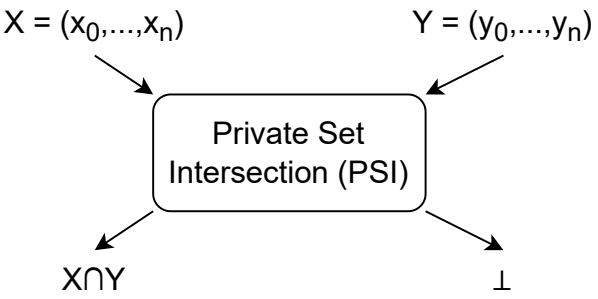


Figura 2.8: Estrutura básica do Private Set Intersection.

PSI usando Distância de Hamming

Existem várias implementações de PSI. O mais primordial dos modelos é o modelo de PSI usando DH [30, 35, 36], descrito na figura 2.9. Nesta estrutura a privacidade é conseguida à custa do uso do DH. O DH usado para gerar um segredo partilhado entre cada par de entidades. Os hashs ficam elevadas aos expoentes, garantindo a propriedade anterior. Esta é a implementação mais eficiente a nível de consumo de rede, mas a mais cara a nível de computação.

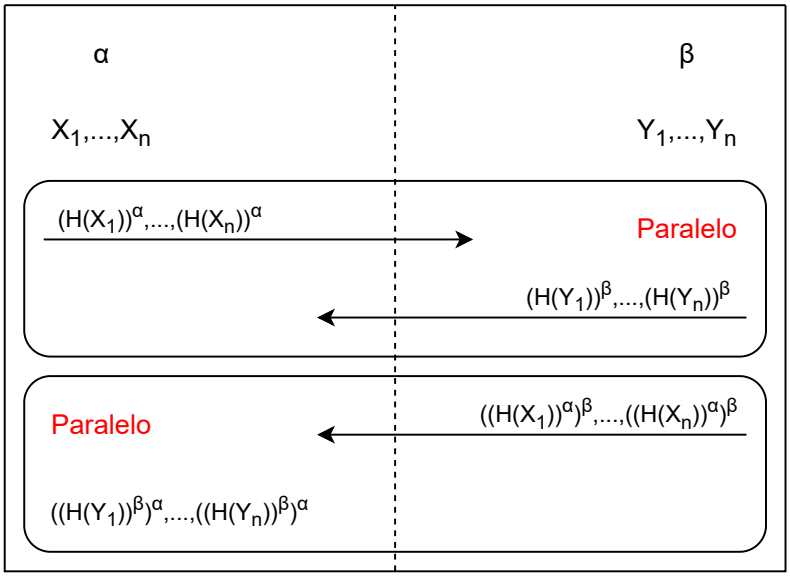


Figura 2.9: Estrutura detalhada do Private Set Intersection Distância de Hamming.

PSI usando Funções Pseudo-aleatórias Oblívias

Por fim também surgiram, por volta de 1980, implementações de PSI mas usando Funções Pseudo-aleatórias Oblívias, também designadas por Oblivious Pseudo-Random Function (OPRF) [10, 30, 35, 36]. A privacidade é adquirida à custa do uso de OT. Esta é uma implementação mais cara a nível de consumo de rede, mas mais eficiente a nível de computação. Este problema tem sido mitigado usando as duas técnicas de OT-E anteriormente descritas na subsecção 2.4.2.

Bloom Filter e Cuckoo Filter

O *bloom filter* e *cuckoo filter* são estruturas de dados probabilísticas e que permitem representar grandes conjuntos de dados em estruturas de dados menores. Estas estruturas são bastante úteis em aplicações onde é necessário realizar consultas de existência rápida como o caso do PSI [30, 35, 36].

O *bloom filter* usa uma estrutura de dados probabilística para representar várias posições no *set* real. A única certeza é quando não existe correspondência, não existem elementos iguais no dataset. É uma estrutura eficiente no consumo de memória. No entanto, à medida que a quantidade de elementos inseridos aumenta, a taxa de falsos positivos também aumenta. Quanto maior o filtro, menor a taxa de falsos positivos, mas maior o consumo de memória. Quanto maior o filtro, menor a quantidade de colisões entre posições, o que melhora o desempenho a custo de mais memória. O *bloom filter* distingue-se do *cuckoo filter* pois oferece uma solução mais compacta para a verificação de existência, mas não suporta a exclusão eficiente de elementos.

O *cuckoo filter* é outra estrutura de dados probabilística semelhante ao *bloom filter* mas com uma implementação distinta. Ele utiliza duas funções hash independentes para calcular duas posições possíveis para armazenar um elemento no conjunto de dados. Se pelo menos uma dessas posições estiver vazia, o elemento é inserido. Caso ambas as posições estejam ocupadas, um elemento existente pode ser substituído. Ao verificar a existência de um elemento, o *cuckoo filter* recalcula as mesmas duas funções hash e verifica se algum desses locais contém o elemento procurado. No entanto, devido à possibilidade de colisões e substituições, existe uma pequena probabilidade de falsos positivos. Comparado ao *bloom filter*, o *cuckoo filter* possui uma capacidade de exclusão mais eficiente.

2.5 Computação Segura baseada em Hardware

A segunda abordagem foi explorar o uso de hardware confiável. O problema principal é que impõe uma assunção de confiança muito forte no manufator deste hardware. No entanto, estas abordagens são muito mais eficientes e escaláveis. A grande vantagem é que permite correr algoritmos altamente complexos com um pequeno *overhead* associado. Este tipo de abordagem também exige um desenho rigoroso da aplicação.

2.5.1 Ambientes de Execução Confiáveis

Os Ambientes de Execução Confiáveis, também conhecidos como Trusted Execution Environment (TEE), são uma área segura num sistema computacional onde operações sensíveis que podem ser executadas com proteção adicional [41]. Dois exemplos notáveis de tecnologias de TEE são o ARM TrustZone (TZ) e o Intel Software Guard Extensions (SGX) [41].

Um TEE deve atender a certos requisitos para ser considerado como tal. Segue uma lista de

algumas das características que um TEE deve ter:

- Isolamento: Um TEE deve fornecer um ambiente isolado dentro do sistema, separado do sistema operativo e de outros aplicações em execução. Isso garante que as operações sensíveis dentro do TEE sejam protegidas contra acesso não autorizado.
- Segurança do código: O TEE deve garantir a integridade e a autenticidade do código que é executado dentro dele. Isso geralmente envolve técnicas como assinaturas digitais para verificar a autenticidade do código.
- Proteção de dados: O TEE deve oferecer proteção para dados sensíveis, garantindo que eles não sejam acessíveis a aplicações ou componentes do sistema não autorizados. Isso pode envolver criptografia dos dados em repouso e em trânsito, além de mecanismos de proteção contra ataques de leitura e gravação não autorizados.
- Interface segura: O TEE deve fornecer uma interface segura para comunicação com aplicações confiáveis e componentes externos. Isso envolve a implementação de protocolos criptográficos e a proteção contra ataques de manipulação ou intercetação de dados.

No entanto, é importante reconhecer que os TEEs também possuem limitações [41]. Algumas das limitações incluem:

- Dependência de hardware: A implementação de um TEE muitas vezes depende de recursos específicos de hardware, o que pode limitar a sua portabilidade para diferentes plataformas ou dispositivos que não suportem esses recursos específicos.
- Ataques físicos: Embora um TEE forneça proteção contra ataques lógicos, como acesso não autorizado a dados ou códigos sensíveis, ele pode ser vulnerável a ataques físicos, como análise de canais laterais ou remoção do chip para acesso direto aos dados.
- Vulnerabilidades de software: Assim como qualquer outro software, os TEEs também podem ter vulnerabilidades que podem ser exploradas por ataques cibernéticos. A aplicação de patches e atualizações de segurança é essencial para mitigar essas vulnerabilidades.
- Confiança em hardware: Um TEE geralmente depende de recursos de hardware confiáveis, como mecanismos de criptografia e isolamento de memória fornecidos por processadores específicos. Esses recursos ajudam a fortalecer a segurança e a confiabilidade do TEE.

2.5.2 Intel Software Guard Extensions

O Intel Software Guard Extensions (SGX) é uma extensão de hardware que permite a criação de enclaves seguros dentro do processador [2, 3, 14]. Esses enclaves são áreas isoladas de memória que protegem a execução de código e os dados sensíveis para resistir a ataques físicos e de software. Com o SGX, os desenvolvedores podem criar aplicações que executam operações confidenciais num ambiente seguro, mesmo em ambientes não confiáveis [14].

Funcionamento

Os enclaves no contexto do SGX são áreas isoladas de memória protegida que executam código confidencial de forma segura. Eles são projetados para garantir a confidencialidade e integridade dos dados, mesmo quando o sistema operativo ou outras aplicações estejam comprometidas [14].

Ao criar um enclave, o programador especifica uma região de memória protegida que será designada como enclave. Essa região é criada e gerida pelo hardware [14]. Quando um enclave é lançado, ocorre o processo de inicialização. Durante essa etapa, o enclave é carregado na memória protegida e o hardware verifica a autenticidade do código do enclave por meio de uma assinatura digital. Isso garante que apenas o código criado e compilado original é o mesmo código que está a ser executado dentro do enclave. Uma vez que o enclave é inicializado, ele entra num estado seguro e isolado do restante do sistema. Isso significa que o código do enclave é executado num ambiente confidencial, protegido contra acessos não autorizados ou modificações externas. Quando o enclave está em execução, os dados confidenciais que estão a ser manipulados são protegidos. O SGX fornece mecanismos de criptografia e verificações de integridade para garantir que os dados sejam mantidos em segurança [14].

Os enclaves podem interagir com o mundo externo por meio de chamadas a funções definidas pelas APIs do SGX. Estas funções podem ser do tipo Enclave CALL (ECALL) e Outside Enclave CALL (OCALL). Estas chamadas de função permitem que o enclave aceda a recursos fora do ambiente protegido, como input/output, comunicação com outros componentes do sistema ou acesso a chaves criptográficas [14].

É importante ressalvar que o código fora do enclave não tem acesso direto aos dados ou código dentro do enclave. O enclave é isolado e protegido pelo hardware, o que significa que o sistema operativo, outras aplicações e até mesmo o hipervisor não pode aceder aos dados dentro do enclave sem a autorização apropriada. O isolamento é ao nível de hardware. Diferente de outros sistemas, o nível de isolamento até impede que os enclaves consigam ler o conteúdo uns dos outros [14].

Atestação Local e Remota

A Atestação Local é um processo pelo qual um enclave verifica a sua própria integridade e autenticidade. Durante a Atestação Local, o enclave gera uma assinatura digital dos seus dados e envia essa assinatura para ser verificada internamente. Isso permite que o enclave ateste para si que está a ser executado num ambiente confiável, sem comprometimento da sua integridade ou segurança [3, 14].

A Atestação Remota vai além da Atestação Local e envolve a verificação do enclave por uma entidade externa, como um serviço em nuvem ou um terceiro confiável. Durante a Atestação Remota, o enclave gera uma assinatura dos seus dados e envia essa assinatura para uma entidade externa, com outras informações relevantes, como identificação e versão do enclave. A entidade

externa verifica a assinatura e as informações para determinar se o enclave é autêntico e confiável. Esta atestação pode ser realizada usando Data Center Attestation Primitives (DCAP) ou Enhanced Privacy ID (EPID) [3, 14].

O DCAP é um conjunto de bibliotecas fornecido pela Intel que permite a implementação de atestação remota em ambientes de datacenters. Ele fornece as ferramentas necessárias para gerar e verificar assinaturas digitais, bem como para estabelecer a comunicação segura entre o enclave e a entidade externa responsável pela verificação [3, 14].

O EPID é um protocolo criptográfico usado no contexto de Atestação Remota para garantir a privacidade dos participantes envolvidos na verificação. Ele permite que um enclave seja verificado sem revelar a identidade ou informações pessoais do enclave, ou do proprietário. O EPID utiliza técnicas criptográficas de *zero-knowledge* para garantir que a entidade externa verifique a autenticidade do enclave sem conhecer detalhes específicos sobre a sua origem [3, 14].

Vulnerabilidades

Os enclaves não protegem contra todas as formas de ataques. Vulnerabilidades como *side channel attacks* podem permitir que um invasor obtenha informações sensíveis, como chaves criptográficas, por meio da análise de padrões de consumo de energia ou tempos de execução. Portanto, é necessário levar em consideração esses aspetos e implementar medidas adicionais de segurança, conforme necessário, para garantir uma proteção abrangente dos sistemas que utilizam enclaves.

2.5.3 ARM TrustZone

ARM TrustZone (TZ) é uma tecnologia de segurança projetada para fornecer um ambiente seguro em sistemas baseados na arquitetura ARM [23, 29]. Ela foi introduzida pela ARM Holdings e é amplamente utilizada em dispositivos como smartphones, tablets, smartwatches e outros dispositivos embebidos.

A ideia por trás do TZ é criar dois "mundos" distintos num único sistema, o "Mundo Seguro" (Secure World) e o "Mundo Normal" (Normal World), isolados um do outro. O "Mundo Seguro" é protegido por hardware e software para fornecer uma área confiável para executar tarefas sensíveis, enquanto o Mundo Normal é onde as tarefas regulares do sistema são executadas. Isto é implementado em hardware recorrendo a um bit. O bit é mais conhecido por bit de controlo de segurança (Security Control Bit). Quando o bit de controlo de segurança está definido como 0, o processador ARM está no modo "Mundo Normal", e o sistema executa tarefas regulares nesse modo. Por outro lado, quando o bit de controlo de segurança está definido como 1, o processador entra no modo "Mundo Seguro", ativando o ambiente seguro e fornecendo proteção adicional para operações sensíveis [23].

A arquitetura do TZ é baseada em dois modos de execução: o "Modo Monitor" (Monitor Mode) e o "Modo Normal" (Normal Mode). O Modo Monitor é responsável pela gestão da

transição entre os mundos seguro e normal, além de fornecer serviços de segurança adicionais, como autenticação e integridade. Já o "Modo Normal" é onde a maioria das operações ocorre e é onde o sistema operativo e as aplicações são executados [23].

A separação entre os dois mundos é implementada por meio do uso de registos especiais no processador ARM, chamados registos de controlo de segurança (Security Control Registers), que definem a configuração e o comportamento do TZ. Esses registos controlam a transição entre os modos seguro e normal, ativando ou desativando a proteção de memória e determinando quais recursos são acessíveis em cada mundo [23].

Uma das principais vantagens do TZ é a capacidade de executar software seguro num ambiente isolado, protegido contra ameaças externas. Isso é especialmente importante em dispositivos que lidam com informações sensíveis, como chaves de criptografia, dados biométricos, informações de autenticação, entre outros. Ao fornecer um ambiente seguro, o TZ ajuda a prevenir ataques de software malicioso, como malware e rootkits [23].

O ARM TZ é amplamente utilizado em dispositivos móveis, onde desempenha um papel fundamental na segurança de plataformas como Android e iOS [23]. Além disso, ele também é utilizado em outros contextos, como sistemas embebidos, Internet of Things (IoT) e até mesmo em servidores e datacenters, onde pode ser usado para fornecer isolamento seguro entre diferentes serviços e clientes [23].

2.5.4 Discussão

Os dois sistemas ARM TZ e Intel SGX podem ser usados para obtermos um TEE [2, 14, 29].

No que diz respeito aos custos energéticos, o ARM TZ tende a ter uma vantagem relativamente ao Intel SGX. Isso ocorre porque o TZ é uma tecnologia integrada diretamente na arquitetura ARM, o que significa que não requer componentes adicionais de hardware para criar um ambiente seguro. Como resultado, não há necessidade de operar e alimentar componentes extras dedicados exclusivamente ao ambiente seguro. Isso geralmente resulta num consumo de energia mais eficiente em comparação com o SGX.

Por outro lado, o Intel SGX pode apresentar requisitos mais altos de consumo de energia. O Intel SGX exige suporte específico do hardware para criar os enclaves protegidos, o que significa que a CPU precisa fornecer recursos extras para executar o código seguro e proteger os dados dentro desses enclaves. Esse processo pode levar a um aumento do consumo de energia, por requerer recursos adicionais do processador para gerir e proteger o ambiente seguro.

Quanto ao isolamento existem algumas diferenças fundamentais entre as duas tecnologias. O ARM TZ fornece uma abordagem baseada em hardware para criar uma zona segura no sistema, onde o ambiente seguro é isolado do ambiente normal por meio da separação de recursos e controle de acesso. Essa separação é implementada diretamente no processador ARM.

Por um lado, o Intel SGX é uma tecnologia de segurança que permite criar enclaves protegidos,

onde o código pode ser executado de forma segura. O Intel SGX oferece isolamento baseado em software, onde o código é executado num ambiente protegido, mas é confiado ao software a responsabilidade de proteger e gerir esse ambiente.

Em suma, o ARM TZ oferece um nível mais alto de isolamento, já que opera no nível do hardware, garantindo que os recursos e dados sensíveis estejam protegidos. No entanto, o Intel SGX oferece flexibilidade e granularidade adicionais, permitindo que aplicações individuais criem os seus próprios enclaves seguros e protejam partes específicas do código e dos dados.

Ambas as tecnologias têm as suas vantagens e casos de uso específicos. O TZ é amplamente utilizado em dispositivos móveis e sistemas embarcados, onde a segurança e a proteção de informações sensíveis são cruciais. O SGX, por sua vez, é frequentemente usado em cenários no qual a proteção de propriedade intelectual e a execução de código de terceiros num ambiente confiável são importantes, como em computação em nuvem e proteção de dados confidenciais.

Visto que o SGX tem uma barreira de entrada mais baixa, e trabalha sobre uma arquitetura compatível com os sistemas que conseguimos adquirir, optamos por explorar esta tecnologia para desenhar abordagens de deteção de fraude sobre dados protegidos.

2.6 Conclusão

Como vimos nas subsecções anteriores, a deteção de fraude em telecomunicações encontra-se bastante limitada atualmente devido ao RGPD. O uso de criptografia convencional não permite realizar computações sobre dados cifrados. No entanto, as abordagens listadas anteriormente permitem realizar algumas computações com algum *overhead*. O uso de hardware confiável pode ser a chave para tornar as operações mais complexas possíveis no contexto atual.

Capítulo 3

Estado da Arte

Nesta secção iremos abordar algumas estruturas recentes que permitem realizar computações sobre dados cifrados que vão ser usadas como mecanismo essenciais para os protótipos desenhados. No caso iremos falar de como é possível estender o SHADE [5] para mais operações e de uma implementação recente de PSI bastante eficiente. Por fim iremos falar do Gramine como framework para desenvolvimento junto do Intel SGX.

3.1 GSHADE

Bringer et al. [6] definiram uma estrutura mais geral, chamada GSHADE, derivada do SHADE [5] que permite computar mais funções. O GSHADE foi original projetado para computação eficiente e privada de identificação biométrica, íris e face. Também já foi desenhada usando as novas versões de OT-E de modo a atingir o desempenho esperado [6].

O protocolo SHADE original estende-se à família $\mathcal{F}^{GSHADE}$ de funções que podem ser expressas por $f(X,Y) = f_X(X) + \sum_{i=1}^{n} f_i(x_i, Y) + f_Y(Y)$, onde $X = (x_1, \ldots, x_n) \in \{0,1\}^n$ é o input de $\mathcal{C}$ e Y é o input de $\mathcal{S}$. (O conjunto S a que Y pertence não afeta o protocolo). Em particular, algumas das métricas utilizadas para a correspondência biométrica estão incluídas nesta família de funções [6]:

- Distância de Hamming: $X = (x_1, \ldots, x_\ell)$ e $Y = (y_1, \ldots, y_\ell)$ são $n = \ell$-bit vectors. Temos que $f_X = f_Y = 0$ e $f_i(x_i, Y) = x_i \oplus y_i$, para cada $i = 1, \ldots, n$.
- Produto Escalar: $X = (X_1, \ldots, X_K)$ com $X_i = \left(x_{K(i-1)+1}, \ldots, x_{K(i-1)+\ell}\right)$ e $Y = (Y_1, \ldots, Y_K)$ com $Y_i = \left(y_{K(i-1)+1}, \ldots, y_{K(i-1)+\ell}\right)$ são $n = K \times \ell$-bit-integer vectors. Temos que $f_X = f_Y = 0$ e $f_{K\cdot(i-1)+j}\left(x_{K(i-1)+j}, Y\right) = 2^{j-1} \cdot x_{K(i-1)+j} \cdot Y_i$, para cada $i = 1, \ldots, K$ e $j = 1, \ldots, \ell$.
- Raiz Quadrada da Distância Euclidiana: $X = (X_1, \ldots, X_K)$ com $X_i = \left(x_{K(i-1)+1}, \ldots, x_{K(i-1)+\ell}\right)$ e $Y = (Y_1, \ldots, Y_K)$ com $Y_i = \left(y_{K(i-1)+1}, \ldots, y_{K(i-1)+\ell}\right)$ são $n = K \times \ell$ bit-integer

vectors. Temos que $f_X(X) = \Sigma_{i=1}^{K} (X_i)^2$, $f_Y(Y) = \Sigma_{i=1}^{K} (Y_i)^2$ e $f_{K\cdot(i-1)+j}\left(x_{K(i-1)+j}, Y\right) = -2^j \cdot x_{K(i-1)+j} \cdot Y_i$, para cada $i = 1, \ldots, K$ e $j = 1, \ldots, \ell$.

Estas estruturas permitem mapear os valores na estrutura antiga e manter as garantias de segurança contra um adversário semi-honesto [6].

A figura 3.1 lista a estrutura usada para o GSHADE [6]. O m é tal que o output de $f(X, Y)$ pertence a $\mathbb{Z}_m$. Por exemplo, se f for a distância de Hamming entre dois vetores de bits ℓ, definimos $m = \ell + 1$.

Input:

$\mathcal{C}$ tem como input uma n-bit string $X = (x_1, \ldots, x_n)$.

$\mathcal{S}$ tem como input uma $Y \in$ S.

Output:

$\mathcal{S}$ obtém $R \in_R \mathbb{Z}_m$.

$\mathcal{C}$ obtém $T = R + f(X, Y)$.

Protocolo:

1. $\mathcal{S}$ escolhe n valores aleatórios $r_1, \ldots, r_n \in_R \mathbb{Z}_m$.
2. Para cada $i = 1, \ldots, n$, $\mathcal{S}$ e $\mathcal{C}$ envolvem-se num $\text{OT}^{\lceil \log_2(m) \rceil}$ onde:
   - $\mathcal{S}$ atua como o emissor e $\mathcal{C}$ como o recetor.
   - O bit de seleção de $\mathcal{C}$ é x_i.
   - O input de $\mathcal{S}$ é $(r_i + f_i(0, Y), r_i + f_i(1, Y))$.
   - O output obtido por $\mathcal{C}$ é consequentemente $t_i = r_i + f_i(x_i, Y)$.
3. $\mathcal{C}$ calcula e obtém $T = \sum_{i=1}^{n} t_i + f_X(X)$.
4. $\mathcal{S}$ calcula e obtém $R = \sum_{i=1}^{n} r_i - f_Y(Y)$.

Figura 3.1: Descrição do protocolo GHADE (adaptado de Bringer et al. [6]).

Estes modelos tanto do SHADE quando o GSHADE podem ser estendidos para casos de 1-N de forma eficiente. No caso seguinte, o cliente tem um input X e onde o servidor tem N input's $Y^1, \ldots, Y^N$ e eles computam todas as distâncias $f\left(X, Y^b\right)$, para $b = 1, \ldots, N$ [5, 6]. Na figura 3.2 está descrito o protocolo modificado para este caso.

3.2 Vole-PSI

Vole-PSI [43] implementa os protocolos descritos por Rindal and Schoppmann [39] e Raghuraman and Rindal [38]. A biblioteca implementa PSI e uma variante chamada Circuit-PSI, em que o resultado é um segredo partilhado entre as duas partes [38, 39, 43].

Rindal and Schoppmann [39] implementaram uma construção para OPRF que é baseada em Vector Oblivious Linear Evaluation (VOLE) e a estrutura Probe-and-XOR of Strings (PaXoS) [37].

1. $\mathcal{S}$ escolhe $n \cdot N$ valores aleatórios $r_{b,i} \in_R \mathbb{Z}_m$, para $b = 1, \ldots, N$.
2. Para cada $i = 1, \ldots, n, \mathcal{S}$ e $\mathcal{C}$ envolvem-se num $\mathrm{OT}^{N\lceil \log_2(m) \rceil}$ onde:
   $\mathcal{S}$ atua como o emissor e $\mathcal{C}$ como o recetor.
   O bit de seleção de $\mathcal{C}$ é x_i.
   O input de $\mathcal{S}$ é $\left(r_{1,i} + f_i\left(0, Y^1\right) \| \ldots \| r_{N,i} + f_i\left(0, Y^N\right),\right.$
$\left. r_{1,i} + f_i\left(1, Y^1\right) \| \ldots \| r_{N,i} + f_i\left(1, Y^N\right)\right)$.
   O output obtido por $\mathcal{C}$ é $(t_{1,i} \| \ldots \| t_{N,i}) \quad =$
$\left(r_{1,i} + f_i\left(x_i, Y^1\right) \| \ldots \| r_{N,i} + f_i\left(x_i, Y^N\right)\right)$.
3. $\mathcal{C}$ calcula e obtém $T^1 = \Sigma_{i=1}^{\ell} t_{1,i} + f_X(X), \ldots, T^N = \Sigma_{i=1}^{\ell} t_{N,i} + f_X(X)$.
4. $\mathcal{S}$ calcula e obtém $R^1 = \sum_{i=1}^{n} r_{1,i} - f_Y\left(Y^1\right), \ldots, R^N = \sum_{i=1}^{n} r_{N,i} -$
$f_Y\left(Y^N\right)$.

Figura 3.2: Extensão do GSHADE para casos 1-N (adaptado de Bringer et al. [6]).

O protocolo é altamente eficiente, com baixos custos de comunicação e de computação [39]. Ele é resistente a um adversário semi-honesto e malicioso [39]. Na mesma publicação, os autores apresentam uma extensão chamada Circuit-PSI, normalmente usada junto de SMC [39].

Raghuraman and Rindal [38] implementaram um PSI baseado em VOLE e uma estrutura melhorada de Oblivious Key-Value Stores (OKVS). Esta implementação contempla cenários em que temos um adversário semi-honesto e malicioso. O protocolo semi-honesto pode ser executado em 0,37 segundos [38]. Recorrendo ao uso de threads ainda é possível ir mais longe, conseguindo reduzir até 0,16 segundos usando 4 threads [38]. Estas melhorias devem-se à redução de consumo de rede devido ao uso do VOLE. A estrutura OKVS, especifica para a implementação de PSI que permite melhorar ainda mais tanto o custo computacional como o custo de comunicação [38].

3.3 Gramine

O Gramine [11, 42] anteriormente conhecido como Graphene é uma framework que permite executar programas genéricos em enclaves recorrendo a um Platform Adaptation Layer (PAL) para traduzir as suas chamadas ao sistema [11, 42]. A grande vantagem para o programador é que remove grande parte das dificuldades e do trabalho necessário para trabalhar usando o Software Development Kit (SDK) do Intel SGX. No entanto, adiciona algum *overhead* ao tempo de execução [11, 42]. A framework está implementada na forma de uma biblioteca dinâmica que age como *Library OS* [11, 42].

Ela traz consigo algumas funcionalidades. Por exemplo, temos um canal Transport Layer Security (TLS) com atestação pela Intel já implementado para os dois tipos de atestação existentes, EPID e DCAP [11, 42]. Este canal é designado por Remote Attestation with Transport Layer Security (RA-TLS) [28]. O funcionamento do canal é descrito nas imagens seguintes que estão

disponíveis na documentação do Gramine [13]. Na figura 3.3 descrevemos a estrutura de certificado do RA-TLS usando DCAP, mas para EPID é muito idêntico. Na segunda figura 3.4 está descrita uma atestação realizada pelo canal RA-TLS usando os serviços da Intel com a atestação EPID.

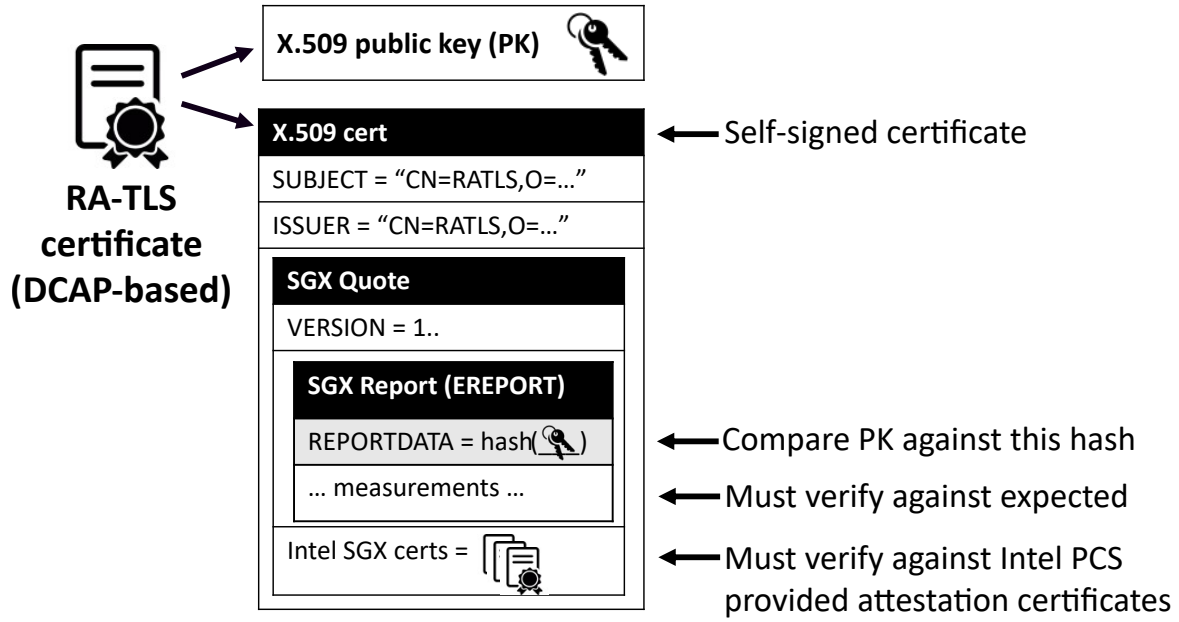


Figura 3.3: Esquema do certificado do RA-TLS para DCAP, mas para EPID é semelhante (adaptado de [13]).

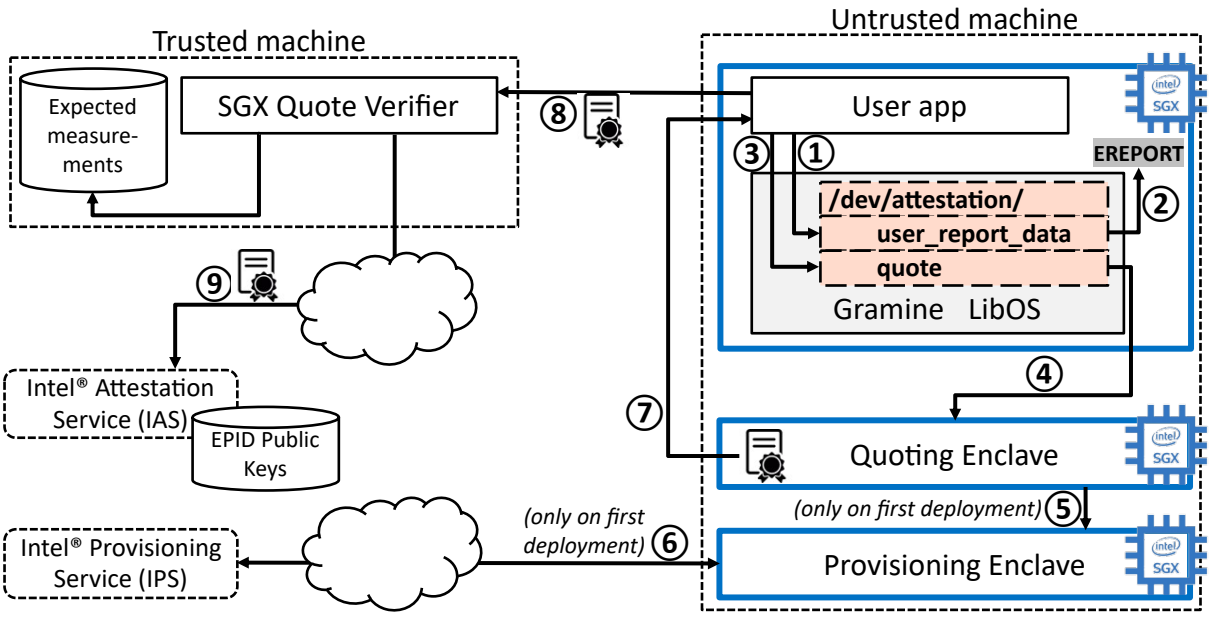


Figura 3.4: Esquema da atestação do canal RA-TLS usando EPID (adaptado de [13]).

3.4 Conclusão

O GSHADE é importante, por demonstrar a existência de uma variedade de funções que podem ser construidas a custa da estrutura SHADE. O VolePSI é bastante eficiente, o que nos irá ajudar devido ao custo associado a estes algoritmos. A combinação de ambos os algoritmos poderá ser a chave para conseguirmos algoritmos para uso em tempo real mesmo usando software.

Por fim, o Gramine irá permitir nos focar no desenvolvimento de uma solução e deixar de lado uma parte das dificuldades que são programar aplicações de hardware confiável. Assim podemos nos dedicar a melhorar a eficiência da nossa implementação.

Capítulo 4

Computação Segura baseada em Software

Neste capítulo vamos descrever um protótipo baseado em software que planeia responder ao modelo em que duas entidades não confiam uma na outra, mas pretende ter algum tipo de computação conjunta. No nosso caso as computações serão a interseção e o cálculo da distância de Hamming. Um dos problemas descrito é a dificuldade em obter a similaridade entre números, então também exploramos formas de obter distância euclidiana a custa da distância de Hamming. Modelos baseados em software têm custos elevados de computação e de rede.

4.1 Modelo e requisitos funcionais

O modelo problema é constituído por duas entidades, a entidade A e a entidade B. As entidades A e B não podem revelar os seus dados, mas pretendem computar uma função $F(x)$ entre os seus dados. Este modelo problema está representado na figura 4.1.

A seguir, mapeamos o modelo no contexto de deteção de fraude em que temos datasets de números em entidades distintas, mas queremos computar funções como Interseção e DH. Seria útil também conseguir computar a Distância Euclidiana (DE).

4.2 Protocolo

O protocolo de atuação entre as duas entidades poderá ser simplesmente a implementação do PSI e SHD. Para este modelo em concreto definimos como o nosso adversário um adversário semi-honesto. Por outras palavras que ele só poderá recuperar a informação que o protocolo fornecer. Como a implementação do PSI e SHD é garantido que apenas o resultado da operação é retornado e nada mais, a segurança contra este adversário fica garantida.

O modelo de software que cogitamos implementar está descrito na figura 4.2. Neste modelo,

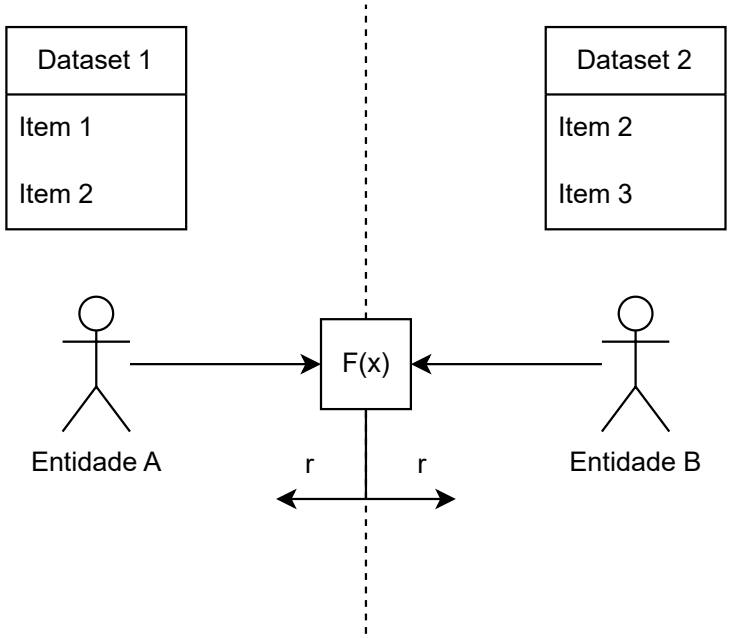


Figura 4.1: Modelo Problema A – Duas entidades não confiam uma na outra, mas pretendem computar uma função $F(x)$.

temos por base a Mobileum e uma entidade externa. Cada um possui a sua base de dados de números fraudulentos. O software a desenvolver terá que ter uma implementação de PSI e outra de SHD. Com a estrutura do SHD ainda iremos explorar abordagens para DE.

4.3 Implementação

Nesta secção iremos descrever como implementamos as funções do protótipo de software.

Para o cálculo do Set Intersection optamos por usar a implementação do VolePSI [38, 39] disponível no Github [43]. Para o cálculo da DH optamos por implementar a estrutura do SHADE [5]. A estrutura do SHADE [5] é bastante útil, por permitir implementar uma série de algoritmos descritos no GSHADE [6]. Para combinarmos os protocolos tivemos que desenvolver a nossa versão do SHD, adaptada a caso dos números de telemóvel. Um dos desafios é conseguirmos detetar similaridade entre números. Para tal precisamos de preservar algumas propriedades numéricas.

Outro desafio é conseguirmos realizar as operações em tempo real, ou no pior dos casos, no menor tempo possível, respeitando os requisitos de privacidade. Tendo em vista o menor tempo possível, usamos otimizações conhecidas de OT e procuramos por implementações de OT-E. Acabamos por encontrar o repositório [34] que implementa algumas variantes de OT-E, contendo também o IKNP, um tipo de OT-E descrito em 2.4.2. Temos uma certa vantagem, pois este repositório é umas das dependências do VolePSI [38, 39]. Assim sendo, ele já estava contido no ponto de partida para o desenvolvimento do SHADE [5].

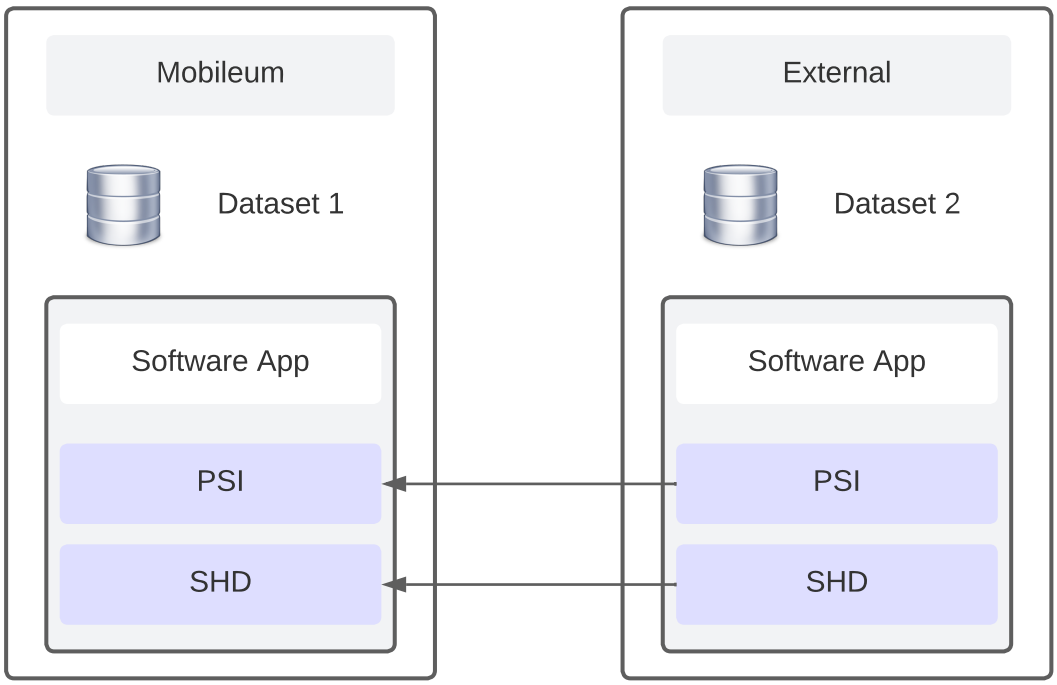


Figura 4.2: Modelo de Software – O protótipo que contem uma implementação de PSI e outra de SHD, que posteriormente poderá ser usado também para o cálculo da DE.

Para o caso dos números de telemóvel optamos por dividir o protocolo em 3 partes:

- Prefixo do Número usando um mecanismo de comparação seguro, usaremos o VolePSI para comparar as partes iniciais dos números e só se forem iguais é que prosseguimos.
- Conversão dos dígitos para a representação de Binário, Inteiro e Buffer. Permite ajustar ao caso pretendido de precisão e tempo esperado. Cada um tem as suas vantagens e desvantagens.
- Secure Hamming Distance Padrão e heurísticas de Máximo e Mínimo para DE. A nossa implementação de SHD modificada para o caso de números de telemóvel.

4.3.1 Prefixo do Número

O padrão é convertermos os dígitos em binário. Com isto conseguimos realizar o cálculo da DH mapeando a representação em binário de cada dígito em 4 OT's, uma para cada bit do dígito.

Após algumas observações identificamos que o custo era demasiado elevado para o uso em todos os dígitos de um número. Para cada dígito teremos que usar 4 OT's. Para um número com 12 dígitos teríamos $12 * 4 = 48$ OT's. Então, para mitigar o problema, decidimos limitar o

cálculo aos últimos 3, 2 e 1 dígitos dos números de telemóvel. Ou seja, passamos a usar 12, 8 e 4 OT's respetivamente por cada número.

O restante do número, a parte inicial dele, teremos que aplicar um algoritmo seguro de comparação como o PSI. Com isto passamos a aplicar apenas o SHD aos últimos dígitos, pois o restante já foi confirmado serem iguais. Um exemplo, se estivemos a aplicar o algoritmo a 3 dígitos. O número seria comparado com a representação com os últimos dígitos substituídos por um X, +351999999XXX, ou mesmo suprimidos que seria mais eficiente.

Segue um exemplo, queremos cruzar os números +351999999001 e +351999999002. Usando o esquema anterior, os números seriam convertidos para +351999999XXX e +351999999XXX. Como são iguais, então seguimos para o passo seguinte.

4.3.2 Codificação dos dígitos em String Binária

A seguir, definimos como seriam representados os dígitos no formato binário, pois o SHD está desenhado para sequências de "0" e "1". Esta representação está descrita na tabela 4.1.

A conversão mais natural é mapear dígitos em binário. Esta é a conversão mais eficiente. Mas, por outro lado, não permite preservar a DE. Nesta conversão são usados 4 bits, apenas, para representar um dígito.

A segunda conversão é mais extensa, menos eficiente, mas permite preservar a DE. Nesta conversão são usados 9 bits para representar um dígito.

Na tabela 4.1 é descrito a conversão de dígitos. Dando seguimento ao exemplo anterior, com os números +351999999001 e +351999999002. Usando os últimos 3 dígitos e a representação em Binário, os números seriam mapeados para "000000000001" e "000000000010".

Dígito	Representação de Binário	Representação de Inteiro
0	"0000"	"000000000"
1	"0001"	"000000001"
2	"0010"	"000000011"
3	"0011"	"000000111"
4	"0100"	"000001111"
5	"0101"	"000011111"
6	"0110"	"000111111"
7	"0111"	"001111111"
8	"1000"	"011111111"
9	"1001"	"111111111"

Tabela 4.1: Descrição das conversões usadas para cada dígito, que permite preservar ou não a DE mesmo usando uma estrutura para o cálculo da DH.

Também foi definida outra conversão que seria quase "fill". Esta conversão é mais usada para o caso em vários dígitos por permite preservar a DE usando a estrutura existente. Esta conversão foi batizada de "Buffer". Veremos em mais pormenor na secção seguinte.

4.3.3 Distância de Hamming Segura

A estrutura SHADE [5] padrão permite aplicar diretamente as conversões da secção anterior para obter a DH. O verdadeiro desafio foi tentar chegar a uma aproximação da DH à DE. Desta forma seria possível dizer que um número está a x de distância. Isto no intervalo de dígitos de 3, 2 e 1 dígitos. Na figura 4.3 tem um exemplo da aplicação deste esquema.

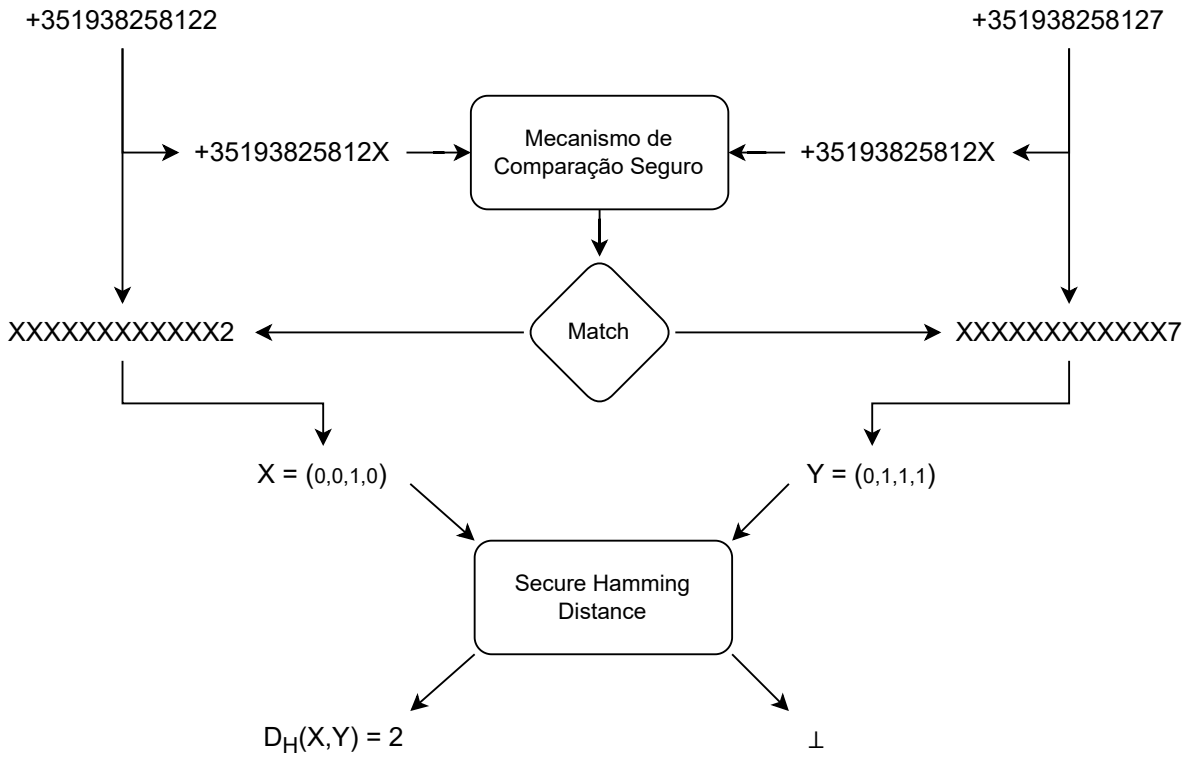


Figura 4.3: Exemplo do uso do SHD padrão para medir a distância no caso de obter match no mecanismo de comparação para um dígito apenas.

Fazendo representar os dígitos por uma sequência de "1" e "0" conseguimos que DH seja igual a DE. O número de "1" iguala o número do dígito. E o número de "0" é o restante até perfazer 9 caracteres. Esta estrutura está descrita na 4.1. Por consequência, cada dígito passa a ser representado por 9 OT's. Na figura 4.4 tem um exemplo da aplicação deste esquema.

Mas esta estrutura tem limitações. Esta propriedade está correta para quando é usado apenas o último dígito. Já não devolve o resultado correto quando temos vários dígitos. Seguem alguns exemplos destas situações:

- $HD_i(5,9) = 4$, $ED(5,9) = 4$

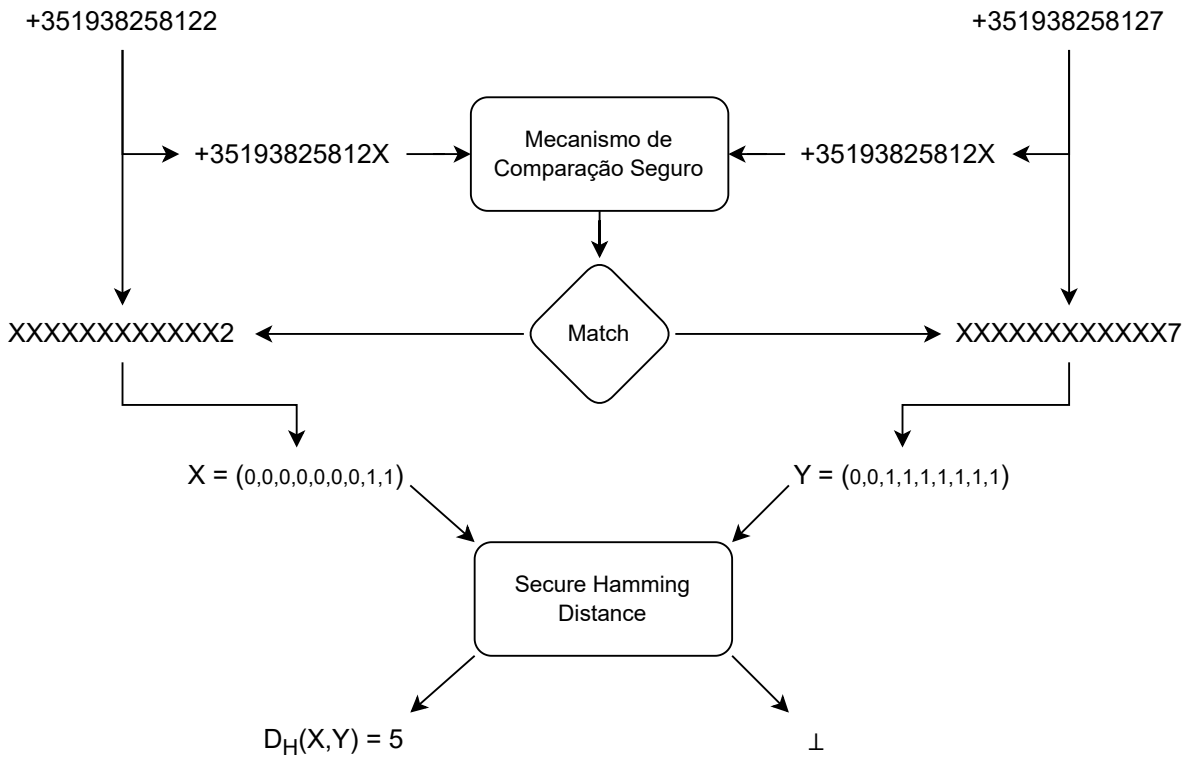


Figura 4.4: Exemplo do uso do SHD modificado para medir a distância no caso de obter match no mecanismo de comparação para um dígito apenas e preservando a DE.

- $HD_i(19, 20) = 10$, $ED(19, 20) = 1$

Para colmatar esta limitação estendemos a representação dos bits para os vários dígitos, o que chamamos previamente de "Buffer". Por outras palavras, o 19 será representado por 80 "0" e 19 "1". O que perfaz um uso de 99 OT's por número. Já no caso de 3 dígitos teremos que usar 999 OT's por número. Embora o custo da operação tenho subido bastante, o cálculo agora está correto. Temos um exemplo de dois dígitos descrito na figura 4.5.

Seguindo o exemplo da subsecção anterior, com os números +351999999001 e +351999999002. Usando os últimos 3 dígitos e a representação em Binário, os números seriam mapeados para "000000000001" e "000000000010". Usando o SHD padrão obtemos que a $D_H = 2$.

4.3.4 Heurísticas para o Máximo e Mínimo

Posteriormente investigamos algumas heurísticas. Será possível não usar as 999 OT's e mesmo assim conseguir ter uma heurística que afirme temos no mínimo x de distância ou até temos no máximo y de distância. Após explorarmos as construções dos algoritmos percebemos que adicionando pesos podemos construir heurísticas que são admissíveis. Esta abordagem foi inspirada nas construções usadas no GSHADE [6].

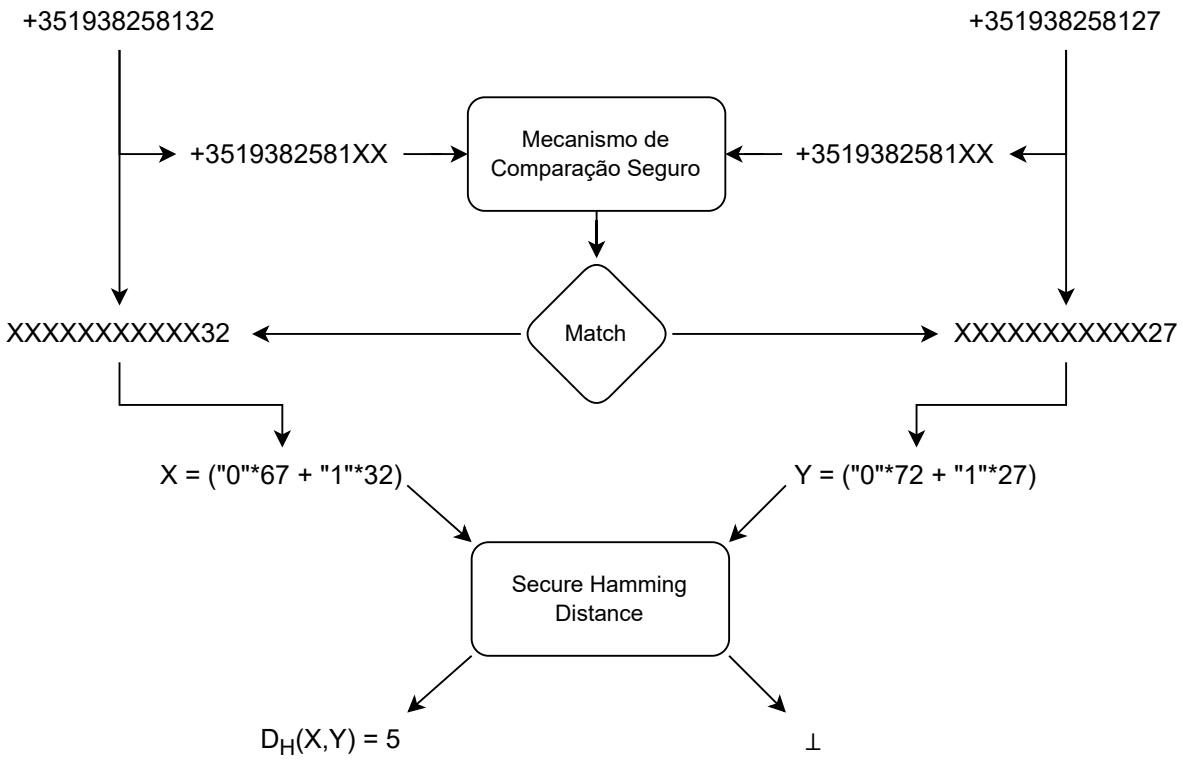


Figura 4.5: Exemplo do uso do SHD modificado para medir a distância no caso de obter match no mecanismo de comparação para dois dígitos e preservando a DE mesmo usando vários dígitos.

Para estas construções voltamos a usar o modelo de representar os dígitos um a um usando as 9 OT's. O resultado do cálculo da heurística para o máximo e mínimo usando apenas 1 dígito é exatamente a DE.

As diferenças surgem aquando do cálculo do R e do T no passo 1 e 3 do algoritmo base de SHADE [5]. São realizadas somas parciais para cada 9 OT's. Ou seja, passamos a ter em R_1 o somatório dos últimos 9 r. Em R_2 o somatório dos 9 r anteriores do R_1. Em R_3 o somatório dos primeiros 9 r. Consoante o cálculo do máximo e mínimo irá variar em como a conta será feita.

Para o cálculo do mínimo:

- No caso de estarmos a usar 1 dígito a heurística iguala o caso inicial.
- No caso de estarmos a usar 2 dígitos usamos o $R = 10 * R_2 - R_1$ e o $T = 10 * T_2 - T_1$.
- No caso de estarmos a usar 3 dígitos usamos o $R = 100 * R_3 - 10 * R_2 - R_1$ e o $T = 100 * T_3 - 10 * T_2 - T_1$.

Para o cálculo do máximo:

- No caso de estarmos a usar 1 dígito a heurística iguala o caso inicial.
- No caso de estarmos a usar 2 dígitos usamos o $R = 10 * R_2 + R_1$ e o $T = 10 * T_2 + T_1$.

- No caso de estarmos a usar 3 dígitos usamos o $R = 100 * R_3 + 10 * R_2 + R_1$ e o $T = 100 * T_3 + 10 * T_2 + T_1$.

Para terminar, mas por que motivo não é possível usando esta construção obter a DE = DH usando a conversão de dígito para inteiro. Isso iria exigir fazer comparações entre cada dígito para saber qual era o menor. Só assim saberemos qual dígito somar ou subtrair. Na prática, no caso de 3 dígitos podemos ter 4 possibilidades e só apenas uma delas é valor da DE. Por exemplo, façamos a operação de $ED(136, 217) = 81$ usando estas operações por dígito. $ED(1, 2) = 1, ED(3, 1) = 2, ED(6, 7) = 1$, o problema surge agora, o valor poderá ser: $100*1+10*2+1 = 121$, $100*1+10*2-1 = 119$, $100*1-10*2+1 = 81$, $100*1-10*2-1 = 79$. Como acabamos de constatar no caso era a opção número 3. Sem a capacidade de comparar de forma segura, não podemos fazer esta operação de forma exata. Mas a estrutura ainda nos permite calcular as heurísticas anteriormente definidas.

4.3.5 Otimizações OT-E

Segundo Ishai et al. [26], o IKNP consegue refazer as mesmas operações que o OT padrão usando apenas $O(\log n)$. Isso reduzia consideravelmente os tempos usados. Mesmo após implementarmos o SHD usando IKNP o crescimento era exponencial. A limitação encontrada é que estávamos a iniciar uma nova OT-E a cada número. Na prática, estávamos a fazer $O(n^2)$ em vez de $O(n \log n)$. Para resolvermos o problema alteramos a estrutura para codificar todas as conversões para bits numa única estrutura. Desta forma, usamos uma única OT-E para todas as computações a realizar. Com esta alteração o tempo já tendia para $O(n \log n)$.

4.4 Avaliação experimental

Agora voltamos a nossa atenção para o desempenho concreto das nossas construções. Os protocolos foram todos avaliados num único portátil com um processador Intel i7 8750H e 16GB de RAM. No portátil está instalado o Pop!_OS 22.04 com o Kernel Linux 6.4. Todos os protocolos foram executados em single thread. Os participantes, cliente e servidor estão na mesma máquina, e como tal descartamos potenciais atrasos derivados de latência de rede.

Os dados usados para teste foram gerados de forma aleatória. Os números têm o formato +XXXXXXXXXXXX. Os casos de teste gerados com quantidades em bases de 2. No caso entre 2^{12} números até 2^{24} números.

Conforme as discussões com os colegas da Mobileum, foi aferido que o tempo esperado para uma query deste tipo em tempo real será por volta de 200 ms. Os tempos obtidos a negrito são os tempos em cumprem a deadline para uso em tempo real. Os tempos restantes são interessantes noutros contextos como deteção de fraude pós-incidente, em que os tempos podem ir até uma semana.

Para cada teste foram executadas três iterações. Após analisar os dados observamos que a variabilidade, no pior caso, era menor que 2%. Este valor é relativamente baixo, o que indica que os resultados são consistentes e não variam significativamente em relação à média. Devido a esta baixa variabilidade, concluímos que não era necessário executar mais do que as três iterações, uma vez que obteríamos resultados semelhantes em repetições adicionais.

Começamos os testes pelo PSI incluso dentro do protótipo. Apenas preenchemos uma parte da matriz, pois terão dados semelhantes. Ou seja, cruzar 2^{12} com 2^{24} é o mesmo tempo que 2^{24} e 2^{12}. Vamos então analisar os resultados obtidos.

PSI (ms)		Quantidade de Números dentro do Servidor												
		2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
Quantidade de Números dentro do Cliente	2^{12}	**57**	**52**	**62**	**78**	**73**	**126**	236	394	806	1543	3187	6656	13620
	2^{13}		**48**	**59**	**58**	**73**	**111**	234	403	769	1565	3225	6554	13385
	2^{14}			**58**	**67**	**81**	**111**	209	406	776	1562	3207	6625	13496
	2^{15}				**71**	**83**	**118**	231	403	798	1566	3253	6661	13528
	2^{16}					**83**	**117**	207	412	796	1570	3191	6639	13648
	2^{17}						**127**	226	421	811	1573	3246	6699	13298
	2^{18}							243	477	832	1638	3240	6708	13603
	2^{19}								509	895	1665	3263	6749	14401
	2^{20}									1013	1710	3356	6842	13646
	2^{21}										1919	3556	6947	13745
	2^{22}											3990	7296	14140
	2^{23}												8103	14909
	2^{24}													17216

Tabela 4.2: Tabela de dados de testes recolhidos na execução do PSI. Estes dados são o resultado do cruzamento de todos os números com todos os números. Os valores a negrito são os elementos que podem ser executados em tempo real.

Após analisar a tabela 4.2, constatámos que conseguimos responder com até 2^{17} números armazenados no servidor em tempo real. Mesmo com os 2^{24} números armazenados conseguimos obter uma interseção em 18 segundos. Isto é bastante pertinente, pois auxilia posteriormente a nossa implementação em conjunto com o SHD. No modelo usado, isto significa que se a Mobileum tiver armazenados na sua base de dados até 2^{17} números fraudulentos e uma entidade externa quiser fazer uma interseção, mesmo usando outros 2^{17}, irá obter a resposta em menos de 200ms. O desvio padrão da tabela durante a execução dos testes foi, no pior caso, de 0,9%.

Na tabela 4.3 temos descritos os resultados da nossa implementação base de SHD usando a conversão em Binário. Este modo para 3, 2 e 1 dígitos usa, respetivamente, 12, 8 e 4 OT's por número. Como podemos observar, conseguimos usando 1 dígito cruzar um número com até 2^{14} números em tempo real. Conseguimos usando 2 dígitos cruzar um número com até 2^{13} números em tempo real. Conseguimos usando 3 dígitos cruzar um número com até 2^{12} números em tempo real. O mais interessante é que mesmo no pior caso em que temos que cruzar 1 número com até

SHD_{bin} (ms)		Quantidade de Números Cruzados										
		2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}
Dígitos	1	**66**	**99**	**153**	265	492	909	1701	3183	6297	12388	24376
	2	**106**	**163**	264	516	889	1722	3188	6308	12554	24989	48516
	3	**130**	212	379	704	1302	2447	4708	9391	18260	37830	73854

Tabela 4.3: Tabela de dados de testes recolhidos na execução de SHD em modo Binário. É de referir que o tempo usado corresponde a aplicação de SHD entre 1 número e os definidos na linha superior. Os valores a negrito são os elementos que podem ser executados em tempo real.

2^{22} números usando 3 dígitos iríamos usar 1 minuto e 13 segundos. O desvio padrão durante a execução dos testes foi, no pior caso, de 1,7%.

SHD_{int} (ms)		Quantidade de Números Cruzados										
		2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}
Dígitos	1	**98**	**166**	297	558	1016	1854	3550	6971	14267	27937	54433
	2	**178**	304	551	995	1840	3628	7145	13981	27835	55969	110957
	3	229	425	765	1445	2703	5243	10371	21109	40942	82137	165946

Tabela 4.4: Tabela de dados de testes recolhidos na execução de SHD em modo Inteiro. É de referir o tempo usado corresponde a aplicação de SHD entre 1 número e os definidos na linha superior. Os valores a negrito são os elementos que podem ser executados em tempo real.

Na tabela 4.4 temos descritos os resultados da nossa implementação base de SHD usando a conversão em Inteiro. Este modo para 3, 2 e 1 dígitos usa, respetivamente, 27, 18 e 9 OT's por número. Como podemos observar, conseguimos usando 1 dígito cruzar um número com até 2^{13} números em tempo real. Conseguimos usando 2 dígitos cruzar um número com até 2^{12} números em tempo real. Com 3 dígitos já não possível em tempo real usando este algoritmo. O mais interessante é que mesmo no pior caso em que temos que cruzar 1 número com até 2^{22} números usando 3 dígitos iríamos precisar de 2 minutos e 46 segundos. O desvio padrão durante a execução dos testes foi, no pior caso, de 1,6%.

SHD_{buf} (ms)		Quantidade de Números Cruzados										
		2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}
Dígitos	1	**109**	**186**	340	555	993	1872	3622	7076	14053	28049	55584
	2	661	1266	2448	4912	9521	19449	37398	74461	152559		
	3	6082	12079	24546	49007	95551	222861					

Tabela 4.5: Tabela de dados de testes recolhidos na execução de SHD em modo Buffer. É de referir o tempo usado corresponde a aplicação de SHD entre 1 número e os definidos na linha superior. Os valores a negrito são os elementos que podem ser executados em tempo real.

Na tabela 4.5 temos descritos os resultados da nossa implementação base de SHD usando o modo Buffer. Este modo para 3, 2 e 1 dígitos usa, respetivamente, 999, 99 e 9 OT's por número.

Este modo em concreto usa muitos recursos computacionais. Os espaços em branco representam situações em que foi atingido o limite de memória na máquina em que usamos para correr os testes. Como podemos observar, os consumos com 1 dígito são os mesmos da tabela 4.4. Este modo não serve para uso em tempo real, mas permite tirar conclusões úteis na deteção de casos de sequenciação, pois o resultado obtido é sempre o correto. Devido à limitação de memória, os resultados obtidos foram, no pior caso, em que temos que cruzar 1 número com até 2^{17} números usando 3 dígitos iríamos precisar de 3 minutos e 43 segundos. No caso de usarmos 2 dígitos podemos ir até 2^{20} e precisamos de 2 minutos e 33 segundos. O desvio padrão durante a execução dos testes foi, no pior caso, de 1,3%.

4.4.1 Heurísticas para Distância Euclidiana

Nesta subsecção vamos falar em concreto das heurísticas para o máximo e mínimo da DE. É particularmente útil, pois não precisaremos de usar tantas OT's e nem arriscamos ficar sem memória.

SHD_{min} (ms)		Quantidade de Números Cruzados										
		2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}
Dígitos	1	**102**	**158**	275	481	939	1800	3475	6839	14202	27757	55585
	2	**155**	265	490	921	1796	3527	6887	13623	27666	54284	112538
	3	251	377	679	1363	2660	5311	10399	20877	42126	82756	176921

Tabela 4.6: Tabela de dados de testes recolhidos na execução da heurística do mínimo da DE. É de referir o tempo usado corresponde a aplicação da heurística entre 1 número e os definidos na linha superior. Os valores a negrito são os elementos que podem ser executados em tempo real.

SHD_{max} (ms)		Quantidade de Números Cruzados										
		2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}
Dígitos	1	**108**	**175**	291	558	1014	1882	3674	7219	13826	28304	54361
	2	**170**	308	545	979	1841	3661	7115	13868	27289	55386	112075
	3	223	380	703	1357	2681	5256	10270	20370	42100	81305	183335

Tabela 4.7: Tabela de dados de testes recolhidos na execução da heurística do máximo da DE. É de referir o tempo usado corresponde a aplicação da heurística entre 1 número e os definidos na linha superior. Os valores a negrito são os elementos que podem ser executados em tempo real.

Iremos analisar as duas tabelas em conjunto, pois as construções são muito semelhantes. Na tabela 4.6 o desvio padrão foi, no pior caso, de 1,6% e na tabela 4.7 foi de 1,2%. Na tabela 4.6 temos descrito os resultados da nossa implementação com o SHD modificada para o cálculo desta heurística do mínimo da DE. Já na tabela 4.7 temos descrito os resultados da nossa implementação para o cálculo do máximo da DE. Os resultados são muito similares, o custo em OT's é o mesmo do da tabela 4.4. Tem um ligeiro *overhead* recorrente da modificação no cálculo. Os tempos em tempo real continuam os mesmos da tabela 4.4. Na implementação do mínimo,

no pior caso, em que temos que cruzar 1 número com até 2^{22} números usando 3 dígitos, iríamos precisar de 2 minutos e 56 segundos. No caso, foram usados mais 10s em comparação com os valores da tabela 4.4. Na implementação do máximo, no pior caso, em que temos que cruzar 1 número com até 2^{22} números usando 3 dígitos, iríamos precisar de 3 minutos e 3 segundos. No caso, foram apenas mais 17s em relação aos da tabela 4.4. Notamos que existem muitas otimizações que podem ser feitas, ao nível da paralelização e de escolha de parâmetros.

4.5 Sumário

O protótipo resultante combina a capacidade de realizar PSI e SHD. Dentro do SHD adaptado conseguimos calcula a Hamming padrão, a diferença DE dígito a dígito usando a versão "Inteiro" e ainda a DE em todos os dígitos usando a versão "Buffer". E ainda, as heurísticas para a aproximação do máximo e do mínimo na diferença de DE entre os dois números. Resumindo, tem todas as peças necessárias para fazer testes de interseção e similaridade entre números, respeitando os requisitos de privacidade.

Após a avaliação experimental verificamos que foram poucas as situações em que podemos usar o protótipo para deteção de fraude em tempo real. No entanto, é perfeitamente funcional para uso *offline*. Em qualquer um dos casos, este tipo de implementações mostrou-se difícil de escalar para a quantidade de dados processados pelo Mobileum diariamente.

Capítulo 5

Computação Segura baseada em Hardware

Neste capítulo vamos descrever a implementação de um protótipo baseado em hardware confiável que cogita responder ao modelo em que se pretende fazer outsourcing de computação. Por outras palavras, planeamos colocar os nossos dados em outro local, mas ter garantias que eles estão protegidos. A privacidade dos dados é garantida, mesmo que o sistema que nos armazena os dados seja comprometido. Uma das propostas da Mobileum é um sistema semelhante. No âmbito do projeto AIDA, a Mobileum visa trabalho colaborativo para deteção de fraude com a CMU. Neste contexto, os dados ficariam armazenados na CMU. No entanto, a Mobileum necessita de manter as garantias de segurança anteriormente descritas. Modelos baseados em hardware tem baixos custos de computação e de rede, mas exigem confiança no manufator.

5.1 Modelo e requisitos funcionais

Concretamente, o nosso objetivo é que o ambiente que manipulará os nossos dados se comporte como uma caixa negra. Os dados são armazenados e computados corretamente, porém estes estão isolados em todos os momentos contra observação e/ou manipulação externa – potencialmente maliciosa. Este modelo problema está descrito na figura 5.1. Nesta figura temos os "Request Users" que aqui no caso é apenas a CMU. A "Cloud" será onde os dados serão armazenados. No caso, a Mobileum tem os dados e a CMU quer fazer processamento sobre os mesmos, mas não pode ter acesso a eles.

5.2 Protocolo

Para este modelo em concreto usamos o Intel SGX como hardware confiável. Ele permite construir um sistema em cima da definição de TEE. No caso do Intel SGX a segurança dos dados é baseada na definição de uma Application Programming Interface (API). Ou seja, ele só revela os dados

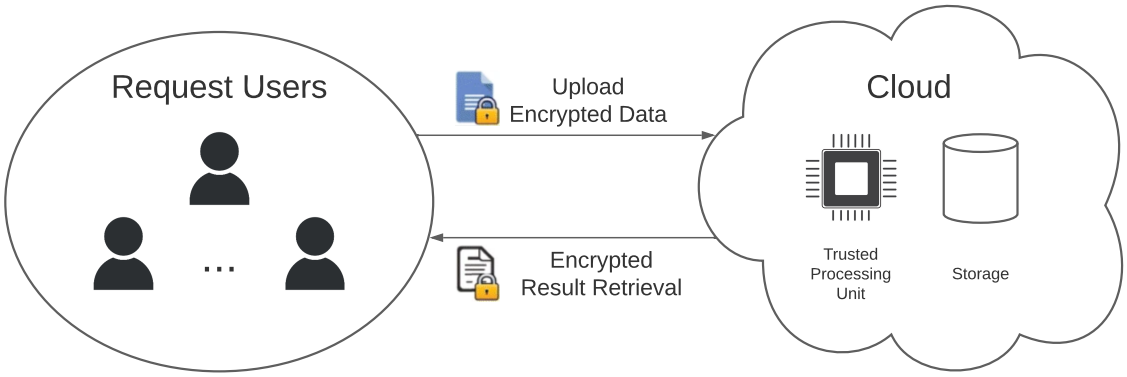


Figura 5.1: Modelo problema B – Outsourcing de computação.

para aquilo que foi desenhado. Já que o software estará a executar em ambiente privilegiado é exigido do programador muito mais cuidado no desenho do protocolo. Na figura 5.2 é descrito duas zonas. Uma que está sobre o domínio da Mobileum em que terá uma aplicação que recebe querys vindas da CMU, avalia-se se são seguras e se sim então assina e devolve a assinatura a CMU. No domínio da CMU temos um enclave criado e assinado pela Mobileum. Além disso, temos uma aplicação da CMU para enviar querys a Mobileum e enviar as querys assinadas para o enclave. A Mobileum remotamente transfere os números para o enclave através da sua aplicação. Por fim, a aplicação da CMU envia a query assinada, o enclave verifica a query e se tudo estiver correto devolve o resultado da operação.

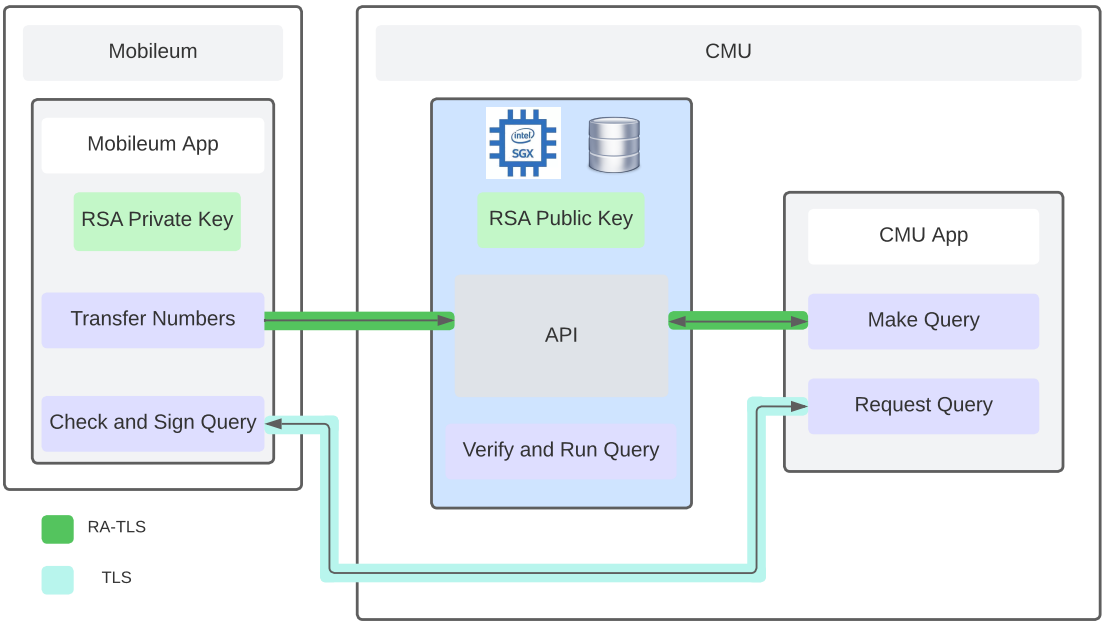


Figura 5.2: Descrição do sistema de interação das partes entre a Mobileum e CMU.

Este modelo é muito mais dinâmico, ao permitir definir quantas operações quisermos, pois não estamos limitados a que algoritmos usamos, mas sim que informação é revelada. As operações que implementamos estão descritas na figura 5.3. Cada uma das cinco operações possui dois tipos de resultado, a contagem ou uma lista de identificadores.

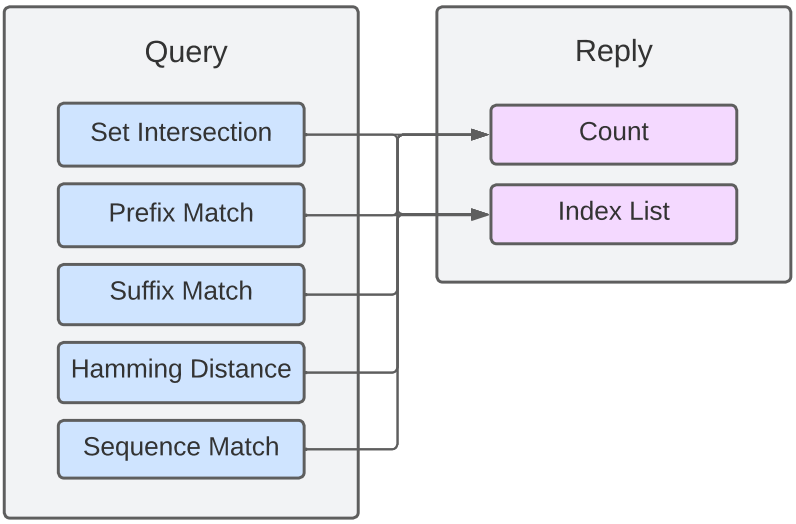


Figura 5.3: Descrição das operações disponíveis no protótipo.

Relativamente à privacidade e garantias de segurança, todas as querys para o enclave responder necessitam de estar assinadas com a chave privada da Mobileum. Todo o controlo do que é executado será sempre mediado pela empresa. Se a query for demasiado reveladora ou der mais informação do que é suposto, cabe a Mobileum recusar a sua assinatura. A grande vantagem deste modelo é que permite adicionar novas operações ao enclave. Mesmo que posteriormente alguma operação se torne insegura, basta a Mobileum recusar assinar mensagens para aquela operação no futuro. As mensagens são assinadas com um número de sequência para impedir ataques de repetição. Desta forma a privacidade fica sempre assegurada.

5.3 Implementação

Para a implementação inicial iríamos usar apenas o SDK do Intel SGX. Posterior descobrimos a existência de um canal RA-TLS que já permitia fazer atestação remota sobre um canal TLS. Para tal tivemos que compilar o Gramine, anteriormente conhecido como Graphene que é uma abstração de um libOS. A grande vantagem é que retira grande parte da complexidade em comparação com uma implementação no SDK. Desta forma partimos do exemplo que o Gramine disponibiliza que inicia um canal TLS, faz a atestação do canal usando os serviços da Intel. No caso foi usamos a atestação EPID.

Para o armazenamento dos dados usamos várias estruturas para indexar prefixos, sufixos e

ainda o número em si na forma de um long. No caso dos prefixos, memoriza em que offset começa os números começados por aquele prefixo e onde termina. No caso do sufixo, ele memoriza uma lista, com números terminados naquele sufixo. No caso, o armazenamento dos números tem que ser realizado de forma ordenada e não permite adicionar ou remover. Sempre que for substituir algo é necessário enviar todo o set de uma só vez e de forma ordenada. Por outro lado, isto permite aplicar pesquisa binária nas operações. Como resultado, os tempos são praticamente na ordem dos $O(\log n)$ para cada pesquisa. Iremos falar disto em mais pormenor na secção de Otimizações.

Para as query's implementamos assinaturas usando uma chave RSA. Para tal tivemos que definir uma API para interagir com o enclave, seguem as mensagens que o enclave interage:

- Op Set Number – Esta operação é do formato "0 $num_1, \ldots, num_n$". Os números têm que ser enviados de forma ordenada. A operação devolve "0" se tudo correr bem ou "1" se tiver algum erro.
- Op Get Info – Esta operação é do formato "1 ops-info". Os vários tipos de ops-info serão descritos abaixo. O retorno depende do tipo de ops-info usado.
- Op Get Size – Esta operação é do formato "2". Ela devolve o tamanho do set de números definido dentro de enclave.
- Op Get Sequence Number – Esta operação é do formato "3". Ela devolve o número de sequência em uso dentro do enclave. É útil para a Mobileum para sincronizar com o número de série a ser usado junto das assinaturas.

Para estas operações temos que definir a posição inicial e final. A posição inicial é sempre maior ou igual a 0. Já a final será menor ou igual ao valor retornado pelo Op Get Size. Estes fatores são interessantes, pois permitem limitar o domínio sobre o qual são realizadas as operações. Outra característica é que temos sempre a opção de devolver apenas a contagem ou uma lista de identificadores únicos para cada posição. Para as ops-info implementamos um conjunto de operações:

- Op Set Intersection – Para obter o resultado do Set Intersection em formato de contagem usar o formato "1 0:posInit:posLast:$num_1, \ldots, num_n$". Para obter a lista de identificadores usar o formato é "1 5:posInit:posLast:$num_1, \ldots, num_n$".
- Op Prefix Match – Para obter o resultado do Prefix Match em formato de contagem usar o formato "1 1:posInit:posLast:$prefix_1, \ldots, prefix_n$". Para obter a lista de identificadores o formato é "1 6:posInit:posLast:$prefix_1, \ldots, prefix_n$".
- Op Suffix Match – Para obter o resultado do Suffix Match em formato de contagem usar o formato "1 2:posInit:posLast:$suffix_1, \ldots, suffix_n$". Para obter a lista de identificadores o formato é "1 7:posInit:posLast:$suffix_1, \ldots, suffix_n$".

- Op Hamming Distance – Para obter o resultado da distância de Hamming em formato de contagem usar o formato "1 3:posInit:posLast:distHamming:$num_1, \ldots, num_n$". Para obter a lista de identificadores o formato é "1 8:posInit:posLast:distHamming:$num_1, \ldots, num_n$". O distHamming é o valor que será usado para juntar na contagem ou adicionado a lista de identificados. Por outras palavras, se a distância de Hamming de um certo número e outro que esta dentro do enclave for menos ou igual ao distHamming então adiciona a lista, ou aumenta na contagem.

- Op Sequnce Match – A operação Sequence Match percorre todo o set a procura de números que tenham a distância entre eles de pelo menos distEuclidiana entre eles. Para obter apenas a contagem o formato é "1 4:posInit:posLast:distEuclidiana". Para obter a lista de identificadores o formato é "1 9:posInit:posLast:distEuclidiana".

Os resultados são todos do tipo "1 op contagem" ou "1 op listaDeIdentificadores".

Como referido, ainda existe a possibilidade de definir novas operações e fazer operações em cascata. Por exemplo, fazer Set Intersection da base de dados inteira. E depois com base no resultado e limitar a pesquisa a uma sequência específica do set fazer uma operação mais pesada como o Sequence Match.

5.3.1 Otimizações

A abordagem inicial considerava um array de char para representar os números armazenados. Isso implicava ter que usar vários ciclos para processar o strcmp(). Para terminar com esta limitação optemos por usar uma representação em long. O long tem espaço suficiente para representar um número na íntegra. A primeira vantagem é que reduzimos o gasto de memória de cada número de 14 char, o que equivale a $14 * sizeof(char) = 14$ bytes. Já um long usa apenas 8 bytes e permite manter o alinhamento de memória necessário a várias operações. A segunda vantagem é o número nesta representação pode ser armazenado num registo do cpu. A última vantagem é que apenas é necessário um ciclo para fazer a comparação entre números. Esta alteração refletiu-se numa redução de tempo de cpu considerável. A aceleração foi 5.6X.

Outra limitação encontrada é que, para todas as operações, estamos a fazer uma pesquisa sequencial. Ou seja, tínhamos que percorrer o set integralmente para poder fazer a computação. A primeira medida foi introduzir os valores posInit e posLast, mas foi possível ir mais longe. A segunda medida foi assumirmos que a Mobileum injeta os dados de forma ordenada dentro do enclave. Como os dados estão ordenados e em muitas das operações implementadas as comparações são de 1 para 1, podemos começar a usar pesquisa binária. Todas as operações menos o Sequence Match passaram de $O(n)$ para $O(\log n)$.

Após uma análise mais pormenorizada vimos ser possível a custo de uso de mais memória criar uma espécie de índices para os prefixos. A estrutura usada memoriza para cada prefixo a posição inicial no set do primeiro número com aquele prefixo e a posição do último número com

aquele prefixo. Como o número de prefixos é limitado e é entre 0-999, foi necessário alocar apenas um array com 1000 posições. Esta otimização permitiu reduzir os tempos no Set Intersection, Prefix Match e Hamming Distance. A operação Prefix Match passou a ser $O(1)$.

A operação Suffix Match era talvez a mais complicada de otimizar, pois usar pesquisa binária não era suficiente para melhorar esta operação. Devido a forma como os números estão armazenados, implementamos outro índice, mas este para sufixos. Este por sua vez é dinâmico, pois não temos como saber a priori quantos números iram estar em cada bucket. Então implementamos um array com 10000 posições, pois os sufixos variam entre 0-9999. Este array é um array de lista ligada. Esta operação passou a ser $O(1)$ mas é mais complexa que a implementação do Prefix Match, ao exigir uma estrutura de memória dinâmica.

A operação mais complexa de todas era o Hamming Distance. Mesmo já tendo beneficiado da otimização das primeiras três otimizações, ainda usava um tempo razoável para fazer o cálculo da Hamming Distance. Após alguma pesquisa sobre o tema descobrimos a existência de uma operação de assembly que fazia o cálculo da distância de Hamming numa única operação. Essa operação chama-se popcount. Devido a nossa estrutura representação tivemos que fazer manipulações algébricas e só depois do xor entre os números aplicar o popcnt. Em C para usarmos o popcnt temos que chamar a função `__builtin_popcount()`. O speedup desta otimização foi de 1.6X.

For fim ainda fomos mais longe após observar que no cálculo da Hamming Distance não é necessário estar a computar todos os dígitos. Em primeiro lugar, a diferença de Hamming entre cada dígito é no máximo de 4. Ou seja, podemos aplicar pesquisa binária no restante do número, pois este mantém a propriedade de igualdade. Se for até 8, dois dígitos e consequentemente os seus múltiplos até ao tamanho do número integral. Usando o exemplo da distância de 4, apenas o último dígito tem que ser comparado usando o cálculo da Hamming. O restante número não tem nenhuma diferença, ou seja, a distância de Hamming é igual a 0. Quando menor foi a distância de Hamming solicitada, menor será o campo de comparação, pois a pesquisa binária encurtará ainda mais o campo de pesquisa. E por fim, em vez de fazer o cálculo da Hamming a todos os dígitos apenas será feito os necessários. No caso, apenas será feito o cálculo no último dígito. Esta otimização permitiu amortizar ainda mais o tempo do cálculo da Hamming Distance.

O código otimizado do cálculo da distância de Hamming de dentro do enclave está listado em 5.1. O código é constituído em 3 partes. A primeira em que acedemos a estrutura dos prefixos para começarmos a pesquisa só onde existem prefixos daquele número. A seguir aplicamos pesquisa binária até onde não podermos distinguir mais dígitos conforme a profundidade. E por fim fazemos a pesquisa sequencial com o bloco de números restantes e calculando a distância de Hamming. As distâncias que foram menores ou iguais são adicionadas ao counter. As divisões e restos de divisão são usados para isolar cada dígito durante o cálculo da Hamming. Importa ressalvar que apenas serão computados os dígitos que podem ter distância diferente de 0.

5.4 Avaliação experimental

Tal como nos testes do protótipo de software, os protocolos foram todos avaliados num único portátil com um processador Intel i7 8750H e 16GB de RAM. No portátil está instalado o Pop!_OS 22.04 com o Kernel Linux 6.4. Todos os protocolos foram executados em single thread. Os participantes, cliente e servidor estão na mesma máquina, e como tal descartamos potenciais atrasos derivados de latência de rede.

Os dados usados para teste foram gerados de forma aleatória. Os números têm o formato +XXXXXXXXXXXX. Os casos de teste gerados com quantidades em bases de 2. No caso entre 2^{12} números até 2^{24} números. Conforme indicado pela Mobileum, o tempo esperado para tempo real é até 200ms. Os tempos obtidos a negrito são os tempos em cumprem a deadline para uso em tempo real. Mas mesmo os tempos restantes são interessantes noutros contextos como deteção de fraude pós-incidente. Existem situações em que o prazo pode ser até uma semana, por exemplo.

Para cada teste foram executadas três iterações. Após analisar os dados observamos que a variabilidade, no pior caso, era menor que 6,5%. Este valor é mais alto que o esperado e por isso podem variar significativamente em relação à média. Mas observando os valores obtidos, ter um desvio padrão de 6,5% num valor de até 10ms é irrisório. Devido a esta baixa variabilidade, concluímos que não era necessário executar mais do que as três iterações, uma vez que obteríamos resultados semelhantes em repetições adicionais. No entanto, analisaremos detalhadamente cada desvio padrão para cada operação em concreto.

Intel SGX (ms)		Quantidade de Números Armazenados no Enclave												
		2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
Operações	Insert Numbers	**8**	**11**	**13**	**15**	**19**	**33**	**70**	**138**	261	572	1509	3578	8605
	Set Intersection	**7**	**8**	**7**	**7**	**7**	**8**	**6**	**7**	**7**	**6**	**7**	**7**	**7**
	Prefix Match	**7**	**7**	**7**	**6**	**6**	**7**	**6**	**6**	**6**	**6**	**7**	**7**	**6**
	Suffix Match	**6**	**7**	**6**	**6**	**6**	**6**	**8**	**6**	**6**	**7**	**6**	**7**	**6**
	Hamming Distance	**6**	**6**	**6**	**6**	**7**	**6**	**7**	**7**	**6**	**6**	**7**	**7**	**6**
	Sequence Match	**6**	**8**	**7**	**7**	**7**	**7**	**6**	**9**	**12**	**15**	**106**	277	573

Tabela 5.1: Tabela de dados de testes recolhidos na execução de protótipo em Intel SGX usando o Gramine. Os valores a negrito são os elementos que podem ser executados em tempo real.

A nossa simulação não contempla a app da Mobileum para responder a pedidos e fazer as assinaturas para a CMU. No caso, colocamos assinaturas válidas a partir da CMU. Com isto conseguimos ter em vista o desempenho do enclave em vez do sistema todo.

Devido à estrutura usada, cada operação foi medida como tempo RA-TLS + tempo da operação. O tempo de estabelecer o canal RA-TLS é em média 670ms. A tabela 5.1 não contempla o tempo usado para estabelecer o canal RA-TLS, que pode ser amortizado estabelecendo o canal apenas uma vez para várias operações a realizar.

Outra limitação é que a estrutura de armazenamento não permite adicionar ou remover

números. Para tal temos de reinserir todo o dataset novamente de forma ordenada sempre que queremos fazer alterações. Cada Insert Numbers injeta no enclave aquele conjunto de números.

Os dados da tabela 5.1 foram gerados com o Set Intersection de um único número, Prefix Match de um único prefixo, Suffix Match de um único sufixo, Hamming Distance de um número e distância 1, Sequence Match de apenas 1 de distância.

Na operação "Insert Numbers" o desvio padrão foi de 0,8%, um valor baixo o suficiente para não executar mais testes. Nas operações "Set Intersection", "Prefix Match", "Suffix Match" e "Hamming Distance" o desvio padrão foi de 6,5%, mas olhando aos valores, o pior valor obtido foi de 9ms. Ou seja, não é relevante o desvio padrão obtido. Por fim, na operação "Sequence Match" o desvio padrão obtido foi de 1,3% um valor baixo o suficiente. Estes foram os motivos que nos levaram a não executar mais que três testes.

O Set Intersection e a Hamming Distance tiram partido da pesquisa binária, o que se reflete nos valores obtidos. O Prefix Match e a Suffix Match usam os índices internos, o que permite obter uma resposta quase constante. O Sequence Match tem um resultado bastante promissor, mas usando paralelização poderíamos chegar a 2^{24} mesmo em tempo real. Também é percetível um salto muito grande entre os tempos de 2^{21} e os de 2^{22}. Suspeitamos que seja devido às limitações de Enclave Page Cache (EPC) do processador Intel usado no teste.

Claramente existe forma de otimizar a inserção de números. Uma delas é em vez de usar a representação de strings para representar números, usar nas querys e as operações todo codificado no formato de um long. Retirando todo o *overhead* da conversão constante. Isto também teria impacto nas outras operações.

Ainda existe um conjunto alargado de otimizações que podem ser realizadas junto deste trabalho que estão descritas na secção 6.1.

5.5 Sumário

O protótipo resultante responde as cinco operações requisitadas pela Mobileum, devolve dois tipos possíveis de resposta, permite adicionar novas operações no futuro, assim como impedir operações de revelarem informação em excesso. As otimizações foram importantes, ao permitirem escalar as operações dentro do enclave. Após a avaliação experimental constatamos que foram muitas as situações em que podemos usar o protótipo para deteção de fraude em tempo real. Este protótipo já se aproxima da quantidade de dados processados pelo Mobileum diariamente.

```
size_t counter = 0;
int depth = 12 - ceil_divide(distance, 4);

for(size_t j = 0; j < numbers_size; j++) {
    long l,r;

    // PREFIX JUMP
    if(depth >= 3) {
        prefix_jump(numbers[j], posInit, posLast, &l, &r, 0);
    } else {
        l = posInit;
        r = posLast;
    }

    //BINARY SEARCH
    while (l <= r) {
        size_t m = l + (r - l) / 2;

        long ret = (storage[m] - numbers[j]) / pow10_s[depth];
        if (ret == 0) {        // Check if ret is present at mid
            break;
        } else if (ret < 0) { // If ret greater, ignore left half
            l = m + 1;
        } else {               // If ret is smaller, ignore right half
            r = m - 1;
        }
    }

    //SEQUENTIAL CHECK
    for (long i = l; i <= r; i++) {
        size_t hd = 0;

        long rest1 = numbers[j];
        long rest2 = storage[i];
        for(int k = depth; k < 12; k++) {
            int dig1 = rest1 % 10;
            int dig2 = rest2 % 10;
            hd += __builtin_popcount(dig1 ^ dig2);
            rest1 /= 10;
            rest2 /= 10;
        }

        if (hd <= distance)
            counter++;
    }
}

get_info_output_gen_counter(result, result_size, op_type, counter);
```

Bloco de Código 5.1: Código usado dentro do enclave para calcular a Hamming Distance.

Capítulo 6

Conclusão

Esta dissertação consistiu na construção de dois protótipos associadas ao problema da deteção de fraude em telecomunicações. Era pretendido realizar um conjunto de cinco operações sobre números de telemóveis e respeitando os requisitos de privacidade. E era aspectável que conseguíssemos realizar todas as operações no menor tempo possível. No protótipo de baseado em software está limitado às características dos algoritmos usados. Ou seja, apenas exploramos o PSI e as variações possíveis da estrutura SHD. Existem outras abordagens seguras que poderiam complementar este protótipo. Os objetivos propostos foram cumpridos no protótipo de hardware confiável, em que conseguimos implementar todas as operações, devolver dois tipos de resposta, contagem e lista de identificadores, conseguindo escalar com a quantidade de números armazenados.

Tendo em vista as propostas por parte da Mobileum e os requisitos de privacidade, conseguimos alcançar resultados satisfatórios. Era esperado que conseguíssemos ter respostas em até 200ms, para casos de resposta em tempo real. Isso aconteceu nos dois protótipos. O protótipo de software tem certas limitações na escalabilidade devido aos algoritmos usados. O protótipo de hardware não apresentou esta limitação, respondendo sempre em tempo real para sets de até 2^{22} em todas as operações.

Acredito que esta dissertação tem mais potencial para ser explorada e também para otimizada, conseguindo obter resultados ainda melhores. Na secção seguinte será descrito em maior pormenor algumas destas melhorias.

6.1 Trabalho Futuro

Nesta secção iremos descrever melhorias que podem ser aplicadas aos protótipos e um conjunto de abordagens futuras que poderão ser exploradas no sentido de alcançar a privacidade em deteção de fraude.

6.1.1 Software

Dentro do protótipo de software foram identificadas várias melhorias.

No curto prazo podemos alterar o OT-E usado no SHD para implementar a estrutura de 1-N descrita no [6]. Isto irá reduzir substancialmente o consumo de rede e o tempo de processamento. Podemos também paralelizar e vetorizar a implementação do SHD. Para as paralelizações podemos usar OpenMP. Já para vectorizações podemos explorar o Advanced Vector Extensions (AVX).

No longo prazo deviam ser exploradas abordagens alternativas para as heurísticas usadas junto do SHD. Quando combinadas com outras técnicas, por exemplo, comparação segura, podemos reduzir ainda mais o tempo usado em cada operação. Por fim, podíamos ir mais longe e em vez de usar uma implementação baseada em Secure Two-Party Computation (S2PC), como no PSI e o SHD, usar uma baseada em SMC. Seria um modelo mais geral, pois estaríamos a aceitar que poderia existir um adversário na computação, mas, por outro lado, poderíamos usar algoritmos bastante mais poderosos.

6.1.2 Hardware

O protótipo de hardware, devido ao seu desenho, tem imensas possibilidades de melhoria.

No curto prazo podemos, assim como no protótipo de software, paralelizar e vetorizar as operações do enclave, principalmente na operação Sequence Match. No caso do Gramine recomenda-se usar OpenMP. Para vectorizações podemos explorar o AVX. Uma das limitações encontradas, foi a incapacidade de usar o canal para várias operações. Então podemos reconstruir o canal RA-TLS para várias querys. Isto poder ser feito usando mensagens de Keepalive dentro da Socket ou usando TLS Session Tickets. Outra limitação encontrada, é a constante serialização para string e depois desserialização de novo para long. Para resolver podemos implementar o canal usando diretamente longs no buffer da Socket. Isto irá permitir reduzir consideravelmente o tempo de todas as operações, especialmente, o armazenamento dos números dentro do enclave. Depois temos pequenas alterações, como alterar a leitura dos números diretamente para a estrutura de armazenamento. Atualmente estamos a usar buffers intermédios em todas as operações. E para terminar, permitir a transferência de chaves de verificação das querys a partir da Mobileum para o enclave. Atualmente estamos a usar uma chave pública guardada dentro do enclave.

No longo prazo, é imperativo que aprofundemos a nossa pesquisa e desenvolvimento, levando em consideração a integração de hardware confiável heterogéneo, como as tecnologias ARM TrustZone e MIT Sanctum. Isso não só aumentará a segurança e a eficiência dos sistemas, mas também abrirá portas para uma ampla gama de aplicações inovadoras e aprimoradas. Deixando assim de depender apenas de Intel SGX.

Referências

[1] Jean-Philippe Aumasson. *Serious cryptography: a practical introduction to modern encryption*. No Starch Press, 2017.

[2] Raad Bahmani, Manuel Barbosa, Ferdinand Brasser, Bernardo Portela, Ahmad-Reza Sadeghi, Guillaume Scerri, and Bogdan Warinschi. Secure multiparty computation from sgx. In Aggelos Kiayias, editor, *Financial Cryptography and Data Security*, pages 477–497, Cham, 2017. Springer International Publishing. ISBN: 978-3-319-70972-7.

[3] Manuel Barbosa, Bernardo Portela, Guillaume Scerri, and Bogdan Warinschi. Foundations of hardware-based attested computation and application to sgx. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 245–260, 2016. doi:10.1109/EuroSP.2016.28.

[4] Richard A. Becker, Chris Volinsky, and Allan R. Wilks. Fraud detection in telecommunications: History and lessons learned. *Technometrics*, 52(1):20–33, 2010. doi:10.1198/TECH.2009.08136.

[5] Julien Bringer, Hervé Chabanne, and Alain Patey. Shade: Secure hamming distance computation from oblivious transfer. In Andrew A. Adams, Michael Brenner, and Matthew Smith, editors, *Financial Cryptography and Data Security*, pages 164–176, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN: 978-3-642-41320-9.

[6] Julien Bringer, Herve Chabanne, Melanie Favre, Alain Patey, Thomas Schneider, and Michael Zohner. Gshade: Faster privacy-preserving distance computation and biometric identification. In *Proceedings of the 2nd ACM Workshop on Information Hiding and Multimedia Security*, IH&MMSec '14, page 187–198, New York, NY, USA, 2014. Association for Computing Machinery. ISBN: 9781450326476. doi:10.1145/2600918.2600922.

[7] Luísa Lopes Caldas. Deteção de fraude em telecomunicações através de machine learning. Master's thesis, Universidade do Minho, 2019.

[8] Mirela Cazzolato, Saranya Vijayakumar, Meng-Chieh Lee, Catalina Vajiac, Xinyi Zheng, Namyong Park, Pedro Fidalgo, Agma J. M. Traina, and Christos Faloutsos. Callmine: Fraud detection and visualization of million-scale call graphs. In *CIKM 2023: Conference on Information and Knowledge Management*, University of Birmingham and Eastside Rooms, UK, 2023.

[9] Mirela T. Cazzolato, Saranya Vijayakumar, Xinyi Zheng, Namyong Park, Meng-Chieh Lee, Pedro Fidalgo, Bruno Lages, Agma J. M. Traina, and Christos Faloutsos. Tgraphspot: Fast and effective anomaly detection for time-evolving graphs. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 2214–2217, Dec 2022. doi:10.1109/BigData55660.2022.10020898.

[10] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious prf. Cryptology ePrint Archive, Paper 2020/729, 2020. https://eprint.iacr.org/2020/729.

[11] Chia che Tsai, Donald E. Porter, and Mona Vij. Graphene-SGX: A practical library OS for unmodified applications on SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 645–658, Santa Clara, CA, jul 2017. USENIX Association. ISBN: 978-1-931971-38-6.

[12] Kelong Cong. Co6gc: Introduction to oblivious transfer, 2020.

[13] Gramine Contributors. Attestation and secret provisioning - gramine documentation, 2023.

[14] Victor Costan and Srinivas Devadas. Intel sgx explained. Cryptology ePrint Archive, Paper 2016/086, 2016. https://eprint.iacr.org/2016/086.

[15] Claude Crépeau. Verifiable disclosure of secrets and applications (abstract). In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology — EUROCRYPT '89*, pages 150–154, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg. ISBN: 978-3-540-46885-1.

[16] Diário da República. Resolução do Conselho de Ministros n.º 41/2018, 2018.

[17] Parlamento Europeu e o Conselho da União Europeia. Regulamento (UE) N.º 2016/679, de 27 de abril de 2016, 2016.

[18] Parlamento Europeu e o Conselho da União Europeia. Retificação do Regulamento (UE) N.º 2016/679, de 27 de abril de 2016, 2018.

[19] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, jun 1985. ISSN: 0001-0782. doi:10.1145/3812.3818.

[20] Pedro Fidalgo, Rui J. Lopes, and Christos Faloutsos. Star-bridge: a topological multidimensional subgraph analysis to detect fraudulent nodes and rings in telecom networks. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 2239–2242, Dec 2022. doi:10.1109/BigData55660.2022.10020714.

[21] Nic Fildes. Fraud costs telecoms industry $17bn a year, Oct 2018.

[22] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, page 365–377, New York, NY, USA, 1982. Association for Computing Machinery. ISBN: 0897910702. doi:10.1145/800070.802212.

[23] Le Guan, Peng Liu, Xinyu Xing, Xinyang Ge, Shengzhi Zhang, Meng Yu, and Trent Jaeger. Trustshadow: Secure execution of unmodified applications with arm trustzone. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '17, page 488–501, New York, NY, USA, 2017. Association for Computing Machinery. ISBN: 9781450349284. doi:10.1145/3081333.3081349.

[24] *X Congresso Internacional de Ciências Jurídico-Empresariais: actas*, 12 2019. Instituto Politécnico de Leiria – Escola Superior de Tecnologia e Gestão.

[25] *XI Congresso Internacional de Ciências Jurídico-Empresariais: atas*, 02 2022. Instituto Politécnico de Leiria – Escola Superior de Tecnologia e Gestão.

[26] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, pages 145–161, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN: 978-3-540-45146-4.

[27] Mehmet S. Kiraz, Berry Schoenmakers, and José Villegas. Efficient committed oblivious transfer of bit strings. In Juan A. Garay, Arjen K. Lenstra, Masahiro Mambo, and René Peralta, editors, *Information Security*, pages 130–144, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN: 978-3-540-75496-1.

[28] Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. Integrating remote attestation with transport layer security. *arXiv preprint arXiv:1801.05863*, 2018.

[29] Wenhao Li, Yubin Xia, and Haibo Chen. Research on arm trustzone. *GetMobile: Mobile Comp. and Comm.*, 22(3):17–22, jan 2019. ISSN: 2375-0529. doi:10.1145/3308755.3308761.

[30] Ziyuan Liang, Weiran Liu, Fan Zhang, Bingsheng Zhang, Jian Liu, Lei Zhang, and Kui Ren. A framework of private set intersection protocols. Cryptology ePrint Archive, Paper 2020/1541, 2020. https://eprint.iacr.org/2020/1541.

[31] Moni Naor. Bit commitment using pseudo-randomness *. *Journal of Cryptology*, 4(2): 151–158, 1991. doi:10.1007/bf00196774.

[32] Corey Neskey. Are your passwords in the green?, May 2023.

[33] Rui Leal de Oliveira. O impacto da proteção de dados (rgpd) no retalho omnicanal. Master's thesis, Instituto Politécnico do Porto, 2021.

[34] Osu-Crypto. Osu-crypto/libote: A fast, portable, and easy to use oblivious transfer library, 2023.

[35] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, page 515–530, USA, 2015. USENIX Association. ISBN: 9781931971232.

[36] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. *ACM Trans. Priv. Secur.*, 21(2), jan 2018. ISSN: 2471-2566. doi:10.1145/3154794.

[37] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Psi from paxos: Fast, malicious private set intersection. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 739–767, Cham, 2020. Springer International Publishing. ISBN: 978-3-030-45724-2.

[38] Srinivasan Raghuraman and Peter Rindal. Blazing fast psi from improved okvs and subfield vole. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 2505–2517, New York, NY, USA, 2022. Association for Computing Machinery. ISBN: 9781450394505. doi:10.1145/3548606.3560658.

[39] Peter Rindal and Phillipp Schoppmann. Vole-psi: Fast oprf and circuit-psi from vector-ole. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 901–930, Cham, 2021. Springer International Publishing. ISBN: 978-3-030-77886-6.

[40] Ronald Rivest. Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer. *Unpublished manuscript*, 1999.

[41] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: What it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64, 2015. doi:10.1109/Trustcom.2015.357.

[42] Chia-Che Tsai, Kumar Saurabh Arora, Nehal Bandi, Bhushan Jain, William Jannen, Jitin John, Harry A. Kalodner, Vrushali Kulkarni, Daniela Oliveira, and Donald E. Porter. Cooperation and security isolation of library oses for multi-process applications. In *Proceedings of the Ninth European Conference on Computer Systems*, EuroSys '14, New York, NY, USA, 2014. Association for Computing Machinery. ISBN: 9781450327046. doi:10.1145/2592798.2592812.

[43] Visa-Research. Visa-research/volepsi: Efficient private set intersection base on vole, 2023.